UNIVERSIDADE FEDERAL DE UBERLÂNDIA
INSTITUTO DE CIÊNCIAS AGRÁRIAS
PROGRAMA DE PÓS-GRADUAÇÃO EM QUALIDADE AMBIENTAL

FLAVYA FERNANDA FRANÇA VILELA

QUALITOOL 2.0: APLICATIVO WEB PARA MODELAGEM DA QUALIDADE DE ÁGUA EM AMBIENTE LÓTICO

UBERLÂNDIA – MG
2025

FLAVYA FERNANDA FRANÇA VILELA

QUALITOOL 2.0: APLICATIVO WEB PARA MODELAGEM DA QUALIDADE DE ÁGUA EM AMBIENTE LÓTICO.

Dissertação apresentada à Universidade Federal de Uberlândia, como parte das exigências do Programa de Pós-graduação em Meio Ambiente e Qualidade Ambiental – Mestrado, área de concentração em Meio Ambiente e Qualidade Ambiental, para a obtenção do título de “Mestre’

Área de concentração: Meio Ambiente e Qualidade Ambiental.

Orientador: Prof. Dr. Marcio Ricardo Salla

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

V699 2025

Vilela, Flavya Fernanda França, 1998-
QUALITOOL 2.0: Aplicativo web para modelagem da qualidade de água em ambiente lótico [recurso eletrônico] / Flavya Fernanda França Vilela. - 2025.

Orientador: Marcio Ricardo Salla.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Qualidade Ambiental.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2025.207
Inclui bibliografia.

1. Desenvolvimento sustentável. I. Salla, Marcio Ricardo,1976-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Qualidade Ambiental. III. Título.

CDU: 502.33

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Qualidade Ambiental
BR 050, Km 78, Bloco 1CCG, Sala 206 - Bairro Glória, Uberlândia-MG, CEP 38400-902
Telefone: (34) 2512-6717 - www.ppgmq.iciag.ufu.br - ppgmq@iciag.ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Qualidade Ambiental (PPGMQ)				
Defesa de:	Dissertação de Mestrado Acadêmico, 03/2025, PPGMQ				
Data:	12 de março de 2025	Hora de início:	13:30	Hora de encerramento:	15:30
Matrícula da Discente:	12312MQA003				
Nome da Discente:	Flavya Fernanda França Vilela				
Título do Trabalho:	QUALITOOL 2.0: Aplicativo WEB para modelagem da qualidade de água em ambiente lótico				
Área de concentração:	Meio Ambiente e Qualidade Ambiental				
Linha de pesquisa:	MONITORAMENTO E GESTÃO AMBIENTAL				
Projeto de Pesquisa de vinculação:	QUALITOOL 2.0: Aplicativo WEB para modelagem da qualidade de água em ambiente lótico				

Reuniu-se por meio de web conferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Qualidade Ambiental (PPGMQ), assim composta: Prof. Dr. Marcio Ricardo Salla (Orientador); Prof. Dr. Ednaldo Carvalho Guimarães (UFU); e Prof. Dr. Fernando Mainardi Fan (UFRGS).

Iniciando os trabalhos o presidente da mesa Prof. Dr. Marcio Ricardo Salla apresentou a Comissão Examinadora e a candidata, agradeceu a presença do público, e concedeu à Discente a palavra para a exposição do seu trabalho. A duração da apresentação da Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir a candidata. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando a candidata:

Aprovada.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento da nova defesa, aprovação da candidata e dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Marcio Ricardo Salla**, **Professor(a) do Magistério Superior**, em 12/03/2025, às 15:29, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Ednaldo Carvalho Guimarães**, **Professor(a) do Magistério Superior**, em 12/03/2025, às 15:37, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fernando Mainardi Fan**, **Usuário Externo**, em 13/03/2025, às 10:57, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **6058304** e o código CRC **FB6460C6**.

Dedico este trabalho à minha mãe, minha maior incentivadora, pela educação que me deu e pelo amor que me dedicou.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus pelo dom da vida, por me permitir evoluir e buscar livremente aquilo em que acredito.

Aos meus pais, Fernanda e Júnior, sou imensamente grata pelo apoio incondicional e pelo carinho ao longo desta jornada. Esta conquista é de vocês também. Ao meu parceiro, Caetano, agradeço pelo companheirismo, pela cumplicidade e pelo suporte em todos os momentos desafiadores deste percurso.

Ao professor Marcio Ricardo Salla, meu orientador, expresso minha profunda gratidão por sua competência, dedicação e confiança em mim, tornando minha jornada mais leve e enriquecedora.

A todos os professores e professoras com quem tive a honra de me instruir ao longo da minha trajetória acadêmica e que contribuíram para meu crescimento pessoal e científico, dedico meu sincero reconhecimento, em especial às professoras Anne Caroline Malvestio e Samara Carbone e ao professor Rafael Genaro. Aos técnicos Glaicon Florisbelo Alves e Márcia Regina Batistela Moraes, minha eterna admiração e carinho.

Às minhas irmãs, Julya e Fanny, e aos meus amigos, em especial André, Guilherme e Marina, sou profundamente grata pela amizade verdadeira e pelo apoio constante ao longo de toda a minha trajetória neste trabalho.

Agradeço à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) pelo fomento, pelo apoio financeiro de reserva técnica (processo APQ01699-17 Demanda Universal), que viabilizou o processo de validação da ferramenta computacional, e pela consolidação do programa de pós-graduação stricto sensu em Qualidade Ambiental, bem como pelo incentivo a tantos outros programas no Brasil.

Também expresso minha gratidão à Universidade Federal de Uberlândia (UFU), à Pró-Reitoria de Pesquisa e Pós-Graduação (PROPP-UFU), ao Instituto de Ciências Agrárias (ICIAG) e ao Programa de Pós-Graduação em Qualidade Ambiental (PPGMQ) pelo incentivo à pesquisa e ao desenvolvimento sustentável e ambiental.

“Procurem deixar este mundo um pouco melhor do que o encontraram.”

(BADEN-POWELL, 1975, p. 368)

RESUMO

Este trabalho apresenta o desenvolvimento do QUALITOOL 2.0, um aplicativo web para simulações da qualidade de água em ambientes lóticos, contribuindo para o gerenciamento de bacias hidrográficas. O aplicativo modela parâmetros como oxigênio dissolvido, demanda bioquímica de oxigênio, nitrogênio e fósforo (e suas frações), além de coliformes, com base em equações matemáticas consagradas na literatura que consideram os termos advectivos de transporte da massa poluente e os termos cinéticos de decaimento dos parâmetros de qualidade da água. O aplicativo aceita entradas de dados topológicos georreferenciados ou não, incluindo as contribuições pontuais ou difusas e retiradas pontuais, além de permitir a entrada de séries temporais para melhor gestão dos resultados. Na modelagem hidrodinâmica, para o formato retangular ou trapezoidal da seção transversal do curso de água, a equação de Manning e a equação da continuidade permitem estimar a profundidade líquida média e a velocidade média do escoamento para cada trecho discretizado de curso de água. O usuário tem a liberdade na escolha da amplitude de discretização longitudinal do curso de água, cuja atividade é utilizada na resolução numérica, por diferenças finitas, das equações diferenciais que representam o transporte e reações bioquímicas da massa poluente dentro do curso de água. O aplicativo possui análise integrada de sensibilidade e calibração dos coeficientes cinéticos por meio de algoritmo de otimização por enxame de partículas (*Particle Swarm Optimization*), um algoritmo evolucionário que aprimora suas estimativas ao longo das iterações. Os resultados das modelagens podem ser obtidos por meio de gráficos dinâmicos, planilhas e visualizações geoespaciais caso haja o georreferenciamento dos dados. A comparação com outras duas ferramentas computacionais, tais como a versão inicial do QUALITOOL e a ferramenta QUAL-UFMG, permitiu validar o aplicativo QUALITOOL 2.0, cuja principal vantagem está relacionada à velocidade de processamento computacional, justificado pela implementação de metodologias de programação paralela e orientada a objetos. Espera-se que o QUALITOOL 2.0 seja um sistema de suporte à decisão na avaliação de impactos ambientais em ecossistemas aquáticos lóticos.

Palavras-chave: modelo de qualidade da água; calibração; otimização por enxame de partículas; aplicativo QUALITOOL 2.0.

ABSTRACT

This paper presents the development of QUALITOOL 2.0, a web application for simulating water quality in lotic environments, contributing to river basin management. The application models key parameters such as dissolved oxygen, biochemical oxygen demand, nitrogen and phosphorus (including their fractions), as well as coliforms, based on well-established mathematical equations in the literature. These equations account for the advective transport of pollutants and the kinetic decay of water quality parameters. The application supports both georeferenced and non-georeferenced topological data inputs, including point and diffuse pollution sources, as well as point withdrawals. Additionally, it allows for the incorporation of time series data, enhancing result management. In the hydrodynamic modeling process, for rectangular or trapezoidal cross-sections of the watercourse, the Manning equation and the continuity equation are employed to estimate the average water depth and flow velocity for each discretized segment of the watercourse. Users have flexibility in defining the longitudinal discretization scale of the watercourse, which is utilized in the numerical solution of the differential equations governing pollutant transport and biochemical reactions, solved using the finite difference method. The application also features integrated sensitivity analysis and kinetic coefficient calibration through the Particle Swarm Optimization algorithm, an evolutionary approach that refines its estimates iteratively. Modeling results can be visualized through dynamic graphs, spreadsheets, and geospatial representations when georeferenced data is available. A comparative analysis with two other computational tools – the initial version of QUALITOOL and the QUAL-UFMG tool – validated the QUALITOOL 2.0 application. Its main advantage lies in its high computational processing speed, achieved through the implementation of parallel and object-oriented programming methodologies. It is expected that QUALITOOL 2.0 will serve as a decision support system for assessing environmental impacts in lotic aquatic ecosystems.

Keywords: water quality model; calibration; particle swarm optimization; QUALITOOL 2.0 application.

LISTA DE ILUSTRAÇÕES

Figura 1. Esquema ilustrativo do procedimento de adoção da aproximação numérica em métodos numéricos e da qualidade da aproximação em função do nível de discretização. 34
Figura 2. Exemplo da discretização dos dados georreferenciados. 35
Figura 3. Ilustração da seção transversal aplicada no QUALITOOL 2.0. 36
Figura 4. Esquema ilustrativo dos processos físicos e bioquímicos do modelo de qualidade da água do aplicativo QUALITOOL 2.0. 41
Figura 5. Perfis de OD (C) e DBO (L) em condições de aerobiose e anaerobiose. 44
Figura 6. Fluxograma do algoritmo PSO..... 51
Figura 7. Esquema de iteração das partículas no PSO. 52
Figura 8. Diagrama de classes do QUALITOOL 2.0. 54
Figura 9. Fluxograma da etapa modelagem no QUALITOOL 2.0. 57
Figura 10. Fluxograma da etapa ajuste e calibração no QUALITOOL 2.0. 59
Figura 11. Esquema ilustrativo de um sistema hídrico fictício. 60
Figura 12. Logomarca do QUALITOOL 2.0. 62
Figura 13. Menu principal do aplicativo QUALITOOL 2.0. 63
Figura 14. Aba "Modelagem" no aplicativo QUALITOOL 2.0. 66
Figura 15. Opção de preenchimento automático ativa. 67
Figura 16. Tela modelagem inicial. 68
Figura 17. Ponto de deságue. 69
Figura 18. Efeito da discretização na concentração de OD. 70
Figura 19. Opção de entrada de dados espaciais, "Manual" ativo. 70
Figura 20. Opção de entrada de dados espaciais, "Intervalo" ativo. 71
Figura 21. Opção de entrada de dados espaciais, "GeoJSON" ativo. 71
Figura 22. Opção de visualização de mapa para dados georreferenciados. 72
Figura 23. Entrada de dados temporais. 72
Figura 24. Seções transversais. 73
Figura 25. Dados de pontos de captação de água. 75
Figura 26. Dados de pontos de contribuição pontual. 76

Figura 27. Dados de pontos de contribuição difusa. 76
Figura 28. Dados de pontos que alteram os coeficientes. 77
Figura 29. Resultados hidrodinâmicos para o Rio principal. 80
Figura 30. Resultados de OD para o Rio principal. 81
Figura 31. Resultados de OD e DBO para o Rio principal. 81
Figura 32. Resultados do Nitrogênio e suas frações para o Rio principal. 82
Figura 33. Resultados no formato de planilha. 82
Figura 34. Resultados no formato de planilha, opção para download. 83
Figura 35. Visualização geoespacial dos resultados. 83
Figura 36. Aba “Ajuste e calibração”. 84
Figura 37. Configurações avançadas. 85
Figura 38. Intervalo de busca dos coeficientes do Rio principal. 86
Figura 39. Intervalo de busca dos coeficientes dos Tributários. 87
Figura 40. Dados observados. 88
Figura 41. Análise de sensibilidade. 89
Figura 42. Opção de fixar um ou mais modelos antes da calibração. 90
Figura 43. Exemplo de conjuntos. 90
Figura 44. Tela de espera. 91
Figura 45. Evolução da busca com PSO. 91
Figura 46. Visualização dos resultados da calibração no trecho. 92
Figura 47. Opção de fixar todos os coeficientes no trecho. 93
Figura 48. Resultados da calibração. 94
Figura 49. Aba “Gráficos”. 94
Figura 50. Ilustração do sistema hídrico do Rio Jacinto (fictício). 97
Figura 51. Resultados hidrodinâmicos no QUALITOOL e QUALITOOL 2.0. 100
Figura 52. Resultados das concentrações no QUALITOOL e QUALITOOL 2.0. 101
Figura 53. Ilustração do sistema hídrico do exemplo 2. 102
Figura 54. Resultados das concentrações no QUAL-UFMG e QUALITOOL 2.0. 105
Figura 55. Resultados da profundidade líquida em QUAL-UFMG e QUALITOOL 2.0. 107
Figura 56. Bacia hidrográfica do rio Uberabinha e pontos de monitoramento. 108

LISTA DE TABELAS

Tabela 1. Valores dos coeficientes predefinidos. 46
Tabela 2. Equações referentes ao coeficiente k2 (20ºC)..... 47
Tabela 3. Valores dos coeficientes *m* e *n* para diversos cursos d'água da Região Metropolitana de Belo Horizonte (período seco). 47
Tabela 4. Coeficientes de rugosidade de Manning. 74
Tabela 5. Valores típicos dos coeficientes de remoção de DBO (base e 20ºC). 78
Tabela 6. Valores do coeficiente de demanda de oxigênio pelo sedimento Sd' para diferentes tipos de fundo de rio (20ºC). 78
Tabela 7. Valores de Referência da demanda de oxigênio pelo sedimento Sd' para diferentes condições de fluxo do rio (20ºC). 78
Tabela 8. Valores de referência para a modelagem do nitrogênio (20°C). 79
Tabela 9. Valores de referência da modelagem do Fósforo (20ºC). 79
Tabela 10. Topologia do sistema hídrico do exemplo 1. 97
Tabela 11. Vazões pontuais e difusas do exemplo 1. 99
Tabela 12. Concentrações pontuais e difusas do exemplo 1..... 99
Tabela 13. Dados geométricos e hidráulicos da seção transversal do exemplo 1. 99
Tabela 14. Coeficientes dos processos físicos e bioquímicos do exemplo 1..... 100
Tabela 15. Tempo de execução do QUALITOOL x QUALITOOL 2.0, em segundos. 100
Tabela 16. Comprimento total de cada trecho do exemplo 2. 103
Tabela 17. Vazões pontuais e difusas do exemplo 2. 103
Tabela 18. Concentrações pontuais e difusas do exemplo 2..... 103
Tabela 19. Dados geométricos e hidráulicos da seção transversal do exemplo 2. 104
Tabela 20. Coeficientes dos processos físicos e bioquímicos do exemplo 2..... 104
Tabela 21. Métricas obtidas no exemplo 2. 106
Tabela 22. Comprimento total de cada trecho do exemplo 3. 109
Tabela 23. Dados espaciais do exemplo 3. 109
Tabela 24. Concentrações de entrada (exemplo 3). 110
Tabela 25. Intervalos dos coeficientes do exemplo 3. 112
Tabela 26. Análise de sensibilidade dos coeficientes. 112

Tabela 27. Resultados da calibração do exemplo 3. 113
Tabela 28. Valores dos coeficientes cinéticos do exemplo 3. 115

SUMÁRIO

1. INTRODUÇÃO 22

1.1. Justificativa 23

1.2. Objetivos 24

1.2.1. Objetivos específicos 24

2. REVISÃO BIBLIOGRÁFICA 26

2.1. Modelos matemáticos de qualidade da água 26

2.2. Softwares para MMQA 27

2.3. Calibração de parâmetros de qualidade da água 29

2.4. Inteligência Artificial e MMQA 30

2.5. Interação de MMQA com SIG 31

3. MATERIAIS E MÉTODOS 33

3.1. Características geométricas 33

3.1.1. Discretização 33

3.1.2. Seção transversal 35

3.2. Características hidráulicas 36

3.2.1. Declividade 36

3.2.2. Profundidade líquida e velocidade média do escoamento 37

3.2.3. Tensão de arraste e número de Froude 38

3.3. Equação de advecção-difusão 39

3.3.1. Nitrogênio e suas frações 41

3.3.2. Oxigênio Dissolvido 42

3.3.3. Demanda Bioquímica de Oxigênio 43

3.3.4. Fósforo e suas frações 44

3.3.5. Coliformes termotolerantes 45

3.3.6. Coeficientes cinéticos da literatura 45

3.4. Equação de mistura 48

3.5. Análise de erros de primeira ordem ou análise de sensibilidade 48

3.6. Calibração com algoritmos evolucionários 49

3.6.1. PSO no QUALITOOL 2.0 50

3.6.2. Aptidão das partículas 52

3.7. Desenvolvimento do QUALITOOL 2.0 53

3.7.1. Programação orientada à objeto 53

3.7.2. Recursos e requerimentos 55

3.7.3. Interação gráfica com Streamlit 56

3.8. Modelagem de qualidade da água com os coeficientes cinéticos predefinidos 57

3.9. Modelagem de qualidade da água com os coeficientes cinéticos otimizados . 58

4. RESULTADOS DA INTERFACE DO QUALITOOL 2.0 62

4.1. Apresentação do aplicativo QUALITOOL 2.0 / ABA "Apresentação" 62

4.1.1. Novidades da versão 2.0 64

4.2. ABA "INSTRUÇÕES" 65

4.3. ABA "MODELAGEM" 65

4.3.1. Entrada de dados 67

4.3.2. Resultados 79

4.4. ABA "AJUSTE E CALIBRAÇÃO" 84

4.4.1. Entrada de dados 85

4.4.2. Análise de sensibilidade 88

4.4.3. Calibração por trecho 90

4.4.4. Resultados 93

4.5. ABA "GRÁFICOS" 94

5. COMPARAÇÃO DO QUALITOOL 2.0 COM OUTRAS FERRAMENTAS DE USO LIVRE 96

5.1. Modelagem QUALITOOL versus QUALITOOL 2.0 96

5.2. Modelagem QUAL-UFMG versus QUALITOOL 2.0 102

5.3. Calibração com séries temporais 107

6. CONCLUSÃO 116

6.1. Conclusões gerais 116

6.2. Trabalhos futuros 117

REFERÊNCIAS 116

APÊNDICE A.....116

1. INTRODUÇÃO

A qualidade da água é fundamental para a saúde pública, o desenvolvimento econômico e a sustentabilidade ambiental. A crescente urbanização e o aumento das atividades industriais têm agravado a poluição dos corpos de água, resultando em sérios desafios para a gestão da qualidade da água. Estima-se que cerca de dois bilhões de pessoas em todo o mundo não tenham acesso à água potável segura; esta situação deve se agravar devido à possibilidade de um aumento da população urbana em situação de escassez hídrica: dos 930 milhões em 2016 para algo entre 1,7 e 2,4 bilhões até 2050, segundo a UNESCO (2023).

Ambientes lóticos, como rios e córregos, enfrentam particular dificuldade devido à combinação de poluição pontual e difusa, incluindo o lançamento inadequado de efluentes domésticos e industriais e o escoamento de produtos químicos agrícolas. A capacidade de avaliar e melhorar a qualidade da água desses sistemas é essencial para mitigar os impactos negativos e promover a gestão sustentável dos recursos hídricos (Von Sperling, 2014a; Zelazny *et al*., 2023).

A modelagem matemática é uma ferramenta essencial para entender e prever a dinâmica dos poluentes em corpos d'água (Mekonnen; Tenagashawu, 2023). Esses modelos permitem simular e avaliar a interação entre diferentes parâmetros de qualidade da água, facilitando a identificação de problemas e a implementação de estratégias eficazes de mitigação. Nos últimos anos, houve avanços significativos no desenvolvimento de ferramentas para a aplicação desses modelos, ampliando suas possibilidades de uso na gestão dos recursos hídricos (Al-Dalimy; Al-Zubaidi, 2023).

No entanto, a aplicabilidade desses modelos ainda enfrenta desafios, especialmente devido à ausência de módulos de otimização integrados (Salla *et al*., 2016; Sánchez-Gutiérrez; Gómez-Castro, 2021; Brum *et al*., 2022). A otimização é fundamental para aprimorar a calibração dos modelos, reduzir incertezas e definir soluções mais eficientes para o gerenciamento da qualidade da água (Ejigu, 2021; Rinjani *et al*., 2023). Sem esse recurso, a análise dos resultados pode ser limitada a uma análise probabilística (Brum *et al*., 2022), comprometendo a eficácia das medidas de controle e de mitigação da poluição.

Nesse contexto, o uso de ferramentas computacionais de código aberto tem se destacado como uma alternativa acessível e eficaz para modelagem e otimização ambiental. Ferramentas de uso livre, como o QUAL-UFMG (Von Sperling, 2014b), QUAL2K (Chapra *et al*., 2022) e o SWAT (Neitsch *et al*., 2023), permitem que pesquisadores e gestores ambientais analisem diversos cenários sem as restrições financeiras impostas por plataformas comerciais. A incorporação de algoritmos de otimização nesses sistemas pode ampliar significativamente sua aplicabilidade, viabilizando a definição de estratégias mais robustas para o controle da poluição e a gestão sustentável dos recursos hídricos (Lashani *et al*., 2024).

1.1. Justificativa

Apesar da disponibilidade de várias ferramentas computacionais para a simulação da qualidade da água em ambiente lótico, a complexidade dos modelos matemáticos e a necessidade de calibração de termos cinéticos apresentam desafios significativos (Salla *et al.*, 2016; Wool *et al*., 2020). Muitos modelos existentes enfrentam dificuldades com a precisão nos resultados, em função do número elevado de parâmetros físicos, químicos e biológicos e da natureza não linear das equações utilizadas (Formiga *et al*., 2016; Kim *et al*., 2023).

A eficácia de um modelo de qualidade da água depende da sua capacidade de calibração de termos cinéticos, na qual os dados monitorados de qualidade da água são ajustados aos dados simulados. Métodos convencionais de calibração, como análises de regressão e erros relativos, são frequentemente insuficientes para lidar com a complexidade dos dados e a dinâmica dos ecossistemas aquáticos lóticos (Griensven; Meixner, 2006; Han; Zheng, 2016). A inteligência artificial, incluindo redes neurais e algoritmos evolucionários, tem mostrado potencial para superar essas limitações (Xue *et al*., 2020; Khodabandeh *et al*., 2021), oferecendo novas abordagens para a otimização e a previsão dos parâmetros do modelo.

O desenvolvimento do aplicativo computacional QUALITOOL 2.0 é justificado pela necessidade de uma ferramenta mais avançada e acessível que possa melhorar a precisão dos modelos de qualidade da água e facilitar a gestão dos recursos hídricos. Esta versão visa a incorporar inovações tecnológicas para aprimorar a eficiência, a usabilidade

e a precisão dos modelos, principalmente pela inclusão de um módulo de calibração de termos cinéticos.

1.2. Objetivos

O principal objetivo desta pesquisa é desenvolver um aplicativo de simulação computacional de qualidade da água em ambiente lótico integrado a um sistema de calibração de termos cinéticos baseado em algoritmos de inteligência computacional. A versão 2.0 nesse aplicativo é uma continuidade de uma ferramenta computacional já existente, denominada QUALITOOL (Magalhães, 2017). A versão atual do aplicativo permite a modelagem dos parâmetros Demanda Bioquímica de Oxigênio (DBO), Oxigênio Dissolvido (OD), nitrogênio total e suas frações (orgânico, amoniacal, nitrito e nitrato), fósforo total e suas frações (orgânico e inorgânico) e coliformes termotolerantes (fecais).

1.2.1. Objetivos específicos

A fim de otimizar a performance do aplicativo QUALITOOL, a construção da versão 2.0 objetiva especificamente:

i. Implementar a eficiência computacional por meio de algoritmos mais rápidos e eficientes para reduzir o tempo de simulação e aumentar a capacidade de processamento de grandes volumes de dados;
ii. Desenvolver uma interface de usuário mais intuitiva e amigável, facilitando o uso do aplicativo por profissionais de diversas áreas, incluindo aqueles com menor experiência em modelagem matemática;
iii. Melhorar a capacidade de integrar dados espaciais, geoespaciais e séries de dados temporais, para fornecer análises mais abrangentes e dinâmicas;
iv. Implementar as etapas de calibração e análise de sensibilidade automatizadas, utilizando algoritmos de inteligência computacional para garantir que os modelos sejam continuamente ajustados e precisos;
v. Aumentar a capacidade do QUALITOOL para suportar simulações em diferentes escalas, desde pequenos trechos de rios até bacias hidrográficas inteiras e complexas.

Essas melhorias visam tornar o QUALITOOL 2.0 um aplicativo mais poderoso e versátil para a modelagem da qualidade da água em ambiente lótico, beneficiando pesquisadores, gestores ambientais e formuladores de políticas públicas.

2. REVISÃO BIBLIOGRÁFICA

2.1. Modelos matemáticos de qualidade da água

Os Modelos Matemáticos de Qualidade da Água (MMQA) aplicados em rios e reservatórios são ferramentas indispensáveis para a gestão dos recursos hídricos e o planejamento de ações para mitigar os impactos da poluição em corpos hídricos naturais e artificiais. O uso desses modelos facilita a tomada de decisões e permite uma melhor compreensão do meio ambiente, fornecendo uma visão integrada que associa informações físicas, químicas e biológicas de uma bacia (Ahmad, 2013; Salla *et al.*, 2016; Alamy Filho *et al.*, 2019).

Se corretamente operados e formulados, os modelos conseguem reproduzir de forma simplificada o comportamento real de um sistema, descrevendo os processos naturais e antrópicos através de formulações matemáticas bem definidas. Isso resulta em representações quantitativas e qualitativas consistentes que se aproximam da realidade (Zelazny *et al*., 2023). Os modelos contêm um nível de abstração do sistema real, variando de simples a complexos.

Os MMQA podem ser descritos por diferentes critérios e classificados conforme o comportamento do sistema estudado, como condições de escoamento, transporte de massa e características das variáveis de estado da qualidade da água. A quantidade de parâmetros abordados em um modelo pode afetar sua aplicabilidade. É crucial avaliar se a quantidade de parâmetros estabelecida é suficiente para representar os requisitos necessários ao objeto de estudo (Sánchez-Gutiérrez; Gómez-Castro, 2021; Uddin *et al*., 2023).

Uma vez definido os parâmetros de qualidade da água a serem simulados, o modelo precisa ser calibrado. Isso significa ajustar os coeficientes cinéticos das equações matemáticas que representam os processos físicos, químicos e biológicos no curso d'água, buscando um melhor ajuste entre os perfis simulados e os dados de qualidade medidos em campo. Portanto, a qualidade dos resultados de um modelo matemático depende tanto da conceituação quanto da qualidade dos dados utilizados na calibração (Salla; Alamy Filho; Pereira; Kim *et al.* 2023).

2.2. Softwares para MMQA

A evolução dos softwares de modelagem matemática de qualidade da água (MMQA) acompanha os avanços tecnológicos e as necessidades crescentes de uma gestão eficaz dos recursos hídricos. Desde os primeiros modelos simplificados até as plataformas integradas e sofisticadas atuais, esses softwares têm se tornado ferramentas indispensáveis para pesquisadores, gestores ambientais e formuladores de políticas públicas (Ejigu, 2021; Al-Dalimy; Al-Zubaidi, 2023).

Os primeiros modelos de qualidade da água surgiram com o objetivo de compreender e prever a distribuição de oxigênio dissolvido e demanda bioquímica de oxigênio em corpos d'água. O modelo Streeter-Phelps, apesar de desenvolvido em 1925, só começou a ser amplamente aplicado com o advento dos computadores nas décadas de 1960 e 1970. Durante esse período, os modelos eram essencialmente analíticos e baseavam-se em equações diferenciais simples (Von Sperling, 2014b; Ejigu, 2021).

Com o avanço da computação, os modelos de qualidade da água tornaram-se mais complexos, incorporando uma gama maior de parâmetros e processos (Hossain; Bappy; Sathi, 2023; Pradipta; Hendriyanto, 2024). Os softwares começaram a incluir a simulação de nutrientes (como nitrogênio e fósforo), metais pesados, substâncias tóxicas e microrganismos patogênicos. Existem diversas ferramentas e softwares utilizados para a modelagem da qualidade da água em ambientes lênticos e lóticos; alguns dos mais conhecidos incluem:

1. **QUAL2K e QUAL2Kw**: Modelos de qualidade da água para rios e córregos, desenvolvidos pela Agência de Proteção Ambiental dos Estados Unidos (EPA). Eles são amplamente utilizados para simular processos de oxidação, nitrificação e eutrofização (Al-Dalimy *et al.,* 2023, Hossain *et al.,* 2023; Lashani *et al.*, 2024, Pradipta *et al.*, 2024);
2. **WASP (*Water Quality Analysis Simulation Program*)**: Também desenvolvido pela EPA, o WASP é um modelo abrangente que simula uma ampla gama de processos de qualidade da água em diferentes ambientes aquáticos, incluindo rios, estuários e lagos. Permite a modelagem de substâncias químicas complexas e o desenvolvimento de cenários detalhados de gestão ambiental, sendo amplamente

utilizado por agências reguladoras e pesquisadores (Salla; Alamy Filho; Pereira, 2013; Wool *et al.*, 2020; Zelazny *et al.*, 2023);

3. **AQUATOOL**: Desenvolvido pelo Instituto de Engenharia de Água e Meio Ambiente da Universidade Politécnica de Valencia, Espanha. Este software é um sistema de suporte à decisão que integra modelos hidrológicos e de qualidade da água, sendo usado para a gestão de bacias hidrográficas e a simulação de cenários de qualidade da água (Salla *et al.*, 2013, 2014; Costa *et al.*, 2019; Arias-Aguila *et al.*, 2024). O AQUATOOL permite uma análise abrangente, combinando dados hidrológicos, hidráulicos e de qualidade da água para fornecer uma visão integrada da bacia hidrográfica;
4. **QUAL-UFMG**: Desenvolvida por Von Sperling (2014b), esta ferramenta permite a modelagem de rios através do programa no Excel® baseada no modelo QUAL-2E (versão anterior do QUAL2K). A QUAL-UFMG é conhecida por sua interface simplificada e amigável, facilitando o entendimento e uso por parte de profissionais e pesquisadores (Gomides *et al.*, 2024, Pani *et al.*, 2024), mesmo aqueles com menor experiência em modelagem matemática;
5. **HEC-RAS**: Um sistema de modelagem de fluxo de água desenvolvido pelo Corpo de Engenheiros do Exército dos Estados Unidos, que inclui módulos para qualidade da água (Rinjani *et al.*, 2023);
6. **SWAT (*Soil and Water Assessment Tool*)**: Utilizado para simular a quantidade e a qualidade da água em bacias hidrográficas, incluindo processos de escoamento, erosão e transporte de nutrientes e pesticidas (Mekonnen; Tenagashawu, 2023; Zhang *et al.*, 2024);
7. **MOHID**: Um sistema de modelagem hidrodinâmica e de qualidade da água desenvolvido pelo MARETEC (Centro de Investigação em Tecnologia Marinha e Ambiental) do Instituto Superior Técnico, pertencente à Universidade de Lisboa em Portugal. Esta ferramenta permite uma avaliação tridimensional dos modelos, além de ser adaptada para ambientes costeiros e estuarinos (Sánchez-Gutiérrez; Gómez-Castro, 2021).

Atualmente há uma integração mais robusta dos modelos de qualidade da água com sistemas de informação geográfica (SIG) e bases de dados avançadas. A automação dos processos de calibração e validação também se tornou uma característica padrão, aumentando significativamente a precisão e a confiabilidade das simulações. Além disso,

a integração entre inteligência artificial (IA) e algoritmos evolutivos está revolucionando os modelos de qualidade da água, oferecendo métodos mais eficientes para a calibração e previsão de parâmetros. A sequência traz uma revisão sucinta sobre calibração de parâmetros de qualidade da água, inteligência artificial e interação de modelo de qualidade da água com SIG.

2.3. Calibração de parâmetros de qualidade da água

A calibração de parâmetros de qualidade da água é considerada uma das etapas mais importantes do processo de modelagem e consiste no ajuste dos perfis simulados dos parâmetros aos dados monitorados. Segundo Bonganha *et al.* (2007), os parâmetros e as condições de contorno são calibrados de maneira que as simulações de qualidade da água se assemelhem aos dados observados no sistema real. O processo de calibração mais utilizado é o de "tentativa e erro", onde os parâmetros são ajustados manualmente (Bonganha *et al.*, 2007; Salla *et al.*, 2016) até se obter um valor aceitável entre os dados calculados e aqueles observados, considerando, assim, que o modelo está calibrado.

Além desse método empírico, existem abordagens convencionais, como a utilização de métodos determinísticos, entre eles a minimização da soma dos quadrados dos erros, em que o erro é definido como a diferença entre o valor observado e o valor estimado. Brum *et al*. (2022) propuseram uma alternativa baseada na minimização do erro por meio do coeficiente de eficiência *Nash-Sutcliffe*, além de um método probabilístico, no qual são geradas aleatoriamente mil combinações de valores dos coeficientes para identificar a melhor configuração.

Outro processo empregado na calibração é a automatização, realizada por um método de otimização, que, através de algoritmos, consegue convergir ao menor valor possível da soma dos quadrados dos erros (Von Sperling, 2014b). Devido à importância da calibração de um modelo matemático, diversos estudos abordaram o assunto tanto no âmbito nacional como no âmbito internacional (Ejigu, 2021; Aldrees *et al.*, 2022; Uddin *et al.*, 2023).

2.4. Inteligência Artificial e MMQA

A inteligência artificial (IA) tem sido amplamente adotada para enfrentar desafios técnicos ecológicos, especialmente em simulações de qualidade da água (Aldrees *et al.*, 2022).

Diversos estudos exemplificam a aplicação de IA para melhorar a modelagem e previsão dos parâmetros de qualidade da água.

Rajaee e Boroumand (2015) utilizaram redes neurais artificiais, funções *wavelets*, algoritmo genético e vetores de suporte para regressão linear multivariada para prever concentrações de clorofila-a na Baía de São Francisco, Estados Unidos. Para comparar os resultados dos modelos, dos valores preditos e reais, os autores utilizaram o coeficiente de eficiência do modelo de *Nash-Sutcliffe* e a raiz do erro quadrático médio. Séries temporais de clorofila-a foram computadas e avaliadas com o coeficiente de determinação para verificar qual seria o melhor ajuste para os dados de entrada do modelo.

Sahay e Dutta (2009) aplicaram algoritmos genéticos para derivar uma nova expressão para a previsão do coeficiente de dispersão longitudinal em rios naturais, o qual fez uso de poucos parâmetros geométricos (largura e profundidade do rio) e hidráulicos (média da seção transversal e velocidades de cisalhamento). Na avaliação de desempenho desta nova expressão, os autores realizaram comparações das previsões da nova fórmula com outras expressões convencionais, usando dados publicados de 29 rios nos Estados Unidos. O estudo de comparação mostrou que a nova expressão tem o menor erro quadrático médio, o maior coeficiente de correlação e a melhor taxa de discrepância.

Na pesquisa de Liu *et al.* (2014), os autores abordaram o problema da identificação multiparamétrica de um modelo de qualidade da água de rios, destacando a dificuldade na implementação de algoritmos de otimização tradicionais. Eles propuseram um novo modelo de identificação de parâmetros baseado em um algoritmo genético (AG) acoplado ao método de diferenças finitas (FDM). Esse modelo visa determinar parâmetros hidráulicos e de qualidade da água, como o coeficiente de dispersão longitudinal, o coeficiente de degradação de poluentes e a velocidade do fluxo. O método foi validado através de dois casos numéricos (em escoamentos estacionários e não estacionários, respectivamente) e uma aplicação prática. Os resultados indicaram que o modelo pode fornecer bons resultados de precisão de identificação e mostrou boas habilidades antirruído para a qualidade da água.

Aldrees *et al.* (2022) propuseram Modelos de Programação Multi-Expressão (MEP) baseados em Algoritmos Evolutivos (EA) para a previsão de condutividade específica e sólidos totais dissolvidos na água. Para isso, utilizaram uma base de dados com 360 leituras mensais, contendo parâmetros como temperatura da água, magnésio, cálcio, sódio, sulfato, cloreto, pH e bicarbonatos. A precisão, confiabilidade e capacidade de generalização dos modelos foram avaliadas com diversas métricas estatísticas, incluindo coeficiente de determinação, erro médio absoluto, erro médio absoluto percentual, erro quadrático médio logarítmico e erro quadrático médio. Comparando com modelos tradicionais de regressão múltipla não linear, os autores descobriram que estes modelos MEP baseados em EA são mais eficazes na previsão de problemas complexos, devido à sua maior eficiência e capacidade de generalização. Além disso, a análise de sensibilidade revelou que todas as oito variáveis consideradas afetam a previsão dos parâmetros de qualidade da água, com uma sensibilidade acima de 0,5 para algumas delas. Esses modelos não só oferecem correlações mais precisas, mas também podem ser úteis para profissionais e tomadores de decisão, ajudando a economizar tempo e recursos na monitorização da qualidade da água.

2.5. Interação de MMQA com SIG

Para construir simulações de qualidade ao longo de um rio, é necessário um modelo capaz de imitar o processo real, embora, utilize simplificações e suposições (Komarudin *et al.*, 2015). Quando integrados com Sistemas de Informações Geográficas (SIG), esses modelos podem utilizar dados espaciais detalhados, como topografia, uso do solo, redes de drenagem e dados meteorológicos, para fornecer uma análise mais precisa e contextualizada das condições da água em diferentes regiões.

O desenvolvimento de técnicas de sensoriamento remoto e as capacidades dos SIG incentivaram e melhoraram o uso expandido de modelos de bacias hidrográficas em todo o mundo (Vázquez; Mokrova, 2019). Os SIG oferecem uma plataforma robusta para a visualização e manipulação de dados espaciais, facilitando a identificação de fontes de poluição e a análise de padrões de distribuição de contaminantes. Por exemplo, através do mapeamento de bacias hidrográficas e áreas de agricultura, é possível identificar áreas críticas que contribuem para a degradação da qualidade da água, especialmente aquelas com potencial de poluição difusa. Entre as ferramentas de modelagem de qualidade de

água que se integram ao SIG, destacam-se AQUATOOL-GESCAL (Arias-Aguila *et al.*, 2024), SWAT (Zhang *et al.*, 2024), HEC-HAS (Rinjani *et al.*, 2023), WARM-GIS Tools (Paiva *et al*., 2024, Santos *et al*., 2024), entre outras.

A conexão entre modelos de qualidade da água e SIG também aprimora a capacidade de simulação e predição. Modelos preditivos podem ser calibrados e validados com dados reais obtidos de sensores e monitoramento ambiental, processados e georreferenciados em SIG, aumentando a confiança na precisão da modelagem (Alamy Filho *et al*., 2019). Isso não apenas melhora a precisão das previsões, mas também permite a visualização dos resultados em mapas interativos, facilitando a comunicação de riscos e a identificação de áreas prioritárias para intervenções.

Além disso, na tomada de decisão e na resolução de problemas relacionados aos recursos hídricos, os SIG permitem a visualização e interpretação das informações de entrada e saída dos modelos de simulação. Essa funcionalidade possibilita uma interação dinâmica do usuário com os dados e com a área de estudo, proporcionando uma visão mais clara da região de interesse (Fan *et al.*, 2013).

3. MATERIAIS E MÉTODOS

Para o correto desenvolvimento do aplicativo QUALITOOL 2.0, foi necessário compreender, além de uma linguagem de programação, as características geométricas e hidráulicas, bem como as equações matemáticas que governam o processo de autodepuração. Isso inclui equações de advecção-difusão, utilizadas para simular o comportamento do poluente na massa líquida, equações que descrevem os processos físicos, químicos e biológicos para diversos parâmetros de qualidade da água, além da equação da mistura.

A seguir, são detalhadas as características geométricas (item 3.1) e hidráulicas (item 3.2), bem como os equacionamentos matemáticos utilizados na versão 2.0 do aplicativo QUALITOOL (itens 3.3 e 3.4). Também são apresentadas as metodologias adotadas para a realização da análise de sensibilidade (item 3.5) e calibração (item 3.6) dos coeficientes empregados, além das metodologias desenvolvidas para otimizar a interação gráfica com o usuário (item 3.7), para o processo de modelagem com coeficientes predefinidos (item 3.8) e para a otimização de coeficientes por meio de algoritmos evolucionários (item 3.9).

3.1. Características geométricas

O modelo matemático implementado no aplicativo QUALITOOL 2.0 leva em consideração o escoamento permanente (estado estacionário) e uniforme a cada intervalo discretizado. Em função do modelo ser unidimensional na direção longitudinal, a equação de mistura completa, em cada segmento ou trecho, e o modelo hidráulico de fluxo, em pistão entre segmentos ou trechos, foram os que melhor se ajustaram (Von Sperling, 2014b).

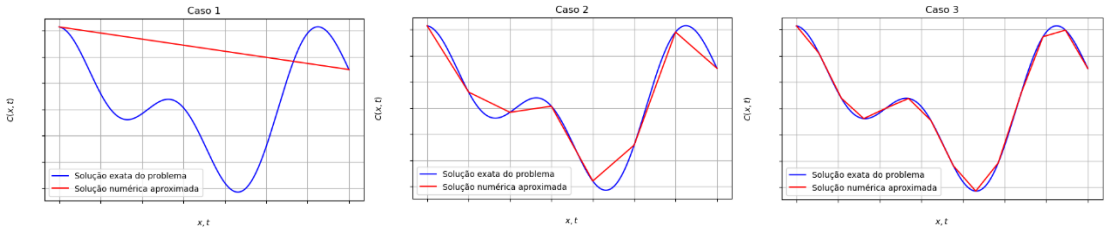
3.1.1. Discretização

Na área de modelagem matemática, discretizar significa dividir ou particionar um todo em partes menores, seja em relação ao tempo ou à distância. A solução numérica das equações de advecção-difusão e das equações dos diversos parâmetros de qualidade da água pelo Método de Diferenças Finitos (MDF) de primeira ordem foi incorporada no aplicativo QUALITOOL 2.0.

O MDF é uma técnica amplamente utilizada na matemática e na engenharia para transformar problemas contínuos em problemas discretos. Esse método é particularmente eficaz para resolver equações diferenciais ao discretizar o domínio contínuo em uma grade de pontos ou nós.

Para facilitar a compreensão, a Figura 1 ilustra o procedimento descrito, apresentando uma função hipotética (em azul), que serve como solução exata para uma determinada equação diferencial, e uma função polinomial de primeira ordem (em vermelho), utilizada como aproximação numérica para a resolução do problema. São exibidos três casos com diferentes níveis de discretização. Observa-se que, no Caso 1, onde não há discretização, a solução numérica não proporciona uma aproximação adequada da solução exata. Nos casos subsequentes, à medida que o tamanho do intervalo de discretização diminui, a qualidade da aproximação numérica em relação à solução exata melhora.

Figura 1. Esquema ilustrativo do procedimento de adoção da aproximação numérica em métodos numéricos e da qualidade da aproximação em função do nível de discretização.



Fonte: Autora (2025).

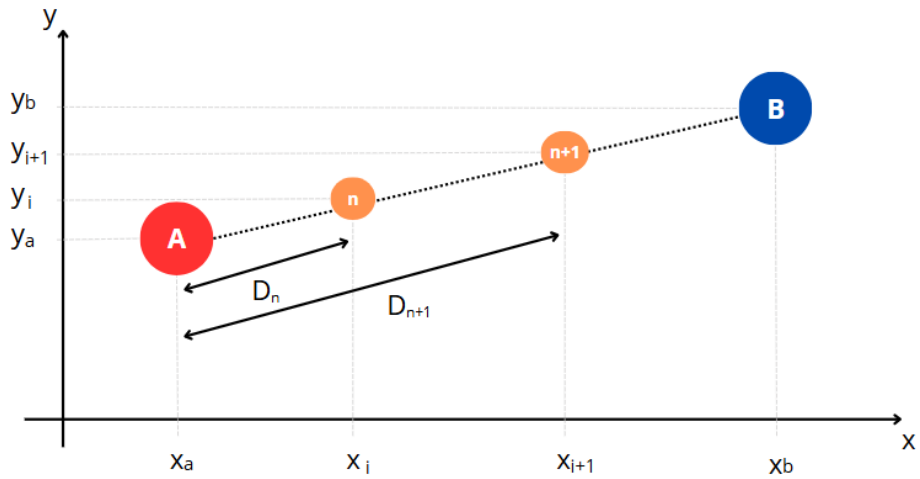
É importante ressaltar que o usuário não apresentará uma solução exata do problema, mas sim uma solução simplificada do trecho a ser modelado, com auxílio de dados de entrada coletados em campo.

A implementação do Método das Diferenças Finitas (MDF) no aplicativo QUALITOOL 2.0 envolve a determinação das coordenadas x e y a partir de um vetor gerado com base na diferença entre um comprimento D e um valor inicial, ajustado por um ângulo. Na prática, essa transformação utiliza funções trigonométricas em que o cosseno e o seno do ângulo são empregados para calcular as coordenadas x e y.

A Figura 2 apresenta um exemplo no qual será realizada a discretização de um comprimento D entre os pontos espaciais A (x_a, y_a) e B (x_b, y_b). Inicialmente, são calculados o comprimento e o ângulo entre A e B. A precisão dessa abordagem é garantida

quando o ângulo está restrito ao intervalo de 0 a $\pi/2$. Caso o ângulo não se enquadre nesse intervalo, ele será ajustado para atender aos critérios estabelecidos, assegurando que o cálculo das coordenadas seja preciso e consistente com o intervalo desejado.

Figura 2. Exemplo da discretização dos dados georreferenciados.



Fonte: Autora (2025).

Por fim, o novo ponto *n*, que pertence ao intervalo entre os pontos *A* e *B*, será determinado pelas expressões:

$$\begin{cases} x_i = x_a + D_n \,.\, \text{cosseno}(\hat{a}ngulo) \\ y_i = y_a + D_n \,.\, \text{seno}(\hat{a}ngulo) \end{cases}$$

Eq.1 e 2

Esse processo será repetido sucessivamente até que D_{n+1} seja igual ou superior ao comprimento entre *A* e *B*.

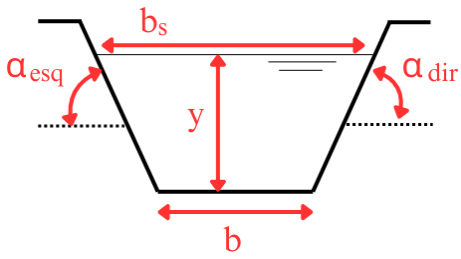
A priori, o valor inicial definido pelo usuário para a discretização (ou seja, o intervalo do comprimento das séries em relação à distância total, em metros) será uniforme ao longo do trecho. Esse valor influencia diretamente no cálculo das seguintes variáveis por trecho: latitude e longitude (UTM); altitude (m); comprimento (m); declividade longitudinal (m/m); largura de base (m); ângulos dos taludes da margem esquerda e direita (graus); e coeficiente de rugosidade de Manning (adimensional).

3.1.2. Seção transversal

Existe uma dificuldade de se atribuir uma seção definida para cursos de água naturais. De forma simplificada, o aplicativo QUALITOOL 2.0 permite escolher três

seções transversais com geometrias regulares. De acordo com a Figura 3, a partir do ajuste da largura de base ou de fundo b e ângulos laterais (α_{esq} e α_{dir} são os ângulos do talude (radiano), do lado esquerdo e direito, respectivamente), é possível definir a geometria transversal em trapezoidal, retangular e triangular.

Figura 3. Ilustração da seção transversal aplicada no QUALITOOL 2.0.



Fonte: Autora (2025).

As equações 3 e 4 descrevem o cálculo da área molhada ($\mathbf{A_m}$) (em m^2) e do raio hidráulico ($\mathbf{R_h}$) (em metro); se a seção transversal for retangular, os ângulos do talude serão de 90° (1,5708 rad.); se a seção transversal for triangular, o valor de b será igual a zero.

$$\boldsymbol{A_m} = \frac{y}{2} . \left[2 .\ b + \left(y . \cot \alpha_{esq}\right) + \left(y . \cot \alpha_{dir}\right)\right]$$

Eq. 3

$$\boldsymbol{R_h} = \frac{A_m}{b + \frac{y}{\sin \alpha_{esq}} + \frac{y}{\sin \alpha_{dir}}}$$

Eq. 4

Nas quais: b é a largura superficial (em metro) e y é a profundidade líquida média (em metro).

3.2. Características hidráulicas

3.2.1. Declividade

Para o cálculo da declividade ($\mathbf{S_o}$) entre os pontos A e B (em m/m), calcula-se a diferença da altitude (em metro) no ponto B ($\mathbf{Z_b}$) e a altitude (em metro) no ponto A ($\mathbf{Z_a}$), e divide-se esse valor pelo $\mathbf{D_{a\text{-}b}}$ (em metro), conforme mostra as Equações 5 e 6.

$$S_o = \frac{Z_b - Z_a}{D_{a-b}}$$

Eq. 5

$$D_{a-b} = \sqrt{(Y_b - Y_a)^2 + (X_b - X_a)^2}$$

Eq. 6

Na qual: $D_{a\text{-}b}$ é a distância do ponto A e B (m); Y_a é a latitude (UTM) do ponto A; Y_b é a latitude (UTM) do ponto B; X_a é a longitude (UTM) do ponto A; X_b é a longitude (UTM) do ponto B.

Se a declividade do trecho for negativa ou igual a zero, adota-se um valor de 0.000001 m/m, para que seja possível utilizar a equação de Manning em escoamento permanente e uniforme.

3.2.2. Profundidade líquida e velocidade média do escoamento

A profundidade líquida pode ser obtida diretamente em campo, caso seja instalada uma estação fluviométrica. Na ausência desta, equações matemáticas podem ser utilizadas para sua estimativa; similar a outros modelos conhecidos, tais como AQUATOOL, QUAL2K e WASP, o aplicativo QUALITOOL 2.0 considerou a equação de Manning na estimativa da profundidade líquida. A equação de Manning (Eq. 7) considera o escoamento permanente e uniforme, partindo-se do princípio que, em cada trecho ou segmento, as variáveis do escoamento vazão volumétrica, declividade de fundo, geometria da seção transversal e rugosidade das paredes laterais e fundo são constantes.

$$\frac{Q \,.\, \eta}{\sqrt{S_o}} = A_m \,.\, {R_h}^{2/3}$$

Eq. 7

Na qual: A_m é a área molhada da seção transversal (m^2); R_h é o raio hidráulico, (m); Q é a vazão (m^3/s); η é o coeficiente de rugosidade de Manning ($m^{-1/3}.s$); S_o é a declividade longitudinal de fundo (m/m).

A equação de Manning, no QUALITOOL 2.0, é solucionada de forma empírica, onde os termos geométricos ($\boldsymbol{A_m \,.\, {R_h}^{2/3}}$) e hidráulicos ($\boldsymbol{Q \,.\, \eta / \sqrt{S_o}}$) são agrupados no mesmo lado da equação. Nesta equação, a única incógnita da função é a profundidade

líquida. Ou seja, a função representada pelo termo geométrico subtraído do termo hidráulico é igual a zero.

A adoção da equação de Manning para a quantificação da profundidade líquida traz limitações, uma vez que considera o escoamento permanente e uniforme para cada trecho de rio. Neste contexto, a metodologia adotada no aplicativo QUALITOOL 2.0, de permitir a entrada de dados georreferenciados da topologia do sistema hídrico para obter a declividade longitudinal de fundo, minimiza a limitação da equação de Manning. Ou seja, a declividade longitudinal de fundo é fixa entre dois pontos de um trecho. Logo, a maior quantidade de pontos representativos da topologia aumenta a discretização da declividade longitudinal de fundo.

Encontrado o valor do nível líquido, detalhado anteriormente, calcula-se a velocidade média do escoamento através da equação da continuidade, conforme ilustra a Equação 8.

$$V = \frac{Q}{A_m}$$

Eq. 8

Na qual: V é a velocidade média, em m/s; Q é a vazão volumétrica, em m^3/s; Am é a área molhada da seção transversal, em m^2.

3.2.3. Tensão de arraste e número de Froude

Para um melhor entendimento do comportamento do fluido, o QUALITOOL 2.0 realiza os cálculos da tensão de arraste e do número de Froude. A tensão de arraste é a tensão de cisalhamento da água nas paredes laterais e no fundo do canal e é calculada pela Equação 9.

$$\tau = \gamma \, . \, R_h \, . \, S_o$$

Eq. 9

Na qual: τ é a tensão de arraste ou de cisalhamento (N/m^2) e γ é o peso específico da água (N/m^3).

O número de Froude desempenha importante papel no estudo dos canais, permitindo definir os regimes de escoamento (subcrítico, supercrítico e crítico) e é calculado por meio da Equação 10.

$$Fr = \frac{V}{\sqrt{g \cdot y}}$$

Eq. 10

Na qual: *Fr* é o Número de Froude (adimensional); *V* é a velocidade média do escoamento na seção (m/s); *g* é a aceleração da gravidade (m²/s); *y* é a profundidade líquida (m).

3.3. Equação de advecção-difusão

O modelo de qualidade da água utilizado se baseia nos fundamentos da equação de advecção-difusão, bem como no modelo clássico de Streeter-Phelps e suas adaptações, descritas por Chapra (1997) e Von Sperlin (2014b). É possível modelar os seguintes constituintes: demanda bioquímica de oxigênio (DBO); oxigênio dissolvido (OD); nitrogênio total e suas frações (orgânico, amoniacal, nitrito e nitrato); fósforo total e suas frações (orgânico e inorgânico); e coliformes termotolerantes (fecais).

As simplificações adotadas no modelo de qualidade da água no aplicativo QUALITOOL 2.0 incluem:

- Escoamento simplificado do fluxo em pistão ideal, com o coeficiente de dispersão nulo, por se tratar de rios e córregos;
- Mistura total e imediata nos pontos de lançamento;
- Integração dos cálculos pelo método de Euler;
- Reações de primeira ordem dos termos cinéticos;
- Correções da influência da temperatura sobre os coeficientes cinéticos, com base na teoria de Arrhenius;
- Modelagem dos parâmetros OD e DBO em condições de anaerobiose; e
- Desconsideração do termo difusivo turbulento, devido ao foco em ambientes lóticos.

A equação de advecção-difusão é utilizada na simulação do comportamento de um poluente na massa líquida ao longo do tempo e distância (BENEDINI, 2011), conforme ilustra a Equação 11.

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} + w\frac{\partial C}{\partial z} = \frac{\partial}{\partial x}\left(D_x\frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial y}\left(D_y\frac{\partial C}{\partial y}\right) + \frac{\partial}{\partial z}\left(D_z\frac{\partial C}{\partial z}\right) \pm Fonte$$

Eq. 11

Na qual: C é a concentração do parâmetro no tempo t (mg/L); t é o tempo decorrido desde a zona de mistura entre o contaminante e o meio líquido (dia); D_x, D_y e D_z correspondem aos coeficientes de difusão turbulenta nas respectivas variáveis espaciais (m²/dia); u, v e w são as componentes do vetor velocidade nas direções x, y e z (m/dia), respectivamente; Fonte é o termo fonte-sumidouro.

A equação (11) considera os fenômenos de difusão (dispersão do poluente devido à diferença de concentração existente entre a massa poluidora e a massa líquida do curso de água), advecção (transporte do poluente devido ao movimento natural da água do curso hídrico) e de decaimento ou acúmulo da massa de um parâmetro de qualidade da água ao longo do tempo, ocasionado por processos físicos, químicos e biológicos (Chapra, 1997; Silvino, 2008).

No aplicativo QUALITOOL 2.0, em função de considerar escoamento unidimensional e permanente e desconsiderar a dispersão ou difusão do poluente, a Equação 11 é reescrita na forma da Equação 12.

$$u\frac{\partial C}{\partial x} = \pm Fonte$$

Eq. 12

A solução numérica, por diferença finita, da Equação 12, leva a Equação 13.

$$C_{n+1} = C_n \pm Fonte(t_{n+1} - t_n)$$

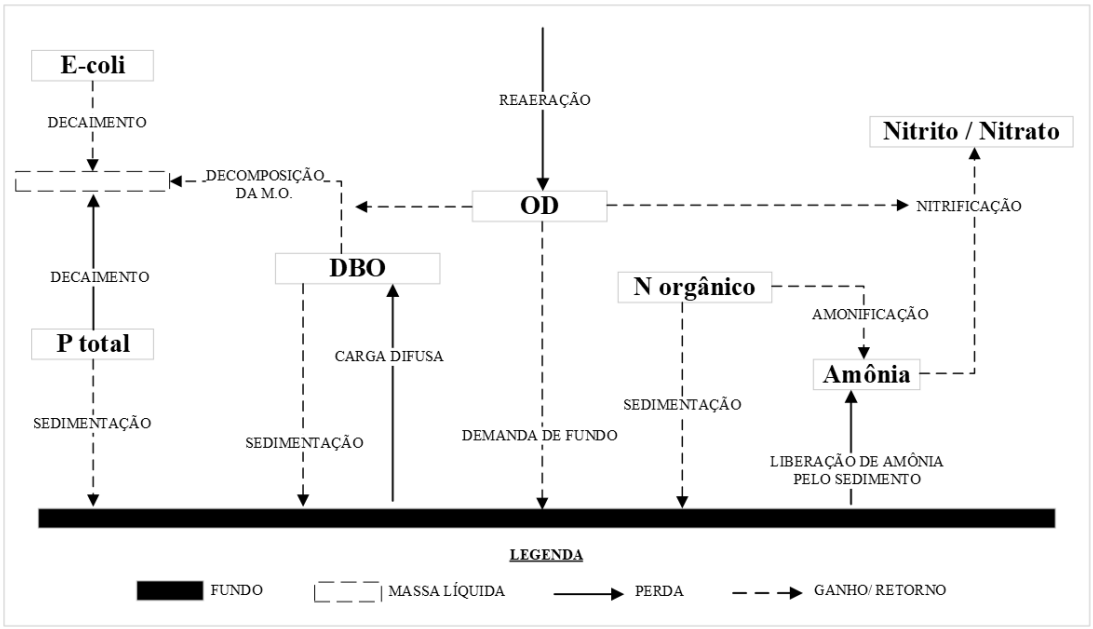
Eq. 13

Na qual: C_{n+1} representa a concentração do parâmetro no tempo t_{n+1}; C_n representa a concentração do parâmetro no tempo t_n.

O termo fonte/sumidouro representa o comportamento do poluente na massa líquida em função de processos físicos, químicos e bioquímicos. Esses processos advêm das interações entre os diversos parâmetros de qualidade da água, conforme ilustrada na Figura 4. Fonte para a simulação dos parâmetros OD, DBO, nitrogênio orgânico, nitrogênio amoniacal, nitrito, nitrato, fósforo orgânico e inorgânico e coliformes fecais (E-coli) são demonstradas nos itens a seguir.

A Figura 4 traz o esquema geral dos processos físicos e bioquímicos aos quais os parâmetros de qualidade da água estão sujeitos, além das interações entre os próprios parâmetros.

Figura 4. Esquema ilustrativo dos processos físicos e bioquímicos do modelo de qualidade da água do aplicativo QUALITOOL 2.0.



Fonte: Autora (2025).

A sequência traz o equacionamento matemático do termo fonte para cada parâmetro de qualidade da água.

3.3.1. Nitrogênio e suas frações

As equações 14 a 17 são representações simplificadas da modelagem do nitrogênio. As interações cinéticas não consideram a incorporação da amônia e do nitrato à biomassa de organismos aquáticos (algas) visto que QUALITOOL 2.0 simula o comportamento do poluente em ambiente lótico.

No modelo de nitrogênio são representados os seguintes processos: sedimentação do nitrogênio orgânico particulado ($\boldsymbol{k_{so} \,.\, N_{org}}$); conversão do nitrogênio orgânico a amônia, amonificação ($\boldsymbol{k_{oa} \,.\, N_{org}}$); liberação de amônia pelo sedimento de fundo ($\frac{\boldsymbol{S_{N_{amon}}'}}{\boldsymbol{y}}$); oxidação da amônia a nitrito, nitrificação ($\boldsymbol{k_{an} \,.\, f_{nitr} \,.\, N_{amon}}$); e oxidação do nitrito a nitrato, nitrificação ($\boldsymbol{k_{nn} \,.\, f_{nitr} \,.\, N_{nitri}}$).

$$\sum F_{N_{org}} = -k_{oa} \,.\, N_{org} - \, k_{so} \,.\, N_{org}$$

Eq. 14

$$\sum F_{N_{amon}} = k_{oa} \,.\, N_{org} - \; k_{an} \,.\, f_{nitr} \,.\, N_{amon} + \frac{S_{N_{amon}}{}'}{y}$$

Eq. 15

$$\sum F_{N_{nitri}} = k_{an} \,.\, f_{nitr} \,.\, N_{amon} - k_{nn} \,.\, f_{nitr} \,.\, N_{nitri}$$

Eq. 16

$$\sum F_{N_{nitra}} = k_{nn} \,.\, f_{nitr} \,.\, N_{nitri}$$

Eq. 17

Nas quais: N_{org} é a concentração de nitrogênio orgânico (mg/L); N_{amon} é a concentração de amônia (mg/L); N_{nitri} é a concentração de nitrito (mg/L); N_{nitra} é a concentração de nitrato (mg/L); k_{so} é o coeficiente de remoção do nitrogênio orgânico por sedimentação (1/dia); k_{an} é o coeficiente de conversão de amônia a nitrito (1/dia); k_{oa} é o coeficiente de conversão de nitrogênio orgânico a amônia (1/dia); k_{nn} é o coeficiente de conversão de nitrito a nitrato (1/dia); $S_{N_{amon}}{}'$ é o coeficiente de liberação de amônia pelo sedimento de fundo ($gO_2/m^2.d$); f_{nitr} é o fator de correção do coeficiente de nitrificação em função do OD (adimensional); e y é a profundidade do curso d'água (m).

O f_{nitr} (Eq. 18) representa o efeito da influência de fatores ambientais na conversão do nitrogênio, em ênfase, a redução da taxa de nitrificação em virtude da limitação ao crescimento das bactérias nitrificantes imposta por baixas concentrações de OD no curso d'água.

$$f_{nitr} = 1 \; - \; e^{-k_{nitrOD} \,.\, C}$$

Eq. 18

Na qual: C é a concentração de OD (mg/L); k_{nitrOD} é o coeficiente de inibição da nitrificação por baixo OD (L/mg).

3.3.2. Oxigênio Dissolvido

Na modelagem do OD (Eq. 19) não foram considerados os processos fotossintéticos e respiratórios dos seres clorofilados. O modelo apresenta os seguintes processos: cinética da reaeração ($\boldsymbol{k_2} \,.\, (\boldsymbol{C_s - C})$); a cinética da desoxigenação carbonácea, decomposição da matéria orgânica, considerando a sedimentação ($(\boldsymbol{k_d + k_s}) \,.\, \boldsymbol{L_r}$); a taxa de consumo de oxigênio pelos sistemas bentônicos, demanda de oxigênio do sedimento ($\frac{\boldsymbol{S_d{}'}}{\boldsymbol{y}}$); o consumo de oxigênio na nitrificação ($\boldsymbol{R_{O2-amon} \,.\, k_{an} \,.\, f_{nitr} \,.\, N_{amon}}$). Caso, se tenha $k_s = S_d{}' = \; l_{dr} = 0$ e $R_{O2-amon} = f_{nitr} = 1$, tem-se a equação clássica de Streeter-Phelps (Von SPerling, 2014b).

$$\sum F_C = k_2 \,.\, (C_s - C) - (k_d + \, k_s) \,.\, l_{dr} - \frac{S_d{}'}{y} - R_{O2-amon} \,.\, k_{an} \,.\, f_{nitr} \,.\, N_{amon}$$

Eq. 19

Na qual: C é a concentração de oxigênio dissolvido (mg/L); C_s é a concentração de saturação de oxigênio (mg/L); l_{dr} é a DBO remanescente (mg/L); k_2 é o coeficiente de reaeração (1/dia); k_d é o coeficiente de decomposição (1/dia); $S_d{}'$ é a demanda de oxigênio por unidade de área superficial do sedimento ($gO_2/m^2.d$); e $R_{O2-amon}$ é a relação entre o consumo de oxigênio e a oxidação da amônia (mgO_2/L por mgN_{amon}/L).

Se o resultado da Equação 19 fornecer valores negativos, a concentração do OD será igual a zero e a modelagem do OD e da DBO será influenciada pelas condições de anaerobiose.

3.3.3. Demanda Bioquímica de Oxigênio

A equação 20 apresenta o modelo de DBO em condições aeróbicas, representando os seguintes processos: decaimento ($\boldsymbol{k_d \,.\, L}$); sedimentação ($\boldsymbol{k_s \,.\, L}$); e carga difusa, sem vazão ($\frac{\boldsymbol{L_{rd}{}'}}{\boldsymbol{y}}$).

$$\sum F_L = -k_d \,.\, L - k_s \,.\, L \, + \frac{l_{rd}{}'}{y}$$

Eq. 20

Na qual: L é a DBO última (mg/L); k_s é o coeficiente de conversão de amônia em nitrito (1/dia); e $l_{rd}{}'$ é Carga difusa de DBO ($mg/m^2.d$).

