P2 MS211 - Vinícius de Oliveira Peixoto Rodrigues (245294)

Abstract

Todos os programas usados para calcular aproximações numéricas foram implementados em Python, por mim, usando os métodos estudados em sala. As bibliotecas utilizadas (Numpy e Scipy) fornecem estruturas de dados otimizadas para cálculos de Álgebra Linear, além de algumas ferramentas convenientes (multiplicação de matrizes, cálculo de inversa, norma, etc), mas a implementação dos algoritmos em si foi feita por mim **(exceto pela quadratura de Gauss-Lagrange)**. O código fonte pode ser encontrado no meu repositório do Github: https://github.com/nukelets/metodos_numericos/tree/main/p2.

1 Problemas quantitativos

Questão 1: O vetor abaixo contém os últimos 66 valores para as mortes diárias em todo o mundo nos últimos 66 dias. Faça uma previsão do número de mortes diárias no dia 80. Justifique sua escolha da função a ser ajustada. Critique avaliativamente seu resultado: por que você tende a acreditar nele?

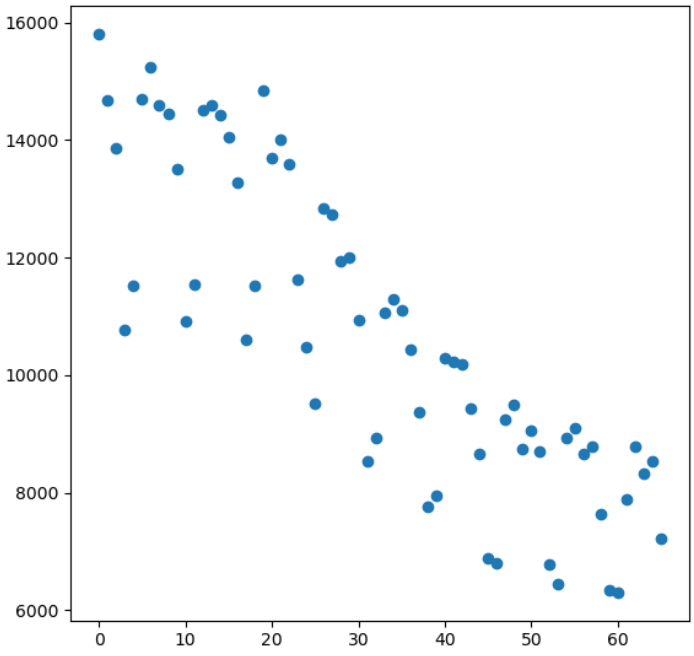

Figure 1. Scatter plot dos valores para o número de mortes diárias por covid nos últimos 66 dias

Ao se observar o gráfico, percebe-se que a forma geral é claramente a de uma curva descrescente (felizmente!). Também é possível inferir visualmente que a curva tem concavidade para cima, sugerindo que um ajuste linear pode não ser a melhor opção. Com isso em mente, fiz testes de

ajuste pelo método dos mínimos quadrados para três opções de função:

i) $y = c_1 + c_2\,x$, com $\varphi_1 = 1,\ \varphi_2 = x$

ii) $y = c_1 + c_2\,x + c_3\,x^2$, com $\varphi_1 = 1,\ \varphi_2 = x,\ \varphi_3 = x^2$

iii) $y = a\,e^{bx} \Rightarrow \ln y = \ln a + bx \equiv \tilde{y} = c_1 + c_2\,x$, com $\varphi_1 = 1,\ \varphi_2 = x$

Para se encontrar os coeficientes c_k, $i = 1, 2, \ldots, n$ pelo MMQ a partir de N pontos $(x_1, y_1), \ldots, (x_N, y_N)$, é necessário resolver o sistema

$$\mathrm{AC} = B$$

$$(A)_{\mathrm{ij}} = \sum_{k=1}^{N} \varphi_i(x_k)\varphi_j(x_k)$$

$$C = \begin{pmatrix} c_1 \\ \ldots. \\ c_n \end{pmatrix}$$

$$B = \begin{pmatrix} \sum_{k=1}^{N} y_k \varphi_1(x_k) \\ \ldots. \\ \sum_{k=1}^{N} y_k \varphi_n(x_k) \end{pmatrix}$$

De modo que os coeficientes são dados por $C = A^{-1}B$.

