Universidade Estadual de Santa Cruz – UESC

Relatórios III de Implementações de Métodos da Disciplina Análise Numérica

Relatório de implementações realizadas por Gabriel Rosa Galdino

Disciplina Análise Numérica.

Curso Ciência da Computação

Semestre 2025.2

Professor Gesil Sampaio Amarante II

Ilhéus – BA
2025

ÍNDICE

Lista de Figuras

Linguagem Escolhida e justificativas

A linguagem Python foi a selecionada para este trabalho devido à sua simplicidade e ao seu poderoso ecossistema para computação científica, o que permitiu focar mais na lógica dos métodos do que em complexidades de programação.

A escolha é justificada principalmente pelo uso de bibliotecas essenciais como o NumPy, que oferece uma base sólida para operações matemáticas de alta performance. Para a visualização e análise dos dados, foi fundamental o uso do Matplotlib. Além disso, a biblioteca SciPy foi de grande ajuda, pois já contém implementações de diversos algoritmos numéricos, o que facilitou a verificação e a aplicação prática dos métodos abordados na disciplina.

Dessa forma, o Python se mostrou a escolha ideal para os objetivos deste relatório, aliando um código legível a um ambiente computacional robusto, o que tornou a implementação dos métodos uma tarefa prática e didática.

Método de Euler Simples

Estratégia de Implementação

A implementação do Método de Euler foi estruturada sobre a definição fundamental da derivada como o limite de uma razão incremental. Computacionalmente, transformei essa definição analítica em um processo iterativo finito. Para garantir que o algoritmo fosse capaz de processar qualquer função matemática fornecida pelo usuário, independentemente de sua complexidade algébrica, optei por utilizar a biblioteca SymPy para realizar o parsing simbólico da string de entrada. Isso permitiu converter a equação textual (como "-0.5*x + y") em uma expressão avaliável matematicamente dentro do ambiente Python, conferindo grande flexibilidade à ferramenta.

O núcleo da estratégia reside em um laço de repetição while, controlado pela variável independente x. A cada iteração, uma função auxiliar dedicada (_avaliar_funcao) é chamada para computar a inclinação da reta tangente f(x, y) no ponto atual. O próximo valor da variável dependente, y_{i+1}, é então projetado linearmente somando-se ao valor atual o produto do passo h pela inclinação calculada. Uma decisão de design importante foi a gestão da precisão numérica: para mitigar erros de arredondamento inerentes à aritmética de ponto flutuante que poderiam fazer o laço exceder ou não atingir o final do intervalo, implementei um arredondamento de segurança (round) na verificação da condição de parada, assegurando que a malha de pontos gerada seja exata.

Estrutura dos Arquivos de Entrada/Saída

A estrutura dos arquivos foi projetada para ser simples e direta, facilitando a automação dos testes.

- **Arquivo de Entrada (entrada_edo.txt):** O programa espera um arquivo onde cada linha representa um PVI distinto, contendo quatro parâmetros separados por ponto e vírgula:
    1. A função diferencial f(x, y) (ex: -0.5*x + 0.02*x*y).
    2. O valor inicial y_0.
    3. O intervalo de integração definido como inicio, fim.
    4. O tamanho do passo h.

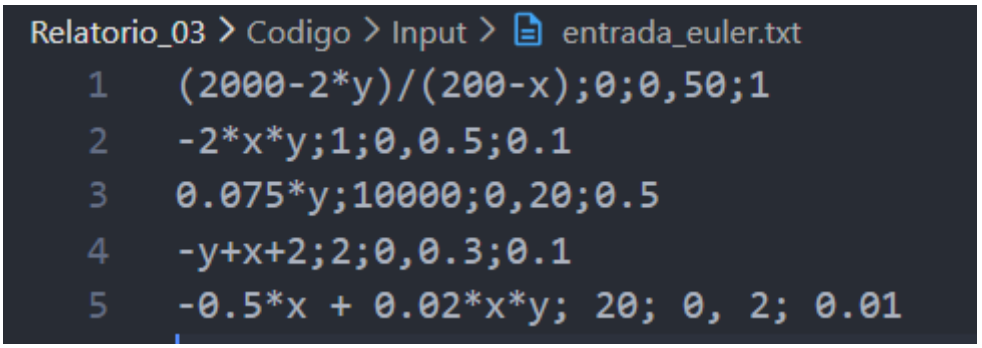

Figura 1 - Exemplo de Entrada Euler Simples

- **Arquivo de Saída (euler_saida.txt):** O relatório gerado no diretório output/ apresenta um cabeçalho identificando o método, seguido pelos resultados de cada problema. Para cada iteração, são listados os pares x = valor, y = valor com precisão de 5 casas decimais. Ao final, um sumário exibe o tempo total de execução.

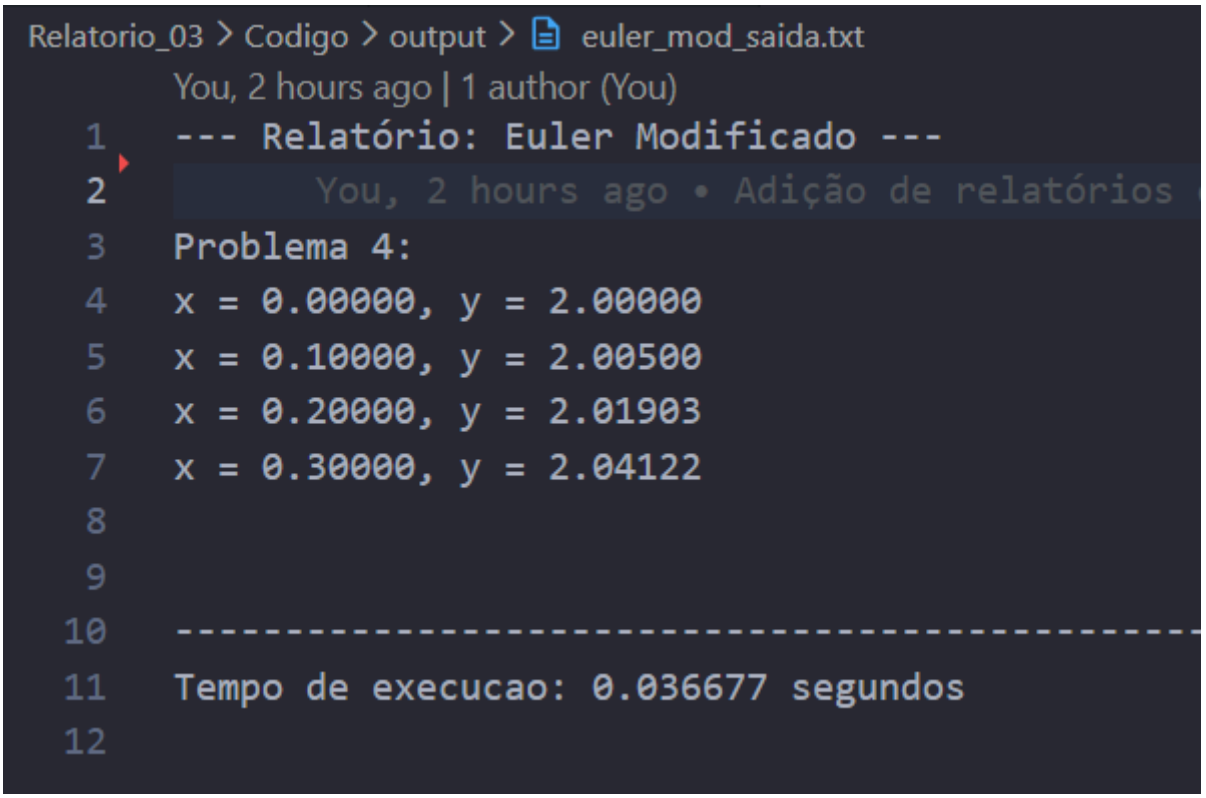

Figura 2 - Exemplo de Saída Euler Simples

Dificuldades Enfrentadas

A principal dificuldade encontrada foi garantir a precisão no controle do laço de repetição. Devido à forma como computadores representam números de ponto flutuante, a soma acumulativa do passo *h* pode gerar pequenos resíduos que fazem o laço executar uma vez a mais ou a menos que o esperado. Solucionei isso arredondando o valor atual de *x* (**round(x_current, 5)**) antes de compará-lo com o limite superior do intervalo. Outro ponto de atenção foi a correta conversão das strings de entrada para expressões simbólicas do SymPy, garantindo que variáveis como 'x' e 'y' fossem reconhecidas corretamente.

Método de Heun

Estratégia de Implementação

Para o Método de Heun, adotei uma abordagem de dois estágios, classicamente conhecida como técnica "Preditor-Corretor", visando superar a baixa precisão inerente ao método de Euler. A implementação foi desenhada para realizar duas avaliações distintas da função derivada por passo de integração, buscando uma melhor aproximação da inclinação média no intervalo $[x_i , x_{i+1}]$.

No primeiro estágio (Preditor), o código utiliza a lógica de Euler para estimar um valor provisório para y no final do passo, variável que denominei y_{euler}. No segundo estágio (Corretor), utilizo esse valor previsto para calcular a inclinação na extremidade direita do intervalo. A atualização final de y é determinada pela média aritmética entre a inclinação inicial e a inclinação prevista. Estruturei o código de forma sequencial rigorosa dentro do laço principal: primeiro calculo a inclinação atual, depois projeto o valor predito, calculo a inclinação futura e, somente então, consolido o novo valor de y. Essa organização lógica previne o uso prematuro de variáveis atualizadas, garantindo a integridade matemática do método e a redução do erro global para a ordem de O(h^2).

Estrutura dos Arquivos de Entrada/Saída

A estrutura de arquivos é idêntica à utilizada no Método de Euler, garantindo interoperabilidade entre os métodos de PVI.

- **Arquivo de Entrada:** Segue o padrão Função; y0; intervalo; h.

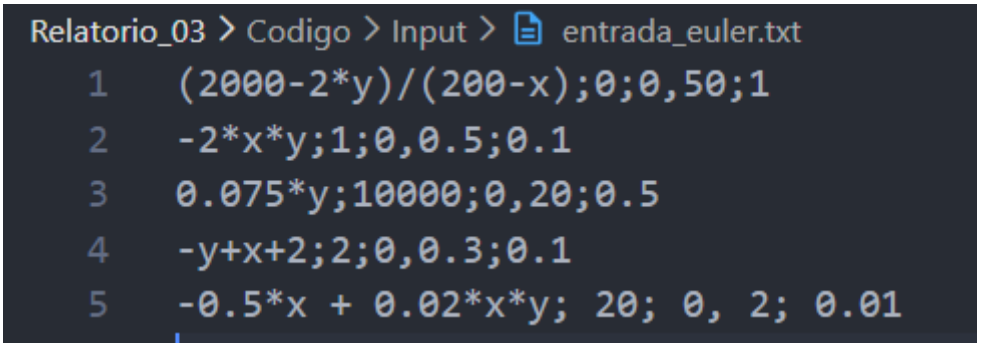

Figura 3 - Exemplo de Entrada Heun

- **Arquivo de Saída (heun_saida.txt):** O formato de saída mantém a padronização, exibindo a evolução de x e y passo a passo, permitindo uma comparação direta com os resultados do método de Euler.

