Matemática Numérica I

Pedro H A Konzen

Konzen, Pedro Henrique de Almeida

Matemática numérica I: notas de aula / Pedro Henrique de Almeida Konzen. –2024. Porto Alegre.- 2024.

"Esta obra é uma edição independente feita pelo próprio autor."

1. Métodos numéricos. 2. Análise numérica. 3. Linguagem Python.

Licença

Prefácio

O site notaspedrok.com.br é uma plataforma que construí para o compartilhamento de minhas notas de aula. Essas anotações feitas como preparação de aulas é uma prática comum de professoras/es. Muitas vezes feitas a rabiscos em rascunhos com validade tão curta quanto o momento em que são concebidas, outras vezes, com capricho de um diário guardado a sete chaves. Notas de aula também são feitas por estudantes - são anotações, fotos, prints, entre outras formas de registros de partes dessas mesmas aulas. Essa dispersão de material didático sempre me intrigou e foi o que me motivou a iniciar o site.

Com início em 2018, o site contava com apenas três notas incipientes. De lá para cá, conforme fui expandido e revisando os materiais, o site foi ganhando acessos de vários locais do mundo, em especial, de países de língua portuguesa. No momento, conta com mais de uma dezena de notas de aula, além de minicursos e uma coleção de vídeos e áudios.

As notas de **Matemática Numérica I** abordam tópicos introdutórios sobre representação de número em máquina, métodos numéricos para a resolução equações, para sistemas de equações lineares e não lineares, interpolação de funções e aproximação por mínimos quadrados. Códigos exemplos são apresentados em linguagem Python.

Aproveito para agradecer a todas(os) que de forma assídua ou esporádica contribuem com correções, sugestões e críticas! ;)

Pedro H A Konzen

https://www.notaspedrok.com.br

Conteúdo

Capítulo 1

Aritmética de Máquina

```
0.1 + 0.2 == 0.3
```

```
False
```

1.1 Sistema de Numeração Posicional

Cotidianamente, usamos o sistema de numeração posicional na base decimal. Por exemplo, temos

$$123.5 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 5 \times 10^{-1}, \tag{1.1}$$

onde o algarismo/dígito 1 está na posição 2 (posição das centenas), o dígito 2 está na posição 1 (posição das dezenas) e o dígito 3 está na posição 0 (posição das unidades). Mais geralmente, temos a representação decimal

$$\pm d_n \dots d_2 d_1 d_0, d_{-1} d_{-2} d_{-3} \dots \tag{1.2}$$

$$:= \pm \Big(d_n \times 10^n + \dots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0 \tag{1.3}$$

$$+ d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + d_{-3} \times 10^{-3} + \cdots \Big), \tag{1.4}$$

cujos os dígitos $d_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $i = n, \dots, 2, 1, 0, -1, -2, -3, \dots$. Observamos que esta representação posicional pode ser generalizada para outras bases numéricas.

Definição 1.1.1. (Representação posicional) Dada uma base $b \in \mathbb{N} \setminus \{0\}$, definimos a representação

$$\pm(d_n \dots d_2 d_1 d_0, d_{-1} d_{-2} d_{-3} \dots)_b \tag{1.5}$$
$$:= \pm \left(d_n \times b^n + \dots + d_2 \times b^2 + d_1 \times b^1 + d_0 \times b^0 \right. \tag{1.6}$$
$$\left. + d_{-1} \times b^{-1} + d_{-2} \times b^{-2} + d_{-3} \times b^{-3} + \dots \right), \tag{1.7}$$

onde os dígitos $d_i \in \{0, 1, \dots, b-1\}$[1], $i = n, \dots, 2, 1, 0, -1, -2, -3, \dots$.

Exemplo 1.1.1. (Representação binária) O número $(11010.101)_2$ está escrito na representação binária (base $b = 2$). Da Definição 1.1.1, temos

$$(\overset{4}{1}\ \overset{3}{1}\ \overset{2}{0}\ \overset{1}{1}\ \overset{0}{0} .\ \overset{-1}{1}\ \overset{-2}{0}\ \overset{-3}{1})_2 \tag{1.8}$$
$$= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \tag{1.9}$$
$$+ 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \tag{1.10}$$
$$= 26.625. \tag{1.11}$$

```
1*2**4 + 1*2**3 + 0*2**2 + 1*2**1 + 0*2**0 +
1*2**-1 + 0*2**-2 + 1*2**-3
```

```
26.625
```

1.1.1 Mudança de Base

Um mesmo número pode ser representado em diferentes bases. A mudança de base da representação de um dado número pode ser feita de várias formas. De forma geral, se temos um número x representado na base b_1 e queremos obter sua representação na base b_2, fazemos

1. Calculamos a representação do número x na base decimal.

2. Da calculada representação decimal, calculamos a representação de x na base b_2.

Observamos que o passo 1. ($b \to 10$) segue imediatamente da Definição 1.1.1. Agora, o passo 2. ($10 \to b$), podemos usar o seguinte procedimento. Suponhamos que x tenha a seguinte representação decimal

$$d_n d_{n-1} d_{n-2} \dots d_0, d_{-1} d_{-2} d_{-3} \dots \tag{1.12}$$

Então, separamos sua parte inteira $I = d_n d_{n-1} d_{n-2} \dots d_0$ e sua parte fracionária $F = 0, d_{-1} d_{-2} d_{-3} \dots$ $(x = I + F)$. Então, usando de sucessivas divisões de I pela base b desejada, obtemos sua representação nesta mesma base. Analogamente, usando de sucessivas multiplicações de F pela base b, obtemos sua representação nesta base. Por fim, basta somar as representações calculadas.

Exemplo 1.1.2. Obtenha a representação em base quartenária ($b = 4$) do número $(11010.101)_2$.

1. $b = 2 \to 10$. A representação de $(11010.101)_2$ segue direto da Definição 1.1.1 (veja, o Exemplo 1.1.1). Ou seja, temos

$$(\overset{4}{1}\ \overset{3}{1}\ \overset{2}{0}\ \overset{1}{1}\ \overset{0}{0}\ .\ \overset{-1}{1}\ \overset{-2}{0}\ \overset{-3}{1})_2 \tag{1.13}$$
$$= 2^4 + 2^3 + 2^1 + 2^{-1} + 2^{-3} \tag{1.14}$$
$$= 26.625. \tag{1.15}$$

```
2**4 + 2**3 + 2 + 2**-1 + 2**-3
```

```
26.625
```

2. $b = 10 \to 4$. Primeiramente, decompomos 26.625 em sua parte inteira $I = 26$ e em sua parte fracionária 0.625. Então, ao fazermos sucessivas divisões de I por $b = 4$, obtemos:

$$I = 26 \tag{1.16}$$
$$= 6 \times 4 + 2 \times 4^0 \tag{1.17}$$
$$= (1 \times 4 + 2) \times 4 + 2 \times 4^0 \tag{1.18}$$
$$= 1 \times 4^2 + 2 \times 4 + 2 \times 4^0 \tag{1.19}$$
$$= (122)_4. \tag{1.20}$$

```
I = int(26.625)
d_int = []
while (I != 0):
  d_int.insert(0, I % 4)
  I //= 4
print('(',*d_int,')_4',sep="")
```

Agora, para a parte fracionária, usamos sucessivas multiplicações de F por $b = 4$, obtendo:

$$F = 0.625 \tag{1.21}$$
$$= 2.5 \times 4^{-1} = 2 \times 4^{-1} + 0.5 \times 4^{-1} \tag{1.22}$$
$$= 2 \times 4^{-1} + 2 \times 4^{-1} \times 4^{-1} \tag{1.23}$$
$$= 2 \times 4^{-1} + 2 \times 4^{-2} \tag{1.24}$$
$$= (0.22)_4. \tag{1.25}$$

```
F = 26.625 % 1
d_fra = []
while (F != 0):
  F *= 4
  d_fra.append(int(F))
  F %= 1
print('(0,',*d_fra,')_4',sep="")
```

Por fim, dos passos 1. e 2., temos $(11010.101)_2 = (122.22)_4$.

```
print('(',*d_int,',',*d_fra,')_4', sep='')
```

```
(122.22)_4
```

1.1.2 Exercícios Resolvidos

ER 1.1.1. Forneça a representação decimal dos seguintes números:

a) $(10101)_2$

b) $(0.4321)_5$

c) $(23.5)_8$

d) $(A2A)_{11}$

e) $(BEBE)_{16}$

Solução.

a) $(1\overset{4}{0}\overset{3}{1}\overset{2}{0}\overset{1}{1}\overset{0}{)}_2$

```
0b10101
```

```
21
```

b) $(\overset{0}{0},\ \overset{-1}{4}\ \overset{-2}{3}\ \overset{-3}{2}\ \overset{-4}{1})_5$

```
4*5**-1+3*5**-2+2*5**-3+5**-4
```

```
0.9376000000000001
```

c) $(\overset{1}{2}\ \overset{0}{3},\ \overset{-1}{5})_8$

```
0o235 / 8**1
```

```
19.625
```

d) $(\overset{2}{A}\ \overset{1}{2}\ \overset{0}{A})_{11}$

```
int('A2A', 11)
```

```
1242
```

e) $(\overset{3}{B}\ \overset{2}{E}\ \overset{1}{B}\ \overset{0}{E})_{16}$

```
0xBEBE
```

```
48830
```

$\Diamond$

ER 1.1.2. Forneça a representação na base indicada dos seguintes números decimais:

a) $203 \rightarrow$ base 2

b) $0.671875 \rightarrow$ base 2

c) $17.25 \rightarrow$ base 8

d) $3245.875 \rightarrow$ base 16

Solução.

a) 203 → base 2 Usando o método Python `bin`, obtemos

```
bin(203)
```

```
'0b11001011'
```

ou seja, $203 = (11001011)_2$.

b) 0.671875 → base 2.

Executando o código

```
F = 0.671875
digs = []
while (F != 0):
  F *= 2
  digs.append(int(F))
  F %= 1
print('(0,',*digs,')_2',sep="")
```

obtemos que $0.671875 = (0.101011)_2$.

c) 17.25 → base 8

Temos que

$$\begin{aligned} 17.25 &= 17 + 0.25 && (1.26)\\ &= 16 + 1 + \frac{2}{8} && (1.27)\\ &= 2 \cdot 8^1 + 1 \cdot 8^0 + 2 \cdot 8^{-1} && (1.28)\\ &= (21.2)_8 && (1.29) \end{aligned}$$

d) 3245.875 → base 16

Executando o seguinte código

```
# base
b = 16
# dígitos
digs = "0123456789ABCDEF"

```

```
# número
x = 3245.875

# parte inteira
I = int(x)
di = []
while (I != 0):
  di.insert(0, I % b)
  I //= b

# parte fracionária
F = x % 1
df = []
while (F != 0):
  F *= b
  df.append(int(F))
  F %= 1

print('(',*[digs[d] for d in di],\
      ',',*[digs[d] for d in df],f')_{b}',sep="")
```

obtemos $3245.875 = (CAD, E)_{16}$.

$\Diamond$

ER 1.1.3. Na base indicada, forneça a representação dos seguintes números:

a) $(1101)_2 \rightarrow$ base 8

b) $(1011.0101)_2 \rightarrow$ base 8

Solução.

a) $(1101)_2 \rightarrow$ base 8

```
oct(0b1101)
```

```
'0o15'
```

Ou seja, $(1101)_2 = (15)_8$.

b) $(1011.0101)_2 \rightarrow$ base 8

Primeiro, convertemos $(1011.0101)_2$ para decimal (base 10).

```
0b10110101 / 2**4
```

```
11.3125
```

Logo, convertemos para a base octal (base 8) com o seguinte código:

```
# base
b = 8

# número
x = 11.3125

# parte inteira
I = int(x)
di = []
while (I != 0):
  di.insert(0, I % b)
  I //= b

# parte fracionária
F = x % 1
df = []
while (F != 0):
  F *= b
  df.append(int(F))
  F %= 1

print('(',*di,',',*df,f')_{b}',sep="")
```

Com este último, obtemos $(1011.0101)_2 = 11.3125 = (13.24)_8$

$\Diamond$

1.1.3 Exercícios

E.1.1.1. Obtenha a representação decimal dos seguinte números:

a) $(101101.00101)_2$

b) $(23.1)_4$

c) $(DAAD)_{16}$

d) $(33.11)_8$

e) $(51)_6$

E.1.1.2. Obtenha a representação dos seguintes números decimais na base indicada:

a) 10 na base 2.

b) 45.5 na base 2.

c) 41 na base octal.

d) 66.31640625 na base hexadecimal.

e) $0,\overline{3}$ na base 3.

E.1.1.3. Obtenha a representação dos seguintes números na base indicada:

a) $(101101.00101)_2$ na base 4.

b) $(23.1)_4$ na base 2.

c) $(2001)_{16}$ na base 8.

E.1.1.4. Obtenha a representação dos seguintes números na base indicada:

a) $(0.1)_3$ na base decimal.

b) $(0,\overline{1})_3$ na base decimal.

c) $0,\overline{3}$ na base octal.

E.1.1.5. Obtenha a representação dos seguintes números na base indicada:

a) 0.3 na base 4.

b) 0.3 na base 9.

c) $(A8)_{16}$ na base 5.

Respostas

E.1.1.1. a) 45.15625; b) 11.25; c) 55981; d) 27.140625; e) 31

E.1.1.2. a) $(1010)_2$; b) $(101101.1)_2$; c) $(51)_8$; d) $(42.51)_{16}$; e) $(0.1)_3$

E.1.1.3. a) $(231.022)_4$; b) $(1011.01)_2$; c) $(20001)_8$

E.1.1.4. a) $0,\overline{3}$; b) 1.5; c) $(0,\overline{25})_8$;

E.1.1.5. a) $(0.1\overline{03})_4$; b) $(0,\overline{27})$; c) $(2.2)_5$

1.2 Representação de Números em Máquina

Usualmente, números são manipulados em máquina através de suas representações em registros com *n-bits*. Ao longo desta seção, vamos usar a seguinte notação

$$[b_1\ b_2\ b_3\ \cdots\ b_n], \tag{1.30}$$

para representar um registro de *n-bits* $b_i \in \{0,1\}$, $i = 1, 2, \ldots, n$.

Na sequência, fazemos uma breve discussão sobre as formas comumente usadas para a manipulação de números em computadores.

1.2.1 Números Inteiros

O sistema de complemento de 2 é utilizado em computadores para a manipulação de números inteiros. Nesta representação, um registro de n *bits*

$$[d_1\ d_2\ d_3\ \cdots\ d_n], \tag{1.31}$$

representa o número inteiro

$$x = (d_{n-1}\ \ldots\ d_2\ d_1)_2 - d_n 2^{n-1}. \tag{1.32}$$

Exemplo 1.2.1. O registro de 8 *bits*[2]

$$[1\ 1\ 0\ 0\ 0\ 0\ 0\ 0] \tag{1.33}$$

representa o número

$$\begin{aligned} x &= -d_8 \cdot 2^{8-1} + (d_7\ d_6\ \ldots\ d_1)_2 && (1.34)\\ &= -0 \cdots 2^7 + (\overset{6}{0}\ \overset{5}{0}\ \overset{4}{0}\ \overset{3}{0}\ \overset{2}{0}\ \overset{1}{1}\ \overset{0}{1})_2 && (1.35)\\ &= 2^1 + 2^0 = 3. && (1.36) \end{aligned}$$

Podemos implementar um conversor de registro para número inteiro como segue

Código 1.1: packbits8.py

```
1 def packBitsInt8(dd):
2   x = -dd[7] * 2**7
3   for i, d in enumerate(dd[:7]):
4       x += d * 2**(i)
5   return x
```

Esta função, converte uma lista de *bits* (registro) no inteiro corresponde ao sistema de complemento 2.

```
1 packBitsInt8([1,1,0,0,0,0,0,0])
```

```
3
```

Na representação de complemento de 2 com n *bits*, o menor e o maior números inteiros são obtidos com os registros

$$-2^{n-1} \sim [0\ 0\ 0\ 0\ \cdots\ 1], \tag{1.37}$$
$$2^{n-1} - 1 \sim [1\ 1\ 1\ \cdots\ 1\ 0], \tag{1.38}$$

respectivamente. Já o zero é obtido com o registro

$$0 \sim [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]. \tag{1.39}$$

Exemplo 1.2.2. Com um registro de 8-*bits*, temos que o menor e o maior números inteiros que podem ser representados são

$$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1] \tag{1.40}$$
$$\sim -2^7 + (0000000)_2 = -128, \tag{1.41}$$

e

$$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 0] \tag{1.42}$$
$$\sim -0 \cdot 2^7 + (1111111)_2 = 127, \tag{1.43}$$

respectivamente.