Além disso, para avaliar o ajuste para cada função escolhida, calculei a raíz do desvio quadrático médio entre o modelo do MMQ e os dados experimentais:

$$\mathrm{erro} = \sqrt{\frac{1}{N}\sum_{i=0}^{N} (y_i - \hat{y}(x_i))^2}$$

m

onde $\hat{y}(x)$ é o modelo obtido pelo MMQ. A minha implementação em Python desse método se encontra na minha pasta do Github dessa prova (`fit/least_squares.py`).

Os resultados obtidos foram:

i) $y(x) = 14526.12 - 117.44x$, com erro de 1371.34 e $y(80) = 5130.92$

ii) $y(x) = 14758.67 - 139.24x + 0.34x^2$, com erro de 1367.01 e $y(80) = 5795.47$

iii) $y(x) = 14915.20\,e^{-0.01x}$, com erro de 1377.47 e $y(80) = 6701.74$

Os resultados se encontram plotados abaixo:

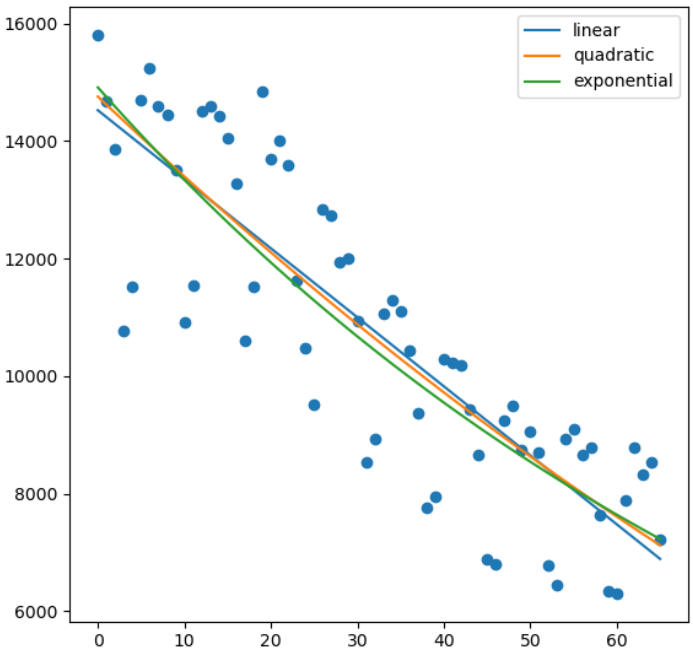

Figure 2. Modelos para a evolução do número de mortes diárias por covid.

Percebe-se que dentro do intervalo dos dados, os três modelos não diferem muito (até os erros são muito próximos). Conforme o tempo passa, contudo, as diferenças nas taxas de decrescimento nos diferentes modelos torna-se expressiva (como se pode observar pela diferença de quase 1500 mortes entre a previsão do modelo linear e do modelo exponencial). Desse modo, decidi escolher a previsão do modelo **exponencial** visto que 1) é uma previsão conservadora e 2) a função exponencial parece (visualmente) se ajustar melhor ao padrão côncavo dos dados experimentais (ao contrário do modelo quadrático e linear, que no intervalo dos dados experimentais são praticamente linhas retas). A resposta final para a previsão do número de mortes no dia 80 é, portanto, **6701 mortes**.

Questão 2. Para se obter o valor numérico da integral $\int_0^{\frac{\pi}{2}} |\sin(x)\cos(x)|^{\frac{3}{2}}\,\mathrm{dx}$, qual seria o método mais adequado para obter tal resultado com uma ótima precisão? Observe, por favor, que o termo "ótima precisão" é bastante subjetivo. É isso mesmo. Justifique sua escolha de fórmula, use algum software para calcular essa integral e avalie o resultado obtido. Qual método seria melhor se não fosse necessária uma sensível precisão e sim rapidez de cálculo – e por quê?

