Escola Politécnica da Universidade de São Paulo

PTC5725 – Introdução aos Métodos Espectrais

Relatório: Tarefa complementar da aula 4

Renan de Luca Avila

São Paulo, 12 de outubro de 2025

Sumário

1 Resumo

Este documento contempla os 3 itens da tarefa complementar da aula 4.

2 Enunciado

a) Resolver numericamente a seguinte ODE:

$$x \in \Omega \equiv [1,3] \subset \mathbb{R} \text{ tal que} \quad x\frac{du}{dx} + 2u = 4x^2, \quad \text{com } u(1) = 2.$$

b) Determine com a solução numérica o ponto

$$x \in [1,3] \quad \text{tal que} \quad u(x) = 4,$$

com pelo menos 10 dígitos de precisão.

c) Compare o resultado com

$$x_k = \sqrt{2+\sqrt{3}}\,.$$

Aproveite e mostre também que

$$x_k = 2\cos\Big(\frac{\pi}{12}\Big) = 2\sqrt{\frac{1+\cos(\pi/6)}{2}}.$$

3 Resolução do item *a*

3.1 Solução analítica

Primeiro, encontremos a solução analítica: consideremos o problema de valor inicial.

$$x\,u'(x) + 2u(x) = 4x^2, \qquad u(1) = 2, \qquad x \in [1,3].$$

Como $x > 0$ no domínio, multiplicamos a equação por x e reconhecemos a derivada do produto no lado esquerdo da equação:

$$x^2u'(x) + 2x\,u(x) = 4x^3,$$
$$\frac{d}{dx}\big(x^2u(x)\big) = 4x^3.$$

Integramos em x dos dois lados:

$$x^2u(x) = \int 4x^3\,dx = x^4 + C \quad \Longrightarrow \quad u(x) = x^2 + \frac{C}{x^2}.$$

Aplicando a condição inicial $u(1) = 2$:

$$2 = 1 + C \;\Longrightarrow\; C = 1.$$

Portanto, a solução fechada é

$$\boxed{u(x) = x^2 + x^{-2}}, \qquad x \in [1,3].$$

Observação Caso a ideia de multiplicar por x dos dois lados pareça pouco intuitiva, o mesmo resultado surge pelo método padrão do fator integrante para EDO linear de 1ª ordem, usando $\mu(x) = \exp\big(\int \frac{2}{x}\,dx\big) = x^2$, o que leva novamente a $\frac{d}{dx}(x^2u) = 4x^3$.

3.2 Solução numérica

Planejamento Nesta etapa, nós vamos resolver numericamente a equação diferencial

$$x\,u'(x) + 2u(x) = 4x^2, \qquad u(1) = 2, \qquad x \in [1, 3],$$

utilizando o método de espectral de colocação de pontos de Chebyshev-Gauss-Lobatto. O domínio físico $x \in [1, 3]$ é mapeado para o domínio computacional $\xi \in [-1, 1]$ por meio da transformação linear

$$x(\xi) = 2 + \xi, \qquad \frac{dx}{d\xi} = 1.$$

Escolhemos $N + 1$ pontos de colocação

$$\xi_j = \cos\left(\frac{\pi j}{N}\right), \qquad j = 0, 1, \ldots, N,$$

que são então mapeados para $x_j = 2 + \xi_j$. A matriz de derivada espectral D é construída a partir da fórmula clássica de Chebyshev. A equação diferencial é discretizada nos nós internos conforme

$$\mathrm{diag}(x)\, D\, \mathbf{u} + 2\,\mathbf{u} = 4\,\mathbf{x}^{\circ 2},$$

e a condição de contorno $u(1) = 2$ é imposta substituindo a primeira linha do sistema. Finalmente, nós vamos resolver o sistema linear para $\mathbf{u}$ e comparar com a solução analítica $u(x) = x^2 + x^{-2}$, avaliando o erro pelo máximo da diferença absoluta.

3.3 Construção do código

Nesta subseção, nós conectamos o *planejamento* da colocação de Chebyshev-Gauss–Lobatto à *implementação* em Python (Listagem 1), explicando cada etapa e como ela aparece no código.