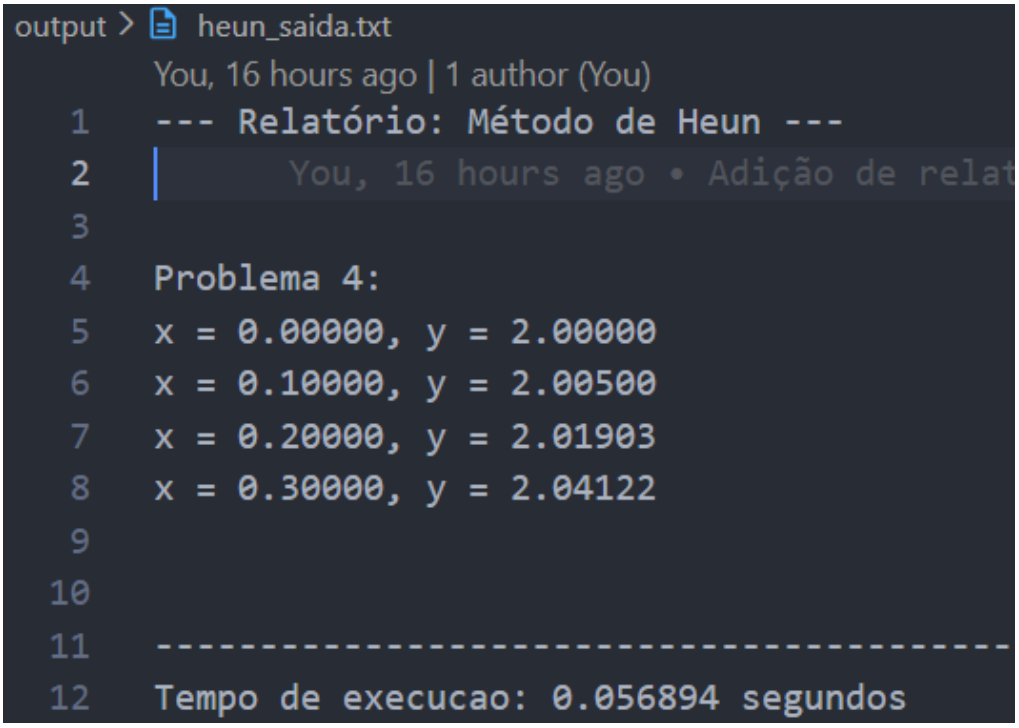

Figura 4 - Exemplo de Saída Heun

Dificuldades Enfrentadas

O desafio neste método foi assegurar que a ordem das operações de previsão e correção estivesse correta. Um erro comum é atualizar a variável x antes de calcular a inclinação no ponto final, o que levaria a resultados incorretos. Tive que estruturar o laço while cuidadosamente para calcular $dydx_{next}$ usando o x futuro (x + h) e o y previsto (y_{euler}), antes de consolidar o novo valor de y e avançar o passo.

Método de Euler Modificado

Estratégia de Implementação

Para o Método de Euler Modificado, também referido na literatura como Método do Ponto Médio, a estratégia de implementação foca na avaliação da derivada no centro do intervalo de integração, em vez de nas extremidades. A lógica foi construída para realizar um "meio passo" de Euler: calcula-se inicialmente a inclinação k_1 no ponto de partida para avançar x em h/2 e estimar um y intermediário.

A implementação computacional envolveu o cálculo explícito de duas constantes de inclinação (k) a cada iteração. A constante k_2, que representa a inclinação no ponto médio, é a única utilizada para projetar o valor final de y através de todo o intervalo h. Utilizei a capacidade de substituição do SymPy para avaliar a função f(x,y) nesses pontos intermediários, passando expressões compostas (como $y_{current}$ + h * k1) diretamente para o avaliador. Essa abordagem garante que a não-linearidade da função diferencial seja capturada com maior fidelidade do que no método de Euler simples, sem a complexidade computacional de múltiplos estágios dos métodos de ordem superior.

Estrutura dos Arquivos de Entrada/Saída

- **Arquivo de Entrada:** Utiliza o mesmo arquivo entrada_edo.txt compartilhado pelos outros métodos de PVI.

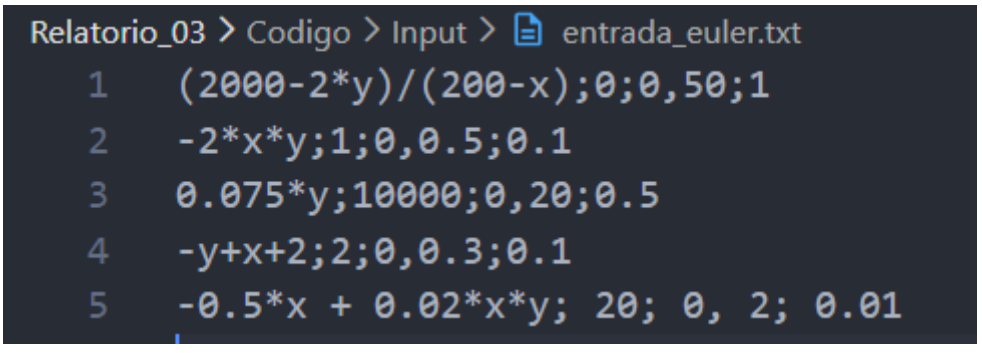

Figura 5 - Exemplo de Entrada Euler Modificado

- **Arquivo de Saída (euler_mod_saida.txt):** Gera um relatório contendo a tabela de valores calculados para cada problema listado na entrada, seguindo a formatação padrão do projeto.

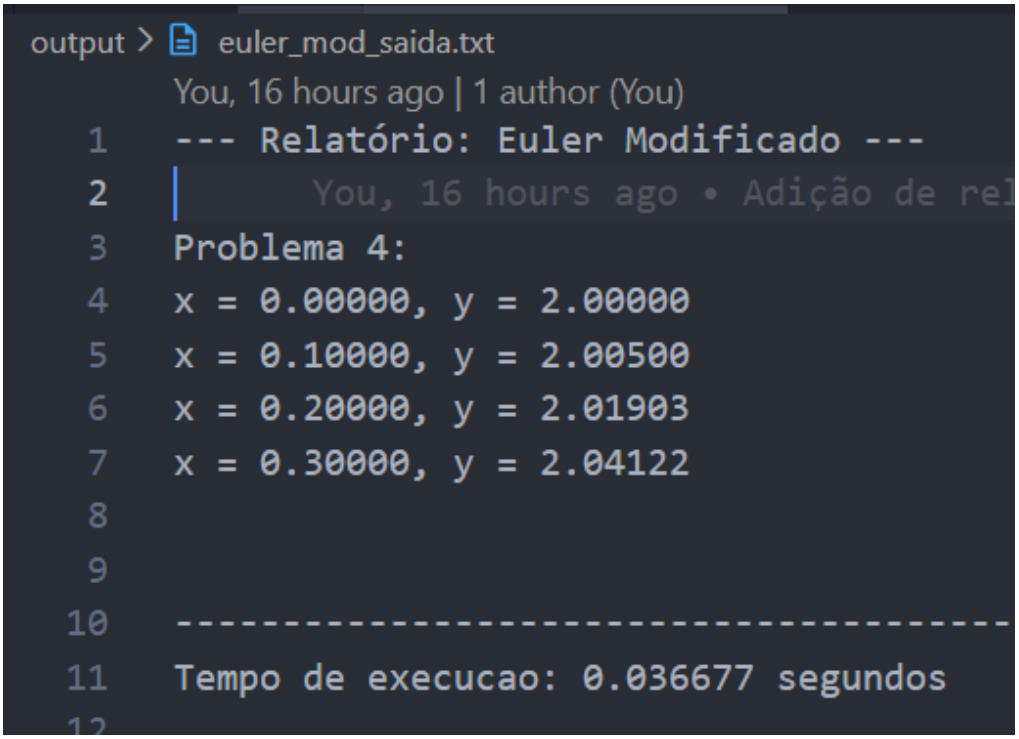

Figura 6 - Exemplo de Saída Euler Modificado

Dificuldades Enfrentadas

Uma dificuldade sutil encontrada foi a distinção conceitual entre este método e o método de Heun, já que ambos são melhorias de segunda ordem do Euler e frequentemente confundidos na literatura. Foi necessário revisar a teoria para garantir que a implementação refletisse estritamente o algoritmo do Ponto Médio (usando a inclinação em x + h/2) e não a média das inclinações nas pontas. Além disso, o tratamento de erros de leitura de arquivo exigiu atenção para garantir que o programa não falhasse caso o usuário inserisse um formato de intervalo inválido.

Método de Ralston

Estratégia de Implementação

Implementei o Método de Ralston seguindo a arquitetura dos métodos de Runge-Kutta de segunda ordem, mas com uma ponderação específica (a_1 = 1/3, a_2= 2/3) otimizada para minimizar o limite do erro de truncamento local. A estratégia difere dos demais métodos de 2ª ordem pela localização da segunda avaliação da derivada: em vez de avaliar no meio ou no fim, o algoritmo avalia a função a 3/4 do passo h.

No código, defini explicitamente os pesos 1/3 para a inclinação inicial (k_1) e 2/3 para a inclinação projetada (k_2). A atualização da variável dependente segue a fórmula ponderada y_{next} = y_{curr} + h($\frac{1}{3}k_1$ + $\frac{2}{3}k_2$). A implementação foi modularizada para manter a consistência com a interface dos outros métodos PVI, recebendo os mesmos parâmetros de entrada. O uso de listas dinâmicas (x_{vals}, y_{vals}) para armazenar o histórico de iterações permitiu desacoplar a lógica matemática da lógica de persistência de dados, facilitando a geração posterior dos relatórios.

Estrutura dos Arquivos de Entrada/Saída

- **Arquivo de Entrada:** Segue o formato padrão de PVI.

Relatorio_03 > Codigo > Input > entrada_euler.txt

```
1   (2000-2*y)/(200-x);0;0,50;1
2   -2*x*y;1;0,0.5;0.1
3   0.075*y;10000;0,20;0.5
4   -y+x+2;2;0,0.3;0.1
5   -0.5*x + 0.02*x*y; 20; 0, 2; 0.01
```

Figura 7 - Exemplo de Entrada Ralston

- **Arquivo de Saída (ralston_saida.txt):** Apresenta os resultados iterativos. A precisão de 5 casas decimais adotada na saída facilita a verificação da eficácia do método em relação aos outros de segunda ordem.

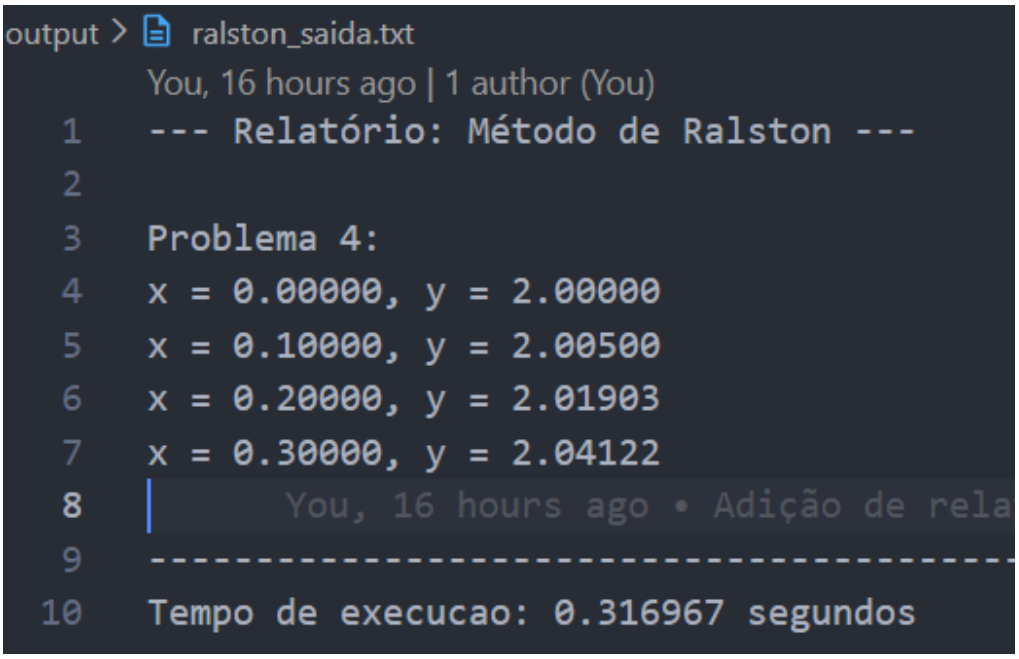

Figura 8 - Exemplo de Saída Ralston

Dificuldades Enfrentadas

A implementação foi direta, mas exigiu cuidado com os coeficientes fracionários (3/4, 1/3, 2/3). Em computação numérica, a ordem das operações pode afetar a precisão final. Certifiquei-me de que as multiplicações pelos pesos ocorressem na ordem correta para evitar perda de precisão antes da soma final. Também foi necessário validar se a função de entrada suportava a avaliação em pontos fracionários do intervalo sem gerar erros de domínio matemático.

Método de Runge-Kutta de 3ª Ordem

Estratégia de Implementação

A implementação do Runge-Kutta de 3ª Ordem (RK3) exigiu uma estrutura mais elaborada dentro do laço iterativo, devido à dependência em cadeia dos coeficientes angulares. A estratégia adotada calcula três inclinações distintas (k_1, k_2, k_3) em sequência, onde cada coeficiente utiliza o resultado do anterior para refinar a estimativa da posição y, aumentando a precisão da aproximação da curva.

O código foi desenvolvido para computar k_1 no início do intervalo, k_2 no ponto médio (utilizando a projeção de k_1) e k_3 no final do intervalo, mas com uma correção baseada em k_2. Um ponto crucial da implementação foi a correta tradução da fórmula para k_3, que envolve uma combinação linear das inclinações anteriores: f(x+h, y - h * k_1 + 2h * k_2). A média ponderada final (k_1 + 4k_2 + k_3)/6, análoga à Regra de Simpson, foi implementada aproveitando a precisão de ponto flutuante do Python, garantindo que o erro global da solução se mantivesse na ordem de O(h^3).