Primeiramente, é importante observar que no intervalo $\left[0, \frac{\pi}{2}\right]$ a função $|\sin(x)\cos(x)|^{\frac{3}{2}}$ é contínua. Para se obter o resultado da integral com uma ótima precisão, eu acredito que o melhor método seja a quadratura de Gauss-Legendre (que pode ser usada porque a função é contínua no intervalo, inclusive nos extremos). O método consiste em fazer a seguinte aproximação:

$$\int_a^b f(x)\,\mathrm{dx} = \sum_{i=0}^{n} w_i f(x_i)$$

escolhendo os "pesos" w_i de forma inteligente – mais precisamente, de modo que para qualquer $f(x)$ = polinômio de grau no máximo $2n+1$, a soma seja exatamente igual à integral. Um teorema devido a Gauss garante que, para todo polinômio $q(x)$ com raízes $x_1, \ldots, x_n$ tal que para $0 \leq k \leq n$

$$\int_a^b x^k q(x)\,\mathrm{dx} = 0$$

vale que a aproximação

$$\int_a^b f(x)\mathrm{dx} \approx \sum_{i=0}^{n} w_i f(x_i)$$

$$w_i(x) = \int_a^b \left(\prod_{j \neq i}^{n} \frac{x - x_i}{x_i - x_j} \right) \mathrm{dx}$$

é exata. A prova é complicada, mas existe um conjunto de polinômios (os polinômios de Legendre) que formam uma base ortogonal para o espaço vetorial dos polinômios e que convenientemente atendem à propriedade do teorema da quadratura de Gauss. Isso significa que é possível subdividir em n partes o intervalo $[a, b]$ (na verdade esse intervalo mapeado para $[-1, 1]$) de modo que as "marcações" de divisão correspondam às raízes do polinômio de Legendre P_n equivalente. Também é possível provar que os pesos w_i são tais que

$$w_i(x) = \frac{2}{(1 - x_i^2) P_n'(x_i)^2}$$

Computacionalmente, isso significa que é possível guardar os valores de x_i e w_i utilizando espaço da ordem de $n \times n$ e computar a soma de n termos $\sum w_i f(x_i)$ como valor exato da integral de polinômios de grau até $2n+1$ (!!)

A aplicação imediata é o fato de que se a função $f(x)$ é bem representada por polinômios (série de Taylor) no intervalo de integração, então é possível aproximar os termos da expansão da integral em série de Taylor com alta precisão.

Além disso, eu também acredito que esse método seja a melhor escolha mesmo que haja preocupação com a rapidez em vez da precisão: não vou conseguir apresentar uma prova concreta aqui, mas cito este artigo onde são feitas "benchmarks" do método da quadratura de Gauss vs. método de Romberg; em todas as aplicações, o método da quadratura de Gauss i) é mais preciso, dado

um número fixo de divisões do intervalo de integração e ii) é significativamente mais rápido, dada uma tolerância de erro fixa. Nas minhas benchmarks simplórias (cálculo da integral do problema realizado 1000 vezes para cada método), o método de Gauss foi 7 vezes mais rápido que o de Romberg.

Finalmente, deixo aqui o resultado da integração numérica por meio da quadratura de Gauss (feita usando o pacote `scipy.integrate`):

```
import scipy.integrate as integrate
import numpy as np
import timeit

def f(x):
    return abs(np.sin(x) * np.cos(x)) ** 1.5

def gauss_quad(f):
    return integrate.quadrature(f, 0, np.pi*0.5)

def romberg(f):
    return integrate.romberg(f, 0, np.pi*0.5)

if __name__ == "__main__":
    print(
        timeit.timeit('gauss_quad(f)',

setup="import␣scipy.integrate␣as␣integrate;␣from␣__main__␣import␣gauss_quad",
        globals=locals(), number=1000))

    print(
        timeit.timeit('romberg(f)',

setup="import␣scipy.integrate␣as␣integrate;␣from␣__main__␣import␣romberg",
        globals=locals(), number=1000))
```

Resultado: 0.3090123

Questão 3. Considere a edo dada por $\frac{dp}{dt} = 0.025\,p(t)\left(1 - \frac{p(t)}{2000 + e^{-t/12}}\right)$ sabendo que $p(0) = 121$. Deseja-se obter a solução no intervalo [0, 75]. Desenvolva esta aproximação numérica, justificando suas escolhas. Se der, forneça um gráfico da aproximação obtida.