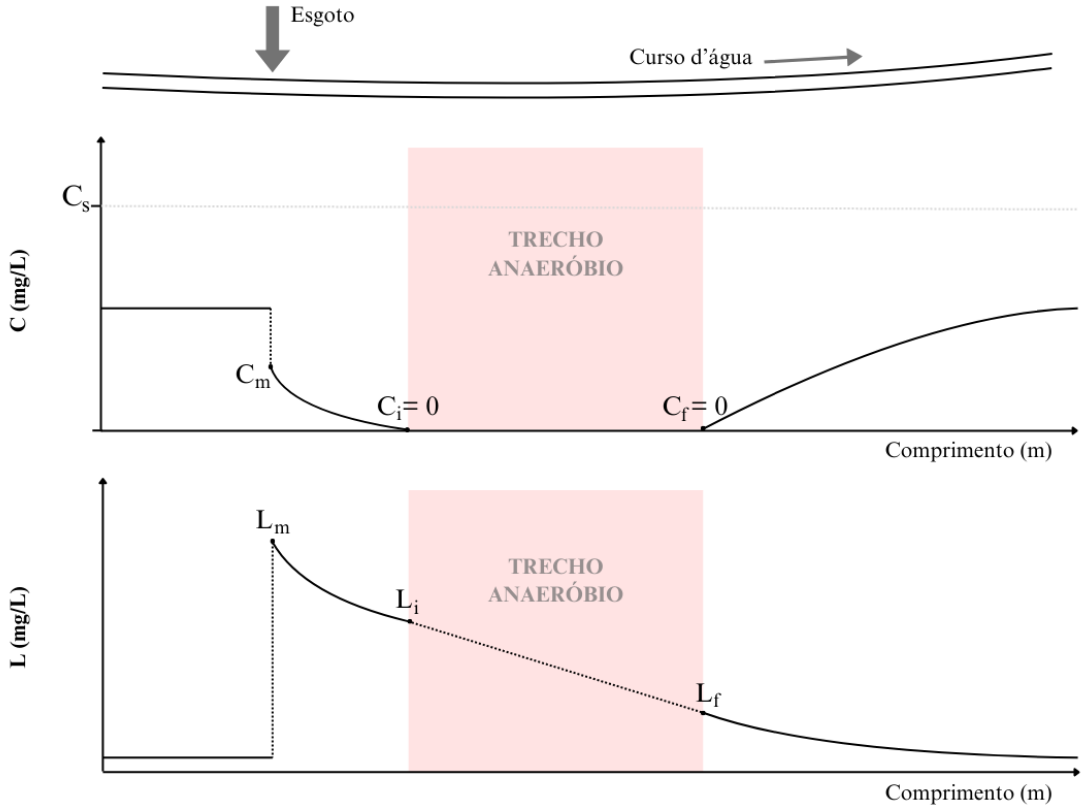
Na modelagem do DBO em condições de anaerobiose ou anóxicas (Eq. 21), Gundelach e Castillo (1976, apud Von Sperling, 2014b) consideraram as seguintes premissas: os mecanismos de transferência de oxigênio sob condições anaeróbias são os mesmos que atuam na fase aeróbia; todo oxigênio que entra na água é consumido na mesma velocidade de sua introdução; e a conversão da matéria orgânica por mecanismos puramente anaeróbios é considerada desprezível, devido à sua ocorrência se processar em taxas bastante lentas, cujo equacionamento matemático é representado na equação (21).

$$\sum F_L = -k_2 \,.\, C_s + \frac{L_{rd}{}'}{y}$$

Eq.21

A Figura 5 mostra o efeito deste decaimento linear da DBO no trecho anaeróbio.

Figura 5. Perfis de OD (C) e DBO (L) em condições de aerobiose e anaerobiose.



Fonte: Adaptado de Von Sperling (2014b).

As condições anaeróbias cessarão em um ponto em que a DBO tenha atingido um nível tal, que a taxa de oxigenação bioquímica seja igual ou menor à taxa de entrada do oxigênio na água (Eq. 22). Nesta condição, no QUALITOOL 2.0, foi condicionado a uma precisão de 0,001.

$$\begin{cases} k_d \,.\, L_f \leq k_2 \,.\, C_s - \dfrac{l_{rd}{}'}{y} \rightarrow retorno\ \text{à}\ aerobiose \\ k_d \,.\, L_f > k_2 \,.\, C_s - \dfrac{L_{rd}{}'}{y} \rightarrow permanência\ na\ anaerobiose \end{cases}$$

Eq.22

Na qual: L_f é a DBO última no ponto final do trecho em anaerobiose (mg/L).

3.3.4. Fósforo e suas frações

As equações 23 e 24 são representações simplificadas da modelagem do fósforo, onde não há modelagem de alga. Desta forma, este modelo não computa os mecanismos de conversão do fósforo inorgânico em biomassa algal e o acréscimo dos teores de fósforo

orgânico devido às algas. No modelo de fósforo são representados os seguintes processos: sedimentação do fósforo orgânico particulado ($\boldsymbol{k_{spo} \, . \, P_{org}}$); conversão do fósforo orgânico particulado a fósforo inorgânico dissolvido ($\boldsymbol{k_{oi} \, . \, P_{org}}$); e liberação de fósforo inorgânico dissolvido pelo sedimento de fundo ($\frac{\boldsymbol{S_{P_{inorg}}'}}{\boldsymbol{y}}$).

$$\sum F_{P_{org}} = - \, k_{oi} \, . P_{org} - \, k_{spo} \, . P_{org}$$

Eq. 23

$$\sum F_{P_{inorg}} = k_{oi} \, . P_{org} + \frac{S_{P_{inorg}}{}'}{y}$$

Eq. 24

Nas quais: P_{org} é a concentração de fósforo orgânico (mg/L); P_{inorg} é a concentração de fósforo inorgânico (mg/L); k_{spo} é o coeficiente de remoção do fósforo orgânico por sedimentação (1/dia); k_{oi} é o coeficiente de conversão de fósforo orgânico a fósforo inorgânico (1/dia); $S_{P_{inorg}}{}'$ é o coeficiente de liberação de fósforo inorgânico pelo sedimento de fundo ($gO_2/m^2.d$); e y é a profundidade do curso d'água (m).

3.3.5. Coliformes termotolerantes

A taxa de mortalidade de bactérias é calculada de acordo com a lei de Chick (Eq. 25), onde a taxa de remoção é proporcional à concentração de bactérias, caracterizando uma reação de primeira ordem. A taxa de microrganismos em diferentes corpos d'água está geralmente ligada a diferentes valores do coeficiente de decaimento ($\boldsymbol{k_b}$), o que depende tanto da natureza do organismo quanto das condições do meio aquático. Por exemplo, nas águas naturais, a mortalidade bacteriana ocorre mais rapidamente em regiões tropicais do que em regiões temperadas (Von Sperling, 2014b).

$$\sum F_N = k_b \, . N$$

Eq. 25

Na qual: N é o número de coliformes (NMP – Número Mais Provável – por 100 mL); e o k_b é o coeficiente de decaimento bacteriano (1/dia).

3.3.6. Coeficientes cinéticos da literatura

A Tabela 1 apresenta os coeficientes cinéticos e de sedimentação inseridos no aplicativo QUALITOOL 2.0, além dos símbolos, os parâmetros associados, as unidades

usuais, os valores base recomendados para uma temperatura de 20°C e suas respectivas referências, além dos valores do coeficiente de temperatura (θ), adimensional, para correção da temperatura.

Tabela 1. Valores dos coeficientes predefinidos.

	Símbolo	Descrição	Unidades	Valores (20°C)	θ	Fonte
OD	k_2	Coeficiente de reaeração	1/dia	-	1,024	A
	k_1	Coeficiente de desoxigenação	1/dia	0,08 – 0,45[1]	-	A
	$R_{O2\text{-}namon}$	Consumo de oxigênio para oxidação da amônia	mgC/mgN_{amon}	3,20	-	A
DBO	k_d	Coeficiente de decomposição da matéria orgânica carbonácea no rio	1/dia	0,08 – 1,00[1]	-	A
	k_s	Coeficiente de sedimentação	1/dia	0,05 – 0,35[1]	-	A
	l_{rd}	Carga difusa de DBO	$gL_5/m.d$	-	-	A
	S_d	Demanda de fundo de OD no rio	gC/m.d	0,05 – 10,00[2]	-	A
N	k_{oa}	Coeficiente de conversão de N orgânico em amônia	mgC/mgN_{amon}	0,15 – 0,25	1,047	A
	k_{so}	Coeficiente de sedimentação do N orgânico	1/dia	0,00 – 0,10[3]	1,024	A
	k_{an}	Coeficiente de conversão da amônia a nitrito	1/dia	0,15 a 0,25	1,080	A
	S_{Namon}	Ressurgimento de fundo da amônia	g/m².d	0,00 – 0,50[4]	-	A
	k_{nn}	Coeficiente de conversão de nitrito em nitrato	1/dia	0,10 – 1,00[4]	1,047	A
	k_{nitrOD}	Coeficiente de inibição da nitrificação por baixo OD	L/mg	0,60	-	A
P	k_{oi}	Coeficiente de conversão do P orgânico a P inorgânico	1/dia	0,20 – 0,30	1,047	A
	k_{spo}	Coeficiente de sedimentação do P orgânico	1/dia	0,02 – 0,05	1,024	A
	S_{Pinorg}	Fluxo de liberação de P inorgânico pelo sedimento de fundo	$gO_2/m^2.d$	0,00 – 0,20	1,074	A
E-coli	k_b	Coeficiente de decaimento de coliforme	1/dia	0,05 – 1,50	1,070	A

Obs.: variações de acordo com: [1]concentração do efluente; [2]tipo de fundo, localização e condições e fluxo; [3]velocidade e concentração do efluente; [4]profundidade do efluente. Fonte: A=Von Sperling (2014b).

Os valores do coeficiente de reaeração k_2, utilizado na modelagem de OD e DBO, podem ser definidos com base em valores médios tabelados ou em função das características hidráulicas do corpo d'água. Se a profundidade do leito no trecho estiver entre 0,1 e 4,0 m e a velocidade entre 0,05 e 1,60 m/s, as equações apresentadas na Tabela 2 devem ser utilizadas.

Tabela 2. Equações referentes ao coeficiente k2 (20ºC).

Pesquisador	Fórmula	Eq.	Faixa de aplicação aproximada
O'Connor & Dobbins (1958)	$3{,}73\ v^{0,5}\ H^{-1,5}$	10	0,6m ≤ H < 4,0m 0,05m/s ≤ v < 0,8m/s
Churchill *et al.* (1962)	$5{,}0\ v^{0,97}\ H^{-1,67}$	11	0,6m ≤ H < 4,0m 0,8m/s ≤ v < 1,5m/s
Owens *et al.* (apud Branco, 1978)	$5{,}3\ v^{0,67}\ H^{-1,85}$	12	0,1m ≤ H < 0,6m 0,05m/s ≤ v < 1,5m/s

Notas:
v: velocidade do curso d'água (m/s); H: altura da lâmina d'água (m)

Fonte: Adaptado de Von Sperling (2014b).

Caso a condição anterior não seja atendida, a modelagem passará a considerar a vazão no trecho: se a vazão estiver entre 0,03 e 8,5 m³/s, será aplicada a equação de Thomann e Mueller (1987, apud Von Sperling, 2014b). Se a vazão estiver entre 0,3 e 8,5 m³/s, será utilizada a Eq. 26 para rios médios, ou a Eq. 27 para rios pequenos, caso a vazão esteja entre 0,03 e 0,30 m³/s.

$$k_2 = 15{,}4\ v\ i$$

Eq. 26

$$k_2 = 31{,}6\ v\ i$$

Eq. 27

Na qual: v é a velocidade do curso d'água (m/s); i é a declividade do curso d'água (m/km).

Se nenhuma dessas condições for atendida, será utilizada a Equação de O'Connor (Eq. 28). É importante avaliar cuidadosamente se as condições anteriores não foram atendidas, pois, neste caso, serão utilizados os valores médios de m = 20,74 e n = -0,42, estipulados para os rios principais da Região Metropolitana de Belo Horizonte no período seco, obtidos pelo Von Sperling (2014b) e apresentados na Tabela 3.

$$k_2 = m\ Q^n$$

Eq. 28

Na qual: Q é a vazão (m³/s); m e n são coeficientes da equação.

Tabela 3. Valores dos coeficientes *m* e *n* para diversos cursos d'água da Região Metropolitana de Belo Horizonte (período seco).

Tipo de Rio	Estação fluviométrica	*m*	*n*
Rio principal (Q > 10m³/s)	Rio das Velhas em Rio Acima	16,58	-0,38
	Rio das Velhas em Honório Bicalho	13,79	-0,32
	Rio das Velhas jusante Rib. Sabará	5,44	-0,23
	Rio das Velhas em Pinhões	17,00	-0,46
	Rio das Velhas em Ponte Raul Soares	67,99	-0,87
	Rio das Velhas em Ponte Nova do Paraopeba	3,62	-0,28
	Média dos rios principais	**20,74**	**-0,42**

Fonte: Von Sperling (2014b).

3.4. Equação de mistura

Uma das simplificações adotadas no aplicativo QUALITOOL 2.0 foi a mistura total e imediata nos pontos de lançamento de carga poluente; ou seja, nos pontos de relação rio/tributário ou rio/esgoto e afins, haverá a mistura completa e instantânea dos constituintes, desconsiderando o efeito de abertura da pluma. A equação geral da mistura perfeita e instantânea é simplesmente uma média ponderada das concentrações com as respectivas vazões dos dois componentes que se misturam (Von Sperling, 2014b), representada pela equação (29).

$$C_o = \frac{Q_1 \,.\, C_1 + Q_2 \,.\, C_2}{Q_1 + Q_2}$$

Eq. 29

Na qual: C_o é a concentração de jusante ou da mistura (mg/L); C_1 é a concentração de montante (mg/L); Q_1 é a vazão de montante (m^3/s); C_2 é a concentração do lançamento (mg/L); Q_2 é a vazão do lançamento (m^3/s).

3.5. Análise de erros de primeira ordem ou análise de sensibilidade

As reações químicas e biológicas que ocorrem em um corpo d'água são representadas por um conjunto de equações que incorporam diversos coeficientes cinéticos. Esses coeficientes podem ser constantes, variar espacialmente ou depender da temperatura (Von Sperling, 2014b). Os valores dos coeficientes utilizados são incertos e sua relação com as previsões do modelo pode ser avaliada por meio da análise de sensibilidade. Para modelos com poucos parâmetros, a análise de sensibilidade tende a ser clara e direta. No entanto, em modelos mais complexos, a análise de sensibilidade pode se tornar difícil devido às diversas interações dinâmicas envolvidas.

O aplicativo QUALITOOL 2.0 oferece uma opção de calibração para os conjuntos de coeficientes dos modelos, totalizando até dezesseis coeficientes quando todos os parâmetros (OD, DBO, nitrogênio, fósforo e coliformes fecais) são selecionados. Embora esse número seja pequeno em comparação com outros modelos, como o SWAT, que pode ter mais de 100 coeficientes (Han; Zheng, 2016), a calibração desses coeficientes pode exigir um alto consumo de recursos computacionais e tempo. Dessa forma, torna-se relevante analisar a representatividade de cada coeficiente, avaliando sua influência nos resultados do modelo. Esse processo pode ser realizado por meio da análise de

sensibilidade, permitindo determinar a importância da precisão de cada coeficiente em relação ao desempenho geral do modelo.

A análise de sensibilidade no QUALITOOL 2.0 é baseada na análise de erros de primeira ordem (Von Sperling, 2014b). Esse método consiste em fixar um conjunto de valores, realizar a modelagem e verificar a variação percentual do parâmetro de qualidade da água em relação a alteração de cada coeficiente com base nos valores normalizados.

Para calibrar os coeficientes, o usuário define um intervalo para cada um, e a média desses intervalos formará o conjunto fixo de valores. Além disso, o usuário pode especificar um percentual de perturbação para cada coeficiente. Por exemplo, se o coeficiente k_2 tiver um intervalo médio de 40 dia^{-1} e o usuário optar por uma perturbação de 20%, o modelo será testado com os valores 40 dia^{-1} (médio), 32 dia^{-1} (-20%) e 48 dia^{-1} (+20%). Assim, é possível calcular a variação das variáveis de saída (a porcentagem da diferença da concentração do parâmetro) e a influência relativa de cada coeficiente na variância total do modelo.

Esse procedimento permite identificar quais coeficientes exercem maior influência na variação dos parâmetros de qualidade da água. Os coeficientes considerados sensíveis são aqueles cujas pequenas variações resultam em grandes impactos nas saídas do modelo. Por outro lado, coeficientes pouco sensíveis apresentam influência reduzida e podem ser fixados sem comprometer a precisão da previsão. Com base nessa análise, o usuário pode decidir quais coeficientes fixar e quais devem seguir para a próxima etapa de calibração.

3.6. Calibração com algoritmos evolucionários

O uso de algoritmos evolucionários tem se tornado cada vez mais comum na calibração de modelos hidrodinâmicos (Formiga *et al.*, 2016; Xue *et al.*, 2020) e de qualidade da água (Pelletier; Chapra; Tao, 2006; Khodabandeh *et al.*, 2021; Taghizadeh; Khani; Rajaee, 2021). O QUALITOOL 2.0 adota algoritmos evolucionários para calibrar os coeficientes cinéticos dos modelos de qualidade da água, utilizando especificamente o algoritmo de otimização por enxame de partículas (em inglês: *Particle Swarm Optimization* – PSO).

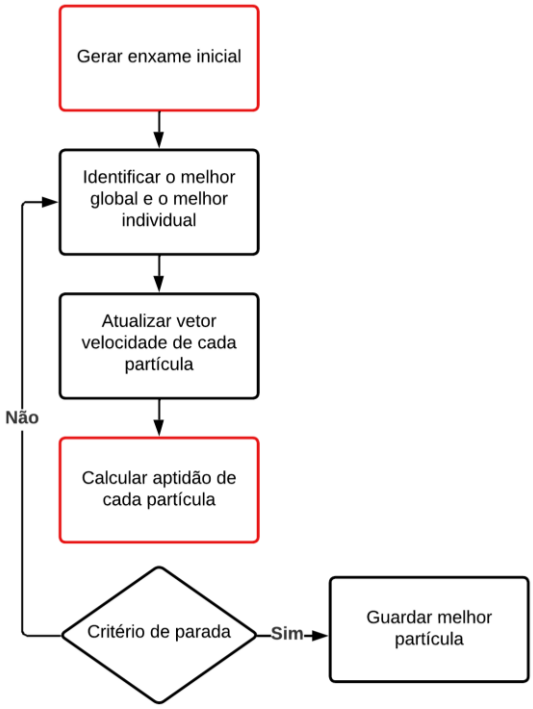
O PSO é um método de otimização proposto por Eberhart e Kennedy (1995), inspirado no comportamento coletivo de bandos de pássaros. Nesse sistema, diversas soluções candidatas, chamadas de "partículas", exploram simultaneamente o espaço de busca para encontrar a melhor solução. Cada partícula ajusta sua posição com base tanto em sua própria experiência quanto na experiência das partículas vizinhas. Durante esse processo, as melhores posições encontradas são armazenadas, permitindo que as partículas utilizem esse conhecimento para aprimorar suas trajetórias ao longo das iterações (Wang; Tan; Liu, 2017).

3.6.1. PSO no QUALITOOL 2.0

A Figura 6 apresenta o fluxograma do algoritmo PSO utilizado no QUALITOOL 2.0. O processo inicia-se com a geração de um enxame inicial de tamanho predefinido (i), composto por um conjunto de partículas. Cada partícula (x_i) é representada por um ponto cartesiano de dimensões equivalentes ao número de coeficientes a serem calibrados (D), sendo definida como: $x_i = [x_{i,1}, x_{i,2}, \dots, x_{i,D}]^T$. Os valores iniciais das partículas, incluindo sua posição (valores dos coeficientes) e velocidade, são atribuídos aleatoriamente, respeitando os limites mínimo e máximo de cada coeficiente cinético. Ainda nesta etapa, é calculada a aptidão de cada partícula, que corresponde a um valor indicando sua qualidade em relação ao problema de otimização. O objetivo do PSO é minimizar essa métrica, conforme detalhado no próximo tópico.

Destaca-se que as etapas de geração do enxame inicial e cálculo da aptidão estão realçadas em vermelho no fluxograma (Figura 6), pois utilizam um método de programação paralela. Essa abordagem otimiza o tempo de processamento dessas fases, permitindo que, em vez de calcular a aptidão de cada partícula de forma sequencial, todas as partículas do enxame sejam processadas simultaneamente, aumentando a eficiência computacional.

Figura 6. Fluxograma do algoritmo PSO.



Fonte: Autora (2025).

Após a geração do enxame, o algoritmo identifica a melhor partícula do grupo, tanto em nível individual quanto global. Em seguida, são realizadas atualizações iterativas no vetor de velocidade (Eq. 30) e na posição de cada partícula (Eq. 31).

$$v_{i,d,(t+1)} = w\, v_{i,d,(t)} + c_1 r_1 \left(p_{i,d,(t)} - x_{i,d,(t)}\right) + \; c_2 r_2 \left(p_{g,d,(t)} - x_{i,d,(t)}\right)$$

Eq. 30

$$x_{i,d,(t+1)} = x_{i,d,(t)} + v_{i,d,(t+1)}$$

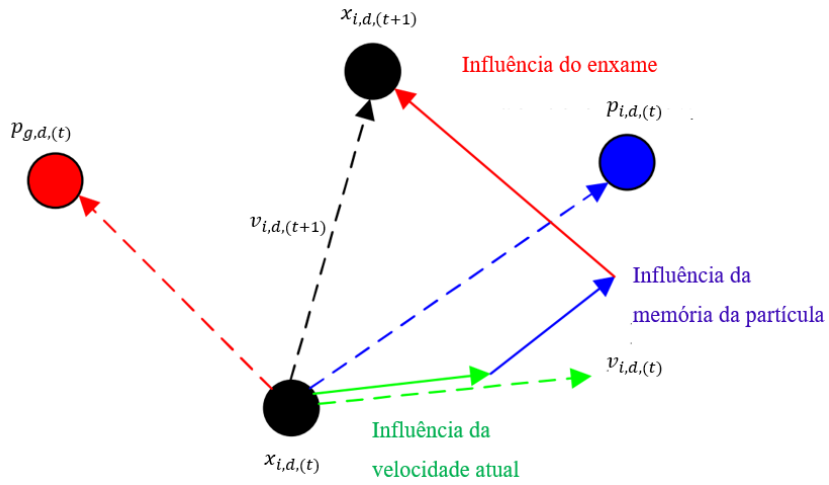
Eq. 31

Na qual: $v_{i,d,(t+1)}$ é a velocidade atualizada da partícula; $x_{i,d,(t+1)}$ é a posição atualizada da partícula; w é o coeficiente de inércia; $v_{i,d,(t)}$ é a velocidade atual da partícula; c_1 e c_2 são constantes de aceleração referentes ao melhor individual e global, respectivamente; r_1 e r_2 são números aleatórios entre 0 e 1; $p_{i,d,(t)}$ é a melhor posição individual conhecida da partícula; $x_{i,d,(t)}$ é a posição atual da partícula; $p_{g,d,(t)}$ é a melhor posição global conhecida no enxame.

A cada iteração, a aptidão de cada partícula é calculada para avaliar seu desempenho em relação ao problema de otimização. Esse processo é repetido até que um critério de parada seja atendido. O critério é considerado atendido se o valor da aptidão for repetido consecutivamente por vinte iterações ou quando o número máximo de iterações for alcançado — um valor que pode ser definido pelo usuário, com um padrão

de 15 iterações. Quando o critério de parada é atendido, a melhor partícula encontrada é registrada como a solução ótima. Caso contrário, novas iterações são realizadas para aprimorar os resultados. A Figura 7 ilustra o deslocamento de uma partícula, que parte da posição $x_{i,d,(t)}$ e se move para a nova posição $x_{i,d,(t+1)}$ na iteração subsequente.

Figura 7. Esquema de iteração das partículas no PSO.



Fonte: Adaptado de Wang *et al.* (2017).

3.6.2. Aptidão das partículas

Para avaliar a aptidão das partículas, utilizou-se o coeficiente de *Nash-Sutcliffe* (NS) de cada modelo selecionado. Esse coeficiente é uma medida de qualidade do modelo, onde o valor ideal é NS = 1,0, o que indica uma correspondência perfeita entre os valores observados e os valores previstos pelo modelo (Eq. 32). No entanto, como o problema é formulado como um problema de minimização, é necessário ajustar esse valor para que quanto menor for a diferença entre o modelo e os dados reais, melhor será o desempenho da partícula. Para isso, calcula-se o módulo da diminuição de 1 menos o valor de NS, transformando assim o problema para um formato de minimização. Quando há mais de um modelo (m) ao final do processo, realiza-se a média quadrática (RMS) dos resultados obtidos de cada modelo, oferecendo uma avaliação mais robusta e representativa do desempenho global das partículas e penalizando os valores mais altos da aptidão (Eq. 33).

$$\mathrm{NS}_m = 1 - \frac{\sum_{t=1}^{n}(R_{obs,m,t} - R_{calc,m,t})^2}{\sum_{t=1}^{n}(R_{obs,m,t} - R_{obs,m,médio})^2}$$

Eq. 32

$$RMS = \sqrt{\overline{|1 - NS_m|^2}}$$

Eq. 33

Na qual: NS_m é o coeficiente de *Nash-Sutcliffe* do modelo m; $R_{obs,m,t}$ é a concentração observada do modelo m; $R_{calc,m,t}$ é a concentração calculada do modelo m; $R_{obs,m,médio}$ é a média de todas as concentrações observadas do modelo m.

3.7. Desenvolvimento do QUALITOOL 2.0

Este subcapítulo não pretende abordar toda a programação desenvolvida no QUALITOOL 2.0, mas sim destacar seus pontos principais, mas o código-fonte se encontra nos Apêndices deste trabalho. O QUALITOOL 2.0 foi desenvolvido na linguagem de programação Python, utilizando a IDE (*Integrated Development Environment)* Visual Studio Code®, integrada com o GitHub® (plataforma de hospedagem de código-fonte), que interage diretamente com o Streamlit Hub® (hub central de gerenciamento de aplicativos Streamlit). Todas essas ferramentas possuem acesso gratuito em suas versões básicas.

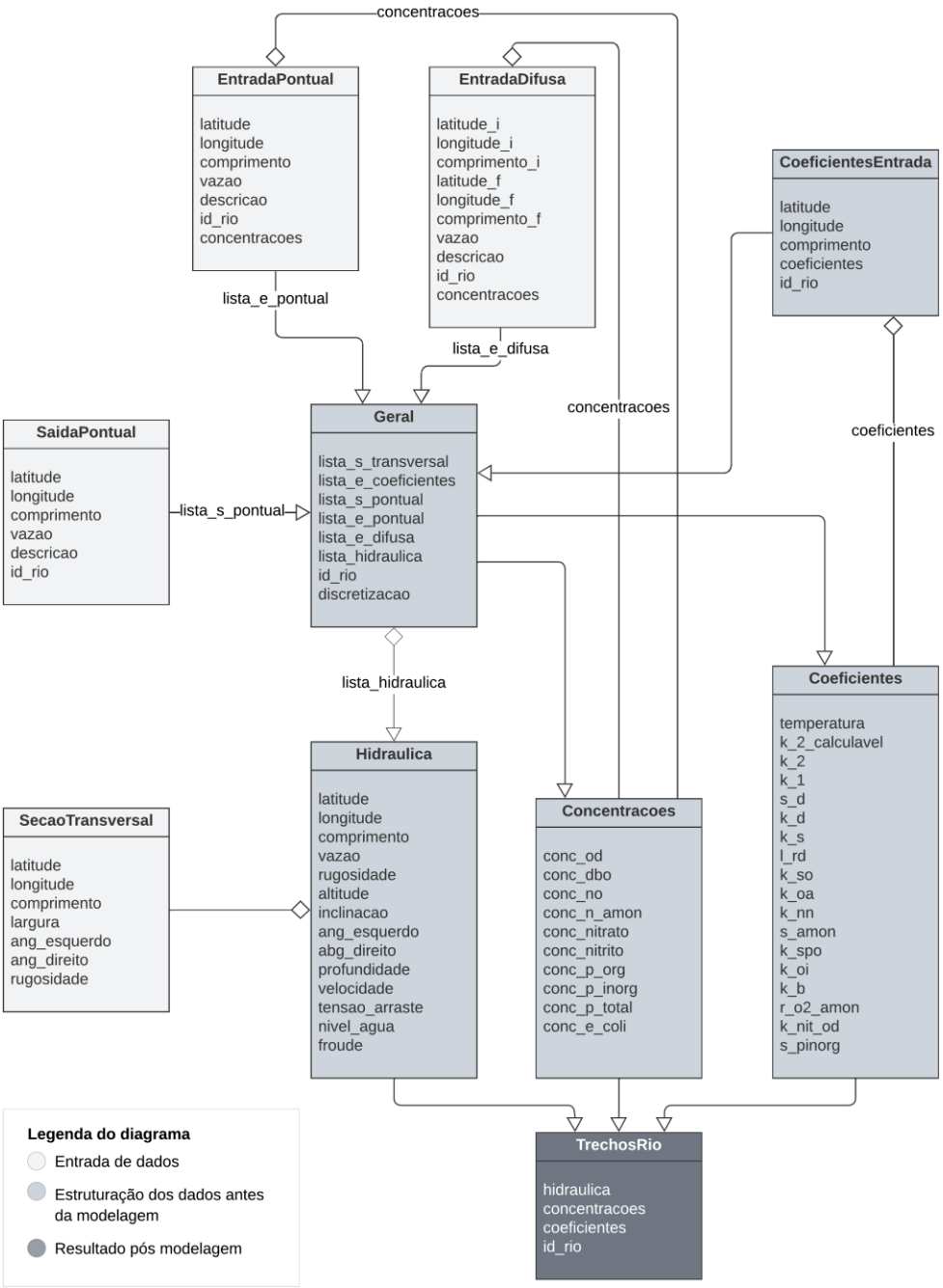
O QUALITOOL 2.0 foi concebido para ser altamente escalável, com cada macro de seção possuindo um arquivo distinto para: *main* (o arquivo principal que faz o aplicativo funcionar); introdução; instruções; layout da modelagem; layout da calibração; layout da aba gráficos; equações; sistema de otimização; resultados e transformações; e para os requerimentos. Além disso, dentro desses arquivos foram criadas funções que otimizam processos repetitivos. Essa estrutura facilitará futuras alterações no aplicativo QUALITOOL 2.0, potencializando seu crescimento como ferramenta.

3.7.1. Programação orientada à objeto

A programação orientada a objetos foi uma escolha estratégica para a construção do QUALITOOL 2.0, proporcionando modularidade e facilitando a manutenção e a escalabilidade do código. A Figura 8 apresenta o diagrama de classes, onde as caixas em cinza mais claro representam as estruturas dos dados de entrada fornecidos pelo usuário, e as de cinza mais escuro representam os dados já transformados e modelados. É possível visualizar a simplicidade de cada objeto/classe, tratados de forma independentes,

permitindo que sejam removidos ou novos itens sejam acrescentados sem afetar os já existentes.

Figura 8. Diagrama de classes do QUALITOOL 2.0.



Fonte: Autora.

Para a inclusão da análise de séries temporais no aplicativo, optou-se pela leitura dos dados de entrada como *arrays* (matrizes) para todos os itens dos objetos "Concentrações" e "Coeficientes". Quando a função de avaliação da série temporal é

ativada, a matriz será de Dx1, onde D representa a quantidade de datas a serem avaliadas. Caso contrário, a matriz será 1x1. Os demais objetos recebem dados do tipo lista, exceto os itens "descrição", "id_rio" e "discretização", que recebem valores do tipo *float* (valores com unidades decimais).

Os dados do objeto “Coeficientes” são diferentes para a modelagem com os coeficientes otimizados, sendo esses no formato lista com dois valores, de mínimo e máximo. Nesta modelagem também acrescenta um objeto, nomeado como “Partícula”, que receberá os itens: posição e a melhor posição individual, no formato lista; velocidade, aptidão atual e melhor aptidão individual, no formato *float*.

3.7.2. Recursos e requerimentos

Os recursos e requerimentos do QUALITOOL 2.0 envolvem diversas bibliotecas e ferramentas em Python, que são essenciais para o funcionamento eficiente e eficaz do aplicativo. A seguir, são detalhadas as principais bibliotecas utilizadas e suas respectivas funções:

- Streamlit: Responsável pelo gerenciamento do aplicativo no formato web, permitindo a criação de interfaces interativas e de fácil utilização para os usuários;
- Pandas: Utilizada para manipulação e análise de dados, oferecendo estruturas de dados rápidas, flexíveis e expressivas, que tornam o processamento de grandes volumes de dados mais eficiente;
- Numpy: Implementa operações matemáticas avançadas e manipulação de *arrays* multidimensionais, fundamentais para cálculos científicos e análise de dados complexos;
- Copy: Permite a realização de operações de cópia de objetos, essencial para a duplicação e gerenciamento eficiente de dados e estruturas internas do aplicativo;
- Random: Gera números aleatórios, necessários para implementar algoritmos de inteligência computacional que exigem variabilidade e simulações estocásticas;
- Scipy: Fornece ferramentas para otimização de soluções empíricas, permitindo a realização de cálculos científicos e técnicas avançadas de otimização;
- Plotly e Altair: Utilizada para a criação de gráficos interativos, facilitando a visualização e interpretação dos dados e resultados gerados pelo aplicativo.

- PIL (Pillow): Biblioteca para manipulação de imagens, permitindo a leitura, alteração e salvamento de diferentes formatos de imagem, integrando visualizações relevantes aos dados analisados;
- Thread Pool Executor, do Concurrent/Futures: Facilita a execução paralela de tarefas, otimizando o desempenho do aplicativo em operações que podem ser realizadas simultaneamente;
- Partial, do Functools: Permite a criação de funções parciais, simplificando a passagem de argumentos em funções, facilitando a chamada da função Thread Pool Executor.

3.7.3. Interação gráfica com Streamlit

Uma interface gráfica de usuário (*graphical user interface* - GUI) é uma interface digital que permite ao usuário interagir com elementos gráficos, como ícones, botões e menus. Em uma GUI, os visuais exibidos fornecem informações importantes ao usuário e indicam as ações disponíveis para ele realizar.

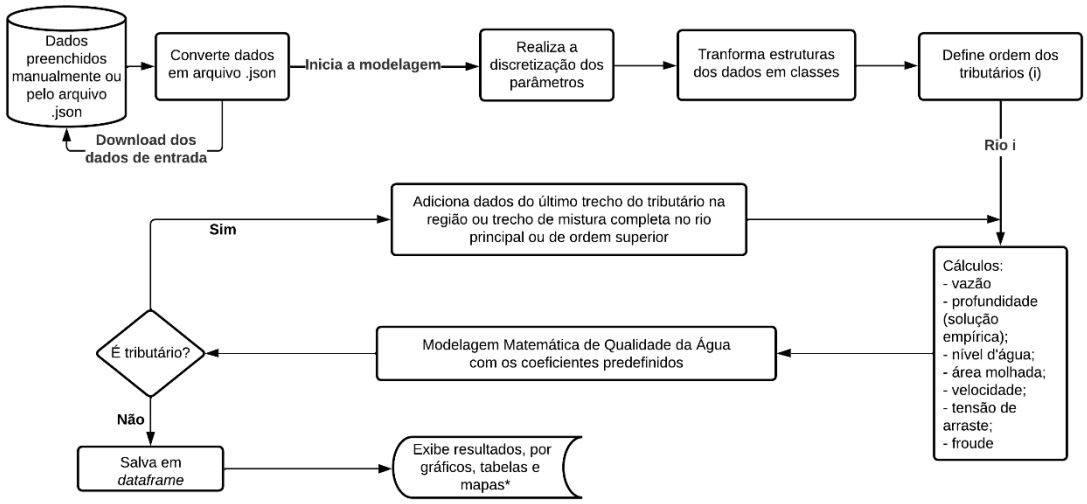
O Streamlit é uma biblioteca Python que facilita a criação de aplicações web interativas para visualização de dados, prototipagem rápida e construção de interfaces amigáveis (Lee *et al.*, 2022; Shi *et al.*, 2024). Com poucas linhas de código, ele transforma scripts Python em aplicativos web interativos, simplificando o desenvolvimento ao focar na lógica de negócios e na análise de dados, sem a necessidade de se preocupar com um *front-end* complexo. Além disso, o Streamlit permite compartilhar o aplicativo na nuvem gratuitamente por meio do Streamlit Cloud, que conecta sua conta do GitHub® e disponibiliza o código fonte do projeto na internet.

Neste trabalho, o Streamlit foi utilizado para construir a versão 2.0 do QUALITOOL no formato web. Dessa forma, qualquer pessoa, mesmo sem conhecimentos técnicos em programação ou inteligência artificial, pode usar o sistema de modelagem e calibração do aplicativo QUALITOOL 2.0 quando e como quiser, com uma interface intuitiva e de fácil entendimento.

3.8. Modelagem de qualidade da água com os coeficientes cinéticos predefinidos

A Figura 9 descreve um processo de modelagem matemática da qualidade da água em rios, utilizando coeficientes predefinidos no aplicativo QUALITOOL 2.0. O processo começa com a entrada de dados, que podem ser preenchidos manualmente ou importados via um arquivo .json. Esses dados são, então, convertidos e preparados para a modelagem, passando por etapas de discretização dos parâmetros geométricos e topológicos e organização em classes.

Figura 9. Fluxograma da etapa modelagem no QUALITOOL 2.0.



Fonte: Autora (2025).

A ordem dos tributários é definida para assegurar que o fluxo de informações ao longo do sistema fluvial seja processado corretamente. Inicialmente, são modelados os rios que não recebem tributários, seguindo depois a ordem de deságue. Por exemplo, se houver três tributários, nomeados como A, B e C, onde A deságua em C, e B e C deságuam no rio principal, a ordem de modelagem será A e B primeiro, depois C, e por fim, o rio principal. Isso garante uma modelagem precisa e coerente com a dinâmica dos rios no sistema.

Após a definição da ordem dos tributários, são realizados cálculos hidrológicos específicos, como vazão, profundidade, velocidade e outros parâmetros essenciais para avaliar a qualidade da água. Em seguida, a modelagem da qualidade da água é executada utilizando coeficientes predefinidos pelo usuário. Se o trecho analisado for um tributário, seus dados finais são integrados ao ponto de interseção com o rio de deságue (região de mistura). Esse processo é repetido para cada tributário, assegurando que todas as

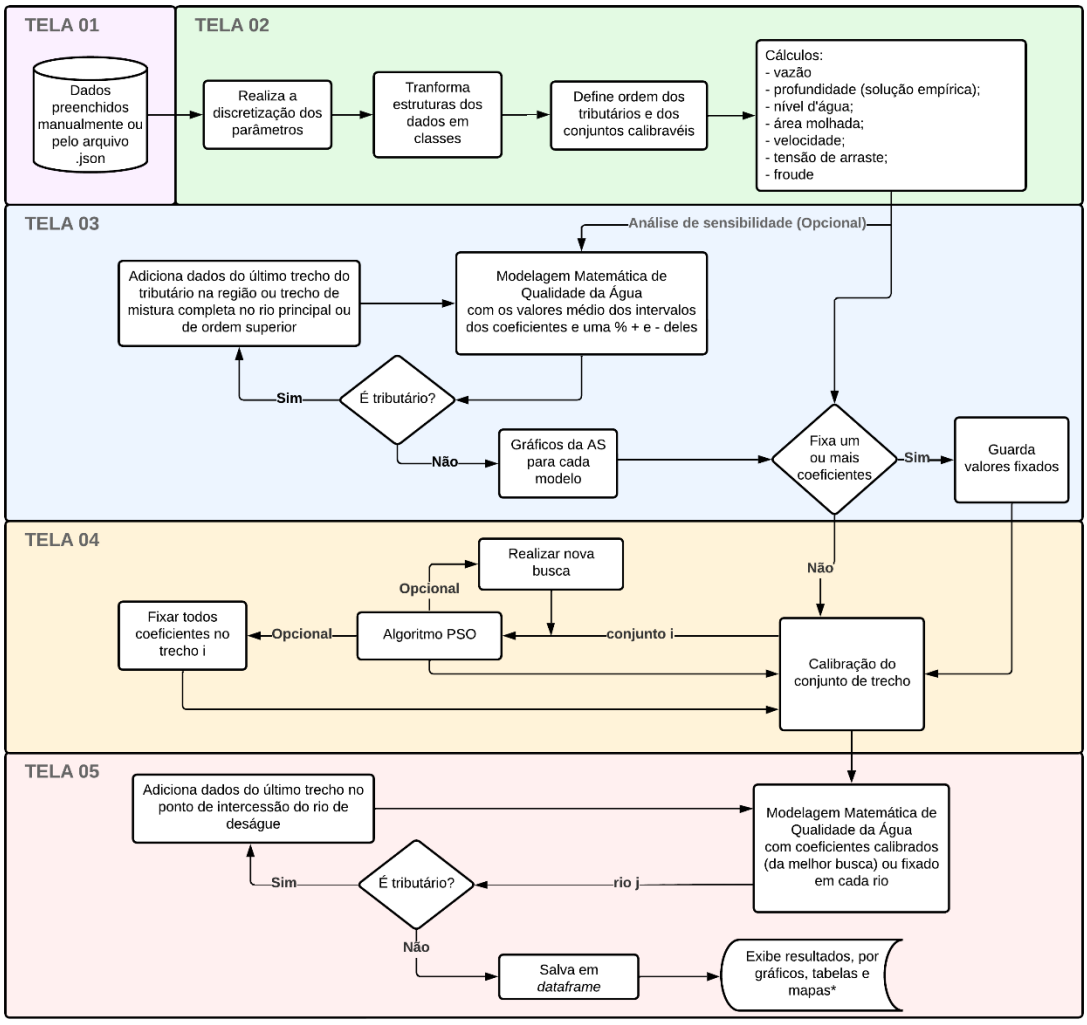
contribuições sejam corretamente incorporadas à modelagem. Por fim, a modelagem do rio principal é concluída, considerando todas as interações acumuladas ao longo do sistema fluvial.

Os resultados da modelagem são salvos em um *dataframe* (estrutura de dados bidimensional em forma de tabelas do pacote Pandas em Python), e, então, apresentados por meio de gráficos, tabelas e mapas, permitindo uma análise detalhada e visual dos dados processados. Todo o processo visa a garantir uma compreensão da qualidade da água ao longo dos diferentes trechos do rio.

3.9. Modelagem de qualidade da água com os coeficientes cinéticos otimizados

A Figura 10 apresenta o processo de ajuste e calibração dos coeficientes no aplicativo QUALITOOL 2.0, dividido em telas que ilustram cada etapa da configuração. O processo se inicia na Tela 01, onde são inseridos os dados dos parâmetros e os intervalos dos coeficientes. Diferentemente da modalidade “Modelagem”, essa etapa não armazena os dados preenchidos ao final. No entanto, como as etapas iniciais são similares, é possível carregar um arquivo salvo previamente na modalidade de modelagem.

Figura 10. Fluxograma da etapa ajuste e calibração no QUALITOOL 2.0.

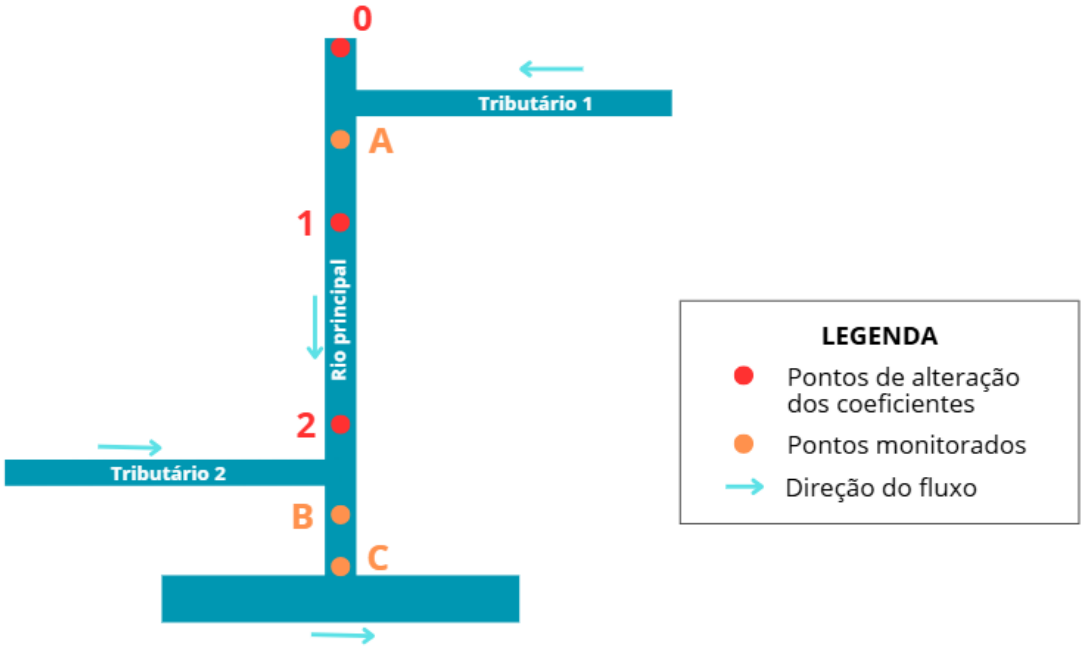


Fonte: Autora (2025).

A Tela 02 é uma etapa oculta do processo, na qual ocorre a discretização, a estruturação dos dados (convertendo-os em variáveis do tipo objeto) e a definição da ordem de modelagem. A sequência dos tributários segue o mesmo padrão da modalidade "Modelagem", mas com um diferencial: a formação dos conjuntos calibráveis.

Suponha a entrada de dados do sistema hídrico ilustrado na Figura 11, onde apenas o rio principal possui pontos de alteração dos valores dos coeficientes cinéticos (pontos 1 e 2). Nesse exemplo, são formados dois conjuntos de calibração. O primeiro abrange o trecho do ponto 0 ao ponto 1, que recebe os dados observados do ponto de monitoramento A. Como o Tributário 1 não possui pontos de alteração dos coeficientes, mas faz parte desse intervalo, ele será incluído nesse conjunto.

Figura 11. Esquema ilustrativo de um sistema hídrico fictício.



Fonte: Autora (2025).

Já entre o ponto 1 e o ponto 2, não será formado um conjunto, pois não há pontos de monitoramento nesse intervalo, impossibilitando a calibração. Dessa forma, o ponto 2 será descartado.

O segundo conjunto será formado entre o ponto 1 e o final do trecho do rio principal, considerando os dados observados dos pontos de monitoramento B e C. Assim como o Tributário 1, o Tributário 2 também não possui pontos de alteração dos coeficientes, e por isso será incluído nesse conjunto, pois está localizado dentro desse intervalo.

Ainda na Tela 02, são realizados os cálculos hidráulicos específicos, como vazão, profundidade e velocidade, além de outros parâmetros essenciais para avaliar a qualidade da água.

Seguindo para a Tela 03, ocorre a análise de sensibilidade dos coeficientes, etapa opcional do processo. Inicialmente, essa análise estará desativada para evitar execuções desnecessárias. Além disso, nesta etapa, o usuário tem a opção de fixar um ou mais coeficientes no modelo, caso deseje.

Na Tela 04, é realizada a calibração dos coeficientes utilizando o algoritmo PSO, sendo processado um conjunto calibrável por vez. Após cada calibração, o usuário pode

optar por reiniciar uma nova busca ou fixar todos os coeficientes daquele conjunto antes de prosseguir.

Por fim, na Tela 05, são apresentados os resultados da modelagem. Nessa etapa, a modelagem é reexecutada, agora utilizando os coeficientes calibrados nas etapas anteriores. Os resultados são salvos em um *dataframe* (estrutura de dados bidimensional do pacote Pandas em Python) e apresentados por meio de gráficos, tabelas e mapas, permitindo uma análise visual e detalhada dos dados processados.

4. RESULTADOS DA INTERFACE DO QUALITOOL 2.0

4.1. Apresentação do aplicativo QUALITOOL 2.0 / ABA "Apresentação"

O aplicativo QUALITOOL, versão 2.0 (logomarca na Figura 12), permite a simulação da qualidade de água em ambiente lótico, acoplada com um sistema de calibração com algoritmos evolucionários, opcional, para a otimização do modelo com base nos dados reais fornecido pelo usuário.

Figura 12. Logomarca do QUALITOOL 2.0.

Fonte: Autora (2025).

O objetivo principal no desenvolvimento deste aplicativo é auxiliar alunos de graduação e pós-graduação das diversas áreas de conhecimento no entendimento dos processos físicos, químicos e biológicos dos diversos parâmetros de qualidade de água em ambiente natural lótico, considerando entradas pontuais e difusas de cargas poluidoras e retiradas pontuais consuntivas e não consuntivas. Esta versão de QUALITOOL não considera a interação das águas superficiais com as subterrâneas.

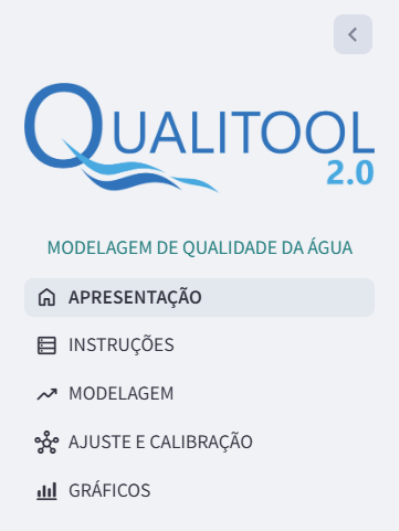
O usuário define quais parâmetros de qualidade da água pretende modelar a partir da necessidade particular da área de estudo. Dependendo do nível de precisão requerido para os processos envolvidos, este aplicativo de uso livre tem potencial para aplicação em trabalhos técnicos de consultoria na área ambiental.

O usuário poderá acessar a versão 2.0, o manual do usuário e o exemplo prático no link "*qualitool.streamlit.app*", e seu código de programação (desenvolvido em Python e com acesso livre ao público) pelo link "*github.com/QUALITOOL/qualitool_2_0*". Esses links e eventuais novidades, alterações e informações sobre o QUALITOOL serão geridas pelo Laboratório de Saneamento (LABSAN) da Universidade Federal de Uberlândia, através da página eletrônica: "*labsanufu.wixsite.com/site*".

A Figura 13 mostra o menu principal do QUALITOOL 2.0, fixada na aba "Apresentação", localizado na barra lateral (*sidebar*) do aplicativo. Este menu contém cinco abas que levam o usuário a diferentes telas, sendo elas:

i. Apresentação: fornece uma breve explicação do QUALITOOL 2.0, informações de contato da equipe gestora do projeto, além da identificação dos desenvolvedores e apoiadores;
ii. Instruções: disponibiliza exemplos resolvidos utilizando dados espaciais, geoespaciais e de séries temporais, além do manual do usuário e recomendações de uso;
iii. Modelagem: permite a modelagem de qualidade da água com coeficientes predefinidos;
iv. Ajuste e Calibração: oferece a modelagem da qualidade da água com coeficientes otimizados por algoritmos de inteligência computacional;
v. Gráficos: proporciona a visualização de gráficos e mapas geoespaciais.

Figura 13. Menu principal do aplicativo QUALITOOL 2.0.



Fonte: Autora (2025).

Essas abas são itens independentes, ou seja, não são etapas sequenciais. Apesar de ser possível usar a aba Modelagem para os dados salvos na aba Calibração. Também é possível plotar na aba Gráficos, resultados gerados nas abas Modelagem e Calibração, ou quaisquer outros dados no formato CSV.

4.1.1. Novidades da versão 2.0

A primeira versão do QUALITOOL, desenvolvida por Magalhães (2017), utiliza a linguagem de programação VBA (*Visual Basic for Applications*). Nessa versão, a entrada de dados e a apresentação dos resultados em gráficos e tabelas eram feitas diretamente na planilha de cálculo Excel®. Para otimizar a performance do aplicativo, a versão 2.0 do QUALITOOL inclui as seguintes melhorias:

- Desenvolvimento utilizando a linguagem de programação Python, que oferece mais recursos para programação orientada a objetos, facilitando a aplicação de algoritmos de aprendizado de máquina, a escalabilidade do aplicativo para futuras melhorias e o *deploy*, que consiste em disponibilizá-la na internet;
- Alterações na interface, agora em formato de site utilizando o Streamlit, integrado ao Python, que oferece uma gestão didática e amigável das informações, acessível em diferentes dispositivos, como computador, tablet e celular;
- Opção de modelar um ou mais parâmetros com um número ilimitado de afluentes, dependendo da performance da internet e do equipamento destinado ao processamento de dados do usuário;
- Facilidade na entrada de dados, que pode ser feita por meio de "copiar e colar" diretamente do Excel®, por intervalos predefinidos ou, no caso de dados espaciais, através de um arquivo GeoJson, criado por ferramentas de SIG (Sistema de Informação Geográfica);
- Recursos de salvamento e preenchimento automático de dados, permitindo que usuários que já preencheram os dados anteriormente possam reutilizá-los, com a possibilidade de alteração posterior dos valores de entrada, se desejado;
- Visualizações de dados espaciais através da base de mapa do Open Street Map®, disponibilizada gratuitamente pelos recursos do pacote Plotly, integrado ao Python;
- Visualização de gráficos dinâmicos, com os quais o usuário pode interagir e adaptar conforme suas necessidades, além da integração dos resultados com a visualização espacial;
- Simplificação e generalização dos códigos para aumentar a velocidade na entrega dos resultados, utilizando programação orientada a objetos;

- Correções nas equações de modelagem e adição do modelo em ambiente anaeróbico. Devido à simplicidade excessiva dos modelos de contaminantes arbitrários (sólidos suspensos, compostos tóxicos, entre outros) e de metais pesados no aplicativo, esses foram removidos na versão 2.0 para serem desenvolvidos em versões futuras;
- Opção de calibração dos coeficientes cinéticos dos modelos por meio de algoritmos de inteligência artificial.

4.2. ABA “INSTRUÇÕES”

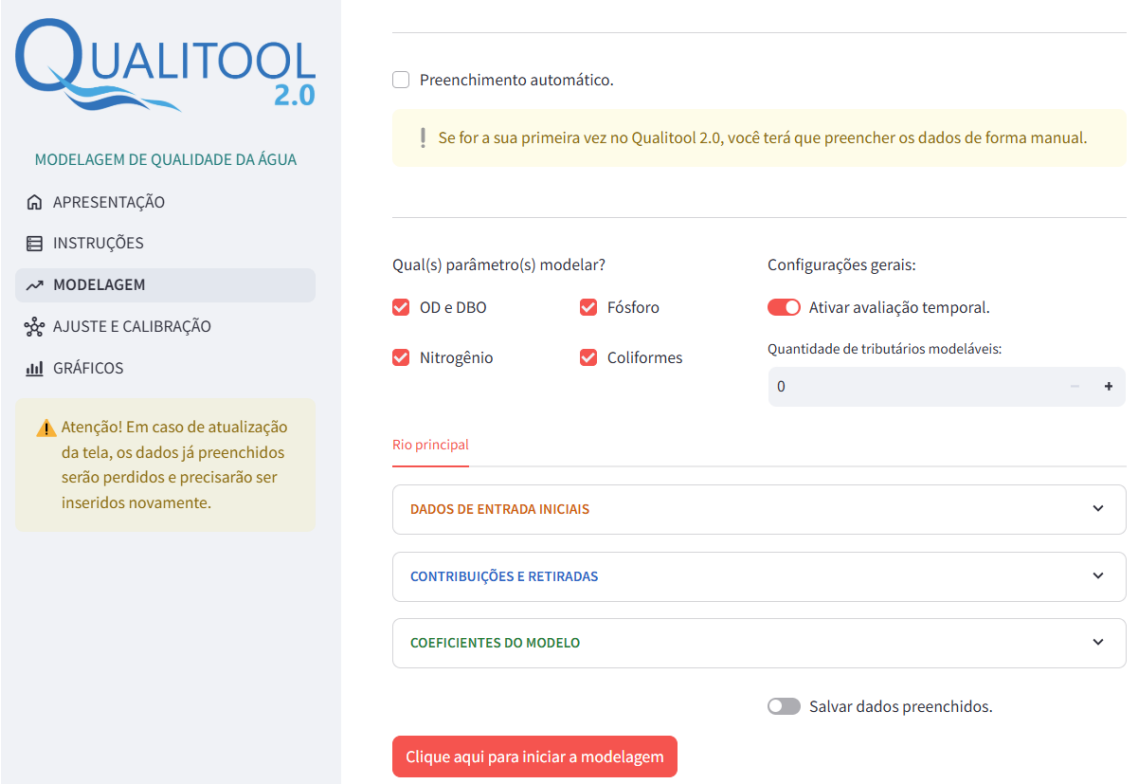
A aba "Instruções" do aplicativo QUALITOOL 2.0 tem como objetivo orientar os usuários na utilização adequada da plataforma, por meio de um manual detalhado e guias passo a passo para a resolução de exemplos práticos. Essas instruções auxiliam na compreensão das funcionalidades do sistema, tornando seu uso mais intuitivo e eficiente. Dessa forma, a plataforma se torna mais acessível a diferentes públicos, facilitando sua aplicação em contextos acadêmicos e profissionais e contribuindo para a otimização dos processos de análise e tomada de decisão.

Além das orientações sobre o uso do aplicativo, a aba também disponibiliza links para o código-fonte do QUALITOOL 2.0 e para o canal gestor do aplicativo, o site do LABSAN. Esses recursos permitem maior transparência no desenvolvimento do sistema e possibilitam a interação dos usuários com a equipe responsável pela plataforma. Dessa maneira, a aba "Instruções" não apenas facilita o uso do aplicativo, mas também incentiva a participação da comunidade na melhoria contínua da ferramenta.

4.3. ABA “MODELAGEM”

A Figura 14 ilustra a tela ativa na aba “Modelagem” do menu principal. Inicialmente, dois pontos importantes devem ser destacados. O primeiro é a mensagem no *sidebar*, com fundo amarelo claro, que alerta: “Atenção! Em caso de atualização da tela, os dados já preenchidos serão perdidos e precisarão ser inseridos novamente.” Esta mensagem tem o propósito de informar ao usuário que qualquer ação que resulte na atualização da página fará com que os dados preenchidos sejam perdidos.

Figura 14. Aba “Modelagem” no aplicativo QUALITOOL 2.0.



Fonte: Autora (2025).

O segundo ponto são as três barras de nomes coloridos na parte inferior da tela, presentes tanto na aba "Modelagem" quanto na aba "Calibração". Essas barras são:

i. “Dados de entradas iniciais”, em laranja: permite a inserção de dados espaciais (latitude, longitude, comprimento e altitude), valor do intervalo de discretização, valores iniciais do corpo d'água em análise (concentrações dos constituintes a serem modelados) e variáveis das seções transversais (rugosidade, largura de fundo do rio e ângulos dos taludes);
ii. “Contribuições e retiradas”, em azul: para preencher dados dos pontos de captação de água, descarga de poluição pontual e/ou difusa, se existentes;
iii. “Coeficientes do modelo”, em verde: permite o preenchimento dos valores dos coeficientes predefinidos.

Outro ponto muito importante a se destacar no QUALITOOL 2.0, é que a entrada de dados está em formato inglês, em que as entradas numéricas utilizam o ponto (.) como separador do decimal. Visto que esse formato é o mais prático para se trabalhar em programação. Entretanto, se o usuário copiar os dados direto de uma planilha Excel®, mesmo que os dados estejam utilizando a vírgula como separador do decimal, o

QUALITOOL 2.0 reconhecerá o formato e irá alterá-lo automaticamente para o formato inglês.

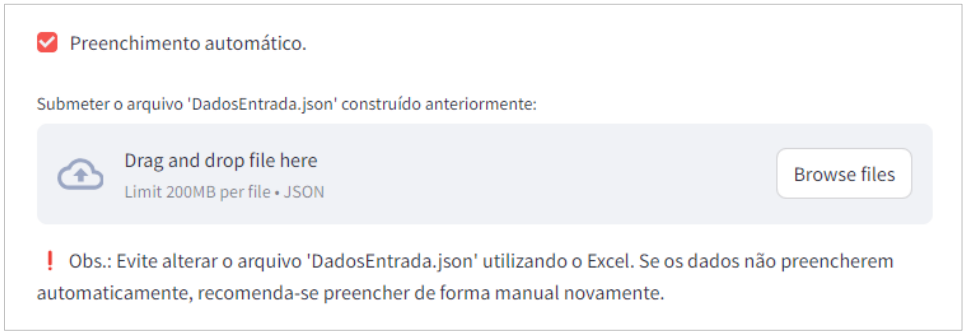
4.3.1. Entrada de dados

4.3.1.1.Preenchimento automático

A inserção de dados pode ser realizada de forma manual ou automática, sendo obrigatório o preenchimento manual na primeira utilização da plataforma ou a cada novo corpo d'água principal a ser modelado. Ao término desse processo inicial, será apresentada a opção de salvar os dados em um arquivo no formato JSON, nomeado como "DadosEntrada.json".

Caso o preenchimento já tenha sido realizado anteriormente e os dados tenham sido salvos em um arquivo JSON, é possível importar esse arquivo, o que permitirá o preenchimento automático com todas as informações previamente armazenadas (Figura 15). Se necessário, os dados preenchidos automaticamente poderão ser ajustados manualmente; entretanto, não será possível adicionar ou remover tributários, nem alterar os parâmetros do modelo.

Figura 15. Opção de preenchimento automático ativa.

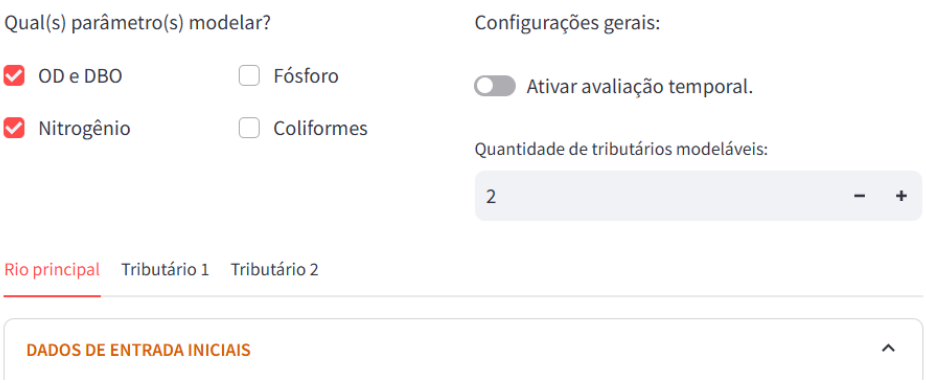


Fonte: Autora (2025).

4.3.1.2.Dados gerais

A modelagem dos parâmetros de OD, DBO, nitrogênio, fósforo e coliformes fecais é opcional (Figura 16). Caso nenhum desses parâmetros seja selecionado, apenas os dados hidráulicos serão considerados, utilizando a equação de Manning. Vale destacar que a modelagem de OD e DBO é interdependente, motivo pelo qual esses parâmetros são sempre tratados em conjunto.

Figura 16. Tela modelagem inicial.



Fonte: Autora (2025).

Nesta etapa, o usuário também pode definir a quantidade de tributários a serem modelados, sem um limite pré-estabelecido. No entanto, é importante considerar que um maior número de tributários aumenta o tempo de processamento.

Por fim, há a opção “Ativar avaliação temporal”, que permite a modelagem considerando diferentes datas para as concentrações iniciais, possibilitando uma análise mais dinâmica e abrangente do sistema.

4.3.1.3.Dados de entrada iniciais

a. Ponto de deságue dos tributários

A estrutura de entrada de dados para o rio principal e seus tributários modeláveis é idêntica para cada rio, com uma única exceção: os tributários precisam informar o ponto onde deságuam no rio principal. Como exemplo, na Figura 17 o Tributário 1 deságua no ponto 7899201 (latitude – y, em UTM) e 786019 (longitude – x, em UTM).

Figura 17. Ponto de deságue.

Fonte: Autora (2025).

Além disso, há a opção de inserir um ID (identificação) para cada tributário. Esse campo aceita apenas valores numéricos e é opcional. Caso um ID seja fornecido, ele será exibido nos gráficos finais, facilitando a identificação e análise dos dados.

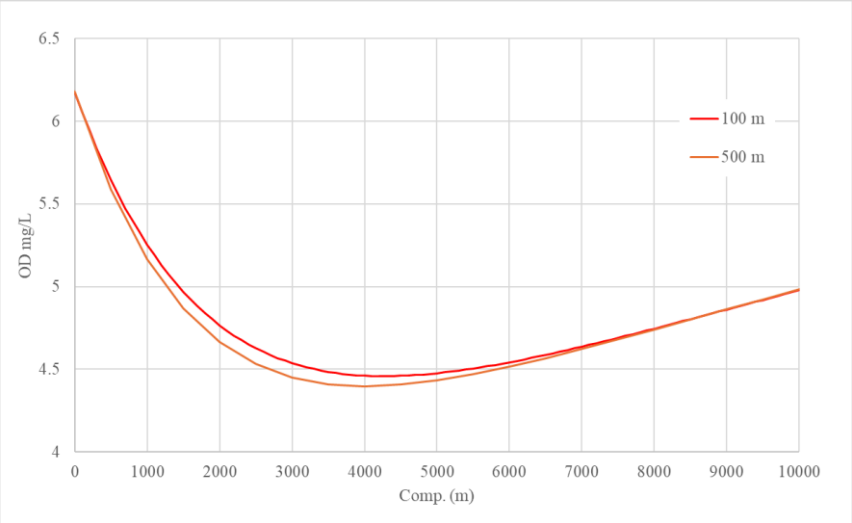
b. Discretização

O usuário deve informar o valor da discretização em metros, que definirá o intervalo a ser segmentado em cada trecho do rio. Como detalhado no subcapítulo 3.1.1, a escolha desse valor pode influenciar a precisão dos resultados.