Usando o Código 1.1, temos

```
1 packBitsInt8([0,0,0,0,0,0,0,1])
```

```
-128
```

```
1 packBitsInt8([1,1,1,1,1,1,1,0])
```

```
127
```

```
1 packBitsInt8([0,0,0,0,0,0,0,0])
```

```
0
```

Observação 1.2.1. No NumPy, o `dtype=numpy.int8` corresponde a inteiros de 8 *bits*.

```
1 import numpy as np
2 np.array([-127, 0, 3, 128, 129], dtype=np.int8)
```

```
array([-127,    0,    3, -128, -127], dtype=int8)
```

Consulte a lista de tipos básicos do NumPy em NumPy:Data types.

A adição de números inteiros na representação de complemento de 2 pode ser feita de maneira simples. Por exemplo, consideremos a soma $3 + 9$ usando registros de 8 *bits*. Temos

$$3 \sim [1\ 1\ 0\ 0\ 0\ 0\ 0\ 0] \tag{1.44}$$
$$9 \sim [1\ 0\ 0\ 1\ 0\ 0\ 0\ 0] + \tag{1.45}$$
$$- - - - - - - - - - \tag{1.46}$$
$$12 \sim [0\ 0\ 1\ 1\ 0\ 0\ 0\ 0] \tag{1.47}$$

No sistema de complemento de 2, a representação de um número negativo $-x$ pode ser obtida da representação de x, invertendo seus *bits* e somando 1. Por exemplo, a representação de -3 pode ser obtida da representação de 3, como segue

$$3 \sim [1\ 1\ 0\ 0\ 0\ 0\ 0\ 0]. \tag{1.48}$$

Invertendo seus *bits* e somando 1, obtemos

$$-3 \sim [1\ 0\ 1\ 1\ 1\ 1\ 1\ 1]. \tag{1.49}$$

A subtração de números inteiros usando a representação de complemento de 2 fica, então, tanto simples quanto a adição. Por exemplo:

$$3 \sim [1\ 1\ 0\ 0\ 0\ 0\ 0\ 0] \tag{1.50}$$
$$-9 \sim [1\ 1\ 1\ 0\ 1\ 1\ 1\ 1] + \tag{1.51}$$
$$- - - - - - - - - - \tag{1.52}$$
$$-6 \sim [0\ 1\ 0\ 1\ 1\ 1\ 1\ 1] \tag{1.53}$$

1.2.2 Ponto Flutuante

A manipulação de números decimais em computadores é comumente realizada usando a representação de ponto flutuante de 64 *bits*[3]. Nesta, um dado registro de 64 *bits*

$$[s \mid c_{10}\ c_9\ \ldots\ c_0 \mid m_1\ m_2\ \ldots\ m_{52}] \tag{1.54}$$

representa o número

$$x = (-1)^s M \cdot 2^{c-1023}, \tag{1.55}$$

onde M é chamada de mantissa e c da característica, as quais são definidas por

$$M := (1, m_1 m_2 m_3 \dots m_{52})_2, \tag{1.56}$$

$$c := (c_{10} \dots c_2 c_1 c_0)_2. \tag{1.57}$$

Exemplo 1.2.3. Por exemplo, na representação em ponto flutuante de 64 *bits*, temos que o registro

$$[1 \mid 1\ 0\ \dots\ 0\ 0 \mid 1\ 0\ 1\ 0\ 0\ \dots\ 0] \tag{1.58}$$

representa o número -3.25.

A seguinte função faz a conversão uma lista de 64 *bits* no número decimal corresponde ao sistema de ponto flutuante de 64 *bits*.

Código 1.2: packBitsDouble.py

```
def packBitsDouble(ld):
  s = ld[0]
  c = 0
  for i, d in enumerate(ld[1:12]):
      c += d * 2**(10-i)
  m = 1.
  for i, d in enumerate(ld[12:]):
      m += d * 2**(-(i+1))
  x = m * 2**(c - 1023)
  return -x if s else x
```

Por exemplo, usando-a para o registro acima, obtemos

```
ld = [0]*14
ld[0]=1
ld[1]=1
ld[12]=1
ld[13]=1
packBitsDouble(ld)
```

```
-3.5
```

1.2.3 Erro de Arredondamento

Dado um número real x, sua representação $fl(x)$ em ponto flutuante é o registro que representa o número mais próximo de x. Este procedimento é chamado de arredondamento por proximidade.

A seguinte função obtém a representação em ponto flutuante de 64 *bits* de um dado número x[4].

Código 1.3: unpackBitsDouble.py

```
import numpy as np

def unpackBitsDouble(x):
  ld = 64*[0]
  if ( x == 0):
      return ld
  elif (x < 0):
      ld [0] = 1
  x = np.fabs(x)
  c = int(np.log2(x) + 1023)
  m = x/2**(c - 1023)
  for i in range(11):
      ld [11 - i] = c % 2
      c //= 2
  m -= 1
  for i in range(52):
      m *= 2
      ld [12+ i] = int(m)
      m %= 1
  return ld
```

Por exemplo, $x = 1.1$ é representado pelo registro

```
ld = unpackBitsDouble(1.1)
ld
```

```
[0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 1,
```

```
1, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 1, 0]
```

que corresponde ao número

```
flx = packBitsDouble(ld)
print(f'{flx:1.51f}')
```

```
1.10000000000000008881784197
00125232338905334472656250
```

O erro de arredondamento é $|x - fl(x)| \approx 8.9 \times 10^{-17}$.

Observação 1.2.2. O seguinte código é uma solução mais pythonica para obter-se o registro em ponto flutuante de 64 *bits* de $x = 1.1$.

```
''.join(f'{c:08b}' for c in struct.pack('!d', 1.1)
)
```

```
'0011111111110001
1001100110011001
1001100110011001
1001100110011010'
```

Recomendamos consultar [4] para mais informações sobre a conversão eficiente de números decimais em pontos flutuantes.

Observemos que o erro de arredondamento varia conforme o número dado, podendo ser zero no caso de $x = fl(x)$. Comumente, utiliza-se o **épsilon de máquina** como uma aproximação desse erro. O épsilon de máquina é definido como a distância entre o número 1 e seu primeiro sucessor em ponto flutuante. Temos

```
ld = unpackBitsDouble(1)
ld
```

```
[0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
```

```
ld[63] = 1
x = packBitsDouble(ld)
x-1
```

```
2.220446049250313e-16
```

Ou seja, o épsilon de máquina é

$$\text{eps} := 2^{-52} \approx 2.22 \times 10^{-16}. \tag{1.59}$$

Observação 1.2.3. O método `numpy.finfo` pode ser usado para obtermos várias informações sobre o sistema de números em ponto flutuante. Por exemplo, temos

```
import numpy as np
finfo = np.finfo(np.double)
finfo.eps
```

```
2.220446049250313e-16
```

```
finfo.min
```

```
-1.7976931348623157e+308
```

```
finfo.max
```

```
1.7976931348623157e+308
```

A aritmética em ponto flutuante requer arredondamentos sucessivos de números. Por exemplo, a computação da soma de dois números dados x e y é feita a partir de suas representações em ponto flutuante $fl(x)$ e $fl(y)$. Então, computa-se $z = fl(x) + fl(y)$ e o resultado é $fl(z)$. Observe, inclusive que $fl(x+y)$ pode ser diferente de $fl(fl(x) + fl(y))$. Por exemplo

```
0.1 + 0.2 == 0.3
```

```
False
```

1.2.4 Exercícios Resolvidos

ER 1.2.1. No sistema de complemento 2 de 8 *bits*, forneça o registro que representa os seguintes números inteiros:

a) 1

b) -1

c) 15

d) -15

Solução. A seguinte função, obtém o registro de complemento 2 de 8-*bits* de um dado número inteiro x.

```
def unpackBitsInt8(x):
  ld = 8*[0]
  if (x < 0):
      ld[7] = 1
      x += 2**(7)
  for i in range(7):
      ld[i] = x % 2
      x //= 2
  return ld
```

Usando-a, obtemos os seguintes resultados:

```
# a) 1
unpackBitsInt8(1)
```

```
[1, 0, 0, 0, 0, 0, 0, 0]
```

```
# b) -1
unpackBitsInt8(-1)
```

```
[1, 1, 1, 1, 1, 1, 1, 1]
```

```
# c) 15
unpackBitsInt8(15)
```

```
[1, 1, 1, 1, 0, 0, 0, 0]
```

```
unpackBitsInt8(-15)
```

```
[1, 0, 0, 0, 1, 1, 1, 1]
```

◊

ER 1.2.2. Qual é o número decimal positivo mais próximo de zero que pode ser representado como um ponto flutuante de 64-*bits*. Também, forneça seu registro.

Solução. Um registro em ponto flutuante de 64-*bits* tem a forma

$$[s \mid c_{10}\ c_9\ \ldots\ c_0 \mid m_1\ m_2\ \ldots\ m_{52}] \tag{1.60}$$

e representa o número

$$x = (-1)^s M \cdot 2^{c-1023}, \tag{1.61}$$

onde M é chamada de mantissa e c da característica, as quais são definidas por

$$M := (1, m_1 m_2 m_3 \ldots m_{52})_2, \tag{1.62}$$
$$c := (c_{10} \ldots c_2 c_1 c_0)_2. \tag{1.63}$$

Tendo em vista que o registro nulo é reservado para o número decimal zero, temos que o número positivo mais próximo de zero é obtido com sinal $s = 0$, a mantissa $M = 1$ e a característica $c = 1$, no que obtemos o decimal

$$x = 2^{-1022} \tag{1.64}$$
$$\approx 2.2250738585072014e - 308 \tag{1.65}$$

Seu registro é

$$[0 \mid 0\ 0\ \ldots\ 1 \mid 0\ 0\ \ldots\ 0] \tag{1.66}$$

O resultado pode ser verificado com os seguintes comandos:

```
import numpy as np
import struct
x = np.finfo(np.double).tiny; x
```

```
2.2250738585072014e-308
```

```
''.join(f'{c:08b}' \
  for c in struct.pack('!d', x))
```

```
'0000000000010000
0000000000000000
0000000000000000
0000000000000000'
```

$\Diamond$

ER 1.2.3. Em aplicações que não necessitam de muita precisão, a representação de números decimais no sistema de ponto flutuante de 32 *bits* é mais eficiente (no sentido de velocidade de processamento computacional). Neste sistema, um registro de 32-*bits*

$$[s \mid c_7\ c_6\ \ldots\ c_0 \mid m_1\ m_2\ \ldots\ m_{23}] \tag{1.67}$$

representa o número

$$x = (-1)^s \cdot M \cdot 2^{c-127} \tag{1.68}$$

onde,

$$M = (1, m_1 m_2 \ldots m_{23})_2 \tag{1.69}$$
$$c = (c_7 c_6 \ldots c_0)_2 \tag{1.70}$$

a) Forneça o registro do ponto flutuante de 32-*bits* que representa o número 42.5.

b) Qual é o sucessor em ponto flutuante de 32-*bits* do número decimal 1. Forneça, também, o épsilon de máquina deste sistema.

Solução.

a) O registro do ponto flutuante de 32-*bits* que representa o número 42.5 pode ser computado com o seguinte código:

```
x = 42.5
ld = 32*[0]
```

```
c = int(np.log2(x) + 127)
m = x/2**(c-127)
for i in range(8):
  ld[8-i] = c % 2
  c //= 2
m -= 1
for i in range(23):
  m *= 2
  ld[9+i] = int(m)
  m %= 1
ld
```

```
[0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 1, 0, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
```

Alternativamente, pode-se obter o registro como segue:

```
''.join(f'{c:08b}' \
for c in struct.pack('!f', 42.5))
```

```
'01000010001010100000000000000000'
```

b) No sistema de ponto flutuante de 32-*bits*, o sucessor de 1 tem o registro

$$[0 \mid 0\ 1\ 1\ \ldots\ 1 \mid 0\ 0\ \ldots\ 0\ 1] \tag{1.71}$$

donde, sua mantissa é $m = 1 + 2^{-23}$, característica $c = 127$ e corresponde ao número decimal

$$x = (-1)^0 \cdot (1 + 2^{-23}) \cdot 2^{127-127} \tag{1.72}$$
$$x = 1 + 2^{-23} \tag{1.73}$$

Portanto, o épsilon de máquina neste sistema é

$$\text{eps} = x - 1 \tag{1.74}$$
$$= 2^{-23} \tag{1.75}$$

```
1 np.float32(2**-23)
```

```
1.1920929e-07
```

$\Diamond$

1.2.5 Exercícios

E.1.2.1. Considerando a representação de complemento de 2 de números inteiros, obtenha os registros de 8-*bits* dos seguintes números:

a) 17

b) -17

c) 32

d) -32

E.1.2.2. Considerando a representação de complemento de 2 de números inteiros, obtenha os registros de 16-*bits* dos seguintes números:

a) 1024

b) -1024

E.1.2.3. Considerando a representação de complemento de 2 de números inteiros, qual é o maior número que pode ser representado por um registro de 32-*bits* da forma

$$[1\ 0\ b_2\ b_3\ b_4\ \cdots\ b_{30}\ 1], \tag{1.76}$$

onde $b_i \in \{0, 1\}$, $i = 2, 3, 4, \cdots, 30$.

E.1.2.4. Obtenha os registros em ponto flutuante de 64-*bits* dos seguintes números:

a) -1.25

b) 3

E.1.2.5. Assumindo o sistema de ponto flutuante de 32-*bits*, obtenha o registro e o erro de arredondamento na representação dos seguintes números decimais:

a) 0.1

b) 10.1

c) 100.1

Respostas

E.1.2.1. a) [10001000]; b) [11110111]
c) [00000100]; d) [00000111]

E.1.2.2. a) [0000000000100000];
b) [0000000000111111];

E.1.2.3. $[10111\ldots11] \sim -3$

E.1.2.4. a) [1 | 0 1 1 ... 1 | 1 0 1 0 0 ... 0];
b) [0 | 1 0 0 ... 0 | 1 0 0 ... 0]

E.1.2.5.

a)
```
[0, 0, 1, 1, 1, 1, 0, 1,
1, 1, 0, 0, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 1]
```

$|0.1 - fl(0.1)| \approx 1.5e^{-9}$

b) `[0, 1, 0, 0, 0, 0, 0, 1,`

```
0, 0, 1, 0, 0, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 1, 0]
```

$|10.1 - fl(10.1)| \approx 3.8e^{-7}$

c) `[0, 1, 0, 0, 0, 0, 1, 0,`
`1, 1, 0, 0, 1, 0, 0, 0,`
`0, 0, 1, 1, 0, 0, 1, 1,`
`0, 0, 1, 1, 0, 0, 1, 1]`

$|100.1 - fl(100.1)| \approx 1.5e^{-6}$

1.3 Notação Científica e Arredondamento

Enquanto que a manipulação computacional de números decimais é feita usando-se da aritmética em ponto flutuante, a interpretação dos parâmetros dos problemas de interesse e seus resultados é normalmente feita com poucos dígitos. Nesta seção, introduziremos algumas notações que serão utilizadas ao longo deste material.