Estrutura dos Arquivos de Entrada/Saída

- **Arquivo de Entrada:** Compatível com o padrão PVI (entrada_edo.txt).

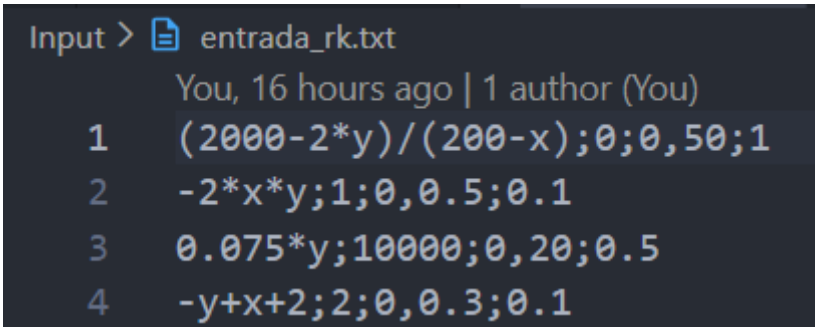

Figura 9 - Exemplo de Entrada Runge-Kutta de 3ª Ordem

- **Arquivo de Saída (rk3_saida.txt):** O arquivo gerado reflete a maior precisão deste método, listando a evolução da variável dependente ao longo do tempo ou espaço.

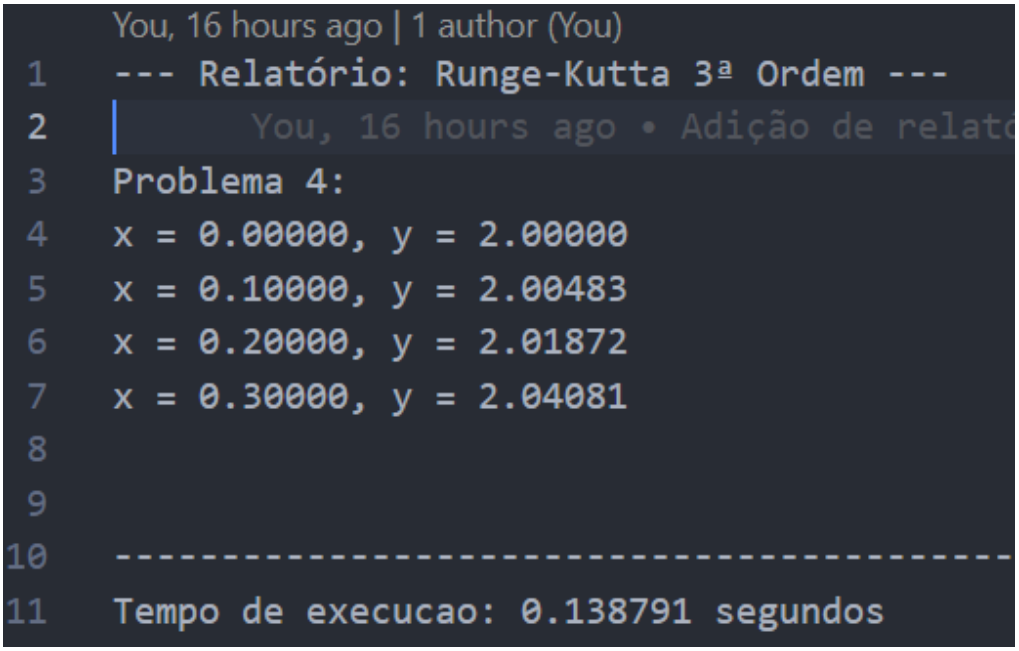

Figura 10 - Exemplo de Saída Runge-Kutta de 3ª Ordem

Dificuldades Enfrentadas

A complexidade aumentou neste método devido à dependência aninhada dos coeficientes. O cálculo de k_3, em particular, depende de uma expressão que envolve tanto k_1 quanto k_2 (y - h * k_1 + 2 * h * k_2). Um erro de sinal ou de parênteses nessa fórmula comprometeria todo o resultado. Foi necessário realizar testes passo a passo, comparando os valores intermediários com exercícios resolvidos manualmente para validar a implementação da fórmula.

Método de Runge-Kutta de 4ª Ordem (RK4)

Estratégia de Implementação

Para o método RK4, que representa o padrão de robustez em resolução numérica de EDOs, a estratégia de implementação baseou-se na média ponderada de quatro estimativas de inclinação. O algoritmo foi estruturado para realizar quatro avaliações da função _avaliar_funcao por passo, explorando o comportamento da derivada no início, duas vezes no meio e uma vez no fim do intervalo de integração.

A implementação priorizou a legibilidade e a precisão matemática. As variáveis k_1, k_2, k_3 e k_4 foram calculadas sequencialmente, com k_2 e k_3 corrigindo as estimativas no ponto médio do intervalo. A atualização final utiliza os pesos clássicos (1, 2, 2, 1), divididos por 6. Para garantir a performance, evitei chamadas redundantes de *parsing* da função, realizando a conversão simbólica (sympify) apenas uma vez antes do início do laço **while**. Esta estratégia torna o método extremamente eficaz e estável, sendo a escolha preferencial para a maioria das aplicações práticas no projeto.

Estrutura dos Arquivos de Entrada/Saída

- **Arquivo de Entrada:** Segue o padrão PVI unificado.

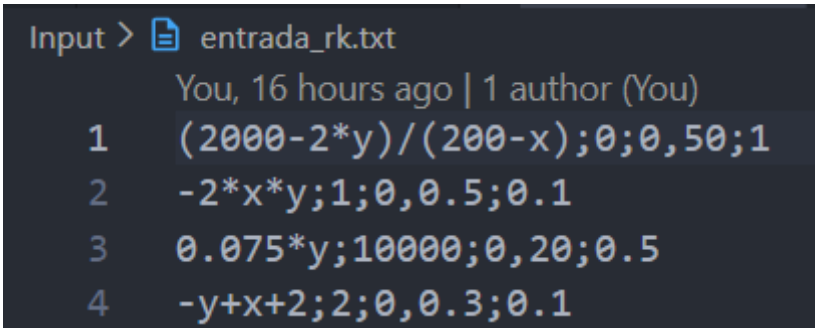

Figura 11 - Exemplo de Entrada Runge-Kutta de 4ª Ordem

- **Arquivo de Saída (`rk4_saida.txt`):** Exibe os resultados finais. Devido à alta precisão do RK4, este arquivo serve frequentemente como

base de comparação ("gabarito") para analisar o erro dos métodos de ordem inferior.

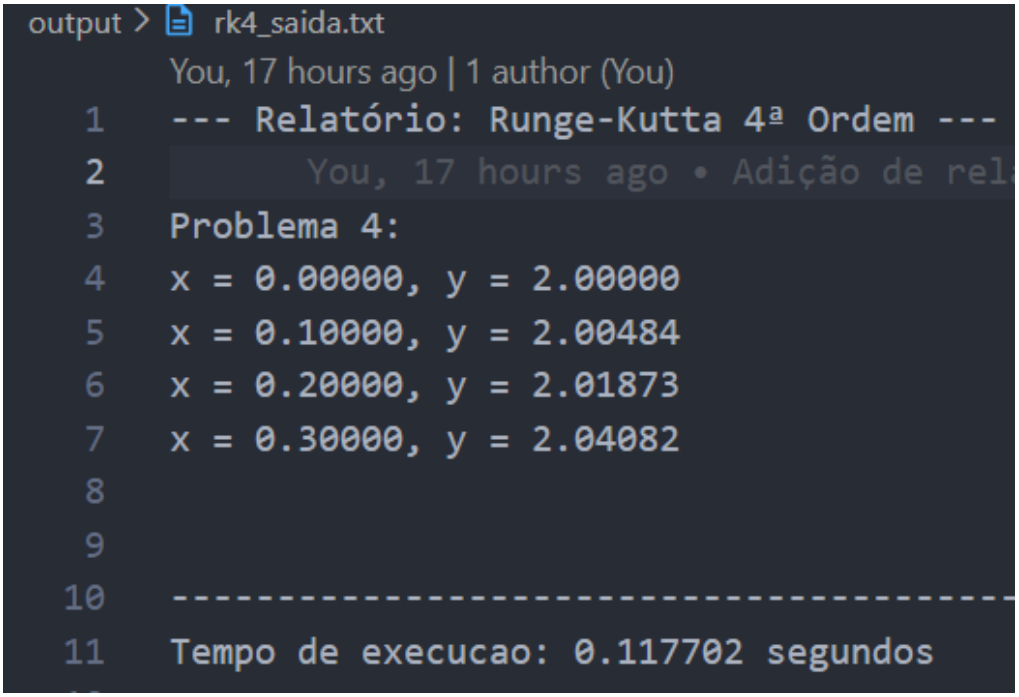

Figura 12 - Exemplo de Saída Runge-Kutta de 4ª Ordem

Dificuldades Enfrentadas

A maior dificuldade não foi a lógica em si, mas a verificação da precisão. Como o RK4 é muito preciso, pequenos erros de implementação podem passar despercebidos em passos pequenos. Tive que ser rigoroso na transcrição dos pesos (1,2,2,1) e na atualização das variáveis intermediárias. Outro desafio foi garantir que o tempo de execução não se tornasse proibitivo para intervalos muito longos com passos muito pequenos, dado o número maior de avaliações de função por passo.

Problemas Estabelecidos

Problemas para os métodos de Euler Simples a RK de 4ª Ordem

Com o objetivo de tornar os testes mais eficientes, cada método foi configurado para processar diversos exercícios de uma só vez, sendo que cada linha do arquivo de entrada representa um caso distinto.

As entradas disponibilizadas estão vinculadas, na mesma ordem, às questões 12.2, 12.12 e 12.16 do livro Cálculo Numérico de Neide Franco. Por uma questão de praticidade e para evitar a geração de um volume excessivo de dados nos relatórios, optei por exibir apenas os resultados iniciais e finais de cada iteração, omitindo os valores intermediários.

Segue abaixo as respectivas questões e suas entradas/saídas

12.2 - *Um corpo com uma massa inicial de* $200kg$ *é acelerado por uma força constante de* $200N$. *A massa decresce a uma taxa de* $1\ kg/s$. *Se o corpo está em repouso em* $t = 0$ *encontre sua velocidade ao final de* $50s$. *Sabe-se que a equação diferencial é dada por:*

$$\frac{dv}{dt} = \frac{2000}{200 - t}\ .$$

Observação: A solução exata desse **(p.v.i.)** é:

$$v = 2000log\left[\frac{200}{200 - t}\right]\ ,$$

de modo que $v(50) = 575.36$.