O problema de valor inicial é da forma

$$y' = f(t, y), y(t_0) = y_0$$

O método que escolhi utilizar para resolver é o Runge-Kutta de 4a ordem. Ele consiste em fazer a aproximação

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \\ t_{n+1} &= t_n + h \\ k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \\ k_3 &= f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right) \\ h_4 &= f(t_n + h, y_n + hk_3) \end{aligned}$$

Foi escolhido esse método em específico devido à sua ótima precisão, tendo erro acumulado (global) de ordem $O(h^4)$. O gráfico da solução no intervalo $[0, 75]$ se encontra na figura abaixo:

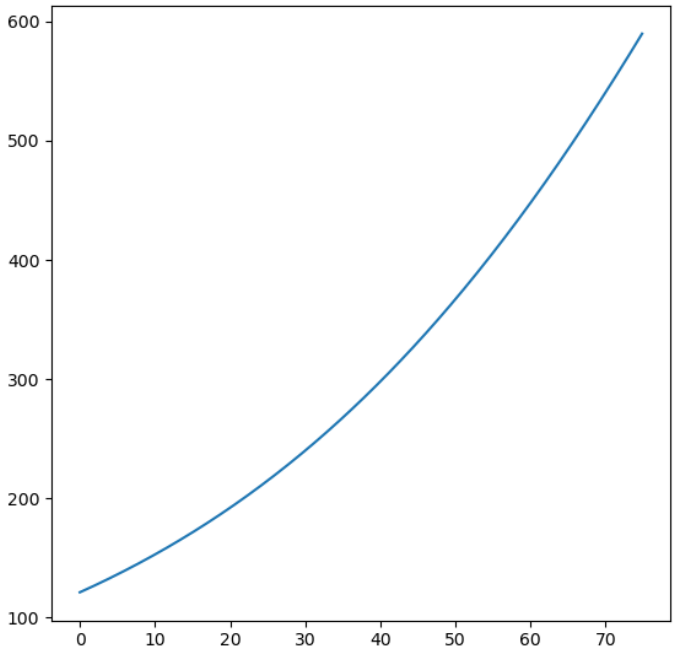

Figure 3. Gráfico da solução para $p(x)$ no intervalo $[0, 75]$

Questão 6. Calcule a solução da equação diferencial de segunda ordem (na verdade, um PVC) dada por $y''(t)+2.12y'(t)+0.078y(t)=f(t)$ para $t\in[0,5]$ e sendo $y(0)=2$ e $y(5)=2$ e sendo $f(t)=\begin{cases}0, \text{se } t\in[0,1)\\ 2, \text{se } t\in[1,3.5]\\ 0, \text{se } t\in(3.5,5]\end{cases}$.

Primeiramente, é preciso encontrar uma aproximação por diferenças finitas para as derivadas de $y(t)$. Da série de Taylor:

$$y(t_0+\Delta t)=y(t_0)+y'(t_0)\Delta t+\frac{1}{2}y''(t_0)\Delta t^2+O(\Delta t^3)$$

Aproximando em primeira ordem:

$$y'(t_0)\approx\frac{y(t_0+\Delta t)-y(t_0)}{\Delta t}$$

Definindo as coisas em termos de diferenças finitas, fazendo $t_{n+1}=t_n+h$:

$$y'_n=\frac{y_{n+1}-y_n}{h}$$

É conveniente reescrever as relações de diferença finita em termos da diferença central. Observando que ao ir de y_{n+1} até y_n, $\Delta t=t_n-(t_n+h)=-h$:

$$y_{n+1}=y_n+hy'_n+\frac{1}{2}h^2y''_n+O(h^3)$$

$$y_{n-1}=y_n-hy'_n+\frac{1}{2}h^2y''_n+O(h^3)$$

Subtraindo as duas equações:

$$y_{n+1}-y_{n-1}=2hy'_n\Rightarrow y'_n=\frac{y_{n+1}-y_n}{2h}$$

Somando as duas equações:

$$y_{n+1}+y_{n-1}=2y_n+h^2y''_n\Rightarrow y''_n=\frac{y_{n-1}-2y_n+y_{n+1}}{h^2}$$