Atualmente, o QUALITOOL 2.0 processa a discretização da seguinte forma: se o rio tiver um comprimento total de 10.000 metros e a discretização for de 12 metros, o programa considerará apenas 834 intervalos de 12 metros. Com isso, o comprimento total modelado será de 10.008 metros, e os últimos 8 metros adotarão os mesmos dados (altitude e informações geométricas) do último ponto conhecido, ou seja, o ponto localizado a 10.000 metros.

A Figura 18 ilustra os resultados de perfil de OD obtidos no QUALITOOL 2.0 utilizando discretizações de 100 e 500 metros, com base nos dados do exemplo apresentado no capítulo 13 do livro *Estudos e Modelagem da Qualidade da Água de Rios*, de Von Sperling (2014b).

Figura 18. Efeito da discretização na concentração de OD.



Fonte: Autora (2025).

c. Dados espaciais e geoespaciais

Para entrada dos dados da topologia do sistema hídrico existem três opções disponíveis, nomeadas de: Manual, Intervalo e GeoJson.

A opção Manual (Figura 19) permite inserir dados de latitude e longitude, em UTM (WGS 84), e altitude e comprimento, em metros. Neste modo, pode-se optar por inserir latitude e longitude ou o comprimento, não sendo obrigatória a entrada de dados geoespaciais. Essa opção também possui uma tabela interativa, que foi adicionada a fim de facilitar a entrada dos dados, sendo possível "copiar e colar" os dados direto de uma planilha de Excel®.

Figura 19. Opção de entrada de dados espaciais, "Manual" ativo.

Fonte: Autora (2025).

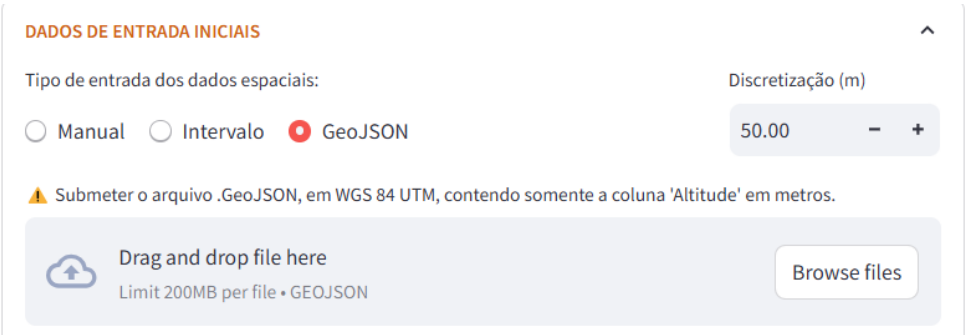
A opção Intervalo (Figura 20) permite inserir dados simplificados e não georreferenciados. Deve-se informar: o comprimento total do rio a ser modelado, em metros; a declividade longitudinal (inclinação), em metros por metros; e a altitude, em metros.

Figura 20. Opção de entrada de dados espaciais, “Intervalo” ativo.

Fonte: Autora (2025).

Também é possível importar um arquivo do tipo GeoJSON (Figura 21) que preenche automaticamente os dados espaciais. Esse arquivo pode ser obtido em ferramentas SIG, como no QGIS® e ArcGIS®. Os dados devem estar em UTM (*Universal Transversa de Mercator*, em metros), e se o usuário quiser utilizar a visualização em mapas, deverá estar no sistema WGS 84. É obrigatório ter dentro desse arquivo as informações de altitude, em metros.

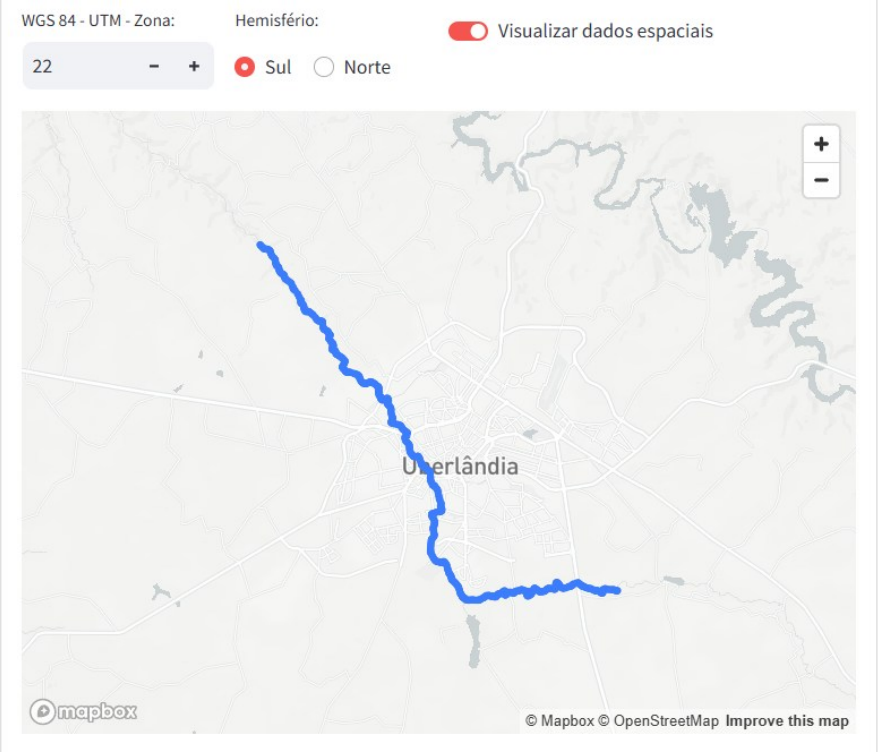
Figura 21. Opção de entrada de dados espaciais, “GeoJSON” ativo.



Fonte: Autora (2025).

Se a opção escolhida for a Manual ou GeoJson e os dados forem geoespaciais em WGS 84 – UTM, será possível visualizar o mapa. A Figura 22 mostra um exemplo dessa visualização para dados coletados, na ferramenta SIG, de um trecho do rio Uberabinha, localizado no município de Uberlândia/MG. Esse *layout* de mapa é fornecido de forma gratuita pela Open Street Map®.

Figura 22. Opção de visualização de mapa para dados georreferenciados.



Fonte: Autora (2025).

d. Dados temporais

Na Figura 16, foi possível visualizar a opção de ativar a avaliação temporal. Essa opção permite ao usuário inserir dados temporais, com a escala mínima permitida sendo diária. Os valores de data deverão estar no formato inglês, sendo esse: mês/dia/ano. O aplicativo QUALITOOL 2.0 está programado para inserir e modificar esses valores somente na aba "Dados de entradas iniciais" do Rio principal (Figura 23), nas outras abas, esses valores de entradas serão visíveis, porém estarão bloqueados, não permitindo alterações das datas.

Figura 23. Entrada de dados temporais.

Fonte: Autora (2025).

e. Seções transversais

O aplicativo QUALITOOL 2.0 também permite ao usuário adicionar as alterações dos valores nas seções transversais (Figura 24) ao longo do rio. Sempre que esse recurso é requerido pelo usuário, é informado na tela colunas com o nome “Ponto X”, sendo x o número do ponto. Esse nome da coluna não estará associado ao ponto, ele serve para o usuário se orientar. Por exemplo, se na seção transversal o usuário fez a identificação de cinco pontos ao longo do rio, irá apresentar cinco colunas de Ponto X. Se então o usuário também, na aba coeficientes, alterar os coeficientes quatro vezes ao longo do rio, irá aparecer nessa aba quatro colunas de Ponto X. O Ponto X da aba seção transversal e o Ponto X da aba coeficientes não estarão correlacionados. A Figura 24 apresenta a inserção desses pontos de alteração em um exemplo hipotético.

Figura 24. Seções transversais.

	0. Descrição	Ponto 0	Ponto 1	Ponto 2	Ponto 3	Ponto 4
0	Latitude (UTM)	7,896,781	7,899,201	7,909,654	7,910,480	7,913,708
1	Longitude (UTM)	787,148	786,019	782,175	781,048	778,636
2	Comprimento (m)	None	None	None	None	None
3	Rugosidade (manning)	0.042	0.042	0.042	0.042	0.042
4	Largura (m)	14	16	16.5	17.5	18
5	Ângulo esquerdo (°)	48	46	46	48	45
6	Ângulo direito (°)	45	47	49	45	47

Fonte: Autora (2025).

A Tabela 4 apresenta os valores de referência para os coeficientes de rugosidade de Manning em canais naturais e artificiais, servindo como um guia para o usuário. Esses coeficientes variam de acordo com as características do leito do rio ou canal, como material da superfície, presença de vegetação e irregularidades do terreno. Além dos valores apresentados, a literatura técnica oferece diversas outras referências sobre o tema, abrangendo estudos específicos para diferentes tipos de cursos d'água e condições ambientais. A escolha do coeficiente de Manning influenciará diretamente a estimativa de velocidades, vazões e níveis d’água ao longo do sistema modelado.

Tabela 4. Coeficientes de rugosidade de Manning.

Natureza das paredes	Condições			
	Muito boa	Boa	Regular	Má
Alvenaria de pedra argamassada	0,017	0,020	0,025	0,030
Alvenaria de pedra aparelhada	0,013	0,014	0,015	0,017
Alvenaria de pedra seca	0,025	0,033	0,033	0,035
Alvenaria de tijolos	0,012	0,013	0,015	0,017
Calhas metálicas lisas (semicirculares)	0,011	0,012	0,013	0,015
Canais abertos em rocha (irregular)	0,035	0,040	0,045	-
Canais com fundo em terra e talude com pedras	0,028	0,030	0,033	0,035
Canais com leito pedregulhoso e talude vegetado	0,025	0,030	0,035	0,040
Canais com revestimento de concreto	0,012	0,014	0,016	0,018
Canais de terra (retilíneos e uniformes)	0,017	0,020	0,023	0,025
Canais dragados	0,025	0,028	0,030	0,033
Condutos de barro (drenagem)	0,011	0,012	0,014	0,017
Condutos de barro vitrificado (esgoto)	0,011	0,013	0,015	0,017
Condutos de prancha de madeira aplainada	0,010	0,012	0,013	0,017
Gabião	0,022	0,030	0,035	-
Superfícies de argamassa de cimento	0,011	0,012	0,013	0,050
Superfícies de cimento alisado	0,010	0,011	0,012	0,030
Córregos e rios limpos, retilíneos e uniformes	0,025	0,028	0,030	0,033
Córregos e rios limpos, retilíneos e uniformes com pedras e vegetação	0,030	0,033	0,035	0,040
Córregos com meandros, bancos e poços, limpos	0,035	0,040	0,045	0,050
Margens espraiadas, pouca vegetação	0,050	0,060	0,070	0,080
Margens espraiadas, muita vegetação	0,075	0,100	0,125	0,150

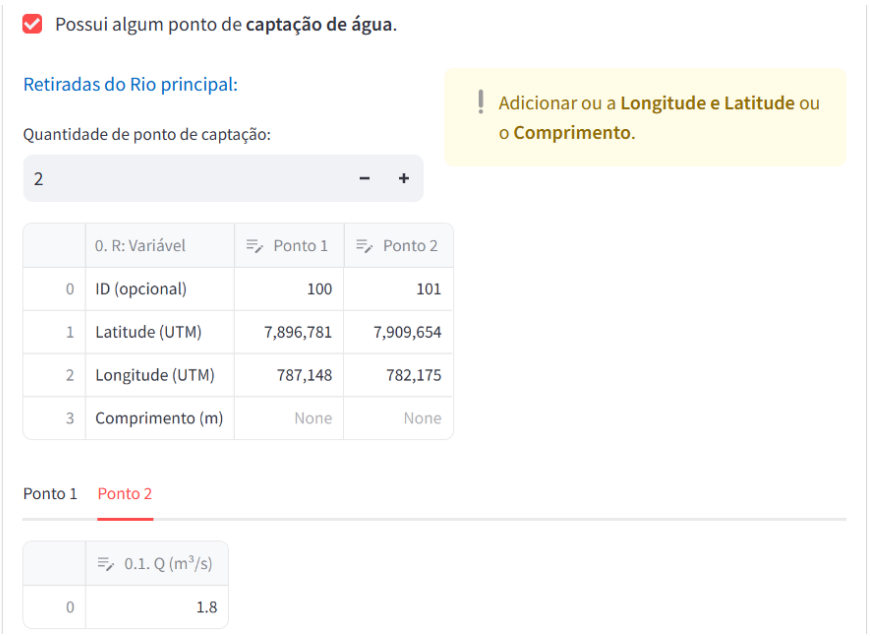
Fonte: Adaptado de Porto (1998).

4.3.1.4.Contribuições e retiradas

Na aba “Contribuições e Retiradas”, o usuário tem a opção de adicionar, caso existam, os pontos de lançamento pontual e difuso, além das retiradas de água no curso d’água selecionado.

A retirada pontual refere-se aos pontos de captação de água para diferentes finalidades, como irrigação, dessedentação animal e abastecimento público. Para inserir os pontos de captação, o usuário deve ativar a opção “Possui algum ponto de captação de água” (Figura 25) e inserir as coordenadas geográficas ou o comprimento (em relação ao ponto inicial do rio em questão) dos pontos, além da vazão retirada (sem o símbolo negativo).

Figura 25. Dados de pontos de captação de água.

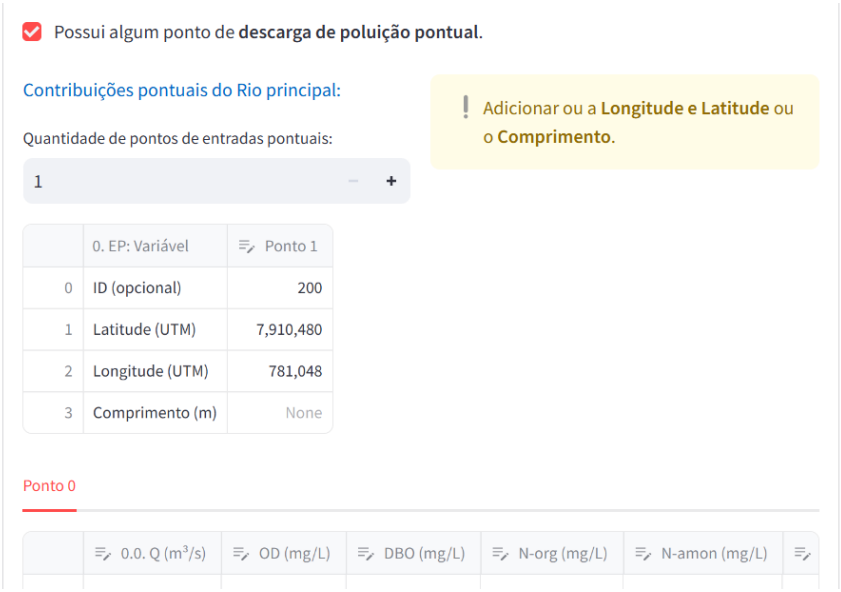


Fonte: Autora (2025).

O lançamento pontual refere-se à descarga de poluição em pontos específicos, como entradas de afluentes/tributários não modeláveis, despejos de efluentes sanitários e outras fontes localizadas. Embora os tributários modeláveis também sejam considerados entradas pontuais, eles não fazem parte desta etapa, pois serão modelados separadamente. Após a modelagem deles, o próprio aplicativo incluirá seus resultados como entradas pontuais.

Para adicionar um lançamento pontual, é necessário ativar a opção “Possui algum ponto de descarga de poluição pontual” (Figura 26) e informar as coordenadas geográficas ou o comprimento (em relação ao ponto inicial do rio em questão), bem como a vazão e as concentrações dos parâmetros.

Figura 26. Dados de pontos de contribuição pontual.

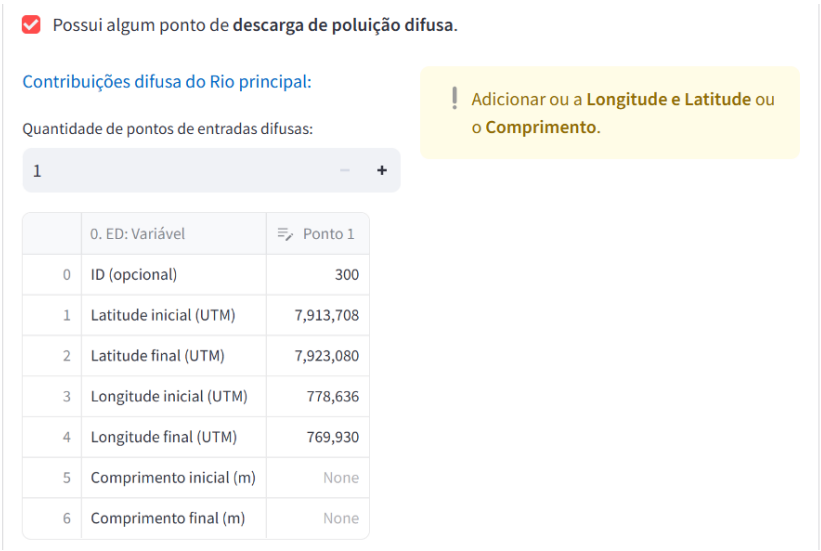


Fonte: Autora (2025).

O lançamento difuso, por sua vez, corresponde a descargas de poluição distribuídas ao longo de um trecho do rio, como a entrada de poluentes químicos através do escoamento superficial em áreas de plantio.

Já para os lançamentos difusos, deve-se ativar a opção “Possui algum ponto de descarga de poluição difusa” (Figura 27) e inserir as coordenadas geográficas ou o comprimento (em relação ao ponto inicial do rio em questão) do ponto inicial e final do trecho afetado, além da vazão total ao longo do trecho e das concentrações dos parâmetros.

Figura 27. Dados de pontos de contribuição difusa.

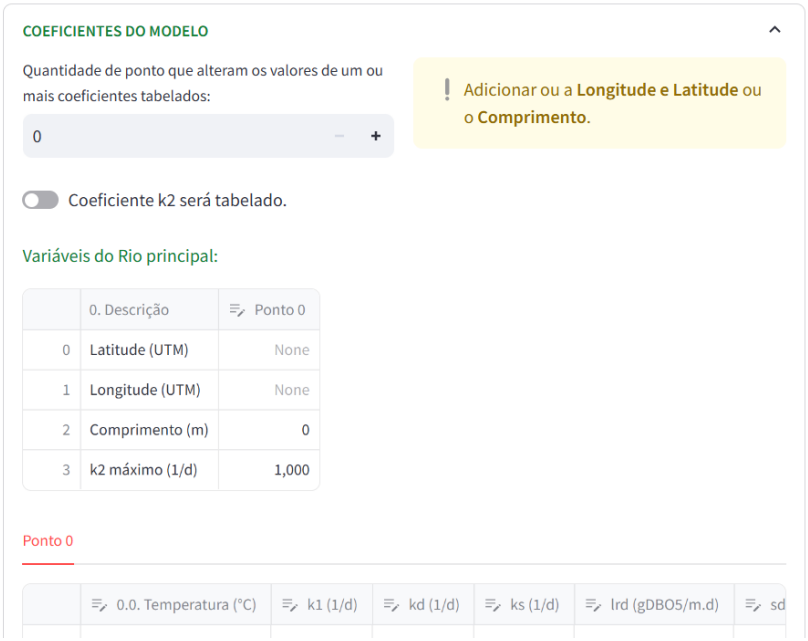


Fonte: Autora (2025).

4.3.1.5.Coeficientes do modelo

Na aba “Coeficientes do Modelo” (Figura 28), é necessário informar os valores dos coeficientes cinéticos no ponto 0, que representa o início do trecho. No entanto, o usuário também pode adicionar outros pontos ao longo do trecho onde ocorra a alteração de um ou mais coeficientes.

Figura 28. Dados de pontos que alteram os coeficientes.



Fonte: Autora (2025).

Conforme detalhado no subcapítulo 3.3.4, o coeficiente k2 pode ser determinado por equações hidráulicas ou por valores tabelados. Caso a opção tabelada seja escolhida, o usuário deve ativar “Coeficiente k2 será tabelado”. Se essa opção for ativada, também é necessário informar o valor máximo permitido para o coeficiente.

Além disso, as Tabelas 5 a 9 apresentam valores desses coeficientes disponíveis na literatura para a temperatura de 20°C. O usuário deve informar os coeficientes nessa temperatura, pois o próprio modelo realiza a conversão para a temperatura do rio.

Tabela 5. Valores típicos dos coeficientes de remoção de DBO (base e 20ºC).

Origem	K_1 (laborat.)	Rios rasos			Rios profundos		
		Decomp. K_d	Sediment. K_s	Remoção K_r ($=K_s+K_d$)	Decomp. K_d	Sediment. K_s	Remoção K_r ($=K_s+K_d$)
Curso d'água recebendo esgoto bruto concentrado	0,35 – 0,45	0,50 – 1,00	0,10 – 0,35	0,60 – 1,35	0,35 – 0,50	0,05 – 0,20	0,40 – 0,70
Curso d'água recebendo esgoto bruto de baixa concentração	0,30 – 0,40	0,40 – 0,80	0,05 – 0,25	0,45 – 1,05	0,30 – 0,45	0,00 – 0,15	0,30 – 0,60
Curso d'água recebendo efluente primário	0,30 – 0,40	0,40 – 0,80	0,05 – 0,10	0,45 – 0,90	0,30 – 0,45	0,00 – 0,05	0,30 – 0,50
Curso d'água recebendo efluente secundário	0,12 – 0,24	0,12 – 0,24	-	0,12 – 0,24	0,12 – 0,24	-	0,12 – 0,24
Curso d'água com águas limpas	0,08 – 0,20	0,08 – 0,20	-	0,08 – 0,20	0,08 – 0,20	-	0,08 – 0,20

Nota: rios rasos: profundidade inferior a cerca de 1,0 ou 1,5m; rios profundos: profundidade superior a cerca de 1,0 ou 1,5m

Fonte: Von Sperling (2014b).

Tabela 6. Valores do coeficiente de demanda de oxigênio pelo sedimento Sd' para diferentes tipos de fundo de rio (20ºC).

Tipo de fundo e localização	Faixa de valores (g/m².d)	Valor médio (g/m².d)
Lodo oriundo de esgoto municipais, próximos ao emissário	2 a 10	4
Lodo oriundo de esgoto municipais, a jusante do emissário	1 a 2	1,5
Lodo estuarino	1 a 2	1,5
Fundo arenoso	0,2 a 1,0	0,5
Solos Minerais	0,05 a 0,1	0,07

Fonte: Thomann (1972, apud VON SPERLING, 2014b).

Tabela 7. Valores de Referência da demanda de oxigênio pelo sedimento Sd' para diferentes condições de fluxo do rio (20ºC).

Condições de Fluxo	Características das águas		
	Natural	Moderadamente poluída	Fortemente poluída
Remanso	0,50	0,70	1,00
Lento	0,20	0,50	1,00
Rápido	0,00	0,20	0,50
Corredeira	0,00	0,00	0,00

Fonte: Aguirre (2000, apud VON SPERLING, 2014b).

Tabela 8. Valores de referência para a modelagem do nitrogênio (20°C).

Símbolo	Descrição	Unidade	Valores usuais do Coeficiente (20°C)	Coeficiente de temperatura θ (adimensional)
Kso	Coef. de sedimentação do N orgânico	d^{-1}	Remansos: 0,10	1,024
			Rios lentos com águas naturais a moderadamente poluídas: 0,05	
			Rios lentos com águas fortemente poluídas: 0,10	
			Rios rápidos com águas naturais: 0,020	
			Rios rápidos com águas moderadamente poluídas: 0,050	
			Rios Rápidos com águas fortemente poluídas: 0,10	
			Corredeiras: 0,00	
Koa	Coef. de conversão do N orgânico a amônia	d^{-1}	0,20 a 0,25	1,047
Kan	Coef. de conversão da amônia a nitrito	d^{-1}	0,15 a 0,25	1,080
Knn	Coef. de conversão de nitrito a nitrato	d^{-1}	Cursos d´água profundos: 0,10 a 0,50	1,047
			Cursos d'água rasos: 0,20 a 1,00	
Samon	Fluxo de liberação de amônia pelo sedimento de fundo	$g/m^2.d$	0,00 a 0,50 (menores valores para rios limpos e rápidos)	1,074
fnitr	Fator de correção do coef. de nitritficação em função do OD	-	$fnitr = 1 - e^{-KnitrOD.OD}$	-
KnitrOD	Coef. de inibição da nitrificação por baixo OD	L/mg	0,60	-
RO_2Amon	Relação entre o oxigênio consumido por cada unidade de amônia oxidada a nitrito	mgO_2/mgN_{amon}	3,20	-
RO_2nitr	Relação entre o oxigênio consumido por cada unidade de nitrito oxidado a nitrato	mgO_2/mgN_{nitri}	1,10	-

Fonte: Von Sperling (2014b).

Tabela 9. Valores de referência da modelagem do Fósforo (20ºC).

Símbolo	Descrição	Unidade	Valores Intermediários do Coeficiente	Coeficiente de temperatura θ (adimensional)
K_{spo}	Coef. de sedimentação do P orgânico	d^{-1}	0,02 a 0,05	1,024
K_{oi}	Coef. de conversão do P orgânico a P inorgânico	d^{-1}	0,2 a 0,3	1,047

Fonte: Von Sperling (2014b).

4.3.2. Resultados

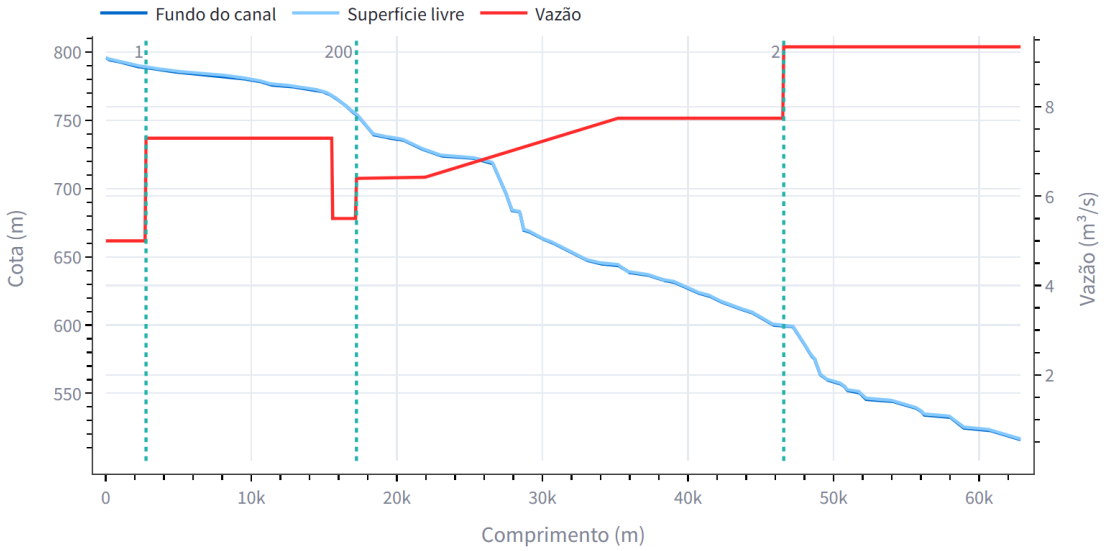
Após preencher os dados e iniciar a modelagem clicando no botão “Clique aqui para iniciar a modelagem”, os resultados serão exibidos no formato de gráficos, tabelas e mapas, caso os dados sejam georreferenciados.

Os gráficos apresentados referem-se exclusivamente ao rio principal, uma vez que a quantidade de tributários é indefinida. E, se a opção de série temporal estiver ativada, será exibida apenas a primeira data. No entanto, há maneiras de visualizar os gráficos dos tributários, conforme será detalhado no próximo subitem.

A Figura 29 ilustra um exemplo de resultados hidrodinâmicos, enquanto a Figura 30 apresenta os resultados de OD. Na Figura 31, o OD é exibido juntamente com a DBO, e a Figura 32 mostra os resultados de nitrogênio e suas frações. Os gráficos disponíveis incluem tanto as concentrações de todos os parâmetros selecionados para a modelagem quanto os dados hidráulicos, como vazão, nível de água e nível do fundo, ambos apresentados em função do comprimento do trecho modelado, em metros.

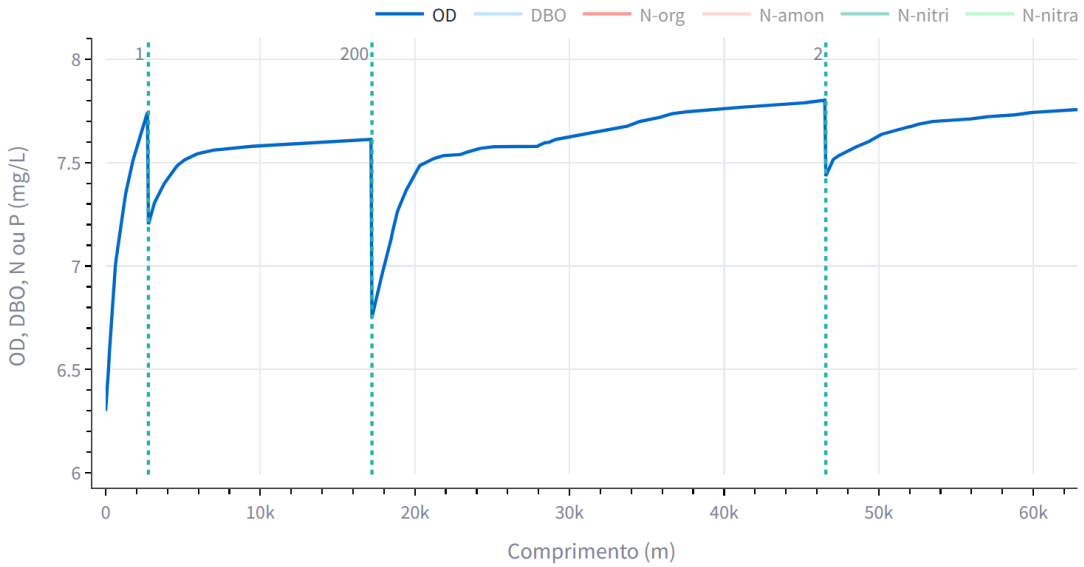
Todos os gráficos são interativos e possuem uma legenda padrão localizada na parte superior. O usuário pode habilitar ou desabilitar qualquer item da legenda com um clique, permitindo uma análise mais dinâmica e personalizada dos resultados.

Figura 29. Resultados hidrodinâmicos para o Rio principal.



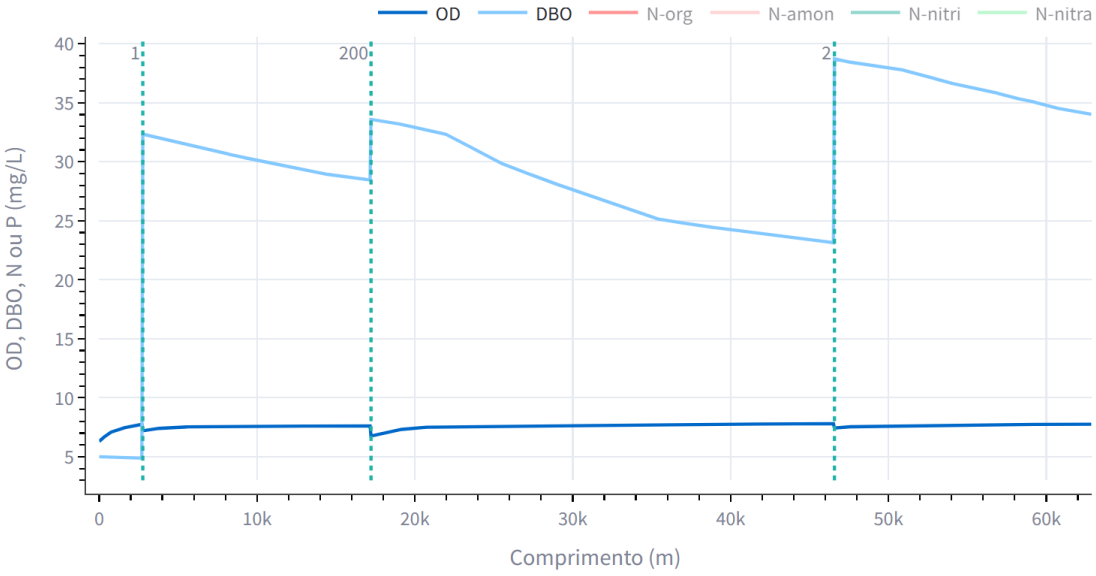
Fonte: Autora (2025).

Figura 30. Resultados de OD para o Rio principal.



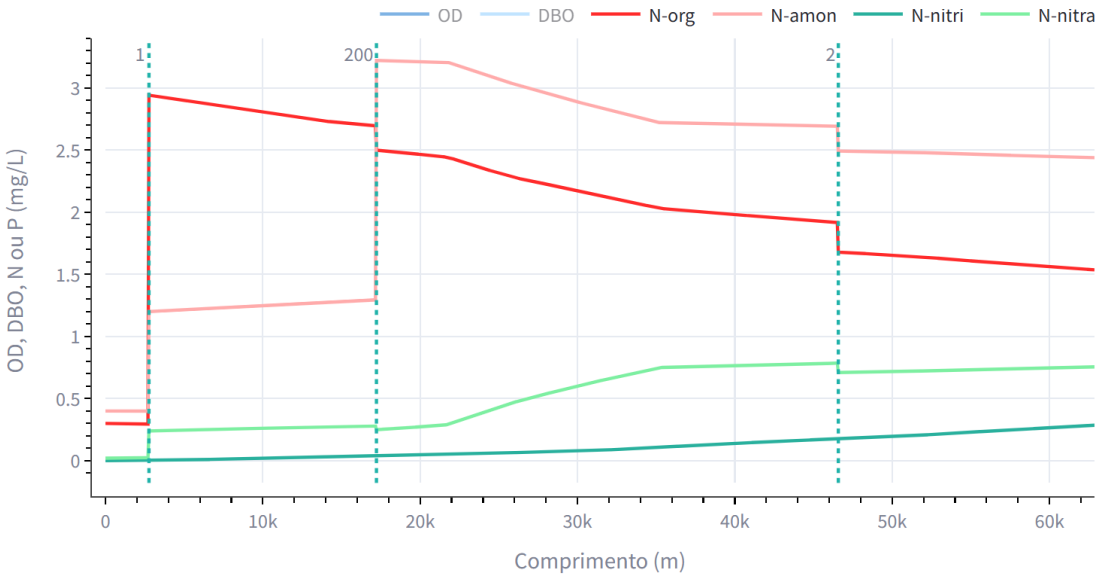
Fonte: Autora (2025).

Figura 31. Resultados de OD e DBO para o Rio principal.



Fonte: Autora (2025).

Figura 32. Resultados do Nitrogênio e suas frações para o Rio principal.



Fonte: Autora (2025).

Na parte de “Tabelas” (Figura 33), estarão disponíveis as tabelas agrupadas e separadas do rio principal e dos tributários modeláveis. O usuário tem a opção de fazer o download dessas tabelas (Figura 34), no formato CSV, essa opção é ativada quando o usuário clica em qualquer parte da tabela.

Figura 33. Resultados no formato de planilha.

Fonte: Autora (2025).

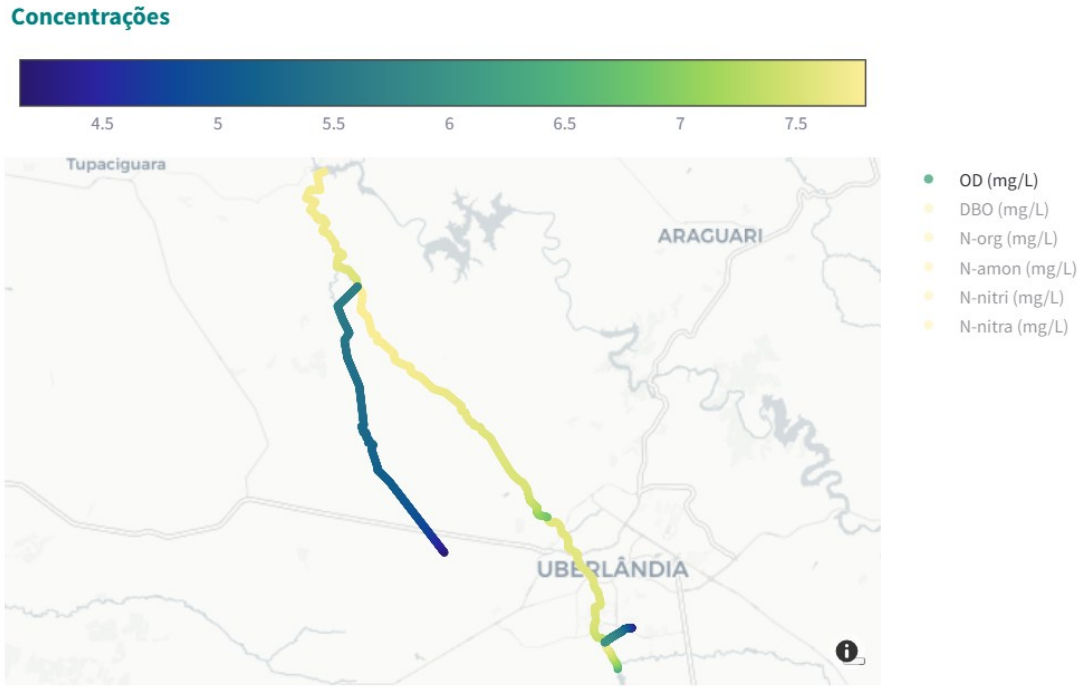
Figura 34. Resultados no formato de planilha, opção para download.

	latitude	longitude	altitude	comprimento	vazao	profundidade	velocidade	t
0	7,897,760.8603	795,868.8084	815	0	4.17	1.9024	0.0953	
1	7,897,748.169	795,820.4459	815	50	4.17	1.9024	0.0953	
2	7,897,762.1605	795,772.4942	815	100	4.17	0.199	0.911	
3	7,897,783.1402	795,727.2439	814.2268	150	4.17	0.187	0.9695	
4	7,897,805.6775	795,682.6114	813.2766	200	4.17	0.2716	0.6676	

Fonte: Autora (2025).

Na seção “Representações geoespaciais”, caso os dados sejam georreferenciados, será disponibilizada a visualização espacial dos resultados de cada parâmetro modelado ao longo de todo o trecho. Esse mapa é interativo, permitindo que o usuário visualize individualmente cada parâmetro ao clicar sobre ele. A Figura 35 apresenta um exemplo dessa representação geoespacial.

Figura 35. Visualização geoespacial dos resultados.



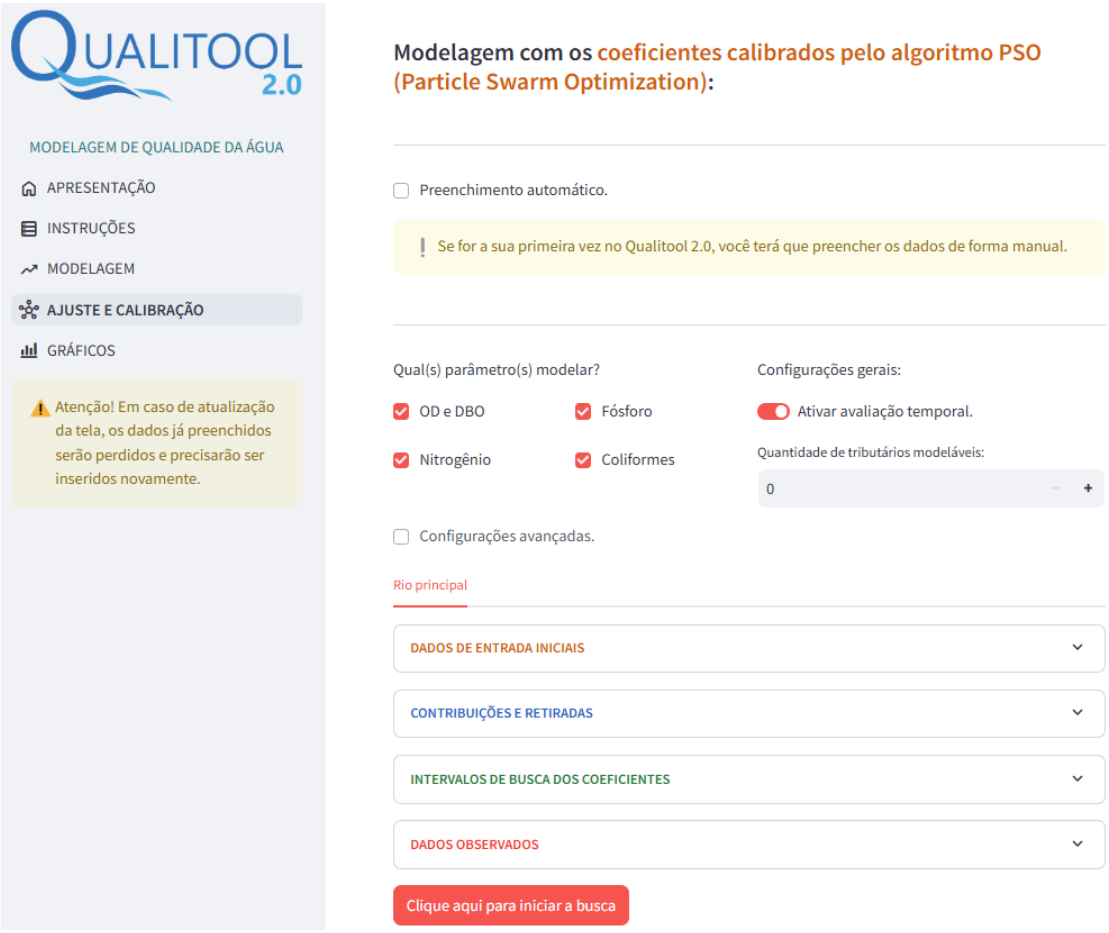
Fonte: Autora (2025).

Após gerar os perfis longitudinais para os diversos parâmetros de qualidade da água, é importante avaliar a consistência dos resultados alcançados. Neste contexto, deve-se verificar se as entradas e saídas pontuais causam variações localizadas nos perfis dos diversos parâmetros de qualidade da água. Analisar se o posicionamento longitudinal dessas variações condiz com a localização das entradas e saídas.

4.4. ABA “AJUSTE E CALIBRAÇÃO”

A opção "Ajuste e calibração" do menu principal (Figura 36) apresenta algumas semelhanças com a opção "Modelagem". Entre elas, destacam-se a funcionalidade de "Preenchimento automático", a entrada de dados gerais (que define quais parâmetros serão modelados, a ativação da avaliação temporal e o número de tributários modeláveis) e as abas "Dados de entrada iniciais" e "Contribuições e retiradas".

Figura 36. Aba “Ajuste e calibração”.



Fonte: Autora (2025).

Caso o usuário já tenha preenchido e salvo esses dados na opção "Modelagem", poderá utilizá-los por meio do preenchimento automático. No entanto, ainda será necessário inserir as informações adicionais, como as das abas "Intervalos de busca dos coeficientes" e "Dados observados".

Além disso, vale destacar a mensagem exibida na barra lateral, em um fundo amarelo claro, que alerta: “Atenção! Em caso de atualização da tela, os dados já preenchidos serão perdidos e precisarão ser inseridos novamente”. Essa mensagem tem o objetivo de informar ao usuário que qualquer ação que atualize a página resultará na perda dos dados inseridos, sendo necessário preenchê-los novamente.

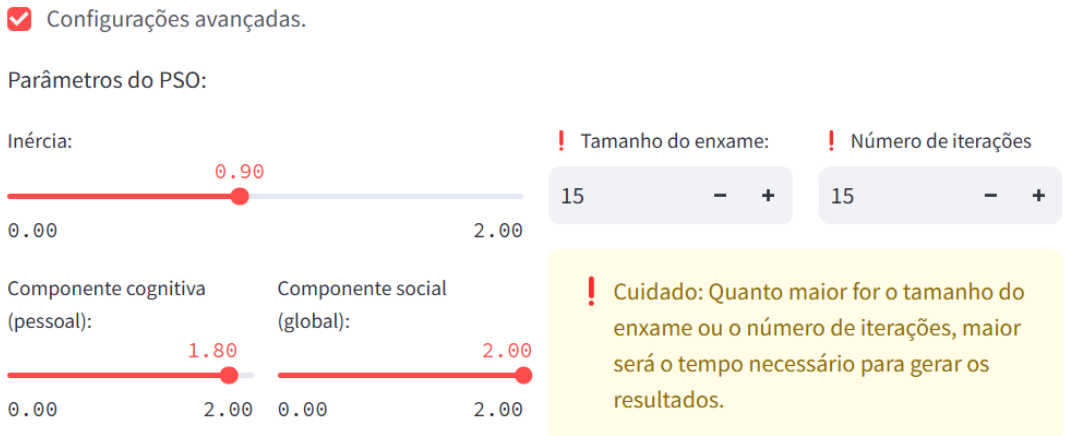
4.4.1. Entrada de dados

4.4.1.1. Configurações avançadas

Na opção "Configurações avançadas", o usuário pode ajustar os parâmetros de entrada do algoritmo PSO. Os parâmetros disponíveis para configuração incluem: coeficiente de inércia (w), componente cognitiva individual (c1), componente cognitiva global (c2), tamanho do enxame e número máximo de iterações.

Caso essa opção não seja ativada e os valores não sejam alterados, os parâmetros assumirão os seguintes valores padrão: 0,9 para o coeficiente de inércia, 1,8 para a componente cognitiva individual, 2,0 para a componente cognitiva global, 15 para o tamanho do enxame e 15 para o número máximo de iterações (Figura 37).

Figura 37. Configurações avançadas.



Fonte: Autora (2025).

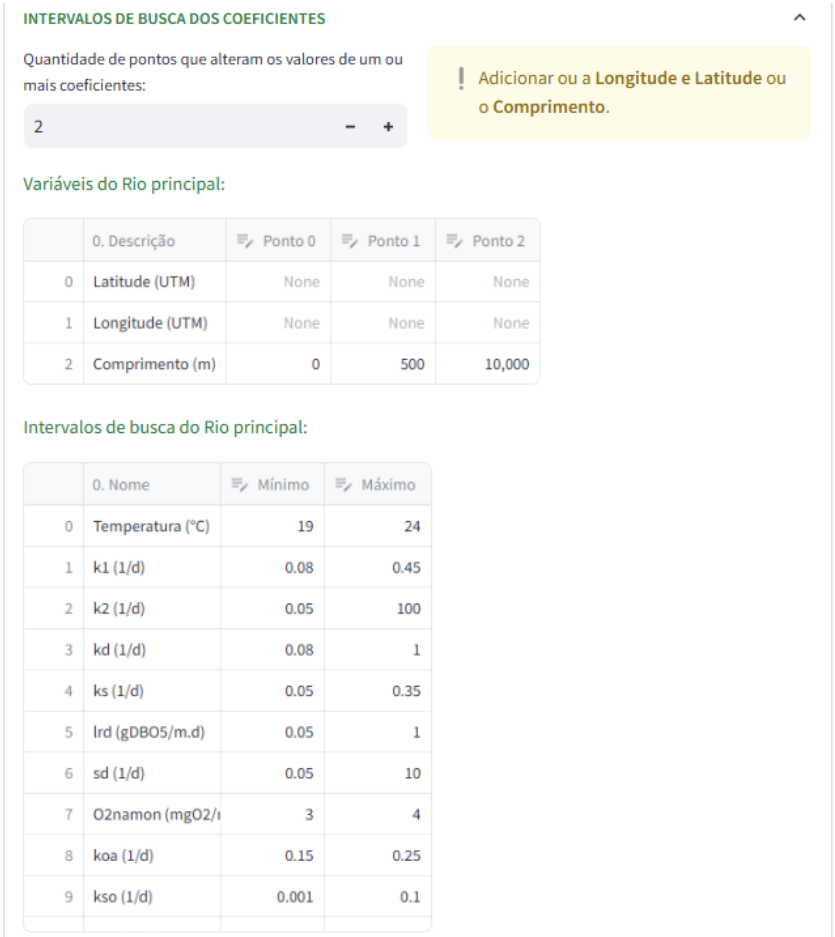
É importante ter cuidado ao extrapolar o tamanho do enxame e o número de iterações, pois isso pode impactar diretamente o tempo de processamento da calibração, tornando-o mais demorado.

4.4.1.2.Intervalos de busca dos coeficientes

Na aba “Intervalos de busca dos coeficientes” (Figura 38), é necessário informar os valores mínimo e máximo para a busca dos coeficientes, pelo menos no ponto inicial (ponto 0). Além disso, o usuário tem a opção de adicionar outros pontos de alteração dos coeficientes.

Recomenda-se consultar a literatura para definir adequadamente o intervalo de busca de cada coeficiente. No entanto, as Tabelas 3 e de 5 a 9 apresentam algumas sugestões de intervalos. Esses valores devem ser fornecidos para a temperatura de 20°C, pois o aplicativo realiza a correção automaticamente para temperatura da água no trecho.

Figura 38. Intervalo de busca dos coeficientes do Rio principal.



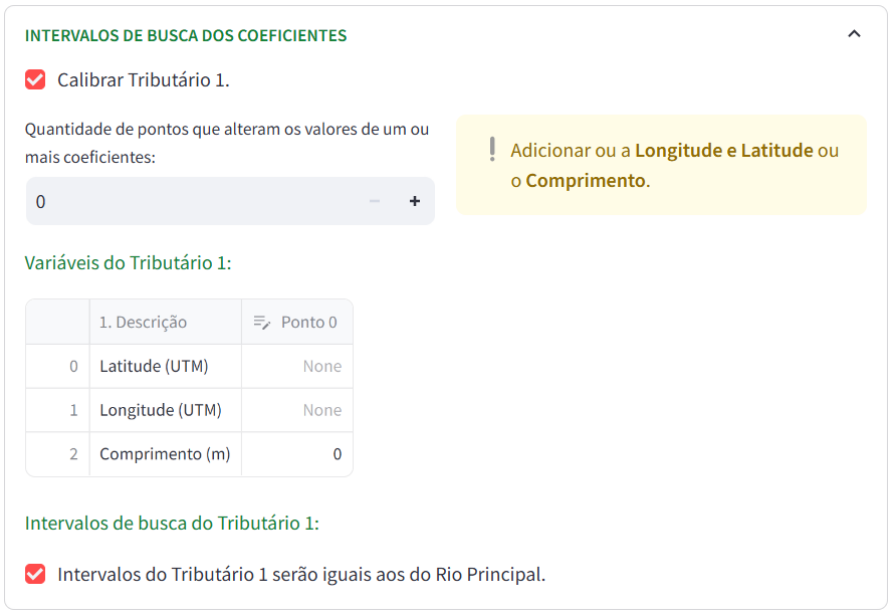
	0. Descrição	Ponto 0	Ponto 1	Ponto 2
0	Latitude (UTM)	None	None	None
1	Longitude (UTM)	None	None	None
2	Comprimento (m)	0	500	10,000

	0. Nome	Mínimo	Máximo
0	Temperatura (°C)	19	24
1	k1 (1/d)	0.08	0.45
2	k2 (1/d)	0.05	100
3	kd (1/d)	0.08	1
4	ks (1/d)	0.05	0.35
5	lrd (gDBO5/m.d)	0.05	1
6	sd (1/d)	0.05	10
7	O2namon (mgO2/	3	4
8	koa (1/d)	0.15	0.25
9	kso (1/d)	0.001	0.1

Fonte: Autora (2025).

Os tributários modeláveis podem ou não ter pontos de alteração dos coeficientes. Caso possuam, deve-se ativar a opção “Calibrar Tributário x” (Figura 39), o que fará com que a aba “Dados reais” desse trecho seja exibida. Outra opção disponível é “Intervalos do Tributário x serão iguais aos do Rio Principal”. Se essa opção for ativada, o usuário não precisará preencher os valores mínimo e máximo de cada coeficiente, pois o aplicativo utilizará os mesmos valores definidos para o rio principal.

Figura 39. Intervalo de busca dos coeficientes dos Tributários.



Fonte: Autora (2025).

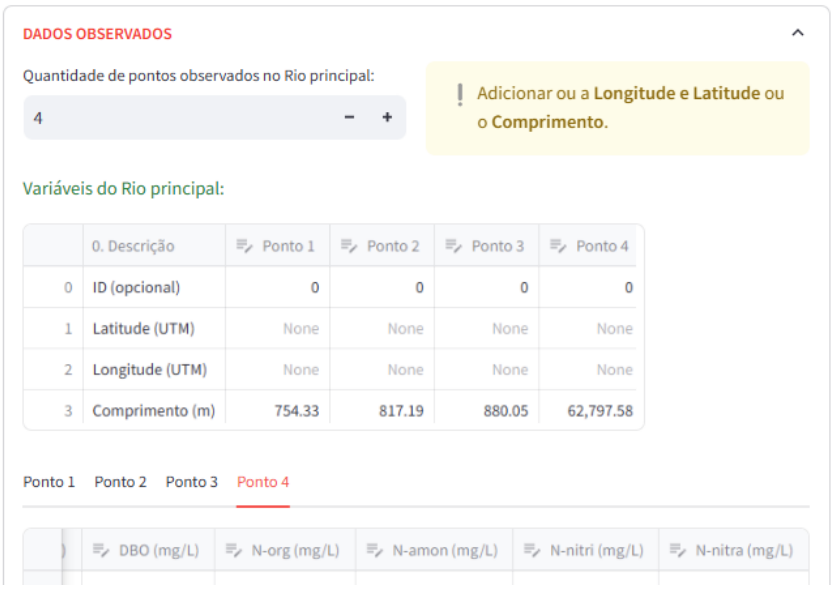
4.4.1.3.Dados observados

Na aba “Dados observados” (Figura 40), devem ser informadas as concentrações observadas em pontos reais de monitoramento, que servirão como referência para a calibração dos coeficientes.

Se a opção “Ativar a avaliação temporal” estiver ativada, o usuário poderá informar as datas em que as concentrações foram medidas. Diferente das etapas anteriores, onde as datas são fixadas, essa aba permite flexibilidade para alterar as datas conforme necessário. Isso é útil, pois o usuário pode ter registros para alguns pontos em determinadas datas e para outros em datas diferentes.

No entanto, é importante ter atenção: os dados informados só serão utilizados se a data fornecida coincidir com alguma das datas presentes nos dados de entrada iniciais.

Figura 40. Dados observados.



Fonte: Autora (2025).

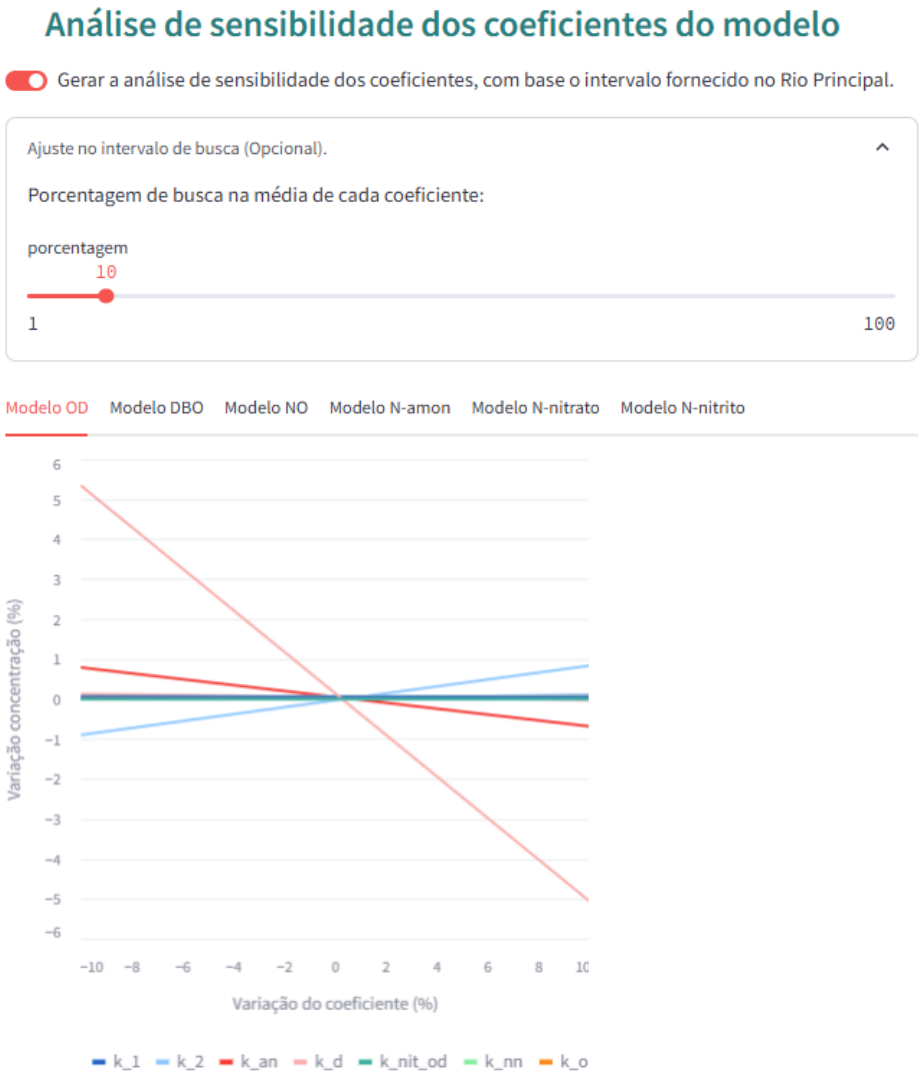
4.4.2. Análise de sensibilidade

Após preencher os dados de entrada e iniciar a processo de ajuste e calibração, o usuário será direcionado para a página "Análise de Sensibilidade dos Coeficientes do Modelo". Nesta página, é possível realizar a análise de sensibilidade por meio da análise de erros de primeira ordem, conforme detalhado no subcapítulo 3.4. Para isso, o usuário deve ativar a opção “Gerar a análise de sensibilidade dos coeficientes, com base no intervalo fornecido no Rio principal”. Ao ativar essa função, também será possível ajustar o valor da porcentagem de variação na média de cada coeficiente (Figura 41).

É importante atentar-se ao fato de que, sempre que essa porcentagem for alterada, uma nova análise será realizada. Dependendo da complexidade do sistema hídrico, esse processo pode demandar um tempo maior. A Figura 41 também apresenta os resultados da análise para cada modelo selecionado.

Ressalta-se que essa análise considera apenas dois valores de coeficiente (-x% e +x%), sendo, portanto, essencial interpretar os gráficos com cautela. Nos resultados hipotéticos da análise de sensibilidade do modelo de OD, apresentados na Figura 41, observa-se visualmente que os coeficientes k_d, k_2 e k_{an} são os mais sensíveis à alteração decorrente de uma perturbação de ±10% nos coeficientes.

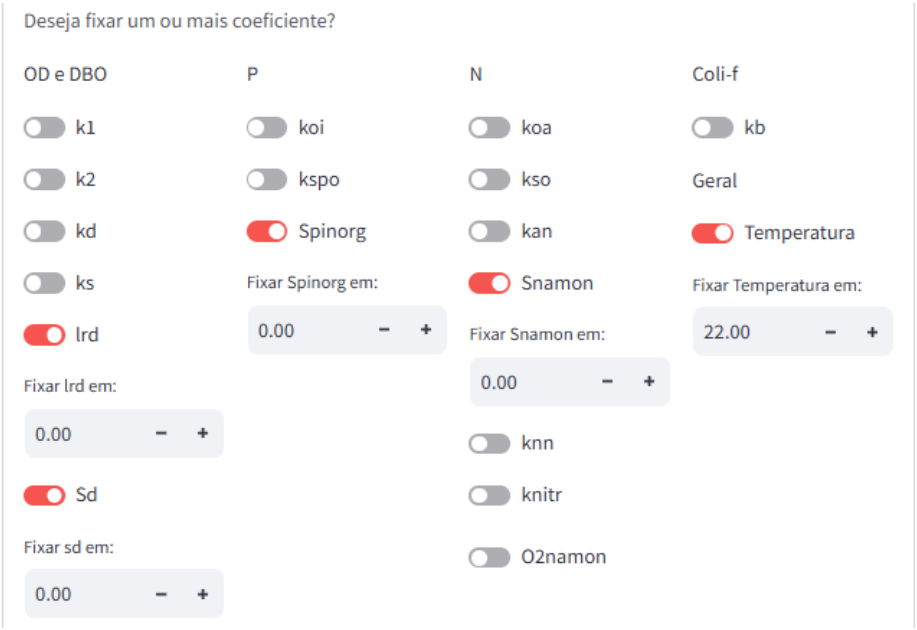
Figura 41. Análise de sensibilidade.



Fonte: Autora (2025).

Após essa etapa, o usuário poderá optar por fixar um ou mais coeficientes (Figura 42). Caso essa opção seja escolhida, os coeficientes serão fixados para todo o sistema hídrico, e não apenas para um trecho específico.

Figura 42. Opção de fixar um ou mais modelos antes da calibração.

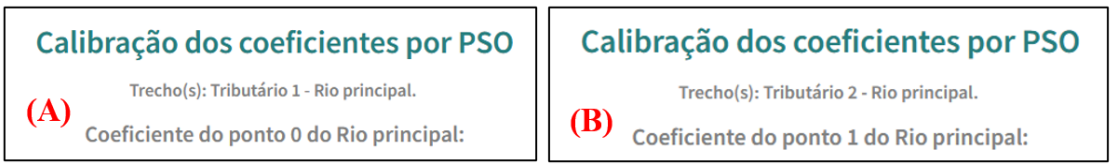


Fonte: Autora (2025).

4.4.3. Calibração por trecho

A calibração é realizada com base na quantidade de pontos a serem ajustados. Dentro de cada intervalo, podem existir tributários que não possuam pontos de alteração dos coeficientes em seu trecho. No exemplo discutido no subtópico 5.1, há dois conjuntos a serem calibrados. Nesse caso, a próxima tela apresentada ao usuário é a do primeiro conjunto (Figura 43 – A). Após a calibração desse conjunto, a tela correspondente ao próximo conjunto é exibida (Figura 43 – B). Nesta etapa, é possível visualizar quais trechos fazem parte de cada conjunto e identificar os pontos específicos de calibração que serão ajustados.

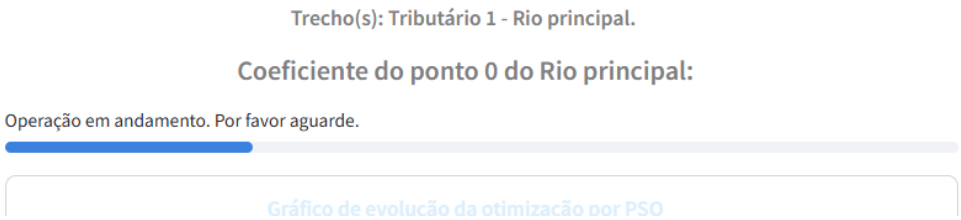
Figura 43. Exemplo de conjuntos.



Fonte: Autora (2025).

A cada execução do algoritmo PSO, uma mensagem de espera é exibida na forma de uma barra de progresso (Figura 44), indicando o número de iterações realizadas em relação ao total definido.

Figura 44. Tela de espera.

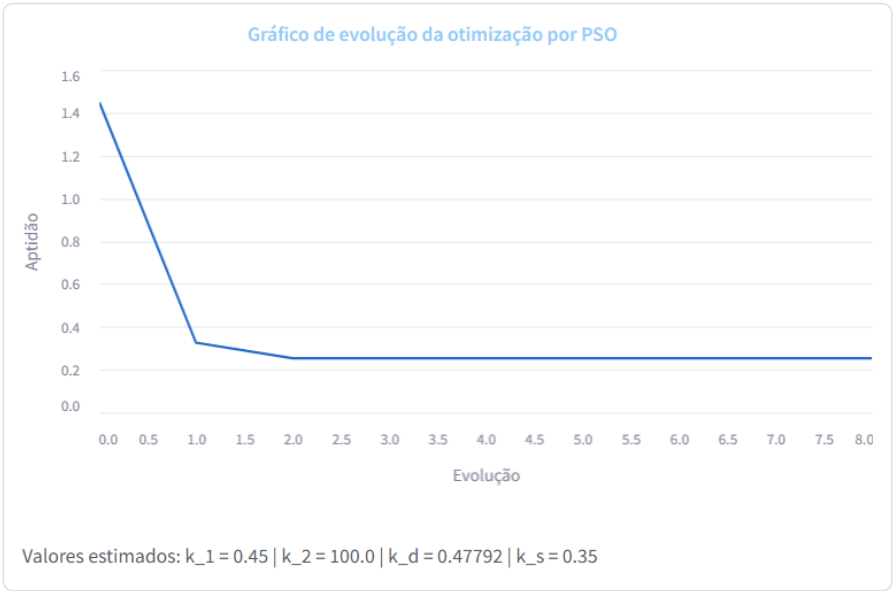


Fonte: Autora (2025).

Ao iniciar essa etapa, a primeira busca é realizada automaticamente. No entanto, ao avançar para a próxima tela, uma nova busca é executada antes da transição, pois as telas são sempre atualizadas quando há uma mudança, seguindo o padrão de funcionamento do Streamlit. A cada busca, o melhor conjunto de coeficientes e seu respectivo valor de aptidão são armazenados, garantindo que, ao final do processo, apenas o melhor dos melhores seja exibido.

A cada conclusão da busca do algoritmo PSO, é gerado um gráfico de evolução que representa o valor da aptidão em função do número de iterações (Figura 45). Como se trata de um problema de minimização, quanto mais próximo de zero, melhor o resultado da busca.

Figura 45. Evolução da busca com PSO.



Fonte: Autora (2025).