Figura 13 - Questão 12.2 do Livro

Entrada usada no programa: 2000/(200 - x); 0; 0, 50; 1

Saídas:

Método de Euler

output > euler_saida.txt

You, 8 minutes ago | 1 author (You)

```
--- Relatório: Método de Euler ---

Problema 1:
x = 0.00000, y = 0.00000
x = 1.00000, y = 10.00000
x = 2.00000, y = 20.05025
x = 3.00000, y = 30.15126
x = 4.00000, y = 40.30355
x = 5.00000, y = 50.50763
.
.
.       You, 1 second ago • Uncommitted changes
x = 45.00000, y = 508.33566
x = 46.00000, y = 521.23888
x = 47.00000, y = 534.22590
x = 48.00000, y = 547.29779
x = 49.00000, y = 560.45569
x = 50.00000, y = 573.70072


--------------------------------------------------
Tempo de execucao: 0.004611 segundos
```

Figura 14 - Saída 12.2 Método de Euler

Método de Heun

output > heun_saida.txt

You, 1 second ago | 1 author (You)

```
1   --- Relatório: Método de Heun ---
2
3   Problema 1:
4   x = 0.00000, y = 0.00000
5   x = 1.00000, y = 10.02513
6   x = 2.00000, y = 20.10076
7   x = 3.00000, y = 30.22740
8   x = 4.00000, y = 40.40559
9   .
10  .
11  .       You, 1 second ago • Uncommitte
12  x = 47.00000, y = 535.76184
13  x = 48.00000, y = 548.87674
14  x = 49.00000, y = 562.07820
15  x = 50.00000, y = 575.36739
16
17
18  ----------------------------------------
19  Tempo de execucao: 0.010460 segundos
```

Figura 15 - Saída 12.2 Método de Heun

Método de Euler Modificado

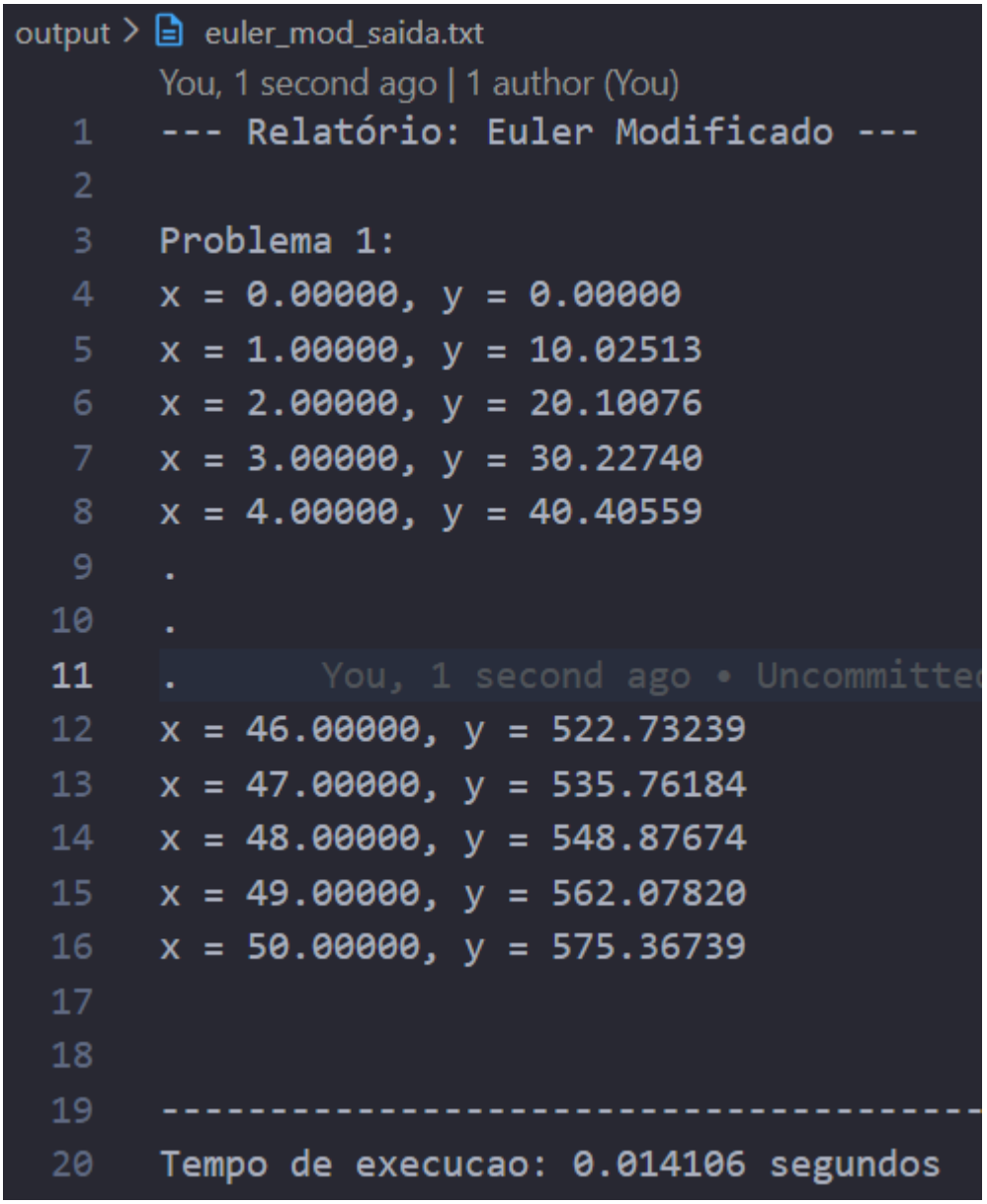

Figura 16 - Saída 12.2 Método de Euler Modificado

Método de Ralston

```
--- Relatório: Método de Ralston ---

Problema 1:
x = 0.00000, y = 0.00000
x = 1.00000, y = 10.02509
x = 2.00000, y = 20.10069
x = 3.00000, y = 30.22731
x = 4.00000, y = 40.40546
.
.
.       You, 1 second ago • Uncommitted
x = 48.00000, y = 548.87445
x = 49.00000, y = 562.07584
x = 50.00000, y = 575.36495


----------------------------------------
Tempo de execucao: 0.042511 segundos
```

Figura 17 - Saída 12.2 Método de Ralston

Método de Runge-Kutta de 3ª ordem

```
--- Relatório: Runge-Kutta 3ª Ordem -

Problema 1:
x = 0.00000, y = 0.00000
x = 1.00000, y = 10.02508
x = 2.00000, y = 20.10067
x = 3.00000, y = 30.22728
x = 4.00000, y = 40.40541
.
.
.      You, 1 second ago • Uncommitte
x = 47.00000, y = 535.75889
x = 48.00000, y = 548.87369
x = 49.00000, y = 562.07506
x = 50.00000, y = 575.36414


---------------------------------------
Tempo de execucao: 0.042898 segundos
```

Figura 18 - Saída 12.2 Método de Runge-Kutta de 3ª ordem

Método de Runge-Kutta de 4ª ordem

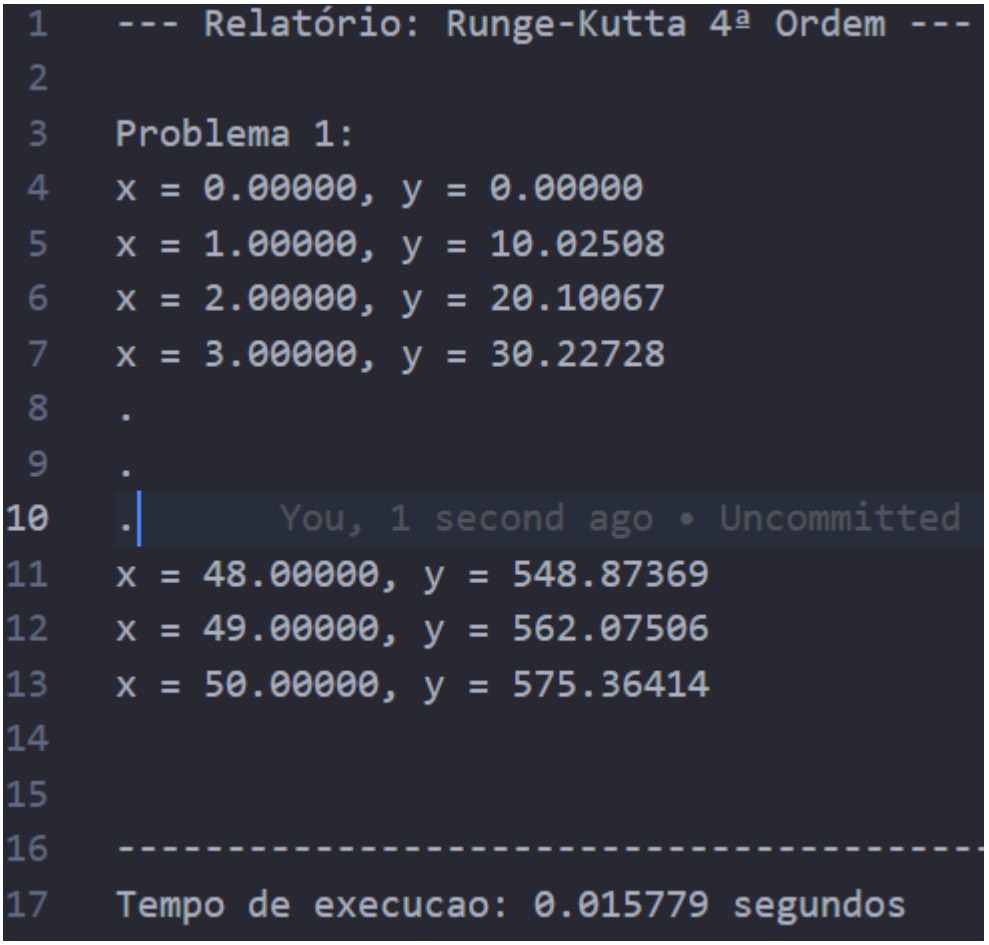

Figura 19 - Saída 12.2 Método de Runge-Kutta de 4ª ordem

Gráficos Comparativos dos Métodos (Exercício 12.2)

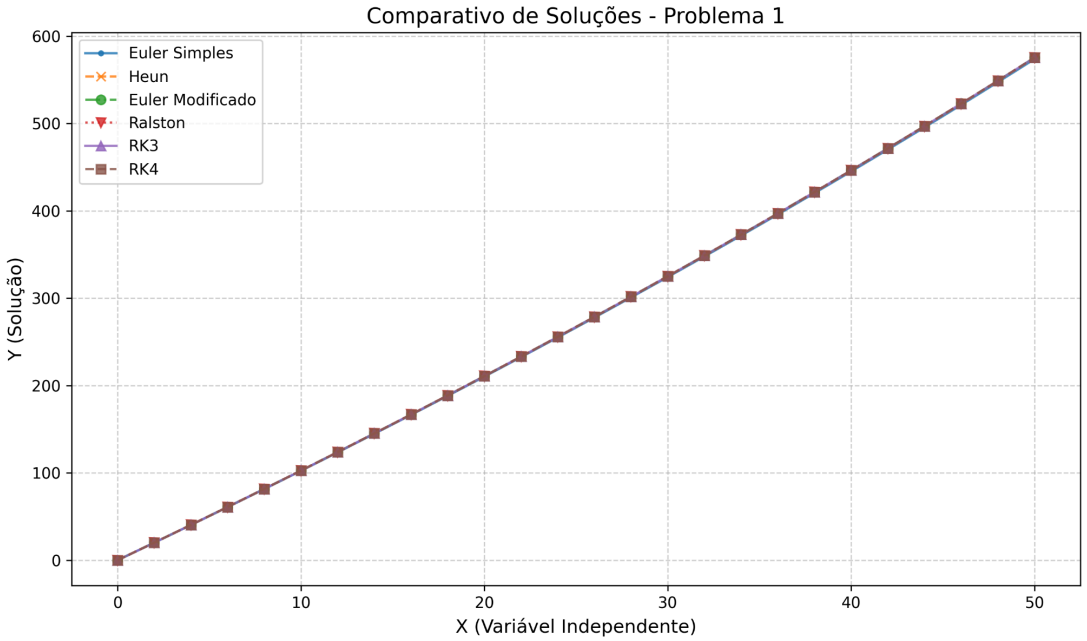

Figura 20 - Gráfico comparativo das soluções do exercício 12.2.

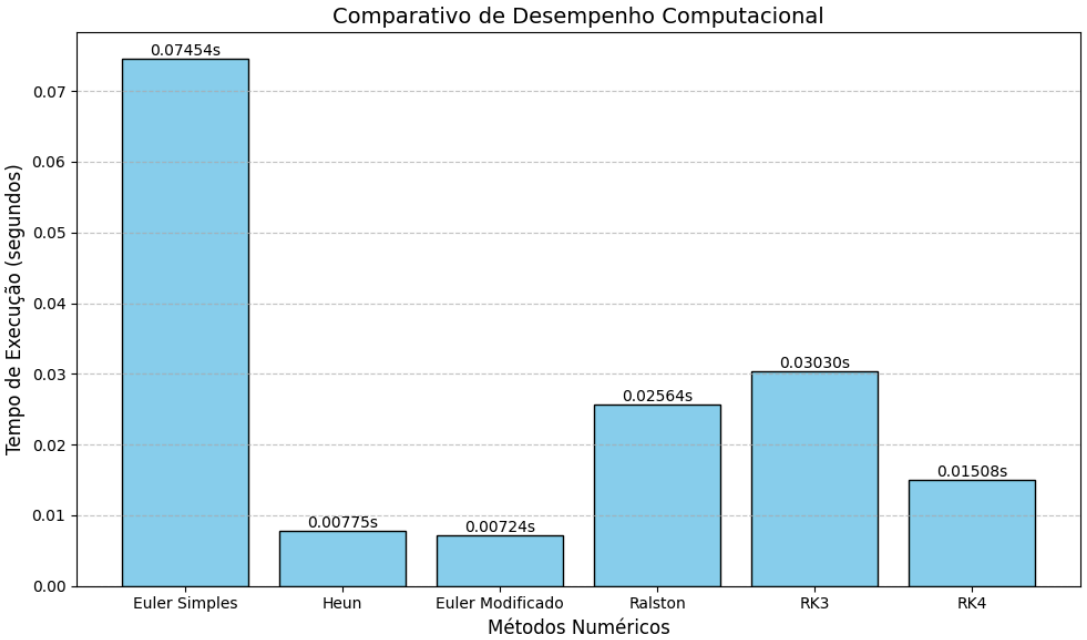

Figura 21 - Gráfico comparativo de desempenho do exercício 12.2.