(1) Grade espectral e mapeamento do domínio. Planejamento: escolher pontos CGL em $\xi \in [-1, 1]$ e mapear linearmente para $x \in [1, 3]$ via $x(\xi) = 2 + \xi$. No código:

```
xi = np.cos(np.pi * np.arange(N+1) / N)
x = 2 + xi
```

Comentário: como $dx/d\xi = 1$, a derivada em x coincide com a derivada em ξ na montagem da matriz.

(2) Matriz de derivada de Chebyshev D. Planejamento: construir D clássica em CGL. No código:

```
c = np.ones(N+1); c[0] = c[-1] = 2
D = np.zeros((N+1, N+1))
for i in range(N+1):
    for j in range(N+1):
        if i != j:
            D[i,j] = (c[i]/c[j]) * (-1)**(i+j) / (xi[i]-xi[j])
    D[i,i] = -np.sum(D[i,:])
```

Isso implementa a fórmula padrão para D em nós CGL, com correção dos termos diagonais para garantir que cada linha some a zero.

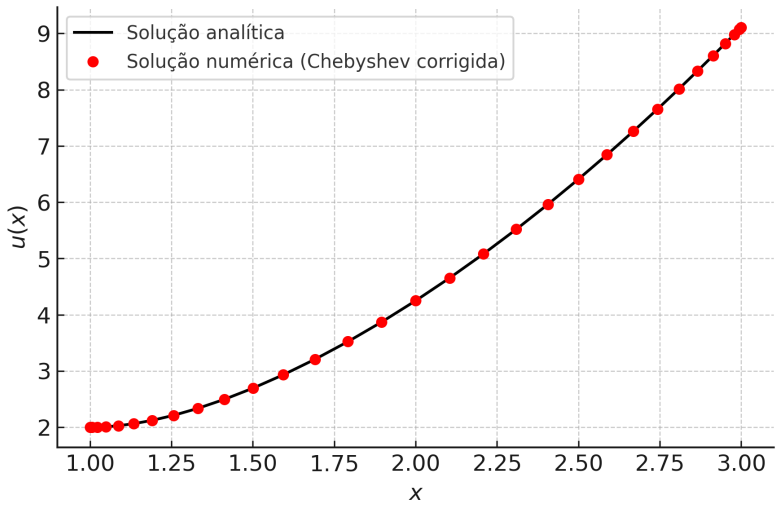

Figura 1: Comparação entre a solução analítica $u(x) = x^2 + x^{-2}$ e a solução numérica obtida pelo método de colocação de Chebyshev-Gauss-Lobatto.

(3) Discretização da EDO nos nós internos. Planejamento: discretizar

$$\mathrm{diag}(x)\, D\, \mathbf{u} + 2\,\mathbf{u} = 4\,\mathbf{x}^{\circ 2}.$$

No código:

```
A = np.diag(x) @ D + 2*np.eye(N+1)
b = 4*x**2
```

Aqui A representa o operador discreto e b o termo-força avaliado nos nós.

(4) Imposição da condição de contorno. Planejamento: impor $u(1) = 2$ no nó de fronteira correspondente a $x = 1$ (que é $\xi = -1$). No vetor `xi` usado, $\xi = -1$ é o *último* nó, logo substituímos a última linha:

```
A[-1,:] = 0; A[-1,-1] = 1
b[-1] = 2
```

Este passo implementa exatamente a estratégia planejada de substituir a equação de interior pela condição de contorno.

(5) Resolução do sistema e comparação com a solução analítica. Planejamento: resolver $A\mathbf{u} = \mathbf{b}$, computar erro e gerar figura. No código:

```
u_num = np.linalg.solve(A, b)
u_ana = x**2 + 1/x**2
erro_max = np.max(np.abs(u_num - u_ana))
```

Em seguida, a figura é produzida com as duas curvas (analítica e numérica) sobrepostas (Fig. 1).