Além do gráfico de evolução, são gerados os valores do coeficiente de *Nash-Sutcliffe* para cada modelo, bem como os pontos utilizados na calibração (Figura 46). Vale ressaltar que, caso a avaliação temporal esteja ativada, todos os dias fornecidos

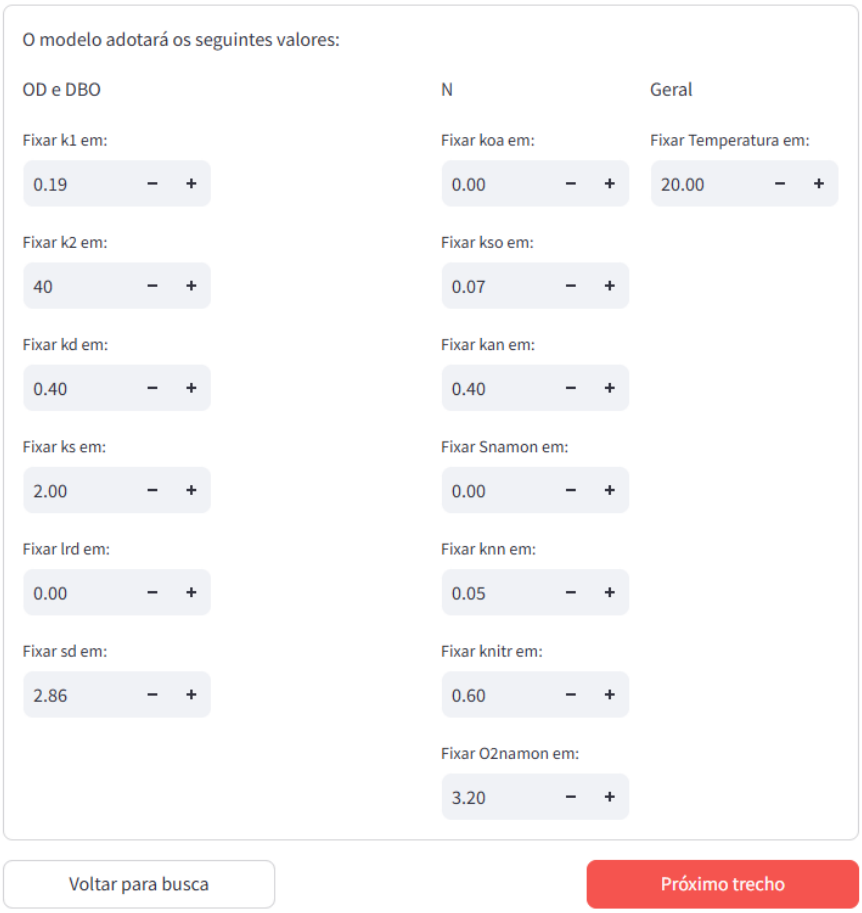
influenciarão na busca. É importante avaliar se é interessante calibrar dados de períodos chuvosos e secos conjuntamente.

Figura 46. Visualização dos resultados da calibração no trecho.

Fonte: Autora (2025).

Após a primeira busca, o usuário terá a opção de reiniciar uma nova busca ou selecionar valores tabelados para o trecho em análise. Caso opte por fixar todos os coeficientes em um conjunto específico (Figura 47), poderá seguir com os valores fixados ou retornar à calibração utilizando o PSO. Por fim, deverá selecionar a opção “Próximo trecho”.

Figura 47. Opção de fixar todos os coeficientes no trecho.

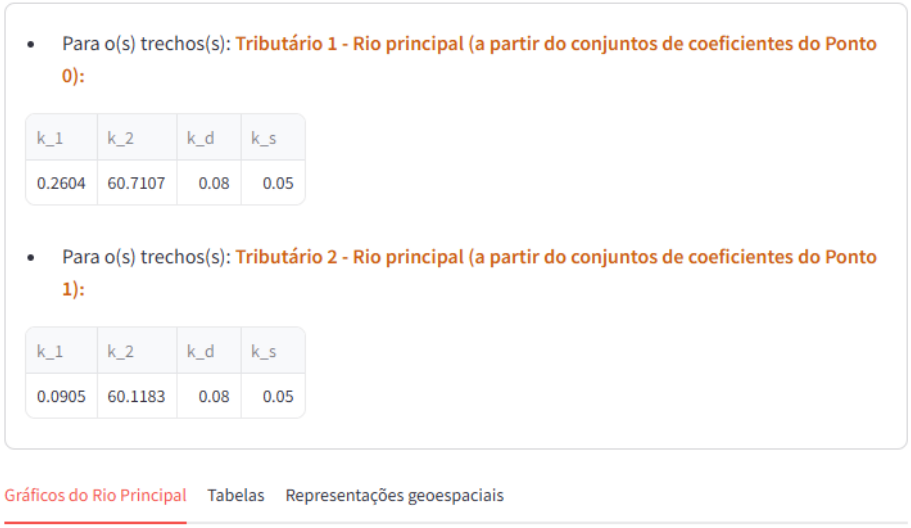


Fonte: Autora (2025).

4.4.4. Resultados

Os resultados da calibração de todos os conjuntos serão exibidos ao final (Figura 48). Além de visualizar os coeficientes obtidos, será realizada uma nova modelagem, agora utilizando os coeficientes encontrados na calibração e/ou fixados ao longo do processo. Esses resultados serão apresentados da mesma forma que na aba “Modelagem”, por meio de gráficos, tabelas e mapas, caso os dados sejam georreferenciados.

Figura 48. Resultados da calibração.



Fonte: Autora (2025).

4.5. ABA “GRÁFICOS”

Na aba “Gráficos” do menu principal, está disponível uma tela interativa para a plotagem de gráficos a partir dos dados do arquivo CSV. Essa funcionalidade foi desenvolvida para que o usuário possa ter mais liberdade para gerar gráficos do rio principal e dos seus afluentes, em escala espacial ou temporal, a partir dos resultados gerados nas abas “Modelagem” e/ou “Ajuste e Calibração”. A Figura 49 apresenta o gráfico de vazão e velocidade em função do comprimento de uma entrada hipotética.

Figura 49. Aba “Gráficos”.

Fonte: Autora (2025).

Se os dados da planilha gerada não estiverem em séries temporais ou forem obtidos a partir de uma planilha agrupada com todos os rios, é possível utilizar algumas funções disponíveis. O usuário pode escolher o rio a ser plotado, sendo "0" o rio principal e os valores subsequentes (1 em diante) correspondentes aos afluentes, seguindo a ordem estabelecida nos dados de entrada. Além disso, é possível selecionar as variáveis para os eixos x e y, permitindo a escolha de múltiplas variáveis para o eixo y. Caso os dados estejam em séries temporais, também é possível selecionar a data como variável no eixo x ou y.

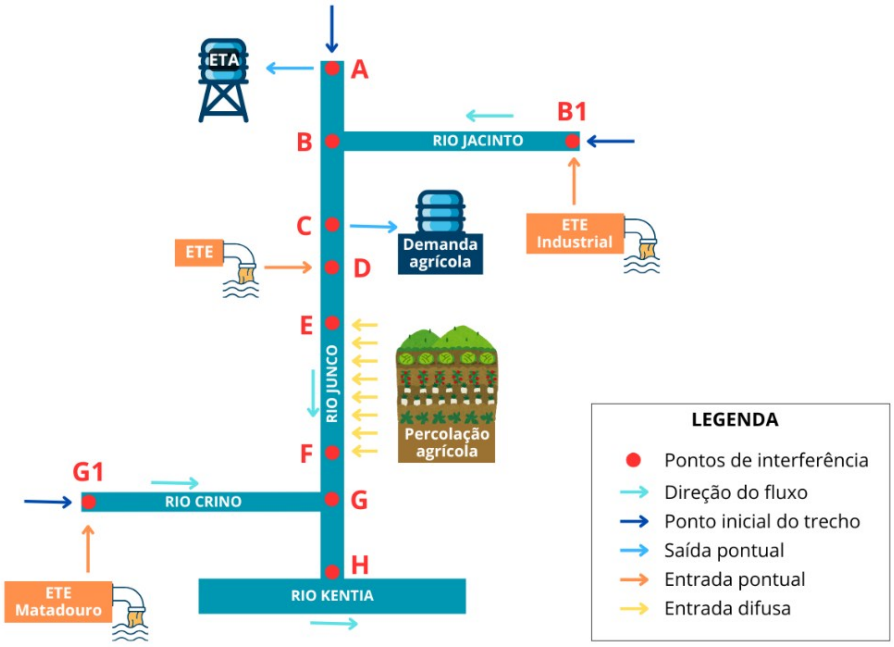
5. COMPARAÇÃO DO QUALITOOL 2.0 COM OUTRAS FERRAMENTAS DE USO LIVRE

Este capítulo apresenta os resultados da aplicação do QUALITOOL 2.0 em exemplos previamente modelados em outras ferramentas de modelagem da qualidade da água. O objetivo é avaliar o desempenho e validar o aplicativo QUALITOOL 2.0. As simulações foram conduzidas em um computador com processador 13th Gen Intel® Core™ i7-1355U, com frequência de 1,7 GHz, 10 núcleos e 12 threads. O sistema conta com 16 GB de memória RAM e um SSD NVMe BC901 de 1 TB. O sistema operacional utilizado foi o Windows 11 versão 23H2. Como o QUALITOOL 2.0 é disponibilizado em formato web, a qualidade da internet durante os testes foi medida com base na conexão de rede utilizada. A conexão foi realizada por meio de uma rede de fibra ótica de 150 Mbps, com velocidade de download de 72,1 Mbps e velocidade de upload de 37,0 Mbps. A latência foi monitorada e apresentou valores médios de 11 ms.

5.1. Modelagem QUALITOOL versus QUALITOOL 2.0

O primeiro exemplo escolhido para a comparação dos resultados é o proposto por Alamy Filho *et al.* (2019), no qual os autores utilizam um exemplo fictício para apresentar a primeira versão do aplicativo QUALITOOL, além de avaliar a acurácia dos resultados gerados por meio da comparação com os resultados obtidos pelas ferramentas QUAL-UFMG e AQUATOOL (módulo GESCAL). Esse exemplo trata de um sistema hídrico fictício (Figura 50), no qual, além do rio principal (rio Junco), são modelados mais dois tributários: o rio Jacinto e o rio Crino.

Figura 50. Ilustração do sistema hídrico do Rio Jacinto (fictício).



Fonte: Adaptado de Alamy Filho *et al.* (2019)

Uma vantagem desse exemplo é a disponibilidade de dados georreferenciados para cada rio (Tabela 10), obtidos por meio de ferramentas SIG. Além disso, ele apresenta diferentes tipos de dados de entrada (Tabelas 11 e 12), incluindo fontes pontuais (Estações de Tratamento de Esgoto – ETE), entrada difusa (por percolação agrícola) e retiradas pontuais (para captação de água na Estação de Tratamento de Água – ETA e para demanda agrícola). Também são apresentados diferentes pontos de alteração das seções transversais dos rios (Tabela 13).

Tabela 10. Topologia do sistema hídrico do exemplo 1.

Trecho	Ponto	Coordenada UTM (m)			Trecho	Ponto	Coordenada UTM (m)		
		X	Y	Z			X	Y	Z
A-B	A	787148	7896781	796,0	D-E	D	781048	7910480	754,5
		787186	7896928	794,5			780443	7910631	746,0
		787024	7897385	794,0			780298	7910629	744,5
		786750	7897742	792,5			780079	7910937	739,5
		786526	7898274	791,0			780161	7911344	739,0
		786248	7898582	789,5			779499	7911952	736,5
	B	786019	7899201	789,0			779339	7912600	736,0
B1-B	B1	788490	7900459	843,0			779276	7912954	734,0
		787849	7900384	839,0			778882	7913608	729,5
		787431	7900070	833,0		E	778636	7913708	729,0
		786670	7899669	812,0	E-F	E	778636	7913708	729,0
		786337	7899418	797,0			778138	7914156	726,0
	B	786019	7899201	789,0			777721	7914664	724,0
B-C	B	786019	7899201	789,0			776762	7916280	723,0
		785835	7899573	788,5			775985	7917673	719,0
		785361	7899627	787,3			775314	7918188	700,0

Tabela 10 – Topologia do sistema hídrico do exemplo 1 (continuação).

		785045	7900194	786,0			775113	7918603	684,0
		785103	7900831	785,5			774547	7918741	683,0
		785276	7901120	785,0			774508	7918964	669,5
		785226	7901491	784,6			774192	7919125	669,0
		785151	7902343	783,8			773868	7919632	665,0
		785257	7902388	783,3	E-F		773796	7919951	663,0
		785698	7902500	783,0			773442	7920084	662,0
		785706	7902684	782,6			773481	7920495	660,0
		785485	7903766	782,2			772047	7921719	649,0
		785221	7904047	782,0			770955	7922075	645,0
B-C		785110	7904376	781,0		F	769930	7923080	644,0
		785096	7904787	780,5		F	769930	7923080	644,0
		784454	7905656	779,0			769875	7923154	643,0
		783935	7906062	776,0			769339	7923377	638,5
		783667	7906789	775,7			768941	7923666	638,0
		783718	7907144	775,2			768947	7924068	637,5
		782875	7907693	774,0			768586	7924183	637,0
		782872	7908481	773,0			768149	7924354	635,0
		782685	7908776	772,0			767503	7924723	632,5
		782312	7909035	771,0	F-G		767465	7925292	632,0
	C	782175	7909654	768,0			766269	7926380	623,5
	C	782175	7909654	768,0			765973	7926399	623,0
		782002	7909898	765,5			765968	7926922	622,0
		781730	7910040	763,5			765531	7927082	619,0
C-D		781467	7909927	761,0			765159	7928591	612,0
		781236	7910040	758,0			764514	7929199	610,0
		781106	7910167	756,0			764729	7930647	600,0
	D	781048	7910480	754,5		G	764292	7931337	599,5
	G1	771737	7907434	793,0		G	764292	7931337	599,5
		766820	7913938	769,0			764183	7931948	599,0
		765827	7914886	761,0			763719	7932519	588,0
		765878	7915037	760,0			763413	7932885	578,0
		765326	7916729	757,0			763097	7932877	575,0
		765481	7917166	756,0			762899	7933042	565,0
		765347	7917280	752,0			762430	7933077	560,0
		765023	7917359	748,0			762530	7933920	558,0
		765118	7917541	745,0			762981	7934050	555,0
		764989	7917761	738,5			763004	7934213	552,0
		764876	7917799	738,0			762353	7934689	551,0
		764941	7918050	734,5			762346	7935136	545,5
		764730	7918190	734,0			761673	7936103	545,0
G1-G		764846	7918255	733,0	G-H		761208	7936285	544,5
		764699	7918309	730,5			761207	7936516	544,0
		764602	7918270	730,0			761557	7937050	542,0
		764568	7918592	729,0			761192	7937500	541,0
		764425	7918794	728,5			760897	7937291	539,0
		764667	7918915	728,0			760645	7937558	537,0
		764304	7922427	708,0			760748	7937696	534,0
		763274	7924932	648,0			760086	7938225	533,5
		763018	7926530	643,0			760617	7938937	533,0
		763486	7927184	636,0			759737	7939281	524,5
		762923	7928319	623,0			760088	7940036	524,0
		762445	7929593	607,0			761039	7939976	523,0
	G	764292	7931337	599,5			760783	7941372	518,0
						H	761419	7941662	516,0

Tabela 11. Vazões pontuais e difusas do exemplo 1.

Rio	Descrição	Tipo	$Ponto_{inicial}$	$Ponto_{final}$	Q (m^3/s)
Jacinto	Afluente	Dados iniciais	B1	-	1,5
	Indústria de alimentos	Entrada pontual	B1	-	0,8
Crino	Afluente	Dados iniciais	G1	-	1,1
	Matadouro	Entrada pontual	G1	-	0,5
Junco	Afluente	Dados iniciais	A	-	6,2
	Abastecimento público	Saída pontual	A	-	1,2
	Demanda agrícola	Saída pontual	C	-	1,8
	ETE	Entrada pontual	D	-	0,9
	Percolação agrícola	Entrada difusa	E	F	1,3461

Tabela 12. Concentrações pontuais e difusas do exemplo 1.

Rio	Descrição	OD (mg/L)	DBO (mg/L)	NO (mg/L)	NH_3 (mg/L)	NO_2- (mg/L)
Jacinto	Afluente	6,0	4,0	0,1	0,2	0,01
	Indústria de alimentos	1,0	260	25	8,0	2,0
Crino	Afluente	5,8	3,0	0,2	0,3	0,02
	Matadouro	0,5	440	1,5	4,5	0,8
Junco	Afluente	6,3	5,0	0,3	0,4	0,02
	ETE	1,5	65	1,3	15,0	0,07
	Percolação agrícola	5,8	6,0	0,9	0,6	2,5

Tabela 13. Dados geométricos e hidráulicos da seção transversal do exemplo 1.

Rio	Ponto	b (m)	α_{esq}	η_{calha}	α_{dir}	Discretização (m)
Jacinto (Tributário 1)	B1	6,0	47	0,030	75	5,67
	Entre B1-B	6,5	50	0,030	70	
	B	7,2	45	0,030	65	
Crino (Tributário 2)	G1	5,0	45	0,035	48	28,63
	Entre G1-G	5,3	46	0,035	50	
	Entre G1-G	6,5	48	0,036	52	
	G	7,0	50	0,038	55	
Junco (Rio principal)	A	14,0	48	0,042	45	62,86
	B	16,0	46	0,042	47	
	C	16,5	46	0,042	49	
	D	17,5	48	0,042	45	
	E	18,0	45	0,042	47	
	F	18,5	46	0,045	48	
	G	21,0	48	0,045	45	
	H	24,0	45	0,043	47	

Neste exemplo, também são fornecidos os valores dos coeficientes do modelo (Tabela 14). Há apenas um conjunto de coeficientes ao longo de todo o sistema hídrico, exceto pelo k_2 (coeficiente de reaeração), que é modificado ao longo do rio por meio de equações hidráulicas da literatura, conforme descrito no subcapítulo 3.3.5.

Tabela 14. Coeficientes dos processos físicos e bioquímicos do exemplo 1.

Geral	Temperatura (ºC)					
	22,0					
OD e DBO	k2 máx. (d^{-1})	k1 (1/d)	kd (1/d)	ks (1/d)	sd (1/d)	lrd (gDBO5/m.d)
	1000	0,30262	0,05	0, 1	0	0
OD + N	O2namon (mgO2/ mgNamon oxid)					
	3,2					
N	knn (1/d)	koa (1/d)	kan (1/d)	kso (1/d)	knitr (1/d)	Snamon (g/m2.d)
	0,5	0,21	0,2	0,1	0,6	0

A fim de comparar o desempenho das duas versões QUALITOOL, foram realizados testes cronometrados manualmente para medir o tempo de execução após o preenchimento de todos os dados de entrada (Tabela 15). Como o trecho é relativamente simples, com apenas dois pequenos tributários, o tempo médio gasto para realizar a simulação no QUALITOOL 2.0 foi de 10,9 segundos, correspondendo a 3,85% do tempo médio gasto no QUALITOOL.

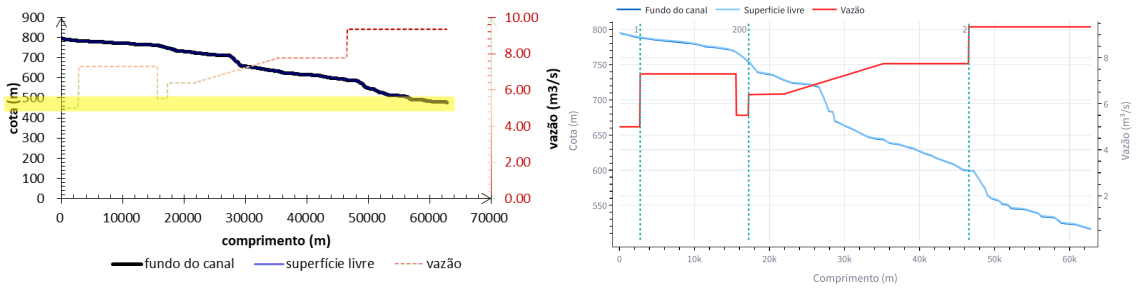
Vale ressaltar que ambos possuem a resolução da equação empírica de Manning (Equação 3) a cada intervalo discretizado. Neste caso, apenas o rio principal possui 620 intervalos, com 100 metros de comprimento cada.

Tabela 15. Tempo de execução do QUALITOOL x QUALITOOL 2.0, em segundos.

Teste	QUALITOOL	QUALITOOL 2.0
1	332,14	12,67
2	249,16	11,00
3	267,10	9,01
Média	282,80	10,90

A Figura 51 apresenta os resultados da vazão e da cota do nível do canal principal, sendo o gráfico à esquerda gerado pelo QUALITOOL e o da direita, pelo QUALITOOL 2.0. De acordo com a Tabela 10, que contém os dados de topologia, a altitude final do canal deveria ser de aproximadamente 516,0 metros.

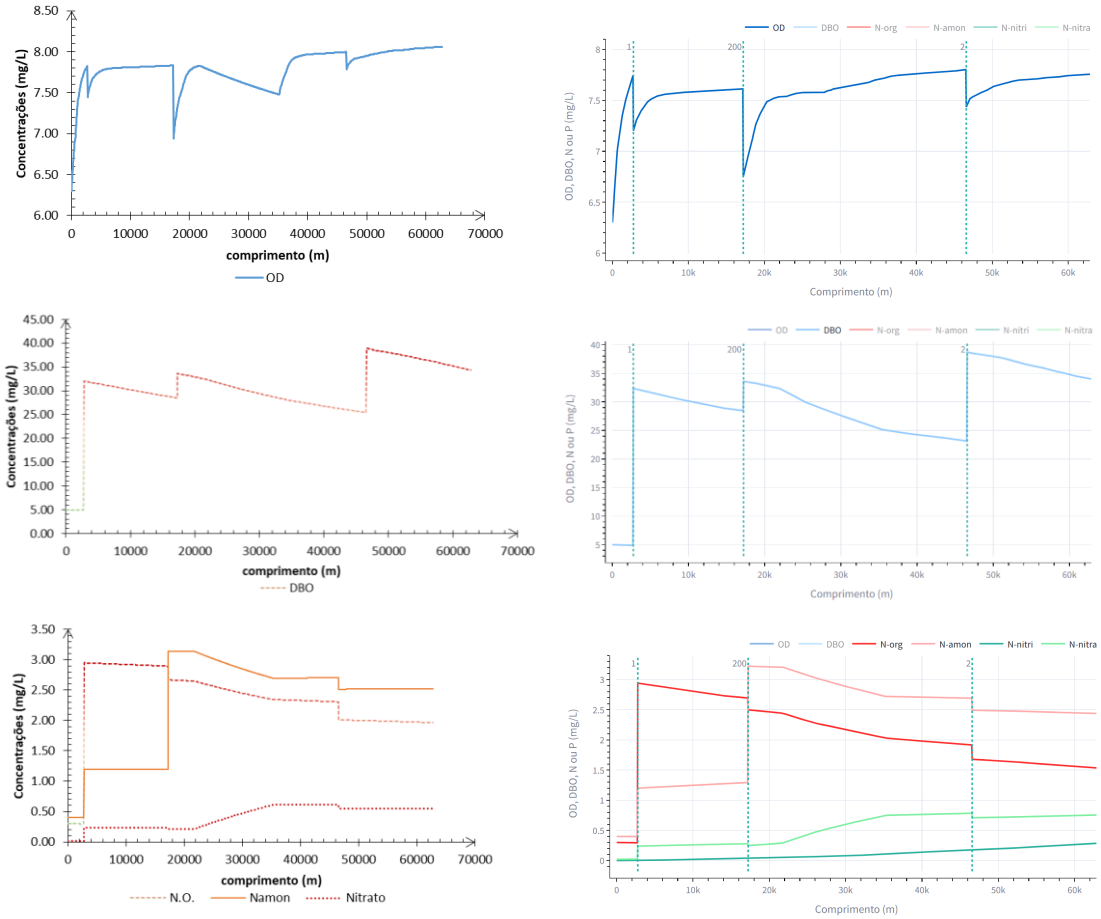
Figura 51. Resultados hidrodinâmicos no QUALITOOL e QUALITOOL 2.0.



Fonte: Autora (2025).

Já a Figura 52 apresenta os resultados das concentrações de OD, DBO e nitrogênio, além de suas frações, sequencialmente. Os resultados da coluna à esquerda foram gerados no QUALITOOL, enquanto os da coluna à direita correspondem ao QUALITOOL 2.0.

Figura 52. Resultados das concentrações no QUALITOOL e QUALITOOL 2.0.



Fonte: Autora (2025).

Todos os resultados das concentrações são muito próximos, exceto na modelagem do OD no ponto de contribuição difusa (aproximadamente entre 20 e 40 km de comprimento). Essa diferença ocorre devido a uma falha na distribuição da vazão de entrada da contribuição difusa no QUALITOOL, que foi devidamente corrigida no QUALITOOL 2.0. Para facilitar o cálculo, agora o usuário deve inserir a vazão total (m^3/s) da contribuição difusa. O objetivo deste exemplo não é invalidar a primeira versão do QUALITOOL, que, dentro de suas limitações, assim como o QUALITOOL 2.0, apresenta um funcionamento adequado. O propósito é reconhecer essas limitações, identificar pontos de melhoria, preservar os aspectos já consolidados e aprimorar ainda mais o QUALITOOL.

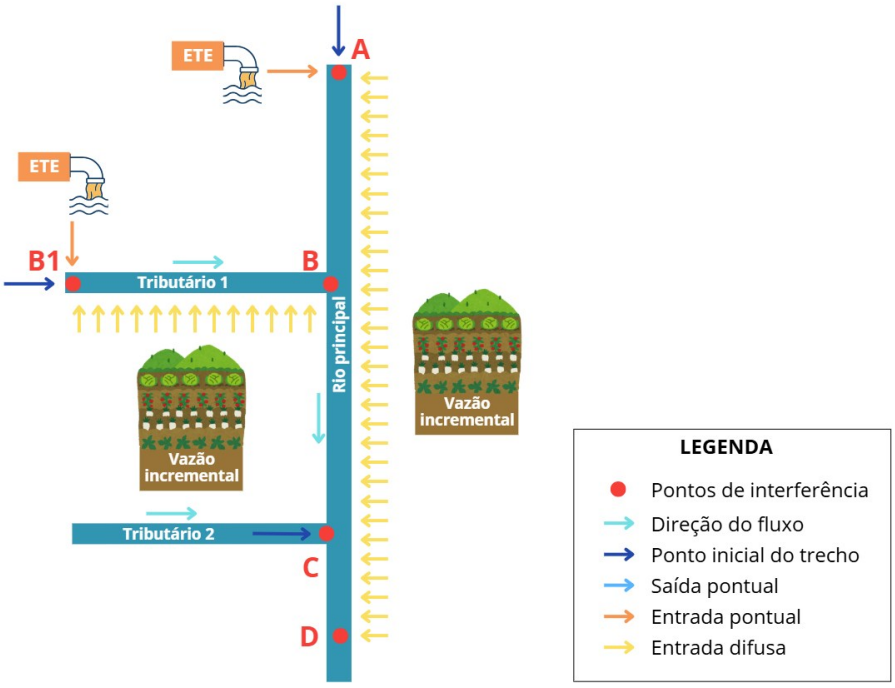
Dessa forma, destacam-se melhorias como a redução do tempo de execução, a eliminação das restrições quanto ao número de tributários e sua estrutura (permitindo que um tributário deságue em outro), a possibilidade de inserção de dados geoespaciais diretamente a partir de arquivos SIG (GeoJSON), a modelagem do OD em condições anaeróbicas, entre diversas outras melhorias já discutidas ao longo deste trabalho.

5.2. Modelagem QUAL-UFMG versus QUALITOOL 2.0

O modelo QUAL-UFMG é amplamente aceito para a modelagem da qualidade da água (BRUM, M. *et al.*, 2022; GOMIDES, C.E. *et al.*; PANI, D.F. *et al.*, 2024). Segundo Von Sperling (2014b), esse modelo é especialmente utilizado em pesquisas majoritariamente brasileiras, pois sua facilidade de uso, interface simples e disponibilidade em português tornam sua aplicação mais acessível.

O próximo exemplo aplicado é o proposto no capítulo 13 do livro *Estudos e Modelagem da Qualidade da Água de Rios*, de Von Sperling (2014b). Esse exemplo trata de um sistema hídrico fictício (Figura 53), no qual, além do rio principal, há dois tributários, sendo um modelável e outro não.

Figura 53. Ilustração do sistema hídrico do exemplo 2.



Fonte: Adaptado de Von Sperling (2014b).

Este segundo exemplo não possui dados georreferenciados; na Tabela 16, está disponível o comprimento total de cada trecho. Neste exemplo, são incluídas fontes pontuais (Estações de Tratamento de Esgoto – ETE) e entradas difusas (denominadas vazão incremental), cujos dados de vazão e concentração estão apresentados nas Tabelas 17 e 18. Para este exemplo, foram modelados os parâmetros de OD, DBO, nitrogênio e suas frações, fósforo e suas frações, além de coliformes.

Tabela 16. Comprimento total de cada trecho do exemplo 2.

Trecho	Comprimento (km)
A-B	50
B-C	20
C-D	25
B1-B	15

Tabela 17. Vazões pontuais e difusas do exemplo 2.

Rio	Descrição	Tipo	$Ponto_{inicial}$	$Ponto_{final}$	Q (m^3/s)
Tributário 1 (T1)	Afluente	Dados iniciais	B1	-	0,2500
	ETE	Entrada pontual	B1	-	0,0400
	Vazão incremental	Entrada difusa	B1	B	0,0975
Rio principal (RP)	Afluente	Dados iniciais	A	-	0,7600
	ETE	Entrada pontual	A	-	0,1140
	Tributário 2	Entrada pontual	C	-	0,4500
	Vazão incremental	Entrada difusa	A	D	0,6175

Tabela 18. Concentrações pontuais e difusas do exemplo 2.

Rio	Descrição	OD[1]	DBO[1]	NO[1]	NH_3[1]	NO_2-[1]	NO_3-[1]	P_{org}[1]	P_{inorg}[1]	E-coli[2]
T1	Afluente	7,1	2,0	1,0	1,0	0,0	0,0	0,01	0,01	10
	ETE	0,0	300,0	20,0	25,0	0,0	0,0	2,0	5,0	5.10^7
	Vazão incremental	7,5	2,0	1,0	1,0	0,0	0,0	0,01	0,01	10
RB	Afluente	7,1	2,0	1,0	1,0	0,0	0,0	0,01	0,01	10
	ETE	0,0	341,0	20,0	25,0	0,0	0,0	2,0	5,0	5.10^7
	Tributário 2	7,7	1,0	1,0	1,0	0,0	0,0	0,01	0,01	10
	Vazão incremental	7,5	2,0	1,0	1,0	0,0	0,0	0,01	0,01	10

Obs.: [1]: mg/L; [2]: NMP/100mL.

O cálculo da velocidade e do nível d'água no QUAL-UFMG é padronizado por meio de equações que correlacionam esses parâmetros com a vazão, exigindo o conhecimento de séries temporais do regime fluviométrico do corpo d'água em estudo. No entanto, o usuário também pode, com pequenos ajustes, modificar essa equação para outra que seja mais adequada à sua análise, como a Equação de Manning (Equação 3).

No exemplo resolvido no QUAL-UFMG, de acordo com Von Sperling 92014b), foram utilizados valores médios fixos monitorados pelo próprio autor em diversos cursos

d’água da região metropolitana do Estado de Minas Gerais. Com os valores de vazão, largura (b), velocidade e nível d’água conhecidos, foi realizado o cálculo inverso da Equação de Manning para determinar a inclinação ao longo desse sistema hídrico fictício. A Tabela 19 apresenta os valores das seções transversais no início e no final de cada trecho.

Tabela 19. Dados geométricos e hidráulicos da seção transversal do exemplo 2.

Rio	Ponto	b (m)	α_{esq}	η_{calha}	α_{dir}	Discretização (m)
Tributário 1	B1	12,485	90	0,045	90	100
	B	12,609	90	0,045	90	
Rio principal	A	12,965	90	0,045	90	
	D	13,406	90	0,045	90	

Além disso, sabendo que a altitude inicial do rio principal é de 720 metros e a inclinação ao longo do corpo d’água, foi possível determinar a altitude ao longo dos trechos. Essa informação de altitude é necessária para a entrada de dados no QUALITOOL 2.0.

Neste exemplo, também são fornecidos os valores dos coeficientes do modelo (Tabela 20). Há apenas um conjunto de coeficientes ao longo de todo o sistema hídrico, exceto pelo k_2 (coeficiente de reaeração), que é modificado ao longo do rio por meio de equações hidráulicas da literatura, conforme descrito no subcapítulo 3.3.5.

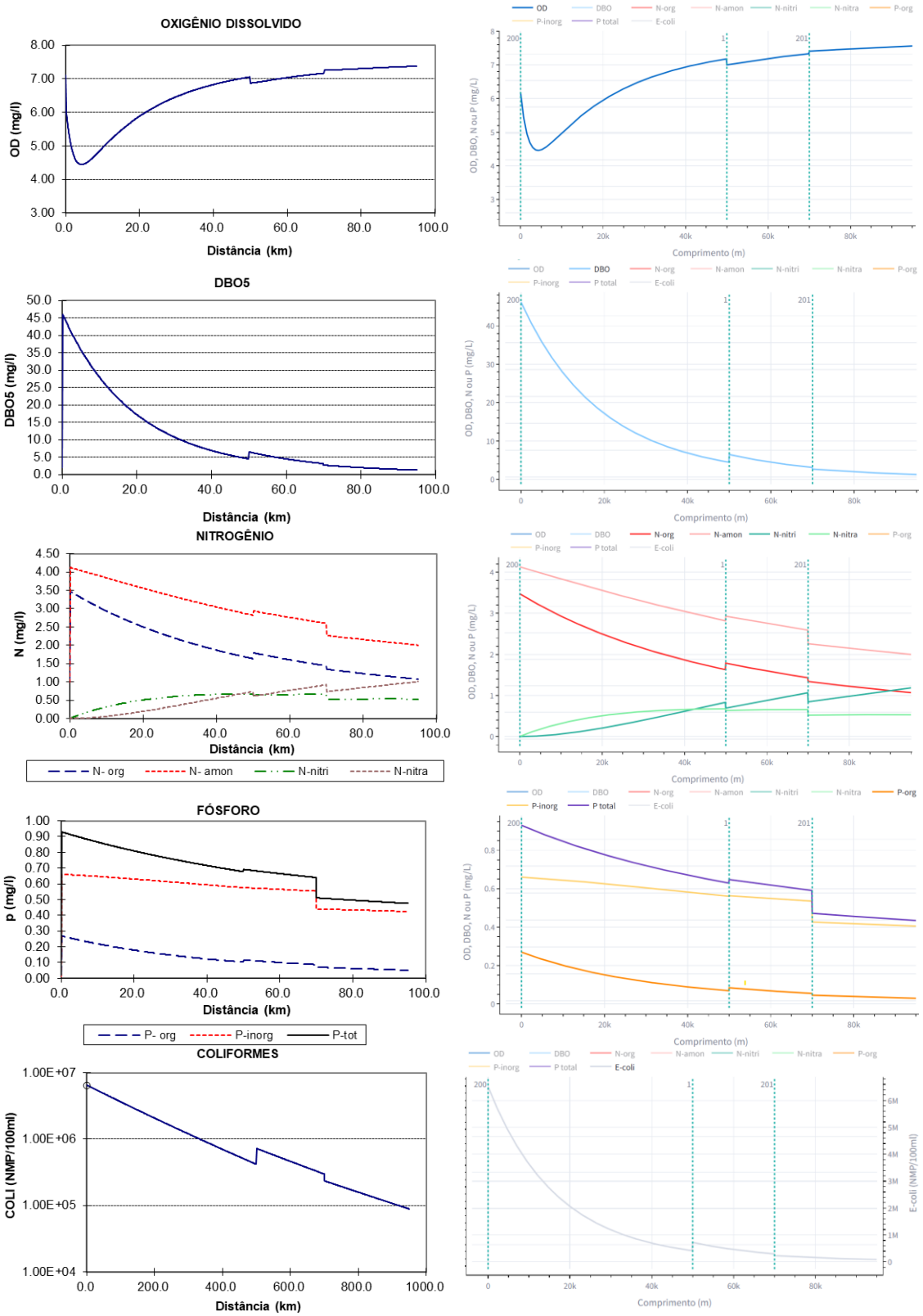
Tabela 20. Coeficientes dos processos físicos e bioquímicos do exemplo 2.

Geral	Temperatura (ºC)					
	22,0					
OD e DBO	k2 máx. (d^{-1})	k1 (1/d)	kd (1/d)	ks (1/d)	sd (1/d)	lrd (gDBO5/m.d)
	10	0,4	0,7	0,2	0	0,2
OD + N	O2namon (mgO2/ mgNamon oxid)					
	3,3					
N	knn (1/d)	koa (1/d)	kan (1/d)	kso (1/d)	knitr (1/d)	Snamon (g/m2.d)
	0,75	0,2	0,2	0,05	0,6	0
P	koi (1/d)		kspo (1/d)		Spinorg (1/d)	
	0,27		0,2		0	
E-coli	kb (1/d)					
	1,0					

A Figura 54 apresenta os resultados das concentrações, com os valores gerados pelo QUAL-UFMG na coluna à esquerda e pelo QUALITOOL 2.0 na coluna à direita, sendo que todas as concentrações exibem, visualmente, curvas idênticas. No entanto, é importante atentar-se à visualização das concentrações de coliformes, pois o gráfico no

QUALITOOL 2.0 é ajustado para melhor interpretação. Observe que os intervalos estão todos na ordem de milhões (10^6).

Figura 54. Resultados das concentrações no QUAL-UFMG e QUALITOOL 2.0.



Fonte: Autora (2025).

Para quantificar a correlação entre os resultados dos dois modelos (Tabela 21), foram utilizados os seguintes indicadores estatísticos: o coeficiente de determinação (R^2), conforme a Equação 35, que mede a proporção da variabilidade dos dados; o erro absoluto médio (MAPE – *Mean Absolute Percentage Error*), conforme a Equação 36; o erro quadrático médio (MSE – *Mean Squared Error*), conforme a Equação 37, que penaliza erros maiores; e a raiz do erro quadrático médio (RMSE – *Root Mean Squared Error*), conforme a Equação 38, que facilita a interpretação do MSE ao manter a unidade dos dados originais.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y_i})^2}{\sum(y_i - \bar{y})^2}$$

Eq. 35

$$MAPE = \frac{1}{n}\sum_{i}^{n}\frac{|y_i - \hat{y_i}|}{y_i}.100$$

Eq. 36

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y_i})^2$$

Eq. 37

$$RMSE = \sqrt{MSE}$$

Eq. 38

Na qual, y_i são os resultados gerados pelo QUAL-UFMG; $\hat{y_i}$ são os resultados gerados pelo QUALITOOL 2.0.

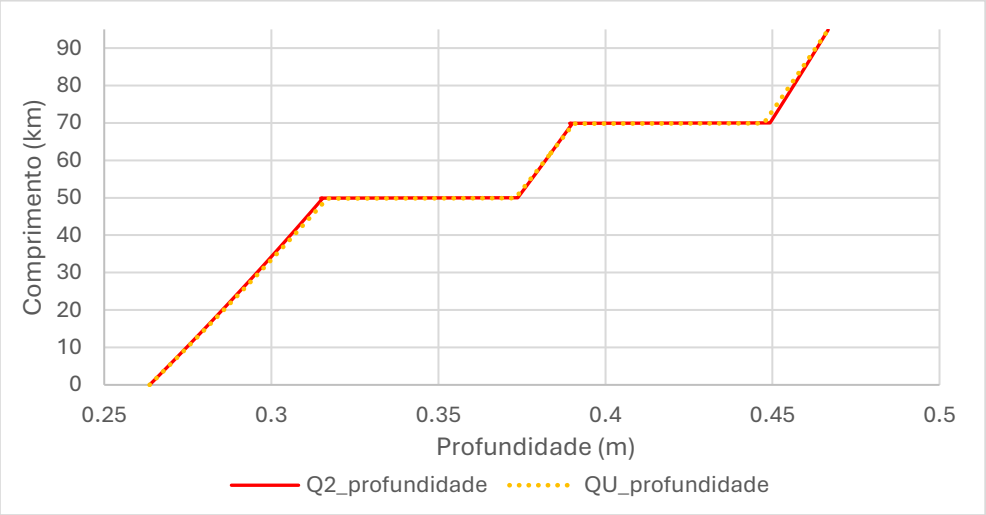
Tabela 21. Métricas obtidas no exemplo 2.

	OD	**DBO**	**N_{org}**	**N_{amon}**	**N_{nitri}**	**N_{nitra}**	**P_{org}**	**P_{inorg}**	**E-coli**
R^2	0,99998	1,0	0,99999	0,99996	0,99995	0,99873	0,91929	0,99662	1,0
MAPE	0,05	1,66	0,60	0,13	0,13	0,10	26,75	2,11	0,89
MSE	0,01601	0,00524	0,00004	0,00014	0,00002	0,08183	0,00077	0,00016	119695426
RMSE	0,12654	0,07236	0,00643	0,01175	0,00466	0,28606	0,02775	0,01254	10940,5

Avaliando os resultados da Tabela 21, observa-se que a correlação entre os resultados das duas ferramentas é alta, apresentando uma variação mínima para cada modelo (OD, DBO, N, P e E-coli). A diferença nos valores, ainda que pequena, pode ser justificada pelo cálculo anterior da altitude. Embora tenha sido feita uma tentativa de reproduzir com exatidão os dados apresentados no exemplo do livro, os valores de velocidade e nível d'água obtidos apresentaram pequenas diferenças, como pode ser

observado na Figura 55. Nela, a linha laranja representa os dados do QUAL-UFMG, enquanto a azul-escuro representa os do QUALITOOL 2.0.

Figura 55. Resultados da profundidade líquida em QUAL-UFMG e QUALITOOL 2.0.



Fonte: Autora (2025).

5.3. Calibração com séries temporais

Embora os modelos de qualidade da água sejam amplamente utilizados como ferramenta de apoio à gestão de corpos hídricos, a incerteza inerente ao processo de modelagem ainda representa um desafio significativo. A calibração dos parâmetros é um aspecto fundamental na aplicação desses modelos, pois a escolha adequada de seus coeficientes cinéticos influencia diretamente a precisão dos resultados das simulações (Khodabandeh *et al.*, 2021).

A eficácia da calibração de um modelo baseado em dados observados depende diretamente da qualidade das informações disponíveis, sendo essencial minimizar erros observacionais, como imprecisões nas medições da qualidade da água. Além disso, os modelos de qualidade da água consideram diversos parâmetros externos que, por razões de simplificação, são representados de forma aproximada nos cálculos, como os processos de erosão do solo, as reações químicas e o transporte de poluentes em bacias hidrográficas, refletindo, assim, as lacunas existentes no conhecimento atual (Han; Zheng, 2016).

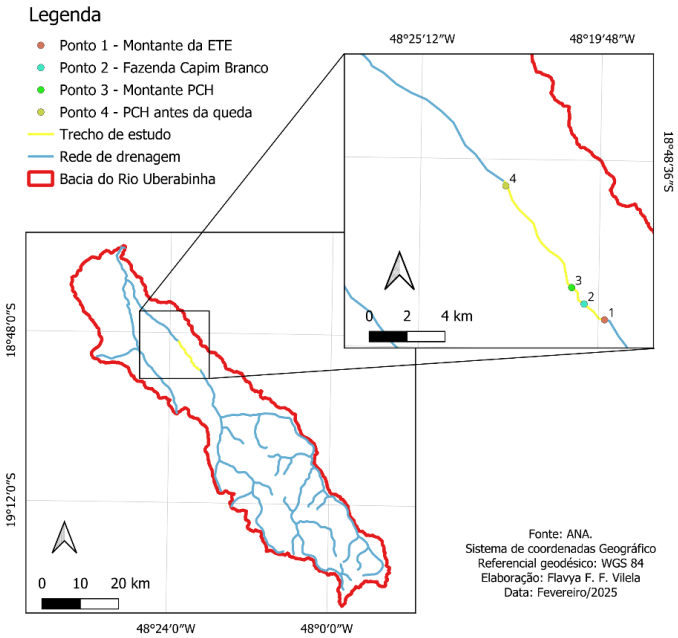
Outro fator crítico é a disponibilidade de dados, uma vez que informações sobre cargas pontuais e, sobretudo, cargas difusas frequentemente apresentam alta imprecisão

devido à ausência de registros com a resolução temporal e espacial adequada. A combinação dessas incertezas resulta em erros que se tornam evidentes quando as saídas do modelo são comparadas aos dados observacionais.

Diante desse cenário, é fundamental que o processo de calibração de um modelo seja embasado em um conhecimento aprofundado sobre o corpo hídrico em estudo. Assim, o objetivo do exemplo a seguir não é obter uma conclusão definitiva sobre a calibração do rio analisado, mas, sim, discutir aspectos relevantes do processo de calibração utilizando o QUALITOOL 2.0.

O exemplo apresentado corresponde a um estudo de caso realizado por Salla *et al.* (2016), no qual os autores analisaram um trecho do rio Uberabinha, afluente do rio Araguari, localizado na região do Triângulo Mineiro, no estado de Minas Gerais (Figura 56). O estudo inclui um levantamento de dados monitorados em seis pontos, além de informações sobre a contribuição de uma estação de tratamento de esgoto (ETE – ponto 2) da região. Foram monitoradas mensalmente, de junho a outubro de 2015, as concentrações de oxigênio dissolvido (OD), demanda bioquímica de oxigênio (DBO), amônia, nitrito, nitrato, fósforo total e metais pesados (cádmio, cromo, cobre, chumbo e zinco). No entanto, para esta discussão, serão considerados apenas os pontos situados antes da Pequena Central Hidrelétrica, especificamente os pontos 1 a 4, e apenas os dados referentes a OD e DBO.

Figura 56. Bacia hidrográfica do rio Uberabinha e pontos de monitoramento.



Fonte: Adaptada de Salla *et al.* (2016).

Os dados referentes à declividade longitudinal do fundo foram obtidos por meio de ferramentas de Sistemas de Informações Geográficas (SIG) e posteriormente generalizados em trechos de 100 metros, conforme apresentado nas Tabelas 22 e 23.

Tabela 22. Comprimento total de cada trecho do exemplo 3.

Trecho	Comprimento (km)
1 – 2	0,9
2 – 3	1,7
3 – 4	9,4

Tabela 23. Dados espaciais do exemplo 3.

Comprimento (m)	Altitude (m)	Comprimento (m)	Altitude (m)	Comprimento (m)	Altitude (m)
0	720,000	3200	695,871	6400	680,850
100	716,323	3300	695,402	6500	680,381
200	712,646	3400	694,932	6600	679,911
300	708,970	3500	694,463	6700	679,442
400	705,293	3600	693,994	6800	678,972
500	705,195	3700	693,524	6900	678,503
600	705,098	3800	693,055	7000	678,034
700	705,001	3900	692,585	7100	677,564
800	704,904	4000	692,116	7200	677,095
900	704,807	4100	691,646	7300	676,625
1000	704,709	4200	691,177	7400	676,156
1100	704,612	4300	690,708	7500	676,040
1200	704,515	4400	690,238	7600	675,924
1300	704,418	4500	689,769	7700	675,809
1400	704,320	4600	689,299	7800	675,693
1500	703,851	4700	688,830	7900	675,577
1600	703,382	4800	688,361	8000	675,461
1700	702,912	4900	687,891	8100	675,345
1800	702,443	5000	687,422	8200	675,230
1900	701,973	5100	686,952	8300	675,114
2000	701,504	5200	686,483	8400	674,998
2100	701,035	5300	686,014	8500	674,882
2200	700,565	5400	685,544	8600	674,766
2300	700,096	5500	685,075	8700	674,651
2400	699,626	5600	684,605	8800	674,535
2500	699,157	5700	684,136	8900	674,419
2600	698,688	5800	683,667	9000	674,303
2700	698,218	5900	683,197	9100	674,187
2800	697,749	6000	682,728	9200	674,104
2900	697,279	6100	682,258	9300	674,021
3000	696,810	6200	681,789	9400	673,938
3100	696,341	6300	681,320	-	-

Neste exemplo, considerou-se uma seção trapezoidal ao longo de todo o trecho, com a largura da base do canal variando entre 23 e 24 metros (do ponto 1 ao ponto 4), inclinação dos taludes fixa em 45° e coeficiente de rugosidade de Manning estimado em 0,046 $m^{-1/3}$.s.

No estudo de caso de Salla *et al.* (2016), a modelagem e calibração foram realizadas utilizando a ferramenta AQUATOOL (módulo GESCAL). Após o processo de calibração por tentativa e erro, os autores concluíram que foram obtidos bons ajustes entre os dados simulados e medidos para todos os parâmetros analisados, com o coeficiente de *Nash-Sutcliffe* variando de 0,91 a 0,99 para a concentração de OD e de 0,96 a 0,99 para a DBO, ao longo dos pontos 3, 4, 5 e 6.

Com base nesses resultados, e considerando que houve perda de dados nos pontos 3 e 4 devido a erros sistemáticos, como um incidente que resultou no comprometimento do recipiente, foram utilizados, como concentração de referência para a calibração no QUALITOOL 2.0, os dados gerados durante a calibração no estudo de Salla *et al.* (2016), apresentados na Tabela 24.

Tabela 24. Concentrações de entrada (exemplo 3).

Mês	**Mês**	**Q (m³/s)**	**OD (mg/L)**	**DBO (mg/L)**
ETE	Junho	2,099	0,0	108,0
	Julho	2,040	0,0	116,0
	Agosto	2,205	0,0	125,0
	Setembro	2,078	0,0	98,0
	Novembro	2,143	0,0	174,0
	Dezembro	2,272	0,0	172,0
1	Junho	4,170	6,24	3,0
	Julho	8,040	5,74	13,0
	Agosto	6,650	5,4	5,0
	Setembro	6,090	3,29	6,0
	Novembro	4,740	3,61	34,0
	Dezembro	10,60	5,57	13,0
3	Junho	-	4,62	29,58
	Julho	-	4,92	33,57
	Agosto	-	3,32	34,59
	Setembro	-	2,57	29,03
	Novembro	-	1,42	88,14
	Dezembro	-	4,48	41,79
4	Junho	-	3,30	10,76
	Julho	-	3,17	32,46
	Agosto	-	1,97	33,39
	Setembro	-	1,84	27,52
	Novembro	-	1,10	130,03
	Dezembro	-	3,38	44,13

Apresentados os dados de entrada do estudo de caso, discute-se agora alguns pontos importantes antes da apresentação dos resultados gerados no QUALITOOL 2.0. Primeiramente, o número de observações distribuídas tanto espacial quanto temporalmente tem impacto direto na qualidade da calibração dos coeficientes no trecho

de estudo. É essencial dispor de dados suficientes para a etapa de calibração, bem como para as etapas subsequentes de verificação dos resíduos e validação do modelo.

O segundo ponto importante refere-se ao fato de que cada coeficiente cinético tem seu próprio fator de influência no meio. Por exemplo, o coeficiente de reaeração k_2 é diretamente afetado pela turbulência na superfície (Von Sperling, 2014b). Assim, para determinar o ponto de calibração do k_2, é necessário considerar parâmetros como a declividade do leito. Esse aspecto foi considerado no estudo de Salla *et al.* (2016), que selecionou pontos de monitoramento antes da queda d'água da PCH (ponto 4) e logo após (ponto 5) para calibrar o valor nesse trecho. Vale destacar que o trecho entre os pontos 4 e 6 não foi considerado para calibração no QUALITOOL 2.0 devido à queda d'água da PCH, uma vez que o modelo de OD no QUALITOOL 2.0 não contempla o incremento de OD em cascata. Este é um ponto relevante a ser considerado para futuras melhorias na ferramenta.

O terceiro ponto a ser considerado diz respeito ao funcionamento do algoritmo PSO. Semelhante a outros algoritmos evolutivos, o PSO inicia com a criação de uma população aleatória de indivíduos, representados por um grupo de partículas. Esse efeito aleatório é vantajoso, pois permite uma maior busca espacial; no entanto, os dados aleatórios iniciais podem ser agrupados em regiões distantes do ótimo. Por isso, é recomendado iniciar a busca com o PSO pelo menos duas vezes, a fim de reduzir a probabilidade de o enxame iniciar em uma região subótima. O uso do PSO é especialmente vantajoso em problemas que apresentam múltiplos mínimos locais, pois o sistema de busca de cada partícula é influenciado tanto individualmente quanto globalmente, o que aumenta a capacidade de busca e reduz a probabilidade de o algoritmo ficar preso em mínimos locais (Wang *et al.*, 2017). No entanto, é importante destacar que os métodos de otimização não garantem a obtenção de um mínimo global.

Após destacar alguns pontos importantes, a sequência retorna ao exemplo em questão. Na Tabela 25, são apresentados os intervalos obtidos na calibração entre os pontos 3 e 6 no AQUATOOL (Intervalo 1) e os intervalos utilizados na calibração no QUALITOOL 2.0 (Intervalo 2).

Tabela 25. Intervalos dos coeficientes do exemplo 3.

Símbolo	Unidades	Intervalo 1	Intervalo 2
k_2	1/dia	0,01 – 62,6	0,05 – 150,0
k_1	1/dia	-	0,08 – 0,45
k_d	1/dia	0,21 – 0,8	0,08 – 1,00
k_s	1/dia	0,0 – 8,0	0,05 – 10,0
l_{rd}	gL/m.d	0,0 – 1036,4	0,05 – 1050,00
S_d	gC/m.d	2,77 – 84,96	0,05 – 100,00

Para a calibração no QUALITOOL 2.0, foram estabelecidos dois trechos para a alternância dos coeficientes, sendo o primeiro entre 0 e 1,8 km (IC1) e o segundo entre 1,8 e 9,4 km (IC2). Além disso, foi aplicado o agrupamento de dados temporais, considerando o primeiro conjunto de junho a agosto (DT1) e o segundo de setembro a outubro (DT2).

Da Tela 01 para a Tela 03 (Figura 10), etapa na qual ocorre a discretização e a modelagem hidrodinâmica, o tempo médio de processamento foi de 4,3 segundos, devido à simplicidade e à reduzida extensão do trecho analisado. Os resultados da análise de sensibilidade (Tela 03) são apresentados na Tabela 26, onde foi realizada uma variação dos coeficientes em 10%, 50% e 100%. No entanto, para aproximar os resultados da calibração entre os dois aplicativos, apenas a temperatura foi fixada para 23,33 °C.

Tabela 26. Análise de sensibilidade dos coeficientes.

Grupo	Var. coef.	OD (%)						DBO (%)					
		k_1	k_2	k_d	k_s	l_{rd}	S_d	k_1	k_2	k_d	k_s	l_{rd}	S_d
DT1	± 10%	0,1	1,2	0,2	0,2	0,0	0,8	0,0	0,0	1,0	9,6	0,2	0,0
	± 50%	1,0	11,2	0,9	1,0	0,0	4,0	0,0	0,0	5,1	57,0	1,0	0,0
	± 100%	100,0	100,0	2,0	2,6	0,0	8,1	528,1	89,8	9,2	128,6	2,0	0,0
DT2	± 10%	0,1	2,2	0,2	0,2	0,1	1,5	0,0	0,0	0,8	7,3	2,8	0,0
	± 50%	1,2	21,0	1,2	1,0	0,3	7,7	0,0	0,0	4,0	43,0	14,1	0,0
	± 100%	100,0	100,0	2,5	2,3	0,6	15,4	529,8	65,9	8,2	106,3	28,2	0,0

Ao avaliar a Tabela 26 de forma simplificada e comparar os dados de DT1 (período estiagem) e DT2 (período de chuvoso), observa-se um aumento na sensibilidade do coeficiente l_{rd}, que refere à contribuição difusa superficial de matéria orgânica, e do S_d, que se refere à demanda bentônica de oxigênio dissolvido (OD), no DT2.

A Tabela 27 apresenta os resultados da calibração realizada pelo QUALITOOL 2.0, na qual foram fixados o tamanho do enxame e o número de iterações em 100. No

critério de tempo, a calibração levou 19,06 segundos para executar 11.100 simulações (3 dias * 100 de tamanho do enxame * 36 iterações + 300 da geração inicial).

Observa-se também que o tempo inicial da primeira busca do IC1 é sempre menor. Isso ocorre devido ao funcionamento do Streamlit, que reprocessa automaticamente a página sempre que o usuário clica em determinados botões de navegação, como "Próximo trecho" ou "Reiniciar busca". Embora os resultados desse reprocessamento não sejam visíveis ao usuário, o QUALITOOL 2.0 foi programado para armazenar os dados, garantindo que a busca não seja perdida. Ao final da calibração, todas as buscas são avaliadas, e os valores de melhor aptidão são retornados. Dessa forma, exceto pelo primeiro tempo do IC1, todos os demais tempos correspondem à soma das iterações iniciadas duas vezes.

Tabela 27. Resultados da calibração do exemplo 3.

Grupo	Conj.	N° de iteração	Tempo (s)	OD		DBO	
				NS	MAPE	NS	MAPE
DT1	IC1	36/100	19,06	0,45214	12,13%	-0,84152	8,88%
		51/100	44,02	0,47575	12,13%	-0,84152	8,87%
		80/100	56,91	0,46206	12,13%	-0,84152	8,88%
	IC2	71/100	77,35	0,17886	19,46%	-0,02685	59,14%
		41/100	89,51	0,14673	19,56%	-0,02186	59,95%
		15/100	50,83	-0,29594	19,57%	-0,10585	59,96%
DT2	IC1	31/100	15,35	0,94026	4,59%	0,96403	6,84%
		37/100	43,38	0,98422	4,63%	0,96139	6,86%
		36/100	39,38	0,98423	4,61%	0,96139	8,85%
	IC2	44/100	43,89	0,75648	6,74%	0,72059	32,34%
		57/100	71,83	0,98846	6,65%	0,72059	32,34%
		22/100	34,59	0,82992	6,69%	0,55461	32,34%

Ao avaliar a Tabela 27 com base exclusivamente nos valores do coeficiente de *Nash-Sutcliffe* (NS), conclui-se que apenas no período DT2 do conjunto IC1 foram alcançados bons resultados, valores próximos a 1. No entanto, ao considerar a média absoluta do erro percentual (MAPE), observa-se uma mudança na percepção da precisão dos dados de referência em relação aos resultados obtidos. Por exemplo, a MAPE da concentração de oxigênio dissolvido (OD) variou de 4,59 a 19,57%, entre as variações espaciais (ponto 3 e 4) e temporais (períodos seco e chuvoso).

Contudo, o objetivo desta etapa não é obter um resultado definitivo, pois, como mencionado anteriormente, a calibração de uma rede de drenagem requer um estudo detalhado. Neste momento, discute-se apenas a metodologia aplicada à calibração. Assim, sem emitir juízo sobre a qualidade estatística dos resultados, pode-se abordar a quantidade

de dados utilizada na calibração de cada conjunto, que consistiu em um ponto monitorado com uma série diária de três dias para cada conjunto de coeficientes cinéticos.

Não há um número mínimo de dados de referência estabelecido para a calibração, mas estudos discutem a estabilidade do coeficiente de *Nash-Sutcliffe* em diferentes tamanhos de amostras e sugerem que um número reduzido de pontos pode resultar em valores pouco representativos do desempenho do modelo (Krause *et al.,* 2005; Moriasi *et al.*, 2015).

Além disso, é fundamental evitar ajuste excessivo durante o período de calibração, pois isso pode resultar em estimativas tendenciosas quando o modelo for aplicado a outros períodos ou condições ambientais (Griensven; Meixner, 2006). Outro desafio relevante é a decomposição da incerteza associada aos resíduos em seus diferentes componentes, tornando a análise de incerteza uma tarefa complexa. De maneira geral, este estudo busca destacar a influência dos erros de entrada e da estrutura do modelo na avaliação da incerteza da modelagem. Além disso, a interpretação desses resultados deve considerar os objetivos e desafios específicos da gestão da qualidade da água.

A calibração no QUALITOOL 2.0 foi realizada por meio da estimativa em blocos/conjuntos, onde os coeficientes permaneceram constantes dentro de cada intervalo, buscando-se a minimização da função objetivo, denominada aptidão, conforme descrito no subcapítulo 3.5.2. No entanto, a aplicação desse procedimento exige o atendimento a determinados critérios estatísticos relacionados à distribuição dos erros, que devem apresentar distribuição normal, média zero, variância constante e independência entre si.

A Tabela 28 apresenta os valores finais de cada coeficiente cinético. Observa-se que alguns desses valores encontram-se nos limites dos intervalos estabelecidos, o que exige uma análise detalhada para cada coeficiente cinético, a fim de determinar se é necessário ampliar ou reduzir o intervalo ou, alternativamente, adotar um valor tabelado recomendado na literatura.

Tabela 28. Valores dos coeficientes cinéticos do exemplo 3.

Grupo	Conj.	k_1 (1/dia)	k_2 (1/dia)	k_d (1/dia)	k_s (1/dia)	l_{rd} (gL5/m.d)	S_d (gC/m.d)
DT1	IC1	0,45	0,90	0,08	2,66	0,05	0,05
	IC2	0,08	14,67	1,00	1,06	0,05	0,05
DT2	IC1	0,08	4,28	0,22	0,05	1050	0,05
	IC2	0,45	12,12	0,08	0,05	1050	32,18

Por fim, neste exemplo, não foram modelados o nitrogênio, fósforo e coliformes. No entanto, é importante ressaltar que a função de minimização da aptidão combina o coeficiente de *Nash-Sutcliffe* com a raiz do erro quadrático médio (RMS). Assim, recomenda-se calibrar cada modelo separadamente, exceto nos casos em que há interdependência entre os modelos, como ocorre com o OD, que depende dos modelos de DBO e nitrogênio. No caso do nitrogênio, essa dependência só se manifesta quando o OD é modelado em conjunto, pois o OD depende do nitrogênio, mas o inverso não ocorre.

6. CONCLUSÃO

6.1. Conclusões gerais

O presente trabalho abordou o desenvolvimento e a aplicação do QUALITOOL 2.0, um aplicativo voltado para a simulação da qualidade da água em ambientes lóticos, acoplado a um sistema de calibração baseado em algoritmos evolucionários. O QUALITOOL 2.0 se destaca por sua interface intuitiva, disponibilizada em formato web, aliada a manuais didáticos que facilitam sua utilização. Além disso, sua alta performance computacional permite obter resultados de modelagem precisos, comparáveis ao modelo QUAL-UFMG, que é uma referência na área de modelagem da qualidade da água.

A integração de diferentes abordagens, incluindo tempo de processamento otimizado, análise de sensibilidade, dados distribuídos espacial e temporalmente, bem como a combinação de métodos empíricos (equação de Manning), heurísticos (algoritmo PSO) e determinísticos (NS e RMSE), reforça o potencial do QUALITOOL 2.0 como uma ferramenta eficaz para calibração de coeficientes em modelos de qualidade da água.

Um dos desafios identificados neste estudo refere-se à dificuldade na modelagem e calibração de modelos de qualidade da água devido à escassez de informações detalhadas sobre a quantidade ou concentração do material lançado, especialmente no caso de entradas difusas. Dessa forma, recomenda-se cautela na interpretação dos resultados gerados pelo aplicativo, visto que, independentemente da confiabilidade dos dados de entrada, um resultado será obtido. Para aumentar a precisão e a confiabilidade das simulações, é aconselhável ampliar a base de dados de referência utilizada na calibração e validação do QUALITOOL 2.0, permitindo sua aplicação em diferentes condições hidrológicas e geográficas. Os exemplos apresentados neste trabalho tiveram como objetivo demonstrar a aplicação do QUALITOOL 2.0, mas é essencial testar sua eficácia em diferentes ambientes lóticos, garantindo sua robustez e aplicabilidade em distintos cenários.

Um diferencial significativo do QUALITOOL 2.0 é sua implementação em linguagem Python, uma linguagem de alto nível que tem se tornado cada vez mais popular entre profissionais e pesquisadores não especializados em programação. Sua simplicidade e a vasta quantidade de materiais de suporte incentivam pesquisadores a desenvolver

variantes do QUALITOOL 2.0 diretamente a partir de seu código-fonte. Para facilitar esse processo, a arquitetura do código foi estruturada com orientação a objetos e com mínimas dependências fixas. Por exemplo, caso seja necessário alterar equações e critérios de cálculo do coeficiente de reação (k2), basta modificar diretamente a função "k2" dentro do arquivo "equacoes.py". O código foi projetado para ser o mais intuitivo possível.

Em comparação com sua versão anterior, o QUALITOOL 2.0 superou a limitação da quantidade de tributários modeláveis, facilitando a integração do rio principal com seus afluentes e permitindo uma representação mais abrangente da rede de drenagem da bacia hidrográfica. No entanto, algumas limitações permanecem, como a adoção da equação de Manning para a quantificação da profundidade líquida, que considera o escoamento permanente e uniforme para cada trecho discretizado. Contudo, a metodologia adotada para obtenção da declividade longitudinal de fundo, por meio de dados de SIG, minimiza essa limitação. A integração opcional do QUALITOOL 2.0 com um SIG também se destaca como um fator relevante, pois a precisão dos dados topográficos de entrada influencia diretamente na qualidade dos resultados obtidos.

O QUALITOOL 2.0 demonstra ser uma ferramenta robusta para a modelagem da qualidade da água, trazendo benefícios como maior precisão e eficiência no processo de simulação. Além disso, sua usabilidade e flexibilidade contribuem para a sua aplicação em diferentes cenários de gestão e monitoramento ambiental.

6.2. Trabalhos futuros

O QUALITOOL 2.0, assim como todo modelo matemático que visa representar o funcionamento do mundo real, possui limitações que podem gerar discrepâncias entre os fenômenos naturais e os resultados simulados. Dessa forma, para validar e consolidar a eficácia do QUALITOOL 2.0, é recomendada a realização de estudos aplicados a situações reais, o que possibilitaria a identificação de eventuais necessidades de ajustes e limitações.