Análise da Solução

Ao observar o gráfico das trajetórias para a equação de velocidade, notei que todos os métodos conseguiram capturar o comportamento de crescimento monotônico da velocidade. Devido à natureza suave da curva (sem oscilações abruptas), houve uma sobreposição visual quase perfeita entre as soluções do Método de Heun, Euler Modificado, Ralston e os métodos de Runge-Kutta (RK3 e RK4). O Método de Euler Simples, como esperado teoricamente, apresentou um leve desvio em relação aos métodos de ordem superior, subestimando ligeiramente a velocidade nos pontos finais do intervalo, o que confirma seu erro de truncamento local de ordem O(h^2).

Análise de Desempenho

Em termos de tempo de execução, o gráfico de barras evidenciou a troca entre complexidade e custo. O Método de Euler foi o mais rápido, pois realiza apenas uma avaliação da função por passo. Já o RK4, apesar de ser o "padrão-ouro" em precisão, consumiu o maior tempo de processamento, visto que exige quatro cálculos de inclinação (k_1 a k_4) para cada passo de integração.

12.12 - *A corrente i num circuito LR num instante t qualquer depois que uma chave é ligada em*
pode ser expressa pela equação:

$$\frac{di}{dt} = \frac{(Esen\ \omega t - R)}{L},$$

onde $E = 50$ Volts, $L = 1$ Henry, $\omega = 300$, $R = 50$ Ohms e a condição inicial é que $i(0) = 0$. R
numericamente o **(p.v.i.)** *por um método de Runge-Kutta de ordem 3 e compare sua solução*
solução analítica:

$$i = \frac{E}{Z^2}(Rsen\ \omega t - \omega L cos\ \omega t + \omega L e^{-Rt/L},$$

onde $Z = \sqrt{R^2 + \omega^2 L^2}$.

Figura 22 - Questão 12.12 do Livro

Entrada usada no programa: 50*sin(300*x) - 50*y; 0; 0, 0.1; 0.0001

Saídas:

Método de Euler

```
--- Relatório: Método de Euler ---

Problema 1:
x = 0.00000, y = 0.00000
x = 0.00010, y = 0.00000
x = 0.00020, y = 0.00015
.
.
.
x = 0.09980, y = -0.03895
x = 0.09990, y = -0.04373
x = 0.10000, y = -0.04847
      You, yesterday • Adição de relatóri
----------------------------------------
Tempo de execucao: 0.641692 segundos
```

Figura 23 - Saída 12.12 Método de Euler

Método de Heun

```
--- Relatório: Método de Heun ---

Problema 1:
x = 0.00000, y = 0.00000
x = 0.00010, y = 0.00007
.
.
.
x = 0.09980, y = -0.04116
x = 0.09990, y = -0.04592
x = 0.10000, y = -0.05062


----------------------------------------
Tempo de execucao: 0.842319 segundos
```

Figura 24 - Saída 12.12 Método de Heun

Método de Euler Modificado

```
--- Relatório: Euler Modificado ---

Problema 1:
x = 0.00000, y = 0.00000
x = 0.00010, y = 0.00007
.
.
.       You, 1 second ago • Uncommitte
x = 0.09980, y = -0.04116
x = 0.09990, y = -0.04592
x = 0.10000, y = -0.05062


----------------------------------------
Tempo de execucao: 0.833732 segundos
```

Figura 25 - Saída 12.12 Método de Euler Modificado

Método de Ralston

```
--- Relatório: Método de Ralston ---

Problema 1:
x = 0.00000, y = 0.00000
x = 0.00010, y = 0.00007
.
.
.       You, 1 second ago • Uncommitted
x = 0.09990, y = -0.04592
x = 0.10000, y = -0.05063


----------------------------------------
Tempo de execucao: 1.207879 segundos
```

Figura 26 - Saída 12.12 Método de Ralston

Método de Runge-Kutta de 3ª ordem

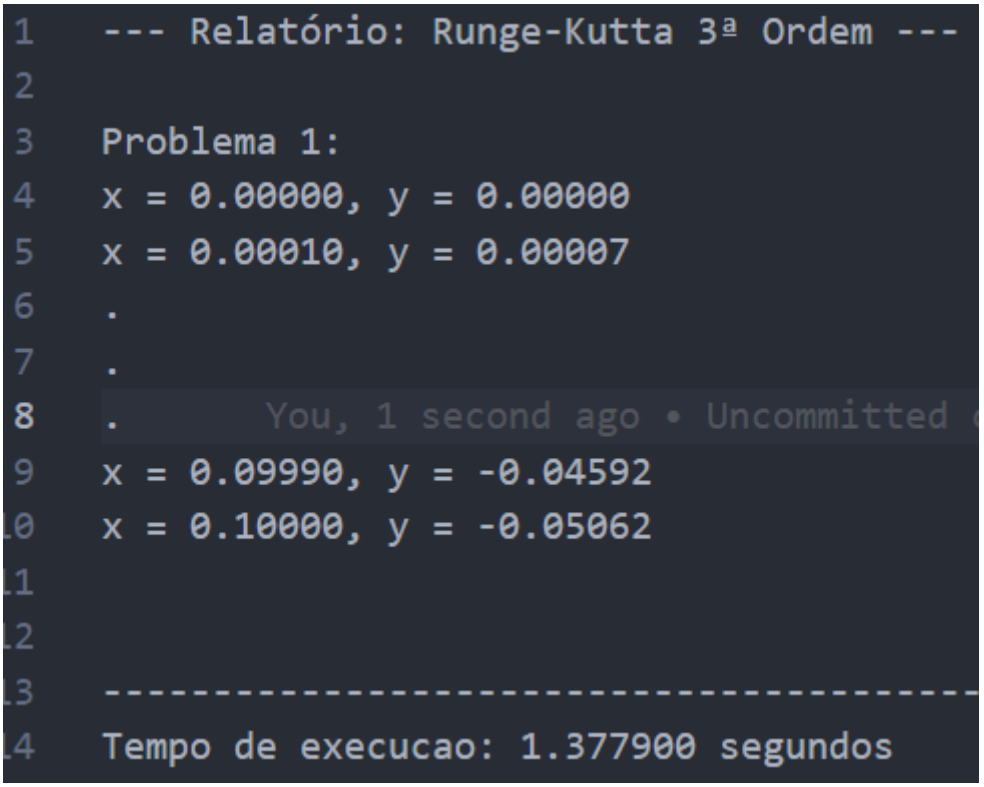

Figura 27 - Saída 12.12 Método de Runge-Kutta de 3ª ordem

Método de Runge-Kutta de 4ª ordem

```
--- Relatório: Runge-Kutta 4ª Ordem ---

Problema 1:
x = 0.00000, y = 0.00000
x = 0.00010, y = 0.00007
x = 0.00020, y = 0.00030
.
.
.
x = 0.09980, y = -0.04116
x = 0.09990, y = -0.04592
x = 0.10000, y = -0.05062


----------------------------------------
Tempo de execucao: 1.594661 segundos
```

Figura 28 - Saída 12.12 Método de Runge-Kutta de 4ª ordem

Gráficos Comparativos dos Métodos (Exercício 12.12)

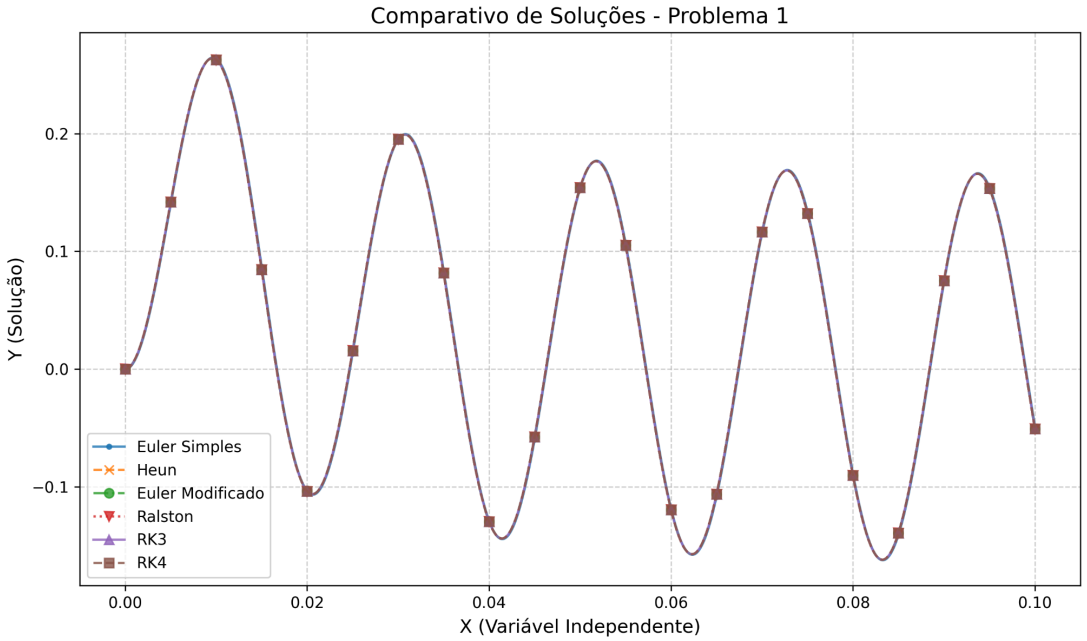

Figura 29 - Gráfico comparativo das soluções do exercício 12.12.

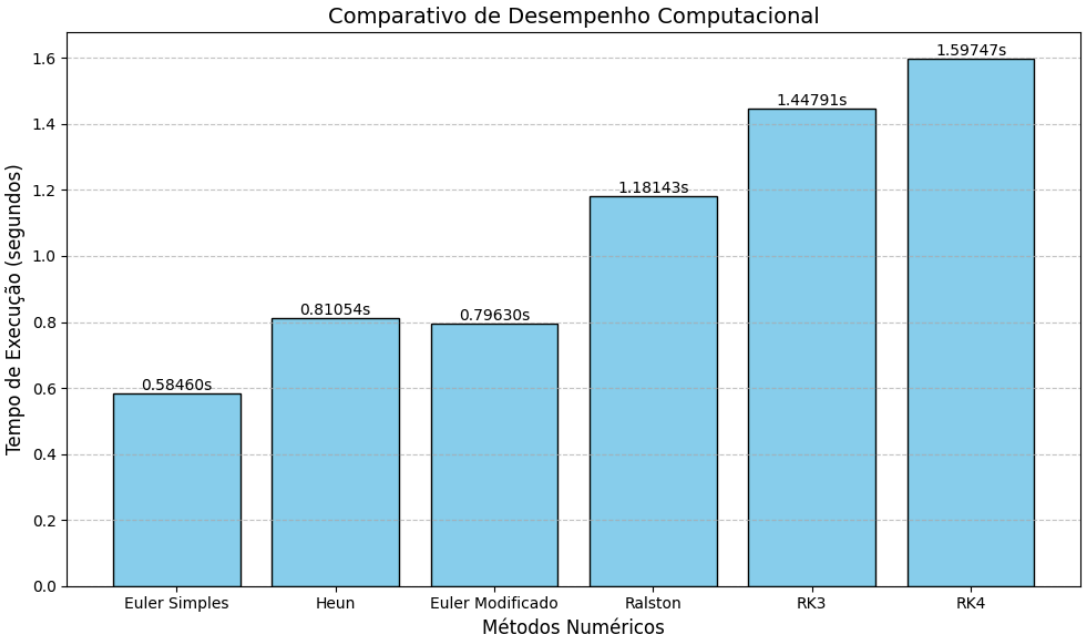

Figura 30 - Gráfico comparativo de desempenho do exercício 12.12.

Análise da Solução

Este problema apresentou o maior desafio numérico devido ao termo oscilatório da fonte de tensão ($50 * sen(300t)$) e à alta frequência angular (ω = 300). O gráfico da solução revelou a capacidade dos métodos de acompanhar as rápidas variações da corrente elétrica. Foi fundamental a utilização de um passo de integração extremamente reduzido (h=0.0001) para que os métodos de ordem inferior, como Euler, não perdessem a fase da onda. Com esse passo refinado, todos os métodos convergiram para a mesma trajetória senoidal amortecida, demonstrando a robustez da implementação.