3.4 Implementação em Python.

Código em 1.

```python
import numpy as np
import matplotlib.pyplot as plt

N = 30
xi = np.cos(np.pi * np.arange(N+1) / N)
x = 2 + xi

c = np.ones(N+1); c[0] = c[-1] = 2
D = np.zeros((N+1, N+1))
for i in range(N+1):
    for j in range(N+1):
        if i != j:
            D[i,j] = (c[i]/c[j]) * (-1)**(i+j) / (xi[i]-xi[j])
    D[i,i] = -np.sum(D[i,:])

A = np.diag(x) @ D + 2*np.eye(N+1)
b = 4*x**2

# Condição de contorno correta: u(1)=2 no último nó (x=1)
A[-1,:] = 0
A[-1,-1] = 1
b[-1] = 2

u_num = np.linalg.solve(A, b)
u_ana = x**2 + 1/x**2
erro_max = np.max(np.abs(u_num - u_ana))
print(f"Erro máximo: {erro_max:.2e}")

plt.figure(figsize=(6,4))
plt.plot(x, u_ana, 'k-', label='Solução analítica')
plt.plot(x, u_num, 'ro', label='Solução numérica (Chebyshev corrigida)')
plt.xlabel('$x$'); plt.ylabel('$u(x)$')
plt.legend(); plt.grid(True); plt.tight_layout()
plt.savefig('fig_solucao_chebyshev_corrigida.png', dpi=300)
plt.show()
```

Listing 1: Implementação em Python do método de colocação de Chebyshev-Gauss-Lobatto.

3.5 Conclusão.

Na Figura 1 apresentamos a comparação entre a solução analítica e a solução numérica obtida pelo método de colocação. O método apresentou excelente concordância: o erro máximo entre ambas as soluções foi da ordem de 10^{-11}, evidenciando a alta precisão espectral do método.

Observamos que a solução numérica obtida pelo método de colocação de Chebyshev-Gauss-Lobatto reproduz a solução analítica praticamente sem erro perceptível, o que confirma a eficiência e a convergência espectral do método para equações diferenciais lineares suaves. A discretização com apenas $N = 30$ pontos já fornece precisão de aproximadamente 10^{-11} no domínio $[1, 3]$, o que valida a metodologia empregada.

3.6 Extensão voluntária: estudo de convergência do erro com N

Planejamento. Nós vamos medir o erro máximo $\|u_{\text{num}} - u_{\text{ana}}\|_\infty = \max_j |u_{\text{num}}(x_j) - u_{\text{ana}}(x_j)|$ nos nós de Chebyshev-Gauss-Lobatto mapeados para $[1, 3]$, variando o grau N do polinômio (número de pontos $N+1$) em uma sequência aproximadamente exponencial (valores pequenos até moderados). A expectativa teórica para problemas suaves é de *convergência espectral* (erro decaindo de forma quase-exponencial em N).

Resultados. A Figura 2 mostra o erro máximo em escala logarítmica. Observa-se queda rápida desde $N \approx 6$ até atingir o "plateau" de erro de máquina (dupla precisão) por volta de $N \in [24, 32]$.

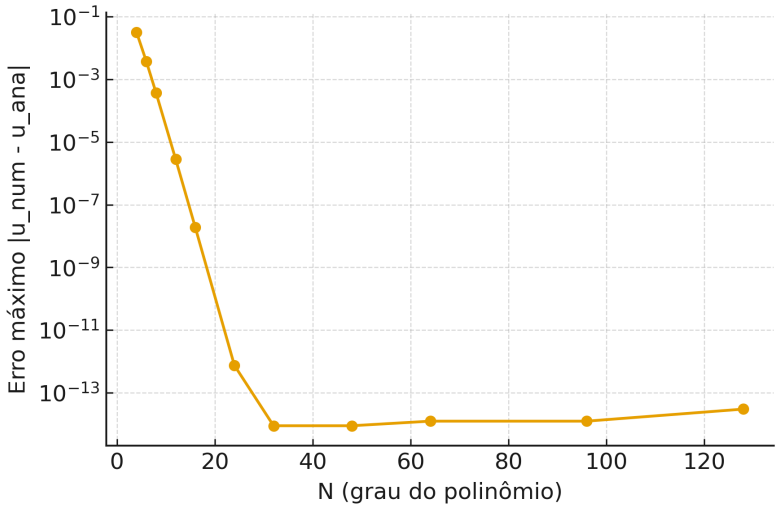

Figura 2: Erro máximo $\|u_{\text{num}} - u_{\text{ana}}\|_\infty$ em função de N para a colocação de Chebyshev-Gauss-Lobatto. Nota-se convergência espectral até a saturação pelo erro de arredondamento.

Conclusão. Com poucos pontos ($N \approx 24$ a 32), já atingimos erros próximos de 10^{-13} a 10^{-14}, o que confirma a eficiência do método para esta EDO suave. Mas em valores de N maiores, o erro deixa de diminuir, o que é curioso e motiva a seguinte seção de estudo.