Agora, para o problema de contorno

$$ay''+by'+cy=f(t),\quad y(t_0)=y_0,\ y(t_f)=y_{N-1}$$

usa-se a discretização

$$a\left(\frac{y_{n-1}-2y_n+y_{n+1}}{h^2}\right)+b\left(\frac{y_{n+1}-y_{n-1}}{2h}\right)+cy_n=f(t_n)$$

Rearranjando os termos:

$$\left(a-\frac{1}{2}bh\right)y_{n-1}+(-2a+ch^2)y_n+\left(a+\frac{1}{2}bh\right)y_{n+1}=h^2f(t_n)$$

Isso gera um sistema de equações com $N-2$ incógnitas; como y_0 e y_{N-1} (valores de contorno) são conhecidos, há $N-2$ equações para $N-2$ variáveis, e é possível resolver o sistema. Em forma matricial:

$$\begin{pmatrix} \beta & \gamma & 0 & 0 & 0 & \dots & 0 \\ \alpha & \beta & \gamma & 0 & 0 & \dots & 0 \\ 0 & \alpha & \beta & \gamma & 0 & \dots & 0 \\ 0 & 0 & \alpha & \beta & \gamma & \dots & 0 \\ & & & \dots & & & \\ 0 & 0 & 0 & 0 & 0 & \dots & \beta \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \dots \\ y_{N-2} \end{pmatrix} = \begin{pmatrix} h^2 f(t_1) - \alpha y_0 \\ h^2 f(t_2) \\ h^2 f(t_3) \\ h^2 f(t_4) \\ \dots \\ h^2 f(t_{N-2}) - \gamma y_{N-1} \end{pmatrix}$$

$$\alpha = a - \frac{1}{2}bh$$

$$\beta = -2a + ch^2$$

$$\gamma = a + \frac{1}{2}bh$$

Reescrevendo o sistema acima na forma $\boldsymbol{Ay = B}$, basta fazer $\boldsymbol{y = A^{-1}B}$ para encontrar os pontos $y_1, \dots, y_n$.

Utilizando-se o método acima para o PVC da questão, implementei uma função em Python `finite_difference(a, b, c, f, y_start, y_end, start, end, n)` que resolve PVCs gerais de segunda ordem (o código está no meu repositório do Github). Fazendo o número de intervalos $n = 1000$ (ou, equivalentemente, $h = 5/1000 = 0.005$) obtém-se o seguinte gráfico:

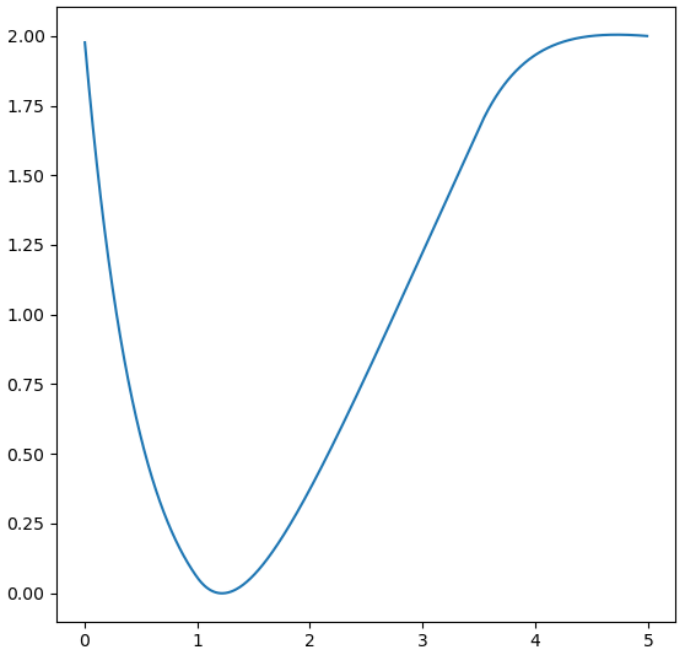

Figure 4. Gráfico da solução do PVC a partir do método das diferenças finitas.

Questão 7. Para calcular y(0.63), use a tabela com valores de x e de y dada abaixo, fornecendo uma estimativa de erro, justificando suas decisões.