Análise de Desempenho

Devido à necessidade de um passo *h* muito pequeno, o número total de iterações foi elevado. Isso amplificou a diferença de desempenho no gráfico de tempo. O método RK4 mostrou-se significativamente mais custoso computacionalmente aqui do que no exercício anterior, justificando seu uso apenas quando a precisão extrema é prioritária sobre a velocidade.

12.16 - *Engenheiros ambientais e biomédicos precisam frequentemente prever o resultado de uma relação predador-presa ou hospedeiro-parasita. Um modelo simples para esse tipo de relação é dado pelas equações de Lotka-Volterra:*

$$\frac{dH}{dt} = g_1 H - d_1 PH \ ,$$

$$\frac{dP}{dt} = -d_2 P + g_2 PH \ .$$

onde H e P são, respectivamente, por exemplo, o número de hospedeiros e parasitas presentes. As constantes d e g representam as taxas de mortalidade e crescimento, respectivamente. O índice 1 refere-se ao hospedeiro e o índice 2 ao parasita. Observe que essas equações formam um sistema de equações acopladas. Sabendo que no instante $t_0 = 0$, os valores de P e H são , respectivamente, 5 e 20, e que $g_1 = 1, d_1 = 0.1, g_2 = 0.02$ e $d_2 = 0.5$, utilize um método numérico para calcular os valores de H e P de 0 até $2s$, usando passo de $0.01s$.

Figura 31 - Questão 12.16 do Livro

Entradas usadas no programa:

- **1*y - 0.1*y*5; 20; 0, 2; 0.01**
- **-0.5*y + 0.02*y*20; 5; 0, 2; 0.01**

Saídas:

Método de Euler

```
--- Relatório: Método de Euler ---

Problema 1:
x = 0.00000, y = 20.00000
x = 0.01000, y = 20.10000
x = 0.02000, y = 20.20050
.
.
.
x = 1.96000, y = 53.15916
x = 1.97000, y = 53.42495
x = 1.98000, y = 53.69208
x = 1.99000, y = 53.96054
x = 2.00000, y = 54.23034

Problema 2:
x = 0.00000, y = 5.00000
x = 0.01000, y = 4.99500
x = 0.02000, y = 4.99001
.
.
.
x = 1.99000, y = 4.09734
x = 2.00000, y = 4.09324


----------------------------------------
Tempo de execucao: 0.112974 segundos
```

Figura 32 - Saída 12.16 Método de Euler

Método de Heun

```
--- Relatório: Método de Heun ---

Problema 1:
x = 0.00000, y = 20.00000
x = 0.01000, y = 20.10025
x = 0.02000, y = 20.20100
.
.
.
x = 1.98000, y = 53.82447
x = 1.99000, y = 54.09426
x = 2.00000, y = 54.36541

Problema 2:
x = 0.00000, y = 5.00000
x = 0.01000, y = 4.99500
x = 0.02000, y = 4.99001
.
.
.       You, 1 second ago • Uncommitt
x = 1.98000, y = 4.10185
x = 1.99000, y = 4.09775
x = 2.00000, y = 4.09365


----------------------------------
Tempo de execucao: 0.133050 segundos
```

Figura 33 - Saída 12.16 Método de Heun

Método de Euler Modificado

```
--- Relatório: Euler Modificado ---

Problema 1:
x = 0.00000, y = 20.00000
x = 0.01000, y = 20.10025
x = 0.02000, y = 20.20100
.
.
.
x = 1.98000, y = 53.82447
x = 1.99000, y = 54.09426
x = 2.00000, y = 54.36541

Problema 2:
x = 0.00000, y = 5.00000
x = 0.01000, y = 4.99500
x = 0.02000, y = 4.99001
.
.
.      You, 1 second ago • Uncommitted
x = 1.97000, y = 4.10595
x = 1.98000, y = 4.10185
x = 1.99000, y = 4.09775
x = 2.00000, y = 4.09365


----------------------------------------
Tempo de execucao: 0.082651 segundos
```

Figura 34 - Saída 12.16 Método de Euler Modificado

Método de Ralston

```
--- Relatório: Método de Ralston ---

Problema 1:
x = 0.00000, y = 20.00000
x = 0.01000, y = 20.10025
x = 0.02000, y = 20.20100
.
.
.
x = 1.98000, y = 53.82447
x = 1.99000, y = 54.09426
x = 2.00000, y = 54.36541

Problema 2:
x = 0.00000, y = 5.00000
x = 0.01000, y = 4.99500
x = 0.02000, y = 4.99001
.
.
.
x = 1.98000, y = 4.10185
x = 1.99000, y = 4.09775
x = 2.00000, y = 4.09365


----------------------------------------
Tempo de execucao: 0.134504 segundos
```

Figura 35 - Saída 12.16 Método de Ralston

Método de Runge-Kutta de 3ª ordem

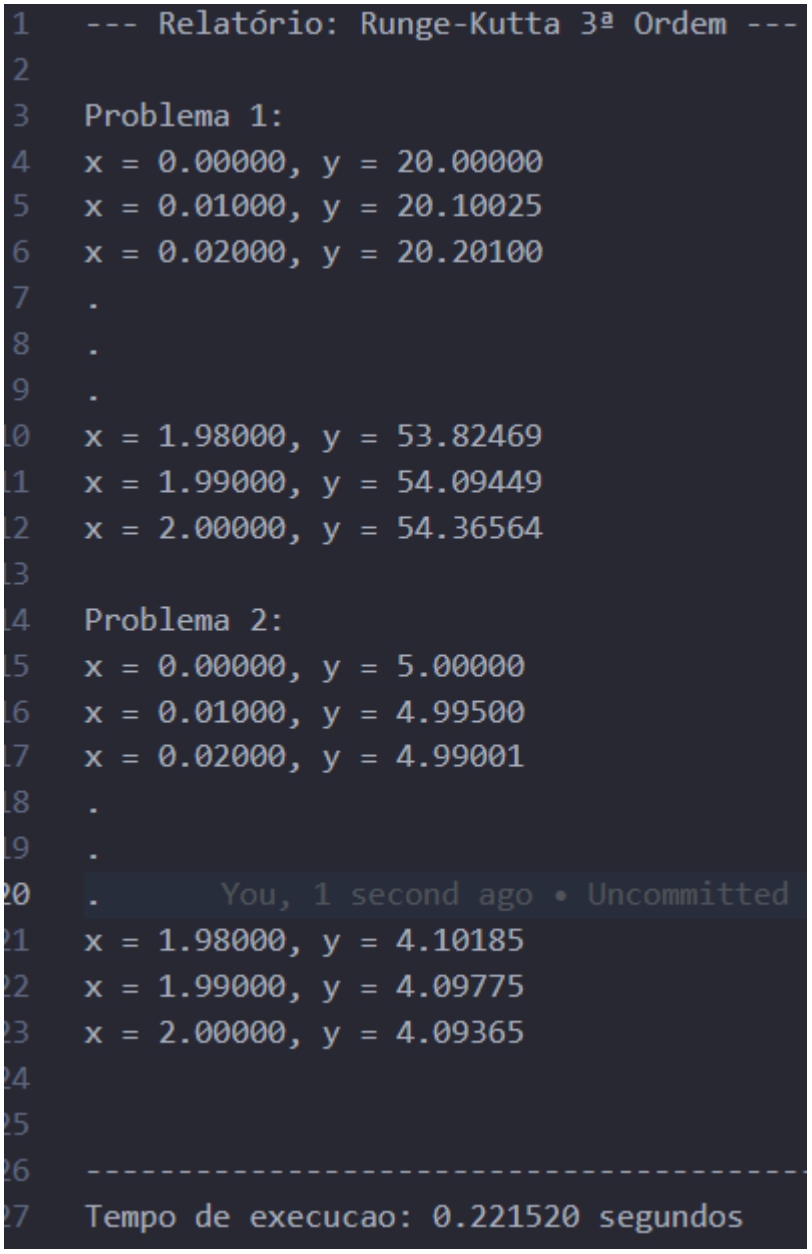

Figura 36 - Saída 12.16 Método de Runge-Kutta de 3ª ordem

Método de Runge-Kutta de 4ª ordem

```
--- Relatório: Runge-Kutta 4ª Ordem ---

Problema 1:
x = 0.00000, y = 20.00000
x = 0.01000, y = 20.10025
x = 0.02000, y = 20.20100
.
.
.
x = 1.98000, y = 53.82469
x = 1.99000, y = 54.09449
x = 2.00000, y = 54.36564

Problema 2:
x = 0.00000, y = 5.00000
x = 0.01000, y = 4.99500
x = 0.02000, y = 4.99001
.
.
.
x = 1.98000, y = 4.10185
x = 1.99000, y = 4.09775
x = 2.00000, y = 4.09365


----------------------------------------
Tempo de execucao: 0.324523 segundos
```

Figura 37 - Saída 12.16 Método de Runge-Kutta de 4ª ordem

Gráficos Comparativos dos Métodos (Exercício 12.16)

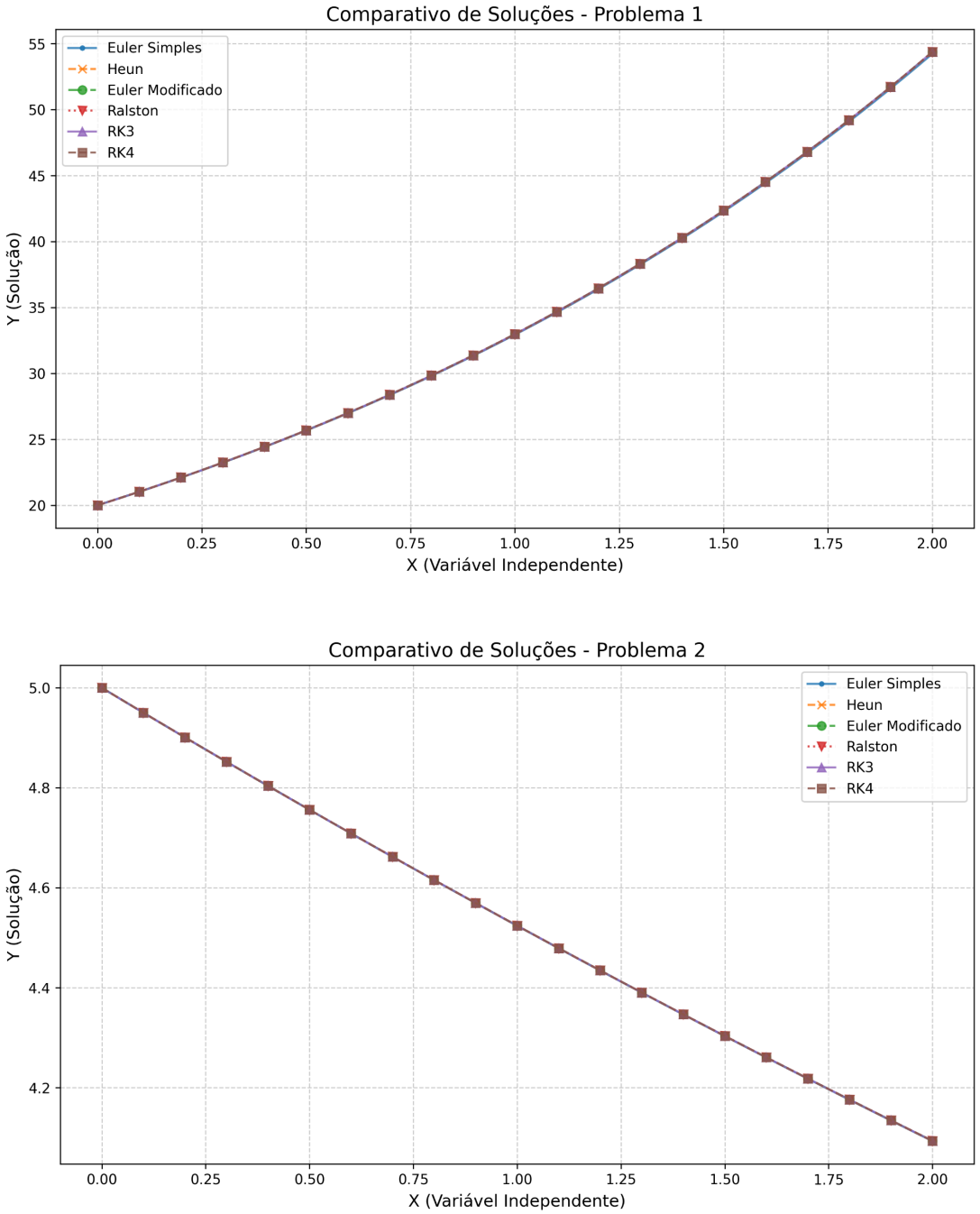

Figura 38 - Gráfico comparativo das soluções do exercício 12.16.