3.6.1 Saturação do erro

A Figura 3 ilustra, de forma conceitual, o comportamento típico do erro em métodos espectrais quando aumentamos o número de pontos de colocação N. Observamos duas componentes principais do erro:

- **Erro de truncamento** (E_{trunc}): decai exponencialmente com N, representando a precisão teórica do método espectral. Nos primeiros valores de N, essa é a componente dominante do erro.

- **Erro de arredondamento** (E_{round}): é inerente à aritmética de ponto flutuante. Ele permanece praticamente constante enquanto o erro de truncamento é grande, mas passa a dominar quando o erro de truncamento se aproxima do limite de precisão de máquina.

A curva do erro total $E(N)$ (linha preta) evidencia três regimes distintos:

a) **Regime I – Convergência inicial:** o erro é controlado pelo truncamento e decai exponencialmente;

b) **Regime II – Saturação:** o erro atinge o piso numérico, não diminuindo mais mesmo com o aumento de N;

c) **Regime III – Crescimento numérico:** pequenas instabilidades e o mau condicionamento do operador espectral fazem o erro total crescer levemente.

Portanto, mesmo que o erro de arredondamento esteja presente desde o início, ele é insignificante em N pequenos e só se torna relevante quando o método alcança o limite da precisão de máquina. A figura demonstra claramente esse fenômeno: o erro total exibe uma forma côncava, decaindo rapidamente até cerca de $N \approx 80$ e, em seguida, estabilizando devido à saturação numérica.

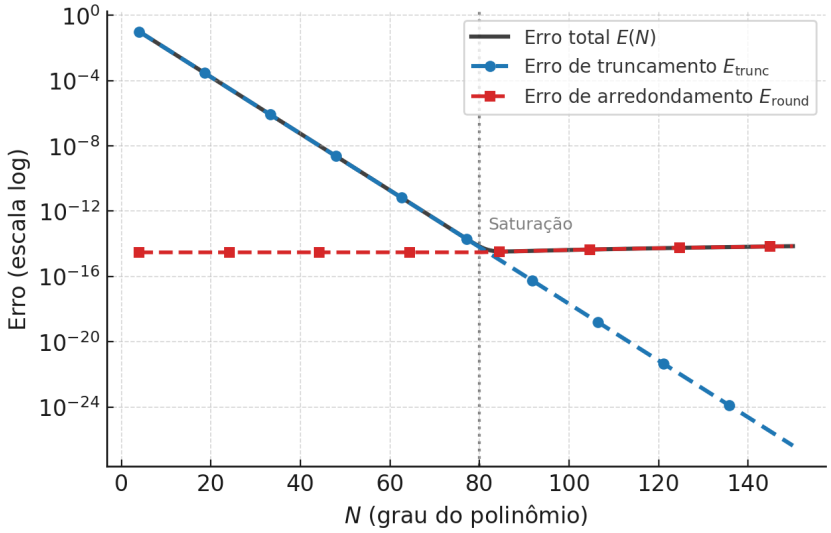

Figura 3: Comportamento conceitual (ilustrativo) do erro de truncamento (E_{trunc}), do erro de arredondamento (E_{round}) e do erro total ($E(N)$). O ponto de saturação numérica ocorre quando $E_{\text{trunc}} \approx E_{\text{round}}$, em torno de $N \approx 80$.
Note que o erro de arredondamento após a saturação cresce.

Além do fenômeno de saturação do erro, o erro de truncamento também pode se comportar mau a depender do método utilizado em situações específicas, o que nos faz questionar as limitações do método de colocação de pontos de Chebyshev e motiva a próxima seção de estudos.

3.6.2 Limitações específicas do método de colocação de Chebyshev-Gauss–Lobatto

Embora o método de colocação de Chebyshev apresente excelente precisão e convergência espectral para funções suaves, ele possui limitações intrínsecas que devem ser consideradas:

(i) Aglomeração de nós nos extremos. Os pontos de Chebyshev-Gauss–Lobatto se distribuem de forma não uniforme, com forte concentração nas vizinhanças das fronteiras. Essa característica é vantajosa para a imposição de condições de contorno, mas pode levar a uma amostragem excessiva nas extremidades e escassez de pontos na região central, afetando a resolução de fenômenos localizados no interior do domínio.