Para aprimorar a modelagem, é relevante simular um ambiente o mais próximo possível da realidade, pois isso aponta direções para futuras melhorias e inovações na ferramenta. Estudos futuros poderiam focar na implementação de estratégias para a integração de modelos bidimensionais, aplicáveis a ambientes lênticos

(KHODABANDEH *et al.*, 2021; KIM *et al.*, 2023), ou até mesmo na complexificação dos modelos utilizados, incorporando relações das algas nos modelos disponíveis (KIM *et al.*, 2023; PRADIPTA; HENDRIYANTO, 2024), ou, ainda, na implementação de novos modelos, como a modelagem de metais pesados (SALLA *et al.*, 2016).

Além disso, considera-se a possibilidade de explorar variações do algoritmo PSO, uma vez que foi implementado o seu método clássico. No entanto, muitos avanços têm sido feitos, como a inclusão de taxa de inflação, pressão de seleção do melhor global, pressão de seleção de exclusão, mutação, entre outros parâmetros, para otimizar a busca (XUE *et al.*, 2020; TAGHIZADEH; KHANI; RAJAEE, 2021).

Por fim, o QUALITOOL 2.0 foi desenvolvido na plataforma web utilizando os recursos gratuitos do Streamlit, que apresentou bom desempenho de processamento dentro dos exemplos aplicados neste estudo. No entanto, o Streamlit oferece uma versão paga com maior capacidade de processamento e menos restrições no uso de CPU/memória, o que resulta em aplicações mais rápidas e suporte a um número maior de usuários simultâneos, além de outros benefícios. Portanto, recomenda-se o desenvolvimento do QUALITOOL 2.0 no formato desktop, preferencialmente utilizando bibliotecas Python como o Tkinter, que permite aproveitar grande parte do código-fonte já disponível. Outra alternativa seria o desenvolvimento do QUALITOOL 2.0 como um plugin para o QGIS®, uma ferramenta SIG que permite o desenvolvimento de plugins em linguagem Python, possibilitando o reaproveitamento do código existente. Ambas as opções são gratuitas e confeririam maior estabilidade ao aplicativo QUALITOOL 2.0.

REFERÊNCIAS

AHMAD, Z. **Prediction of longitudinal dispersion coefficient using laboratory and field data: relationship comparisons**. Hydrology Research, v. 44, n. 2, p. 362-376, 2013. DOI: https://doi.org/10.2166/nh.2012.047.

ALAMY FILHO, J.E. *et al.* **Aplicação da ferramenta computacional QUALI-TOOL na avaliação da qualidade da água em ambiente lótico**. Revista DAE, v. 67, n. 215, 2019. DOI: https://doi.org/10.4322/dae.2019.006.

AL-DALIMY, S.Z., AL-ZUBAIDI, H.A.M. **Application of QUAL2K Model for Simulating Water Quality in Hilla River, Iraq**. Journal of Ecological Engineering, v. 24, n. 6, p. 272-280, 2023. DOI: https://doi.org/10.12911/22998993/162873.

ALDREES, A. *et al.* **Multi-Expression Programming (MEP): Water Quality Assessment Using Water Quality Indices**. Journal Water, v. 14, p. 947, 2022. DOI: https://doi.org/10.3390/w14060947.

ARIAS-AGUILA, E., VELA-CARDICH, R., RAMOS-FERNANDEZ, L. **Desarrollo y aplicación del modelamiento de calidad del agua con GESCAL-AQUATOOL en el río Lurín-Lima-Perú**. Tecnología y ciencias del agua, v. 15, n. 3, p. 250-288, 2024. DOI: https://doi.org/10.24850/j-tyca-2024-03-06.

BONGANHA, C. A. *et al.* **Conceitos e fundamentos da modelagem matemática para gerenciamento de recursos hídricos subterrâneos**. Revista Analytica, v. 30, p. 116-120, 2007.

BRUM, M., FAN, F.M., SALLA, M.R., VON SPERLING, M. **Analysis of a probabilistic approach for modelling and assessment of the water quality of rivers**. Journal of Hydroinformatics, v. 24, e. 4, p. 783-797, 2022. DOI: https://doi.org/10.2166/hydro.2022.157.

CHAPRA, S.C.; PELLETIER, G.J.; TAO, H. **QUAL2K: A Modeling Framework for Simulating River and Stream Water Quality – Version 2.12**. Civil and Environmental Engineering Department, Tufts University, 2022. Disponível em: https://www.qual2k.com.

COSTA, E.S. *et al.* **Otimização do aproveitamento hídrico superficial no alto curso do rio Uberaba, Triângulo Mineiro**. Sociedade e Natureza, v. 31, 2019. DOI: https://doi.org/10.14393/SN-v31-2019-41033.

EJIGU, M.T. **Overview of water quality modeling**. Cogent Engineering, v. 8, n. 1, 2021. DOI: https://doi.org/10.1080/23311916.2021.1891711.

EBERHART, R., KENNEDY, J. **A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micro Machine and Human Science**, p. 39-43, 1995. DOI: https://doi.org/10.1109/MHS.1995.494215.

FAN, F. M., COLLISCHONN, W., RIGO, D. **Modelo analítico de qualidade da água acoplado com Sistema de Informação Geográfica para simulação de lançamentos com duração variada**. Engenharia Sanitária e Ambiental, v. 18, n. 4, p. 359–370, 2013.

FORMIGA, K.T.M. *et al.* **Calibração do Storm Water Management Model (SWMM) utilizando algoritmos evolucionários multiobjetivo**. Engenharia Sanitária e Ambiental, v. 21, n. 4, 2016. DOI: https://doi.org/10.1590/S1413-41522016131862.

GOMIDES, C.E., MATOS, M.P. de, FIA, R., FONSECA, A. R. **Estimativa da qualidade das águas das vertentes do Rio Grande – Brasil em diferentes cenários: modelagem das variáveis DBO e OD utilizando o Qual-UFMG**. Revista de Geociências do Nordeste, v. 10, n. 2, p. 1-18, 2024. DOI: https://doi.org/10.21680/2447-3359.2024v10n2ID35630.

GRIENSVEN, A.V., MEIXNER, T. **Methods to quantify and identify the sources of uncertainty for river basin water quality models**. Water Science and Technology, v. 53, e. 1, p. 51-59, 2006. DOI: https://doi.org/10.2166/wst.2006.007.

HAN, F.; ZHENG, Y. **Multiple-response Bayesian calibration of watershed water quality models with significant input and model structure errors**. Advances in Water Resources, v. 88, p. 109-123, 2016. DOI: https://doi.org/10.1016/j.advwatres.2015.12.007.

HOSSAIN, I., BAPPY, A., SATHI, M.A. **Water quality modelling and assessment of the Buriganga river using QUAL2K**. Global Mainstream Journal of Innovation, Engineering & Emerging Technology, v. 2, n. 3, p. 1-11, 2023. DOI: https://doi.org/10.62304/jieet.v2i03.64.

KHODABANDEH, F. *et al.* **Reservoir quality management with CE-QUAL-W2/ANN surrogate model and PSO algorithm (case study: Pishin Dam, Iran)**. Arabian Journal of Geosciences, v. 14, n. 5, 2021. DOI: https://doi.org/10.1007/s12517-021-06735-x.

KIM, S. *et al.* **Impact assessment of water-level management on water quality in an estuary reservoir using a watershed-reservoir linkage model**. Agricultural Water Management, Journal Elsevier, v. 280, e. 108234, 2023. DOI: https://doi.org/10.1016/j.agwat.2023.108234.

KOMARUDINA, M., HARIYADIB, S., KURNIAWAN, B. **Analisis daya tampung beban pencemarsungai Pesanggrahan (segmen kota Depok) dengan menggunakan model numerik dan spasial**. Jurnal Pengelolaan Sumberdaya Alam dan Lingkungan, v. 5, n. 2, p. 121-132, 2015. DOI: https://doi.org/10.29244/jpsl.5.2.121.

KRAUSE, P., BOYLE, D.P., BÄSE, F. **Comparison of different efficiency criteria for hydrological model assessment**. Advances in Geosciences, v. 5, p. 89-97, 2005. DOI: https://doi.org/10.5194/adgeo-5-89-2005.

LASHANI, M. *et al.* **Simulation of quality and pollution of Dez river using QUAL2Kw model**. Advanced Technologies in Water Efficiency, v. 4, n. 2, p. 1-21, 2024. DOI: https://doi.org/10.22126/atwe.2024.10383.1115.

LEE, C. *et al.* **StarGazer: A Hybrid Intelligence Platform for Drug Target Prioritization and Digital Drug Repositioning Using Streamlit**. Frontiers in Genetics, v. 13, 2022. DOI: https://doi.org/10.3389/fgene.2022.868015.

LIU, X. *et al.* **Parameter identification of river water quality models using a genetic algorithm**. Journal Water Science and Technology, v. 69, n. 4, p. 687-693, 2014. DOI: https://doi.org/10.2166/wst.2013.740.

MAGALHÃES, A.A.B. **Elaboração de uma ferramenta computacional de uso livre para simulação da qualidade da água em ambiente lótico - QUALI-TOOL**. Dissertação de Mestrado, Faculdade de Engenharia Civil, Universidade Federal de Uberlândia, p. 180, 2017.

MEKONNEN, Y.A., TENAGASHAWU, D.Y. **Modeling of streamflow and water quality using SWAT model in the Ribb reservoir, Ethiopia**. Environment, Development and Sustainability, 2023. DOI: https://doi.org/10.1007/s10668-023-04213-w.

MORIASI, J.G. *et al.* **Model Evaluation Guidelines for Systematic Quantification of Accuracy in Watershed Simulations**. American Society of Agricultural and Biological Engineers, v. 50, i. 3, 2007. DOI: https://doi.org/10.13031/2013.23153.

NEITSCH, S. L.; ARNOLD, J. G.; KINIRY, J. R.; WILLIAMS, J. R. **Soil and Water Assessment Tool Theoretical Documentation – Version 2012**. Texas Water Resources Institute, Texas A&M University, 2023. Disponível em: https://swat.tamu.edu.

PANI, D.F. *et al.* **Prospects for reducing the contribution of organic load in a water basin with significant urban occupation**. Engenharia Sanitária e Ambiental, v. 29, e. 20230079, 2024. DOI: https://doi.org/10.1590/S1413-415220230079.

PELLETIER, G.J., CHAPRA, S.C., TAO, H. **QUAL2Kw – A framework for modeling water quality in streams and rivers using a genetic algorithm for calibration**. Environmental Modelling & Software, v. 21, n. 3, p. 419-425, 2006. DOI: https://doi.org/10.1016/j.envsoft.2005.07.002.

PORTO, R. M. **Hidráulica Básica**. São Carlos: EESC/USP, e. 18, p. 540, 1998.

PRADIPTA, M.A., HENDRIYANTO, O. **Strategi Penanggulangan Pencemaran Air Sungai Rejo Agung Menggunakan Pemodelan Qual2kw**. Jurnal Serambi Engineering, v. 9, n. 2, 2024.

RINJANI, R.R. *et al.* **A simplified approach to hydraulic model calibration using HEC-RAS for water quality simulation Ciujung river**. IOP Conf. Series: Earth and Environmental Science, sci. 267 012060, 2023. DOI: https://doi.org/10.1088/1755-1315/1267/1/012060.

SAHAY, R.R., DUTTA, S. **Prediction of longitudinal dispersion coefficients in natural rivers using genetic algorithm**. Hydrology Research, v. 40, n. 6, p. 544-552, 2009. DOI: https://doi.org/10.2166/nh.2009.014.

SALLA, M.R. ALAMY FILHO, J.E., PEREIRA, C.R. **Modeling of Aquatic Ecosystem Dynamics in a Run-of-River Reservoir in the Brazil**. Clean Soil Air Water, v. 51, i. 5, 2023. DOI: https://doi.org/10.1002/clen.202200286.

SALLA, M.R. *et al.* **Estudo da autodepuração do rio Jordão, localizado na bacia hidrográfica do rio Dourados**. Engenharia Sanitária e Ambiental, v. 18, n. 2, 2013. DOI: https://doi.org/10.1590/S1413-41522013000200002.

SALLA, M.R. *et al.* **Sistema de Suporte à Decisão em Recursos Hídricos na Bacia Hidrográfica do Rio Uberabinha, Minas Gerais**. Revista Brasileira de Recursos Hídricos, v. 19, n.1, p. 189-204, 2014. DOI: https://doi.org/10.21168/rbrh.v19n1.p189-204.

SALLA, M.R. *et al.* **Sensibilidade de características morfológicas no comportamento de parâmetros de qualidade da água em rio de porte médio**. Science & Engineering Journal, v. 24, n. 2, p. 29-37, 2015. DOI: https://doi.org/10.14393/19834071.2015.32256.

SALLA, M.R. *et al.* **Importance of calibration for mathematical modeling of self-purification of lotic environments**. Acta Limnologica Brasiliensia, v. 28, n. 27, 2016. DOI: https://doi.org/10.1590/S2179-975X5016.

SÁNCHEZ-GUTIÉRREZ, R., GÓMEZ-CASTRO, C. **Acercamiento a los procesos de modelación de la calidad del agua en una subcuenca: Caso del río Virilla, Costa Rica**. Uniciencia, v. 35, n. 1, p. 71-89, 2021. DOI: https://doi.org/10.15359/ru.35-1.5.

SANTOS, G.B.D. *et al*. **Quantitative water balance of surface waters for a transboundary basin in South America**. Environment Development and Sustainability, v. 26, p. 21755-21781, 2024. DOI: https://doi.org/10.1007/s10668-023-03430-7.

SHI, X. *et al.* **Development and validation of a web-based artificial intelligence prediction model to assess massive intraoperative blood loss for metastatic spinal disease using machine learning techniques**. The Spine Journal, Journal Elsevier, v. 24, n. 1, p. 146-160, 2024. DOI: https://doi.org/10.1016/j.spinee.2023.09.001.

TAGHIZADEH, S., KHANI, S., RAJAEE, T. **Hybrid SWMM and Particle Swarm Optimization Model for Urban Runoff Water Quality Control by Using Green Infrastructures (LIDBMPs)**. Urban Forestry & Urban Greening, v. 60, e. 127032, 2021. DOI: https://doi.org/10.1016/j.ufug.2021.127032.

UDDIN, M.G., NASH, S., RAHMAN, A., OLBERT, A.I. **Performance analysis of the water quality index model for predicting water state using machine learning techniques**. Process Safety and Environmental Protection, Journal Elsevier, v. 169, p. 808-828, 2023. DOI: https://doi.org/10.1016/j.psep.2022.11.073.

UNESCO. **The United Nations World Water Development Report 2023: partnerships and cooperation for water**. United Nations Educational, Scientific and Cultural Organization, p. 210, 2023.

VÁZQUEZ, S.R., MOKROV, N.V. **The integration of mathematical models of the dams in GIS**. IOP Conf. Series: Earth and Environmental Science, sci. 1425 012145, 2019. DOI: https://doi.org/10.1088/1742-6596/1425/1/012145.

VON SPERLING, M. **Princípios do Tratamento Biológico de Águas Residuárias: Introdução à Qualidade das Águas e ao Tratamento de Esgotos**. Editora UFMG, v. 1, n. 4, p. 452, 2014a.

VON SPERLING, M. P**rincípios do Tratamento Biológico de Águas Residuárias: Estudos e modelagem da qualidade da água de rios**. Editora UFMG, v. 7, n. 2, p. 592, 2014b.

XUE, F. *et al.* **Parameter Calibration of SWMM Model Based on Optimization Algorithm**. Computer, Materials & Continua, v. 65, n. 3, p. 2189-2199, 2020. DOI: https://doi.org/10.32604/cmc.2020.06513.

WANG, D., TAN, D., LIU, L. **Particle swarm optimization algorithm: an overview**. Soft Computing, v. 22, e. 2, p. 387-408, 2017. DOI: https://doi.org/10.1007/s00500-016-2474-6.

WOOL, T., AMBROSE JR., R.B., MARTIN, J.L., COMER, A. **WASP 8: The Next Generation in the 50-year Evolution of USEPA's Water Quality Model**. Journal Water, v. 12, e. 1398, 2020. DOI: https://doi.org/10.3390/w12051398.

ZELAZNY, M., BRYLA, M., OZGA-ZIELINSKI, B., WALCZYKIEWICZ, T. **Applicability of the WASP Model in an Assessment of the Impact of Anthropogenic Pollution on Water Quality - Dunajec River Case Study**. Journal Sustainability, v. 15, n. 3, 2023. DOI: https://doi.org/10.3390/su15032444.

APÊNDICE A - CÓDIGO-FONTE DO APLICATIVO QUALITOOL 2.0 (~ 5151 linhas)

I. QT_2.py:

```
# biblioteca a serem importadas
import streamlit as st
from PIL import Image

#criar páginas
pg = st.navigation([st.Page("pages/introducao.py",
                            title="QUALITOOL 2.0"),
                    st.Page("pages/instrucoes.py",
                            title="QUALITOOL-Instruções"),
                    st.Page("pages/modelagem.py",
                            title="QUALITOOL-Modelagem"),
                    st.Page("pages/calibragem.py",
                            title="QUALITOOL-Ajuste"),
                    st.Page("pages/graficos.py",
                            title="QUALITOOL-Gráficos"),
                    st.Page("pages/resultados_model.py",
                            title="QUALITOOL-Modelagem-Resultados"),
                    st.Page("pages/resultados_calib.py",
                            title="QUALITOOL-Ajuste-Resultados"),
                    st.Page("pages/resultados_calib_as.py",
                            title="QUALITOOL-Ajuste-Resultados"),
                    st.Page("pages/resultados_calib_pso.py",
                            title="QUALITOOL-Ajuste-Resultados")],
                    position="hidden")

# busca imagem da logo
image = Image.open('imagens/LOGO.png')

# formatação da barra lateral
bar = st.sidebar
bar.image(image)
bar.text(' ')
bar.markdown('''<div style='text-align: center;
                    color: teal;
                    '>MODELAGEM DE QUALIDADE DA ÁGUA </div>''',
                    unsafe_allow_html=True)
bar.text(' ')

bar.page_link("pages/introducao.py", label="APRESENTAÇÃO",
              icon=":material/home:")
bar.page_link("pages/instrucoes.py", label="INSTRUÇÕES",
              icon=":material/data_table:")
bar.page_link("pages/modelagem.py", label="MODELAGEM",
              icon=":material/trending_up:")
bar.page_link("pages/calibragem.py", label="AJUSTE E CALIBRAÇÃO",
              icon=":material/hub:")
bar.page_link("pages/graficos.py", label="GRÁFICOS",
              icon=":material/bar_chart_4_bars:")

# ativa as páginas
pg.run()
```

II. pages/introducao.py:

```
# biblioteca a serem importadas
import streamlit as st
```

```
# formatação da tela/texto
st.markdown('''<h3 style='text-align: center;
            color: teal;
            '>Apresentação </h3>''',
            unsafe_allow_html=True)

st.markdown('''<div style='text-align: justify;
            '><b>QUALITOOL 2.0</b> é uma ferramenta de simulação
            computacional otimizada para o desenvolvimento de
            Sistemas de Apoio à Decisão (SSD) para o planejamento
            de gestão de qualidade de água em ambientes lóticos.
            </div>''',
            unsafe_allow_html=True)
st.write('')
st.markdown('''<div style='text-align: justify;
            '>A versão 2.0 do QUALITOOL foi desenvolvida como
            parte de um projeto de pesquisa de mestrado pela
            <b>Ma. Flavya Fernanda França Vilela[1]</b> e pelo
            professor <b>Dr. Márcio Ricardo Salla[2]</b>,
            ambos vinculados ao Programa de Pós-Graduação em
            Qualidade Ambiental (PPGMQ) da Universidade Federal
            de Uberlândia - UFU.
            O projeto recebeu apoio financeiro da Fundação de
            Amparo à Pesquisa do Estado de Minas (Fapemig).
            </div>''',
            unsafe_allow_html=True)
st.write('')
st.markdown('''<div style='text-align: justify;
            '>Esta plataforma é disponibilizada de forma gratuita
            com o objetivo de contribuir para a comunidade. Os
            autores não assumem responsabilidade por eventuais
            erros no programa, mas ficam gratos pela comunicação
            de qualquer problema identificado.</div>''',
            unsafe_allow_html=True)
st.write('')
st.write('')
st.write('')
st.markdown('''<div style='text-align: justify;
            '>[1] Mestre em Qualidade Ambiental pela Universidade
            Federal de Uberlândia (2025). </div>''',
            unsafe_allow_html=True)
st.markdown('''<div style='text-align: justify;
            '>[2] Doutor em Engenharia Hidráulica e Saneamento pela
            Universidade de São Paulo (2006); Pós-doutorado pelo
            Instituto de Ingeniería del Agua y Medio Ambiente de
            la Universidad Politecnica de Valencia -
            IIAMA/UPV (2013).  </div>''',
            unsafe_allow_html=True)
st.write('')
st.markdown('''Contatos: qualitool.labhidro.ufu@gmail.com |
            flavya2310@gmail.com | marcio.salla@ufu.br
            ''')

st.write("   ")

st.markdown('<h6 style="text-align: right;">Apoio:</h6>',
            unsafe_allow_html=True)

imagem_1 = Image.open('imagens/logos_geral.png')
st.image(imagem_1, output_format="PNG")
```

III. pages/instrucoes.py:

```
# biblioteca a serem importadas
```

```
import streamlit as st
from PIL import Image

# formatação da tela/texto
st.markdown('''<h3 style='text-align: center;
            color: teal;
            '>Instruções e dicas</h3>''',
            unsafe_allow_html=True)

st.write(' ')
st.write(' ')
st.markdown('''<div style='text-align: justify;
            '>Mensagem de texto para instruir o usuário.</div>''',
            unsafe_allow_html=True)

st.write(' ')
st.page_link("https://github.com/QUALITOOL/qualitool_2_0",
             label="Repositório no GITHUB",
             icon=":material/code:")
st.page_link("https://labsanufu.wixsite.com/site",
             label="Visite o site do LABSAN",
             icon=":material/find_in_page:")

st.divider()
st.markdown('''<h5 style='text-align: center;
            color: teal;
            '>Manuais</h5>''',
            unsafe_allow_html=True)

munual_qt1 = "documentos/qt1_manual.pdf"

with open(munual_qt1, "rb") as file:
    pdf_qt1_manual = file.read()

st.write(' ')

col1, col2 = st.columns(2)
col1.download_button(
    label="QT1 - MANUAL DO USUÁRIO",
    data=pdf_qt1_manual,
    file_name="qt1_manual.pdf",
    use_container_width=True
)

st.divider()

st.markdown('''<h5 style='text-align: center;
            color: teal;
            '>Exemplos resolvidos</h5>''',
            unsafe_allow_html=True)

exemplo_qt1 = "documentos/qt1_exemplo_pratico.pdf"

with open(exemplo_qt1, "rb") as file:
    pdf_qt1_ex = file.read()

st.write(' ')
col_1, col_2 = st.columns(2)

col_1.download_button(
    label="QT1 - EXEMPLO PRÁTICO",
    data=pdf_qt1_ex,
    file_name="qt1_exemplo_pratico.pdf",
    use_container_width=True
)

st.divider()
```

IV. pages/modelagem.py:

```
# biblioteca a serem importadas
import streamlit as st
from funcoes.layout_modelagem import inicio, dados_iniciais
from funcoes.layout_modelagem import fun_contrib_retirad
from funcoes.layout_modelagem import coeficientes, salvararquivo
from funcoes.ferramentas import transformacao
import copy

# formatação da tela/texto
st.markdown('#### Modelagem com os'
            + ' :orange[coeficientes predefinidos]:')

paramentro = {'m_od': True, 'm_dbo': True, 'm_n': True,
              'm_p': True, 'm_c': True, 'n_tb': 0, 's_t': True}

lista_modelagem, data, labels, lista_tabs = inicio(paramentro)
n_trib = lista_modelagem['n_tb']

lista_parametros,  list_name,  list_valores,  zona,  hemisferio,  dias  =
dados_iniciais(data, lista_modelagem, n_trib, labels, lista_tabs)

list_name_salvo = copy.deepcopy(list_name)
ponto_af = lista_parametros['p_af']
ordem_desague = lista_parametros['l_des']

lista_contr_retir = fun_contrib_retirad(data, n_trib, labels, lista_tabs,
                                                    list_name,  list_valores,
lista_modelagem, dias)

lista_coeficientes = coeficientes(data, lista_modelagem, n_trib,
                                  labels, lista_tabs, dias)

salvararquivo(lista_modelagem,     lista_parametros,     lista_coeficientes,
lista_contr_retir, list_name_salvo, list_valores)

# botão
botao = st.button(
    'Clique aqui para iniciar a modelagem',
    type='primary')


if botao:

    list_tranfor = transformacao(lista_modelagem, lista_parametros,
                                 lista_coeficientes,
                                 lista_contr_retir, list_name_salvo)


    st.session_state['reslt_model'] = [n_trib, list_tranfor,
                                       ponto_af, lista_modelagem,
                                       ordem_desague, dias, labels,
                                       zona, hemisferio]

    st.switch_page("pages/resultados_model.py")
```

V. pages/resultados_model.py:

```
# biblioteca a serem importadas
import streamlit as st
```

```
import pandas as pd
from funcoes.equacoes import modelagem_Final
from funcoes.ferramentas import plotar
import copy

# formatação da tela
reslt_model = st.session_state.get('reslt_model', "ERRO")

if reslt_model == [] or reslt_model == 'ERRO':
    st.switch_page("pages/modelagem.py")

n_tributarios = reslt_model[0]
list_tranfor = reslt_model[1]
ponto_af = reslt_model[2]
lista_modelagem = reslt_model[3]
ordem_desague = reslt_model[4]
dias = reslt_model[5]
labels = reslt_model[6]
zona = reslt_model[7]
hemisferio = reslt_model[8]

print('aqui2')
st.markdown('''<h3 style='text-align: center;
            color: black;
            '>Resultados </h3>''',
            unsafe_allow_html=True)

if n_tributarios > 0:
    ordem_modelagem = []
    ordem_da = copy.deepcopy(ordem_desague)
    ordem = list(range(1, len(ordem_da) + 1))
    while len(ordem) > 0:
        remov = []
        for ia in range(len(ordem)):
            if (ordem[ia] in ordem_da) == False:
                ordem_modelagem.append(ordem[ia])
                remov.append(ia)
        remov.reverse()
        for irem in remov:
            ordem.pop(irem)
            ordem_da.pop(irem)
    ordem_modelagem.append(0)
else:
    ordem_modelagem = [0]

lista_final, list_entr = modelagem_Final(list_tranfor, ponto_af,
                                         lista_modelagem,
                                         ordem_desague,
                                         ordem_modelagem)

lidt_df = []
for r in range(n_tributarios + 1):
    rio = lista_final[ordem_modelagem.index(r)]

    df = None
    obj_to_dict = {'rio': [],'latitude': [], 'longitude': [],
                   'altitude': [], 'comprimento': [], 'vazao': [],
                   'profundidade': [], 'velocidade': [],
                   'tensao_c': [], 'nivel_dagua': [],
                   'froude': []}
    if lista_modelagem['s_t']:
        dt = {'data': []}
        dt.update(obj_to_dict)
        obj_to_dict = dt
    if lista_modelagem['m_od']:
        obj_to_dict['conc_od'] = []
        obj_to_dict['conc_dbo'] = []
```

```
    if lista_modelagem['m_n']:
        obj_to_dict['conc_no'] = []
        obj_to_dict['conc_n_amon'] = []
        obj_to_dict['conc_nitrito'] = []
        obj_to_dict['conc_nitrato'] = []
    if lista_modelagem['m_p']:
        obj_to_dict['conc_p_org'] = []
        obj_to_dict['conc_p_inorg'] = []
        obj_to_dict['conc_p_total'] = []
    if lista_modelagem['m_c']:
        obj_to_dict['conc_e_coli'] = []


    for idata in (range(len(rio[0].hidraulica.vazao))):
        for i in range(len(rio)):
            h = rio[i].hidraulica
            cc = rio[i].concentracoes

            obj_to_dict['rio'].append(rio[i].rio)
            obj_to_dict['latitude'].append(h.latitude)
            obj_to_dict['longitude'].append(h.longitude)
            obj_to_dict['altitude'].append(h.altitude)
            obj_to_dict['comprimento'].append(h.comprimento)
            obj_to_dict['vazao'].append(h.vazao[idata])
            obj_to_dict['profundidade'].append(h.profundidade[idata])
            obj_to_dict['velocidade'].append(h.velocidade[idata])
            obj_to_dict['tensao_c'].append(h.tensao_c[idata])
            obj_to_dict['nivel_dagua'].append(h.nivel_dagua[idata])
            obj_to_dict['froude'].append(h.froude[idata])

            if lista_modelagem['s_t']:
                obj_to_dict['data'].append(dias[idata])
            if lista_modelagem['m_od']:
                obj_to_dict['conc_od'].append(cc.conc_od[idata])
                obj_to_dict['conc_dbo'].append(cc.conc_dbo[idata])
            if lista_modelagem['m_n']:
                obj_to_dict['conc_no'].append(cc.conc_no[idata])
                obj_to_dict['conc_n_amon'].append(cc.conc_n_amon[idata])
                obj_to_dict['conc_nitrito'].append(cc.conc_nitrito[idata])
                obj_to_dict['conc_nitrato'].append(cc.conc_nitrato[idata])
            if lista_modelagem['m_p']:
                obj_to_dict['conc_p_org'].append(cc.conc_p_org[idata])
                obj_to_dict['conc_p_inorg'].append(cc.conc_p_inorg[idata])
                obj_to_dict['conc_p_total'].append(cc.conc_p_total[idata])
            if lista_modelagem['m_c']:
                obj_to_dict['conc_e_coli'].append(cc.conc_e_coli[idata])

    df = pd.DataFrame(obj_to_dict)
    lidt_df.append(df)

plotar(n_tributarios, lista_modelagem, lidt_df, list_entr, labels,
       zona, hemisferio, dias)

st.session_state['reslt_model'] = []
```

VI. pages/calibragem.py:

```
# biblioteca a serem importadas
import streamlit as st
from funcoes.layout_modelagem import dados_iniciais, fun_contrib_retirad
from    funcoes.layout_calibragem    import    inicio_calib,    dados_reais,
coef_intervalo
from  funcoes.ferramentas  import  transformacao_calib,  ordem_desague_geral,
lista_hidraulica
```

```
import copy
import pandas as pd

st.markdown('#### Modelagem com os'
            + ' :orange[coeficientes calibrados pelo algoritmo PSO
(Particle Swarm Optimization)]:')

paramentro = {'m_od': True, 'm_dbo': True, 'm_n': True, 'm_p': True,
                'm_c': True, 'n_tb': 0, 's_t': True}
lista_modelagem,   data,   labels,   lista_tabs,   lista_par_pos   =
inicio_calib(paramentro)
n_trib = lista_modelagem['n_tb']

lista_parametros, list_name, list_valores, zona, hemisferio, dias =
dados_iniciais(data, lista_modelagem, n_trib,
                                labels, lista_tabs)

list_name_salvo = copy.deepcopy(list_name)
ponto_af = lista_parametros['p_af']
ordem_desague = lista_parametros['l_des']

lista_contr_retir   =   fun_contrib_retirad(data,   n_trib,   labels,
lista_tabs, list_name, list_valores, lista_modelagem, dias)

dados_inter_coef  =  coef_intervalo(lista_modelagem,  n_trib,  labels,
lista_tabs)

lista_dados_reais   =   dados_reais(n_trib,   labels,   lista_tabs,
list_name, list_valores,
                                lista_modelagem, dados_inter_coef)

# botão
botao = st.button(
    'Clique aqui para iniciar a busca',
    type='primary')

if botao:

       list_tranfor_Geral   =   transformacao_calib(lista_modelagem,
lista_parametros, dados_inter_coef,
                                 lista_contr_retir, list_name_salvo,
lista_dados_reais)


     ordem_modelagem,  ordem_final,  list_ordem_coef,  list_ordem_dr,
ordem_final_coef   =   ordem_desague_geral(ordem_desague,   n_trib,
dados_inter_coef,

      lista_dados_reais, list_tranfor_Geral, ponto_af)

       lista_hidraulica_ord  =  lista_hidraulica(list_tranfor_Geral,
ordem_modelagem, ordem_desague, ponto_af)

    dias = pd.to_datetime(dias)

        st.session_state['reslt_calb']   =    {'list_tranfor_Geral':
list_tranfor_Geral, 'ordem_final': ordem_final,
                                        'n_trib': n_trib, 'ponto_af':
ponto_af, 'lista_modelagem': lista_modelagem,
                                    'ordem_desague': ordem_desague,
'dias': dias, 'labels': labels,
```

```
                                                'zona': zona, 'hemisferio':
hemisferio, 'list_ordem_coef': list_ordem_coef,
                                            'list_ordem_dr': list_ordem_dr,
'lista_hidraulica_ord': lista_hidraulica_ord,
                                                                'ordem_modelagem':
ordem_modelagem, 'ordem_final_coef': ordem_final_coef,
                                            'lista_par_pos': lista_par_pos}

    st.switch_page("pages/resultados_calib_as.py")
```

VII. pages/resultados_calib_as.py:

```
# biblioteca a serem importadas
import streamlit as st
import copy
from funcoes.ferramentas import ordem_analise_sensibilidade, ajuste_trecho

reslt_calb = st.session_state.get('reslt_calb', 'ERRO')

if reslt_calb == 'ERRO':
    st.switch_page("pages/calibragem.py")


list_tranfor = copy.deepcopy(reslt_calb['list_tranfor_Geral'])

st.markdown('''<h3 style='text-align: center;
            color: teal;
            '>Análise de sensibilidade dos coeficientes do modelo </h3>''',
            unsafe_allow_html=True)

fixar_coef,     seq_coef     =     ordem_analise_sensibilidade(list_tranfor,
reslt_calb['ponto_af'],
                                            reslt_calb['lista_modelagem'],
reslt_calb['ordem_desague'],
                                        reslt_calb['lista_hidraulica_ord'],
reslt_calb['ordem_modelagem'])

botao_as = st.button('Calibrar esse conjunto', type='primary')

if botao_as:
                  trecho_hidr         =         ajuste_trecho(list_tranfor,
reslt_calb['lista_hidraulica_ord'], reslt_calb['list_ordem_coef'],
                                          0,  0,  reslt_calb['ordem_final'],
reslt_calb['ordem_modelagem'])


          st.session_state['anotacoes']     =     {'marcador_global':     0,
'marcador_global_interno': 0, 'trecho': trecho_hidr,
                                             'fixar': [fixar_coef, seq_coef],
'VarBolean': False, 'transf_final': copy.deepcopy(list_tranfor),
                                                    'lista_coef_gerados': [],
'lista_apt_gerados': []}

    st.switch_page("pages/resultados_calib_pso.py")
```

VIII. pages/resultados_calib_pso.py:

```
# biblioteca a serem importadas
import streamlit as st
import copy
from funcoes.ferramentas import estrutura_calibracao, ajuste_trecho, tabelar
```

```
from funcoes.equacoes import salvando_conc

anotacoes = st.session_state.get('anotacoes', 'ERRO')
reslt_calb = st.session_state.get('reslt_calb', 'ERRO')

if reslt_calb == 'ERRO' or anotacoes == 'ERRO':
    st.switch_page("pages/calibragem.py")

marcador_conj_global = anotacoes['marcador_global']
marcador_conj_interno = anotacoes['marcador_global_interno']
fixar = anotacoes['fixar']
trecho_hidr = anotacoes['trecho']
var_bol = anotacoes['VarBolean']
transf_final = anotacoes['transf_final']
lista_coef_gerados = anotacoes['lista_coef_gerados']
lista_apt_gerados = anotacoes['lista_apt_gerados']

list_tranfor = copy.deepcopy(reslt_calb['list_tranfor_Geral'])

ordem_final = reslt_calb['ordem_final']
n_trib = reslt_calb['n_trib']
ponto_af = reslt_calb['ponto_af']
lista_modelagem = reslt_calb['lista_modelagem']
ordem_desague = reslt_calb['ordem_desague']
dias = reslt_calb['dias']
labels = reslt_calb['labels']
list_ordem_coef = reslt_calb['list_ordem_coef']
list_ordem_dr = reslt_calb['list_ordem_dr']
lista_hidr_model = reslt_calb['lista_hidraulica_ord']
ordem_modelagem = reslt_calb['ordem_modelagem']
ordem_final_coef = reslt_calb['ordem_final_coef']
lista_par_pos = reslt_calb['lista_par_pos']

text_lb = ''
atv = False
if len(ordem_final_coef[marcador_conj_global]) == 0:
    text_lb = labels[ordem_final[marcador_conj_global][-1]]
else:
                                    for                 ord                 in
ordem_final_coef[marcador_conj_global][marcador_conj_interno]:
        if atv:
            text_lb += ' - '
        text_lb += labels[ord]
        atv = True
    text_lb += ' - ' + labels[ordem_final[marcador_conj_global][-1]]
st.markdown('''<h3 style='text-align: center;
            color: teal;
            '>Calibração dos coeficientes por PSO</h3>''',
            unsafe_allow_html=True)
st.markdown('''<h6 style='text-align: center;
            color: gray;
            '>Trecho(s): ''' + str(text_lb) + '. </h6>',
            unsafe_allow_html=True)
st.markdown('''<h5 style='text-align: center;
            color: gray;
            '>Coeficiente do ponto '''
                                                                              +
str(list_ordem_coef[marcador_conj_global][marcador_conj_interno])
            + ' do ' + str(labels[ordem_final[marcador_conj_global][-1]]) +
':</h5>',
            unsafe_allow_html=True)

if var_bol:
    coef_ponto = tabelar(lista_modelagem)
else:
```

```
     coef_ponto, apt_ponto = estrutura_calibracao(list_tranfor, fixar[0],
fixar[1], list_ordem_coef, list_ordem_dr,
                                                  marcador_conj_global,
marcador_conj_interno, ordem_final, lista_par_pos,
                                                ponto_af, lista_modelagem,
lista_hidr_model, ordem_desague, trecho_hidr, dias)
    lista_coef_gerados.append(coef_ponto)
    lista_apt_gerados.append(apt_ponto)

esquerda, centro, direita = st.columns(3)

if var_bol == False:
      botao_rerodar = esquerda.button("Reiniciar busca neste trecho",
use_container_width=True)

    if botao_rerodar:

        st.switch_page("pages/resultados_calib_pso.py")

        botao_tabelar = centro.button("Escolher valores tabelados",
use_container_width=True)

    if botao_tabelar:

                  st.session_state['anotacoes'] = {'marcador_global':
marcador_conj_global, 'marcador_global_interno': marcador_conj_interno,
                                           'trecho': trecho_hidr, 'fixar':
fixar, 'VarBolean': True, 'transf_final': transf_final,
                                                 'lista_coef_gerados':
lista_coef_gerados, 'lista_apt_gerados': lista_apt_gerados}
        st.switch_page("pages/resultados_calib_pso.py")
else:
    voltar = esquerda.button("Voltar para busca", use_container_width=True)

    if voltar:
                  st.session_state['anotacoes'] = {'marcador_global':
marcador_conj_global, 'marcador_global_interno': marcador_conj_interno,
                                           'trecho': trecho_hidr, 'fixar':
fixar, 'VarBolean': False, 'transf_final': transf_final,
                                                 'lista_coef_gerados':
lista_coef_gerados, 'lista_apt_gerados': lista_apt_gerados}
        st.switch_page("pages/resultados_calib_pso.py")
botao_proximo = direita.button('Próximo trecho', type='primary',
use_container_width=True)

if botao_proximo:
# Salvando os valores do coeficiente calibrado ou tabelado
    mudou = False
    if len(lista_coef_gerados) > 0:
                                                    coef_ponto           =
copy.deepcopy(lista_coef_gerados[lista_apt_gerados.index(min(lista_apt_gera
dos))])

    id_rio_calb = ordem_final[marcador_conj_global][-1]
                                           ordem_coef                    =
list_ordem_coef[marcador_conj_global][marcador_conj_interno]
    transf_final[id_rio_calb].lista_e_coeficientes[ordem_coef].coeficientes
= coef_ponto
    if len(ordem_final_coef[marcador_conj_global]) > 0:
                                        for         rio_af_calb         in
ordem_final_coef[marcador_conj_global][marcador_conj_interno]:
            transf_final[rio_af_calb].lista_e_coeficientes[0].coeficientes
= coef_ponto
           transf_final[rio_af_calb].lista_e_coeficientes[0].comprimento =
0.0

    # Salvando a concentração do último ponto calibrado ou tabelado
```

```
          list_tranfor    =    salvando_conc(list_tranfor,    ordem_final,
marcador_conj_global,
                                  trecho_hidr, coef_ponto, lista_modelagem,
lista_hidr_model, ordem_desague, ponto_af)

    if marcador_conj_interno == (len(list_ordem_coef[marcador_conj_global])
- 1):
        if marcador_conj_global == (len(list_ordem_coef) - 1):
            id_rio_calb = ordem_final[marcador_conj_global][-1]
                                                        ordem_coef        =
list_ordem_coef[marcador_conj_global][marcador_conj_interno]
            transf_final[id_rio_calb].lista_e_coeficientes[ordem_coef].coef
icientes = coef_ponto
            st.switch_page("pages/resultados_calib.py")

        else:
            marcador_conj_interno = 0
            marcador_conj_global += 1
            mudou = True

    else:
        marcador_conj_interno += 1
        mudou = True


                    trecho_hidr        =        ajuste_trecho(list_tranfor,
reslt_calb['lista_hidraulica_ord'], reslt_calb['list_ordem_coef'],
                                marcador_conj_global, marcador_conj_interno,
                                 reslt_calb['ordem_final'], ordem_modelagem)

    if mudou:

                    st.session_state['anotacoes']    =    {'marcador_global':
marcador_conj_global, 'marcador_global_interno': marcador_conj_interno,
                                                'trecho': trecho_hidr, 'fixar':
fixar, 'VarBolean': False, 'transf_final': transf_final,
                                                    'lista_coef_gerados': [],
'lista_apt_gerados': []}
    else:

                    st.session_state['anotacoes']    =    {'marcador_global':
marcador_conj_global, 'marcador_global_interno': marcador_conj_interno,
                                                'trecho': trecho_hidr, 'fixar':
fixar, 'VarBolean': False, 'transf_final': transf_final,
                                                        'lista_coef_gerados':
lista_coef_gerados, 'lista_apt_gerados': lista_apt_gerados}

     st.session_state['reslt_calb'] =  {'list_tranfor_Geral': list_tranfor,
'ordem_final': ordem_final,
                                     'n_trib': n_trib, 'ponto_af': ponto_af,
'lista_modelagem': lista_modelagem,
                                 'ordem_desague': ordem_desague, 'dias': dias,
'labels': labels,
                                    'zona': reslt_calb['zona'], 'hemisferio':
reslt_calb['hemisferio'], 'list_ordem_coef': list_ordem_coef,
                                               'list_ordem_dr':  list_ordem_dr,
'lista_hidraulica_ord': lista_hidr_model,
                                          'ordem_modelagem':  ordem_modelagem,
'ordem_final_coef': ordem_final_coef,
                                  'lista_par_pos': lista_par_pos}


    st.switch_page("pages/resultados_calib_pso.py")
```

IX. pages/resultados_calib.py:

```
# biblioteca a serem importadas
import streamlit as st
import pandas as pd
from funcoes.equacoes import modelagem_Final_2
from funcoes.ferramentas import plotar
import copy

reslt_calb = st.session_state.get('reslt_calb', 'ERRO')
anotacoes = st.session_state.get('anotacoes', 'ERRO')
reslt_calb = st.session_state.get('reslt_calb', 'ERRO')
fixar = anotacoes['fixar']
trecho_hidr = anotacoes['trecho']
var_bol = anotacoes['VarBolean']
transf_final = anotacoes['transf_final']

if reslt_calb == 'ERRO':
    st.switch_page("pages/calibragem.py")

st.markdown('''<h3 style='text-align: center;
            color: black;
            '>Resultados </h3>''',
            unsafe_allow_html=True)

container4 = st.container(border=True)
with container4:
    for cj_rio in range(len(reslt_calb['ordem_final_coef'])):
        for cj_cf in range(len(reslt_calb['list_ordem_coef'][cj_rio])):
            text_lb = ''
            atv = False
            if len(reslt_calb['ordem_final_coef'][cj_rio]) == 0:
                                                        text_lb      =
reslt_calb['labels'][reslt_calb['ordem_final'][cj_rio][-1]]
            else:
                for ord in reslt_calb['ordem_final_coef'][cj_rio][cj_cf]:
                    if atv:
                        text_lb += ' - '
                    text_lb += reslt_calb['labels'][ord]
                    atv = True
                                        text_lb    +=    '    -    '    +
reslt_calb['labels'][reslt_calb['ordem_final'][cj_rio][-1]]
            text_lb += ' (a partir do conjuntos de coeficientes do Ponto '
+ str(reslt_calb['list_ordem_coef'][cj_rio][cj_cf]) + '):'
             st.markdown('- Para o(s) trechos(s): :orange[**' + text_lb +
'**]')

                                                          coef_final        =
transf_final[cj_rio].lista_e_coeficientes[cj_cf].coeficientes
            conj_cf_final = {}
            for tx in range(len(fixar[1])):
                        conj_cf_final[fixar[1][tx]]  =  [getattr(coef_final,
fixar[1][tx])]
            st.dataframe(copy.deepcopy(conj_cf_final))

lista_modelagem = reslt_calb['lista_modelagem']
n_tributarios = reslt_calb['n_trib']
dias = reslt_calb['dias']
ordem_modelagem = reslt_calb['ordem_modelagem']
labels = reslt_calb['labels']
zona = reslt_calb['zona']
hemisferio = reslt_calb['hemisferio']
lista_final,       list_entr       =       modelagem_Final_2(transf_final,
reslt_calb['ponto_af'], lista_modelagem,
                                                reslt_calb['ordem_desague'],
ordem_modelagem, reslt_calb['lista_hidraulica_ord'])
```

```
lidt_df = []
for r in range(n_tributarios + 1):
    rio = lista_final[ordem_modelagem.index(r)]
    df = None
     obj_to_dict = {'rio': [],'latitude': [], 'longitude': [], 'altitude':
[], 'comprimento': [], 'vazao': [],
                    'profundidade': [], 'velocidade': [], 'tensao_c': [],
'nivel_dagua': [],
                   'froude': []}
    if lista_modelagem['s_t']:
        dt = {'data': []}
        dt.update(obj_to_dict)
        obj_to_dict = dt
    if lista_modelagem['m_od']:
        obj_to_dict['conc_od'] = []
        obj_to_dict['conc_dbo'] = []
    if lista_modelagem['m_n']:
        obj_to_dict['conc_no'] = []
        obj_to_dict['conc_n_amon'] = []
        obj_to_dict['conc_nitrito'] = []
        obj_to_dict['conc_nitrato'] = []
    if lista_modelagem['m_p']:
        obj_to_dict['conc_p_org'] = []
        obj_to_dict['conc_p_inorg'] = []
        obj_to_dict['conc_p_total'] = []
    if lista_modelagem['m_c']:
        obj_to_dict['conc_e_coli'] = []

    for idata in (range(len(rio[r].hidraulica.vazao))):
        for i in range(len(rio)):
            h = rio[i].hidraulica
            cc = rio[i].concentracoes

            obj_to_dict['rio'].append(rio[i].rio)
            obj_to_dict['latitude'].append(h.latitude)
            obj_to_dict['longitude'].append(h.longitude)
            obj_to_dict['altitude'].append(h.altitude)
            obj_to_dict['comprimento'].append(h.comprimento)
            obj_to_dict['vazao'].append(h.vazao[idata])
            obj_to_dict['profundidade'].append(h.profundidade[idata])
            obj_to_dict['velocidade'].append(h.velocidade[idata])
            obj_to_dict['tensao_c'].append(h.tensao_c[idata])
            obj_to_dict['nivel_dagua'].append(h.nivel_dagua[idata])
            obj_to_dict['froude'].append(h.froude[idata])

            if lista_modelagem['s_t']:
                obj_to_dict['data'].append(dias[idata])
            if lista_modelagem['m_od']:
                obj_to_dict['conc_od'].append(cc.conc_od[idata])
                obj_to_dict['conc_dbo'].append(cc.conc_dbo[idata])
            if lista_modelagem['m_n']:
                obj_to_dict['conc_no'].append(cc.conc_no[idata])
                obj_to_dict['conc_n_amon'].append(cc.conc_n_amon[idata])
                obj_to_dict['conc_nitrito'].append(cc.conc_nitrito[idata])
                obj_to_dict['conc_nitrato'].append(cc.conc_nitrato[idata])
            if lista_modelagem['m_p']:
                obj_to_dict['conc_p_org'].append(cc.conc_p_org[idata])
                obj_to_dict['conc_p_inorg'].append(cc.conc_p_inorg[idata])
                obj_to_dict['conc_p_total'].append(cc.conc_p_total[idata])
            if lista_modelagem['m_c']:
                obj_to_dict['conc_e_coli'].append(cc.conc_e_coli[idata])

    df = pd.DataFrame(obj_to_dict)
    lidt_df.append(df)
```

```
plotar(n_tributarios, lista_modelagem, lidt_df, list_entr, labels, zona,
hemisferio, dias)
```

X. pages/graficos.py:

```
# biblioteca a serem importadas
import streamlit as st
import pandas as pd
from plotly.subplots import make_subplots
import plotly.graph_objects as go

bar = st.sidebar
bar.warning('''Atenção! As tabelas com os resultados finais podem
            ser baixadas ao final da modelagem/calibração.''',
            icon="⚠️")
st.markdown('##### :orange[Plotagem dos resultados (Opcional)]')
uploaded_file = st.file_uploader("Submeter o arquivo .csv"
                          + " construído na etapa de modelagem/calibração:",
                            type=["csv"])
if uploaded_file is None:
    st.markdown("""❗ Obs.: Evite alterar o arquivo gerado na modelagem
        utilizando o Excel. Se os dados não preencherem automaticamente,
        recomenda-se excecutar a modelagem novamente.""")

else:
    st.markdown('''<h4 style='text-align: center;
            color: teal;
            '>Visualização de dados </h4>''',
            unsafe_allow_html=True)
    df_data = pd.read_csv(uploaded_file, index_col=False)

    nomes = list(df_data.columns)

    id_rio = sorted(df_data['rio'].unique())
    col1, col2 = st.columns(2)
    rio = col1.radio('ID do rio - *a ser plotado*:', id_rio,
                     horizontal=True)

    df_data = df_data.loc[df_data['rio'] == rio]

    fig = make_subplots(specs=[[{"secondary_y": True}]])
    col11, col12 = st.columns(2)
    col21, col22 = col12.columns(2)

    options_y = col11.multiselect(
        "Y:",
        nomes,
        nomes[6])


    if 'data' in nomes:

        col2.write('')
        tipo = col2.toggle('Visualizar série temporal')

        if tipo == False:

            option_x = col21.selectbox(
                "X:",
                nomes, index=5)

            id_dia = sorted(df_data['data'].unique())
            option_dia = col22.selectbox(
                "Data:",
                id_dia, index=0)
```

```
            fig.update_layout(title_text="Resultados do Rio " + str(rio)
                              + ' - ' + str(option_dia),
                              title_font_color="teal")

            df_data = df_data.loc[df_data['data'] == option_dia]


        else:
            option_x = 'data'

            option_col_var = col21.selectbox(
                "Agrupar dados da coluna:",
                nomes, index=6)

            id_var = sorted(df_data[option_col_var].unique())
            option_var = col22.selectbox(
                "... com valores IGUAIS a:",
                id_var, index=0)

            fig.update_layout(title_text="Resultados do Rio " + str(rio)
                              + ' - ' + str(rio) + ': ' + str(rio),
                              title_font_color="teal")

            df_data = df_data.loc[df_data[option_col_var] == option_var]


    else:
        fig.update_layout(title_text="Resultados do Rio " + str(rio),
                          title_font_color="teal")

        option_x = col21.selectbox(
            "X:",
            nomes, index=5)

    for i in range(len(options_y)):
                            fig.add_trace(go.Scatter(x=df_data[option_x],
y=df_data[options_y[i]],
                                     name=options_y[i]))

    fig.update_xaxes(minor=dict(ticklen=3, tickcolor="black"))
    fig.update_yaxes(minor=dict(ticklen=3, tickcolor="black"))
    fig.update_layout(
        legend=dict(orientation="h",
                    yanchor="bottom",
                    y=1,
                    xanchor="right",
                    x=0.95),
                xaxis=dict(title=option_x,   ticklen=4,   tickcolor="black",
showgrid=True, showline=True),
        yaxis=dict(showline=True, tickcolor="black", ticklen=4))

    st.plotly_chart(fig, use_container_width=True)

    st.write(df_data)
```