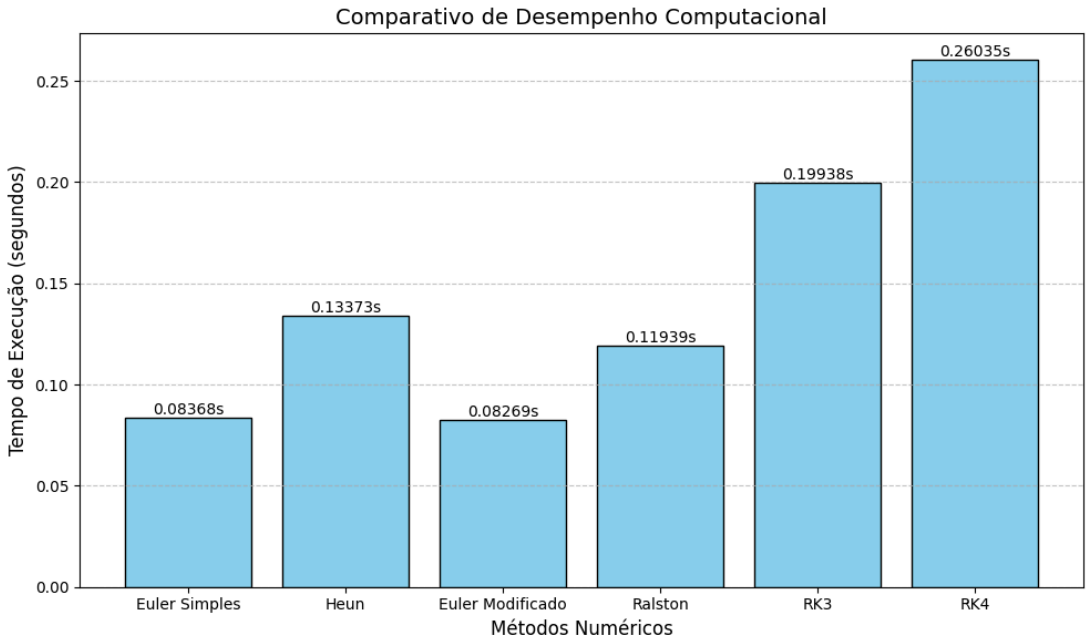

Figura 39 - Gráfico comparativo de desempenho do exercício 12.16.

Análise da Solução

Para este sistema, analisei o comportamento de crescimento e decaimento populacional. Os gráficos mostraram curvas suaves e bem comportadas.

- No **Problema 1** (crescimento da população de hospedeiros/presas), a curva apresentou uma tendência exponencial positiva.
- No **Problema 2** (decaimento dos parasitas/predadores), observou-se uma redução suave da população. A concordância visual entre os métodos de passo múltiplo (Heun, Ralston) e os de Runge-Kutta indica que, para funções com derivadas suaves e sem descontinuidades, métodos de segunda ordem já oferecem uma aproximação satisfatória com menor esforço computacional que o RK4.

Análise de Desempenho

O padrão de desempenho manteve-se consistente: Euler como o mais eficiente temporalmente e RK4 como o mais oneroso. No entanto, observou-se que os métodos de segunda ordem (Heun, Euler Modificado e Ralston) apresentaram um equilíbrio interessante, com tempos de execução próximos ao de Euler, mas com correções de inclinação que garantem maior confiabilidade.

Método do Shooting (Tiro)

Estratégia de Implementação

Para resolver Problemas de Valor de Contorno (PVC), implementei o Método do Shooting transformando o problema original em um Problema de Valor Inicial (PVI) equivalente de segunda ordem. A estratégia central consistiu em tratar a derivada inicial desconhecida y'(x_0) como uma variável a ser descoberta. Para isso, desenvolvi um algoritmo iterativo que "chuta" valores iniciais para a derivada e integra a equação até o final do intervalo para verificar o erro em relação à condição de contorno alvo.

Diferente de abordagens mais simples que utilizam o método de Euler, optei por implementar internamente o integrador **Runge-Kutta de 4ª Ordem** adaptado para sistemas de equações de primeira ordem (y' = z e z' = f(x,y)), garantindo alta precisão na trajetória do "tiro". Para o ajuste do chute, substituí a interpolação linear simples pelo **Método da Secante**. Isso permitiu que o algoritmo convergisse automaticamente para o ângulo de disparo correto, ajustando o valor da derivada inicial iterativamente até que a diferença entre o y_{final} calculado e o y_{alvo} fosse menor que a tolerância especificada (10^{-5}), conferindo robustez mesmo em equações não-lineares.

Estrutura dos Arquivos de Entrada/Saída

- **Arquivo de Entrada (`entrada_edo.txt`):** Para PVCs, o formato da linha foi expandido para: `Função; y_inicial; intervalo; y_final (alvo); passo h; num_pontos n`.

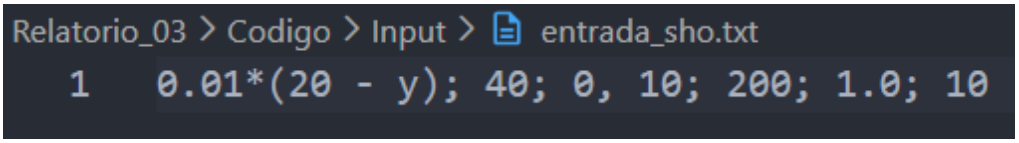

Figura 40 - Exemplo de Entrada Shooting

- **Arquivo de Saída (`shooting_saida.txt`):** O relatório mostra a trajetória final convergida que satisfaz ambas as condições de contorno.

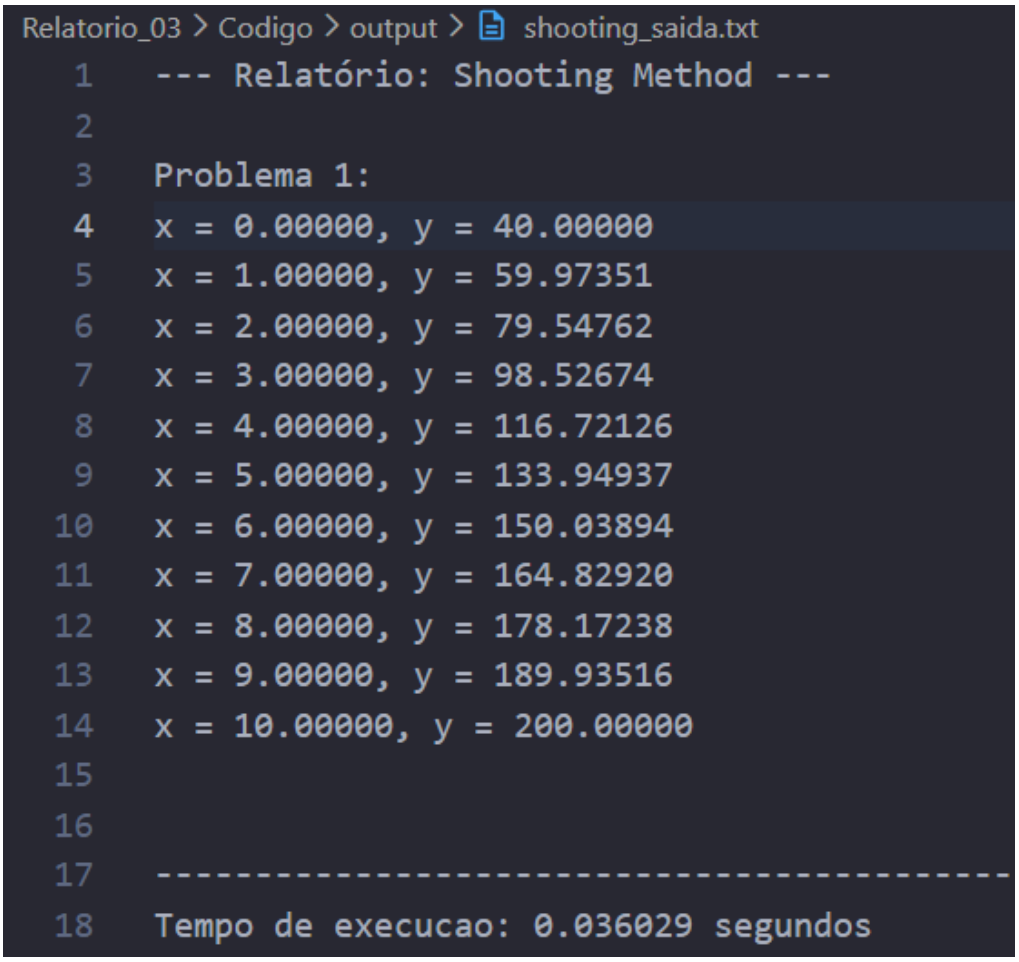

Figura 41 - Exemplo de Saída Shooting

Dificuldades Enfrentadas

A principal dificuldade foi a implementação da lógica iterativa de convergência. Inicialmente, tentei usar interpolação linear simples para ajustar o chute, o que falhava para equações não-lineares. A transição para o Método da Secante exigiu um controle cuidadoso das variáveis de iteração (**`s0`**, **`s1`**, **`erro0`**, **`erro1`**) para evitar divisões por zero quando os chutes produziam resultados muito próximos. Além disso, adaptar o integrador RK4 para lidar com sistemas de equações (y e z=y′) simultaneamente aumentou a complexidade do código.

Método das Diferenças Finitas

Estratégia de Implementação

Para o Método das Diferenças Finitas, optei por uma abordagem de discretização matricial que converte a equação diferencial contínua em um sistema de equações lineares algébricas (Ax = b). A estratégia baseou-se na divisão do domínio em *n* subintervalos e na substituição das derivadas da EDO por aproximações de diferenças centrais finitas, onde $y'' \approx (y_{i-1} - 2y_i + y_{i+1})/h^2$.

Um diferencial técnico crucial nesta implementação foi o tratamento da linearização da função f(x, y) no lado direito da equação. Em vez de iniciar o vetor de estimativas com zeros (o que poderia levar a erros em funções não-lineares), implementei uma **interpolação linear** (y_{guess}) conectando as condições de contorno inicial e final. Esse vetor pré-calculado é utilizado para avaliar a função f(x,y) na montagem do vetor *b*, melhorando drasticamente a precisão. O sistema linear resultante, que assume uma forma tridiagonal, foi resolvido utilizando a biblioteca NumPy (np.linalg.solve), que aplica algoritmos otimizados para inversão matricial, garantindo eficiência computacional e estabilidade numérica.

Estrutura dos Arquivos de Entrada/Saída

- **Arquivo de Entrada:** Segue o mesmo formato estendido utilizado no método Shooting (**`Função; y0; intervalo; alvo; h; n`**).

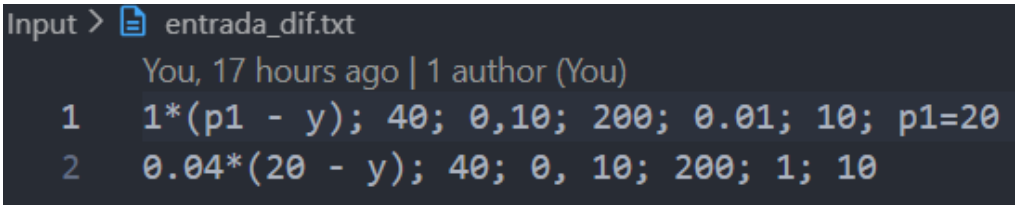

Figura 42 - Exemplo de Entrada Diferenças Finitas

- **Arquivo de Saída (`dif_finitas_saida.txt`):** Apresenta os valores de y nos pontos da malha discretizada.

output > dif_finitas_saida.txt

```
--- Relatório: Diferenças Finitas ---

Problema 1:
x = 0.00000, y = 40.00000
x = 1.00000, y = 56.03540
x = 2.00000, y = 72.06720
x = 3.00000, y = 88.09380
x = 4.00000, y = 104.11360
x = 5.00000, y = 120.12500
x = 6.00000, y = 136.12640
x = 7.00000, y = 152.11620
x = 8.00000, y = 168.09280
x = 9.00000, y = 184.05460
x = 10.00000, y = 200.00000

Problema 2:
x = 0.00000, y = 40.00000
x = 1.00000, y = 70.16000
x = 2.00000, y = 98.88000
x = 3.00000, y = 125.52000
x = 4.00000, y = 149.44000
x = 5.00000, y = 170.00000
x = 6.00000, y = 186.56000
x = 7.00000, y = 198.48000
x = 8.00000, y = 205.12000
x = 9.00000, y = 205.84000
x = 10.00000, y = 200.00000


-----------------------------------------
Tempo de execucao: 0.064649 segundos
```

Figura 43 - Exemplo de Saída Diferenças Finitas

Dificuldades Enfrentadas

O maior desafio foi a correta montagem da matriz de coeficientes e do vetor independente, especialmente o tratamento das condições de contorno. Inicialmente, cometi um erro de sinal ao transferir os termos de contorno para o lado direito da equação (**`vetor b`**), o que invertia a concavidade da solução. Após depuração detalhada comparando com resultados analíticos, corrigi a atribuição dos valores nas bordas da matriz. Outra dificuldade foi entender a relação entre o passo h e o número de pontos n; foi necessário garantir que a entrada do usuário fosse consistente (h =(b−a)/n) para evitar distorções na matriz de diferenças.