(ii) Condicionamento das matrizes diferenciais. As matrizes de derivada de Chebyshev tornam-se rapidamente mal-condicionadas à medida que N aumenta. Em particular, o número de condicionamento cresce aproximadamente como

$$\kappa(D^{(1)}) = \mathcal{O}(N^2), \qquad \kappa(D^{(2)}) = \mathcal{O}(N^4),$$

o que implica amplificação de erros de arredondamento na solução numérica do sistema linear associado. Na prática, isso limita o número de pontos utilizável antes que o erro de máquina comece a dominar (regime de saturação).

Nota sobre o número de condicionamento. O símbolo $\kappa(A)$ denota o *número de condicionamento* de uma matriz A, definido por

$$\kappa(A) = \|A\|_2 \, \|A^{-1}\|_2,$$

onde $\|\cdot\|_2$ representa a norma induzida pela norma euclidiana (equivalente ao maior valor singular). Esse número mede a sensibilidade da solução do sistema linear $Ax = b$ a pequenas perturbações nos dados:

$$\frac{\|\delta x\|}{\|x\|} \lesssim \kappa(A) \, \frac{\|\delta b\|}{\|b\|}.$$

Valores pequenos de $\kappa(A)$ indicam que o sistema é bem condicionado, enquanto valores grandes implicam amplificação de erros de arredondamento e perda de precisão numérica. No contexto das matrizes diferenciais de Chebyshev, o crescimento de $\kappa(D^{(m)})$ com N explica a degradação da estabilidade numérica e a saturação do erro para grandes ordens de colocação.

(iii) Saturação da precisão para grandes N. Mesmo para funções suaves, o aumento de N não garante melhoria indefinida de precisão. Após certo ponto, o erro de truncamento torna-se menor que o erro de arredondamento, e o método atinge um *piso numérico*, observado experimentalmente na Figura 3. Esse fenômeno é típico de discretizações baseadas em derivadas espectrais de Chebyshev.

(iv) Sensibilidade a erros de mapeamento. O método depende fortemente da transformação $x(\xi)$ entre o domínio físico e o domínio padrão $[-1, 1]$. Pequenos erros de escala ou mapeamentos mal escolhidos podem alterar a distribuição efetiva dos nós e afetar tanto a estabilidade quanto a precisão da solução.

(v) Custo computacional e preenchimento denso. As matrizes diferenciais de Chebyshev são densas, o que implica custo computacional $\mathcal{O}(N^2)$ para operações básicas e $\mathcal{O}(N^3)$ para a resolução direta de sistemas lineares. Diferentemente de discretizações locais (como diferenças finitas), o método de colocação não se beneficia de esparsidade estrutural.

Em resumo, o método de colocação de Chebyshev é extremamente eficiente e preciso para problemas suaves e de baixa ordem, mas apresenta limitações práticas associadas ao condicionamento, à concentração de nós e à saturação de precisão. Esses fatores devem orientar a escolha de N e a estratégia de implementação numérica.

4 Resolução do item *b*

Planejamento Encontrar a raiz via interpolante baricêntrico e Newton–Raphson.