A **notação científica** é a representação de um dado número na forma

$$d_n \dots d_2 d_1 d_0, d_{-1} d_{-2} d_{-3} \dots \times 10^E, \tag{1.77}$$

onde d_i, $i = n, \dots, 1, 0, -1, \dots$, são algarismos da base 10. A parte à esquerda do sinal $\times$ é chamada de **mantissa** do número e E é chamado de **expoente** (ou ordem de grandeza).

Exemplo 1.3.1. O número 31.515 pode ser representado em notação científica das seguintes formas

$$\begin{aligned} 31.415 \times 10^0 &= 3.1415 \times 10^1 && (1.78)\\ &= 314.15 \times 10^{-1} && (1.79)\\ &= 0.031415 \times 10^3, && (1.80) \end{aligned}$$

entre outras tantas possibilidades.

Em Python, usa-se a letra `e` para separar a mantissa do expoente na notação científica. Por exemplo

```
# 31.415 X 10^0
31.415e0
```

```
31.515
```

```
# 3.1415 X 10^1
3.1415e1
```

```
31.515
```

```
# 314.15 X 10^-1
314.15e-1
```

```
31.515
```

```
# 0.031415 X 10^3
0.031415e3
```

```
31.415
```

No exemplo anterior (Exemplo 1.3.1), podemos observar que a representação em notação científica de um dado número não é única. Para contornar isto, introduzimos a **notação científica normalizada**, a qual tem a forma

$$d_0, d_{-1}d_{-2}d_{-3}\ldots \times 10^E, \tag{1.81}$$

com $d_0 \neq 0$[5].

Exemplo 1.3.2. O número 31.415 representado em notação científica normalizada é 3.1415×10^1.

Em Python, podemos usar de `strings` formatadas para imprimir um número em notação científica normalizada. Há várias especificações de formatação disponíveis[6]. Por exemplo, temos

```
x = 31.415
print(f"{x:e}")
```

```
3.141500e+01
```

Como vimos na seção anterior, usamos da aritmética de ponto flutuante nas computações, com a qual os números são representados com muito mais

dígitos dos quais costumeiramente estamos interessados na interpretação dos resultados. Isto nos leva de volta a questão do arredondamento.

Dizemos que um número está representado com n **dígitos significativos** (na notação científica normalizada) quando está escrito na forma

$$d_0, d_1 d_2 \dots d_{n-1} \times 10^E, \tag{1.82}$$

com $d_0 \neq 0$.

Exemplo 1.3.3. Estudamos as seguintes representações do número 31.415:

a) com 5 dígitos significativos

```
x = 31.415
print(f"{x:.4e}")
```

```
3.1415e+01
```

b) com 6 dígitos significativos

```
print(f"{x:.5e}")
```

```
3.14150e+01
```

c) com 4 dígitos significativos

```
print(f"{x:.3e}")
```

```
3.142e+01
```

Neste último caso, fez-se necessário arredondar o número.

1.3.1 Arredondamento

Observamos que pode ocorrer a necessidade de se arredondar um número para obter sua representação com um número finito de dígitos significativos. Por exemplo, para representarmos o número $x = 3.1415 \times 10^1$ com 3 dígitos significativos, precisamos determinar de que forma vamos considerar a contribuição de seus demais dígitos a direita. Isto, por sua vez, é determinado pelo tipo de arredondamento que iremos utilizar.

O tipo de arredondamento mais comumente utilizado é o chamado **arredondamento por proximidade com desempate par**. Neste, a representação escolhida é aquela mais próxima do número dado. Por exemplo, a representação de

$$x = 3.1415 \times 10^1 \tag{1.83}$$

com três dígitos significativos é

$$x = 3.14 \times 10^1. \tag{1.84}$$

Agora, sua representação com apenas dois dígitos significativos é

$$x = 3.1 \times 10^1. \tag{1.85}$$

No caso de empate, usa-se a seguinte regra: 1) se o último dígito significativo ser par, este é mantido; 2) se o último dígito significativo ser ímpar, este é acrescido de uma unidade. Por exemplo, no caso do número $x = 3.1415 \times 10^1$, sua representação com 4 dígitos significativos é

$$x = 3.142 \times 10^1. \tag{1.86}$$

Observação 1.3.1. O arredondamento por proximidade com desempate par é o padrão do IEEE 754[7]. No entanto, devemos lembrar que a maioria dos números decimais não tem representação exata no sistema de ponto flutuante. Por exemplo,

```
1 x = 31.415
2 print(f'{x:.3e}')
```

```
3.141e+01
```

Embora o arrendamento não seja o esperado, o que ocorre é que $x = 31.415$ não tem representação exata em ponto flutuante, de fato

```
1 print(f'{x:.25e}')
```

```
3.1414999999999999147348717e+01
```

No restante deste material estaremos assumindo a notação científica normalizada com arredondamento por proximidade com desempate par.

1.3.2 Exercícios Resolvidos

ER 1.3.1. Faça o cálculo exato e a computação de

$$\frac{0.33411 \times 10^2 - 271.28 \times 10^{-1}}{2000 \times 10^{-3}} \tag{1.87}$$

Forneça os resultados com 4 dígitos significados.

Solução.

- Por cálculo exato.

$$\frac{0.33411 \times 10^2 - 271.28 \times 10^{-1}}{2000 \times 10^{-3}} \tag{1.88}$$

$$= \frac{334.11 \times 10^{-1} - 271.28 \times 10^{-1}}{2 \times 10^0} \tag{1.89}$$

$$= \frac{63.83 \times 10^{-1}}{2} \tag{1.90}$$

$$= 31.415 \times 10^{-1} \tag{1.91}$$

Arredondando o resultado para 4 dígitos significativos, obtemos 3.142.

- Por computação.

```
x = (0.33411e2 - 271.28e-1)/2000e-3
x
```

```
3.1415000000000006
```

```
print(f'{x:.3e}')
```

```
3.142e+00
```

◊

ER 1.3.2. Obtenha os arredondamentos dos seguintes números decimais para quantidade de dígitos significativos indicada em cada caso. Então, compare com a computada em ponto flutuante.

a) 2.7128 com 4 dígitos significativos.

b) 2.7128 com 2 dígitos significativos.

c) 1.9910 com 3 dígitos significativos.

d) 1.9910 com 2 dígitos significativos.

e) 5.5555 com 4 dígitos significativos.

f) 5.6555 com 4 dígitos significativos.

Solução.

a) 2.7128 com 4 dígitos significativos = 2.713

```
1 f'{2.7128:.3e}'
```

```
'2.713e+00'
```

b) 2.7128 com 2 dígitos significativos = 2.7

```
1 f'{2.7128:.1e}'
```

```
'2.7e+00'
```

c) 1.9910 com 3 dígitos significativos = 1.99

```
1 f'{1.9910:.2e}'
```

```
'1.99e+00'
```

d) 1.9910 com 2 dígitos significativos = 2.0

```
1 f'{1.9910:.1e}'
```

```
'2.0e+00'
```

e) 5.5555 com 4 dígitos significativos = 5.556

```
1 f'{5.5555:.3e}'
```

```
'5.556e+00'
```

f) 5.6555 com 4 dígitos significativos = 5.556

```
1 f'{5.6555:.3e}'
```

```
'5.655e+00'
```

$\Diamond$

1.3.3 Exercícios

E.1.3.1. Obtenha a representação dos seguintes números decimais em notação científica normalizada com a quantidade de dígitos indicada em cada caso. Então, compare com o arredondamento computado em ponto flutuante. Caso haja diferença, explique.

a) π com 6 dígitos significativos.

b) $\pi/10$ com 6 dígitos significativos.

c) $\sqrt{2}/\sqrt{3}$ com 7 dígitos significativos.

E.1.3.2. Compute a seguinte expressão

$$\frac{\sqrt{\pi} - \ln(0.9)}{75\cos\left(\frac{\pi}{4}\right)}. \tag{1.92}$$

Forneça a resposta com 7 dígitos significativos.

E.1.3.3. Forneça o arredondamento dos seguintes números decimais para 2 dígitos significativos. Então, compare com o arrendamento computado em ponto flutuante. Caso haja diferença, explique.

a) 0.625

b) 0.615

c) 0.635

E.1.3.4. Seja f_s a função que recebe número decimal e retorna sua aproximação por arrendamento com 2 dígitos significativos. Calcule

a) $f_s(2\pi - e)$

b) $2f_s(\pi) - f_s(e)$

c) Por que $f_s(2\pi - e) \neq 2f_s(\pi) - f_s(e)$?

E.1.3.5. Explique o porquê de

```
1 np.sqrt(3)**2 == 3
```

```
False
```

Respostas

E.1.3.1. a) 3.14159×10^0, `3.14159e00+`; b) 3.14159×10^{-1}, `3.14159e-01`; c) 8.164922×10^{-1}, `8.164966e-01`

E.1.3.2. 3.540841×10^{-1}

E.1.3.3. a) 6.2×10^{-1}, `6.2e-01`; b) 6.2×10^{-1}, `6.1e-01`; c) 6.4×10^{-1}; `6.4e-01`

E.1.3.4. a) 3.5; b) 3.6; c) Operar sobre números arredondados acarreta perda de exatidão.

E.1.3.5. Dica: $\sqrt{3}$ não tem representação exata em ponto flutuante.

1.4 Tipos e Medidas de Erros

Ao utilizarmos computadores na resolução de problemas matemáticos, acabamos obtendo soluções aproximadas. A diferença entre a solução exata e a solução aproximada computada é chamada de erro. O erro é comumente classificado nas seguintes duas categorias:

- **Erro de arredondamento**

  Este é o erro que ocorre na representação aproximada de números na máquina.

- **Erro de truncamento**

Este é o erro que ocorre na interrupção (truncamento) de um procedimento com infinitos passos.

Exemplo 1.4.1. (Erro de Arredondamento.) O erro de arredondamento em aproximar π por 3.1415×10^0 é de aproximadamente 9.3×10^{-5}.

```
import numpy as np
np.pi - 3.1415e0
```

```
9.265358979293481e-05
```

Exemplo 1.4.2. (Erro de Truncamento.) Consideramos a seguinte série numérica $\sum_{n=0}^{\infty} 1/n! = e \approx 2.7183 \times 10^0$. Ao computarmos esta série no computador, precisamos truncá-la em algum n suficientemente grande. Por exemplo, truncando a série em seu nono termo, temos

$$\sum_{n=0}^{\infty} \frac{1}{n!} \approx \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{8!} \tag{1.93}$$

$$\approx 2.71827876984127 =: \tilde{e}. \tag{1.94}$$

```
import math
x = 0
for n in range(9):
  x += 1./math.factorial(n)
print(math.fabs(math.e - x))
```

A diferença $|e - \tilde{e}| \approx 3 \times 10^{-6}$ é o erro de truncamento associado.

Suponhamos, agora, que x seja o valor exato (valor esperado) de uma quantidade de interesse e $\tilde{x}$ o valor computado (aproximação de x). Em matemática numérica, utilizamos frequentemente as seguintes medidas de erro:

- **Erro absoluto**:

$$\varepsilon_{\text{abs}} := |x - \tilde{x}|. \tag{1.95}$$

- **Erro relativo**:

$$\varepsilon_{\text{rel}} := \frac{|x - \tilde{x}|}{|x|} \, (\times 100\%) \, . \tag{1.96}$$

A vantagem do erro relativo é em levar em conta a ordem de grandeza da quantidade x.

Exemplo 1.4.3. Estudamos os seguintes casos:

a) $x = 1.0$ e $\tilde{x} = 1.1$:

$$\begin{aligned}
\varepsilon_{\text{abs}} &= |x - \tilde{x}| && (1.97)\\
&= |1.0 - 1.1| && (1.98)\\
&= |-0.1| && (1.99)\\
&= 1 \times 10^{-1}. && (1.100)\\
\varepsilon_{\text{rel}} &= \frac{|x - \tilde{x}|}{|x|} && (1.101)\\
&= \frac{|1.0 - 1.1|}{|1.0|} && (1.102)\\
&= \frac{|-0.1|}{|1.0|} && (1.103)\\
&= 1 \times 10^{-1} = 10\%. && (1.104)
\end{aligned}$$

```
x = 1.0
xa = 1.1
eabs = abs(x - xa); eabs
```

```
0.10000000000000009
```

```
erel = eabs/abs(x)
erel
```

```
0.10000000000000009
```

b) $x = 1000.0$ e $\tilde{x} = 1100.0$:

$$\begin{aligned}
\varepsilon_{\text{abs}} &= |x - \tilde{x}| && (1.105)\\
&= |1000.0 - 1100.0| && (1.106)\\
&= 1 \times 10^{2}. && (1.107)\\
\varepsilon_{\text{rel}} &= \frac{|x - \tilde{x}|}{|x|} && (1.108)
\end{aligned}$$

$$= \frac{|1000.0 - 1100.0|}{|1000.0|} \tag{1.109}$$

$$= \frac{|-100.0|}{|1000.0|} \tag{1.110}$$

$$= 1 \times 10^{-1} = 10\%. \tag{1.111}$$

```
x = 1000.0; xa = 1100.0
eabs = abs(x - xa); eabs
```

```
100.0
```

```
erel = eabs/abs(x); erel
```

```
0.1
```

Outra medida de erro comumente empregada é o **número de dígitos significativos corretos**. Dizemos que $\tilde{x}$ aproxima x com n dígitos significativos corretos, quando

$$\underbrace{\frac{|x - \tilde{x}|}{|x|}}_{\varepsilon_{\text{rel}}} < 5 \times 10^{-n}. \tag{1.112}$$

Isso significa que ao arredondarmos x e $\tilde{x}$ ambos com n dígitos, obtemos o mesmo resultado.

Exemplo 1.4.4. Estudamos os seguintes casos:

- $x = 2$ e $\tilde{x} = 2.4$

  $$\varepsilon_{\text{rel}} = 0.2 < 5 \times 10^{-1} \tag{1.113}$$

  Temos que $\tilde{x} = 2.4$ aproxima $x = 2$ com um dígito significativo correto. Note que ambos são iguais quando os arredondamos para um dígito.

- $x = 2$ e $\tilde{x} = 2.5$

  $$\frac{|x - \tilde{x}|}{|x|} = 0.25 < 5 \times 10^{-1} \tag{1.114}$$

  Temos que $\tilde{x} = 2.5$ é uma aproximação com 1 dígito significativo correto de $x = 2$. Note que ambos são iguais quando os arredondamos para um dígito.

- $x = 1$ e $\tilde{x} = 1.5$:

$$\frac{|x - \tilde{x}|}{|x|} = 0.5 < 5 \times 10^0, \tag{1.115}$$

Temos que $\tilde{x} = 1.5$ é uma aproximação com zero dígito significativo correto de $x = 1$. Note que ao arredondarmos[8] $\tilde{x}$ para um dígito, obtemos $\tilde{x} \approx 2$, enquanto que $x = 1$.