XI. funcoes/equacoes.py:

```
# biblioteca a serem importadas
import numpy as np
import scipy.optimize as opt
import copy

##########################################################################
##############################
```

```
# OBJETOS
# Hidráulica
class Hidraulica:
     def __init__(self, lat, long, comp, vazao, rug, l_rio, altitude,
inclinacao, ang_esq,
                  ang_dir, prof, veloc, to, nivel_ag, froude):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.vazao = vazao
        self.rugosidade_n = rug
        self.largura_rio = l_rio
        self.altitude = altitude
        self.inclinacao = inclinacao
        self.ang_esquerdo = ang_esq
        self.ang_direito = ang_dir
        self.profundidade = prof
        self.velocidade = veloc
        self.tensao_c = to
        self.nivel_dagua = nivel_ag
        self.froude = froude

# Entradas Pontuais
class EntradaPontual:
    def __init__(self, lat, long, comp, c_gerais, vazao, descricao, rio):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.concentracoes = c_gerais
        self.vazao = vazao
        self.descricao = descricao
        self.rio = rio

# Coeficientes
class Coeficientes:
    def __init__(self, temperatura, k_2_calculavel, k_2_max, k_2, k_1, s_d,
k_d, k_s, l_rd, k_so,
                 k_oa, k_an, k_nn, s_amon, k_spo, k_oi, k_b, r_o2_amon,
k_nit_od, s_pinorg):
        self.temperatura = temperatura
        self.k_2_calculavel = k_2_calculavel
        self.k_2_max = k_2_max
        self.k_2 = k_2
        self.k_1 = k_1
        self.s_d = s_d
        self.k_d = k_d
        self.k_s = k_s
        self.l_rd = l_rd
        self.k_so = k_so
        self.k_oa = k_oa
        self.k_an = k_an
        self.k_nn = k_nn
        self.s_amon = s_amon
        self.k_spo = k_spo
        self.k_oi = k_oi
        self.k_b = k_b
        self.r_o2_amon = r_o2_amon
        self.k_nit_od = k_nit_od
        self.s_pinorg = s_pinorg

# Concentrações
class Concentracoes:
    def __init__(self, od, dbo, no, n_amon, nitrato, nitrito, p_org, p_inorg,
p_total, e_coli):
        self.conc_od = od
        self.conc_dbo = dbo
        self.conc_no = no
```

```
        self.conc_n_amon = n_amon
        self.conc_nitrato = nitrato
        self.conc_nitrito = nitrito
        self.conc_p_org = p_org
        self.conc_p_inorg = p_inorg
        self.conc_p_total = p_total
        self.conc_e_coli = e_coli

# Resultados finais
class Quanti_Qualitativo:
    def __init__(self, hidraulica, concentracoes, coeficientes, rio):
        self.hidraulica = hidraulica
        self.concentracoes = concentracoes
        self.coeficientes = coeficientes
        self.rio = rio


#########################################################################
def fun_hipotenusa(x1, x2, y1, y2):
    hipotenusa = np.sqrt((x2-x1)**2 + (y2 -y1)**2)
    return hipotenusa

def     fun_discr(posterior,     anterior,     x,     parametro_posterior,
parametro_anterior):
    if posterior - anterior == 0:
        razao = 1
    else:
        razao = (x - anterior)/(posterior - anterior)
         discret    =    parametro_anterior    +    ((parametro_posterior    -
parametro_anterior) * razao)
    return discret

def lista_hidr(longitude, latitude, altitude, comprimento, discretizacao):
    lista_dist_real = []
    lista_acum = [0]
    acumulado = 0

    if longitude[0] != None:
        for i in range(len(longitude)-1):
                hipotenusa  =  fun_hipotenusa(longitude[i],  longitude[i+1],
latitude[i], latitude[i+1])
            acumulado += hipotenusa
            lista_acum.append(acumulado)
            lista_dist_real.append(hipotenusa)

        m_trecho = lista_acum[-1]
            intervalos  =  list(np.arange(0,  (m_trecho  +  discretizacao),
discretizacao))

        lista_hidraulica= []

        for i in range(len(intervalos)):
            for j in range(len(lista_acum) - 1):
                if lista_acum[j] <= intervalos[i] <= lista_acum[j + 1]:
                        long  =  fun_discr(lista_acum[j  +  1],  lista_acum[j],
intervalos[i],
                                           longitude[j + 1], longitude[j])
                         lat  =  fun_discr(lista_acum[j  +  1],  lista_acum[j],
intervalos[i],
                                           latitude[j + 1], latitude[j])
                         alt  =  fun_discr(lista_acum[j  +  1],  lista_acum[j],
intervalos[i],
                                           altitude[j + 1], altitude[j])

             hidraulica = Hidraulica(lat, long, intervalos[i], None, None,
None, alt,
```

```
                                    None, None, None, None, None, None,
None)
            lista_hidraulica.append(copy.deepcopy(hidraulica))
    else:
         intervalos = list(np.arange(0, (comprimento[-1] + discretizacao),
discretizacao))
        lat = [None]
        long = [None]
        lista_hidraulica= []

        for i in range(len(intervalos)):
            for j in range(len(comprimento) - 1):
                if comprimento[j] <= intervalos[i] <= comprimento[j + 1]:
                       alt = fun_discr(comprimento[j + 1], comprimento[j],
intervalos[i],
                                       altitude[j + 1], altitude[j])

             hidraulica = Hidraulica(None, None, intervalos[i], None, None,
None, alt,
                                    None, None, None, None, None, None, None,
None)
            lista_hidraulica.append(copy.deepcopy(hidraulica))

    # Inclinação
    for i in range(len(lista_hidraulica)):
        if i != (len(lista_hidraulica) - 1):
            incl = (lista_hidraulica[i].altitude
                    - lista_hidraulica[i+1].altitude)/ discretizacao
            if incl <= 0:
                if i == 0:
                    incl = 0.00001
                else:
                    incl = lista_hidraulica[i - 1].inclinacao
        lista_hidraulica[i].inclinacao = np.abs(incl)

    return lista_hidraulica

def  func_hidraulica(lista_hidraulica,  lista_s_pontual,  lista_e_pontual,
lista_e_difusa, lista_s_transversal, discretizacao):
    vazao_atual = 0
    lista_final = []
    a_esq = lista_s_transversal[0].ang_esquerdo
    a_dir = lista_s_transversal[0].ang_direito
    l_rio = lista_s_transversal[0].largura_rio
    rug = lista_s_transversal[0].rugosidade_n
    for i in range(len(lista_hidraulica)):

        # ENTRADAS E SAÍDAS
        for j in range(len(lista_e_pontual)):
                                if    lista_hidraulica[i].comprimento    ==
lista_e_pontual[j].comprimento:
                vazao_atual += lista_e_pontual[j].vazao

        if len(lista_s_pontual) > 0:
            for k in range(len(lista_s_pontual)):
                                    if   lista_hidraulica[i].comprimento   ==
lista_s_pontual[k].comprimento:
                    vazao_atual -= lista_s_pontual[k].vazao
        if len(lista_e_difusa) > 0:
            for n in range(len(lista_e_difusa)):
                                if   lista_e_difusa[n].comprimento_inicial   <=
lista_hidraulica[i].comprimento < lista_e_difusa[n].comprimento_final:

                    fator_divisor = (lista_e_difusa[n].comprimento_final -
lista_e_difusa[n].comprimento_inicial) / discretizacao
                    vazao_atual += (lista_e_difusa[n].vazao / fator_divisor)
```

```
        # SECÇÃO TRANSVERSAL
        for m in range(len(lista_s_transversal) - 1):
                                if    lista_s_transversal[m].comprimento    <=
lista_hidraulica[i].comprimento <= lista_s_transversal[m + 1].comprimento:
                    a_esq = fun_discr(lista_s_transversal[m + 1].comprimento,
lista_s_transversal[m].comprimento,
                                                    lista_hidraulica[i].comprimento,
lista_s_transversal[m + 1].ang_esquerdo,
                                      lista_s_transversal[m].ang_esquerdo)
                    a_dir = fun_discr(lista_s_transversal[m + 1].comprimento,
lista_s_transversal[m].comprimento,
                                                    lista_hidraulica[i].comprimento,
lista_s_transversal[m + 1].ang_direito,
                                      lista_s_transversal[m].ang_direito)
                  l_rio =  fun_discr(lista_s_transversal[m + 1].comprimento,
lista_s_transversal[m].comprimento,
                                                    lista_hidraulica[i].comprimento,
lista_s_transversal[m + 1].largura_rio,
                                      lista_s_transversal[m].largura_rio)
                     rug =  fun_discr(lista_s_transversal[m + 1].comprimento,
lista_s_transversal[m].comprimento,
                                                    lista_hidraulica[i].comprimento,
lista_s_transversal[m + 1].rugosidade_n,
                                      lista_s_transversal[m].rugosidade_n)
        for _ in range(len(lista_s_transversal)):
            list_profundidade = []
            for iq in range(len(vazao_atual)):
                                  funcao_1   =   (vazao_atual[iq]   *   rug)   /
np.sqrt(lista_hidraulica[i].inclinacao)
                funcao_2 = lambda y : (((((2 * l_rio + (y / np.tan(a_esq *
np.pi / 180)) + (y / np.tan(a_dir * np.pi / 180))) * (y / 2)) ** (5/3)) / (
                    ((y / np.sin(a_esq * np.pi / 180)) + (y / np.sin(a_dir
* np.pi / 180)) + l_rio) ** (2/3))) - funcao_1)
                prof = opt.bisect(funcao_2, 0,500)
                list_profundidade.append(prof)

            list_profundidade = np.array(list_profundidade)
            nivel_dagua = lista_hidraulica[i].altitude + list_profundidade
             area = ((2 * l_rio + (list_profundidade / np.tan(a_esq * np.pi
/  180))  +  (list_profundidade  /  np.tan(a_dir  *  np.pi  /  180)))  *
(list_profundidade / 2))
            veloc = vazao_atual / area
            tensao_c = 9810 * lista_hidraulica[i].inclinacao * (area / (
                    (list_profundidade  /  np.sin(a_esq  *  np.pi  /  180))  +
(list_profundidade / np.sin(a_dir * np.pi / 180)) + l_rio))
            froude = veloc / (np.sqrt(9.81 * list_profundidade))

        lista_hidraulica[i].vazao = vazao_atual
        lista_hidraulica[i].ang_esquerdo = a_esq
        lista_hidraulica[i].ang_direito = a_dir
        lista_hidraulica[i].rugosidade_n = rug
        lista_hidraulica[i].largura_rio = l_rio
        lista_hidraulica[i].profundidade = list_profundidade
        lista_hidraulica[i].nivel_dagua = nivel_dagua
        lista_hidraulica[i].velocidade = veloc
        lista_hidraulica[i].tensao_c = tensao_c
        lista_hidraulica[i].froude = froude

        final = Quanti_Qualitativo(lista_hidraulica[i], None, None, None)
        lista_final.append(copy.deepcopy(final))

    return lista_final

def k2(hidraulica):

    if 0.1 <= hidraulica.profundidade < 4 and 0.05 <= hidraulica.velocidade
< 1.6:
```

```
            if 0.6 <= hidraulica.profundidade < 4 and 0.05 <=
hidraulica.velocidade < 0.8:
                    k_2 = 3.73 * (hidraulica.velocidade ** 0.5) *
(hidraulica.profundidade ** -1.5)
            elif 0.6 <= hidraulica.profundidade < 4 and 0.8 <=
hidraulica.velocidade < 1.6:
                    k_2 = 5 * (hidraulica.velocidade ** 0.97) *
(hidraulica.profundidade ** -1.67)
        else:
                    k_2 = 5.3 * (hidraulica.velocidade ** 0.67) *
(hidraulica.profundidade ** -1.85)
    elif 0.03 <= hidraulica.vazao <= 8.5:
        if 0.03 <= hidraulica.vazao < 0.3:
            k_2 = 31.6 * hidraulica.velocidade * hidraulica.inclinacao *
1000
        else:
            k_2 = 15.4 * hidraulica.velocidade * hidraulica.inclinacao *
1000
    else:
        k_2 = 20.74 * (hidraulica.vazao ** -0.42)
    return k_2

def mistura(parametro, e_paramentro, vazao, e_vazao):
    conc = ((parametro * (vazao - e_vazao)) + (e_paramentro * e_vazao)) /
vazao
    return conc

def od(tempo_delta, coeficientes, concentracoes, hidraulica, f_nitr,
anaerobiose, mod_n):

    od_saturacao = (1 - (hidraulica.altitude/9450)) * (
        14.652 - (4.1022 * (10 ** -1) * coeficientes.temperatura)
        + (7.991 * (10 ** -3) * (coeficientes.temperatura ** 2))
        - (7.7774 * (10 ** -5) * (coeficientes.temperatura ** 3)))

    lista_c_od = []
    for iod in range(len(anaerobiose)):

        if anaerobiose[iod]:
            conc_od = 0

        else:
            k_t = 1 / (1 - np.exp(-5 * (coeficientes.k_1[iod] * (1.047 **
(coeficientes.temperatura[iod] - 20)))))

                        reac = ((coeficientes.k_2[iod] * (1.024 **
(coeficientes.temperatura[iod] - 20)) * (od_saturacao[iod] -
concentracoes.conc_od[iod]) * tempo_delta[iod]))
                    dbo_carb = (- ((coeficientes.k_d[iod] * (1.047 **
(coeficientes.temperatura[iod] - 20))) * k_t * concentracoes.conc_dbo[iod]
* tempo_delta[iod]))
                    carga_int = (- (coeficientes.s_d[iod] * (1.06 **
(coeficientes.temperatura[iod] - 20)) / hidraulica.profundidade[iod]) *
(tempo_delta[iod]))

            if mod_n:
                oxi_am = (- (coeficientes.r_o2_amon[iod] * f_nitr[iod] *
concentracoes.conc_n_amon[iod] * coeficientes.k_an[iod] * (1.08 **
(coeficientes.temperatura[iod] - 20)) * tempo_delta[iod]))

            else:
                oxi_am = 0

               conc_od = concentracoes.conc_od[iod] + (reac + dbo_carb +
carga_int + oxi_am)
            if conc_od <= 0:
                anaerobiose[iod] = True
```

```
                conc_od = 0
            lista_c_od.append(conc_od)
        lista_c_od = np.array(lista_c_od)

        return lista_c_od, od_saturacao, anaerobiose

def dbo(tempo_delta, coeficientes, concentracoes, hidraulica, anaerobiose,
od_saturacao):


        k_r = (coeficientes.k_d * (1.047 ** (coeficientes.temperatura - 20))
                ) + (coeficientes.k_s * (1.024 ** (coeficientes.temperatura -
20)))

        dbo5 = (- k_r * concentracoes.conc_dbo * tempo_delta)
           carg_inte   =   (coeficientes.l_rd   /   (hidraulica.profundidade   *
hidraulica.largura_rio)) * tempo_delta

        conc_dbo_aerobio = concentracoes.conc_dbo + dbo5 + carg_inte
        lista_c_dbo = []
        for idbo in range(len(anaerobiose)):
            if anaerobiose[idbo]:
                        conc_ana   =   -   (coeficientes.k_2[idbo]   *   (1.024   **
(coeficientes.temperatura[idbo] - 20)) * od_saturacao[idbo])
                         conc_dbo_anaerobio   =   concentracoes.conc_dbo[idbo]   +
(tempo_delta[idbo] * conc_ana)

                if abs(conc_dbo_anaerobio - conc_dbo_aerobio[idbo]) <= 0.001:
                    anaerobiose[idbo] = False
                    conc_dbo = conc_dbo_aerobio[idbo]
                else:
                    conc_dbo = conc_dbo_anaerobio
            else:
                conc_dbo = conc_dbo_aerobio[idbo]

            lista_c_dbo.append(conc_dbo)
        lista_c_dbo = np.array(lista_c_dbo)


        return lista_c_dbo, anaerobiose

def no(tempo_delta, coeficientes, concentracoes):
        conc = - ((coeficientes.k_oa * (1.047 ** (coeficientes.temperatura - 20))
                 ) + (coeficientes.k_so * (1.024568 ** (coeficientes.temperatura
- 20)))) * concentracoes.conc_no
        conc_no = concentracoes.conc_no + (tempo_delta * conc)
        return conc_no

def n_amon(tempo_delta, coeficientes, concentracoes, hidraulica, f_nitr):
        conc = (coeficientes.k_oa * (1.047 ** (coeficientes.temperatura - 20))
* concentracoes.conc_no
                  ) - (f_nitr * concentracoes.conc_n_amon *coeficientes.k_an *
(1.08 ** (coeficientes.temperatura - 20))
                                    )   +   (coeficientes.s_amon   *   (1.074   **
(coeficientes.temperatura - 20)) / hidraulica.profundidade)
        conc_n_amon = concentracoes.conc_n_amon + (tempo_delta * conc)
        return conc_n_amon

def nitrito(tempo_delta, coeficientes, concentracoes, f_nitr):
        conc = (f_nitr * concentracoes.conc_n_amon * coeficientes.k_an * (1.08
** (coeficientes.temperatura - 20))
                ) - (f_nitr * concentracoes.conc_nitrito * coeficientes.k_nn *
(1.047 ** (coeficientes.temperatura - 20)))
        conc_nitrito = concentracoes.conc_nitrito + (tempo_delta * conc)
        return conc_nitrito

def nitrato(tempo_delta, coeficientes, concentracoes, f_nitr):
```

```
    conc = f_nitr * coeficientes.k_nn * concentracoes.conc_nitrito * (1.047
** (coeficientes.temperatura - 20)) * tempo_delta
    conc_nitrato = concentracoes.conc_nitrato + conc

    return conc_nitrato

def p_org(tempo_delta, coeficientes, concentracoes):
    conc = - ((coeficientes.k_oi * (1.047 ** (coeficientes.temperatura - 20))
              ) + (coeficientes.k_spo * (1.024 ** (coeficientes.temperatura
- 20)))) * concentracoes.conc_p_org
    conc_p_org = concentracoes.conc_p_org + (tempo_delta * conc)
    return conc_p_org

def p_inorg(tempo_delta, coeficientes, concentracoes, hidraulica):
    conc = (coeficientes.k_oi * (1.047 ** (coeficientes.temperatura - 20))
* concentracoes.conc_p_org
           ) + (coeficientes.s_pinorg * (1.074 ** (coeficientes.temperatura
- 20)) / hidraulica.profundidade)
    conc_p_inorg = concentracoes.conc_p_inorg + (tempo_delta * conc)
    return conc_p_inorg

def e_coli(tempo_delta, coeficientes, concentracoes):
      conc = coeficientes.k_b * concentracoes.conc_e_coli * (1.07 **
(coeficientes.temperatura - 20)) * tempo_delta
    conc_e_coli = concentracoes.conc_e_coli - conc
    return conc_e_coli

def    modelagem(lista_final,    lista_e_coeficientes,    lista_s_pontual,
lista_e_pontual,
              lista_e_difusa, rio, discretizacao, lista_modelagem):
    anaerobiose = [False] * len(lista_e_pontual[0].vazao)
    vazao = 0
    coeficientes = Coeficientes(None, None, None, None, None, None, None,
None, None, None,
                                None, None, None, None, None, None, None,
None, None, None)

    for i in range(len(lista_final)):
        hidraulica = lista_final[i].hidraulica

        for j in range(len(lista_e_coeficientes)):
            atual = lista_e_coeficientes[j]
            if atual.comprimento == hidraulica.comprimento:
                k_2_calculavel = atual.coeficientes.k_2_calculavel
                k_2_max = atual.coeficientes.k_2_max
                  if lista_modelagem['m_od'] == True and k_2_calculavel ==
True:
                    k_2 = np.array(k2(lista_final[0].hidraulica))
                        k_2 = np.where(k_2 > atual.coeficientes.k_2_max,
atual.coeficientes.k_2_max, k_2)
                else:
                    k_2 = atual.coeficientes.k_2
                temperatura = atual.coeficientes.temperatura
                k_1 = atual.coeficientes.k_1
                s_d = atual.coeficientes.s_d
                k_d = atual.coeficientes.k_d
                k_s = atual.coeficientes.k_s
                l_rd = atual.coeficientes.l_rd
                k_so = atual.coeficientes.k_so
                k_oa = atual.coeficientes.k_oa
                k_an = atual.coeficientes.k_an
                k_nn = atual.coeficientes.k_nn
                s_amon = atual.coeficientes.s_amon
                k_spo = atual.coeficientes.k_spo
                k_oi = atual.coeficientes.k_oi
                k_b = atual.coeficientes.k_b
                r_o2_amon = atual.coeficientes.r_o2_amon
```

```
                k_nit_od = atual.coeficientes.k_nit_od
                s_pinorg = atual.coeficientes.s_pinorg

        coeficientes.k_2 = k_2
        coeficientes.k_2_calculavel = k_2_calculavel
        coeficientes.k_2_max = k_2_max
        coeficientes.temperatura = temperatura
        coeficientes.k_1 = k_1
        coeficientes.s_d = s_d
        coeficientes.k_d = k_d
        coeficientes.k_s = k_s
        coeficientes.l_rd = l_rd
        coeficientes.k_so = k_so
        coeficientes.k_oa = k_oa
        coeficientes.k_an = k_an
        coeficientes.k_nn = k_nn
        coeficientes.s_amon = s_amon
        coeficientes.k_spo = k_spo
        coeficientes.k_oi = k_oi
        coeficientes.k_b = k_b
        coeficientes.r_o2_amon = r_o2_amon
        coeficientes.k_nit_od = k_nit_od
        coeficientes.s_pinorg = s_pinorg

        if i == 0:
            concentracoes = copy.deepcopy(lista_e_pontual[0].concentracoes)

        for k in range(len(lista_e_pontual)):
            ep_concetracoes = lista_e_pontual[k].concentracoes
            if hidraulica.comprimento == lista_e_pontual[k].comprimento:
                vazao += lista_e_pontual[k].vazao
                if lista_modelagem['m_od']:
                     concentracoes.conc_od = mistura(concentracoes.conc_od,
ep_concetracoes.conc_od, vazao, lista_e_pontual[k].vazao)
                if lista_modelagem['m_od']:
                   concentracoes.conc_dbo = mistura(concentracoes.conc_dbo,
ep_concetracoes.conc_dbo, vazao, lista_e_pontual[k].vazao)
                if lista_modelagem['m_n']:
                     concentracoes.conc_no = mistura(concentracoes.conc_no,
ep_concetracoes.conc_no, vazao, lista_e_pontual[k].vazao)
                                          concentracoes.conc_n_amon    =
mistura(concentracoes.conc_n_amon,    ep_concetracoes.conc_n_amon,    vazao,
lista_e_pontual[k].vazao)
                                         concentracoes.conc_nitrito    =
mistura(concentracoes.conc_nitrito,   ep_concetracoes.conc_nitrito,   vazao,
lista_e_pontual[k].vazao)
                                         concentracoes.conc_nitrato    =
mistura(concentracoes.conc_nitrato,   ep_concetracoes.conc_nitrato,   vazao,
lista_e_pontual[k].vazao)
                if lista_modelagem['m_p']:
                                           concentracoes.conc_p_org    =
mistura(concentracoes.conc_p_org,     ep_concetracoes.conc_p_org,     vazao,
lista_e_pontual[k].vazao)
                                         concentracoes.conc_p_inorg    =
mistura(concentracoes.conc_p_inorg,   ep_concetracoes.conc_p_inorg,   vazao,
lista_e_pontual[k].vazao)
                    concentracoes.conc_p_total = concentracoes.conc_p_org +
concentracoes.conc_p_inorg
                if lista_modelagem['m_c']:
                                          concentracoes.conc_e_coli    =
mistura(concentracoes.conc_e_coli,    ep_concetracoes.conc_e_coli,    vazao,
lista_e_pontual[k].vazao)

        if len(lista_e_difusa) > 0 :
            for n in range(len(lista_e_difusa)):
                ed_concetracoes = lista_e_difusa[n].concentracoes
```

```
                            if  lista_e_difusa[n].comprimento_inicial  <=
hidraulica.comprimento <= lista_e_difusa[n].comprimento_final:

                    vazao_difusa = (lista_e_difusa[n].vazao *  discretizacao
                                    ) / (lista_e_difusa[n].comprimento_final
- lista_e_difusa[n].comprimento_inicial)
                    vazao += vazao_difusa

                    if lista_modelagem['m_od']:
                      concentracoes.conc_od = mistura(concentracoes.conc_od,
ed_concetracoes.conc_od, hidraulica.vazao, vazao_difusa)
                    if lista_modelagem['m_od']:
                                                concentracoes.conc_dbo   =
mistura(concentracoes.conc_dbo, ed_concetracoes.conc_dbo, hidraulica.vazao,
vazao_difusa)
                    if lista_modelagem['m_n']:
                      concentracoes.conc_no = mistura(concentracoes.conc_no,
ed_concetracoes.conc_no, hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_n_amon   =
mistura(concentracoes.conc_n_amon,             ed_concetracoes.conc_n_amon,
hidraulica.vazao, vazao_difusa)
                                              concentracoes.conc_nitrito   =
mistura(concentracoes.conc_nitrito,            ed_concetracoes.conc_nitrito,
hidraulica.vazao, vazao_difusa)
                                              concentracoes.conc_nitrato   =
mistura(concentracoes.conc_nitrato,            ed_concetracoes.conc_nitrato,
hidraulica.vazao, vazao_difusa)
                    if lista_modelagem['m_p']:
                                                concentracoes.conc_p_org   =
mistura(concentracoes.conc_p_org,               ed_concetracoes.conc_p_org,
hidraulica.vazao, vazao_difusa)
                                              concentracoes.conc_p_inorg   =
mistura(concentracoes.conc_p_inorg,            ed_concetracoes.conc_p_inorg,
hidraulica.vazao, vazao_difusa)
                       concentracoes.conc_p_total = concentracoes.conc_p_org
+ concentracoes.conc_p_inorg
                    if lista_modelagem['m_c']:
                                               concentracoes.conc_e_coli   =
mistura(concentracoes.conc_e_coli,             ed_concetracoes.conc_e_coli,
hidraulica.vazao, vazao_difusa)


        if len(lista_s_pontual) > 0 :
            for p in range(len(lista_s_pontual)):
                                          if   hidraulica.comprimento   ==
lista_s_pontual[p].comprimento:
                        vazao -= lista_s_pontual[p].vazao


        tempo_delta = discretizacao / (hidraulica.velocidade * 86400)

        if lista_modelagem['m_n']:
                f_nitr  =  1  -  np.exp(1)  **  (-  coeficientes.k_nit_od  *
concentracoes.conc_od)
        else:
            f_nitr = 0 * concentracoes.conc_od

        if lista_modelagem['m_od']:
                     concentracoes.conc_od,  od_saturacao,  anaerobiose  =
od(tempo_delta,    coeficientes,    concentracoes,    hidraulica,    f_nitr,
anaerobiose, lista_modelagem['m_n'])
        if lista_modelagem['m_od']:
                   concentracoes.conc_dbo,  anaerobiose  =  dbo(tempo_delta,
coeficientes, concentracoes, hidraulica, anaerobiose, od_saturacao)
        if lista_modelagem['m_n']:
                    concentracoes.conc_no  =  no(tempo_delta,  coeficientes,
concentracoes)
```

```
            concentracoes.conc_n_amon = n_amon(tempo_delta, coeficientes,
concentracoes, hidraulica, f_nitr)
            concentracoes.conc_nitrito = nitrito(tempo_delta, coeficientes,
concentracoes, f_nitr)
            concentracoes.conc_nitrato = nitrato(tempo_delta, coeficientes,
concentracoes, f_nitr)
        if lista_modelagem['m_p']:
              concentracoes.conc_p_org = p_org(tempo_delta, coeficientes,
concentracoes)
            concentracoes.conc_p_inorg = p_inorg(tempo_delta, coeficientes,
concentracoes, hidraulica)
                concentracoes.conc_p_total = concentracoes.conc_p_org +
concentracoes.conc_p_inorg
        if lista_modelagem['m_c']:
             concentracoes.conc_e_coli = e_coli(tempo_delta, coeficientes,
concentracoes)


        lista_final[i].concentracoes = copy.deepcopy(concentracoes)
        lista_final[i].coeficientes = copy.deepcopy(coeficientes)
        lista_final[i].rio = rio
    return lista_final

def menor_dist(list_lat, list_long, list_comp, lat_af, long_af, comp_af):
    dist_ant = 10 ** 50

    for k in range(len(list_lat)):

        if str(lat_af) != 'None' and str(lat_af) != 'nan':
            distancia = fun_hipotenusa(lat_af, list_lat[k],
                                       long_af, list_long[k])
        else:
            distancia = abs(comp_af - list_comp[k])
        if distancia <= dist_ant:
            comp = list_comp[k]
            dist_ant = distancia

    return comp

def menor_dist2(dados_desague, lat_af, long_af, comp_af):
    dist_ant = 10 ** 50

    for k in range(len(dados_desague.lista_hidraulica)):
        if str(lat_af) != 'None' and str(lat_af) != 'nan':
                                  distancia    =    fun_hipotenusa(lat_af,
dados_desague.lista_hidraulica[k].latitude,
                                                                   long_af,
dados_desague.lista_hidraulica[k].longitude)
        else:
                                       distancia     =     abs(comp_af     -
dados_desague.lista_hidraulica[k].comprimento)
        if distancia <= dist_ant:
            comp = dados_desague.lista_hidraulica[k].comprimento
            dist_ant = distancia

    return comp

def modelagem_Final(lista_rio, ponto_af, lista_modelagem,
                    ordem_desague, ordem_modelagem):

    lista_completa_final = []
    for ior in ordem_modelagem:
        dados = lista_rio[ior]
             lista_hidr_model   =   func_hidraulica(dados.lista_hidraulica,
dados.lista_s_pontual,
                                                    dados.lista_e_pontual,
dados.lista_e_difusa,
```

```
                                                    dados.lista_s_transversal,
dados.discretizacao)

                          lista_final     =     modelagem(lista_hidr_model,
dados.lista_e_coeficientes, dados.lista_s_pontual,
                                dados.lista_e_pontual, dados.lista_e_difusa,
dados.rio,
                                 dados.discretizacao, lista_modelagem)


        if ior != 0:
                    comp  =  menor_dist2(lista_rio[ordem_desague[ior  -  1]],
ponto_af[ior - 1][1],
                                ponto_af[ior - 1][2], ponto_af[ior - 1][3])

             afluente = EntradaPontual(ponto_af[ior - 1][1], ponto_af[ior -
1][2],
                                       comp, lista_final[-1].concentracoes,
                                           lista_final[-1].hidraulica.vazao,
ponto_af[ior - 1][0], lista_final[-1].rio)

                                             lista_rio[ordem_desague[ior      -
1]].lista_e_pontual.append(copy.deepcopy(afluente))

        lista_completa_final.append(copy.deepcopy(lista_final))
    return lista_completa_final, lista_rio[0].lista_e_pontual

##########################################################################
####################################

def  cb_od(tempo_delta,  coeficientes,  concentracoes,  hidraulica,  f_nitr,
anaerobiose, mod_n):

    od_saturacao = (1 - (hidraulica.altitude/9450)) * (
        14.652 - (4.1022 * (10 ** -1) * coeficientes.temperatura)
        + (7.991 * (10 ** -3) * (coeficientes.temperatura ** 2))
        - (7.7774 * (10 ** -5) * (coeficientes.temperatura ** 3)))

    lista_c_od = []
    for iod in range(len(anaerobiose)):

        if anaerobiose[iod]:
            conc_od = 0

        else:
              k_t  =  1  /  (1  -  np.exp(-5  *  (coeficientes.k_1  *  (1.047  **
(coeficientes.temperatura - 20)))))

            reac = ((coeficientes.k_2 * (1.024 ** (coeficientes.temperatura
- 20)) * (od_saturacao - concentracoes.conc_od[iod]) * tempo_delta[iod]))
                     dbo_carb   =   (-   ((coeficientes.k_d   *   (1.047   **
(coeficientes.temperatura  -  20)))  *  k_t  *  concentracoes.conc_dbo[iod]  *
tempo_delta[iod]))
                      carga_int   =   (-   (coeficientes.s_d   *   (1.06   **
(coeficientes.temperatura   -   20))   /   hidraulica.profundidade[iod])   *
(tempo_delta[iod]))

            if mod_n:
                    oxi_am  =  (-  (coeficientes.r_o2_amon  *  f_nitr[iod]  *
concentracoes.conc_n_amon[iod]    *    coeficientes.k_an    *    (1.08    **
(coeficientes.temperatura - 20)) * tempo_delta[iod]))

            else:
                oxi_am = 0

               conc_od  =  concentracoes.conc_od[iod]  +  (reac  +  dbo_carb  +
carga_int + oxi_am)
```

```
            if conc_od <= 0:
                anaerobiose[iod] = True
                conc_od = 0
        lista_c_od.append(conc_od)
    lista_c_od = np.array(lista_c_od)


    return lista_c_od, od_saturacao, anaerobiose

def    cb_dbo(tempo_delta,    coeficientes,    concentracoes,    hidraulica,
anaerobiose, od_saturacao):

    k_r = (coeficientes.k_d * (1.047 ** (coeficientes.temperatura - 20))
           ) + (coeficientes.k_s * (1.024 ** (coeficientes.temperatura -
20)))

    dbo5 = (- k_r * concentracoes.conc_dbo * tempo_delta)
       carg_inte   =   (coeficientes.l_rd   /   (hidraulica.profundidade   *
hidraulica.largura_rio)) * tempo_delta

    conc_dbo_aerobio = concentracoes.conc_dbo + dbo5 + carg_inte
    lista_c_dbo = []
    for idbo in range(len(anaerobiose)):
        if anaerobiose[idbo]:
                      conc_ana   =   -   (coeficientes.k_2   *   (1.024   **
(coeficientes.temperatura - 20)) * od_saturacao)
                  conc_dbo_anaerobio   =   concentracoes.conc_dbo[idbo]   +
(tempo_delta[idbo] * conc_ana)
            if abs(conc_dbo_anaerobio - conc_dbo_aerobio[idbo]) <= 0.001:
                anaerobiose[idbo] = False
                conc_dbo = conc_dbo_aerobio[idbo]
            else:
                conc_dbo = conc_dbo_anaerobio
        else:
            conc_dbo = conc_dbo_aerobio[idbo]

        lista_c_dbo.append(conc_dbo)
    lista_c_dbo = np.array(lista_c_dbo)

    return lista_c_dbo, anaerobiose

def cb_no(tempo_delta, coeficientes, concentracoes):
    conc = - ((coeficientes.k_oa * (1.047 ** (coeficientes.temperatura - 20))
             ) + (coeficientes.k_so * (1.024568 ** (coeficientes.temperatura
- 20)))) * concentracoes.conc_no
    conc_no = concentracoes.conc_no + (tempo_delta * conc)
    return conc_no

def cb_n_amon(tempo_delta, coeficientes, concentracoes, hidraulica, f_nitr):
    conc = (coeficientes.k_oa * (1.047 ** (coeficientes.temperatura - 20))
* concentracoes.conc_no
              ) - (f_nitr * concentracoes.conc_n_amon *coeficientes.k_an *
(1.08 ** (coeficientes.temperatura - 20))
                                )   +   (coeficientes.s_amon   *   (1.074   **
(coeficientes.temperatura - 20)) / hidraulica.profundidade)
    conc_n_amon = concentracoes.conc_n_amon + (tempo_delta * conc)
    return conc_n_amon

def cb_nitrito(tempo_delta, coeficientes, concentracoes, f_nitr):
    conc = (f_nitr * concentracoes.conc_n_amon * coeficientes.k_an * (1.08
** (coeficientes.temperatura - 20))
           ) - (f_nitr * concentracoes.conc_nitrito * coeficientes.k_nn *
(1.047 ** (coeficientes.temperatura - 20)))
    conc_nitrito = concentracoes.conc_nitrito + (tempo_delta * conc)
    return conc_nitrito
```

```
def cb_nitrato(tempo_delta, coeficientes, concentracoes, f_nitr):
    conc = (f_nitr * coeficientes.k_nn * concentracoes.conc_nitrito * (1.047
** (coeficientes.temperatura - 20)))
    conc_nitrato = concentracoes.conc_nitrato + (tempo_delta * conc)
    return conc_nitrato

def cb_p_org(tempo_delta, coeficientes, concentracoes):
    conc = - ((coeficientes.k_oi * (1.047 ** (coeficientes.temperatura - 20))
            ) + (coeficientes.k_spo * (1.024 ** (coeficientes.temperatura
- 20)))) * concentracoes.conc_p_org
    conc_p_org = concentracoes.conc_p_org + (tempo_delta * conc)
    return conc_p_org

def cb_p_inorg(tempo_delta, coeficientes, concentracoes, hidraulica):
    conc = (coeficientes.k_oi * (1.047 ** (coeficientes.temperatura - 20))
* concentracoes.conc_p_org
          ) + (coeficientes.s_pinorg * (1.074 ** (coeficientes.temperatura
- 20)) / hidraulica.profundidade)
    conc_p_inorg = concentracoes.conc_p_inorg + (tempo_delta * conc)
    return conc_p_inorg

def cb_e_coli(tempo_delta, coeficientes, concentracoes):
     conc = - coeficientes.k_b * concentracoes.conc_e_coli * (1.07 **
(coeficientes.temperatura - 20))
    conc_e_coli = concentracoes.conc_e_coli + (tempo_delta * conc)
    return conc_e_coli

def     modelagem_as(lista_final,     e_coeficientes,     lista_s_pontual,
lista_e_pontual,
              lista_e_difusa, discretizacao, lista_modelagem, valor_coef,
id_coef):
    coef_od = ['k_1', 'k_2', 'k_d', 'k_s', 'l_rd', 's_d', 'r_o2_amon',
'k_oa', 'k_so', 'k_an', 's_amon', 'k_nn', 'k_nit_od', 'temperatura']
     coef_n = ['k_oa', 'k_so', 'k_an', 's_amon', 'k_nn', 'k_nit_od',
'temperatura']
    coef_p = ['k_oi', 'k_spo', 's_pinorg', 'temperatura']
    coef_cf = ['k_b', 'temperatura']
    anaerobiose = [False] * len(lista_e_pontual[0].vazao)
    vazao = 0
    coeficientes = copy.deepcopy(e_coeficientes)
    setattr(coeficientes, id_coef, valor_coef)

    for i in range(len(lista_final)):
        hidraulica = lista_final[i].hidraulica

        if i == 0:
            concentracoes = copy.deepcopy(lista_e_pontual[0].concentracoes)


        for k in range(len(lista_e_pontual)):
            ep_concetracoes = lista_e_pontual[k].concentracoes
            if hidraulica.comprimento == lista_e_pontual[k].comprimento:
                vazao += lista_e_pontual[k].vazao
                if (lista_modelagem['m_od']) and (id_coef in coef_od):
                     concentracoes.conc_od = mistura(concentracoes.conc_od,
ep_concetracoes.conc_od, vazao, lista_e_pontual[k].vazao)
                if (lista_modelagem['m_od']) and (id_coef in coef_od):
                   concentracoes.conc_dbo = mistura(concentracoes.conc_dbo,
ep_concetracoes.conc_dbo, vazao, lista_e_pontual[k].vazao)
                if (lista_modelagem['m_n']) and (id_coef in coef_n):
                     concentracoes.conc_no = mistura(concentracoes.conc_no,
ep_concetracoes.conc_no, vazao, lista_e_pontual[k].vazao)
                                            concentracoes.conc_n_amon    =
mistura(concentracoes.conc_n_amon,    ep_concetracoes.conc_n_amon,    vazao,
lista_e_pontual[k].vazao)
```

```
                                            concentracoes.conc_nitrito    =
mistura(concentracoes.conc_nitrito,   ep_concetracoes.conc_nitrito,   vazao,
lista_e_pontual[k].vazao)
                                            concentracoes.conc_nitrato    =
mistura(concentracoes.conc_nitrato,   ep_concetracoes.conc_nitrato,   vazao,
lista_e_pontual[k].vazao)
                if (lista_modelagem['m_p']) and (id_coef in coef_p):
                                              concentracoes.conc_p_org    =
mistura(concentracoes.conc_p_org,     ep_concetracoes.conc_p_org,     vazao,
lista_e_pontual[k].vazao)
                                            concentracoes.conc_p_inorg    =
mistura(concentracoes.conc_p_inorg,   ep_concetracoes.conc_p_inorg,   vazao,
lista_e_pontual[k].vazao)
                    concentracoes.conc_p_total = concentracoes.conc_p_org +
concentracoes.conc_p_inorg
                if (lista_modelagem['m_c']) and (id_coef in coef_cf):
                                              concentracoes.conc_e_coli    =
mistura(concentracoes.conc_e_coli,    ep_concetracoes.conc_e_coli,    vazao,
lista_e_pontual[k].vazao)

        if len(lista_e_difusa) > 0 :
            for n in range(len(lista_e_difusa)):
                ed_concetracoes = lista_e_difusa[n].concentracoes
                            if   lista_e_difusa[n].comprimento_inicial   <=
hidraulica.comprimento <= lista_e_difusa[n].comprimento_final:

                   vazao_difusa = (lista_e_difusa[n].vazao *  discretizacao
                                   ) / (lista_e_difusa[n].comprimento_final
- lista_e_difusa[n].comprimento_inicial)
                    vazao += vazao_difusa

                    if (lista_modelagem['m_od']) and (id_coef in coef_od):
                      concentracoes.conc_od = mistura(concentracoes.conc_od,
ed_concetracoes.conc_od, hidraulica.vazao, vazao_difusa)
                    if (lista_modelagem['m_od']) and (id_coef in coef_od):
                                                  concentracoes.conc_dbo    =
mistura(concentracoes.conc_dbo, ed_concetracoes.conc_dbo, hidraulica.vazao,
vazao_difusa)
                    if (lista_modelagem['m_n']) and (id_coef in coef_n):
                      concentracoes.conc_no = mistura(concentracoes.conc_no,
ed_concetracoes.conc_no, hidraulica.vazao, vazao_difusa)
                                                concentracoes.conc_n_amon    =
mistura(concentracoes.conc_n_amon,              ed_concetracoes.conc_n_amon,
hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_nitrito    =
mistura(concentracoes.conc_nitrito,             ed_concetracoes.conc_nitrito,
hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_nitrato    =
mistura(concentracoes.conc_nitrato,             ed_concetracoes.conc_nitrato,
hidraulica.vazao, vazao_difusa)
                    if (lista_modelagem['m_p']) and (id_coef in coef_p):
                                                 concentracoes.conc_p_org    =
mistura(concentracoes.conc_p_org,                ed_concetracoes.conc_p_org,
hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_p_inorg    =
mistura(concentracoes.conc_p_inorg,             ed_concetracoes.conc_p_inorg,
hidraulica.vazao, vazao_difusa)
                       concentracoes.conc_p_total = concentracoes.conc_p_org
+ concentracoes.conc_p_inorg
                    if (lista_modelagem['m_c']) and (id_coef in coef_cf):
                                                concentracoes.conc_e_coli    =
mistura(concentracoes.conc_e_coli,              ed_concetracoes.conc_e_coli,
hidraulica.vazao, vazao_difusa)


        if len(lista_s_pontual) > 0 :
            for p in range(len(lista_s_pontual)):
```

```
                                            if   hidraulica.comprimento   ==
lista_s_pontual[p].comprimento:
                        vazao -= lista_s_pontual[p].vazao

        tempo_delta = discretizacao / (hidraulica.velocidade * 86400)

        if lista_modelagem['m_n']:
                f_nitr  =  1  -  np.exp(1)  **  (-  coeficientes.k_nit_od  *
concentracoes.conc_od)
        else:
            f_nitr = 0 * concentracoes.conc_od

        if (lista_modelagem['m_od']) and (id_coef in coef_od):
                    concentracoes.conc_od,  od_saturacao,  anaerobiose  =
cb_od(tempo_delta,   coeficientes,   concentracoes,   hidraulica,   f_nitr,
anaerobiose, lista_modelagem['m_n'])
                concentracoes.conc_dbo,  anaerobiose  =  cb_dbo(tempo_delta,
coeficientes, concentracoes, hidraulica, anaerobiose, od_saturacao)
        if (lista_modelagem['m_n']) and (id_coef in coef_n):
                 concentracoes.conc_no  =  cb_no(tempo_delta,  coeficientes,
concentracoes)
            concentracoes.conc_n_amon = cb_n_amon(tempo_delta, coeficientes,
concentracoes, hidraulica, f_nitr)
                    concentracoes.conc_nitrito  =  cb_nitrito(tempo_delta,
coeficientes, concentracoes, f_nitr)
                    concentracoes.conc_nitrato  =  cb_nitrato(tempo_delta,
coeficientes, concentracoes, f_nitr)
        if (lista_modelagem['m_p']) and (id_coef in coef_p):
             concentracoes.conc_p_org = cb_p_org(tempo_delta, coeficientes,
concentracoes)
                    concentracoes.conc_p_inorg  =  cb_p_inorg(tempo_delta,
coeficientes, concentracoes, hidraulica)
                 concentracoes.conc_p_total  =  concentracoes.conc_p_org  +
concentracoes.conc_p_inorg
        if (lista_modelagem['m_c']) and (id_coef in coef_cf):
            concentracoes.conc_e_coli = cb_e_coli(tempo_delta, coeficientes,
concentracoes)

    return concentracoes

def ajust_porc(coef_mm, porcentagem):
    media = np.mean([coef_mm[0], coef_mm[1]])
    value_porc = media * (porcentagem / 100)
    min_value = media - value_porc
    max_value = media + value_porc

    return [min_value, max_value], media


def modelagem_as_final(id_chave, lista_hidr_model, lista_media_coef,
                       lista_rio, lista_modelagem, ordem_modelagem,
                       ordem_desague, ponto_af, i_coef, conj_coeficientes):
    if i_coef == None:
        valor_coef = lista_media_coef.temperatura
    else:
        coef = getattr(conj_coeficientes, id_chave)
        valor_coef = coef[i_coef]

    lista_rio_temporaria = copy.deepcopy(lista_rio)

    for ior in ordem_modelagem:
        dados = lista_rio_temporaria[ior]
         conc_final = modelagem_as(lista_hidr_model[ior], lista_media_coef,
dados.lista_s_pontual,
                                  dados.lista_e_pontual, dados.lista_e_difusa,
                                        dados.discretizacao,  lista_modelagem,
valor_coef, id_chave)
```

```
        if ior != 0:
            comp = menor_dist2(lista_rio_temporaria[ordem_desague[ior - 1]],
ponto_af[ior - 1][1],
                               ponto_af[ior - 1][2], ponto_af[ior - 1][3])

            afluente = EntradaPontual(ponto_af[ior - 1][1], ponto_af[ior -
1][2],
                                      comp, conc_final, lista_hidr_model[ior][-
1].hidraulica.vazao,
                                         ponto_af[ior - 1][0], ior)

                                  lista_rio_temporaria[ordem_desague[ior    -
1]].lista_e_pontual.append(copy.deepcopy(afluente))

    return conc_final, id_chave

def    modelagem_calib(e_coeficientes,    lista_final,    lista_s_pontual,
lista_e_pontual,
                lista_e_difusa, discretizacao, lista_modelagem, ordem_dr,
dados_reais_to):

    anaerobiose = [False] * len(lista_e_pontual[0].vazao)
    vazao = 0
    coeficientes = copy.deepcopy(e_coeficientes)
    lista_concentracoes = []

    for i in range(len(lista_final)):
        hidraulica = lista_final[i].hidraulica

        if i == 0:
            concentracoes = copy.deepcopy(lista_e_pontual[0].concentracoes)


        for k in range(len(lista_e_pontual)):
            ep_concetracoes = lista_e_pontual[k].concentracoes
            if hidraulica.comprimento == lista_e_pontual[k].comprimento:
                vazao += lista_e_pontual[k].vazao
                if lista_modelagem['m_od']:
                     concentracoes.conc_od = mistura(concentracoes.conc_od,
ep_concetracoes.conc_od, vazao, lista_e_pontual[k].vazao)
                if lista_modelagem['m_od']:
                    concentracoes.conc_dbo = mistura(concentracoes.conc_dbo,
ep_concetracoes.conc_dbo, vazao, lista_e_pontual[k].vazao)
                if lista_modelagem['m_n']:
                     concentracoes.conc_no = mistura(concentracoes.conc_no,
ep_concetracoes.conc_no, vazao, lista_e_pontual[k].vazao)
                                              concentracoes.conc_n_amon    =
mistura(concentracoes.conc_n_amon,    ep_concetracoes.conc_n_amon,    vazao,
lista_e_pontual[k].vazao)
                                             concentracoes.conc_nitrito    =
mistura(concentracoes.conc_nitrito,   ep_concetracoes.conc_nitrito,   vazao,
lista_e_pontual[k].vazao)
                                             concentracoes.conc_nitrato    =
mistura(concentracoes.conc_nitrato,   ep_concetracoes.conc_nitrato,   vazao,
lista_e_pontual[k].vazao)
                if lista_modelagem['m_p']:
                                               concentracoes.conc_p_org    =
mistura(concentracoes.conc_p_org,     ep_concetracoes.conc_p_org,     vazao,
lista_e_pontual[k].vazao)
                                             concentracoes.conc_p_inorg    =
mistura(concentracoes.conc_p_inorg,   ep_concetracoes.conc_p_inorg,   vazao,
lista_e_pontual[k].vazao)
                    concentracoes.conc_p_total = concentracoes.conc_p_org +
concentracoes.conc_p_inorg
                if lista_modelagem['m_c']:
```

```
                                                concentracoes.conc_e_coli    =
mistura(concentracoes.conc_e_coli,    ep_concetracoes.conc_e_coli,    vazao,
lista_e_pontual[k].vazao)

        if len(lista_e_difusa) > 0 :
            for n in range(len(lista_e_difusa)):
                ed_concetracoes = lista_e_difusa[n].concentracoes
                            if   lista_e_difusa[n].comprimento_inicial   <=
hidraulica.comprimento <= lista_e_difusa[n].comprimento_final:

                    vazao_difusa = (lista_e_difusa[n].vazao *  discretizacao
                                    ) / (lista_e_difusa[n].comprimento_final
- lista_e_difusa[n].comprimento_inicial)
                    vazao += vazao_difusa

                    if lista_modelagem['m_od']:
                      concentracoes.conc_od = mistura(concentracoes.conc_od,
ed_concetracoes.conc_od, hidraulica.vazao, vazao_difusa)
                    if lista_modelagem['m_od']:
                                                  concentracoes.conc_dbo   =
mistura(concentracoes.conc_dbo, ed_concetracoes.conc_dbo, hidraulica.vazao,
vazao_difusa)
                    if lista_modelagem['m_n']:
                      concentracoes.conc_no = mistura(concentracoes.conc_no,
ed_concetracoes.conc_no, hidraulica.vazao, vazao_difusa)
                                                 concentracoes.conc_n_amon   =
mistura(concentracoes.conc_n_amon,             ed_concetracoes.conc_n_amon,
hidraulica.vazao, vazao_difusa)
                                                concentracoes.conc_nitrito   =
mistura(concentracoes.conc_nitrito,            ed_concetracoes.conc_nitrito,
hidraulica.vazao, vazao_difusa)
                                                concentracoes.conc_nitrato   =
mistura(concentracoes.conc_nitrato,            ed_concetracoes.conc_nitrato,
hidraulica.vazao, vazao_difusa)
                    if lista_modelagem['m_p']:
                                                  concentracoes.conc_p_org   =
mistura(concentracoes.conc_p_org,               ed_concetracoes.conc_p_org,
hidraulica.vazao, vazao_difusa)
                                                concentracoes.conc_p_inorg   =
mistura(concentracoes.conc_p_inorg,            ed_concetracoes.conc_p_inorg,
hidraulica.vazao, vazao_difusa)
                     concentracoes.conc_p_total = concentracoes.conc_p_org
+ concentracoes.conc_p_inorg
                    if lista_modelagem['m_c']:
                                                 concentracoes.conc_e_coli   =
mistura(concentracoes.conc_e_coli,             ed_concetracoes.conc_e_coli,
hidraulica.vazao, vazao_difusa)

        if len(lista_s_pontual) > 0 :
            for p in range(len(lista_s_pontual)):
                                           if   hidraulica.comprimento   ==
lista_s_pontual[p].comprimento:
                     vazao -= lista_s_pontual[p].vazao

        tempo_delta = discretizacao / (hidraulica.velocidade * 86400)
        if lista_modelagem['m_n']:
                 f_nitr  =  1  -  np.exp(1)  **  (-  coeficientes.k_nit_od  *
concentracoes.conc_od)
        else:
            f_nitr = 0 * concentracoes.conc_od



        if lista_modelagem['m_od']:
                    concentracoes.conc_od,  od_saturacao,  anaerobiose  =
cb_od(tempo_delta,   coeficientes,   concentracoes,   hidraulica,   f_nitr,
anaerobiose, lista_modelagem['m_n'])
```

```
                concentracoes.conc_dbo, anaerobiose = cb_dbo(tempo_delta,
coeficientes, concentracoes, hidraulica, anaerobiose, od_saturacao)
        if lista_modelagem['m_n']:
                 concentracoes.conc_no = cb_no(tempo_delta, coeficientes,
concentracoes)
            concentracoes.conc_n_amon = cb_n_amon(tempo_delta, coeficientes,
concentracoes, hidraulica, f_nitr)
                    concentracoes.conc_nitrito = cb_nitrito(tempo_delta,
coeficientes, concentracoes, f_nitr)
                    concentracoes.conc_nitrato = cb_nitrato(tempo_delta,
coeficientes, concentracoes, f_nitr)
        if lista_modelagem['m_p']:
             concentracoes.conc_p_org = cb_p_org(tempo_delta, coeficientes,
concentracoes)
                    concentracoes.conc_p_inorg = cb_p_inorg(tempo_delta,
coeficientes, concentracoes, hidraulica)
                 concentracoes.conc_p_total = concentracoes.conc_p_org +
concentracoes.conc_p_inorg
        if lista_modelagem['m_c']:
            concentracoes.conc_e_coli = cb_e_coli(tempo_delta, coeficientes,
concentracoes)

        if ordem_dr == None:
            if i == (len(lista_final) - 1):
                lista_concentracoes.append(copy.deepcopy(concentracoes))

        else:
            for dr_to in range(len(dados_reais_to)):
                                        if   hidraulica.comprimento   ==
dados_reais_to[dr_to].comprimento:
                  lista_concentracoes.append(copy.deepcopy(concentracoes))


    return lista_concentracoes

def modelagem_calib_final(lista_pos, seq_coef, lista_hidr_model,
                          list_tranfor, lista_modelagem, ordem_rio,
                          ordem_desague, ponto_af, fixar_coef,
                          ordem_dr, trecho_hidr):
    e_coeficientes = copy.deepcopy(fixar_coef)
    for id_c in range(len(seq_coef)):
        setattr(e_coeficientes, seq_coef[id_c], lista_pos[id_c])

    lista_rio_temporaria = copy.deepcopy(list_tranfor)

    for ior in ordem_rio:
        dados = lista_rio_temporaria[ior]

        if ior == ordem_rio[-1]:
            lista_conc_final = modelagem_calib(e_coeficientes, trecho_hidr,
dados.lista_s_pontual,
                                                   dados.lista_e_pontual,
dados.lista_e_difusa,
                                       dados.discretizacao, lista_modelagem,
ordem_dr, dados.lista_dados_reais)

        else:
                    lista_conc_final   =   modelagem_calib(e_coeficientes,
lista_hidr_model[ior], dados.lista_s_pontual,
                                                   dados.lista_e_pontual,
dados.lista_e_difusa,
                                       dados.discretizacao, lista_modelagem,
None, None)

            comp = menor_dist2(lista_rio_temporaria[ordem_desague[ior - 1]],
ponto_af[ior - 1][1],
                               ponto_af[ior - 1][2], ponto_af[ior - 1][3])
```

```
            afluente = EntradaPontual(ponto_af[ior - 1][1], ponto_af[ior -
1][2],
                                                    comp, lista_conc_final[0],
lista_hidr_model[ior][-1].hidraulica.vazao,
                                        ponto_af[ior - 1][0], ior)

                                    lista_rio_temporaria[ordem_desague[ior    -
1]].lista_e_pontual.append(copy.deepcopy(afluente))



    return lista_conc_final

def    modelagem_2(lista_final,    lista_e_coeficientes,    lista_s_pontual,
lista_e_pontual,
              lista_e_difusa, rio, discretizacao, lista_modelagem):
    anaerobiose = [False] * len(lista_e_pontual[0].vazao)
    vazao = 0
     coeficientes = Coeficientes(None, None, None, None, None, None, None,
None, None, None,
                                    None, None, None, None, None, None, None,
None, None, None)

    for i in range(len(lista_final)):
        hidraulica = lista_final[i].hidraulica

        for j in range(len(lista_e_coeficientes)):
            atual = lista_e_coeficientes[j]
            if atual.comprimento == hidraulica.comprimento:
                k_2_calculavel = atual.coeficientes.k_2_calculavel
                k_2_max = atual.coeficientes.k_2_max
                k_2 = atual.coeficientes.k_2
                temperatura = atual.coeficientes.temperatura
                k_1 = atual.coeficientes.k_1
                s_d = atual.coeficientes.s_d
                k_d = atual.coeficientes.k_d
                k_s = atual.coeficientes.k_s
                l_rd = atual.coeficientes.l_rd
                k_so = atual.coeficientes.k_so
                k_oa = atual.coeficientes.k_oa
                k_an = atual.coeficientes.k_an
                k_nn = atual.coeficientes.k_nn
                s_amon = atual.coeficientes.s_amon
                k_spo = atual.coeficientes.k_spo
                k_oi = atual.coeficientes.k_oi
                k_b = atual.coeficientes.k_b
                r_o2_amon = atual.coeficientes.r_o2_amon
                k_nit_od = atual.coeficientes.k_nit_od
                s_pinorg = atual.coeficientes.s_pinorg

        coeficientes.k_2 = k_2
        coeficientes.k_2_calculavel = k_2_calculavel
        coeficientes.k_2_max = k_2_max
        coeficientes.temperatura = temperatura
        coeficientes.k_1 = k_1
        coeficientes.s_d = s_d
        coeficientes.k_d = k_d
        coeficientes.k_s = k_s
        coeficientes.l_rd = l_rd
        coeficientes.k_so = k_so
        coeficientes.k_oa = k_oa
        coeficientes.k_an = k_an
        coeficientes.k_nn = k_nn
        coeficientes.s_amon = s_amon
        coeficientes.k_spo = k_spo
        coeficientes.k_oi = k_oi
```

```
        coeficientes.k_b = k_b
        coeficientes.r_o2_amon = r_o2_amon
        coeficientes.k_nit_od = k_nit_od
        coeficientes.s_pinorg = s_pinorg

        if i == 0:
            concentracoes = copy.deepcopy(lista_e_pontual[0].concentracoes)


        for k in range(len(lista_e_pontual)):
            ep_concetracoes = lista_e_pontual[k].concentracoes
            if hidraulica.comprimento == lista_e_pontual[k].comprimento:
                vazao += lista_e_pontual[k].vazao
                if lista_modelagem['m_od']:
                     concentracoes.conc_od = mistura(concentracoes.conc_od,
ep_concetracoes.conc_od, vazao, lista_e_pontual[k].vazao)
                if lista_modelagem['m_od']:
                   concentracoes.conc_dbo = mistura(concentracoes.conc_dbo,
ep_concetracoes.conc_dbo, vazao, lista_e_pontual[k].vazao)
                if lista_modelagem['m_n']:
                     concentracoes.conc_no = mistura(concentracoes.conc_no,
ep_concetracoes.conc_no, vazao, lista_e_pontual[k].vazao)
                                              concentracoes.conc_n_amon    =
mistura(concentracoes.conc_n_amon,    ep_concetracoes.conc_n_amon,    vazao,
lista_e_pontual[k].vazao)
                                             concentracoes.conc_nitrito    =
mistura(concentracoes.conc_nitrito,   ep_concetracoes.conc_nitrito,   vazao,
lista_e_pontual[k].vazao)
                                             concentracoes.conc_nitrato    =
mistura(concentracoes.conc_nitrato,   ep_concetracoes.conc_nitrato,   vazao,
lista_e_pontual[k].vazao)
                if lista_modelagem['m_p']:
                                               concentracoes.conc_p_org    =
mistura(concentracoes.conc_p_org,     ep_concetracoes.conc_p_org,     vazao,
lista_e_pontual[k].vazao)
                                             concentracoes.conc_p_inorg    =
mistura(concentracoes.conc_p_inorg,   ep_concetracoes.conc_p_inorg,   vazao,
lista_e_pontual[k].vazao)
                    concentracoes.conc_p_total = concentracoes.conc_p_org +
concentracoes.conc_p_inorg
                if lista_modelagem['m_c']:
                                              concentracoes.conc_e_coli    =
mistura(concentracoes.conc_e_coli,    ep_concetracoes.conc_e_coli,    vazao,
lista_e_pontual[k].vazao)

        if len(lista_e_difusa) > 0 :
            for n in range(len(lista_e_difusa)):
                ed_concetracoes = lista_e_difusa[n].concentracoes
                            if  lista_e_difusa[n].comprimento_inicial  <=
hidraulica.comprimento <= lista_e_difusa[n].comprimento_final:

                   vazao_difusa = (lista_e_difusa[n].vazao *  discretizacao
                                   ) / (lista_e_difusa[n].comprimento_final
- lista_e_difusa[n].comprimento_inicial)
                    vazao += vazao_difusa

                    if lista_modelagem['m_od']:
                      concentracoes.conc_od = mistura(concentracoes.conc_od,
ed_concetracoes.conc_od, hidraulica.vazao, vazao_difusa)
                    if lista_modelagem['m_od']:
                                                 concentracoes.conc_dbo   =
mistura(concentracoes.conc_dbo, ed_concetracoes.conc_dbo, hidraulica.vazao,
vazao_difusa)
                    if lista_modelagem['m_n']:
                      concentracoes.conc_no = mistura(concentracoes.conc_no,
ed_concetracoes.conc_no, hidraulica.vazao, vazao_difusa)
```

```
                                                concentracoes.conc_n_amon   =
mistura(concentracoes.conc_n_amon,              ed_concetracoes.conc_n_amon,
hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_nitrito   =
mistura(concentracoes.conc_nitrito,             ed_concetracoes.conc_nitrito,
hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_nitrato   =
mistura(concentracoes.conc_nitrato,             ed_concetracoes.conc_nitrato,
hidraulica.vazao, vazao_difusa)
                    if lista_modelagem['m_p']:
                                                 concentracoes.conc_p_org   =
mistura(concentracoes.conc_p_org,               ed_concetracoes.conc_p_org,
hidraulica.vazao, vazao_difusa)
                                               concentracoes.conc_p_inorg   =
mistura(concentracoes.conc_p_inorg,             ed_concetracoes.conc_p_inorg,
hidraulica.vazao, vazao_difusa)
                      concentracoes.conc_p_total = concentracoes.conc_p_org
+ concentracoes.conc_p_inorg
                    if lista_modelagem['m_c']:
                                                concentracoes.conc_e_coli   =
mistura(concentracoes.conc_e_coli,              ed_concetracoes.conc_e_coli,
hidraulica.vazao, vazao_difusa)

        if len(lista_s_pontual) > 0 :
            for p in range(len(lista_s_pontual)):
                                          if   hidraulica.comprimento   ==
lista_s_pontual[p].comprimento:
                        vazao -= lista_s_pontual[p].vazao

        tempo_delta = discretizacao / (hidraulica.velocidade * 86400)

        if lista_modelagem['m_n']:
                f_nitr  =  1  -  np.exp(1)  **  (-  coeficientes.k_nit_od  *
concentracoes.conc_od)
        else:
            f_nitr = 0 * concentracoes.conc_od

        if lista_modelagem['m_od']:
                    concentracoes.conc_od,  od_saturacao,  anaerobiose  =
cb_od(tempo_delta,   coeficientes,   concentracoes,   hidraulica,   f_nitr,
anaerobiose, lista_modelagem['m_n'])
                concentracoes.conc_dbo,  anaerobiose  =  cb_dbo(tempo_delta,
coeficientes, concentracoes, hidraulica, anaerobiose, od_saturacao)
        if lista_modelagem['m_n']:
                concentracoes.conc_no  =  cb_no(tempo_delta,  coeficientes,
concentracoes)
           concentracoes.conc_n_amon = cb_n_amon(tempo_delta, coeficientes,
concentracoes, hidraulica, f_nitr)
                    concentracoes.conc_nitrito  =  cb_nitrito(tempo_delta,
coeficientes, concentracoes, f_nitr)
                    concentracoes.conc_nitrato  =  cb_nitrato(tempo_delta,
coeficientes, concentracoes, f_nitr)
        if lista_modelagem['m_p']:
             concentracoes.conc_p_org = cb_p_org(tempo_delta, coeficientes,
concentracoes)
                    concentracoes.conc_p_inorg  =  cb_p_inorg(tempo_delta,
coeficientes, concentracoes, hidraulica)
                concentracoes.conc_p_total  =  concentracoes.conc_p_org  +
concentracoes.conc_p_inorg
        if lista_modelagem['m_c']:
            concentracoes.conc_e_coli = cb_e_coli(tempo_delta, coeficientes,
concentracoes)

        lista_final[i].concentracoes = copy.deepcopy(concentracoes)
        lista_final[i].coeficientes = copy.deepcopy(coeficientes)
        lista_final[i].rio = rio
    return lista_final
```

```
def modelagem_Final_2(lista_rio, ponto_af, lista_modelagem,
                      ordem_desague, ordem_modelagem, lista_hidr_model):

    lista_completa_final = []
    marc = 0
    for ior in ordem_modelagem:
        dados = lista_rio[ior]
                      lista_final    =    modelagem_2(lista_hidr_model[marc],
dados.lista_e_coeficientes, dados.lista_s_pontual,
                                dados.lista_e_pontual, dados.lista_e_difusa,
dados.rio,
                                 dados.discretizacao, lista_modelagem)
        marc += 1
        if ior != 0:
                    comp  =  menor_dist2(lista_rio[ordem_desague[ior  -  1]],
ponto_af[ior - 1][1],
                                ponto_af[ior - 1][2], ponto_af[ior - 1][3])

            afluente = EntradaPontual(ponto_af[ior - 1][1], ponto_af[ior -
1][2],
                                      comp, lista_final[-1].concentracoes,
                                          lista_final[-1].hidraulica.vazao,
ponto_af[ior - 1][0], lista_final[-1].rio)

                                             lista_rio[ordem_desague[ior      -
1]].lista_e_pontual.append(copy.deepcopy(afluente))

        lista_completa_final.append(copy.deepcopy(lista_final))
    return lista_completa_final, lista_rio[0].lista_e_pontual

def salvando_conc(list_tranfor, ordem_final, marcador_conj_global,
                  trecho_hidr, coef_ponto, lista_modelagem, lista_hidr_model,
ordem_desague, ponto_af):
    ordem_rio = ordem_final[marcador_conj_global]
    for ior in ordem_rio:
        dados = list_tranfor[ior]

        if ior == ordem_rio[-1]:
                lista_conc_final  =  modelagem_calib(coef_ponto,  trecho_hidr,
dados.lista_s_pontual,
                                                     dados.lista_e_pontual,
dados.lista_e_difusa,
                                      dados.discretizacao, lista_modelagem,
None, None)
            comp = trecho_hidr[-1].hidraulica.comprimento
                        afluente   =   EntradaPontual(None,   None,   comp,
copy.deepcopy(lista_conc_final[0]), trecho_hidr[-1].hidraulica.vazao,
                                      None, ior)
            list_tranfor[0].lista_e_pontual.append(copy.deepcopy(afluente))


        else:
                          lista_conc_final   =   modelagem_calib(coef_ponto,
lista_hidr_model[ior], dados.lista_s_pontual,
                                                     dados.lista_e_pontual,
dados.lista_e_difusa,
                                      dados.discretizacao, lista_modelagem,
None, None)

                  comp  =  menor_dist2(list_tranfor[ordem_desague[ior  -  1]],
ponto_af[ior - 1][1],
                                ponto_af[ior - 1][2], ponto_af[ior - 1][3])

            afluente = EntradaPontual(ponto_af[ior - 1][1], ponto_af[ior -
1][2],
```

```
                                                    comp, lista_conc_final[0],
lista_hidr_model[ior][-1].hidraulica.vazao,
                                        ponto_af[ior - 1][0], ior)

                                         list_tranfor[ordem_desague[ior     -
1]].lista_e_pontual.append(copy.deepcopy(afluente))
    return list_tranfor
```