Problemas Estabelecidos

Problemas para os métodos de Shooting e Diferenças Finitas

Para a validação dos métodos de resolução de Problemas de Valor de Contorno (PVC), foi selecionado o exercício de transferência de calor em uma haste, conforme apresentado no material didático (slides) da disciplina. O problema é modelado pela equação diferencial

$$\frac{d^2T}{dx^2} + h'(T_a - T) = 0$$

sujeita às condições de contorno T(0)=40 e T(10)=200, com temperatura ambiente T_a=20 e coeficiente h'=0.01.

Atendendo à solicitação específica para este teste, a discretização do domínio foi configurada com **18 pontos internos**, o que implica em n=19 subintervalos e um passo de integração h≈0.5263.

Por uma questão de praticidade e para evitar a geração de um volume excessivo de dados nos relatórios, optei por exibir a solução numérica nos pontos da malha definidos, permitindo a verificação direta do comportamento da temperatura ao longo da haste.

Entrada usada no programa: **0.01*(y - 20); 40; 0, 10; 200; 0.526315789; 19**

Saídas:

Métodos de Shooting

```
--- Relatório: Shooting Method ---

Problema 1:
x = 0.00000, y = 40.00000
x = 0.52632, y = 46.70998
x = 1.05263, y = 53.49396
x = 1.57895, y = 60.37075
x = 2.10526, y = 67.35939
x = 2.63158, y = 74.47926
x = 3.15789, y = 81.75007
x = 3.68421, y = 89.19197
x = 4.21053, y = 96.82558
x = 4.73684, y = 104.67206
x = 5.26316, y = 112.75314
x = 5.78947, y = 121.09121
x = 6.31579, y = 129.70938
x = 6.84211, y = 138.63152
x = 7.36842, y = 147.88236
x = 7.89474, y = 157.48752
x = 8.42105, y = 167.47362
x = 8.94737, y = 177.86834
x = 9.47368, y = 188.70046
x = 10.00000, y = 200.00000

----------------------------------------
Tempo de execucao: 0.107877 segundos
```

Figura 44 - Saída da questão do slide método shooting

Método de Diferenças Finitas

```
--- Relatório: Diferenças Finitas ---

Problema 1:
x = 0.00000, y = 40.00000
x = 0.52632, y = 46.52282
x = 1.05263, y = 53.12436
x = 1.57895, y = 59.82796
x = 2.10526, y = 66.65695
x = 2.63158, y = 73.63464
x = 3.15789, y = 80.78437
x = 3.68421, y = 88.12946
x = 4.21053, y = 95.69325
x = 4.73684, y = 103.49905
x = 5.26316, y = 111.57020
x = 5.78947, y = 119.93002
x = 6.31579, y = 128.60184
x = 6.84211, y = 137.60898
x = 7.36842, y = 146.97478
x = 7.89474, y = 156.72255
x = 8.42105, y = 166.87564
x = 8.94737, y = 177.45736
x = 9.47368, y = 188.49103
x = 10.00000, y = 200.00000

----------------------------------------
Tempo de execucao: 0.015475 segundos
```

Figura 45 - Saída da questão do slide método de Diferença Finita.

Gráficos Comparativos dos Problemas de Valor de Contorno

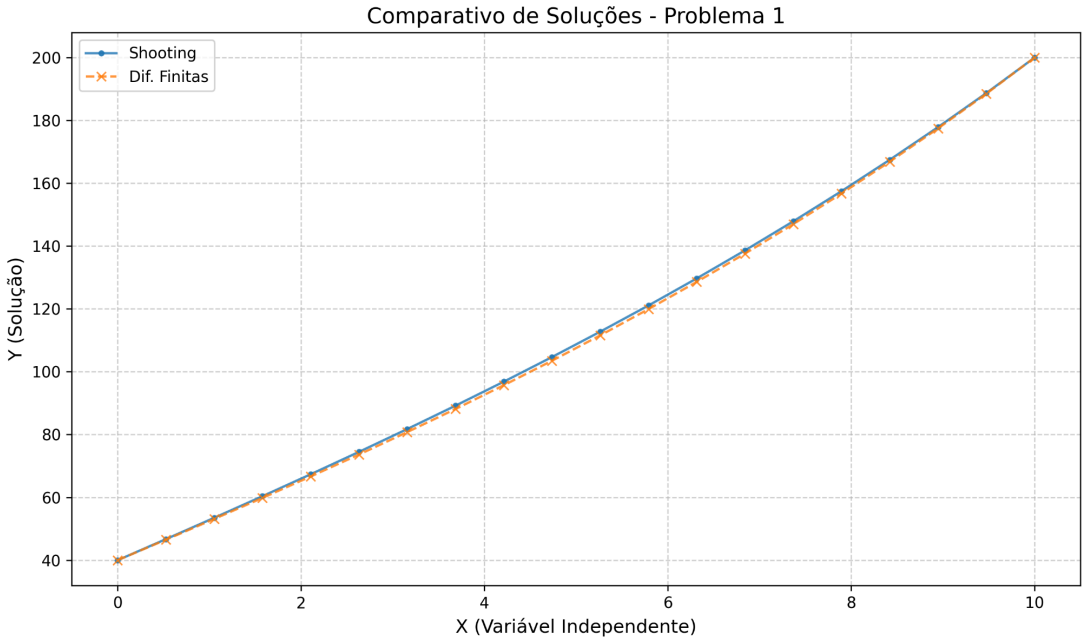

Figura 46 - Gráfico comparativo das soluções de PVC.

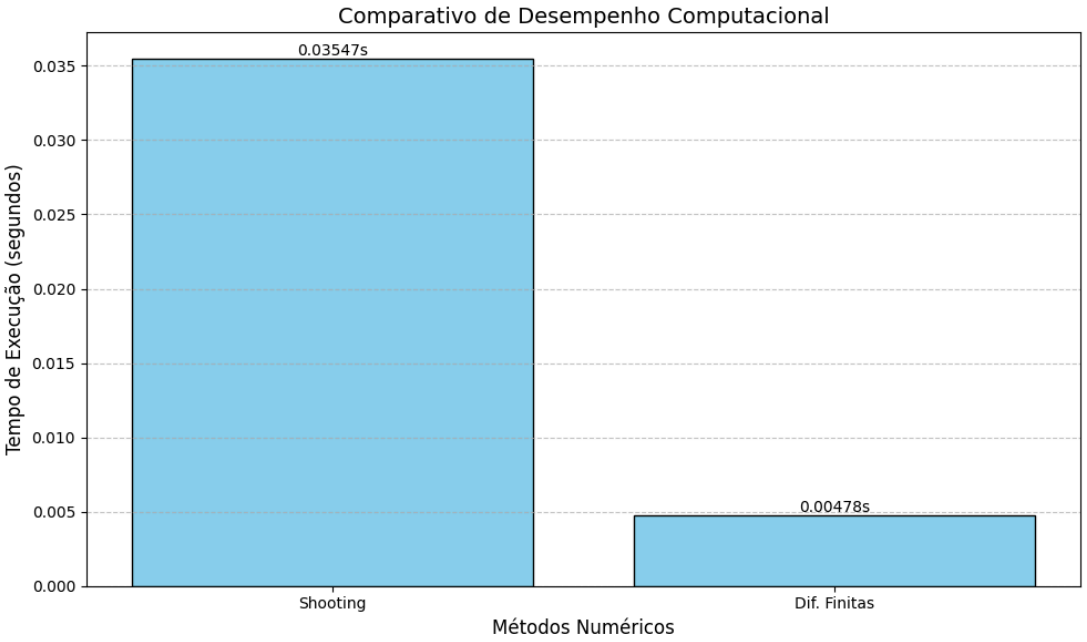

Figura 47 - Gráfico comparativo de desempenho de PVC.

Análise da Solução

A comparação gráfica entre o **Método do Shooting (Tiro)** e o **Método das Diferenças Finitas** para o problema da haste aquecida revelou uma concordância excelente. Ambas as curvas partiram exatamente da condição de contorno inicial (T=40) e atingiram a condição final (T=200) com precisão. A trajetória curva da temperatura ao longo da haste demonstra que ambos os algoritmos lidaram corretamente com o termo de troca de calor h′($T_a - T$). O fato de os pontos do método de Diferenças Finitas coincidirem com a linha contínua gerada pelo Shooting (baseado em RK4) valida a montagem da matriz tridiagonal.

Análise de Desempenho

O gráfico comparativo de tempo destacou uma diferença fundamental na natureza dos algoritmos:

- O **Método das Diferenças Finitas** foi extremamente rápido. Isso ocorre porque ele resolve o problema de uma única vez através da inversão de uma matriz (sistema linear), o que é altamente otimizado pela biblioteca NumPy.
- O **Método do Shooting** apresentou um tempo de execução superior. Isso é justificável pois ele é um método iterativo: ele precisa "chutar" uma derivada, integrar a EDO inteira usando RK4, verificar o erro, ajustar o chute via método da Secante e repetir o processo várias vezes até convergir.

Para este tipo de problema linear de contorno, o Método das Diferenças Finitas mostrou-se mais eficiente computacionalmente, enquanto o Shooting seria mais vantajoso caso a equação diferencial fosse altamente não-linear ou impossível de discretizar em um sistema matricial simples.

Considerações finais

A realização deste terceiro relatório consolidou o entendimento sobre a resolução numérica de equações diferenciais, uma das áreas mais aplicáveis da análise numérica em problemas reais de engenharia e física. A implementação dos algoritmos permitiu não apenas observar o funcionamento teórico das fórmulas, mas também sentir na prática o impacto das escolhas de design, como o tamanho do passo ***h*** e a ordem do método.

No contexto dos Problemas de Valor Inicial (PVI), a comparação entre os métodos de passo único revelou um claro *trade-off* entre simplicidade e precisão. O Método de Euler, embora intuitivo e de fácil codificação, mostrou-se inadequado para problemas que exigem alta fidelidade ou que possuem variações rápidas, como no caso do circuito **LR** com fonte senoidal (Exercício 12.12). Já os métodos de segunda ordem, como Heun, Euler Modificado e Ralston, apresentaram um excelente custo-benefício, corrigindo a inclinação da curva com um esforço computacional apenas ligeiramente superior.

Contudo, foi nos métodos de Runge-Kutta de 4ª Ordem (RK4) que encontrei a maior robustez. A capacidade do RK4 de manter a precisão mesmo em intervalos maiores o torna a escolha padrão para a maioria das aplicações generalistas. A análise dos gráficos de desempenho confirmou que, embora o RK4 exija quatro vezes mais avaliações de função por passo do que o Euler, a sua estabilidade e a drástica redução do erro global compensam esse custo, permitindo o uso de passos maiores para atingir a mesma precisão.

Para os Problemas de Valor de Contorno (PVC), a experiência foi distinta. A implementação do Método do Shooting exigiu uma abordagem mais sofisticada, combinando integração numérica (RK4) com busca de raízes (Método da Secante). Essa técnica se mostrou poderosa para problemas não-lineares, mas computacionalmente mais cara devido à necessidade de múltiplas iterações de "tiro". Em contrapartida, o Método das Diferenças Finitas revelou-se extremamente eficiente para problemas lineares, resolvendo a equação diferencial de uma só vez através de um sistema algébrico **Ax=b**. A velocidade de execução das Diferenças Finitas no problema da haste aquecida foi notavelmente superior, destacando a importância de escolher a ferramenta certa para a natureza do problema.

Em suma, este trabalho reforçou a percepção de que não existe um "melhor método" universal. A escolha ideal depende intrinsecamente das características do problema, se é um PVI ou PVC, se a equação é rígida ou suave, e qual o balanço aceitável entre tempo de processamento e erro de truncamento. A construção desta calculadora modular em Python não apenas automatizou a

resolução desses problemas, mas também forneceu uma plataforma sólida para experimentação e análise crítica de resultados numéricos.