1.4.1 Propagação de Erros

Nesta seção, vamos introduzir uma estimativa para a propagação de erros (de arredondamento) na computação de um problema. Para tando, vamos considerar o caso de se calcular o valor de uma dada função f em um dado ponto x, i.e. queremos calcular y com

$$y = f(x). \tag{1.116}$$

Agora, assumindo que x seja conhecido com um erro $\varepsilon(x)$, este se propaga no cálculo da f, levando a um erro $\varepsilon(y)$ no valor calculado de y. Ou seja, temos

$$y + \varepsilon(y) = f(x + \varepsilon(x)). \tag{1.117}$$

Denotamos $\varepsilon_{\text{abs}}(x) = |\varepsilon(x)|$ o erro absoluto associado a x e $\varepsilon_{\text{abs}}(y) = |\varepsilon(y)|$ o erro absoluto associado a y.

Nosso objetivo é estimar $\varepsilon_{\text{abs}}(y)$ com base em $\varepsilon_{\text{abs}}(x)$. Para tanto, tomamos a aproximação de $f(x + \varepsilon(x))$ dada pelo polinômio de Taylor de grau 1 de f em torno de x, i.e.

$$f(x + \varepsilon(x)) = f(x) + f'(x)\varepsilon(x) + O\left(\varepsilon^2(x)\right). \tag{1.118}$$

Então, de (1.116) e (1.117), temos

$$\varepsilon(y) = f'(x)\varepsilon(x) + O(\varepsilon^2(x)). \tag{1.119}$$

Daí, passando ao valor absoluto e usando a desigualdade triangular, obtemos

$$\varepsilon_{\text{abs}}(y) = \left|f'(x)\varepsilon(x) + O(\varepsilon^2(x))\right| \tag{1.120}$$

$$\leq |f'(x)|\varepsilon_{\text{abs}}(x) + O\left(\varepsilon^2_{\text{abs}}(x)\right). \tag{1.121}$$

Deste resultado, obtemos a seguinte estimativa de propagação de erro

$$\varepsilon_{\text{abs}}(y) \approx |f'(x)|\varepsilon_{\text{abs}}(x). \tag{1.122}$$

Exemplo 1.4.5. Consideramos o problema em se calcular

$$y = f(x) = x^2 \operatorname{sen}(x) \tag{1.123}$$

com $x = \pi/3 \pm 0.1$. Usando (1.122) para estimarmos o erro absoluto $\varepsilon_{\text{abs}}(y)$ no cálculo de y com base no erro absoluto $\varepsilon_{\text{abs}}(x) = 0.1$, calculamos

$$\begin{aligned} \varepsilon_{\text{abs}}(y) &= |f'(x)|\varepsilon_{\text{abs}}(x) && (1.124)\\ &= |2x\operatorname{sen}(x) + x^2\cos(x)|\varepsilon_{\text{abs}}(x) && (1.125)\\ &= 2.3621 \times 10^{-1}. && (1.126) \end{aligned}$$

```
1 import math
2 x = math.pi/3; eabsx = 0.1
3 eabsy = math.fabs(2*x*math.sin(x) \
4   + x**2 * math.cos(x)) * eabsx
5 print(f"{eabsy:.4e}")
```

```
2.3621e-01
```

Com isso, concluímos que um erro em x de tamanho 0.1 é propagado no cálculo de $f(x)$, causando um erro pelo menos duas vezes maior em y. Também, podemos interpretar este resultado do ponto de vista do erro relativo. O erro relativo associado a x é

$$\begin{aligned} \varepsilon_{\text{rel}}(x) &= \frac{\varepsilon_{\text{abs}}(x)}{|x|} && (1.127)\\ &= \frac{0.1}{\pi/3} && (1.128)\\ &= 9.5493 \times 10^{-2} \approx 10\%, && (1.129) \end{aligned}$$

acarretando um erro relativo em y de

$$\begin{aligned} \varepsilon_{\text{rel}}(y) &= \frac{\varepsilon_{\text{abs}}(y)}{|y|} && (1.130)\\ &= \frac{\varepsilon_{\text{abs}}(y)}{|f(x)|} && (1.131)\\ &= 2.4872 \times 10^{-2} \approx 25\%. && (1.132) \end{aligned}$$

```
import math
x = math.pi/3; eabsx = 0.1
erelx = eabsx/math.fabs(x)
print(f"{erelx*100: .0f} %")
```

```
10 %
```

```
f = lambda x: x**2 * math.sin(x)
df = lambda x: 2*x*math.sin(x) \
  + x**2 * math.cos(x)
eabsy = math.fabs(df(x)) * eabsx
erely = eabsy/math.fabs(f(x))
print(f"{erely*100: .0f} %")
```

```
25 %
```

Associada à estimativa (1.4.5), temos

$$\begin{aligned}\varepsilon_{\text{rel}}(y) &= \frac{\varepsilon_{\text{abs}}(y)}{|y|} \\ &= \frac{|f'(x)|}{|y|}\varepsilon_{\text{abs}}(x) \\ &= \frac{|x|\cdot|f'(x)|}{|f(x)|}\frac{\varepsilon_{\text{abs}}(x)}{|x|} \\ &= \left|\frac{xf'(x)}{f(x)}\right|\varepsilon_{\text{rel}}(x).\end{aligned}$$

Desta última equação, definimos o **número de condicionamento** de f, denotado por

$$\kappa_f(x) := \left|\frac{xf'(x)}{f(x)}\right|. \tag{1.133}$$

Observamos que $\kappa_f(x)$ é a escala com que erros em x são propagados no cálculo de $y = f(x)$.

Exemplo 1.4.6. O número de condicionamento da função $f(x) = x^2\,\text{sen}(x)$ no ponto $x = \pi/3$ é calculado por

$$\kappa_f(x) = \left|\frac{xf'(x)}{f(x)}\right| \tag{1.134}$$

$$= \left| \frac{x\,[2x\,\mathrm{sen}(x) + x^2\cos(x)]}{x^2\,\mathrm{sen}(x)} \right|. \tag{1.135}$$

Substituindo x por $\pi/3$, obtemos

$$\kappa_f(\pi/3) = 2.6046. \tag{1.136}$$

Observamos que o resultado é compatível com os obtidos no Exemplo 1.4.5.

```
import math
f = lambda x: x**2 * math.sin(x)
df = lambda x: 2*x*math.sin(x) \
  + x**2 * math.cos(x)
x = math.pi/3
kf = math.fabs(x*df(x)/f(x))
print(f"{kf:.4f}")
```

```
2.6046
```

A estimativa (1.122) pode ser generalizada para uma função de várias variáveis. No caso de uma função $y = f(x_1, x_2, \ldots, x_n)$, temos

$$\varepsilon_{\mathrm{abs}}(y) = \sum_{k=1}^{n} \left| \frac{\partial f}{\partial x_k} \right| \varepsilon_{\mathrm{abs}}(x_k). \tag{1.137}$$

Exemplo 1.4.7. Consideremos o problema em se calcular

$$z = f(x, y) = x^2\,\mathrm{sen}(x)\cos(y) \tag{1.138}$$

com

$$x = \frac{\pi}{3} \pm 0.1, \tag{1.139}$$

$$y = \frac{\pi}{4} \pm 0.02. \tag{1.140}$$

Usando (1.137) para estimarmos o erro absoluto $e_{\mathrm{abs}}(z)$ no cálculo de z com base nos erros absolutos $e_{\mathrm{abs}}(x) = 0.1$ e $e_{\mathrm{abs}}(y) = 0.02$, calculamos

$$e_{\mathrm{abs}}(z) = \left| \frac{\partial f}{\partial x} \right| e_{\mathrm{abs}}(x) + \left| \frac{\partial f}{\partial y} \right| e_{\mathrm{abs}}(y) \tag{1.141}$$

$$= |(2x\,\mathrm{sen}(x) + x^2\cos(x))\cos(y)|e_{\mathrm{abs}}(x) \tag{1.142}$$

$$+ \left|-x^2\,\mathrm{sen}(x)\,\mathrm{sen}(y)\right| e_{\mathrm{abs}}(y) \tag{1.143}$$

$$= 1.8046 \times 10^{-1}. \tag{1.144}$$

```
import math
x = math.pi/3
eabsx = 0.1
y = math.pi/4
eabsy = 0.02
eabsz = math.fabs((2*x*math.sin(x) \
  + x**2*math.cos(x))*math.cos(y))*eabsx \
  + math.fabs(-x**2*math.sin(x)*math.sin(y))*eabsy
print(f"{eabsz:1.4e}")
```

```
1.8046e-01
```

1.4.2 Cancelamento Catastrófico

No computador (com aritmética de ponto flutuante de 64-*bits*), as operações e funções elementares são computadas, usualmente, com um erro próximo do épsilon de máquina (eps $\approx 10^{-16}$). Entretanto, em algumas situações estas operações fundamentais acarretam erros maiores, causando uma perda de precisão.

O chamado cancelamento catastrófico ocorre quando computamos a diferença entre dois números próximos. Para ilustrá-lo, considaremos os seguintes números

$$x = 314150000001549, \tag{1.145}$$

$$y = 314150000002356. \tag{1.146}$$

Assumindo os arredondamentos de x e y com 12 dígitos significativos, temos

$$\tilde{x} = 314150000002000, \tag{1.147}$$

$$\tilde{y} = 314150000002000. \tag{1.148}$$

Os erros relativos associados às aproximações de x e y por $\tilde{x}$ e $\tilde{y}$ são

$$e_{rel}(x) = \frac{|x - \tilde{x}|}{|x|} \approx 10^{-10}\%, \tag{1.149}$$

$$e_{rel}(y) = \frac{|y - \tilde{y}|}{|y|} \approx 10^{-10}\%, \tag{1.150}$$

respectivamente. Agora, temos

$$y - x = 807, \tag{1.151}$$
$$\tilde{y} - \tilde{x} = 0. \tag{1.152}$$

Ou seja, o erro relativo na aproximação de $y - x$ por $\tilde{y} - \tilde{x}$ é

$$e_{rel}(y - x) = \frac{|(y - x) - (\tilde{y} - \tilde{x})|}{(y - x)} \tag{1.153}$$
$$= \frac{807}{807} = 100\%! \tag{1.154}$$

Exemplo 1.4.8. Na tabela abaixo temos os erros em se computar

$$\frac{(1 + x^4) - 1}{x^4} \tag{1.155}$$

para diferentes valores de x.

x	erro
1	0
10^{-1}	1.1×10^{-13}
10^{-2}	6.1×10^{-9}
10^{-3}	8.9×10^{-5}
10^{-4}	1.0×10^{0}
10^{-5}	1.0×10^{0}

Observamos que, para o valor de $x = 0.001$ o erro na computação já é da ordem de 10^{-5} e para valores de x menores ou iguais a 0.0001 o erro é catastrófico. Isto ocorre, pois se $x \leq 10^{-4}$, então $x^4 \leq 10^{-16} <$ eps e, portanto, $(1 + x^4) - 1 = 0$.

Exemplo 1.4.9. Uma equação de segundo grau $ax^2 + bx + c = 0$ tem raízes

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \tag{1.156}$$
$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{1.157}$$

Entretanto, no caso de b ser positivo, a fórmula (1.156) não é adequada para a computação da raiz x_1, pois pode ocorrer cancelamento catastrófico. Podemos contornar este problema reescrevendo (1.156) da seguinte forma

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \cdot \frac{-b-\sqrt{b^2-4ac}}{-b-\sqrt{b^2-4ac}} \tag{1.158}$$

$$= \frac{b^2-b^2+4ac}{2a(-b-\sqrt{b^2-4ac})} \tag{1.159}$$

$$= \frac{-2c}{b+\sqrt{b^2-4ac}}, \tag{1.160}$$

a qual não sofre mais de cancelamento catastrófico. Observamos que também pode ocorrer cancelamento catastrófico no cálculo de x_2 pela fórmula (1.157), no caso de b ser negativo.

1.4.3 Exercícios Resolvidos

ER 1.4.1. O número de Euler é definido por

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} \tag{1.161}$$

Determine o erro relativo da aproximação de e pelo truncamento da série com 4 termos.

Solução. Denotamos $x = e$ e

$$\tilde{x} = \sum_{n=0}^{3} \frac{1}{n!} \tag{1.162}$$

$$= \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} \tag{1.163}$$

$$= \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} \tag{1.164}$$

$$= 2 + \frac{1}{2} + \frac{1}{6} \tag{1.165}$$

$$= \frac{16}{6} \tag{1.166}$$

O erro relativo é

```
import math as m
x = m.e
xa = 16./6
eabs = m.fabs(x-xa)
erel = eabs/m.fabs(x)
print(f"{erel*100:1.1f} %")
```

```
1.9 %
```

Concluímos que o erro relativo é de 1.9%.

◊

ER 1.4.2. Calcule o número de condicionamento $\kappa_f(x)$ para $f(x) = x^n$.

Solução. Calculamos o número de condicionamento como segue

$$\kappa_f(x) = \left|\frac{xf'(x)}{f(x)}\right| \tag{1.167}$$

$$= \left|\frac{x \cdot nx^{n-1}}{x^n}\right| \tag{1.168}$$

$$= \left|\frac{nx^n}{x^n}\right| \tag{1.169}$$

$$= n, \quad x \neq 0. \tag{1.170}$$

◊

ER 1.4.3. Calcule as raízes do seguinte polinômio quadrático

$$p(x) = 10^{-6}x^2 + 10^2x + 3 \times 10^{-3} \tag{1.171}$$

com 10 dígitos significativos corretos.

Solução. As raízes do polinômio quadrático podem ser calculados pela fórmula de Bhaskara

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1.172}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{1.173}$$

No entanto, a computação da raiz x_1 sofre de cancelamento catastrófico. Para contornar este problema, usamos (1.160), i.e.

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \tag{1.174}$$

Com o código

```
import math as m

a = 1e-6
b = 1e2
c = 3e-3

delta = b**2 - 4*a*c

x1 = -2*c/(b + m.sqrt(delta))
x2 = (-b - m.sqrt(delta))/(2*a)

print(f"{x1:1.9e}, {x2:1.9e}")
```

obtemos as saídas

```
x_1 = -3.000000000e-05
x_2 = -1.000000000e+08
```

◊

1.4.4 Exercícios

E.1.4.1. Calcule o erro absoluto na aproximação de

a) π por 3.14.

b) $10e$ por 27.18.

Forneça as respostas com 4 dígitos significativos.

E.1.4.2. Calcule o erro relativo na aproximação de

a) π por 3.14.

b) $10e$ por 27.18.

Forneça as respostas em porcentagem.

E.1.4.3. Com quantos dígitos significativos corretos

a) 3.13 aproxima π?

b) 27.21 aproxima $10e$?

E.1.4.4. Obtenha uma estimativa do erro de truncamento em se aproximar o valor de sen(1) usando-se $p_5(1)$, onde $p_5(x)$ é o polinômio de Taylor de grau 5 da função sen(x) em torno de $x = 0$.

E.1.4.5. Considerando que $x = 2 \pm 0.1$, estime o erro absoluto em se calcular $y = e^{-x^2}\cos(\pi x/3)$. Forneça a estimativa com 7 dígitos significativos por arredondamento.

E.1.4.6. Considerando que $x = 2 \pm 2\%$ e $y = 1.5 \pm 0.3$, estime o erro absoluto em se calcular $y = e^{-x^2}\cos(\pi y/3)$. Forneça a estimativa com 6 dígitos significativos por arredondamento.