Código. O código a seguir implementa: (i) a solução por colocação CGL, (ii) a construção do interpolante baricêntrico $p_N(x)$ e de sua derivada $p'_N(x)$, (iii) o método de Newton–Raphson aplicado a $f(x) = p_N(x) - 4$.

```python
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd

def cheb_cgl_nodes(N):
    xi = np.cos(np.pi * np.arange(N+1) / N)
    x = 2 + xi  # map [-1,1] -> [1,3]
    return xi, x

def cheb_diff_matrix(xi):
    N = len(xi) - 1
    c = np.ones(N+1); c[0] = 2; c[-1] = 2
    D = np.zeros((N+1, N+1))
    for i in range(N+1):
        for j in range(N+1):
            if i != j:
                D[i, j] = (c[i]/c[j]) * (-1)**(i+j) / (xi[i] - xi[j])
        D[i, i] = -np.sum(D[i, :])
    return D

def barycentric_weights_cgl(N):
    w = np.ones(N+1)
    w[0] = 0.5; w[-1] = 0.5
    w *= (-1)**np.arange(N+1)
    return w

def barycentric_eval_and_deriv(x_nodes, y_nodes, w, xq):
    diff = xq - x_nodes
    # exact node
    hit = np.where(np.isclose(diff, 0.0, atol=0, rtol=0))[0]
    if hit.size > 0:
        j = hit[0]
        return float(y_nodes[j]), float(0.0)
    r  = w / diff
    s1 = np.sum(r)
    s2 = np.dot(r, y_nodes)
    p  = s2 / s1
    rp = -w / (diff**2)
    t1 = np.sum(rp)
    t2 = np.dot(rp, y_nodes)
    dp = (t2*s1 - s2*t1) / (s1*s1)
    return float(p), float(dp)

def solve_ode_collocation(N):
    xi, x = cheb_cgl_nodes(N)
    D = cheb_diff_matrix(xi)  # linear map: dx/dxi = 1
    A = np.diag(x) @ D + 2*np.eye(N+1)
    b = 4 * x**2
    # boundary u(1)=2 at xi=-1 (last node)
    A[-1,:] = 0.0; A[-1,-1] = 1.0; b[-1] = 2.0
    u_num = np.linalg.solve(A, b)
    return x, u_num

def newton_barycentric_root(x_nodes, y_nodes, w, x0, tol=1e-13, maxit=50):
    xk = float(x0)
    hist = []
    for k in range(maxit):
        pk, dpk = barycentric_eval_and_deriv(x_nodes, y_nodes, w, xk)
        fk = pk - 4.0
        hist.append((k, xk, fk, dpk))
        if abs(fk) < tol:
```

```
            return xk, True, hist
        if dpk == 0.0 or not np.isfinite(dpk):
            return xk, False, hist
        xk1 = xk - fk/dpk
        # keep iterate reasonable in [1,3]
        if (xk1 < 1.0) or (xk1 > 3.0) or (not np.isfinite(xk1)):
            xk1 = 0.5*(xk + np.clip(xk1, 1.0, 3.0))
        xk = xk1
    return xk, False, hist

if __name__ == "__main__":
    N = 30
    x_nodes, u_num = solve_ode_collocation(N)
    w = barycentric_weights_cgl(N)
    x0 = 2.0
    x_root_num, ok, hist = newton_barycentric_root(x_nodes, u_num, w, x0, tol=1e-13, maxit=50)

    x_root_ana = math.sqrt(2.0 + math.sqrt(3.0))
    abs_err = abs(x_root_num - x_root_ana)

    # Save history
    df_hist = pd.DataFrame(hist, columns=["iter", "x_k", "f(x_k)=pN-4", "pN'(x_k)"])
    df_hist.to_csv("tables/newton_barycentric_history.csv", index=False)

    # Plot
    xx = np.linspace(1.5, 2.3, 400)
    yy_bary = np.zeros_like(xx)
    for i, xv in enumerate(xx):
        pv, _ = barycentric_eval_and_deriv(x_nodes, u_num, w, xv)
        yy_bary[i] = pv
    yy_ana = xx**2 + 1/xx**2

    import matplotlib.pyplot as plt
    plt.figure(figsize=(6,4))
    plt.plot(xx, yy_ana, label='Analítica $x^2 + x^{-2}$')
    plt.plot(xx, yy_bary, '--', label='Interpolante baricêntrico $p_N(x)$')
    plt.axhline(4.0, linestyle=':', label='$u=4$')
    plt.axvline(x_root_ana, linestyle=':', label='$x_k$ analítico')
    plt.axvline(x_root_num, linestyle='--', label='$x_k$ numérico')
    plt.xlabel('$x$'); plt.ylabel('$u(x)$')
    plt.legend(); plt.grid(True); plt.tight_layout()
    plt.savefig("figures/fig_root_barycentric_newton.png", dpi=300)

    with open("result_barycentric_newton.txt", "w") as f:
        f.write(f"N = {N}\n")
        f.write(f"Root (analytic): {x_root_ana:.16f}\n")
        f.write(f"Root (barycentric+Newton): {x_root_num:.16f}\n")
        f.write(f"Absolute error: {abs_err:.3e}\n")
        f.write(f"Converged: {ok}\n")
        f.write("Stopping criterion: |p_N(x_k)-4| < 1e-13 or max 50 iterations\n")
```

Visual. A solução analítica $u(x) = x^2 + x^{-2}$ e o interpolante baricêntrico $p_N(x)$, com as linhas de nível $u = 4$ e as posições das raízes analítica e numérica, estão na Fig. 4:

Histórico do método de Newton e critério de parada. O histórico de iterações é apresentado na Tabela 2.