XII. funcoes/ferramentas.py:

```
# biblioteca a serem importadas
import streamlit as st
import pandas as pd
import plotly.graph_objects as go
from funcoes.equacoes import lista_hidr, menor_dist, modelagem_as_final,
func_hidraulica, menor_dist2, ajust_porc, modelagem_calib_final
import copy
import numpy as np
from concurrent.futures import ThreadPoolExecutor
from    funcoes.otimizador    import    gera_enxame_inicial,    dict_obtj,
melhores_resultados, pso
from functools import partial
from plotly.subplots import make_subplots
import random
import altair as alt

# OBJETOS
# Secções Tranversais
class SeccaoTransversal:
    def __init__(self, lat, long, comp, l_rio, ang_esq, rug, ang_dir):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.largura_rio = l_rio
        self.ang_esquerdo = ang_esq
        self.rugosidade_n = rug
        self.ang_direito = ang_dir


# Hidráulica
class Hidraulica:
    def __init__(self, lat, long, comp, vazao, rug, l_rio, altitude,
inclinacao, ang_esq,
                 ang_dir, prof, veloc, to, nivel_ag, froude):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.vazao = vazao
        self.rugosidade_n = rug
        self.largura_rio = l_rio
        self.altitude = altitude
        self.inclinacao = inclinacao
        self.ang_esquerdo = ang_esq
        self.ang_direito = ang_dir
        self.profundidade = prof
        self.velocidade = veloc
        self.tensao_c = to
        self.nivel_dagua = nivel_ag


# Coeficientes
class CoeficientesEntrada:
    def __init__(self, lat, long, comp, coeficientes, rio):
        self.latitude = lat
        self.longitude = long
```

```
        self.comprimento = comp
        self.coeficientes = coeficientes
        self.rio = rio

class Coeficientes:
    def __init__(self, temperatura, k_2_calculavel, k_2_max, k_2, k_1, s_d,
k_d, k_s, l_rd, k_so,
                 k_oa, k_an, k_nn, s_amon, k_spo, k_oi, k_b, r_o2_amon,
k_nit_od, s_pinorg):
        self.temperatura = temperatura
        self.k_2_calculavel = k_2_calculavel
        self.k_2_max = k_2_max
        self.k_2 = k_2
        self.k_1 = k_1
        self.s_d = s_d
        self.k_d = k_d
        self.k_s = k_s
        self.l_rd = l_rd
        self.k_so = k_so
        self.k_oa = k_oa
        self.k_an = k_an
        self.k_nn = k_nn
        self.s_amon = s_amon
        self.k_spo = k_spo
        self.k_oi = k_oi
        self.k_b = k_b
        self.r_o2_amon = r_o2_amon
        self.k_nit_od = k_nit_od
        self.s_pinorg = s_pinorg


# Concentrações
class Concentracoes:
    def __init__(self, od, dbo, no, n_amon, nitrato, nitrito, p_org, p_inorg,
p_total, e_coli):
        self.conc_od = od
        self.conc_dbo = dbo
        self.conc_no = no
        self.conc_n_amon = n_amon
        self.conc_nitrato = nitrato
        self.conc_nitrito = nitrito
        self.conc_p_org = p_org
        self.conc_p_inorg = p_inorg
        self.conc_p_total = p_total
        self.conc_e_coli = e_coli


# Dados reais
class DadosReais:
    def __init__(self, lat, long, comp, c_gerais, data_dr, vazao, descricao,
rio):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.concentracoes = c_gerais
        self.data_dr = data_dr
        self.vazao = vazao
        self.descricao = descricao
        self.rio = rio

# Saídas Pontuais
class SaidaPontual:
    def __init__(self, lat, long, comp, vazao, descricao, rio):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.vazao = vazao
```

```
        self.descricao = descricao
        self.rio = rio


# Entradas Pontuais
class EntradaPontual:
    def __init__(self, lat, long, comp, c_gerais, vazao, descricao, rio):
        self.latitude = lat
        self.longitude = long
        self.comprimento = comp
        self.concentracoes = c_gerais
        self.vazao = vazao
        self.descricao = descricao
        self.rio = rio


# Entradas Difusas
class EntradaDifusa:
    def  __init__(self,  lat_i,  long_i,  lat_f,  long_f,  comp_i,  comp_f,
c_gerais, vazao, descricao, rio):
        self.latitude_inicial = lat_i
        self.longitude_inicial = long_i
        self.comprimento_inicial = comp_i
        self.latitude_final = lat_f
        self.longitude_final = long_f
        self.comprimento_final = comp_f
        self.concentracoes = c_gerais
        self.descricao = descricao
        self.vazao = vazao
        self.rio = rio


# Resultados finais
class Quanti_Qualitativo:
    def __init__(self, hidraulica, concentracoes, coeficientes, rio):
        self.hidraulica = hidraulica
        self.concentracoes = concentracoes
        self.coeficientes = coeficientes
        self.rio = rio


# Classe Geral
class Geral:
    def __init__(self, s_transversal, e_coeficientes, s_pontual, e_pontual,
                 e_difusa, hidraulica, rio, discretizacao):
        self.lista_s_transversal = s_transversal
        self.lista_e_coeficientes = e_coeficientes
        self.lista_s_pontual = s_pontual
        self.lista_e_pontual = e_pontual
        self.lista_e_difusa = e_difusa
        self.lista_hidraulica = hidraulica
        self.rio = rio
        self.discretizacao = discretizacao


# Classe Geral Calib
class GeralCalib:
    def __init__(self, s_transversal, e_coeficientes, s_pontual, e_pontual,
                 e_difusa, dados_reais, hidraulica, rio, discretizacao):
        self.lista_s_transversal = s_transversal
        self.lista_e_coeficientes = e_coeficientes
        self.lista_s_pontual = s_pontual
        self.lista_e_pontual = e_pontual
        self.lista_e_difusa = e_difusa
        self.lista_dados_reais = dados_reais
        self.lista_hidraulica = hidraulica
```

```
        self.rio = rio
        self.discretizacao = discretizacao


def transformacao(lista_modelagem, lista_parametros, lista_coeficiente,
                  lista_contr_retir, list_name):

    list_tranfor = []


    for i in range(lista_modelagem['n_tb'] + 1):
        hidraulica = (lista_hidr(lista_parametros['l_lo'][i],
                                 lista_parametros['l_la'][i],
                                 lista_parametros['l_a'][i],
                                 lista_parametros['l_c'][i],
                                 lista_parametros['l_d'][i]))

        lat_dis = []
        long_dis = []
        comp_dis = []
        for llc in range(len(hidraulica)):
            lat_dis.append(hidraulica[llc].latitude)
            long_dis.append(hidraulica[llc].longitude)
            comp_dis.append(hidraulica[llc].comprimento)


        coeficiente = []
        for k in range(lista_coeficiente['l_n_p'][i] + 1):
            k2_calc = False if lista_modelagem['é_calc'] else True

            coef  =  Coeficientes(np.array(lista_coeficiente['l_coe'][i  +
1][k][1][lista_coeficiente['l_coe'][0].index('Temperatura (°C)')]),
                                  k2_calc, 0, 0, 0,
                                  0, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0)

            if lista_modelagem['é_calc']:
                coef.k_2    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('k2 (1/d)')])
            if      lista_modelagem['é_calc']      ==      False      and
lista_modelagem['m_od'] == True:
                coef.k_2_max = lista_coeficiente['l_coe'][i + 1][k][0][3]
            if lista_modelagem['m_od']:
                coef.k_1    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('k1 (1/d)')])
                coef.k_d    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('kd (1/d)')])
                coef.k_s    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('ks (1/d)')])
                coef.l_rd    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('lrd (gDBO5/m.d)')])
                coef.s_d    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('sd (1/d)')])
            if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] ==
True:
                coef.r_o2_amon  =  np.array(lista_coeficiente['l_coe'][i  +
1][k][1][lista_coeficiente['l_coe'][0].index('O2namon         (mgO2/mgNamon
oxid)')])
            if lista_modelagem['m_n']:
                coef.k_oa    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('koa (1/d)')])
                coef.k_so    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('kso (1/d)')])
                coef.k_an    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('kan (1/d)')])
```

```
                coef.s_amon   =   np.array(lista_coeficiente['l_coe'][i   +
1][k][1][lista_coeficiente['l_coe'][0].index('Snamon (g/m2.d)')])
                coef.k_nn    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('knn (1/d)')])
                coef.k_nit_od  =  np.array(lista_coeficiente['l_coe'][i  +
1][k][1][lista_coeficiente['l_coe'][0].index('knitr (1/d)')])
            if lista_modelagem['m_p']:
                coef.k_oi    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('koi (1/d)')])
                coef.k_spo    =   np.array(lista_coeficiente['l_coe'][i   +
1][k][1][lista_coeficiente['l_coe'][0].index('kspo (1/d)')])
                coef.s_pinorg  =  np.array(lista_coeficiente['l_coe'][i  +
1][k][1][lista_coeficiente['l_coe'][0].index('spinorg (1/d)')])
            if lista_modelagem['m_c']:
                coef.k_b    =    np.array(lista_coeficiente['l_coe'][i    +
1][k][1][lista_coeficiente['l_coe'][0].index('kb (1/d)')])

            comp_C = menor_dist(lat_dis, long_dis,
                                comp_dis,   lista_coeficiente['l_coe'][i   +
1][k][0][0],
                                lista_coeficiente['l_coe'][i + 1][k][0][1],
lista_coeficiente['l_coe'][i + 1][k][0][2])


coeficiente.append(copy.deepcopy(CoeficientesEntrada(lista_coeficiente['l_c
oe'][i + 1][k][0][0],

lista_coeficiente['l_coe'][i + 1][k][0][1],
                                                                    comp_C,

copy.deepcopy(coef),
                                                                    i)))


        geometria = []
        for j in range(lista_parametros['l_q_s'][i] + 1):
            comp_g = menor_dist(lat_dis, long_dis,
                                comp_dis,
lista_parametros['l_sc'][i][j][0],
                                lista_parametros['l_sc'][i][j][1],
lista_parametros['l_sc'][i][j][2])


geometria.append(copy.deepcopy(SeccaoTransversal(lista_parametros['l_sc'][i
][j][0],

lista_parametros['l_sc'][i][j][1],
                                                            comp_g,

lista_parametros['l_sc'][i][j][4],

lista_parametros['l_sc'][i][j][5],

lista_parametros['l_sc'][i][j][3],

lista_parametros['l_sc'][i][j][6])))



        conc = Concentracoes([0], [0], [0], [0], [0],
                             [0], [0], [0], [0], [0])
        # Entradas Pontuais
        if lista_modelagem['m_od']:
            id_od = list_name.index('OD (mg/L)')
            conc.conc_od = np.array(lista_parametros['l_v_i'][i][id_od])
            id_dbo = list_name.index('DBO (mg/L)')
            conc.conc_dbo = np.array(lista_parametros['l_v_i'][i][id_dbo])
```

```
        if lista_modelagem['m_n']:
            id_no = list_name.index('N-org (mg/L)')
            conc.conc_no = np.array(lista_parametros['l_v_i'][i][id_no])
            id_n_amon = list_name.index('N-amon (mg/L)')
            conc.conc_n_amon                                                =
np.array(lista_parametros['l_v_i'][i][id_n_amon])
            id_nitrito = list_name.index('N-nitri (mg/L)')
            conc.conc_nitrito                                               =
np.array(lista_parametros['l_v_i'][i][id_nitrito])
            id_nitrato = list_name.index('N-nitra (mg/L)')
            conc.conc_nitrato                                               =
np.array(lista_parametros['l_v_i'][i][id_nitrato])
        if lista_modelagem['m_p']:
            id_p_org = list_name.index('P-org (mg/L)')
            conc.conc_p_org                                                 =
np.array(lista_parametros['l_v_i'][i][id_p_org])
            id_p_inorg = list_name.index('P-inorg (mg/L)')
            conc.conc_p_inorg                                               =
np.array(lista_parametros['l_v_i'][i][id_p_inorg])
            conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
        if lista_modelagem['m_c']:
            id_e_coli = list_name.index('E-coli (NMP/100ml)')
            conc.conc_e_coli                                                =
np.array(lista_parametros['l_v_i'][i][id_e_coli])

        id = 1 if lista_modelagem['s_t'] else 0
        id_q = id

        conc_ep                                                             =
[copy.deepcopy(EntradaPontual(lista_parametros['l_la'][i][0],

lista_parametros['l_lo'][i][0],
                                                0.0,
                                                copy.deepcopy(conc),

np.array(lista_parametros['l_v_i'][i][id_q]),
                                                'Início',
                                                i))]

        id_q = 0
        # Entradas Pontuais
        if lista_contr_retir['l_q'][i][4]:

            for p in range(lista_contr_retir['l_q'][i][1]):

                if lista_modelagem['m_od']:
                    conc.conc_od                                            =
np.array(lista_contr_retir['l_ep'][i][p][1][id_od - id])
                    conc.conc_dbo                                           =
np.array(lista_contr_retir['l_ep'][i][p][1][id_dbo - id])
                if lista_modelagem['m_n']:
                    conc.conc_no                                            =
np.array(lista_contr_retir['l_ep'][i][p][1][id_no - id])
                    conc.conc_n_amon                                        =
np.array(lista_contr_retir['l_ep'][i][p][1][id_n_amon - id])
                    conc.conc_nitrito                                       =
np.array(lista_contr_retir['l_ep'][i][p][1][id_nitrito - id])
                    conc.conc_nitrato                                       =
np.array(lista_contr_retir['l_ep'][i][p][1][id_nitrato - id])
                if lista_modelagem['m_p']:
                    conc.conc_p_org                                         =
np.array(lista_contr_retir['l_ep'][i][p][1][id_p_org - id])
                    conc.conc_p_inorg                                       =
np.array(lista_contr_retir['l_ep'][i][p][1][id_p_inorg - id])
                    conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
                if lista_modelagem['m_c']:
```

```
                    conc.conc_e_coli                                        =
np.array(lista_contr_retir['l_ep'][i][p][1][id_e_coli - id])

                comp_ep = menor_dist(lat_dis, long_dis,
                                     comp_dis,
lista_contr_retir['l_ep'][i][p][0][1],
                                     lista_contr_retir['l_ep'][i][p][0][2],
lista_contr_retir['l_ep'][i][p][0][3])


conc_ep.append(copy.deepcopy(EntradaPontual(lista_contr_retir['l_ep'][i][p]
[0][1],

lista_contr_retir['l_ep'][i][p][0][2],
                                                            comp_ep,

copy.deepcopy(conc),

np.array(lista_contr_retir['l_ep'][i][p][1][id_q]),

lista_contr_retir['l_ep'][i][p][0][0],
                                                            i)))

        # Entradas Difusas
        conc_ed = []
        if lista_contr_retir['l_q'][i][5]:
            for d in range(lista_contr_retir['l_q'][i][2]):

                if lista_modelagem['m_od']:
                    conc.conc_od                                            =
np.array(lista_contr_retir['l_ed'][i][d][1][id_od - id])
                    conc.conc_dbo                                           =
np.array(lista_contr_retir['l_ed'][i][d][1][id_dbo - id])
                if lista_modelagem['m_n']:
                    conc.conc_no                                            =
np.array(lista_contr_retir['l_ed'][i][d][1][id_no - id])
                    conc.conc_n_amon                                        =
np.array(lista_contr_retir['l_ed'][i][d][1][id_n_amon - id])
                    conc.conc_nitrito                                       =
np.array(lista_contr_retir['l_ed'][i][d][1][id_nitrito - id])
                    conc.conc_nitrato                                       =
np.array(lista_contr_retir['l_ed'][i][d][1][id_nitrato - id])
                if lista_modelagem['m_p']:
                    conc.conc_p_org                                         =
np.array(lista_contr_retir['l_ed'][i][d][1][id_p_org - id])
                    conc.conc_p_inorg                                       =
np.array(lista_contr_retir['l_ed'][i][d][1][id_p_inorg - id])
                    conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
                if lista_modelagem['m_c']:
                    conc.conc_e_coli                                        =
np.array(lista_contr_retir['l_ed'][i][d][1][id_e_coli - id])


                comp_ed_i = menor_dist(lat_dis, long_dis,
                                       comp_dis,
lista_contr_retir['l_ed'][i][d][0][1],

lista_contr_retir['l_ed'][i][d][0][3],
lista_contr_retir['l_ed'][i][d][0][5])
                comp_ed_f = menor_dist(lat_dis, long_dis,
                                       comp_dis,
lista_contr_retir['l_ed'][i][d][0][2],

lista_contr_retir['l_ed'][i][d][0][4],
lista_contr_retir['l_ed'][i][d][0][6])
```

```
conc_ed.append(copy.deepcopy(EntradaDifusa(lista_contr_retir['l_ed'][i][d][
0][1],

lista_contr_retir['l_ed'][i][d][0][3],

lista_contr_retir['l_ed'][i][d][0][2],

lista_contr_retir['l_ed'][i][d][0][4],
                                                        comp_ed_i,
                                                        comp_ed_f,

copy.deepcopy(conc),

np.array(lista_contr_retir['l_ed'][i][d][1][id_q]),

lista_contr_retir['l_ed'][i][d][0][0],
                                                        i)))


        # Saídas Pontuais
        conc_r = []
        if lista_contr_retir['l_q'][i][3]:
            for r in range(lista_contr_retir['l_q'][i][0]):

                comp_r = menor_dist(lat_dis, long_dis,
                                    comp_dis,
lista_contr_retir['l_r'][i][r][0][1],
                                    lista_contr_retir['l_r'][i][r][0][2],
lista_contr_retir['l_r'][i][r][0][3])


conc_r.append(copy.deepcopy(SaidaPontual(lista_contr_retir['l_r'][i][r][0][
1],

lista_contr_retir['l_r'][i][r][0][2],
                                                    comp_r,

np.array(lista_contr_retir['l_r'][i][r][1][id_q]),

lista_contr_retir['l_r'][i][r][0][0],
                                                    i)))

        list_tranfor.append(copy.deepcopy(Geral(geometria,
                                                coeficiente,
                                                conc_r,
                                                conc_ep,
                                                conc_ed,
                                                hidraulica,
                                                i,

lista_parametros['l_d'][i])))

    return list_tranfor


def plot_map(maps, df_new, colun, title, inverse_b):

    maps.add_trace(
        go.Scattermapbox(lon=df_new['lon'], lat=df_new['lat'], name=title,
                         marker={"autocolorscale": False,
                                 "showscale":True, "size": 6, "opacity": 0.8,
                                 "color": colun, "colorscale": 'haline',
                                 "colorbar": dict(orientation='h')},
                         marker_reversescale=inverse_b))

    return
```

```
def plotar(n_tributarios, lista_modelagem, lidt_df, list_entr, labels, zona,
hemisferio, dias):

    list_tab = ['Gráficos do Rio Principal', 'Tabelas']
    str_lat = str(lidt_df[0]['latitude'][0])
    if str_lat != 'nan' and str_lat != 'None':
        list_tab.append('Representações geoespaciais')

    result_tabs = st.tabs(list_tab)

    with result_tabs[0]:
        if lista_modelagem['s_t']:
            df = lidt_df[0].loc[lidt_df[0]['data'] == dias[-1]]
            fig = make_subplots(specs=[[{"secondary_y": True}]])
            fig.update_layout(title_text="Concentrações   do   dia   "   +
str(dias[-1].date()),
                              title_font_color="teal")

        else:
            df = lidt_df[0]

            fig = make_subplots(specs=[[{"secondary_y": True}]])
            fig.update_layout(title_text="Concentrações",
                              title_font_color="teal")

        if lista_modelagem['m_od']:
            fig.add_trace(go.Scatter(x=df["comprimento"], y=df["conc_od"],
                              mode='lines',
                              name='OD'))

        if lista_modelagem['m_od']:
            fig.add_trace(go.Scatter(x=df["comprimento"], y=df["conc_dbo"],
                              mode='lines',
                              name='DBO'))

        if lista_modelagem['m_n']:
            fig.add_trace(go.Scatter(x=df["comprimento"], y=df["conc_no"],
                              mode='lines',
                              name='N-org'))
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_n_amon"],
                              mode='lines',
                              name='N-amon'))
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_nitrato"],
                              mode='lines',
                              name='N-nitri'))
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_nitrito"],
                              mode='lines',
                              name='N-nitra'))
        if lista_modelagem['m_p']:
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_p_org"],
                              mode='lines',
                              name='P-org'))
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_p_inorg"],
                              mode='lines',
                              name='P-inorg'))
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_p_total"],
                              mode='lines',
                              name='P total'))
        if lista_modelagem['m_c']:
```

```
            fig.add_trace(go.Scatter(x=df["comprimento"],
y=df["conc_e_coli"],
                                     mode='lines',
                                     name='E-coli', yaxis="y2"))

        fig.update_xaxes(minor=dict(ticklen=3, tickcolor="black"))
        fig.update_yaxes(minor=dict(ticklen=3, tickcolor="black"))
        fig.update_layout(
            legend=dict(orientation="h",
                        yanchor="bottom",
                        y=1,
                        xanchor="right",
                        x=0.95),
            xaxis=dict(title="Comprimento (m)",
                        ticklen=4,
                        tickcolor="black",
                        showgrid=True,
                        showline=True),
            yaxis=dict(title="OD, DBO, N ou P (mg/L)",
                        showline=True,
                        tickcolor="black",
                        ticklen=4,
            ),
            yaxis2=dict(title="E-coli (NMP/100ml)",
                        overlaying="y",
                        side="right",
                        showline=True,
                        tickcolor="black",
                        ticklen=4
            ),)

        for pt in range(len(list_entr) - 1):
            if str(list_entr[pt + 1].descricao) == 'nan'or str(list_entr[pt
+ 1].descricao) == 'None':

                fig.add_vline(
                    x=list_entr[pt + 1].comprimento,
                    line_width=2,
                    line_dash="dot",
                    line_color="LightSeaGreen",
                    )
            else:
                fig.add_vline(
                    x=list_entr[pt + 1].comprimento,
                    line_width=2,
                    line_dash="dot",
                    line_color="LightSeaGreen",
                    annotation_text=int(list_entr[pt + 1].descricao),
                    annotation_position="top left",
                    )



        # Plot!
        st.plotly_chart(fig, use_container_width=True)

        fig2 = make_subplots(specs=[[{"secondary_y": True}]])
        fig2.update_layout(title_text="Hidráulica",
                           title_font_color="teal")

        fig2.add_trace(go.Scatter(x=df["comprimento"], y=df["altitude"],
                           mode='lines',
                           name='Fundo do canal'))
        fig2.add_trace(go.Scatter(x=df["comprimento"], y=df["nivel_dagua"],
                           mode='lines',
                           name='Superfície livre'))
        fig2.add_trace(go.Scatter(x=df["comprimento"], y=df["vazao"],
```

```
                            mode='lines',
                            name='Vazão', yaxis="y2"))

        fig2.update_xaxes(minor=dict(ticklen=3, tickcolor="black"))
        fig2.update_yaxes(minor=dict(ticklen=3, tickcolor="black"))
        fig2.update_layout(
            legend=dict(orientation="h",
                        yanchor="bottom",
                        y=1,
                        xanchor="right",
                        x=0.5),
            xaxis=dict(title="Comprimento (m)",
                        ticklen=4,
                        tickcolor="black",
                        showgrid=True,
                        showline=True),
            yaxis=dict(title="Cota (m)",
                        showline=True,
                        tickcolor="black",
                        ticklen=4,
            ),
            yaxis2=dict(title="Vazão (m³/s)",
                        overlaying="y",
                        side="right",
                        showline=True,
                        tickcolor="black",
                        ticklen=4,
            ),)

        for pt in range(len(list_entr) - 1):

            if str(list_entr[pt + 1].descricao) == 'nan' or str(list_entr[pt
+ 1].descricao) == 'None':

                fig2.add_vline(
                    x=list_entr[pt + 1].comprimento,
                    line_width=2,
                    line_dash="dot",
                    line_color="LightSeaGreen",
                    )
            else:
                fig2.add_vline(
                    x=list_entr[pt + 1].comprimento,
                    line_width=2,
                    line_dash="dot",
                    line_color="LightSeaGreen",
                    annotation_text=int(list_entr[pt + 1].descricao),
                    annotation_position="top left",
                    )

        st.plotly_chart(fig2, use_container_width=True)

    with result_tabs[1]:
        for r1 in range(n_tributarios + 1):
            ex = st.expander('Resultados do ' + labels[r1])
            ex.write(lidt_df[r1])

        df_new = pd.concat(lidt_df)
        if len(lidt_df) > 1:
            ex2 = st.expander('Resultados Agrupado')
            ex2.write(df_new)

    if str_lat != 'nan' and str_lat != 'None':
        with result_tabs[2]:

            from pyproj import Proj
```

```
            if lista_modelagem['s_t']:
                df_new = df_new.loc[df_new['data'] == dias[-1]]
                titulo = 'Concentrações do dia ' + str(dias[-1])
            else:
                titulo = 'Concentrações'

            myProj = Proj('+proj=utm +zone=' + str(zona)
                          + ' +' + str(hemisferio) + ' +ellps=WGS84',
                          preserve_units=False)
            df_new['lon'],                 df_new['lat']                  =
myProj(df_new['longitude'].values,

df_new['latitude'].values,
                                                 inverse=True)


            maps = go.Figure()

            maps.update_layout(title_text=titulo, title_font_color="teal",
                               mapbox=dict(style='carto-positron',

center=dict(lat=df_new["lat"].mean(),

lon=df_new["lon"].mean()),
                                                                zoom=10),
                                   legend_itemclick="toggleothers",
                                   margin={"r":0,"t":0,"l":0,"b":0})


            if lista_modelagem['m_od']:
                plot_map(maps,  df_new,  df_new["conc_od"],  'OD  (mg/L)',
False)
            if lista_modelagem['m_dbo']:
                plot_map(maps,  df_new,  df_new["conc_dbo"],  'DBO  (mg/L)',
True)
            if lista_modelagem['m_p']:
                plot_map(maps, df_new, df_new["conc_p_org"], 'P-org (mg/L)',
True)
                plot_map(maps,  df_new,  df_new["conc_p_inorg"],  'P-inorg
(mg/L)', True)
                plot_map(maps,  df_new,  df_new["conc_p_total"],  'P  total
(mg/L)', True)
            if lista_modelagem['m_n']:
                plot_map(maps,  df_new,  df_new["conc_no"],  'N-org  (mg/L)',
True)
                plot_map(maps,   df_new,   df_new["conc_n_amon"],   'N-amon
(mg/L)', True)
                plot_map(maps,  df_new,  df_new["conc_nitrato"],  'N-nitri
(mg/L)', True)
                plot_map(maps,  df_new,  df_new["conc_nitrito"],  'N-nitra
(mg/L)', True)
            if lista_modelagem['m_c']:
                plot_map(maps,   df_new,   df_new["conc_e_coli"],   'E-coli
(NMP/100ml)', True)
            st.plotly_chart(maps, use_container_width=True)

    return


def         transformacao_calib(lista_modelagem,          lista_parametros,
lista_coeficiente,
                  lista_contr_retir, list_name, lista_dados_reais):

    list_tranfor = []
    marcador_calib = 0
```

```
    for i in range(lista_modelagem['n_tb'] + 1):

        hidraulica = (lista_hidr(lista_parametros['l_lo'][i],
                                 lista_parametros['l_la'][i],
                                 lista_parametros['l_a'][i],
                                 lista_parametros['l_c'][i],
                                 lista_parametros['l_d'][i]))

        lat_dis = []
        long_dis = []
        comp_dis = []
        for llc in range(len(hidraulica)):
            lat_dis.append(hidraulica[llc].latitude)
            long_dis.append(hidraulica[llc].longitude)
            comp_dis.append(hidraulica[llc].comprimento)


        coeficiente = []
        if lista_coeficiente['calb_trib'][i]:
            for k in range(lista_coeficiente['l_n_p'][i] + 1):
                coef_Minimo = lista_coeficiente['l_coe'][i + 1][1][0]
                coef_Maximo = lista_coeficiente['l_coe'][i + 1][1][1]
                nomes_coef_int = lista_coeficiente['l_coe'][0]
                coef                                                    =
Coeficientes([coef_Minimo[nomes_coef_int.index('Temperatura (°C)')],

coef_Maximo[nomes_coef_int.index('Temperatura (°C)')]],
                                     False, 0, 0, 0,
                                     0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0)

                if lista_modelagem['m_od']:
                    coef.k_2     =     [coef_Minimo[nomes_coef_int.index('k2
(1/d)')], coef_Maximo[nomes_coef_int.index('k2 (1/d)')]]
                    coef.k_1     =     [coef_Minimo[nomes_coef_int.index('k1
(1/d)')], coef_Maximo[nomes_coef_int.index('k1 (1/d)')]]
                    coef.k_d     =     [coef_Minimo[nomes_coef_int.index('kd
(1/d)')], coef_Maximo[nomes_coef_int.index('kd (1/d)')]]
                    coef.k_s     =     [coef_Minimo[nomes_coef_int.index('ks
(1/d)')], coef_Maximo[nomes_coef_int.index('ks (1/d)')]]
                    coef.l_rd    =    [coef_Minimo[nomes_coef_int.index('lrd
(gDBO5/m.d)')], coef_Maximo[nomes_coef_int.index('lrd (gDBO5/m.d)')]]
                    coef.s_d     =     [coef_Minimo[nomes_coef_int.index('sd
(1/d)')], coef_Maximo[nomes_coef_int.index('sd (1/d)')]]

                if      lista_modelagem['m_od']      ==      True      and
lista_modelagem['m_n'] == True:
                    coef.r_o2_amon                                      =
[coef_Minimo[nomes_coef_int.index('O2namon      (mgO2/mgNamon      oxid)')],
coef_Maximo[nomes_coef_int.index('O2namon (mgO2/mgNamon oxid)')]]
                if lista_modelagem['m_n']:
                    coef.k_oa    =    [coef_Minimo[nomes_coef_int.index('koa
(1/d)')], coef_Maximo[nomes_coef_int.index('koa (1/d)')]]
                    coef.k_so    =    [coef_Minimo[nomes_coef_int.index('kso
(1/d)')], coef_Maximo[nomes_coef_int.index('kso (1/d)')]]
                    coef.k_an    =    [coef_Minimo[nomes_coef_int.index('kan
(1/d)')], coef_Maximo[nomes_coef_int.index('kan (1/d)')]]
                    coef.s_amon = [coef_Minimo[nomes_coef_int.index('Snamon
(g/m2.d)')], coef_Maximo[nomes_coef_int.index('Snamon (g/m2.d)')]]
                    coef.k_nn    =    [coef_Minimo[nomes_coef_int.index('knn
(1/d)')], coef_Maximo[nomes_coef_int.index('knn (1/d)')]]
                    coef.k_nit_od                                       =
[coef_Minimo[nomes_coef_int.index('knitr                              (1/d)')],
coef_Maximo[nomes_coef_int.index('knitr (1/d)')]]
                if lista_modelagem['m_p']:
```

```
                    coef.k_oi    =    [coef_Minimo[nomes_coef_int.index('koi
(1/d)')], coef_Maximo[nomes_coef_int.index('koi (1/d)')]]
                    coef.k_spo   =   [coef_Minimo[nomes_coef_int.index('kspo
(1/d)')], coef_Maximo[nomes_coef_int.index('kspo (1/d)')]]
                    coef.s_pinorg                                          =
[coef_Minimo[nomes_coef_int.index('spinorg                          (1/d)')],
coef_Maximo[nomes_coef_int.index('spinorg (1/d)')]]
                if lista_modelagem['m_c']:
                    coef.k_b     =    [coef_Minimo[nomes_coef_int.index('kb
(1/d)')], coef_Maximo[nomes_coef_int.index('kb (1/d)')]]

                comp_C = menor_dist(lat_dis, long_dis,
                                    comp_dis,  lista_coeficiente['l_coe'][i
+ 1][0][k][0],
                                    lista_coeficiente['l_coe'][i          +
1][0][k][1], lista_coeficiente['l_coe'][i + 1][0][k][2])


coeficiente.append(copy.deepcopy(CoeficientesEntrada(lista_coeficiente['l_c
oe'][i + 1][0][k][0],

lista_coeficiente['l_coe'][i + 1][0][k][1],
                                                                    comp_C,

copy.deepcopy(coef),
                                                                    i)))
        else:
            coeficiente    =    [CoeficientesEntrada(None,    None,    0.0,
Coeficientes(0, False, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0), i)]


        geometria = []
        for j in range(lista_parametros['l_q_s'][i] + 1):
            comp_g = menor_dist(lat_dis, long_dis,
                                comp_dis,
lista_parametros['l_sc'][i][j][0],
                                lista_parametros['l_sc'][i][j][1],
lista_parametros['l_sc'][i][j][2])


geometria.append(copy.deepcopy(SeccaoTransversal(lista_parametros['l_sc'][i
][j][0],

lista_parametros['l_sc'][i][j][1],
                                                          comp_g,

lista_parametros['l_sc'][i][j][4],

lista_parametros['l_sc'][i][j][5],

lista_parametros['l_sc'][i][j][3],

lista_parametros['l_sc'][i][j][6])))



        conc = Concentracoes([0], [0], [0], [0], [0],
                             [0], [0], [0], [0], [0])

        # Entradas Pontuais
        if lista_modelagem['m_od']:
            id_od = list_name.index('OD (mg/L)')
            conc.conc_od = np.array(lista_parametros['l_v_i'][i][id_od])
            id_dbo = list_name.index('DBO (mg/L)')
            conc.conc_dbo = np.array(lista_parametros['l_v_i'][i][id_dbo])
        if lista_modelagem['m_n']:
```

```
            id_no = list_name.index('N-org (mg/L)')
            conc.conc_no = np.array(lista_parametros['l_v_i'][i][id_no])
            id_n_amon = list_name.index('N-amon (mg/L)')
            conc.conc_n_amon                                                =
np.array(lista_parametros['l_v_i'][i][id_n_amon])
            id_nitrito = list_name.index('N-nitri (mg/L)')
            conc.conc_nitrito                                               =
np.array(lista_parametros['l_v_i'][i][id_nitrito])
            id_nitrato = list_name.index('N-nitra (mg/L)')
            conc.conc_nitrato                                               =
np.array(lista_parametros['l_v_i'][i][id_nitrato])
        if lista_modelagem['m_p']:
            id_p_org = list_name.index('P-org (mg/L)')
            conc.conc_p_org                                                 =
np.array(lista_parametros['l_v_i'][i][id_p_org])
            id_p_inorg = list_name.index('P-inorg (mg/L)')
            conc.conc_p_inorg                                               =
np.array(lista_parametros['l_v_i'][i][id_p_inorg])
            conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
        if lista_modelagem['m_c']:
            id_e_coli = list_name.index('E-coli (NMP/100ml)')
            conc.conc_e_coli                                                =
np.array(lista_parametros['l_v_i'][i][id_e_coli])

        id = 1 if lista_modelagem['s_t'] else 0
        id_q = id

        conc_ep                                                             =
[copy.deepcopy(EntradaPontual(lista_parametros['l_la'][i][0],

lista_parametros['l_lo'][i][0],
                                                0.0,
                                                copy.deepcopy(conc),

np.array(lista_parametros['l_v_i'][i][id_q]),
                                                'Início',
                                                i))]

        id_q = 0
        # Entradas Pontuais
        if lista_contr_retir['l_q'][i][4]:

            for p in range(lista_contr_retir['l_q'][i][1]):


                if lista_modelagem['m_od']:
                    conc.conc_od                                            =
np.array(lista_contr_retir['l_ep'][i][p][1][id_od - id])
                    conc.conc_dbo                                           =
np.array(lista_contr_retir['l_ep'][i][p][1][id_dbo - id])
                if lista_modelagem['m_n']:
                    conc.conc_no                                            =
np.array(lista_contr_retir['l_ep'][i][p][1][id_no - id])
                    conc.conc_n_amon                                        =
np.array(lista_contr_retir['l_ep'][i][p][1][id_n_amon - id])
                    conc.conc_nitrito                                       =
np.array(lista_contr_retir['l_ep'][i][p][1][id_nitrito - id])
                    conc.conc_nitrato                                       =
np.array(lista_contr_retir['l_ep'][i][p][1][id_nitrato - id])
                if lista_modelagem['m_p']:
                    conc.conc_p_org                                         =
np.array(lista_contr_retir['l_ep'][i][p][1][id_p_org - id])
                    conc.conc_p_inorg                                       =
np.array(lista_contr_retir['l_ep'][i][p][1][id_p_inorg - id])
                    conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
                if lista_modelagem['m_c']:
```

```
                    conc.conc_e_coli                                        =
np.array(lista_contr_retir['l_ep'][i][p][1][id_e_coli - id])

                comp_ep = menor_dist(lat_dis, long_dis,
                                     comp_dis,
lista_contr_retir['l_ep'][i][p][0][1],
                                     lista_contr_retir['l_ep'][i][p][0][2],
lista_contr_retir['l_ep'][i][p][0][3])


conc_ep.append(copy.deepcopy(EntradaPontual(lista_contr_retir['l_ep'][i][p]
[0][1],

lista_contr_retir['l_ep'][i][p][0][2],
                                                            comp_ep,

copy.deepcopy(conc),

np.array(lista_contr_retir['l_ep'][i][p][1][id_q]),

lista_contr_retir['l_ep'][i][p][0][0],
                                                            i)))

        # Entradas Difusas
        conc_ed = []
        if lista_contr_retir['l_q'][i][5]:
            for d in range(lista_contr_retir['l_q'][i][2]):

                if lista_modelagem['m_od']:
                    conc.conc_od                                            =
np.array(lista_contr_retir['l_ed'][i][d][1][id_od - id])
                    conc.conc_dbo                                           =
np.array(lista_contr_retir['l_ed'][i][d][1][id_dbo - id])
                if lista_modelagem['m_n']:
                    conc.conc_no                                            =
np.array(lista_contr_retir['l_ed'][i][d][1][id_no - id])
                    conc.conc_n_amon                                        =
np.array(lista_contr_retir['l_ed'][i][d][1][id_n_amon - id])
                    conc.conc_nitrito                                       =
np.array(lista_contr_retir['l_ed'][i][d][1][id_nitrito - id])
                    conc.conc_nitrato                                       =
np.array(lista_contr_retir['l_ed'][i][d][1][id_nitrato - id])
                if lista_modelagem['m_p']:
                    conc.conc_p_org                                         =
np.array(lista_contr_retir['l_ed'][i][d][1][id_p_org - id])
                    conc.conc_p_inorg                                       =
np.array(lista_contr_retir['l_ed'][i][d][1][id_p_inorg - id])
                    conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
                if lista_modelagem['m_c']:
                    conc.conc_e_coli                                        =
np.array(lista_contr_retir['l_ed'][i][d][1][id_e_coli - id])


                comp_ed_i = menor_dist(lat_dis, long_dis,
                                       comp_dis,
lista_contr_retir['l_ed'][i][d][0][1],

lista_contr_retir['l_ed'][i][d][0][3],
lista_contr_retir['l_ed'][i][d][0][5])
                comp_ed_f = menor_dist(lat_dis, long_dis,
                                       comp_dis,
lista_contr_retir['l_ed'][i][d][0][2],

lista_contr_retir['l_ed'][i][d][0][4],
lista_contr_retir['l_ed'][i][d][0][6])
```

```
conc_ed.append(copy.deepcopy(EntradaDifusa(lista_contr_retir['l_ed'][i][d][
0][1],

lista_contr_retir['l_ed'][i][d][0][3],

lista_contr_retir['l_ed'][i][d][0][2],

lista_contr_retir['l_ed'][i][d][0][4],
                                                        comp_ed_i,
                                                        comp_ed_f,

copy.deepcopy(conc),

np.array(lista_contr_retir['l_ed'][i][d][1][id_q]),

lista_contr_retir['l_ed'][i][d][0][0],
                                                        i)))


        # Saídas Pontuais
        conc_r = []
        if lista_contr_retir['l_q'][i][3]:
            for r in range(lista_contr_retir['l_q'][i][0]):

                comp_r = menor_dist(lat_dis, long_dis,
                                    comp_dis,
lista_contr_retir['l_r'][i][r][0][1],
                                    lista_contr_retir['l_r'][i][r][0][2],
lista_contr_retir['l_r'][i][r][0][3])


conc_r.append(copy.deepcopy(SaidaPontual(lista_contr_retir['l_r'][i][r][0][
1],

lista_contr_retir['l_r'][i][r][0][2],
                                                    comp_r,

np.array(lista_contr_retir['l_r'][i][r][1][id_q]),

lista_contr_retir['l_r'][i][r][0][0],
                                                    i)))


        # Dados reais

        conc_dr = []
        if lista_coeficiente['calb_trib'][i]:
            for dr in range(lista_dados_reais['n_pontos'][marcador_calib]):

                if lista_modelagem['s_t']:
                    conj_data                                           =
list(lista_dados_reais['l_dr'][marcador_calib][dr][2][0])
                else:
                    conj_data = []
                if lista_modelagem['m_od']:
                    conc.conc_od                                        =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_od - id])
                    conc.conc_dbo                                       =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_dbo - id])
                if lista_modelagem['m_n']:
                    conc.conc_no                                        =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_no - id])
                    conc.conc_n_amon                                    =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_n_amon - id])
                    conc.conc_nitrito                                   =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_nitrito - id])
```

```
                    conc.conc_nitrato                                       =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_nitrato - id])
                if lista_modelagem['m_p']:
                    conc.conc_p_org                                         =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_p_org - id])
                    conc.conc_p_inorg                                       =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_p_inorg - id])
                    conc.conc_p_total = conc.conc_p_org + conc.conc_p_inorg
                if lista_modelagem['m_c']:
                    conc.conc_e_coli                                        =
np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_e_coli - id])

                comp_dr = menor_dist(lat_dis, long_dis,
                                     comp_dis,
lista_dados_reais['l_dr'][marcador_calib][dr][0][1],

lista_dados_reais['l_dr'][marcador_calib][dr][0][2],
lista_dados_reais['l_dr'][marcador_calib][dr][0][3])



conc_dr.append(copy.deepcopy(DadosReais(lista_dados_reais['l_dr'][marcador_
calib][dr][0][1],

lista_dados_reais['l_dr'][marcador_calib][dr][0][2],
                                                        comp_dr,

copy.deepcopy(conc),
                                                        conj_data,

np.array(lista_dados_reais['l_dr'][marcador_calib][dr][1][id_q]),

lista_dados_reais['l_dr'][marcador_calib][dr][0][0],
                                                        i)))

            marcador_calib += 1


        list_tranfor.append(copy.deepcopy(GeralCalib(geometria,
                                                     coeficiente,
                                                     conc_r,
                                                     conc_ep,
                                                     conc_ed,
                                                     conc_dr,
                                                     hidraulica,
                                                     i,

lista_parametros['l_d'][i])))

    return list_tranfor


def lista_hidraulica(lista_rio, ordem_modelagem, ordem_desague, ponto_af):
    lista_rio_too = copy.deepcopy(lista_rio)
    lista_hidr_final = []
    for ior in ordem_modelagem:

        dados = lista_rio_too[ior]
        lista_hidr_model      =      func_hidraulica(dados.lista_hidraulica,
dados.lista_s_pontual,
                                             dados.lista_e_pontual,
dados.lista_e_difusa,
                                             dados.lista_s_transversal,
dados.discretizacao)
        lista_hidr_final.append(lista_hidr_model)

        if ior != 0:
```

```
            comp  =  menor_dist2(lista_rio_too[ordem_desague[ior  -  1]],
ponto_af[ior - 1][1],
                               ponto_af[ior - 1][2], ponto_af[ior - 1][3])

            afluente = EntradaPontual(ponto_af[ior - 1][1], ponto_af[ior -
1][2],
                                      comp,    None,    lista_hidr_model[-
1].hidraulica.vazao, ponto_af[ior - 1][0], None)

            lista_rio_too[ordem_desague[ior                                -
1]].lista_e_pontual.append(copy.deepcopy(afluente))



    return lista_hidr_final


def plot_map(maps, df_new, colun, title, inverse_b):

    maps.add_trace(
        go.Scattermapbox(lon=df_new['lon'], lat=df_new['lat'], name=title,
                         marker={"autocolorscale": False,
                                 "showscale":True, "size": 6, "opacity": 0.8,
                                 "color": colun, "colorscale": 'haline',
                                 "colorbar": dict(orientation='h')},
                         marker_reversescale=inverse_b))

    return


def    ordem_desague_geral(ordem_desague,    n_trib,    lista_coeficiente,
lista_dados_reais, list_tranfor, ponto_af):
    if n_trib > 0:
        ordem_modelagem = []
        ordem_da = copy.deepcopy(ordem_desague)
        ordem = list(range(1, len(ordem_da) + 1))
        while len(ordem) > 0:
            remov = []
            for ia in range(len(ordem)):
                if (ordem[ia] in ordem_da) == False:
                    ordem_modelagem.append(ordem[ia])
                    remov.append(ia)
            remov.reverse()
            for irem in remov:
                ordem.pop(irem)
                ordem_da.pop(irem)
        ordem_modelagem.append(0)
    else:
        ordem_modelagem = [0]

    dict_ordem_grupo = {}

    for id_rio in ordem_modelagem:
        dict_ordem_grupo[str(id_rio)] = []

    for id_rio in ordem_modelagem:
        if lista_coeficiente['calb_trib'][id_rio] == True or id_rio == 0:

            dict_ordem_grupo[str(id_rio)].append(id_rio)

        else:
            id_efluente = ordem_desague[id_rio - 1]
            marcador = True
            while marcador:

                if lista_coeficiente['calb_trib'][id_efluente]:
                    dict_ordem_grupo[str(id_efluente)].append(id_rio)
```

```
                    marcador = False

                else:
                    id_efluente = ordem_desague[id_efluente - 1]

    ordem_final = []
    for id_rio in ordem_modelagem:
        if dict_ordem_grupo[str(id_rio)] != []:
            ordem_final.append(dict_ordem_grupo[str(id_rio)])

    list_ids_ordenados_coef = []
    list_lista_grupo_coef_final = []
    lista_conj_calib_final = []

    for conj_calb in ordem_final:
        comp_pts_coef = []
        comp_pts_dr = []
        dict_grupo_coef = {}
        comp_final_rio   =   list_tranfor[conj_calb[-1]].lista_hidraulica[-
1].comprimento

        # Organização da ordem dos coeficientes e dados reais do rio que será
calibrado
        for coef_id in range(lista_coeficiente['l_n_p'][conj_calb[-1]] + 1):
            dict_grupo_coef[str(coef_id)] = []
            comp_pts_coef.append(list_tranfor[conj_calb[-
1]].lista_e_coeficientes[coef_id].comprimento)
        pares_ordenados_coef          =          sorted(zip(comp_pts_coef,
range(len(comp_pts_coef))), key=lambda x: x[0])
        valores_ordenados_coef,             ids_ordenados_coef             =
zip(*pares_ordenados_coef)
        valores_ordenados_coef = list(valores_ordenados_coef)
        ids_ordenados_coef = list(ids_ordenados_coef)
        list_ids_ordenados_coef.append(list(ids_ordenados_coef))

        for coef_id in range(lista_dados_reais['n_pontos'][conj_calb[-1]]):
            comp_pts_dr.append(list_tranfor[conj_calb[-
1]].lista_dados_reais[coef_id].comprimento)
        pares_ordenados_dr             =            sorted(zip(comp_pts_dr,
range(len(comp_pts_dr))), key=lambda x: x[0])
        valores_ordenados_dr, ids_ordenados_dr = zip(*pares_ordenados_dr)
        valores_ordenados_dr = list(valores_ordenados_dr)
        ids_ordenados_dr = list(ids_ordenados_dr)

        # Verificação se o coeficiente possui dados reais para calibração
        lista_grupo_coef_final = []
        for id_coef in range(len(valores_ordenados_coef)):
            lista_grupo_coef = []
            if id_coef != (len(valores_ordenados_coef) - 1):
                comp_post = valores_ordenados_coef[id_coef + 1]
            else:
                comp_post = comp_final_rio

            for id_dr in range(len(valores_ordenados_dr)):

                if             valores_ordenados_coef[id_coef]             <
valores_ordenados_dr[id_dr] <= comp_post:
                    lista_grupo_coef.append(ids_ordenados_dr[id_dr])

            lista_grupo_coef_final.append(lista_grupo_coef)
        list_lista_grupo_coef_final.append(lista_grupo_coef_final)

        # Direcionamente dos afluentes para o conjunto que será calibrado
        for rio_conj in conj_calb:

            if   lista_coeficiente['calb_trib'][rio_conj]   ==   False   and
rio_conj != 0:
```

```
                comp  =  menor_dist2(list_tranfor[ordem_desague[rio_conj  -
1]], ponto_af[rio_conj - 1][1],
                                  ponto_af[rio_conj          -          1][2],
ponto_af[rio_conj - 1][3])

                id_efluente = ordem_desague[rio_conj - 1]

                marcador = True
                while marcador:
                    if lista_coeficiente['calb_trib'][id_efluente]:

                        for id_coef in range(len(valores_ordenados_coef)):

                            if id_coef != (len(valores_ordenados_coef) - 1):
                                comp_post  =  valores_ordenados_coef[id_coef
+ 1]
                            else:
                                comp_post = comp_final_rio

                            if  valores_ordenados_coef[id_coef]  <=  comp  <
comp_post:

dict_grupo_coef[str(ids_ordenados_coef[id_coef])].append(rio_conj)

                            marcador = False

                    else:
                        id_efluente = ordem_desague[id_efluente - 1]

        list_grupo_coef = []
        for id_coef_rio in ids_ordenados_coef:
            if dict_grupo_coef[str(id_coef_rio)] != []:
                list_grupo_coef.append(dict_grupo_coef[str(id_coef_rio)])

        lista_conj_calib_final.append(list_grupo_coef)

    return    ordem_modelagem,    ordem_final,    list_ids_ordenados_coef,
list_lista_grupo_coef_final, lista_conj_calib_final


def porcent(movel, fixo):
    pc = ((movel * 100) / fixo) - 100

    return pc


def  ordem_analise_sensibilidade(list_tranfor,  ponto_af,  lista_modelagem,
ordem_desague,
                                 lista_hidr_model, ordem_modelagem):


    an_ses = st.toggle("Gerar a análise de sensibilidade dos coeficientes,
com base o intervalo fornecido no Rio Principal.")

    if an_ses:
        par_as   =   st.expander(":grey[Ajuste   no   intervalo   de   busca
(Opcional).]")

        par_as.write('Porcentagem de busca na média de cada coeficiente:')

        porcentagem = 10
        porcentagem = par_as.slider('porcentagem', 1, 100, 10)


        coef_max_min                                                    =
copy.deepcopy(list_tranfor[0].lista_e_coeficientes[0].coeficientes)
```

```
        conj_coeficientes = Coeficientes(0, False, False, 0, 0, 0, 0,
                                         0, 0, 0, 0, 0, 0, 0,
                                         0, 0, 0, 0, 0, 0)

        media_coef = Coeficientes(0, False, False, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0, 0)

        lista_chaves = []
        labels_modelos = []
        if lista_modelagem['m_od']:
            conj_coeficientes.k_1,              media_coef.k_1              =
ajust_porc(coef_max_min.k_1, porcentagem)
            conj_coeficientes.k_2,              media_coef.k_2              =
ajust_porc(coef_max_min.k_2, porcentagem)
            conj_coeficientes.k_d,              media_coef.k_d              =
ajust_porc(coef_max_min.k_d, porcentagem)
            conj_coeficientes.k_s,              media_coef.k_s              =
ajust_porc(coef_max_min.k_s, porcentagem)
            conj_coeficientes.l_rd,             media_coef.l_rd             =
ajust_porc(coef_max_min.l_rd, porcentagem)
            conj_coeficientes.s_d,              media_coef.s_d              =
ajust_porc(coef_max_min.s_d, porcentagem)
            lista_chaves.extend(['k_1', 'k_2', 'k_d', 'k_s', 'l_rd', 's_d'])
            labels_modelos.extend(['Modelo OD', 'Modelo DBO'])
        if lista_modelagem['m_n']:
            conj_coeficientes.k_oa,             media_coef.k_oa             =
ajust_porc(coef_max_min.k_oa, porcentagem)
            conj_coeficientes.k_so,             media_coef.k_so             =
ajust_porc(coef_max_min.k_so, porcentagem)
            conj_coeficientes.k_an,             media_coef.k_an             =
ajust_porc(coef_max_min.k_an, porcentagem)
            conj_coeficientes.s_amon,           media_coef.s_amon           =
ajust_porc(coef_max_min.s_amon, porcentagem)
            conj_coeficientes.k_nn,             media_coef.k_nn             =
ajust_porc(coef_max_min.k_nn, porcentagem)
            conj_coeficientes.k_nit_od,         media_coef.k_nit_od         =
ajust_porc(coef_max_min.k_nit_od, porcentagem)
            lista_chaves.extend(['k_oa', 'k_so', 'k_an', 's_amon', 'k_nn',
'k_nit_od'])
            labels_modelos.extend(['Modelo NO', 'Modelo N-amon', 'Modelo N-
nitrato', 'Modelo N-nitrito'])
        if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] ==
True:
            conj_coeficientes.r_o2_amon,        media_coef.r_o2_amon        =
ajust_porc(coef_max_min.r_o2_amon, porcentagem)
            lista_chaves.append('r_o2_amon')
        if lista_modelagem['m_p']:
            conj_coeficientes.k_oi,             media_coef.k_oi             =
ajust_porc(coef_max_min.k_oi, porcentagem)
            conj_coeficientes.k_spo,            media_coef.k_spo            =
ajust_porc(coef_max_min.k_spo, porcentagem)
            conj_coeficientes.s_pinorg,         media_coef.s_pinorg         =
ajust_porc(coef_max_min.s_pinorg, porcentagem)
            lista_chaves.extend(['k_oi', 'k_spo', 's_pinorg'])
            labels_modelos.append('Modelo P-total')
        if lista_modelagem['m_c']:
            conj_coeficientes.k_b,              media_coef.k_b              =
ajust_porc(coef_max_min.k_b, porcentagem)
            lista_chaves.append('k_b')
            labels_modelos.append('Modelo Coliformes')
        conj_coeficientes.temperatura,        media_coef.temperatura        =
ajust_porc(coef_max_min.temperatura, porcentagem)
        lista_chaves.append('temperatura')

        result_media,          _        =      modelagem_as_final('temperatura',
lista_hidr_model,  media_coef, list_tranfor, lista_modelagem,
```

```
                                        ordem_modelagem,   ordem_desague,
ponto_af, None, media_coef)

        coef_od = ['k_1', 'k_2', 'k_d', 'k_s', 'l_rd', 's_d', 'r_o2_amon',
'k_oa', 'k_so', 'k_an', 's_amon', 'k_nn', 'k_nit_od', 'temperatura']
        coef_n  =  ['k_oa',  'k_so',  'k_an',  's_amon',  'k_nn',  'k_nit_od',
'temperatura']
        coef_p = ['k_oi', 'k_spo', 's_pinorg', 'temperatura']
        coef_cf = ['k_b', 'temperatura']

        variacao_parametro = [-porcentagem, porcentagem]
        lista_as_od = {'PORC': variacao_parametro}
        lista_as_dbo = {'PORC': variacao_parametro}
        lista_as_no = {'PORC': variacao_parametro}
        lista_as_n_amon = {'PORC': variacao_parametro}
        lista_as_nitrato = {'PORC': variacao_parametro}
        lista_as_nitrito = {'PORC': variacao_parametro}
        lista_as_p_total = {'PORC': variacao_parametro}
        lista_as_e_coli = {'PORC': variacao_parametro}


        for id_v in range(2):

            partial_function = partial(modelagem_as_final,
                                       lista_hidr_model=lista_hidr_model,
                                       lista_media_coef=media_coef,
                                       lista_rio=list_tranfor,
                                       lista_modelagem=lista_modelagem,
                                       ordem_modelagem=ordem_modelagem,
                                       ordem_desague=ordem_desague,
                                       ponto_af=ponto_af,
                                       i_coef=id_v,

conj_coeficientes=conj_coeficientes)

            with ThreadPoolExecutor() as executor:

                # futuros = {executor.submit(partial_function, valor): idx
for idx, valor in enumerate(lista_chaves)}
                resultados      =      list(executor.map(partial_function,
lista_chaves))

            for futuro in resultados:
                resultado, chave = futuro

                if id_v == 0:
                    if lista_modelagem['m_od'] and (chave in coef_od):
                        lista_as_od[chave]                                  =
[porcent(copy.deepcopy(resultado.conc_od[0]), result_media.conc_od[0])]
                        lista_as_dbo[chave]                                 =
[porcent(copy.deepcopy(resultado.conc_dbo[0]), result_media.conc_dbo[0])]
                    if (lista_modelagem['m_n']) and (chave in coef_n):
                        lista_as_no[chave]                                  =
[porcent(copy.deepcopy(resultado.conc_no[0]), result_media.conc_no[0])]
                        lista_as_n_amon[chave]                              =
[porcent(copy.deepcopy(resultado.conc_n_amon[0]),
result_media.conc_n_amon[0])]
                        lista_as_nitrato[chave]                             =
[porcent(copy.deepcopy(resultado.conc_nitrato[0]),
result_media.conc_nitrato[0])]
                        lista_as_nitrito[chave]                             =
[porcent(copy.deepcopy(resultado.conc_nitrito[0]),
result_media.conc_nitrito[0])]
                    if (lista_modelagem['m_p']) and (chave in coef_p):
                        lista_as_p_total[chave]                             =
[porcent(copy.deepcopy(resultado.conc_p_total[0]),
result_media.conc_p_total[0])]
```

```
                    if (lista_modelagem['m_c']) and (chave in coef_cf):
                        lista_as_e_coli[chave]                              =
[porcent(copy.deepcopy(resultado.conc_e_coli[0]),
result_media.conc_e_coli[0])]
                else:
                    if lista_modelagem['m_od'] and (chave in coef_od):

lista_as_od[chave].append(porcent(copy.deepcopy(resultado.conc_od[0]),
result_media.conc_od[0]))

lista_as_dbo[chave].append(porcent(copy.deepcopy(resultado.conc_dbo[0]),
result_media.conc_dbo[0]))
                    if (lista_modelagem['m_n']) and (chave in coef_n):

lista_as_no[chave].append(porcent(copy.deepcopy(resultado.conc_no[0]),
result_media.conc_no[0]))

lista_as_n_amon[chave].append(porcent(copy.deepcopy(resultado.conc_n_amon[0
]), result_media.conc_n_amon[0]))

lista_as_nitrato[chave].append(porcent(copy.deepcopy(resultado.conc_nitrato
[0]), result_media.conc_nitrato[0]))

lista_as_nitrito[chave].append(porcent(copy.deepcopy(resultado.conc_nitrito
[0]), result_media.conc_nitrito[0]))
                    if (lista_modelagem['m_p']) and (chave in coef_p):

lista_as_p_total[chave].append(porcent(copy.deepcopy(resultado.conc_p_total
[0]), result_media.conc_p_total[0]))
                    if (lista_modelagem['m_c']) and (chave in coef_cf):

lista_as_e_coli[chave].append(porcent(copy.deepcopy(resultado.conc_e_coli[0
]), result_media.conc_e_coli[0]))

        tabs_modelo = st.tabs(labels_modelos)
        if lista_modelagem['m_od']:
            tabs_modelo[labels_modelos.index('Modelo
OD')].line_chart(lista_as_od, x='PORC',
                             x_label="Variação do coeficiente (%)",
                             y_label="Variação concentração (%)",
                             height=500, width=450, use_container_width=False)
            tabs_modelo[labels_modelos.index('Modelo
DBO')].line_chart(lista_as_dbo, x='PORC',
                             x_label="Variação do coeficiente (%)",
                             y_label="Variação concentração (%)",
                             height=550, width=420, use_container_width=False)

        if lista_modelagem['m_n']:
            tabs_modelo[labels_modelos.index('Modelo
NO')].line_chart(lista_as_no, x='PORC',
                             x_label="Variação do coeficiente (%)",
                             y_label="Variação concentração (%)",
                             height=550, width=420, use_container_width=False)
            tabs_modelo[labels_modelos.index('Modelo                      N-
amon')].line_chart(lista_as_n_amon, x='PORC',
                             x_label="Variação do coeficiente (%)",
                             y_label="Variação concentração (%)",
                             height=550, width=420, use_container_width=False)
            tabs_modelo[labels_modelos.index('Modelo                      N-
nitrato')].line_chart(lista_as_nitrato, x='PORC',
                             x_label="Variação do coeficiente (%)",
                             y_label="Variação concentração (%)",
                             height=550, width=420, use_container_width=False)
            tabs_modelo[labels_modelos.index('Modelo                      N-
nitrito')].line_chart(lista_as_nitrito, x='PORC',
                             x_label="Variação do coeficiente (%)",
                             y_label="Variação concentração (%)",
```

```
                            height=550, width=420, use_container_width=False)

        if lista_modelagem['m_p']:
            tabs_modelo[labels_modelos.index('Modelo                        P-
total')].line_chart(lista_as_p_total, x='PORC',
                            x_label="Variação do coeficiente (%)",
                            y_label="Variação concentração (%)",
                            height=550, width=420, use_container_width=False)

        if lista_modelagem['m_c']:
            tabs_modelo[labels_modelos.index('Modelo
Coliformes')].line_chart(lista_as_e_coli, x='PORC',
                            x_label="Variação do coeficiente (%)",
                            y_label="Variação concentração (%)",
                            height=550, width=420, use_container_width=False)


    fixar_coef = Coeficientes(0, False, False, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0)
    seq_coef = []

    container2 = st.container(border=True)
    container2.markdown(":grey[Deseja fixar um ou mais coeficiente?]")
    col_1, col_2, col_3, col_4 = container2.columns(4)
    if lista_modelagem['m_od']:
        col_1.write('OD e DBO')
        b_k1 = col_1.toggle('k1')
        if b_k1:
            fixar_coef.k_1 = col_1.number_input('Fixar k1 em:')
        else:
            seq_coef.append('k_1')

        b_k2 = col_1.toggle('k2')
        if b_k2:
            fixar_coef.k_2 = col_1.number_input('Fixar k2 em:')
        else:
            seq_coef.append('k_2')

        b_kd = col_1.toggle('kd')
        if b_kd:
            fixar_coef.k_d = col_1.number_input('Fixar kd em:')
        else:
            seq_coef.append('k_d')

        b_ks = col_1.toggle('ks')
        if b_ks:
            fixar_coef.k_s = col_1.number_input('Fixar ks em:')
        else:
            seq_coef.append('k_s')

        b_lrd = col_1.toggle('lrd')
        if b_lrd:
            fixar_coef.l_rd = col_1.number_input('Fixar lrd em:')
        else:
            seq_coef.append('l_rd')

        b_sd = col_1.toggle('Sd')
        if b_sd:
            fixar_coef.s_d = col_1.number_input('Fixar sd em:')
        else:
            seq_coef.append('s_d')

    if lista_modelagem['m_n']:
        col_3.write('N')
        b_koa = col_3.toggle('koa')
        if b_koa:
```

```
            fixar_coef.k_oa = col_3.number_input('Fixar koa em:')
        else:
            seq_coef.append('k_oa')

        b_kso = col_3.toggle('kso')
        if b_kso:
            fixar_coef.k_so = col_3.number_input('Fixar kso em:')
        else:
            seq_coef.append('k_so')

        b_kan = col_3.toggle('kan')
        if b_kan:
            fixar_coef.k_an = col_3.number_input('Fixar kan em:')
        else:
            seq_coef.append('k_an')

        b_Snamon = col_3.toggle('Snamon')
        if b_Snamon:
            fixar_coef.s_amon = col_3.number_input('Fixar Snamon em:')
        else:
            seq_coef.append('s_amon')

        b_knn = col_3.toggle('knn')
        if b_knn:
            fixar_coef.k_nn = col_3.number_input('Fixar knn em:')
        else:
            seq_coef.append('k_nn')

        b_knitr = col_3.toggle('knitr')
        if b_knitr:
            fixar_coef.k_nit_od = col_3.number_input('Fixar knitr em:')
        else:
            seq_coef.append('k_nit_od')

    if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] == True:
        b_O2namon = col_3.toggle('O2namon')
        if b_O2namon:
            fixar_coef.r_o2_amon = col_3.number_input('Fixar O2namon em:')
        else:
            seq_coef.append('r_o2_amon')

    if lista_modelagem['m_p']:
        col_2.write('P')
        b_koi = col_2.toggle('koi')
        if b_koi:
            fixar_coef.k_oi = col_2.number_input('Fixar koi em:')
        else:
            seq_coef.append('k_oi')

        b_kspo = col_2.toggle('kspo')
        if b_kspo:
            fixar_coef.k_spo = col_2.number_input('Fixar kspo em:')
        else:
            seq_coef.append('k_spo')

        b_spinorg = col_2.toggle('Spinorg')
        if b_spinorg:
            fixar_coef.s_pinorg = col_2.number_input('Fixar Spinorg em:')
        else:
            seq_coef.append('s_pinorg')

    if lista_modelagem['m_c']:
        col_4.write('Coli-f')
        b_kb = col_4.toggle('kb')
        if b_kb:
            fixar_coef.k_b = col_4.number_input('Fixar kb em:')
        else:
```

```
            seq_coef.append('k_b')

    col_4.write('Geral')
    b_temp = col_4.toggle('Temperatura')
    if b_temp:
        fixar_coef.temperatura = col_4.number_input('Fixar Temperatura em:')
    else:
        seq_coef.append('temperatura')


    return fixar_coef, seq_coef



def    ajuste_trecho(list_tranfor,    lista_hidr_model,    list_ordem_coef,
marcador_conj_global, marcador_conj_interno,
                  ordem_final, ordem_modelagem):

    id_rio_calb = ordem_final[marcador_conj_global][-1]
    if len(list_ordem_coef[marcador_conj_global]) == 1:
        trecho_hidr                                                    =
copy.deepcopy(lista_hidr_model[ordem_modelagem.index(id_rio_calb)])


    else:
        id_coef_atual                                                  =
list_ordem_coef[marcador_conj_global][marcador_conj_interno]
        comp_atual                                                     =
list_tranfor[id_rio_calb].lista_e_coeficientes[id_coef_atual].comprimento

        if id_coef_atual == list_ordem_coef[marcador_conj_global][-1]:
            comp_post                                                  =
lista_hidr_model[ordem_modelagem.index(id_rio_calb)][-
1].hidraulica.comprimento
        else:
            comp_post                                                  =
list_tranfor[id_rio_calb].lista_e_coeficientes[list_ordem_coef[marcador_con
j_global][marcador_conj_interno + 1]].comprimento

        trecho_hidr = []

        for                              item                              in
range(len(lista_hidr_model[ordem_modelagem.index(id_rio_calb)])):
            if                          comp_atual                         <=
lista_hidr_model[ordem_modelagem.index(id_rio_calb)][item].hidraulica.compr
imento <= comp_post:

trecho_hidr.append(copy.deepcopy(lista_hidr_model[ordem_modelagem.index(id_
rio_calb)][item]))

    return trecho_hidr


def      estrutura_calibracao(list_tranfor,      fixar_coef,      seq_coef,
list_ordem_coef, list_ordem_dr,
                         marcador_conj_global,       marcador_conj_interno,
ordem_final,
                         lista_par_pos,      ponto_af,      lista_modelagem,
lista_hidr_model,
                         ordem_desague, trecho_hidr, dias):

    id_rio_calb = ordem_final[marcador_conj_global][-1]
    coef_max_min                                                       =
list_tranfor[id_rio_calb].lista_e_coeficientes[list_ordem_coef[marcador_con
j_global][marcador_conj_interno]].coeficientes
    ordem_dr = list_ordem_dr[marcador_conj_global][marcador_conj_interno]
```

```
    ordem_rio = ordem_final[marcador_conj_global]

    limite_repet = 20
    cont = 0
    g = 0
    ap_ant = 0
    tam_enxame = lista_par_pos[0]
    n_ger = lista_par_pos[1]
    w = lista_par_pos[2]  # Inércia
    c1 = lista_par_pos[3]  # Componente cognitiva (pessoal)
    c2 = lista_par_pos[4]  # Componente social (global)
    # w = 0.7
    # c1 = 2
    # c2 = 1.5
    # tam_enxame = 50
    # n_ger = 100

    random.seed()


    dic_apitoes_ger = {'apt': [], 'ger': []}
    enx,  m_ps_gl,  m_ap_gl  =  gera_enxame_inicial(tam_enxame,  seq_coef,
coef_max_min, list_tranfor, fixar_coef,
                                                    ordem_dr,         ponto_af,
lista_modelagem, lista_hidr_model,
                                                    ordem_desague,   ordem_rio,
trecho_hidr, dias)
    dic_apitoes_ger = dict_obtj(enx, g, dic_apitoes_ger)
    lista_media_aptidao = []
    lista_melhor_aptidao = []
    lista_melhor_seq = []
    media, cr = melhores_resultados(enx)
    lista_media_aptidao.append(media)
    lista_melhor_aptidao.append(cr.aptidao)
    lista_melhor_seq.append(cr.posicao)
    g = 1

    progress_text = "Operação em andamento. Por favor aguarde."
    my_bar = st.progress(0, text=progress_text)
    while cont <= limite_repet and g <= n_ger:
        my_bar.progress(int((g * 100)/ n_ger), text=progress_text)
        if g == 10:
            w  = 0.4

        enx, m_ps_gl, m_ap_gl = pso(enx, w, c1, c2, seq_coef, coef_max_min,
list_tranfor, fixar_coef,
                                                ordem_dr,             ponto_af,
lista_modelagem, lista_hidr_model,
                                                ordem_desague,       ordem_rio,
trecho_hidr, dias, m_ps_gl, m_ap_gl)
        dic_apitoes_ger = dict_obtj(enx, g, dic_apitoes_ger)
        media, cr = melhores_resultados(enx)
        lista_media_aptidao.append(media)
        lista_melhor_aptidao.append(cr.aptidao)
        lista_melhor_seq.append(cr.posicao)
        if ap_ant == cr.aptidao:
            cont += 1
        else:
            cont = 0
            ap_ant = cr.aptidao
        g += 1
    my_bar.empty()
    container3 = st.container(border=True)
    container3.markdown('''<h6 style='text-align: center;
                        color: LightSkyBlue;
                        '>Gráfico de evolução da otimização por PSO</h6>''',
                        unsafe_allow_html=True)
```

```
    container3.line_chart(lista_melhor_aptidao,         x_label='Evolução',
y_label='Aptidão', use_container_width=True)

    melhor_h = lista_melhor_seq[-1]

    text_sq = ':gray[Valores estimados: '
    atvo = False
    for tx in range(len(melhor_h)):
        if atvo:
            text_sq += ' | '
        text_sq += str(seq_coef[tx]) + ' = ' + str(melhor_h[tx])
        atvo = True
    text_sq += ']'
    container3.markdown(text_sq)


    precisao = 5

    lista_conc_final     =     modelagem_calib_final(melhor_h,     seq_coef,
lista_hidr_model,
                                                    list_tranfor, lista_modelagem,
ordem_rio,
                                                    ordem_desague,       ponto_af,
fixar_coef,
                                                    ordem_dr, trecho_hidr)

    list_sim_real = {}
    labels_modelos = []
    if lista_modelagem['m_od']:
        list_sim_real['conc_od'] = {'real':[], 'simulado': []}
        list_sim_real['conc_dbo'] = {'real':[], 'simulado': []}
        labels_modelos.extend(['Modelo OD', 'Modelo DBO'])
    if lista_modelagem['m_n']:
        list_sim_real['conc_no'] = {'real':[], 'simulado': []}
        list_sim_real['conc_n_amon'] = {'real':[], 'simulado': []}
        list_sim_real['conc_nitrito'] = {'real':[], 'simulado': []}
        labels_modelos.extend(['Modelo  NO',  'Modelo  N-amon',  'Modelo  N-
nitrito'])
    if lista_modelagem['m_p']:
        list_sim_real['conc_p_org'] = {'real':[], 'simulado': []}
        list_sim_real['conc_p_inorg'] = {'real':[], 'simulado': []}
        labels_modelos.extend(['Modelo P-org', 'Modelo P-inorg'])
    if lista_modelagem['m_c']:
        list_sim_real['conc_e_coli'] = {'real':[], 'simulado': []}
        labels_modelos.append('Modelo Coliformes')

    nome_modelos = list(list_sim_real.keys())

    list_pontos_trecho = []
    if lista_modelagem['s_t']:
        list_date = []
        for id_dias in range(len(dias)):
            for id_dr in range(len(ordem_dr)):
                lista_dador           =           list_tranfor[ordem_rio[-
1]].lista_dados_reais[ordem_dr[id_dr]]
                for id_dia_dr in range(len(lista_dador.data_dr)):
                    if               dias[id_dias].date()               ==
lista_dador.data_dr[id_dia_dr].date():

                        list_date.append(dias[id_dias].date())

list_pontos_trecho.append(copy.deepcopy(lista_dador.comprimento))
                        if lista_modelagem['m_od']:

list_sim_real['conc_od']['real'].append(copy.deepcopy(lista_dador.concentra
coes.conc_od[id_dia_dr]))
```

```
list_sim_real['conc_od']['simulado'].append(copy.deepcopy(lista_conc_final[
id_dr].conc_od[id_dias]))

list_sim_real['conc_dbo']['real'].append(copy.deepcopy(lista_dador.concentr
acoes.conc_dbo[id_dia_dr]))

list_sim_real['conc_dbo']['simulado'].append(copy.deepcopy(lista_conc_final
[id_dr].conc_dbo[id_dias]))
                        if lista_modelagem['m_n']:

list_sim_real['conc_no']['real'].append(copy.deepcopy(lista_dador.concentra
coes.conc_no[id_dia_dr]))

list_sim_real['conc_no']['simulado'].append(copy.deepcopy(lista_conc_final[
id_dr].conc_no[id_dias]))

list_sim_real['conc_n_amon']['real'].append(copy.deepcopy(lista_dador.conce
ntracoes.conc_n_amon[id_dia_dr]))

list_sim_real['conc_n_amon']['simulado'].append(copy.deepcopy(lista_conc_fi
nal[id_dr].conc_n_amon[id_dias]))

list_sim_real['conc_nitrito']['real'].append(copy.deepcopy(lista_dador.conc
entracoes.conc_nitrito[id_dia_dr]))

list_sim_real['conc_nitrito']['simulado'].append(copy.deepcopy(lista_conc_f
inal[id_dr].conc_nitrito[id_dias]))
                        if lista_modelagem['m_p']:

list_sim_real['conc_p_org']['real'].append(copy.deepcopy(lista_dador.concen
tracoes.conc_p_org[id_dia_dr]))

list_sim_real['conc_p_org']['simulado'].append(copy.deepcopy(lista_conc_fin
al[id_dr].conc_p_org[id_dias]))

list_sim_real['conc_p_inorg']['real'].append(copy.deepcopy(lista_dador.conc
entracoes.conc_p_inorg[id_dia_dr]))

list_sim_real['conc_p_inorg']['simulado'].append(copy.deepcopy(lista_conc_f
inal[id_dr].conc_p_inorg[id_dias]))
                        if lista_modelagem['m_c']:

list_sim_real['conc_e_coli']['real'].append(copy.deepcopy(lista_dador.conce
ntracoes.conc_e_coli[id_dia_dr]))

list_sim_real['conc_e_coli']['simulado'].append(copy.deepcopy(lista_conc_fi
nal[id_dr].conc_e_coli[id_dias]))
    else:
        for id_dr in range(len(ordem_dr)):
            lista_dador             =             list_tranfor[ordem_rio[-
1]].lista_dados_reais[ordem_dr[id_dr]]

list_pontos_trecho.append(copy.deepcopy(lista_dador.comprimento))
            if lista_modelagem['m_od']:

list_sim_real['conc_od']['real'].append(copy.deepcopy(lista_dador.concentra
coes.conc_od[0]))

list_sim_real['conc_od']['simulado'].append(copy.deepcopy(lista_conc_final[
id_dr].conc_od[0]))

list_sim_real['conc_dbo']['real'].append(copy.deepcopy(lista_dador.concentr
acoes.conc_dbo[0]))

list_sim_real['conc_dbo']['simulado'].append(copy.deepcopy(lista_conc_final
[id_dr].conc_dbo[0]))
```

```
            if lista_modelagem['m_n']:

list_sim_real['conc_no']['real'].append(copy.deepcopy(lista_dador.concentra
coes.conc_no[0]))

list_sim_real['conc_no']['simulado'].append(copy.deepcopy(lista_conc_final[
id_dr].conc_no[0]))

list_sim_real['conc_n_amon']['real'].append(copy.deepcopy(lista_dador.conce
ntracoes.conc_n_amon[0]))

list_sim_real['conc_n_amon']['simulado'].append(copy.deepcopy(lista_conc_fi
nal[id_dr].conc_n_amon[0]))

list_sim_real['conc_nitrito']['real'].append(copy.deepcopy(lista_dador.conc
entracoes.conc_nitrito[0]))

list_sim_real['conc_nitrito']['simulado'].append(copy.deepcopy(lista_conc_f
inal[id_dr].conc_nitrito[0]))
            if lista_modelagem['m_p']:

list_sim_real['conc_p_org']['real'].append(copy.deepcopy(lista_dador.concen
tracoes.conc_p_org[0]))

list_sim_real['conc_p_org']['simulado'].append(copy.deepcopy(lista_conc_fin
al[id_dr].conc_p_org[0]))

list_sim_real['conc_p_inorg']['real'].append(copy.deepcopy(lista_dador.conc
entracoes.conc_p_inorg[0]))

list_sim_real['conc_p_inorg']['simulado'].append(copy.deepcopy(lista_conc_f
inal[id_dr].conc_p_inorg[0]))
            if lista_modelagem['m_c']:

list_sim_real['conc_e_coli']['real'].append(copy.deepcopy(lista_dador.conce
ntracoes.conc_e_coli[0]))

list_sim_real['conc_e_coli']['simulado'].append(copy.deepcopy(lista_conc_fi
nal[id_dr].conc_e_coli[0]))

    tabs_modelo = st.tabs(labels_modelos)

    for idn in range(len(nome_modelos)):
        model = nome_modelos[idn]
        coln_1, conl_2 = tabs_modelo[idn].columns(2)
        soma_1 = 0
        soma_2 = 0
        if len(list_sim_real[model]['simulado']) > 0:
            for id in range(len(list_sim_real[model]['simulado'])):
                soma_1      +=      (list_sim_real[model]['real'][id]      -
list_sim_real[model]['simulado'][id])**2
                soma_2      +=      (list_sim_real[model]['real'][id]      -
np.mean(list_sim_real[model]['real']))**2

            if len(list_sim_real[model]['simulado']) == 1 or soma_2 == 0:
                f_1 = 1 - (soma_1 / 0.0001)
            else:
                f_1 = 1 - (soma_1 / soma_2)

            coln_1.markdown('Coeficiente    de    Nash-Sutcliffe:    '    +
str(np.round(f_1, precisao)))
        else:
            coln_1.markdown('ERRO - 9999')

        if lista_modelagem['s_t']:
```

```
            df_dict = {'Data': list_date, 'Comprimento': list_pontos_trecho,
'Real':              list_sim_real[model]['real'],              'Simulado':
list_sim_real[model]['simulado']}

        else:
            df_dict    =    {'Comprimento':    list_pontos_trecho,    'Real':
list_sim_real[model]['real'], 'Simulado': list_sim_real[model]['simulado']}

        coln_1.dataframe(df_dict)
        df = pd.DataFrame(df_dict)
        melted_df  =  df.melt(id_vars='Comprimento',  var_name='Resultado',
value_name='Valor')

        chart = alt.Chart(melted_df).mark_point(point=True).encode(
            x=alt.X('Comprimento:Q', title='Comprimento'),
            y=alt.Y('Valor:Q',                          title='Concentração',
scale=alt.Scale(zero=False)),
            color=alt.Color('Resultado:N',  legend=alt.Legend(orient='top',
title=None)),
            tooltip=['Comprimento', 'Resultado', 'Valor']
        ).properties()

        conl_2.altair_chart(chart, use_container_width=True)
    e_coeficientes = copy.deepcopy(fixar_coef)
    for id_c in range(len(seq_coef)):
        setattr(e_coeficientes, seq_coef[id_c], melhor_h[id_c])

    return e_coeficientes, lista_melhor_aptidao[-1]



def tabelar(lista_modelagem):

    coef_tabelados = Coeficientes(0, False, False, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0, 0)

    container4 = st.container(border=True)
    container4.markdown("O modelo adotará os seguintes valores:")
    col_1, col_2, col_3, col_4 = container4.columns(4)
    if lista_modelagem['m_od']:
        col_1.write('OD e DBO')
        coef_tabelados.k_1 = col_1.number_input('Fixar k1 em:', value=0.19)
        coef_tabelados.k_2 = col_1.number_input('Fixar k2 em:', value=40)
        coef_tabelados.k_d = col_1.number_input('Fixar kd em:', value=0.4)
        coef_tabelados.k_s = col_1.number_input('Fixar ks em:', value=2.0)
        coef_tabelados.l_rd = col_1.number_input('Fixar lrd em:', value=0.0)
        coef_tabelados.s_d = col_1.number_input('Fixar sd em:', value=2.86)

    if lista_modelagem['m_n']:
        col_3.write('N')
        coef_tabelados.k_oa    =    col_3.number_input('Fixar    koa    em:',
value=0.001)
        coef_tabelados.k_so    =    col_3.number_input('Fixar    kso    em:',
value=0.07)
        coef_tabelados.k_an = col_3.number_input('Fixar kan em:', value=0.4)
        coef_tabelados.s_amon   =   col_3.number_input('Fixar   Snamon   em:',
value=0.0)
        coef_tabelados.k_nn    =    col_3.number_input('Fixar    knn    em:',
value=0.05)
        coef_tabelados.k_nit_od  =  col_3.number_input('Fixar  knitr  em:',
value=0.6)

    if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] == True:
        coef_tabelados.r_o2_amon = col_3.number_input('Fixar O2namon em:',
value=3.2)
```

```
    if lista_modelagem['m_p']:
        col_2.write('P')
        coef_tabelados.k_oi = col_2.number_input('Fixar koi em:', value=0.6)
        coef_tabelados.k_spo   =   col_2.number_input('Fixar   kspo   em:',
value=0.0)
        coef_tabelados.s_pinorg  =  col_2.number_input('Fixar  Spinorg  em:',
value=3.4)

    if lista_modelagem['m_c']:
        col_4.write('Coli-f')
        coef_tabelados.k_b = col_4.number_input('Fixar kb em:', value=0.2)

    col_4.write('Geral')
    coef_tabelados.temperatura = col_4.number_input('Fixar Temperatura em:',
value=20.0)

    return coef_tabelados
```