E.1.4.7. Considere a computação de

$$y = \frac{1 - \cos(h)}{h} \tag{1.175}$$

para $h = 10^{-9}$. Compute o valor de y reescrevendo esta expressão de forma a mitigar o cancelamento catastrófico. Forneça o valor computado de y com 2 dígitos significativos por arredondamento.

Respostas

E.1.4.1. a) 1.593×10^{-3}; b) 2.818×10^{-1};

E.1.4.2. a) 0.051%; b) 0.01%;

E.1.4.3. a) 3; b) 3

E.1.4.4. $1/6! \approx 1.4 \times 10^{-3}$.

E.1.4.5. 2.002083×10^{-3}

E.1.4.6. 5.75403×10^{-3}

E.1.4.7. 5.0×10^{-10}

Capítulo 2

Equação com Uma Incógnita

Neste capítulo, discutiremos sobre métodos numéricos para resolver equações com uma incógnita real. Observamos que toda equação pode ser reescrita na seguinte forma equivalente

$$f(x) = 0, \tag{2.1}$$

onde f é uma função adequada. Isto é, o problema de se encontrar a incógnita de uma dada equação pode ser reescrito como um problema de encontrar os zeros (ou raízes) de uma função de uma variável real.

Os métodos numéricos que abordaremos ao longo deste capítulo são descritos para problemas da forma (2.1).

2.1 Método da Bisseção

O método da bisseção explora o fato de que toda função contínua f com $f(a) \cdot f(b) < 0$ (i.e., $f(a)$ e $f(b)$ tem sinais diferentes) tem pelo menos um zero no intervalo (a, b)[9].

Exemplo 2.1.1. Consideramos o problema de resolver a equação

$$\begin{aligned} \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) = x^3 - \frac{\pi}{4}x^2 \\ - \frac{5\pi^2}{16}x - \frac{3\pi^3}{64}. \end{aligned} \tag{2.2}$$

Este problema é equivalente a encontrar os zeros da seguinte função

$$f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \tag{2.3}$$

Os zeros exatos[10] desta função são $x_1 = 3\pi/4 \approx 2.3562$ e $x_2 = x_3 = -\pi/4 \approx -0.78540$ (consulte a Figura 2.1).

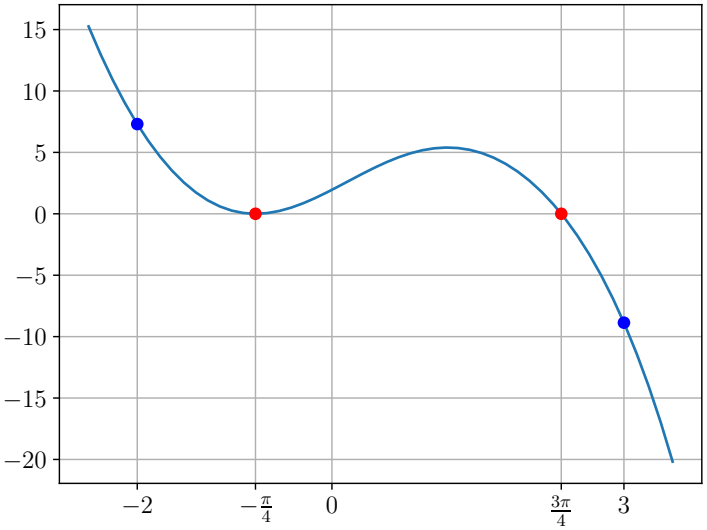

Figura 2.1: Esboço da função f do Exemplo 2.1.1.

Observamos que esta função é contínua e que, por exemplo, $f(-2) > 0$ e $f(3) < 0$, logo $f(-2) \cdot f(3) < 0$ e, de fato, f tem pelo menos um zero[11] no intervalo $(-2, 3)$.

Consideramos, então, uma função f contínua tal que $f(a) \cdot f(b) < 0$. O método da bisseção é iterativo, a primeira aproximação para uma solução de $f(x) = 0$ é tomada como o ponto médio do intervalo (a, b), i.e.

$$x^{(0)} = \frac{a^{(1)} + b^{(1)}}{2}, \tag{2.4}$$

onde $a^{(0)} = a$ e $b^{(0)} = b$. Daí, se ocorrer $f(x^{(0)}) = 0$ o problema está resolvido. Caso contrário, f tem pelo menos um zero num dos subintervalos $(a^{(0)}, x^{(0)})$ ou $(x^{(0)}, b^{(0)})$, pois $f(a^{(0)}) \cdot f(x^{(0)}) < 0$ ou $f(x^{(0)}) \cdot f(b^{(0)}) < 0$, respectivamente e exclusivamente. No primeiro caso, escolhemos $(a^{(1)}, b^{(1)}) = (a^{(0)}, x^{(0)})$ ou, no segundo caso, tomamos $(a^{(1)}, b^{(1)}) = (x^{(0)}, b^{(0)})$. Então, a segunda aproximação para uma solução é computada como

$$x^{(1)} = \frac{a^{(1)} + b^{(1)}}{2}. \tag{2.5}$$

O procedimento se repete até obtermos uma aproximação com a precisão desejada.

Exemplo 2.1.2. Consideremos o problema de encontrar um zero da função

$$\begin{aligned} f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.6}$$

Do esboço de seu gráfico (Figura 2.1), observamos que $f(2) \cdot f(3) \neq 0$ sendo que o zero $x = 3\pi/4 \approx 2.3562$ de f está no intervalo $(2, 3)$. Aplicando o método da bisseção com intervalo inicial $(a^{(0)}, b^{(0)}) = (2, 3)$ e aproximação inicial $x^{(0)} = (a^{(0)} + b^{(0)})/2$, obtemos as aproximações apresentadas na Tabela 2.1.

```
1  import numpy as np
2
3  f = lambda x: np.sin(x+np.pi/4)**2 \
4    - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
5    + 3*np.pi**3/64
6
7  a = 2
8  b = 3
9  x = (a + b)/2
10 print(f"0: a={a:.4f}, b={b:.4f}, x={x:.4f}")
11 for k in range(9):
12   s = np.sign(f(a)*f(x))
13   if (s == -1):
```

Tabela 2.1: Resultados referentes ao Exemplo 2.1.2.

k	$a^{(k)}$	$b^{(k)}$	$x^{(k)}$	s
0	2.0000	3.0000	2.5000	-1
1	2.0000	2.5000	2.2500	1
2	2.2500	2.5000	2.3750	-1
3	2.2500	2.3750	2.3125	1
4	2.3125	2.3750	2.3438	1
5	2.3438	2.3750	2.3594	-1
6	2.3438	2.3594	2.3516	1
7	2.3516	2.3594	2.3555	1
8	2.3555	2.3594	2.3574	-1
9	2.3555	2.3574	2.3564	-1

$s := f(a^{(k)}) \cdot f(x^{(k)})$

```
14      b = x
15    elif (s == 1):
16      a = x
17    else:
18      break
19    x = (a + b)/2
20    print(f"{k+1}: a={a:.4f}, b={b:.4f}, x={x:.4f}")
```

2.1.1 Análise Numérica

Dada uma função contínua $f : [a, b] \to \mathbb{R}$ com $f(a) \cdot f(b) < 0$, vamos mostrar que o método da bisseção é **globalmente convergente** e tem **ordem de convergência linear**.

Definição 2.1.1. (Método Iterativo Globalmente Convergente.) Um método iterativo

$$x^{(0)} = \text{aprox. inicial}, \tag{2.7}$$

$$x^{(k+1)} = g\left(x^{k}\right), \tag{2.8}$$

com $k = 0.1, \ldots$, é dito **globalmente convergente**, quando

$$\lim_{k\to\infty} \left|x^{(k+1)} - x^{(k)}\right| = 0 \tag{2.9}$$

para qualquer escolha de $x^{(0)}$.

Definição 2.1.2. (Ordem de convergência.) Seja $\left(x^{(k)}\right)_{k=1}^{\infty}$ uma sequência convergente com

$$\lim_{k\to\infty} \left|x^{(k+1)} - x^{(k)}\right| = 0, \tag{2.10}$$

com $x^{(k+1)} - x^{(k)} \neq 0$ para todo k. Dizemos que $\left(x^{(k)}\right)_{k=1}^{\infty}$ converge com ordem $\alpha > 0$ e com constante de erro assintótica $C > 0$, quando

$$\lim_{k\to\infty} \frac{\left|x^{(k+1)} - x^{*}\right|}{\left|x^{(k)} - x^{*}\right|^{\alpha}} = C. \tag{2.11}$$

Em geral, quando maior o valor de α, mais rapidamente é a convergência das iteradas de um método iterativo. Os seguintes casos são particularmente importantes:

- Se $\alpha = 1$ e $C < 1$, o método é **linearmente convergente**.
- Se $\alpha = 2$, o método é **quadraticamente convergente**.

Convergência e Precisão

Teorema 2.1.1. *Seja $f : [a, b] \to \mathbb{R}$ função contínua com $f(a) \cdot f(b) < 0$. Então, o método da bisseção é globalmente convergente.*

Demonstração. Seja $(x^{(k)})_{k=1}^{\infty}$ a sequência de aproximações[12] do método da bisseção. Por construção, temos

$$\begin{aligned}
\left|x^{(k)} - x^{(k-1)}\right| &\leq b^{(k-1)} - a^{(k-1)} && (2.12)\\
&\leq \frac{b^{(k-2)} - a^{(k-2)}}{2} && (2.13)\\
&\vdots && (2.14)\\
&\leq \frac{b^{(0)} - a^{(0)}}{2^{k-1}} && (2.15)
\end{aligned}$$

Ou seja, obtemos a **estimativa de convergência**

$$\left|x^{(k)} - x^{(k-1)}\right| \leq \frac{b^{(0)} - a^{(0)}}{2^{k-1}} \tag{2.16}$$

Daí, segue que

$$\lim_{k\to\infty} \left|x^{(k)} - x^{(k-1)}\right| = \lim_{k\to\infty} b^{(k)} - a^{(k-1)} \tag{2.17}$$

$$= 0. \tag{2.18}$$

□

Exemplo 2.1.3. No Exemplo 2.1.2 aplicamos o método da bisseção para a função

$$\begin{aligned} f(x) &= \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ &+ \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.19}$$

no intervalo $(2, 3)$. Ao recuperarmos os valores de $\left|x^{(k)} - x^{(k-1)}\right|$ obtemos

k	$x^{(k)}$	$\left\|x^{(k)} - x^{(k-1)}\right\|$
0	2.5000	-x-
1	2.2500	2.5×10^{-1}
2	2.3750	1.2×10^{-1}
3	2.3125	6.2×10^{-2}
4	2.3438	3.1×10^{-2}
5	2.3594	1.6×10^{-2}
6	2.3516	7.8×10^{-3}
7	2.3555	3.9×10^{-3}
8	2.3574	2.0×10^{-3}
9	2.3564	9.8×10^{-4}

Observamos que os valores estão de acordo com a estimativa de convergência 2.16, donde

$$|x^{(9)} - x^{(8)}| \leq \frac{b^{(0)} - a^{(0)}}{2^9} \tag{2.20}$$

$$= \frac{3-2}{2^9} = 1.9 \times 10^{-3} \tag{2.21}$$

O Teorema 2.1.1 nos garante a convergência do método da bisseção e uma estimativa de **precisão** (Equação 2.16). O teorema não garante que o método converge para um zero da função objetivo, apenas garante que as aproximações convergem para algum valor no intervalo inicial dado.

Convergência e Exatidão

Dada uma função contínua e estritamente monótona[13] $f : [a, b] \to \mathbb{R}$ com $f(a) \cdot f(b) < 0$, temos que o método da bisseção converge para o zero de f em $[a, b]$.

Teorema 2.1.2. *Seja $f : [a, b] \to \mathbb{R}$ função contínua e estritamente monótona com $f(a) \cdot f(b) < 0$. Então, o método da bisseção converge para o zero de f em $[a, b]$.*

Demonstração. Das hipóteses temos que f tem um único zero x^* em (a, b). Seja $(x^{(k)})_{k=1}^{\infty}$ a sequência de aproximações[14] do método da bisseção. Por construção, temos

$$|x^{(k)} - x^*| \leq \frac{b^{(k)} - a^{(k)}}{2} \tag{2.22}$$

$$\leq \frac{b^{(k-1)} - a^{(k-1)}}{2^2} \tag{2.23}$$

$$\vdots \tag{2.24}$$

$$\leq \frac{b^{(1)} - a^{(1)}}{2^k}, \tag{2.25}$$

donde, obtemos a seguinte estimativa do **erro de truncamento**

$$|x^{(k)} - x^*| \leq \frac{b^{(1)} - a^{(1)}}{2^k}. \tag{2.26}$$

E, daí também, segue que o método converge para o zero de f, pois

$$\lim_{k\to\infty} |x^{(k)} - x^*| = \lim_{k\to\infty} \frac{b^{(1)} - a^{(1)}}{2^k} = 0. \tag{2.27}$$

□

Observação 2.1.1. (Estimativa de Exatidão.) A estimativa de truncamento 2.26 é também um **estimativa de exatidão**, i.e. nos fornece uma medida do erro na k-ésima aproximação do método da bisseção.

Exemplo 2.1.4. No Exemplo 2.1.2 aplicamos o método da bisseção para a função

$$f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \tag{2.28}$$

no intervalo $(2, 3)$. A aplicação do método, nos fornece

k	$x^{(k)}$	$\left\|x^{(k)} - x^*\right\|$
0	2.3560	1.4×10^{-1}
1	2.2500	1.1×10^{-1}
2	2.3750	1.9×10^{-2}
3	2.3125	4.4×10^{-2}
4	2.3438	1.2×10^{-2}
5	2.3594	3.2×10^{-3}
6	2.3516	4.6×10^{-3}
7	2.3555	7.3×10^{-4}
8	2.3574	1.2×10^{-3}
9	2.3564	2.5×10^{-4}

onde, $x^* = x_1 = 3\pi/4$. Observamos que este resultado é consistente com a estimativa do erro de truncamento (2.26), da qual temos

$$|x^{(9)} - x^*| \leq \frac{b^{(0)} - a^{(0)}}{2^{10}} \tag{2.29}$$

$$= \frac{1}{2^{10}} = 1.0\text{e}{-3}. \tag{2.30}$$

Observação 2.1.2. (Ordem de Convergência Linear.) A estimativa de convergência (2.26) também pode ser usada para mostrarmos que, assintoticamente, o método da bisseção tem a seguinte taxa de convergência linear

$$\left|x^{(k+1)} - x^{(k)}\right| \lesssim \frac{1}{2}\left|x^{(k)} - x^{(k-1)}\right|^{\mathbf{1}}. \tag{2.31}$$

2.1.2 Zeros de Multiplicidade Par

Sejam f uma função suave e x^* um zero de multiplicidade par de f. Observamos que o método da bisseção não é diretamente aplicável para aproximar x^*. Isto ocorre, pois, neste caso, x^* será um ponto de mínimo ou de máximo local de f, não havendo pontos a e b próximos de x^* tal que $f(a) \cdot f(b) < 0$.

Agora, sendo x^* um zero de f de multiplicidade $2m$, temos que ela admite a seguinte decomposição

$$f(x) = (x - x^*)^{2m} g(x), \tag{2.32}$$

onde g é uma função suave e $g(x^*) \neq 0$. Daí, a derivada de f

$$f'(x) = 2m(x - x^*)^{2m-1} g(x) + (x - x^*)^{2m} g'(x), \tag{2.33}$$

tem x^* como um zero de multiplicidade $2m - 1$ (ímpar) e, desta forma, podemos aplicar o método da bisseção em f' para aproximar x^*.