Primeiramente, se encontra abaixo um scatter plot dos dados na tabela:

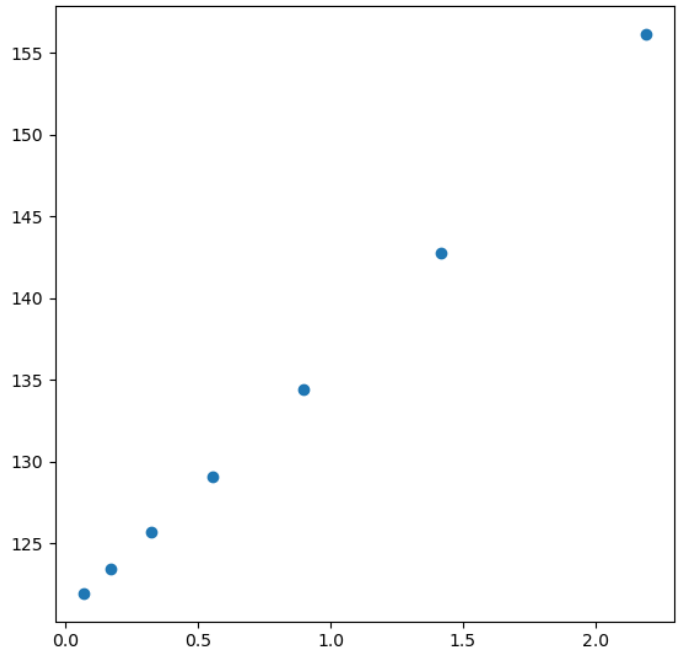

Figure 5.

Parece claro visualmente que se tratam de pontos dispostos linearmente (pelo menos aproximadamente). Usando o mesmo método (mínimos quadrados) e o mesmo código da questão 1, ao se fazer o ajuste para $y = a + bx$, obtém-se

$$y = 120.47 + 16.05x$$

$$\text{erro} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}(x_i))^2} = 0.371$$

Tomando-se a raíz do desvio quadrático médio como a estimativa de erro, chega-se no resultado

$$y = (120.47 + 16.05x) \pm 0.37$$

De modo que a previsão é $\boldsymbol{y(0.63) = 130.58}$. Abaixo se encontra um gráfico com a reta ajustada:

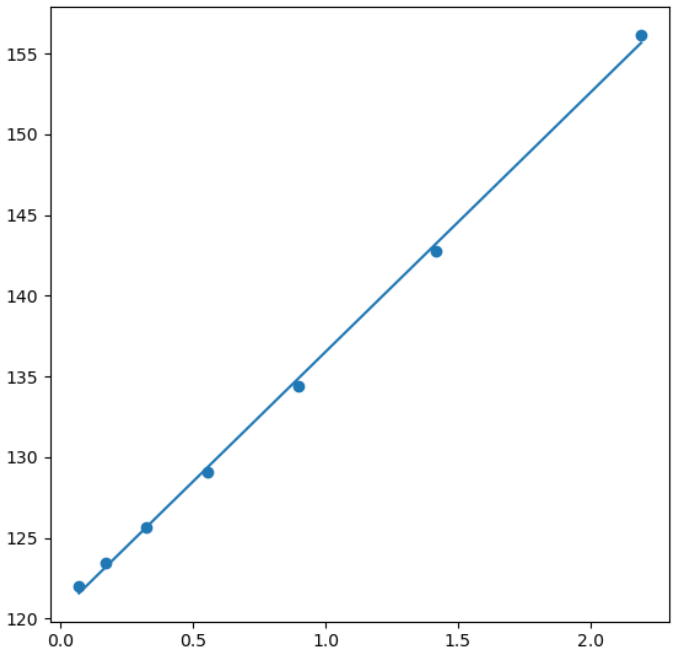

Figure 6.

Trabalho 2: Encontrar a solução no intervalo $[0, 80]$ do PVC $-\alpha y'' + vy' + \mu y = f(x)$, com $f(x) = \begin{cases} F, \text{ se } x = 0.2 \\ 0, \text{caso contrário} \end{cases}$.