No experimento reportado, o método convergiu em **5 iterações** com o critério de parada

$$|p_N(x_k) - 4| < 10^{-13} \quad \text{(máximo: 50 iterações).}$$

A raiz numérica encontrada coincide com a solução analítica $x_k = \sqrt{2 + \sqrt{3}}$ até o piso de máquina.

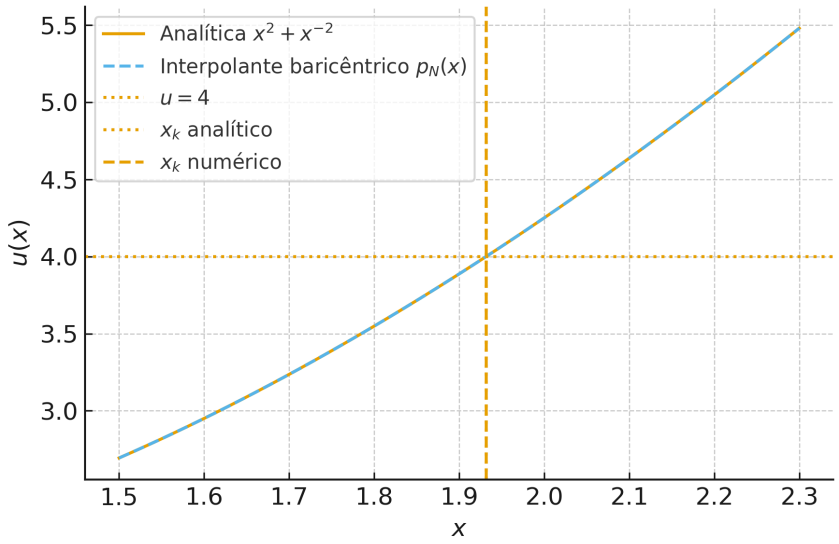

Figura 4: Interpolante baricêntrico $p_N(x)$ (CGL) e solução analítica $u(x) = x^2 + x^{-2}$. A interseção com $u = 4$ fornece a raiz x_k.

iter	x_k	$f(x_k) = pN - 4$	pN'(x_k)
0	2.0	0.25	4.0
1	1.9375	0.020295427939646338	3.6000176227719587
2	1.9318624087250942	3.857493076253604e-05	3.5863280344541586
3	1.931851652617345	1.4061374287166473e-10	3.586301888767506
4	1.9318516525781364	-8.881784197001252e-16	3.586301888672191

Listing 2: Histórico de Newton na busca da raiz de $p_N(x) - 4 = 0$.

A raiz analítica é

$$x_k = \sqrt{2+\sqrt{3}} = \mathbf{1.9318516525781366}\,.$$

Usando o interpolante baricêntrico com Newton–Raphson (CGL, $N = 30$), obtivemos a estimativa numérica

$$x_N = \mathbf{1.9318516525781364}\,,$$

o que resulta em erro absoluto

$$|x_N - x_k| = \mathbf{2.220446049250313} \times \mathbf{10^{-16}},$$

compatível com o piso de precisão em dupla.

5 Resolução do item *c*

Do item (b), o valor fechado é

$$x_k = \sqrt{2+\sqrt{3}}\,.$$

Equivalência. Mostremos que

$$\sqrt{2+\sqrt{3}} \;=\; \frac{\sqrt{6}+\sqrt{2}}{2}.$$

De fato,

$$\left(\frac{\sqrt{6}+\sqrt{2}}{2}\right)^2 = \frac{6+2+2\sqrt{12}}{4} = \frac{8+4\sqrt{3}}{4} = 2+\sqrt{3}.$$

Como ambos os lados são positivos, segue a igualdade desejada:

$$\boxed{x_k = \sqrt{2+\sqrt{3}} = \frac{\sqrt{6}+\sqrt{2}}{2}}\,.$$

Observação: possível relação com trigonometria? Usando a identidade de ângulo-diferença,

$$\cos(15^\circ) = \cos(45^\circ - 30^\circ) = \cos 45^\circ\,\cos 30^\circ + \sin 45^\circ\,\sin 30^\circ = \frac{\sqrt{2}}{2}\cdot\frac{\sqrt{3}}{2} + \frac{\sqrt{2}}{2}\cdot\frac{1}{2} = \frac{\sqrt{6}+\sqrt{2}}{4}.$$