Exemplo 2.1.5. A função

$$\begin{aligned} f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.34}$$

tem $x = -\pi/4 \approx -0.7854$ como um zero de multiplicidade par (veja Figura 2.2).

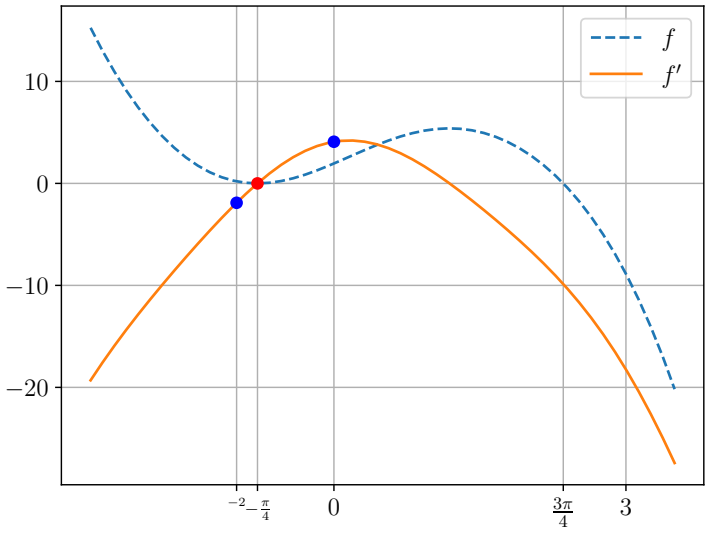

Figura 2.2: Esboço do gráfico da f e de sua derivada f' dada no Exemplo 2.1.5.

Para aplicarmos o método da bisseção para aproximarmos este zero, primeiramente, derivamos f

$$f'(x) = 2\sin(x+\pi/4)\cos(x+\pi/4) - 3x^2 + \frac{\pi}{2}x + \frac{5\pi^2}{16}. \tag{2.35}$$

O esboço do gráfico de f' (Figura 2.2) mostra que $f'(-1)\cdot f'(0) < 0$ sendo que no intervalo $(-1,0)$ f' tem um zero de multiplicidade ímpar. Então, aplicando o método da bisseção a f' no intervalo inicial $(a^{(1)}, b^{(1)}) = (-1,0)$, obtemos os resultados apresentados na Tabela 2.2. Nesta tabela são apresentados as iteradas até a convergência da solução com precisão de 10^{-3}.

Tabela 2.2: Resultados referentes ao Exemplo 2.1.2.

k	$a^{(k)}$	$b^{(k)}$	$x^{(k)}$	s
0	−1.0000e+0	0.0000e+0	−5.0000e−1	-1
1	−1.0000e+0	−5.0000e−1	−7.5000e−1	-1
2	−1.0000e+0	−7.5000e−1	−8.7500e−1	1
3	−8.7500e−1	−7.5000e−1	−8.1250e−1	1
4	−8.1250e−1	−7.5000e−1	−7.8125e−1	-1
5	−8.1250e−1	−7.8125e−1	−7.9688e−1	1
6	−7.9688e−1	−7.8125e−1	−7.8906e−1	1
7	−7.8906e−1	−7.8125e−1	−7.8516e−1	-1
8	−7.8906e−1	−7.8516e−1	−7.8711e−1	1
9	−7.8711e−1	−7.8516e−1	−7.8613e−1	1

$s := f'(a^{(k)}) \cdot f'(x^{(k)})$

2.1.3 Exercícios

E.2.1.1. Use o método da bisseção para aproximar um zero de

$$f(x) = x^3 \operatorname{sen}(x) - \cos(x) \tag{2.36}$$

aplicando como intervalo inicial $(a^{(0)}, b^{(0)}) = (0.5, 1)$ e aproximação inicial $x^{(0)} = (a^{(0)} + b^{(0)})/2$. Faça, então, 6 iterações de forma a obter a aproximação $x^{(6)}$ e forneça-a com 7 dígitos significativos por arredondamento.

E.2.1.2. Considere o método da bisseção para aproximar um zero de $f(x) = x^3 \operatorname{sen}(x) - \cos(x)$, aplicando como intervalo inicial $(a^{(0)}, b^{(0)}) = (0.5, 1)$ e aproximação inicial $x^{(0)} = (a^{(0)} + b^{(0)})/2$. Use a estimativa de convergência (2.26)

$$\left| x^{(k)} - x^* \right| \leq \frac{b^{(0)} - a^{(0)}}{2^k}, \tag{2.37}$$

para estimar o número mínimo de iterações k_{conv} necessárias para se obter a solução com exatidão de 10^{-4}. Então, compute $x^{(k_{conv})}$ e forneça-o com 6 dígitos significativos por arredondamento.

E.2.1.3. Use o método da bisseção para computar a(s) solução(ões) das seguintes equações com precisão de 8 dígitos significativos.

a) $x = 2^{-x}$ para $0 \leq x \leq 2$.

b) $e^{-x^2} = 3x - x^2$ para $-1 \leq x \leq 4$.

E.2.1.4. Use o método da bisseção para encontrar uma aproximação com precisão de 10^{-4} do zero de

$$\begin{aligned} f(x) = &(-x^2 + 1.154x - 0.332929)\cos(x) + x^2 \\ &- 1.154x + 0.332929 \end{aligned} \tag{2.38}$$

no intervalo $(0.55, 0.65)$. Forneça a aproximação computada com 7 dígitos significativos por arredondamento.

E.2.1.5. Aplique o método da bisseção para encontrar o ponto crítico[15] de

$$f(x) = (1 - x^2)e^{-x^2} \tag{2.39}$$

no intervalo $(0, 2)$. Obtenha o resultado com precisão de 5 dígitos significativos por arredondamento.

Respostas

E.2.1.1. 9.179688×10^{-1}

E.2.1.2. 9.15833e−1

E.2.1.3. a) 6.4118574×10^{-1}; b) 3.3536470×10^{-1}; 2.9999589

E.2.1.4. 5.770508×10^{-1}

E.2.1.5. 1.4142e+0

2.2 Método da Falsa Posição

O método da falsa posição é uma variação do método da bisseção. Dada uma função f contínua, escolhemos um intervalo inicial (a, b) tal que $f(a) \cdot f(b) < 0$ (i.e. f tem sinais trocados nos pontos a e b). Então, uma aproximação para o zero de f neste intervalo é computada como o ponto de interseção da reta secante a f pelos pontos $(a, f(a))$ e $(b, f(b))$, i.e.

$$x = a - \frac{b-a}{f(b) - f(a)} f(a). \tag{2.40}$$

Veja a Figura 2.3.

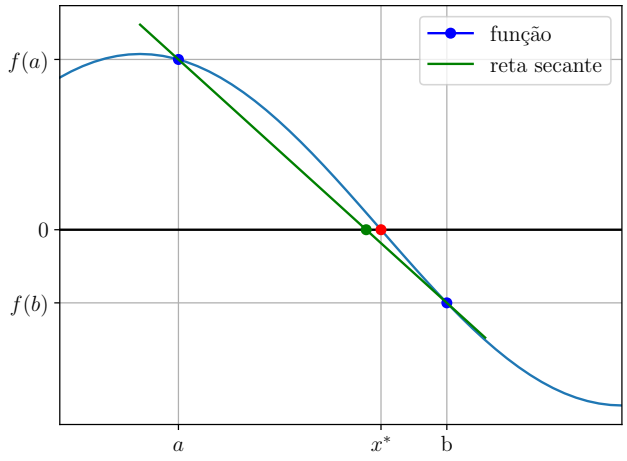

Figura 2.3: Ilustração do método da falsa posição.

Mais explicitamente, o método consiste no seguinte procedimento iterativo:

1. Determinar um intervalo $(a^{(1)}, b^{(1)})$ tal que $f(a^{(1)}) \cdot f(b^{(1)}) < 0$.
2. Para $k = 1, 2, 3, \cdots, N$:

   2.1 $x^{(k)} = a^{(k)} - \dfrac{b^{(k)} - a^{(k)}}{f(b^{(k)}) - f(a^{(k)})} f(a^{(k)})$

2.2 Verificar critério de parada.

2.3 Se $f(a^{(k)}) \cdot f(x^{(k)}) < 0$, então $a^{(k+1)} = a^{(k)}$ e $b^{(k+1)} = x^{(k)}$.

2.4 Se $f(x^{(k)}) \cdot f(b^{(k)}) > 0$, então $a^{(k+1)} = x^{(k)}$ e $b^{(k+1)} = b^{(k)}$.

Exemplo 2.2.1. Consideremos o problema de aproximar o zero de

$$f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \tag{2.41}$$

no intervalo $(0, 3)$. A tabela abaixo mostra os resultados obtidos da aplicação do método da falsa posição com intervalo inicial $(a^{(1)}, b^{(1)}) = (2, 3)$. Aqui, o método foi iterado até a convergência com cinco dígitos significativos.

k	$a^{(k)}$	$b^{(k)}$	$x^{(k)}$	s
0	2.0000	3.0000	2.2455	1
1	2.2455	3.0000	2.3240	1
2	2.3240	3.0000	2.3470	1
3	2.3470	3.0000	2.3536	1
4	2.3536	3.0000	2.3555	1
5	2.3555	3.0000	2.3560	1
6	2.3560	3.0000	2.3561	1
7	2.3561	3.0000	2.3562	1
8	2.3562	3.0000	2.3562	1
9	2.3562	3.0000	2.3562	1

$s := f'(a^{(k)}) \cdot f'(x^{(k)})$

```
1 import numpy as np
2
3 f = lambda x: np.sin(x+np.pi/4)**2 \
4   - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
5   + 3*np.pi**3/64
6
7 a = 2.
8 b = 3.
9 for k in range(10):
10   x = a - (b-a)/(f(b)-f(a))*f(a)
```

```
11    print(f"{k+1}: {x:.4f}")
12
13    s = np.sign(f(a)*f(x))
14    if (s == -1):
15      b = x
16    elif (s == 1):
17      a = x
18    else:
19      break
```

Observação 2.2.1. (Ordem de Convergência.) O método da falsa posição é **globalmente convergente** e tem **ordem de convergência linear** [7, Seção 8.3].

2.2.1 Exercícios

E.2.2.1. Use o método da falsa posição para aproximar um zero de

$$f(x) = x^3 \operatorname{sen}(x) - \cos(x) \tag{2.42}$$

aplicando, como intervalo inicial $(a^{(1)}, b^{(1)}) = (0.5, 1)$ e aproximação inicial

$$x^{(0} = a^{(1)} - \frac{b^{(1)} - a^{(1)}}{f(b^{(1)}) - f(a^{(1)})} f(a^{(1)}). \tag{2.43}$$

Faça, então, 4 iterações deste método de forma a obter a aproximação $x^{(4)}$ e forneça-a com 7 dígitos significativos por arredondamento.

E.2.2.2. Use o método da falsa posição para computar a(s) solução(ões) das seguintes equações com precisão de 8 dígitos significativos.

a) $x = 2^{-x}$ para $0 \leq x \leq 2$.

b) $e^{-x^2} = 3x - x^2$ para $-1 \leq x \leq 4$.

E.2.2.3. Use o método da falsa posição para encontrar uma aproximação com precisão de 4 dígitos significativos do zero de

$$\begin{aligned} f(x) = (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 \\ - 1.154x + 0.332929 \end{aligned} \tag{2.44}$$

no intervalo $[-1, 0]$.

E.2.2.4. Use o método da falsa posição para encontrar uma aproximação com precisão de 10^{-4} do zero de

$$f(x) = (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 - 1.154x + 0.332929 \tag{2.45}$$

no intervalo $(0.55, 0.65)$. Forneça a aproximação computada com 7 dígitos significativos por arredondamento.

E.2.2.5. Aplique o método da falsa posição para encontrar o ponto crítico[16] de

$$f(x) = (1 - x^2)e^{-x^2} \tag{2.46}$$

no intervalo $(0, 2)$. Obtenha o resultado com precisão de 5 dígitos significativos por arredondamento.

Respostas

E.2.2.1. 9.158079×10^{-1}

E.2.2.2. a) 6.4118574×10^{-1}; b) 3.3536470×10^{-1}; 2.9999589

E.2.2.3. -7.861×10^{-1}

E.2.2.4. 5.770508×10^{-1}

2.3 Iteração de Ponto Fixo

Um **ponto fixo de uma função** g é um ponto x tal que

$$g(x) = x. \tag{2.47}$$

Geometricamente, pontos fixos são interseções do gráfico da g com a reta $y = x$, veja a Figura 2.4.

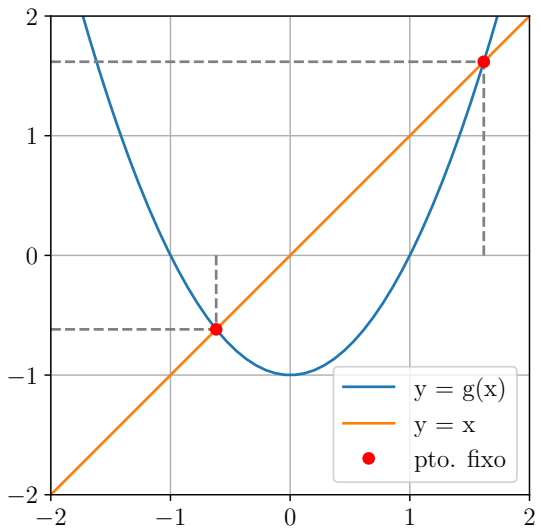

Figura 2.4: Exemplos de pontos fixos.

Observamos que toda equação de uma incógnita pode ser reescrita de forma equivalente como um problema de ponto fixo.

Exemplo 2.3.1. Consideremos o problema de resolver

$$\begin{aligned}\operatorname{sen}^2\left(x+\frac{\pi}{4}\right) &= x^3 - \frac{\pi}{4}x^2 \\ &\quad - \frac{5\pi^2}{16}x - \frac{3\pi^3}{64}.\end{aligned} \tag{2.48}$$

Podemos reescrevê-la como o problema de se obter os zeros da seguinte função

$$\begin{aligned}f(x) &= \operatorname{sen}^2\left(x+\frac{\pi}{4}\right) - x^3 \\ &\quad + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}.\end{aligned} \tag{2.49}$$

Por sua vez, este problema é equivalente aos seguintes problemas de ponto fixo (entre outros):

a)

$$g_1(x) = \frac{16}{5\pi^2}\left[-\operatorname{sen}^2\left(x+\frac{\pi}{4}\right) + x^3 - \frac{\pi}{4}x^2 - \frac{3\pi^3}{64}\right] = x. \tag{2.50}$$

b)

$$g_2(x) = \left[\operatorname{sen}^2\left(x+\frac{\pi}{4}\right) + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}\right]^{\frac{1}{3}} = x \tag{2.51}$$

Na Figura 2.5 podemos observar que os zeros da f (a saber, $x_1 = 3\pi/4 \approx 2.3562$ e $x_2 = x_3 = -\pi/4 \approx -0.78540$) coincidem com os pontos fixos das funções g_1 e g_2.