XIII. funcoes/otimizador.py:

```
# biblioteca a serem importadas
from concurrent.futures import ThreadPoolExecutor
from functools import partial
import numpy as np
import random
import copy
from funcoes.equacoes import modelagem_calib_final

# Particula
class Particula:
      def  __init__(self,  posicao,  aptidao,  velocidade,  melhor_pos_ind,
melhor_apt_ind):
        self.posicao = posicao
        self.aptidao = aptidao
        self.velocidade = velocidade
        self.melhor_pos_ind = melhor_pos_ind
        self.melhor_apt_ind = melhor_apt_ind

# Aptidão
def calc_aptidao(seq_coef, lista_lista_pos, list_tranfor, fixar_coef,
                 ordem_dr, ponto_af, lista_modelagem, lista_hidr_model,
                 ordem_desague, ordem_rio, trecho_hidr, dias):


    precisao = 5
    lista_aptidoes = []

    partial_function = partial(modelagem_calib_final,
                               seq_coef=seq_coef,
                               lista_hidr_model=lista_hidr_model,
                               list_tranfor=list_tranfor,
                               lista_modelagem=lista_modelagem,
                               ordem_rio=ordem_rio,
                               ordem_desague=ordem_desague,
                               ponto_af=ponto_af,
                               fixar_coef=fixar_coef,
                               ordem_dr=ordem_dr,
                               trecho_hidr=trecho_hidr)

    with ThreadPoolExecutor() as executor:

        resultados = list(executor.map(partial_function, lista_lista_pos))

    for lista_conc_final in resultados:
        list_sim_real = {}
        if lista_modelagem['m_od']:
```

```
            list_sim_real['conc_od'] = {'real':[], 'simulado': []}
            list_sim_real['conc_dbo'] = {'real':[], 'simulado': []}
        if lista_modelagem['m_n']:
            list_sim_real['conc_no'] = {'real':[], 'simulado': []}
            list_sim_real['conc_n_amon'] = {'real':[], 'simulado': []}
            list_sim_real['conc_nitrito'] = {'real':[], 'simulado': []}
        if lista_modelagem['m_p']:
            list_sim_real['conc_p_org'] = {'real':[], 'simulado': []}
            list_sim_real['conc_p_inorg'] = {'real':[], 'simulado': []}
        if lista_modelagem['m_c']:
            list_sim_real['conc_e_coli'] = {'real':[], 'simulado': []}

        nome_modelos = list(list_sim_real.keys())


        if lista_modelagem['s_t']:
            for id_dias in range(len(dias)):
                for id_dr in range(len(ordem_dr)):
                                    lista_dador   =  list_tranfor[ordem_rio[-
1]].lista_dados_reais[ordem_dr[id_dr]]
                    for id_dia_dr in range(len(lista_dador.data_dr)):

                                                if   dias[id_dias].date()   ==
lista_dador.data_dr[id_dia_dr].date():
                                if lista_modelagem['m_od']:
                                    list_sim_real['conc_od']['real'].append(cop
y.deepcopy(lista_dador.concentracoes.conc_od[id_dia_dr]))
                                    list_sim_real['conc_od']['simulado'].append
(copy.deepcopy(lista_conc_final[id_dr].conc_od[id_dias]))
                                    list_sim_real['conc_dbo']['real'].append(co
py.deepcopy(lista_dador.concentracoes.conc_dbo[id_dia_dr]))
                                    list_sim_real['conc_dbo']['simulado'].appen
d(copy.deepcopy(lista_conc_final[id_dr].conc_dbo[id_dias]))
                                if lista_modelagem['m_n']:
                                    list_sim_real['conc_no']['real'].append(cop
y.deepcopy(lista_dador.concentracoes.conc_no[id_dia_dr]))
                                    list_sim_real['conc_no']['simulado'].append
(copy.deepcopy(lista_conc_final[id_dr].conc_no[id_dias]))
                                    list_sim_real['conc_n_amon']['real'].append
(copy.deepcopy(lista_dador.concentracoes.conc_n_amon[id_dia_dr]))
                                    list_sim_real['conc_n_amon']['simulado'].ap
pend(copy.deepcopy(lista_conc_final[id_dr].conc_n_amon[id_dias]))
                                    list_sim_real['conc_nitrito']['real'].appen
d(copy.deepcopy(lista_dador.concentracoes.conc_nitrito[id_dia_dr]))
                                    list_sim_real['conc_nitrito']['simulado'].a
ppend(copy.deepcopy(lista_conc_final[id_dr].conc_nitrito[id_dias]))
                                if lista_modelagem['m_p']:
                                    list_sim_real['conc_p_org']['real'].append(
copy.deepcopy(lista_dador.concentracoes.conc_p_org[id_dia_dr]))
                                    list_sim_real['conc_p_org']['simulado'].app
end(copy.deepcopy(lista_conc_final[id_dr].conc_p_org[id_dias]))
                                    list_sim_real['conc_p_inorg']['real'].appen
d(copy.deepcopy(lista_dador.concentracoes.conc_p_inorg[id_dia_dr]))
                                    list_sim_real['conc_p_inorg']['simulado'].a
ppend(copy.deepcopy(lista_conc_final[id_dr].conc_p_inorg[id_dias]))
                                if lista_modelagem['m_c']:
                                    list_sim_real['conc_e_coli']['real'].append
(copy.deepcopy(lista_dador.concentracoes.conc_e_coli[id_dia_dr]))
                                    list_sim_real['conc_e_coli']['simulado'].ap
pend(copy.deepcopy(lista_conc_final[id_dr].conc_e_coli[id_dias]))
        else:

            for id_dr in range(len(ordem_dr)):
                                    lista_dador   =   list_tranfor[ordem_rio[-
1]].lista_dados_reais[ordem_dr[id_dr]]
                if lista_modelagem['m_od']:
```

```
                    list_sim_real['conc_od']['real'].append(copy.deepcopy(l
ista_dador.concentracoes.conc_od[0]))
                    list_sim_real['conc_od']['simulado'].append(copy.deepco
py(lista_conc_final[id_dr].conc_od[0]))
                    list_sim_real['conc_dbo']['real'].append(copy.deepcopy(
lista_dador.concentracoes.conc_dbo[0]))
                    list_sim_real['conc_dbo']['simulado'].append(copy.deepc
opy(lista_conc_final[id_dr].conc_dbo[0]))
                if lista_modelagem['m_n']:
                    list_sim_real['conc_no']['real'].append(copy.deepcopy(l
ista_dador.concentracoes.conc_no[0]))
                    list_sim_real['conc_no']['simulado'].append(copy.deepco
py(lista_conc_final[id_dr].conc_no[0]))
                    list_sim_real['conc_n_amon']['real'].append(copy.deepco
py(lista_dador.concentracoes.conc_n_amon[0]))
                    list_sim_real['conc_n_amon']['simulado'].append(copy.de
epcopy(lista_conc_final[id_dr].conc_n_amon[0]))
                    list_sim_real['conc_nitrito']['real'].append(copy.deepc
opy(lista_dador.concentracoes.conc_nitrito[0]))
                    list_sim_real['conc_nitrito']['simulado'].append(copy.d
eepcopy(lista_conc_final[id_dr].conc_nitrito[0]))
                if lista_modelagem['m_p']:
                    list_sim_real['conc_p_org']['real'].append(copy.deepcop
y(lista_dador.concentracoes.conc_p_org[0]))
                    list_sim_real['conc_p_org']['simulado'].append(copy.dee
pcopy(lista_conc_final[id_dr].conc_p_org[0]))
                    list_sim_real['conc_p_inorg']['real'].append(copy.deepc
opy(lista_dador.concentracoes.conc_p_inorg[0]))
                    list_sim_real['conc_p_inorg']['simulado'].append(copy.d
eepcopy(lista_conc_final[id_dr].conc_p_inorg[0]))
                if lista_modelagem['m_c']:
                    list_sim_real['conc_e_coli']['real'].append(copy.deepco
py(lista_dador.concentracoes.conc_e_coli[0]))
                    list_sim_real['conc_e_coli']['simulado'].append(copy.de
epcopy(lista_conc_final[id_dr].conc_e_coli[0]))

        lista_coef_Nash_Sutcliffe = []
        for model in nome_modelos:
            soma_1 = 0
            soma_2 = 0
            if len(list_sim_real[model]['simulado']) > 0:
                for id in range(len(list_sim_real[model]['simulado'])):
                            soma_1  +=  (list_sim_real[model]['real'][id]  -
list_sim_real[model]['simulado'][id])**2
                            soma_2  +=  (list_sim_real[model]['real'][id]  -
np.mean(list_sim_real[model]['real']))**2

                 if len(list_sim_real[model]['simulado']) == 1 or soma_2 ==
0:
                    f_1 = 1 - (soma_1 / 0.0001)
                else:
                    f_1 = 1 - (soma_1 / soma_2)

                lista_coef_Nash_Sutcliffe.append(abs(1 - f_1))
            else:
                lista_coef_Nash_Sutcliffe = [9999]

        lista_coef_Nash_Sutcliffe = np.array(lista_coef_Nash_Sutcliffe)

        lista_aptidoes.append(round(np.sqrt(np.mean(lista_coef_Nash_Sutclif
fe**2)), precisao))

    return lista_aptidoes

# Gerando enxame
def  gera_enxame_inicial(tam_pop,  seq_coef,  coef_max_min,  list_tranfor,
fixar_coef,
```

```
                      ordem_dr, ponto_af, lista_modelagem, lista_hidr_model,
                        ordem_desague, ordem_rio, trecho_hidr, dias):

    lista_enxame = []
    lista_lista_pos = []
    for p in range(tam_pop):
        lista_pos = []
        lista_vel = []
        for nome_coef in seq_coef:
            var = getattr(coef_max_min, nome_coef)
            lista_pos.append(random.uniform(var[0], var[1]))
            lista_vel.append(random.uniform(-1, 1))
        lista_lista_pos.append(lista_pos)

        particula = Particula(copy.deepcopy(lista_pos), None,
                                                    copy.deepcopy(lista_vel),
copy.deepcopy(lista_pos),
                              None)

        lista_enxame.append(particula)

     list_aptidao = calc_aptidao(seq_coef, lista_lista_pos, list_tranfor,
fixar_coef,
                                    ordem_dr, ponto_af, lista_modelagem,
lista_hidr_model,
                            ordem_desague, ordem_rio, trecho_hidr, dias)

    for p in range(tam_pop):
        aptidao = list_aptidao[p]
        lista_enxame[p].aptidao = aptidao
        lista_enxame[p].melhor_apt_ind = aptidao

        if p == 0:
            melhor_posicao_geral = copy.deepcopy(lista_pos)
            melhor_aptidao_geral = aptidao

        elif aptidao < melhor_aptidao_geral:
            melhor_posicao_geral = copy.deepcopy(lista_pos)
            melhor_aptidao_geral = aptidao

    return lista_enxame, melhor_posicao_geral, melhor_aptidao_geral

# Melhores resultados
def melhores_resultados(populacao):
    soma_aptidao = 0
    melhor_cr = populacao[0]
    for cr in populacao:
        soma_aptidao += cr.aptidao
        if cr.aptidao < melhor_cr.aptidao:
            melhor_cr = cr
    media_aptidao = soma_aptidao / len(populacao)
    return media_aptidao, melhor_cr

def dict_obtj(enxame, ger, list_dict_apt):
    for enx_i in enxame:
        list_dict_apt['apt'].append(enx_i.aptidao)
        list_dict_apt['ger'].append(ger)
    return list_dict_apt

# PSO
def pso(enxame, w, c1, c2, seq_coef, coef_max_min, list_tranfor, fixar_coef,
        ordem_dr, ponto_af, lista_modelagem, lista_hidr_model,
         ordem_desague, ordem_rio, trecho_hidr, dias, melhor_posicao_geral,
melhor_aptidao_geral):
    precisao = 5
    # Atualizar o melhor geral
    for i in range(len(enxame)):
```

```
        if enxame[i].aptidao < melhor_aptidao_geral:
            melhor_posicao_geral = enxame[i].posicao
            melhor_aptidao_geral = enxame[i].aptidao

    count = 0
    list_lista_posicao = []
    list_lista_veloc = []
    for j in range(len(enxame)):
        var = enxame[j]
        lista_veloc = []
        lista_posicao = []
        for k in range(len(var.posicao)):
            r1, r2 = random.random(), random.random()
            vel = (w * var.velocidade[k]
                ) + (c1 * r1 * (var.melhor_pos_ind[k] - var.posicao[k])
                ) + (c2 * r2 * (melhor_posicao_geral[k] - var.posicao[k]))

            pos = round(var.posicao[k] + vel, precisao)
            max_min = getattr(coef_max_min, seq_coef[k])
            pos = max(max_min[0], min(pos, max_min[1]))
            lista_veloc.append(vel)
            lista_posicao.append(pos)
        list_lista_posicao.append(lista_posicao)
        list_lista_veloc.append(lista_veloc)

       lista_aptidao_atual   =   calc_aptidao(seq_coef,   list_lista_posicao,
list_tranfor, fixar_coef,
                                        ordem_dr, ponto_af, lista_modelagem,
lista_hidr_model,
                                     ordem_desague, ordem_rio, trecho_hidr,
dias)

    for j in range(len(enxame)):
        var = enxame[j]
        aptidao_atual = lista_aptidao_atual[j]
        lista_veloc = list_lista_veloc[j]
        lista_posicao = list_lista_posicao[j]
        if var.aptidao == melhor_aptidao_geral and count == 0:
            count = 1
            if aptidao_atual > var.aptidao:
                lista_veloc = var.velocidade
                lista_posicao = var.posicao
                aptidao_atual = var.aptidao

        var.velocidade = lista_veloc
        var.posicao = lista_posicao
        if aptidao_atual < var.melhor_apt_ind:
            var.melhor_pos_ind = var.posicao
            var.melhor_apt_ind = aptidao_atual
        var.aptidao = aptidao_atual

    lista_pso = copy.deepcopy(enxame)
    return lista_pso, melhor_posicao_geral, melhor_aptidao_geral
```

XIV. funcoes/layout_modelagem.py:

```
# biblioteca a serem importadas
import streamlit as st
import pandas as pd
import numpy as np
import copy

def inicio(paramentro):

    st.divider()
    dados_anterior = st.checkbox("Preenchimento automático.")
```

```
    data = False
    if dados_anterior:
                uploaded_json   =   st.file_uploader("Submeter   o   arquivo
'DadosEntrada.json'"
                                                     + " construído anteriormente:",
                                                     type=["json"])
        if uploaded_json is not None:
            from json import load
            data_dict = load(uploaded_json)
            data = data_dict['dados'][0]
            paramentro = data['Paramêtros']

        else:
                        st.markdown("""❗  Obs.:  Evite  alterar  o  arquivo
'DadosEntrada.json'
             utilizando o Excel. Se os dados não preencherem automaticamente,
              recomenda-se preencher de forma manual novamente.""")
    else:
        st.warning('''Se for a sua primeira vez no Qualitool 2.0,
                       você terá que preencher os dados de forma manual.''',
                       icon="❕")
    st.divider()

    col_1, col_2 = st.columns(2)
    bar = st.sidebar
    bar.warning('''Atenção! Em caso de atualização da tela, os dados
                já preenchidos serão perdidos e precisarão ser
                inseridos novamente.''',
                  icon="⚠️")

    col_1.markdown('Qual(s) parâmetro(s) modelar?')
    col_1_1, col_1_2 = col_1.columns(2)
    disab = False
    if data != False:
        disab = True
        k2_calc_ask = paramentro['é_calc']
    else:

        k2_calc_ask = False
     modelar_od  =  col_1_1.checkbox('OD  e  DBO',  value=paramentro['m_od'],
disabled=disab)
    modelar_dbo = modelar_od
      modelar_n  =  col_1_1.checkbox('Nitrogênio',  value=paramentro['m_n'],
disabled=disab)
       modelar_p   =   col_1_2.checkbox('Fósforo',   value=paramentro['m_p'],
disabled=disab)
    modelar_colif = col_1_2.checkbox('Coliformes', value=paramentro['m_c'],
disabled=disab)

    col_2.markdown('Configurações gerais:')
         serie_tempo    =    col_2.toggle('Ativar    avaliação    temporal.',
value=paramentro['s_t'], disabled=disab)

    n_tributarios = col_2.number_input(
        'Quantidade de tributários modeláveis:', min_value=0,
        value=paramentro['n_tb'], disabled=disab)


     lista_modelagem  =  {'m_od':  modelar_od,  'm_dbo':  modelar_dbo,  'm_n':
modelar_n,
                          'm_p': modelar_p, 'm_c': modelar_colif, 'n_tb':
n_tributarios,
                      's_t': serie_tempo, 'é_calc': k2_calc_ask}

    lista_tabs = ["tab0"]
    labels = ["Rio principal"]
    for n_trib in range(n_tributarios):
```

```
        lista_tabs.append("tab" + str(n_trib))
        labels.append("Tributário " + str(n_trib + 1))
    lista_tabs = st.tabs(labels)

    return (lista_modelagem, data, labels, lista_tabs)

######################### DADOS INICIAIS ##########################
def    dados_iniciais(data,    lista_modelagem,    n_tributarios,    labels,
lista_tabs):
    # DADOS DE ENTRADA INICIAIS
    ponto_af = []
    list_desague = []
    list_valor_i = []
    list_discretizacao = []
    list_comprimento = []
    list_longitude = []
    list_latitude = []
    list_altitude = []
    list_secaotrav = []
    list_qnt_secaotrav = []
    zona = None
    hemisferio = None
    id_hmf = 0


    list_name_hid = ['Latitude (UTM)',
                     'Longitude (UTM)',
                     'Comprimento (m)',
                     'Rugosidade (manning)',
                     'Largura (m)',
                     'Ângulo esquerdo (°)',
                     'Ângulo direito (°)']
    list_valores_hid = [None, None, 0.0, 0.001, 5, 45, 45]

    for i in range(n_tributarios + 1):

        # DADOS PARA AUTOMATIZAÇÃO NA ENTRADA
        dis_i = 50.0
        qt_st = 0
        disab = False
        if data != False:
            dis_i = data['Dados gerais']['l_d'][i]
            disab = True
            qt_st = data['Dados gerais']['l_q_s'][i]

        comprimento = []
        longitude = []
        latitude = []
        altitude = []
        preecheu = False
        expander = lista_tabs[i].expander(
            "**:orange[DADOS DE ENTRADA INICIAIS]**")

        if i > 0:
            coll1, coll2 = expander.columns(2)
            labels_mod = copy.deepcopy(labels)
            labels_mod.remove(labels[i])

            if data != False:
                af_valor = data['Dados gerais']['p_af'][i - 1]
                dt_desague = data['Dados gerais']['l_des'][i - 1] - 1
                dt_desague = 0 if dt_desague < 0 else dt_desague
            else:
                af_valor = [0.0, None, None, 0.0]
                dt_desague = 0
            desague = coll1.selectbox(str(i) + '. Deságua no:', labels_mod,
index=dt_desague)
```

```
            coll2.markdown("Ponto de deságue em relação o " + desague + ":")

            df_afl = pd.DataFrame({
                str(i) + '. Descrição': ['ID (opcional)', 'Latitude (UTM)',
'Longitude (UTM)', 'Comprimento (m)'],
                'Valores': af_valor})
               df_afl_f = coll2.data_editor(df_afl, disabled=[str(i) + '.
Descrição'])
            ponto_af.append(list(df_afl_f['Valores']))
            list_desague.append(labels.index(desague))

            expander.divider()


        col_11, col_12 = expander.columns(2)
        _, col_122 = col_12.columns(2)
        discret = col_122.number_input(
            str(i) + '. Discretização (m)', value=dis_i,
            min_value=0.1, step=1e-2, format="%.2f")

         tipo_entrada = col_11.radio(str(i) + ". Tipo de entrada dos dados
espaciais:",
                                     ["Manual", "Intervalo", "GeoJSON"],
                                     horizontal=True)
        df_esp = pd.DataFrame(columns=[str(i) + '. Latitude (UTM)', str(i)
+ '. Longitude (UTM)',
                                       str(i) + '. Altitude (m)', str(i) +
'. Comprimento (m)'])

        if tipo_entrada == "Intervalo":
            if data != False:
                valor_comp = data['Dados gerais']['l_c'][i][-1]
            else:
                valor_comp = 150.0
            col0_1, col0_2 = expander.columns(2)

            comp = col0_1.number_input(
                'Comprimento do ' + str(labels[i]) + ' (m)'
                + ' *a ser modelado*',
               min_value= 150.0, value=valor_comp, step=1e-2, format="%.2f")

            col1_1, col1_2 = col0_2.columns(2)

            altit = col1_1.number_input(
                str(i) + '. Altitude inicial (m)', value=50.0,
                min_value=1.0, step=1e-2, format="%.2f")

            incl = col1_2.number_input(
                str(i) + '. Inclinação (m/m)',
                min_value=0.0001, step=1e-4, format="%.4f")

        if tipo_entrada == "Manual":

            if data != False:
                    df_esp[str(i) + '. Latitude (UTM)'] = data['Dados
gerais']['l_la'][i]
                    df_esp[str(i) + '. Longitude (UTM)'] = data['Dados
gerais']['l_lo'][i]
                     df_esp[str(i) + '. Altitude (m)'] = data['Dados
gerais']['l_a'][i]
                    df_esp[str(i) + '. Comprimento (m)'] = data['Dados
gerais']['l_c'][i]

             expander.warning('''Adicionar ou a **Longitude e Latitude** ou
o **Comprimento**.''',
                    icon="❗")
            df_espacial = expander.data_editor(df_esp,
```

```
                                                num_rows="dynamic",
                                                column_config={
                                                    str(i) + '. Latitude
(UTM)':st.column_config.NumberColumn(format="%.2f"),
                                                   str(i) + '. Longitude
(UTM)':st.column_config.NumberColumn(format="%.2f"),
                                                    str(i) + '. Altitude
(m)':st.column_config.NumberColumn(format="%.2f"),
                                                  str(i) + '. Comprimento
(m)':st.column_config.NumberColumn(format="%.2f")})

            if len(df_espacial[str(i) + '. Latitude (UTM)']) > 1:
                latitude = list(df_espacial.iloc[:,0])
                longitude = list(df_espacial.iloc[:,1])
                altitude = list(df_espacial.iloc[:,2])
                comprimento = list(df_espacial.iloc[:,3])
                if df_espacial[str(i) + '. Latitude (UTM)'][0] != None:
                    preecheu = True

        if tipo_entrada == "GeoJSON":
            df_espacial = expander.file_uploader(str(i) +
                                                 ". ⚠️ Submeter o arquivo
.GeoJSON, em WGS 84 UTM," +
                                                " contendo somente a coluna
'Altitude' em metros.",
                                                type=["geojson"])

            if df_espacial is not None:
                from geopandas import read_file
                gdf = read_file(df_espacial)
                preecheu = True
                for x in range(len(gdf)):
                    longitude.append(gdf["geometry"][x].x)
                    latitude.append(gdf["geometry"][x].y)

                altitude = list(gdf.iloc[:,1])
                comprimento = []
                for _ in range(len(altitude)):
                    comprimento.append(None)
        if preecheu:
            col1, col2 = expander.columns(2)
            coln1, coln2 = col1.columns(2)
            if data != False:
                id_hmf = data['Dados gerais']['i_hmf']
                zona_dt = data['Dados gerais']['zn']

            else:
                zona_dt = 22
            zona = coln1.number_input(
               str(i) + '. WGS 84 - UTM - Zona:', min_value=0, value=zona_dt)
              hemisferio = coln2.radio(str(i) + '. Hemisfério:', ['Sul',
'Norte'],
                                       horizontal=True, index=id_hmf)
            if hemisferio == 'Sul':
                hemisferio = 'south'
                id_hmf = 0
            else:
                hemisferio = 'north'
                id_hmf = 1
            on = col2.toggle(str(i) + '. Visualizar dados espaciais')

            if on:

                from pyproj import Proj

                    df_plot = pd.DataFrame({'UTMlon': longitude, 'UTMlat':
latitude})
```

```
                myProj = Proj('+proj=utm +zone=' + str(zona)
                              + ' +' + str(hemisferio) + ' +ellps=WGS84',
                              preserve_units=False)
                                        df_plot['lon'],    df_plot['lat']    =
myProj(df_plot['UTMlon'].values,
                                                          df_plot['UTMlat'].values,
                                                               inverse=True)
                expander.map(df_plot, size = 1, color='#007FFF')

        else:
            if tipo_entrada == "Intervalo":
                comprimento = list(np.arange(0, comp + discret, discret))

                for k in range(len(comprimento)):
                    altitude.append(altit - (discret * incl))
                    latitude.append(None)
                    longitude.append(None)

        expander.divider()

        expander.markdown(":orange[Variáveis iniciais do " + str(labels[i])
+ ":]")

        ##########################################
        if lista_modelagem['s_t']:
            list_name = ['Data',str(i) + '. Q (m³/s)']
            list_valores = [None, [0.0]]
        else:
            list_name = [str(i) + '. Q (m³/s)']
            list_valores = [[0.0]]

        if lista_modelagem['m_od'] or lista_modelagem['m_dbo']:
            list_name.extend(['OD (mg/L)',
                              'DBO (mg/L)'])
            list_valores.extend([[0.0], [0.0]])
        if lista_modelagem['m_n']:
            list_name.extend(['N-org (mg/L)',
                              'N-amon (mg/L)',
                              'N-nitri (mg/L)',
                              'N-nitra (mg/L)'])
            list_valores.extend([[0.0], [0.0], [0.0], [0.0]])
        if lista_modelagem['m_p']:
            list_name.extend(['P-org (mg/L)',
                              'P-inorg (mg/L)'])
            list_valores.extend([[0.0], [0.0]])
        if lista_modelagem['m_c']:
            list_name.append('E-coli (NMP/100ml)')
            list_valores.extend([[0.0]])
        colmdisab = None
        if data != False:
            colmdisab = 'Data'
            list_valores = data['Dados gerais']['l_v_i'][i]
            if lista_modelagem['s_t']:
                list_valores[0] = pd.to_datetime(list_valores[0])

        if lista_modelagem['s_t'] and data == False:
            num_rows = "dynamic"
        else:
            num_rows = "fixed"
        df_conc = pd.DataFrame(columns=list_name)

        if i == 0:
            valores = copy.deepcopy(list_valores)
            for y in range(len(list_name)):
                df_conc[list_name[y]] = list_valores[y]
            df_conc_f = expander.data_editor(df_conc, num_rows=num_rows,
                                             column_config={
```

```
                                                'Data':st.column_config.Dat
eColumn(
                                                        format="MM.DD.YYYY",
step=1),
                                                },
                                            disabled=[colmdisab])


            dias = list(df_conc_f[list_name[0]])

        else:
            valores2 = copy.deepcopy(valores)
            valores2.pop(0)
            if lista_modelagem['s_t'] and data == False:
                l_valor = []
                for id in range(len(valores2)):
                    l_valor.append(valores2[id][0])
                for id2 in range(len(dias)):
                    soma = [dias[id2]] + l_valor
                    df_conc.loc[id2] = soma

            else:
                for y in range(len(list_name)):
                    df_conc[list_name[y]] = list_valores[y]

            df_conc_f = expander.data_editor(df_conc,
                                            column_config={
                                                'Data':st.column_config.Dat
eColumn(
                                                        format="MM.DD.YYYY",
step=1),
                                                },
                                            disabled=['Data'])

        list_valores_f = []
        for yf in range(len(list_name)):
            lista = list(df_conc_f[list_name[yf]])
                        if   yf   ==   0    and   lista_modelagem['s_t']   and
df_conc_f[list_name[yf]][0] != None and data != False:
                list_valores_f.append(list(df_conc_f[list_name[yf]].dt.strf
time('%Y %m %d')))

            else:
                list_valores_f.append(lista)

        expander.divider()
        expander.markdown(":orange[Seções transversais do " + str(labels[i])
+ ":]")
        col_3_1, _ = expander.columns(2)
        n_pontos_st = col_3_1.number_input(
            str(i) + '. Quantidade de pontos que alteram os valores'
            + ' de uma ou mais variáveis hidráulicas:',
            min_value=0, value= qt_st, disabled=disab)
        df_hid = pd.DataFrame({str(i) + '. Descrição': list_name_hid})
        for k in range(n_pontos_st + 1):
            if data != False:
              df_hid['Ponto ' + str(k)] = data['Dados gerais']['l_sc'][i][k]

            else:
                df_hid['Ponto ' + str(k)] = list_valores_hid
           df_hid_f  =  expander.data_editor(df_hid,  disabled=[str(i)  +  '.
Descrição'])
        hidr = []
        for k in range(n_pontos_st + 1):
            hidr.append(list(df_hid_f['Ponto ' + str(k)]))
```

```
        list_comprimento.append(comprimento)
        list_longitude.append(longitude)
        list_latitude.append(latitude)
        list_altitude.append(altitude)
        list_discretizacao.append(discret)
        list_valor_i.append(list_valores_f)
        list_secaotrav.append(hidr)
        list_qnt_secaotrav.append(n_pontos_st)

     lista_parametros = {'l_v_i': list_valor_i, 'l_c': list_comprimento,
'l_lo': list_longitude,
                         'l_la': list_latitude, 'l_a': list_altitude, 'l_sc':
list_secaotrav,
                                          'l_d': list_discretizacao, 'l_q_s':
list_qnt_secaotrav, 'p_af': ponto_af,
                         'l_des': list_desague, 'zn': zona, 'i_hmf': id_hmf}

    return lista_parametros, list_name, valores, zona, hemisferio, dias

########################## COEFICIENTES ############################
def coeficientes(data, lista_modelagem, n_tributarios, labels, lista_tabs,
dias):
    # COEFICIENTES DDO MODEDELO

    lista_n_pontos = []

    for i in range(n_tributarios + 1):

        expander = lista_tabs[i].expander(
            "**:green[COEFICIENTES DO MODELO]**")
        col_11, col_21 = expander.columns(2)

        qt_coe = 0
        disab = False
        if data != False:
            qt_coe = data['Coeficientes']['l_n_p'][i]
            disab = True

        if i == 0:
            if lista_modelagem['m_od']:
                lista_modelagem['é_calc'] = expander.toggle('Coeficiente k2
será tabelado.', value=lista_modelagem['é_calc'], disabled=disab)

            list_name = ['Temperatura (°C)']
            if lista_modelagem['é_calc']:
                list_name.append('k2 (1/d)')
            if lista_modelagem['m_od']:
                list_name.extend(['k1 (1/d)',
                                  'kd (1/d)',
                                  'ks (1/d)',
                                  'lrd (gDBO5/m.d)',
                                  'sd (1/d)'])
            if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] ==
True:
                list_name.append('O2namon (mgO2/mgNamon oxid)')
            if lista_modelagem['m_n']:
                list_name.extend(['koa (1/d)',
                                  'kso (1/d)',
                                  'kan (1/d)',
                                  'Snamon (g/m2.d)',
                                  'knn (1/d)',
                                  'knitr (1/d)'])
            if lista_modelagem['m_p']:
                list_name.extend(['koi (1/d)',
                                      'kspo (1/d)',
                                      'spinorg (1/d)'])
            if lista_modelagem['m_c']:
```

```
                list_name.append('kb (1/d)')

            lista_coeficiente = [list_name]


        list_name_c = ['Latitude (UTM)',
                       'Longitude (UTM)',
                       'Comprimento (m)']
        list_valores_c = [None, None, 0.0]
         if lista_modelagem['é_calc'] == False and (lista_modelagem['m_od']
or lista_modelagem['m_dbo']):
            list_name_c.append('k2 máximo (1/d)')
            list_valores_c.append(1000.0)



          col_21.warning('''Adicionar ou a **Longitude e Latitude** ou o
**Comprimento**.''',
                                 icon="❗")

        expander.markdown(":green[Variáveis do " + str(labels[i]) +":]")
        coef = []

        n_pontos = col_11.number_input(
            str(i) + '. Quantidade de ponto que alteram os valores'
            + ' de um ou mais coeficientes tabelados:',
            min_value=0, value=qt_coe, disabled=disab)
        lista_n_pontos.append(n_pontos)
        df_coef = pd.DataFrame({str(i) + '. Descrição': list_name_c})
        labels_c = []
        for k in range(n_pontos + 1):
            labels_c.append(str(i) + '. Ponto ' + str(k))
            if data != False:
                                        df_coef['Ponto    '    +    str(k)]    =
data['Coeficientes']['l_coe'][i+1][k][0]

            else:
                df_coef['Ponto ' + str(k)] = list_valores_c
          df_coef_f  =  expander.data_editor(df_coef,  disabled=[str(i)  +  '.
Descrição'])

        tabs_c = expander.tabs(labels_c)
        for tc in range(len(labels_c)):
            if lista_modelagem['s_t']:
               list_name_c2 = ['Data', str(i) + '.' + str(tc) +'. Temperatura
(°C)']
            else:
               list_name_c2 = [str(i) + '.' + str(tc) +'. Temperatura (°C)']
            list_valores_c2 = [[22.0]]
            if lista_modelagem['é_calc']:
                list_name_c2.append('k2 (1/d)')
                list_valores_c2.append([0.0])
            if lista_modelagem['m_od'] or lista_modelagem['m_dbo']:
                list_name_c2.extend(['k1 (1/d)',
                                     'kd (1/d)',
                                     'ks (1/d)',
                                     'lrd (gDBO5/m.d)',
                                     'sd (1/d)'])
                list_valores_c2.extend([[0.0], [0.0], [0.0], [0.0], [0.0]])
           if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] ==
True:
                list_name_c2.append('O2namon (mgO2/mgNamon oxid)')
                list_valores_c2.append([0.0])
            if lista_modelagem['m_n']:
                list_name_c2.extend(['koa (1/d)',
                                     'kso (1/d)',
                                     'kan (1/d)',
```

```
                                'Snamon (g/m2.d)',
                                'knn (1/d)',
                                'knitr (1/d)'])
                     list_valores_c2.extend([[0.0], [0.0], [0.0], [0.0], [0.0],
[0.0]])
            if lista_modelagem['m_p']:
                list_name_c2.extend(['koi (1/d)',
                                     'kspo (1/d)',
                                     'spinorg (1/d)'])
                list_valores_c2.extend([[0.0], [0.0], [0.0]])
            if lista_modelagem['m_c']:
                list_name_c2.append('kb (1/d)')
                list_valores_c2.append([0.0])

            df_coef2 = pd.DataFrame(columns=list_name_c2)

            if data != False:
                list_valores_c2 = data['Coeficientes']['l_coe'][i+1][tc][1]

            if lista_modelagem['s_t'] and data == False:
                l_valor_c = []
                for id in range(len(list_valores_c2)):
                    l_valor_c.append(list_valores_c2[id][0])
                for id2 in range(len(dias)):
                    soma = [dias[id2]] + l_valor_c
                    df_coef2.loc[id2] = soma

            else:
                for yc in range(len(list_name_c2)):
                    if lista_modelagem['s_t'] == True:
                        if yc == 0:
                            df_coef2[list_name_c2[yc]] = dias
                        else:
                            df_coef2[list_name_c2[yc]] = list_valores_c2[yc
- 1]
                    else:
                        df_coef2[list_name_c2[yc]] = list_valores_c2[yc]

            df_coef2_f = tabs_c[tc].data_editor(df_coef2,
                                                column_config={
                                                    'Data':st.column_config.Dat
eColumn(
                                                            format="MM.DD.YYYY",
step=1),
                                                },
                                                disabled=['Data'])
            list_valores_c2 = []
            for yf in range(len(list_name_c2)):
                if yf == 0 and lista_modelagem['s_t'] == True:
                    pass
                else:
                    list_valores_c2.append(list(df_coef2_f[list_name_c2[yf]
]))
            list_c1 = []

            for c1 in list(df_coef_f['Ponto ' + str(tc)]):
                if c1 != None:
                    list_c1.append(float(c1))
                else:
                    list_c1.append(c1)

            coef.append([list_c1, list_valores_c2])

        lista_coeficiente.append(coef)
    list_coef_f = {'l_coe': lista_coeficiente, 'l_n_p': lista_n_pontos}
    return list_coef_f
```

```
################### RETIRADAS E CONTRIBUIÇÕES ####################
def fun_contrib_retirad(data, n_tributarios, labels, lista_tabs, list_name,
list_valores, lista_modelagem, dias):

    list_name_cr = ['ID (opcional)',
                    'Latitude (UTM)',
                    'Longitude (UTM)',
                    'Comprimento (m)']
    list_valores_cr = [None, None, None, 0.0]
    list_name.pop(0)
    list_valores.pop(0)
    if lista_modelagem['s_t']:
        list_name.pop(0)
        list_valores.pop(0)

    list_retiradas = []
    list_ep = []
    list_ed = []
    list_qnt = []

    for j in range(n_tributarios + 1):

             expander  =  lista_tabs[j].expander("**:blue[CONTRIBUIÇÕES  E
RETIRADAS]**")

        qt_ret = 1
        on_ret = False
        qt_ep = 1
        on_ep = False
        qt_ed = 1
        on_ed = False
        disab = False
        if data != False:
            qt_ret = data['Contr e Retir']['l_q'][j][0]
            qt_ep = data['Contr e Retir']['l_q'][j][1]
            qt_ed = data['Contr e Retir']['l_q'][j][2]
            on_ret = data['Contr e Retir']['l_q'][j][3]
            on_ep = data['Contr e Retir']['l_q'][j][4]
            on_ed = data['Contr e Retir']['l_q'][j][5]
            disab = True

         retiradas = expander.checkbox(str(j) + '. Possui algum ponto de
**captação de água**.',
                                       value=on_ret, disabled=disab)
        ret = []
        n_pontos_r = 0

        if retiradas:
            col4_1, col4_2 = expander.columns(2)
            col4_1.markdown(":blue[Retiradas do " + str(labels[j]) + ":]")
             col4_2.warning('''Adicionar ou a **Longitude e Latitude** ou o
**Comprimento**.''',
                           icon="❗")
            n_pontos_r = col4_1.number_input(
                str(j) + '. Quantidade de ponto de captação:',
                min_value=1, value=qt_ret, disabled=disab)
            dfret = pd.DataFrame({str(j) + '. R: Variável': list_name_cr})
            labels_r = []
            for k in range(n_pontos_r):
                labels_r.append(str(j) + '. Ponto ' + str(k + 1))


                if data != False:
                    dfret['Ponto  '  +  str(k  +  1)]  =  data['Contr  e
Retir']['l_r'][j][k][0]

                else:
```

```
                    dfret['Ponto ' + str(k + 1)] = list_valores_cr
            df_ret_f = expander.data_editor(dfret, disabled=[str(j) + '. R:
Variável'])



            tabs_r = expander.tabs(labels_r)
            for tr in range(len(labels_r)):
                if lista_modelagem['s_t']:
                      list_name_r = ['Data', str(j) + '.' + str(tr) +'. Q
(m³/s)']
                else:
                    list_name_r = [str(j) + '.' + str(tr) +'. Q (m³/s)']

                list_valores_r = [[0.0]]
                df_ret2 = pd.DataFrame(columns=list_name_r)
                if data != False:
                    list_valores_r = data['Contr e Retir']['l_r'][j][tr][1]

                if lista_modelagem['s_t'] and data == False:
                    l_valor_r = []
                    for id in range(len(list_valores_r)):
                        l_valor_r.append(list_valores_r[id][0])
                    for id2 in range(len(dias)):
                        soma = [dias[id2]] + l_valor_r
                        df_ret2.loc[id2] = soma

                else:
                    for yc in range(len(list_name_r)):
                        if lista_modelagem['s_t'] == True:
                            if yc == 0:
                                df_ret2[list_name_r[yc]] = dias
                            else:
                               df_ret2[list_name_r[yc]] = list_valores_r[yc
- 1]
                        else:
                            df_ret2[list_name_r[yc]] = list_valores_r[yc]

                df_ret2_f = tabs_r[tr].data_editor(df_ret2,
                                                   column_config={
                                                       'Data':st.column_con
fig.DateColumn(
                                                           format="MM.DD.YYYY",
step=1),
                                                               },
                                                   disabled=['Data'])
                list_valores_r = []
                for rf in range(len(list_name_r)):
                    if rf == 0 and lista_modelagem['s_t'] == True:
                        pass
                    else:
                        list_valores_r.append(list(df_ret2_f[list_name_r[rf
]]))

                     ret.append([list(df_ret_f['Ponto  '  +  str(tr  +  1)]),
list_valores_r])

        expander.divider()
        contr_pontual = expander.checkbox(str(j) + '. Possui algum ponto de
**descarga de poluição pontual**.',
                                          value=on_ep, disabled=disab)
        ep = []
        n_pontos_ep = 0
        if contr_pontual:
            col4_1, col4_2 = expander.columns(2)
                  col4_1.markdown(":blue[Contribuições  pontuais  do  "  +
str(labels[j]) + ":]")
```

```
            col4_2.warning('''Adicionar ou a **Longitude e Latitude** ou o
**Comprimento**.''',
                           icon="❗")
            n_pontos_ep = col4_1.number_input(
                str(j) + '. Quantidade de pontos de entradas pontuais:',
                min_value=1, value=qt_ep, disabled=disab)
            dfep = pd.DataFrame({str(j) + '. EP: Variável': list_name_cr})
            labels_ep = []
            for k in range(n_pontos_ep):
                labels_ep.append(str(j) + '. Ponto ' + str(k + 1))
                if data != False:
                           dfep['Ponto ' + str(k + 1)] = data['Contr e
Retir']['l_ep'][j][k][0]

                else:
                    dfep['Ponto ' + str(k + 1)] = list_valores_cr
             df_ep_f = expander.data_editor(dfep, disabled=[str(j) + '. EP:
Variável'])

            tabs_ep = expander.tabs(labels_ep)
            for tep in range(len(labels_ep)):
                if lista_modelagem['s_t']:
                     list_name_ep = ['Data', str(j) + '.' + str(tep) +'. Q
(m³/s)'] + list_name
                else:
                     list_name_ep = [str(j) + '.' + str(tep) +'. Q (m³/s)']
+ list_name

                list_valores_ep = [[0.0]] + list_valores
                df_ep2 = pd.DataFrame(columns=list_name_ep)

                if data != False:
                  list_valores_ep = data['Contr e Retir']['l_ep'][j][tep][1]

                if lista_modelagem['s_t'] and data == False:
                    l_valor = []
                    for id in range(len(list_valores_ep)):
                        l_valor.append(list_valores_ep[id][0])
                    for id2 in range(len(dias)):
                        soma = [dias[id2]] + l_valor
                        df_ep2.loc[id2] = soma

                else:
                    for yc in range(len(list_name_ep)):
                        if lista_modelagem['s_t'] == True:
                            if yc == 0:
                                df_ep2[list_name_ep[yc]] = dias
                            else:
                               df_ep2[list_name_ep[yc]] = list_valores_ep[yc
- 1]
                        else:
                            df_ep2[list_name_ep[yc]] = list_valores_ep[yc]

                df_ep2_f = tabs_ep[tep].data_editor(df_ep2,
                                                    column_config={
                                                        'Data':st.column_con
fig.DateColumn(
                                                        format="MM.DD.YYYY",
step=1),
                                                            },
                                                    disabled=['Data'])
                list_valores_ep = []
                for epf in range(len(list_name_ep)):
                    if epf == 0 and lista_modelagem['s_t'] == True:
                        pass
                    else:
```

```
                        list_valores_ep.append(list(df_ep2_f[list_name_ep[e
pf]]))

                    ep.append([list(df_ep_f['Ponto ' + str(tep + 1)]),
list_valores_ep])

        expander.divider()
        contr_difusa = expander.checkbox(str(j) + '. Possui algum ponto de
**descarga de poluição difusa**.',
                                         value=on_ed, disabled=disab)
        ed = []
        n_pontos_ed = 0
        if contr_difusa:
            col4_1, col4_2 = expander.columns(2)
            col4_1.markdown(":blue[Contribuições difusa do " + str(labels[j])
+ ":]")
            col4_2.warning('''Adicionar ou a **Longitude e Latitude** ou o
**Comprimento**.''',
                           icon="❗")
            n_pontos_ed = col4_1.number_input(
                str(j) + '. Quantidade de pontos de entradas difusas:',
                min_value=1, value=qt_ed, disabled=disab)
            list_name_ed = ['ID (opcional)',
                            'Latitude inicial (UTM)',
                            'Latitude final (UTM)',
                            'Longitude inicial (UTM)',
                            'Longitude final (UTM)',
                            'Comprimento inicial (m)',
                            'Comprimento final (m)']
            list_valores_ed = [None, None, None, None, None, 0.0, 0.0]
            dfed = pd.DataFrame({str(j) + '. ED: Variável': list_name_ed})
            labels_ed = []
            for k in range(n_pontos_ed):
                labels_ed.append(str(j) + '. Ponto ' + str(k + 1))
                if data != False:
                    dfed['Ponto ' + str(k + 1)] = data['Contr e
Retir']['l_ed'][j][k][0]

                else:
                    dfed['Ponto ' + str(k + 1)] = list_valores_ed
            df_ed_f = expander.data_editor(dfed, disabled=[str(j) + '. ED:
Variável'])

            tabs_ed = expander.tabs(labels_ed)
            for ted in range(len(labels_ed)):
                if lista_modelagem['s_t']:
                    list_name_ed = ['Data', str(j) + '.' + str(ted) +'. Q
TOTAL (m³/s)'] + list_name
                else:
                    list_name_ed = [str(j) + '.' + str(ted) +'. Q TOTAL
(m³/s)'] + list_name

                list_valores_ed = [[0.0]] + list_valores
                df_ed2 = pd.DataFrame(columns=list_name_ed)

                if data != False:
                  list_valores_ed = data['Contr e Retir']['l_ed'][j][ted][1]

                if lista_modelagem['s_t'] and data == False:
                    l_valor_ed = []
                    for id in range(len(list_valores_ed)):
                        l_valor_ed.append(list_valores_ed[id][0])
                    for id2 in range(len(dias)):
                        soma = [dias[id2]] + l_valor_ed
                        df_ed2.loc[id2] = soma

                else:
```

```
                    for yc in range(len(list_name_ed)):
                        if lista_modelagem['s_t'] == True:
                            if yc == 0:
                                df_ed2[list_name_ed[yc]] = dias
                            else:
                              df_ed2[list_name_ed[yc]] = list_valores_ed[yc
- 1]
                        else:
                            df_ed2[list_name_ed[yc]] = list_valores_ed[yc]


                df_ed2_f = tabs_ed[ted].data_editor(df_ed2,
                                                    column_config={
                                                        'Data':st.column_con
fig.DateColumn(
                                                        format="MM.DD.YYYY",
step=1),
                                                            },
                                                    disabled=['Data'])
                list_valores_ed = []
                for edf in range(len(list_name_ed)):
                    if edf == 0 and lista_modelagem['s_t'] == True:
                        pass
                    else:
                        list_valores_ed.append(list(df_ed2_f[list_name_ed[e
df]]))

                  ed.append([list(df_ed_f['Ponto ' + str(ted + 1)]),
list_valores_ed])

        expander.divider()
        list_retiradas.append(ret)
        list_ep.append(ep)
        list_ed.append(ed)
        list_qnt.append([n_pontos_r, n_pontos_ep, n_pontos_ed,
                         retiradas, contr_pontual, contr_difusa])

     lista_contr_retir = {'l_r': list_retiradas, 'l_ep': list_ep, 'l_ed':
list_ed, 'l_q': list_qnt}

    return lista_contr_retir

######################## SALVAR ARQUIVO JSON ########################
def  salvararquivo(lista_modelagem,  lista_parametros,  lista_coeficiente,
lista_contr_retir,
                   list_name_salvo, list_valores):
    col_1, col_2 = st.columns(2)
    salvar = col_2.toggle('Salvar dados preenchidos.')
    if salvar:
         data = {'dados': [{'Paramêtros': lista_modelagem, 'Dados gerais':
lista_parametros,
                       'Coeficientes': lista_coeficiente, 'Contr e Retir':
lista_contr_retir,
                  'Nomes': list_name_salvo, 'Valores': list_valores}]}
        from json import dumps
        json_string = dumps(data)

        col_1.download_button(
            label="Clique para fazer o Download",
            file_name="DadosEntrada.json",
            mime="application/json",
            data=json_string,
        )
    return
```

XV. funcoes/layout_calibragem.py:

```
# biblioteca a serem importadas
import streamlit as st
import pandas as pd

def inicio_calib(paramentro):

    st.divider()
    dados_anterior = st.checkbox("Preenchimento automático.")
    data = False
    if dados_anterior:
                uploaded_json   =   st.file_uploader("Submeter   o   arquivo
'DadosEntrada.json'"
                                                     + " construído anteriormente:",
                                                     type=["json"])
        if uploaded_json is not None:
            from json import load
            data_dict = load(uploaded_json)
            data = data_dict['dados'][0]
            paramentro = data['Paramêtros']

        else:
                        st.markdown(""" ❗  Obs.:  Evite  alterar  o  arquivo
'DadosEntrada.json'
             utilizando o Excel. Se os dados não preencherem automaticamente,
              recomenda-se preencher de forma manual novamente.""")
    else:
        st.warning('''Se for a sua primeira vez no Qualitool 2.0,
                       você terá que preencher os dados de forma manual.''',
                       icon="❗")
    st.divider()

    col_1, col_2 = st.columns(2)
    bar = st.sidebar
    bar.warning('''Atenção! Em caso de atualização da tela, os dados
                já preenchidos serão perdidos e precisarão ser
                inseridos novamente.''',
                  icon="⚠️")

    col_1.markdown('Qual(s) parâmetro(s) modelar?')
    col_1_1, col_1_2 = col_1.columns(2)
    disab = False
    if data != False:
        disab = True
     modelar_od  =  col_1_1.checkbox('OD  e  DBO',  value=paramentro['m_od'],
disabled=disab)
    modelar_dbo = modelar_od
      modelar_n  =  col_1_1.checkbox('Nitrogênio',  value=paramentro['m_n'],
disabled=disab)
       modelar_p   =   col_1_2.checkbox('Fósforo',   value=paramentro['m_p'],
disabled=disab)
    modelar_colif = col_1_2.checkbox('Coliformes', value=paramentro['m_c'],
disabled=disab)

    col_2.markdown('Configurações gerais:')
         serie_tempo    =    col_2.toggle('Ativar    avaliação    temporal.',
value=paramentro['s_t'], disabled=disab)

    n_tributarios = col_2.number_input(
        'Quantidade de tributários modeláveis:', min_value=0,
        value=paramentro['n_tb'], disabled=disab)


     lista_modelagem  =  {'m_od':  modelar_od,  'm_dbo':  modelar_dbo,  'm_n':
modelar_n,
```

```
                            'm_p': modelar_p, 'm_c': modelar_colif, 'n_tb':
n_tributarios,
                        's_t': serie_tempo}

    alterar_par_pso = st.checkbox(":grey[Configurações avançadas.]")

    if alterar_par_pso:
        st.markdown('Parâmetros do PSO:')
        col1, col2 = st.columns(2)
        col2_1, col2_2 = col2.columns(2)
        tam_enxame = col2_1.number_input("❗ Tamanho do enxame:", value=15,
min_value=1)
         n_ger = col2_2.number_input("❗ Número de iterações", value=15,
min_value=1)
        col2.warning('''Cuidado: Quanto maior for o tamanho do enxame ou o
número de iterações,
                            maior será o tempo necessário para gerar os
resultados.''',
                     icon="❗")
        w = col1.slider("Inércia:", 0.0, 2.0, 0.9)
        col1_1, col1_2 = col1.columns(2)
        c1 = col1_1.slider("Componente cognitiva (pessoal):", 0.0, 2.0, 1.8)
        c2 = col1_2.slider("Componente social (global):", 0.0, 2.0, 2.0)

    else:
        tam_enxame = 15
        n_ger = 15
        w = 0.9
        c1 = 1.8
        c2 = 2.0

    lista_par_pos = [tam_enxame, n_ger, w, c1, c2]

    lista_tabs = ["tab0"]
    labels = ["Rio principal"]
    for n_trib in range(n_tributarios):
        lista_tabs.append("tab" + str(n_trib))
        labels.append("Tributário " + str(n_trib + 1))
    lista_tabs = st.tabs(labels)

    return lista_modelagem, data, labels, lista_tabs, lista_par_pos

def coef_intervalo(lista_modelagem, n_tributarios, labels, lista_tabs):
    # COEFICIENTES DDO MODEDELO

    lista_n_pontos = []
    list_name_c2 = ['Temperatura (°C)']
    list_valores_c2_mx = [24.0]
    list_valores_c2_mn = [19.0]
    if lista_modelagem['m_od']:
        list_name_c2.extend(['k1 (1/d)', 'k2 (1/d)', 'kd (1/d)',
                             'ks (1/d)', 'lrd (gDBO5/m.d)',
                             'sd (1/d)'])

        list_valores_c2_mx.extend([0.45, 100.0, 1.0, 0.35, 1.0, 10.0])
        list_valores_c2_mn.extend([0.08, 0.05, 0.08, 0.05, 0.05, 0.05])

    if lista_modelagem['m_od'] == True and lista_modelagem['m_n'] == True:
        list_name_c2.append('O2namon (mgO2/mgNamon oxid)')
        list_valores_c2_mx.append(4.0)
        list_valores_c2_mn.append(3.0)
    if lista_modelagem['m_n']:
        list_name_c2.extend(['koa (1/d)',
                         'kso (1/d)',
                         'kan (1/d)',
                         'Snamon (g/m2.d)',
                         'knn (1/d)',
```

```
                        'knitr (1/d)'])
        list_valores_c2_mx.extend([0.25, 0.10, 0.25, 0.5, 1.0, 1.0])
        list_valores_c2_mn.extend([0.15, 0.001, 0.15, 0.001, 0.1, 0.1])
    if lista_modelagem['m_p']:
        list_name_c2.extend(['koi (1/d)',
                             'kspo (1/d)',
                             'spinorg (1/d)'])
        list_valores_c2_mx.extend([0.3, 0.5, 0.2])
        list_valores_c2_mn.extend([0.2, 0.02, 0.001])
    if lista_modelagem['m_c']:
        list_name_c2.append('kb (1/d)')
        list_valores_c2_mx.append(1.5)
        list_valores_c2_mn.append(0.05)

    lista_coeficiente = [list_name_c2]

    list_name_c = ['Latitude (UTM)', 'Longitude (UTM)', 'Comprimento (m)']
    list_valores_c = [None, None, 0.0]
    list_calb_trib = []

    for i in range(n_tributarios + 1):

        expander = lista_tabs[i].expander(
            "**:green[INTERVALOS DE BUSCA DOS COEFICIENTES]**")

        calb_trib = False

        if i != 0:
           calb_trib = expander.checkbox('Calibrar ' + str(labels[i]) + '.')
        else:
            calb_trib = True

        list_calb_trib.append(calb_trib)

        if calb_trib == True or i == 0:

            col_11, col_21 = expander.columns(2)
             col_21.warning('''Adicionar ou a **Longitude e Latitude** ou o
**Comprimento**.''',
                                icon="❗")

            expander.markdown(":green[Variáveis do " + str(labels[i]) +":]")

            n_pontos = col_11.number_input(
                str(i) + '. Quantidade de pontos que alteram os valores'
                + ' de um ou mais coeficientes:',
                min_value=0)
            lista_n_pontos.append(n_pontos)

            df_coef = pd.DataFrame({str(i) + '. Descrição': list_name_c})
            for k in range(n_pontos + 1):
                df_coef['Ponto ' + str(k)] = list_valores_c
            df_coef_f = expander.data_editor(df_coef, disabled=[str(i) + '.
Descrição'])

            list_c1 = []
            for n_p in range(n_pontos + 1):
                list_c1.append(list(df_coef_f['Ponto ' + str(n_p)]))

                  expander.markdown(":green[Intervalos de busca do " +
str(labels[i]) +":]")

        else:
            lista_n_pontos.append(n_pontos)
            list_c1 = list_valores_c

        coef_igual = True
```

```
        if calb_trib and i != 0:
            coef_igual = expander.checkbox('Intervalos do ' +
                                           str(labels[i])
                                              + ' serão iguais aos do Rio
Principal.',
                                           value=True)

        if coef_igual == False or i == 0:
             df_coef2 = pd.DataFrame(columns=[str(i) + '. Nome', 'Mínimo',
'Máximo'])

            df_coef2[str(i) + '. Nome'] = list_name_c2
            df_coef2['Mínimo'] = list_valores_c2_mn
            df_coef2['Máximo'] = list_valores_c2_mx

            df_coef2_f = expander.data_editor(df_coef2,
                                           disabled=[str(i) + '. Nome'])

            if i == 0:
                coef_min_0 = df_coef2_f['Mínimo']
                coef_max_0 = df_coef2_f['Máximo']

            coef_min = df_coef2_f['Mínimo']
            coef_max = df_coef2_f['Máximo']

        else:
            coef_min = coef_min_0
            coef_max = coef_max_0

        lista_coeficiente.append([list_c1, [coef_min, coef_max]])



     list_coef_f = {'l_coe': lista_coeficiente, 'l_n_p': lista_n_pontos,
'calb_trib': list_calb_trib}
    return list_coef_f

def dados_reais(n_tributarios, labels, lista_tabs, list_name, list_valores,
                lista_modelagem, dados_calib):


    lista_n_pontos = []

    list_name_dr_1 = ['ID (opcional)', 'Latitude (UTM)', 'Longitude (UTM)',
'Comprimento (m)']
    list_valores_c = [0, None, None, 0.0]
    list_dr = []

    for i in range(n_tributarios + 1):
        dr = []
        if i == 0 or dados_calib['calb_trib'][i - 0] == True:

            expander = lista_tabs[i].expander("**:red[DADOS OBSERVADOS]**")

            col_11, col_21 = expander.columns(2)
             col_21.warning('''Adicionar ou a **Longitude e Latitude** ou o
**Comprimento**.''',
                                icon="❗")

            expander.markdown(":green[Variáveis do " + labels[i] +":]")

            n_pontos = col_11.number_input(
                str(i) + '. Quantidade de pontos observados no ' + labels[i]
+ ':',
                min_value=dados_calib['l_n_p'][i] + 1)
```

```
            lista_n_pontos.append(n_pontos)

                    df_coef_dr  =  pd.DataFrame({str(i)  +  '.  Descrição':
list_name_dr_1})
            labels_dr = []
            for k in range(n_pontos):
                df_coef_dr['Ponto ' + str(k + 1)] = list_valores_c
                labels_dr.append(str(i) + '. Ponto ' + str(k + 1))
            df_coef_f = expander.data_editor(df_coef_dr, disabled=[str(i) +
'. Descrição'])


            tabs_dr = expander.tabs(labels_dr)

            for tdr in range(len(labels_dr)):

                if lista_modelagem['s_t']:
                     list_name_dr = ['Data', str(i) + '.' + str(tdr) +'. Q
(m³/s)'] + list_name
                else:
                     list_name_dr = [str(i) + '.' + str(tdr) +'. Q (m³/s)']
+ list_name

                list_valores_dr = [[0.0]] + list_valores
                df_dr2 = pd.DataFrame(columns=list_name_dr)

                for yc in range(len(list_name_dr)):
                    if lista_modelagem['s_t'] == True:
                        if yc == 0:
                            df_dr2[list_name_dr[yc]] = [None]
                        else:
                            df_dr2[list_name_dr[yc]] = list_valores_dr[yc -
1][0]
                    else:
                        df_dr2[list_name_dr[yc]] = list_valores_dr[yc]

                         df_dr2_f   =   tabs_dr[tdr].data_editor(df_dr2,
num_rows="dynamic",
                                                 column_config={
                                                     'Data':st.column_con
fig.DateColumn(
                                                     format="MM.DD.YYYY",
step=1),
                                                          str(i) + '.' +
str(tdr) +'. Q (m³/s)':st.column_config.NumberColumn(min_value=0),
                                                              'OD
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                             'DBO
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                               'N-org
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                              'N-amon
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                             'N-nitri
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                             'N-nitra
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                               'P-org
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                             'P-inorg
(mg/L)':st.column_config.NumberColumn(min_value=0),
                                                              'E-coli
(NMP/100ml)':st.column_config.NumberColumn(min_value=0),
                                                 })
                list_valores_dr = []
                list_datas_dr = []
                for drf in range(len(list_name_dr)):
```

```
                    if drf == 0 and lista_modelagem['s_t'] == True:
                        list_datas_dr.append(pd.to_datetime(df_dr2_f[list_n
ame_dr[drf]]))
                    else:
                        list_valores_dr.append(list(df_dr2_f[list_name_dr[d
rf]]))

                    dr.append([list(df_coef_f['Ponto ' + str(tdr + 1)]),
list_valores_dr, list_datas_dr])

            list_dr.append(dr)


    lista_dados_reais = {'l_dr': list_dr, 'n_pontos': lista_n_pontos}

    return lista_dados_reais
```