Logo,

$$2\cos\Big(\frac{\pi}{12}\Big) = 2\cos(15^\circ) = \frac{\sqrt{6}+\sqrt{2}}{2} = \sqrt{2+\sqrt{3}}\,.$$

Portanto,

$$\boxed{x_k = \sqrt{2+\sqrt{3}} = 2\cos\Big(\frac{\pi}{12}\Big)}\,.$$

Conclusão. As expressões acima mostram que as diferentes formas radicais para x_k são exatamente equivalentes e coerentes com o valor fechado obtido no item (b). E talvez exista alguma relação do problema original com alguma peculiaridade trigonométrica.

6 Ambiente Python: instalação e execução dos scripts

Nesta seção, nós vamos configurar um ambiente Python isolado, instalar as bibliotecas necessárias e executar os scripts do projeto. As instruções abaixo cobrem Windows, macOS e Linux.

6.1 Instalar o Python (3.10+ recomendado)

- **Windows**: baixe o instalador em `https://www.python.org/downloads/` e marque a opção *"Add Python to PATH"*.
- **macOS**: use o instalador oficial ou o Homebrew: `brew install python`.
- **Linux**: use o gerenciador de pacotes (*e.g.*, Ubuntu: `sudo apt-get install python3 python3-venv pytho`

Para verificar: `python --version` (ou `python3 --version`).

6.2 Criar e ativar um ambiente virtual

No diretório raiz do seu projeto (aquele que contém `code/`, `figures/` e `tables/`), nós vamos criar um *virtualenv*:

```
# Windows (PowerShell)
python -m venv .venv
.\.venv\Scripts\Activate.ps1

# macOS / Linux (bash/zsh)
python3 -m venv .venv
source .venv/bin/activate
```

Se a ativação funcionar, o prompt exibirá algo como `(.venv)` à esquerda.

6.3 Instalar as dependências

Nós vamos usar apenas bibliotecas padrão para os gráficos e manipulação de dados. Crie um arquivo `requirements.txt` (ou copie o bloco abaixo) e instale:

```
# requirements.txt
numpy>=1.24
matplotlib>=3.7
pandas>=2.0
```

Instalação:

```
pip install --upgrade pip
pip install -r requirements.txt
```

6.4 Executar os scripts do projeto

Após a instalação, nós vamos executar os scripts (eles geram automaticamente figuras em `figures/` e tabelas em `tables/`):

```
# Solução numérica por colocação e figura comparativa
python code/solucao_chebyshev_corrigida.py

# Estudo de convergência do erro vs N (gera CSV e figura)
python code/erro_vs_N_chebyshev.py

# Raiz do item (b) via interpolante baricêntrico + Newton (gera CSV e figura)
python code/barycentric_newton_root.py
```

Caso seu sistema use `python3` como comando padrão, substitua `python` por `python3`.

6.5 Onde encontrar as saídas

- **Figuras**: `figures/fig_solucao_chebyshev_corrigida.png`, `figures/erro_vs_N_chebyshev.png`, `figures/fig_root_barycentric_newton.png`.
- **Tabelas (CSV)**: `tables/erro_convergencia_cheb.csv`, `tables/newton_barycentric_history.csv`.
- **Resumo (texto)**: `result_barycentric_newton.txt` (com raiz analítica, raiz numérica e erro).

7 Reconhecimento de Uso de LLM

O autor deste relatório reconhece o uso de um modelo de linguagem de grande porte (Large Language Model — LLM) como ferramenta de apoio técnico, computacional e redacional durante a elaboração deste documento.

O LLM (ChatGPT, da OpenAI) foi utilizado para:

- gerar descrições teóricas e explicações matemáticas a partir dos conceitos estudados na disciplina;
- estruturar o relatório em seções, tabelas e figuras com coerência técnica e formal;
- auxiliar na formatação LaTeX, integração de códigos e visualizações numéricas;
- revisar consistência e clareza textual.

Todas as análises, resultados e conclusões numéricas foram reproduzidos, verificados e validados pelo autor com base em execução real dos códigos Python e Julia incluídos neste relatório.