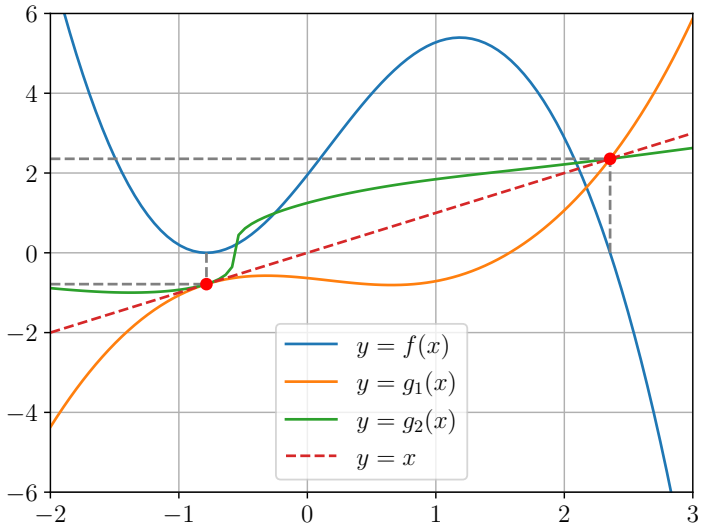

Figura 2.5: Esboço da função f, g_1 e g_2 do Exemplo 2.3.1.

Em muitos casos, é possível obter aproximações de um ponto fixo de uma dada função g pela chamada **iteração de ponto fixo**:

$$x^{(0)} = \text{aprox. inicial}, \tag{2.52}$$
$$x^{(k+1)} = g(x^{(k)}), \tag{2.53}$$

com $k = 0, 1, 2, \ldots$.

Exemplo 2.3.2. Vamos estudar as seguintes iterações de ponto fixo com as funções $g1$ e g_2 consideradas no Exemplo 2.3.1.

a) Função g_1 com $x^{(0)} = 0.7$.

$$x^{(0)} = -0.70000, \tag{2.54}$$
$$x^{(1)} = g_1\left(x^{(0)}\right) \tag{2.55}$$
$$= -0.70959, \tag{2.56}$$
$$x^{(2)} = g_1\left(x^{(1)}\right) \tag{2.57}$$
$$= -0.71716, \tag{2.58}$$
$$\vdots$$
$$x^{(99)} = g_1\left(x^{(98)}\right) \tag{2.59}$$
$$= -0.77862, \tag{2.60}$$
$$\vdots$$
$$x^{(999)} = g_1\left(x^{(998)}\right) \tag{2.61}$$
$$= -0.78466, \tag{2.62}$$
$$\vdots$$
$$x^{(19999)} = g_1\left(x^{(19998)}\right) \tag{2.63}$$
$$= -0.78536. \tag{2.64}$$

Neste caso as iterações de ponto fixo convergem (lentamente) para o ponto fixo $x = -\pi/4 \approx -0.78540$.

b) Função $g1$ com $x^{(0)} = 2.5$.

Este valor inicial está próximo do ponto fixo $x = 3\pi/4 \approx 2.3562$, entretanto as iterações de ponto fixo divergem:

$$x^{(0)} = 2.50000, \tag{2.65}$$
$$x^{(1)} = g_1\left(x^{(0)}\right) \tag{2.66}$$
$$= 2.9966, \tag{2.67}$$
$$x^{(2)} = 5.8509, \tag{2.68}$$
$$\vdots$$
$$x^{(7)} = 4.8921 \times 10^{121}. \tag{2.69}$$

c) Função g_2 com $x^{(0)} = 2.5$. Neste caso, as iterações de ponto fixo convergem (rapidamente) para o ponto fixo próximo:

$$x^{(0)} = 2.50000, \tag{2.70}$$
$$x^{(1)} = g_2\left(x^{(0)}\right) \tag{2.71}$$
$$= 2.4155, \tag{2.72}$$
$$x^{(2)} = 2.3805, \tag{2.73}$$
$$\vdots \tag{2.74}$$
$$x^{(9)} = 2.3562. \tag{2.75}$$

Este último exemplo mostra que a iteração do ponto fixo nem sempre é convergente. Antes de vermos condições suficientes para a convergência, vejamos sua interpretação geométrica.

2.3.1 Interpretação Geométrica

A Figura 2.6 apresenta o caso de uma iteração de ponto fixo convergente. As iterações iniciam-se no ponto $x^{(0)}$ e seguem para $x^{(1)} = g(x^{(0)})$ e $x^{(2)} = g(x^{(1)})$.

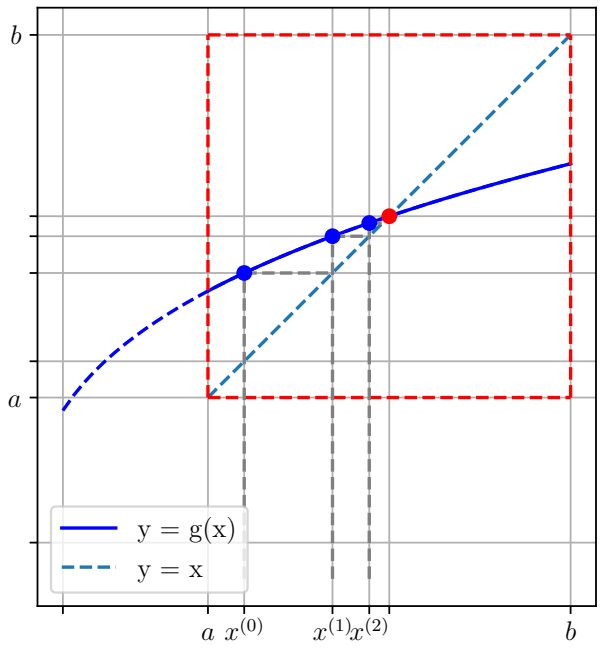

Figura 2.6: Interpretação geométrica da iteração de ponto fixo.

2.3.2 Análise Numérica

O seguinte teorema nos fornece condições suficientes para a convergência das iterações de ponto fixo.

Teorema 2.3.1. (Teorema do Ponto Fixo.) Seja g função continuamente diferenciável satisfazendo ambas as seguintes condições

a) $g([a,b]) \subset [a,b]$,

b) $|g'(x)| < K < 1$ para todo $x \in [a,b]$.

Então, g tem um único ponto fixo $x^* \in [a,b]$ e as iterações

$$x^{(k+1)} = g\left(x^{(k)}\right), k = 0, 1, 2, \ldots, \tag{2.76}$$

convergem para x^*, para qualquer escolha de $x^{(0)} \in [a, b]$.

Demonstração. Da hipótese b), temos que g é uma contração com

$$|g(x) - g(y)| < K \cdot |x - y|, \tag{2.77}$$

para quaisquer $x, y \in [a, b]$. Com isso, da hipótese a) e tomando $x^{(0)} \in [a, b]$, temos

$$\begin{aligned} \left|x^{(k+1)} - x^{(k)}\right| &= \left|g(x^{(k)}) - g(x^{(k-1)})\right| && (2.78)\\ &\leq K \left|x^{(k)} - x^{(k-1)}\right| && (2.79)\\ &\vdots \\ &\leq K^{k-1} \left|x^{(2)} - x^{(1)}\right|, && (2.80) \end{aligned}$$

para $k = 1, 2, \ldots$. Como $K < 1$, temos $\left|x^{(k+1)} - x^{(k)}\right| \to 0$ quando $k \to \infty$ e, portanto, $x^{(k)}$ converge para algum $x^* \in [a, b]$.

De fato, x^* é ponto fixo de g, pois da continuidade da g, temos

$$\begin{aligned} x^* &= \lim_{k\to\infty} x^{(k+1)} && (2.81)\\ &= \lim_{k\to\infty} g(x^{(k)}) = g(x^*). && (2.82) \end{aligned}$$

Por fim, x^* é único, pois assumindo a existência de outro ponto fixo $x^{**} \neq x^*$ teríamos

$$\begin{aligned} |x^* - x^{**}| &= |g(x^*) - g(x^{**})| && (2.83)\\ &< K|x^* - x^{**}| && (2.84)\\ &< |x^* - x^{**}|. && (2.85) \end{aligned}$$

□

Observação 2.3.1. (Ordem de Convergência.) A iteração de ponto fixo tem ordem de convergência linear

$$|x^{(k+1)} - x^{(k)}| < K|x^{(k)} - x^{(k-1)}|^{\mathbf{1}}, \tag{2.86}$$

onde $0 < K < 1$ é a constante dada na hipótese b) do Teorema do Ponto Fixo. Além disso, isso mostra que quanto menor o valor da constante K, mais rápida é a convergência das iterações de ponto fixo.

2.3.3 Zero de Funções

Dado um problema de encontrar um zero de uma função f (i.e., resolver $f(x) = 0$), podemos construir uma função g com ponto fixo no zero de f e aplicarmos a iteração de ponto fixo para computá-lo. Para tanto, observamos que

$$
\begin{aligned}
& f(x) = 0 && (2.87)\\
& \quad \Leftrightarrow && (2.88)\\
& \underbrace{x - \alpha f(x)}_{=:g(x)} = x, && (2.89)
\end{aligned}
$$

com $\alpha \in \mathbb{R}$ escolhido de forma a satisfazer as hipóteses do Teorema do Ponto Fixo (Teorema 2.3.1).

Exemplo 2.3.3. Retornamos ao problema de encontrar o zero da função

$$
\begin{aligned}
f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\
+ \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}.
\end{aligned} \tag{2.90}
$$

no intervalo $[2, 3]$. Para construir uma função g para a iteração de ponto fixo neste intervalo, podemos tomar

$$
g(x) = x - \alpha f(x), \tag{2.91}
$$

com $\alpha = -0.1$. A Figura 2.7 mostra esboços dos gráficos de g e $|g'|$ no intervalos $[2, 3]$ e podemos observar que esta escolha de α faz com que a g satisfaça o Teorema do Ponto Fixo.

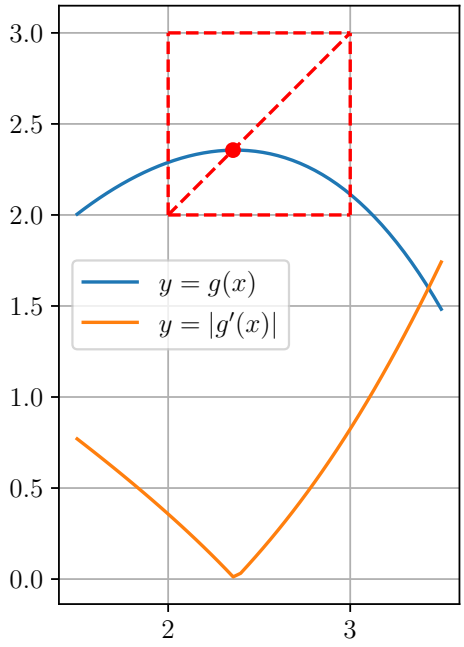

Figura 2.7: Esboço dos gráficos de g e $|g'|$ discutidas no Exemplo 2.3.3.

Então, fazendo as iterações de ponto fixo com aproximação inicial $x^{(0)} = 2.6$, obtemos os resultados apresentados na Tabela 2.3.

Tabela 2.3: Resultados referentes ao Exemplo 2.3.3.

k	$x^{(k)}$	$\lvert x^{(k)} - x^{(k-1)}\rvert$
0	2.6000	-x-
1	2.3264	2.7e−1
2	2.3553	2.9e−2
3	2.3562	8.4e−4
4	2.3562	1.1e−5

```
1 import numpy as np
2
```

```
# fun obj
f = lambda x: np.sin(x+np.pi/4)**2 \
  - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
  + 3*np.pi**3/64

# param
alpha = -0.1
# fun pto fixo
g = lambda x: x - alpha*f(x)

# aprox inicial
x0 = 2.6
print(f'\n{1}: {x0:.4f}')
for k in range(4):
  x = g(x0)
  nd = np.fabs(x-x0)
  print(f'{k+1}: {x:.4f}, {nd:.1e}')
  x0 = x
```

2.3.4 Exercícios

E.2.3.1. Forneça o(s) ponto(s) fixo(s) de

$$g(x) = x^2 e^{-x^2}. \tag{2.92}$$

E.2.3.2. Verifique se a iteração de ponto fixo é convergente para as seguintes funções e aproximações iniciais:

a) $g_1(x) = \cos(x)$, $x^{(1)} = 0.5$

b) $g_2(x) = x^2$, $x^{(1)} = 1.01$

Justifique sua resposta.

E.2.3.3. Considere o problema de computar uma aproximação do zero de $f(x) = x - \cos(x)$. Resolva-o aplicando a iteração de ponto fixo para a função

auxiliar

$$g(x) = x - \alpha f(x), \tag{2.93}$$

restrita ao intervalo $[a, b] = [0.5, 1]$ com aproximação inicial $x^{(0)} = (a + b)/2$. Escolha o melhor valor de α entre os seguintes:

1. $\alpha = 1$
2. $\alpha = 0.5$
3. $\alpha = -0.5$
4. $\alpha = 0.6$

Então, compute uma aproximação do zero de f com 5 dígitos significativos de precisão.

E.2.3.4. Seja

$$\begin{aligned} f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.94}$$

a) Aplique a iteração de ponto fixo na função auxiliar

$$g(x) = x - \alpha f(x) \tag{2.95}$$

para algum α adequado, de forma que aproximação inicial $x^{(0)} = -0.5$ leve a iterações de ponto fixo que convirjam para $x^* = -\pi/4$, zero de multiplicidade par de f.

b) Mostre que $g'(x^*) = 1$ para qualquer valor de α. Por que isso explica a lenta convergência observada no item a)?

c) Alternativamente, verifique que a abordagem da iteração de ponto fixo converge muito mais rápido para x^* se aplicada à derivada de f, i.e. aplicando a iteração à função auxiliar

$$h(x) = x - \alpha f'(x), \tag{2.96}$$

para um valor de α adequado.

E.2.3.5. Use o Método da Iteração de Ponto Fixo para aproximar um zero de

$$f(x) = x^3 \operatorname{sen}(x) - \cos(x) \tag{2.97}$$

no intervalo inicial $[0.5, 1]$.

E.2.3.6. Use o Método da Iteração de Ponto Fixo para computar a(s) solução(ões) das seguintes equações com precisão de 8 dígitos significativos.

a) $x = 2^{-x}$ para $0 \leq x \leq 2$.

b) $e^{-x^2} = 3x - x^2$ para $-1 \leq x \leq 4$.

E.2.3.7. Use o Método de Iteração de Ponto Fixo para encontrar uma aproximação com precisão de 4 dígitos significativos do zero de

$$\begin{aligned} f(x) = &(-x^2 + 1.154x - 0.332929)\cos(x) + x^2 \\ &- 1.154x + 0.332929 \end{aligned} \tag{2.98}$$

no intervalo $[-1, 0]$.

E.2.3.8. Use o Método de Iteração de Ponto Fixo para encontrar uma aproximação com precisão de 10^{-4} do zero de

$$\begin{aligned} f(x) = &(-x^2 + 1.154x - 0.332929)\cos(x) + x^2 \\ &- 1.154x + 0.332929 \end{aligned} \tag{2.99}$$

no intervalo $(0.55, 0.65)$. Forneça a aproximação computada com 7 dígitos significativos por arredondamento.

E.2.3.9. Use o Método da Iteração de Ponto Fixo para encontrar o ponto crítico[17] de

$$f(x) = (1 - x^2)e^{-x^2} \tag{2.100}$$

no intervalo $(0, 2)$. Obtenha o resultado com precisão de 5 dígitos significativos por arredondamento.

Respostas

E.2.3.1. $x = 0$

E.2.3.2. a) Convergente; b) Divergente.