Aqui será usado o mesmo método (diferenças finitas) e o mesmo programa utilizado na resolução da questão 6.

No enunciado completo do problema é dito que, em $x = 0$, o sangue inicialmente tem uma concentração C de impurezas; ao final, quando $x = 80$cm, o valor cai porque as impurezas saem por poros dos tubos (que são semipermeáveis). Considerando que o valor $F = 0.2$, escolhi os valores de contorno $\boldsymbol{y(0) = 0.4}$ **e** $\boldsymbol{y(80) = 0.1}$.

Para as constantes, escolhi os valores $\boldsymbol{\alpha = 0.5, v = 1, \mu = 0.05}$; além disso, escolhi ter $n = 400$ intervalos (ou $\Delta x = 0.2$). Tomei o cuidado de fazer $\frac{v\Delta x}{\alpha} = \frac{0.2}{0.5} = 0.4 < \frac{1}{2}$. O gráfico do resultado foi o seguinte:

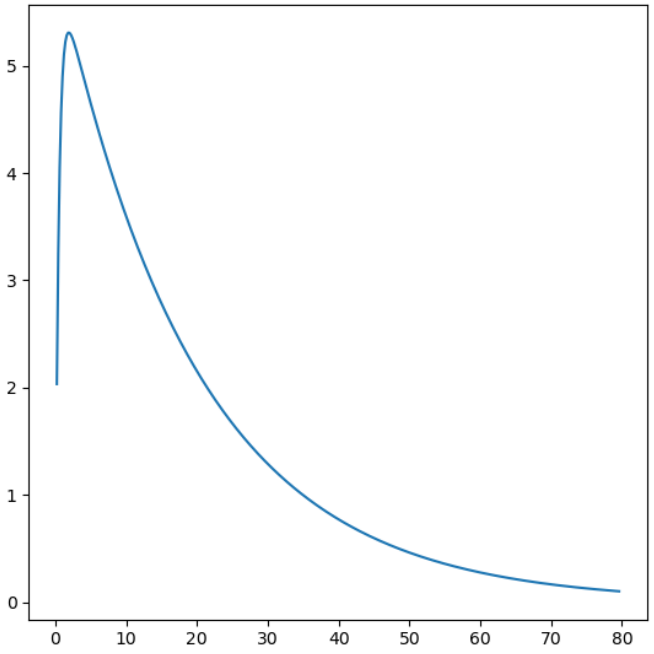

Figure 7. Resultado para o gráfico da concentração de impurezas no sangue ao longo do tubo de hemodiálise

2 Problemas qualitativos

Questão 4. Dos métodos apresentados para aproximar numericamente o valor de uma integral de uma função de que não se conhece a primitiva, há algum método que você acredita que nunca vai usar? Justifique cuidadosamente suas opiniões.

Como eu detalhei melhor na resposta da questão 8, eu acredito que na prática eu nunca chegarei a usar a regra do trapézio (pelo menos não em aplicações computacionais); isso porque os outros métodos (Simpson, Romberg, Gauss-Lagrange, etc) convergem mais rápido e são computacionalmente menos caros do que a regra do trapézio.

A regra do trapézio parece mais útil para estimar integrais numericamente "à mão", sem a ajuda de um computador.

Questão 5. Na resolução de um sistema do tipo A.x = B, qual ou quais métodos você tentaria – e em que ordem? Justifique cuidadosamente suas escolhas. Como é que se poderia avaliar se o resultado está satisfatório?

Essa escolha depende bastante da aplicação/do formato do sistema, mas de modo geral, em ordem dos que eu tentaria primeiro para os que eu tentaria por último:

1. Eliminação de Gauss-Jordan: se eu não tivesse nenhuma informação a respeito da matriz $\boldsymbol{A}$ e só precisasse resolver o sistema uma vez (sem reaproveitar a matriz $\boldsymbol{A}$), eu escolheria esse método, visto que é o método mais geral

2. Decomposição LU: eu usaria esse método se soubesse que precisaria reutilizar a matriz $\boldsymbol{A}$ várias vezes (por exemplo, resolvendo vários sistemas da forma $\mathbf{A}\mathbf{x} = \boldsymbol{B_i}$)

3. Gauss-Seidel: eu usaria esse método se soubesse que a matriz $\boldsymbol{A}$ é diagonal-dominante ($|a_{\text{ii}}| > \sum_{j \neq i} |a_{ij}|$); um exemplo em potencial são as matrizes que surgem ao resolver os sistemas lineares no método das diferenças finitas (como na questão 6 e no Trabalho 2), onde é fácil verificar a condição de diagonal-dominância.

Questão 8. Comente o que lhe pareceu útil num futuro exercício daquilo que foi visto em MS211 e os motivos dessa indicação. Faça o mesmo com o que lhe pareceu menos útil com o por quê dessa opinião.

Eu acredito que a parte mais útil para mim (fora os métodos para sistemas lineares/cálculo de inversas, dos quais já falei na primeira prova) foram os métodos de soluções de equações diferenciais. Na verdade estudar esses métodos "formalmente" me esclareceu uma série de dúvidas/problemas que eu tive no passado ao trabalhar em projetos envolvendo equações diferenciais.

Por exemplo, há algum tempo eu trabalhei em um projeto pessoal envolvendo simulações de gravitação com vários corpos (o clássico *n-body problem*). Em projetos para iniciantes, sempre que é preciso lidar com velocidade/aceleração, é sugerido o uso do método de Euler "seco" (sem ser dito explicitamente que se trata desse método), ou seja, algo como

```
def update_physics():
    ...
    position += velocity * delta_t
    velocity += acceleration * delta_t
```

Ou seja, $x_{n+1} = x_n + v_n(x_n, t)$ e $v_{n+1} = v_n + f(x_n, t)$, onde nesse exemplo específico do *n-body problem* $f(x,t) = f(x)$ representaria a aceleração gravitacional. Esse método funciona satisfatoriamente para simulações simples (de curta duração), mas passa a criar problemas quando a simulação dura muito tempo.

Antes do curso de MS211 eu não saberia dizer por que nas minhas simulações de gravitação a velocidade dos corpos eventualmente passava a aumentar rapidamente, gerando um resultado claramente não coerente fisicamente. Esse tipo de problema (o aumento artificial da energia mecânica de sistemas de partículas) com o método de Euler se repetiu em vários outros projetos (simulações de pêndulos acoplados, de sistemas de massa-mola pra várias partículas, etc).

Hoje, contudo, eu entendo o porquê desses resultados incoerentes (o erro global de ordem 1 do método de Euler) e consigo encontrar soluções muito melhores usando os métodos aprendidos no curso (em particular os de Runge-Kutta). Eu tenho certeza de que ainda encontrarei várias situações que envolvem a solução numérica de equações diferenciais, e os tópicos aprendidos no curso certamente serão muito úteis.

Quanto ao método menos útil: eu acredito que talvez a regra do trapézio para o cálculo de integrais seja o menos útil, simplesmente porque i) existem outros métodos (mais complexos, como Gauss-Legendre ou Romberg) que são muito mais rápidos e mais precisos e ii) mesmo dentre os métodos mais simples existem outros (como a regra de Simpson) que são mais rápidos e precisos que a regra do trapézio. O método em si, entretanto, certamente tem o seu valor educacional (é a forma mais intuitiva de se visualizar o funcionamento da integração numérica).

Questão 9. Comente algum tópico que, para você, não foi abordado de modo adequado na disciplina de MS211.

Eu não chegaria a dizer que não foi abordado de modo adequado, mas eu senti um pouco de falta de mais métodos de ajuste de curvas além do MMQ e das interpolações polinomiais. Por exemplo, um tema sobre o qual eu tinha curiosidade e esperava talvez chegar a ver no curso eram curvas de Bézier (que são bastante importantes em computação gráfica). Também gostaria, se possível, de ter visto tópicos envolvendo séries de Fourier (como métodos numéricos para o cálculo da transformada de Fourier e interpolação trigonométrica, por exemplo). Contudo, eu tenho plena consciência de que esses temas (em particular esses últimos envolvendo séries de Fourier) provavelmente fogem do escopo da disciplina. De modo geral, estou bastante satisfeito com o que aprendi no curso (especialmente na parte de métodos numéricos para equações diferenciais) e sinto que foi uma experiência rica e produtiva.