E.2.3.3. $\alpha = 0.6$; 7.3909×10^{-1}

E.2.3.4. a) $\alpha = 0.25$; b) Pois, não há α que satisfaz o Teorema do Ponto Fixo.; c) $\alpha = 0.1$

E.2.3.6. a) 6.4118574×10^{-1}; b) 3.3536470×10^{-1}; 2.9999589

E.2.3.7. -7.861×10^{-1}

E.2.3.8. 5.770508×10^{-1}

2.4 Método de Steffensen

O método de Steffensen[18] é uma aplicação do método de aceleração de convergência Δ^2 de Aitken[19] à iteração de ponto fixo.

2.4.1 Acelerador Δ^2 de Aitken

Seja dada uma sequência $(x^{(k)})_{k=1}^{\infty}$ monotonicamente convergente para x^*. Assumimos que k seja suficientemente grande tal que

$$\frac{x^{(k+1)} - x^*}{x^{(k)} - x^*} \approx \frac{x^{(k+2)} - x^*}{x^{(k+1)} - x^*}. \tag{2.101}$$

Então, isolando x^* obtemos

$$x^* \approx \frac{x^{(k)}x^{(k+2)} - (x^{(k+1)})^2}{x^{(k)} - 2x^{(k+1)} + x^{(k+2)}}. \tag{2.102}$$

Ainda, somando e subtraindo $(x^{(k)})^2$ e $2x^{(k)}x^{(k+1)}$ no numerador acima e rearranjando os termos, obtemos

$$x^* \approx x^{(k)} - \frac{(x^{(k+1)} - x^{(k)})^2}{x^{(k+2)} - 2x^{(k+1)} + x^{(k)}}. \tag{2.103}$$

O observado acima, nos motiva a introduzir o acelerador Δ^2 de Aitken

$$\Delta^2\{x^{(k)}, x^{(k+1)}, x^{(k+2)}\} := x^{(k)} - \frac{(x^{(k+1)} - x^{(k)})^2}{x^{(k+2)} - 2x^{(k+1)} + x^{(k)}}. \tag{2.104}$$

Exemplo 2.4.1. Consideremos o problema de encontrar o zero da função

$$\begin{aligned} f(x) &= \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ &\quad + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.105}$$

no intervalo $[2, 3]$. Para tanto, podemos aplicar a iteração de ponto fixo dada por

$$\begin{aligned} x^{(k+1)} &= g(x^{(k)}) \\ &:= x^{(k)} - \alpha f(x^{(k)}), \quad k = 1, 2, \ldots, \end{aligned} \tag{2.106}$$

com $\alpha = -0.05$ e $x^{(0)} = 2.6$. Na Tabela 2.4 temos os valores das iteradas $x^{(k)}$ e das correções $\Delta^2 = \Delta^2\{x^{(k)}, x^{(k+1)}, x^{(k+2)}\}$ de Aitken. Neste caso, a aceleração de convergência é notável.

Tabela 2.4: Resultados referentes ao Exemplo 2.4.1.

k	$x^{(k)}$	Δ^2
0	2.6000	-x-
1	2.4632	-x-
2	2.4073	2.3687
3	2.3814	2.3590
4	2.3688	2.3569
5	2.3625	2.3564
6	2.3594	2.3562
7	2.3578	2.3562

```python
import numpy as np

# fun obj
f = lambda x: np.sin(x+np.pi/4)**2 \
  - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
  + 3*np.pi**3/64

# fun pto fixo
alpha = -0.05
g = lambda x: x - alpha*f(x)

x0 = 2.6
print(f'\n1: {x0:.4f}')
for k in range(7):
  x1 = g(x0)
  x2 = g(x1)
  x = x0 - (x1-x0)**2/(x2-2*x1+x0)
  print(f'\n{k+2}: {x1:.4f}, {x:.4f}')
  x0 = x1
```

2.4.2 Análise Numérica

Definição 2.4.1. (Diferença Progressiva.) Para uma sequência $\left(x^{(k)}\right)_{k=1}^{\infty}$, $\Delta x^{(k)}$ denota o **operador de diferença progressiva** e é definido por

$$\Delta x^{(k)} := x^{(k+1)} - x^{(k)} \tag{2.107}$$

Potências maiores do operador são definidas recursivamente por

$$\Delta^n x^{(k)} = \Delta\left(\Delta^{n-1} x^{(k)}\right), \quad n \geq 2. \tag{2.108}$$

Da definição acima, temos que

$$\begin{aligned}
\Delta^2 x^{(k)} &:= \Delta\left(\delta x^{(k)}\right) && (2.109)\\
&= \Delta\left(x^{(k+1)} - x^{(k)}\right) && (2.110)\\
&= \Delta x^{(k+1)} - \Delta x^{(k)} && (2.111)\\
&= \left(x^{(k+2)} - x^{(k+1)}\right) - \left(x^{(k+1)} - x^{(k)}\right) && (2.112)
\end{aligned}$$

$$= x^{(k+2)} - 2x^{(k+1)} + x^{(k)} \tag{2.113}$$

Com isso, temos que o acelerador Δ^2 de Aitken (2.104) pode ser reescrito como

$$\Delta^2 \left\{x^{(k)}, x^{(k+1)}, x^{(k+2)}\right\} := x^{(k)} - \frac{\left(\Delta x^{(k)}\right)^2}{\Delta^2 x^{(k)}}. \tag{2.114}$$

Teorema 2.4.1. *Seja* $\left(x^{(k)}\right)_{k=1}^{\infty}$ *uma sequência linearmente convergente para* x^* *e*

$$\lim_{k\to\infty} \frac{x^{(k+1)} - x^*}{x^{(k)} - x^*} < 1. \tag{2.115}$$

Então, a sequência Δ^2 *de Aitken* $\left(\hat{x}^{(k)}\right)_{k=1}^{\infty}$, *com*

$$\hat{x}^{(k)} := \Delta^2 \left\{x^{(k)}, x^{(k+1)}, x^{(k+2)}\right\}, \tag{2.116}$$

converge para x^* *mais rápido que* $\left(x^{(k)}\right)$ *no sentido de que*

$$\lim_{k\to\infty} \frac{\hat{x}^{(k)} - x^*}{x^{(k)} - x^*} = 0. \tag{2.117}$$

Demonstração. Em construção ... □

2.4.3 Algoritmo de Steffensen

O método de Steffensen consiste em aplicar o acelerador Δ^2 de Aitken à iteração de ponto fixo. Mais especificamente, sejam uma aproximação inicial $x^{(0)}$ e uma iteração de ponto fixo

$$x^{(k+1)} = g(x^{(k)}), \quad k = 0, 1, 2, \ldots. \tag{2.118}$$

O algoritmo de Steffensen consiste em:

1. $x \leftarrow x^{(0)}$.
2. Para $k = 0, 1, 2, \ldots, N-1$:

(a) $x_1 \leftarrow g(x)$.

(b) $x_2 \leftarrow g(x_1)$.

(c) $x^{(k+1)} \leftarrow \Delta^2\{x, x_1, x_2\}$.

(d) $x \leftarrow x^{(k+1)}$.

Exemplo 2.4.2. Retornamos ao exemplo anterior (Exemplo 2.4.1. Na Tabela 2.5 temos os valores das iteradas de Steffensen $x^{(k)}$ e do indicador de convergência $|x^{(k)} - x^{(k-1)}|$.

Tabela 2.5: Resultados referentes ao Exemplo 2.4.2.

k	$x^{(k)}$	$\lvert x^{(k)} - x^{(k-1)}\rvert$
0	2.6000	-x-
1	2.3687	2.3e−1
2	2.3562	1.2e−2
3	2.3562	4.2e−5

```
1 import numpy as np
2
3 # fun obj
4 f = lambda x: np.sin(x+np.pi/4)**2 \
5   - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
6   + 3*np.pi**3/64
7
8 # fun pto fixo
9 alpha = -0.05
10 g = lambda x: x - alpha*f(x)
11
12 x0 = 2.6
13 print(f'\n1: {x0:.4f}')
14 for k in range(3):
15   x1 = g(x0)
16   x2 = g(x1)
17   x = x0 - (x1-x0)**2/(x2-2*x1+x0)
18   nd = np.fabs(x-x0)
19   print(f'\n{k+2}: {x:.4f}, {nd:.1e}')
20   x0 = x
```

Exercícios

E.2.4.1. Use o método de Steffensen para obter uma aproximação do zero de $f(x) = x^3 \operatorname{sen}(x) - \cos(x)$ no intervalo $[0.5, 1]$ com precisão de 10^{-6}.

E.2.4.2. Use o Método da Iteração de Ponto Fixo para aproximar um zero de

$$f(x) = x^3 \operatorname{sen}(x) - \cos(x) \tag{2.119}$$

no intervalo inicial $[0.5, 1]$.

E.2.4.3. Use o Método de Steffensen para computar a(s) solução(ões) das seguintes equações com precisão de 8 dígitos significativos.

a) $x = 2^{-x}$ para $0 \leq x \leq 2$.

b) $e^{-x^2} = 3x - x^2$ para $-1 \leq x \leq 4$.

E.2.4.4. Use o Método de Steffensen para encontrar uma aproximação com precisão de 4 dígitos significativos do zero de

$$f(x) = (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 - 1.154x + 0.332929 \tag{2.120}$$

no intervalo $[-1, 0]$.

E.2.4.5. Use o Método de Steffensen para encontrar uma aproximação com precisão de 10^{-4} do zero de

$$\begin{aligned} f(x) = {} & (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 \\ & - 1.154x + 0.332929 \end{aligned} \tag{2.121}$$

no intervalo $(0.55, 0.65)$. Forneça a aproximação computada com 7 dígitos significativos por arredondamento.

E.2.4.6. Use o Método de Steffensen para encontrar o ponto crítico[20] de

$$f(x) = (1 - x^2)e^{-x^2} \tag{2.122}$$

no intervalo $(0, 2)$. Obtenha o resultado com precisão de 5 dígitos significativos por arredondamento.

Respostas

E.2.4.1. 9.15811×10^{-1}

E.2.4.3. a) 6.4118574×10^{-1}; b) 3.3536470×10^{-1}; 2.9999589

E.2.4.4. -7.861×10^{-1}

E.2.4.5. 5.770508×10^{-1}

2.5 Método de Newton

Seja x^* um zero de uma dada função f, i.e.

$$f(x^*) = 0. \tag{2.123}$$

A expansão em polinômio de Taylor[21] de f em um ponto $\tilde{x}$ dado, é

$$f(x^*) = f(\tilde{x}) + f'(\tilde{x})(x^* - \tilde{x}) + O\left(|x^* - \tilde{x}|^2\right). \tag{2.124}$$

Como $f(x^*) = 0$, temos

$$0 = f(\tilde{x}) + f'(\tilde{x})(x^* - \tilde{x}) + O\left(|x^* - \tilde{x}|^2\right) \tag{2.125}$$

$$x^* = \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})} + O\left(|x^* - \tilde{x}|^2\right) \tag{2.126}$$

Esta última expressão nos indica que dada uma aproximação $\tilde{x}$ do zero de f a expressão

$$\tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}, \tag{2.127}$$

aproxima x^* com um erro da ordem de $|x^* - \tilde{x}|^2$.

Estas observações nos levam a **iteração de Newton**[22]

$$x^{(0)} = \text{aprox. inicial}, \tag{2.128}$$

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \tag{2.129}$$

com $k = 0, 1, 2, \ldots$.

Exemplo 2.5.1. Consideramos o problema de encontrar o zero da função

$$\begin{aligned} f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.130}$$

no intervalo $[2, 3]$ (Consulte a Figura 2.8).

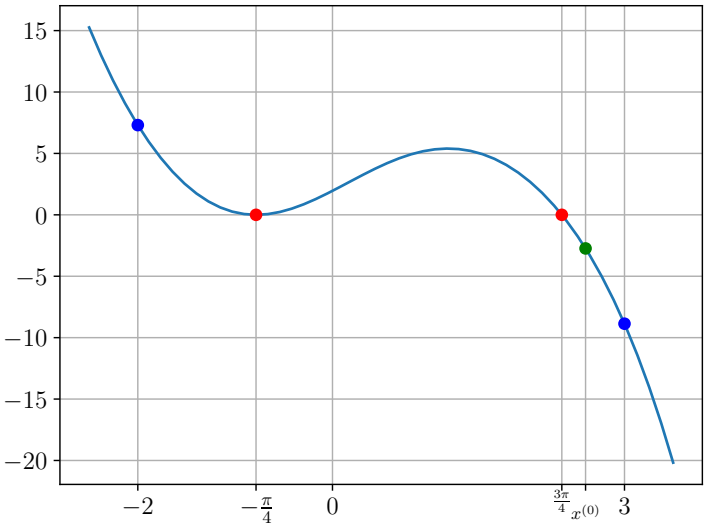

Figura 2.8: Esboço da função f do Exemplo 2.5.1.

Fazendo as iterações de Newton com aproximação inicial $x^{(0)} = 2.6$, obtemos os resultados apresentados na Tabela 2.6.

Tabela 2.6: Resultados referentes ao Exemplo 2.5.1.

k	$x^{(k)}$	$\lvert x^{(k)} - x^{(k-1)}\rvert$
0	2.6000	-x-
1	2.3836	2.2e−1
2	2.3566	2.7e−2
3	2.3562	3.9e−4
4	2.3562	8.3e−8

```
import numpy as np

# fun obj
f = lambda x: np.sin(x+np.pi/4)**2 \
  - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
  + 3*np.pi**3/64

fl = lambda x: 2*np.sin(x+np.pi/4)*np.cos(x+np.pi
/4) \
  -3*x**2 + np.pi/2*x + 5*np.pi**2/16

# aprox. inicial
x0 = 2.6
print(f'0: {x0:.4e}')

# iterações
for k in range(4):
  x = x0 - f(x0)/fl(x0)
  print(f'{k+1}: {x:.4e}')
  x0 = x
```

Observação 2.5.1. O método de Newton é uma iteração de ponto fixo ótima. Do Teorema do Ponto Fixo (Teorema 2.3.1), a iteração

$$x^{k+1} = g(x^{(k)}) \tag{2.131}$$

$$:= x^{(k)} - \alpha f(x) \tag{2.132}$$

tem taxa de convergência[23]

$$|x^{(k+1)} - x^{(k)}| \leq K|x^{(k)} - x^{(k-1)}|, \tag{2.133}$$

com K tal que $|g'(x)| = |1 - \alpha f'(x)| < K < 1$. Isto nos indica que a melhor escolha para α é

$$\alpha = \frac{1}{f'(x)}, \tag{2.134}$$

de forma que (2.132) coincide com a iteração de (2.129).

2.5.1 Interpretação Geométrica

Dadas uma aproximação $x^{(k)}$ de um zero de uma função f, a iteração de Newton fornece uma nova aproximação $x^{(k+1)}$ com

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}. \tag{2.135}$$

Subtraindo $x^{(k+1)}$ e multiplicando por $-f'(x^{(k)})$, obtemos

$$0 = f'(x^{(k)})(x^{(k+1)} - x^{(k)}) + f(x^{(k)}), \tag{2.136}$$

Observemos que o lado direito desta última equação corresponde a expressão da reta tangente ao gráfico de f pelo ponto $(x^{(k)}, f(x^{(k)}))$, avaliada em $x^{(k+1)}$. Mais precisamente, a equação desta reta tangente é

$$y = f'(x^{(k)})(x - x^{(k)}) + f(x^{(k)}) \tag{2.137}$$

e a equação (2.136) nos informa que em $x = x^{(k+1)}$ a reta tangente cruza o eixo x.

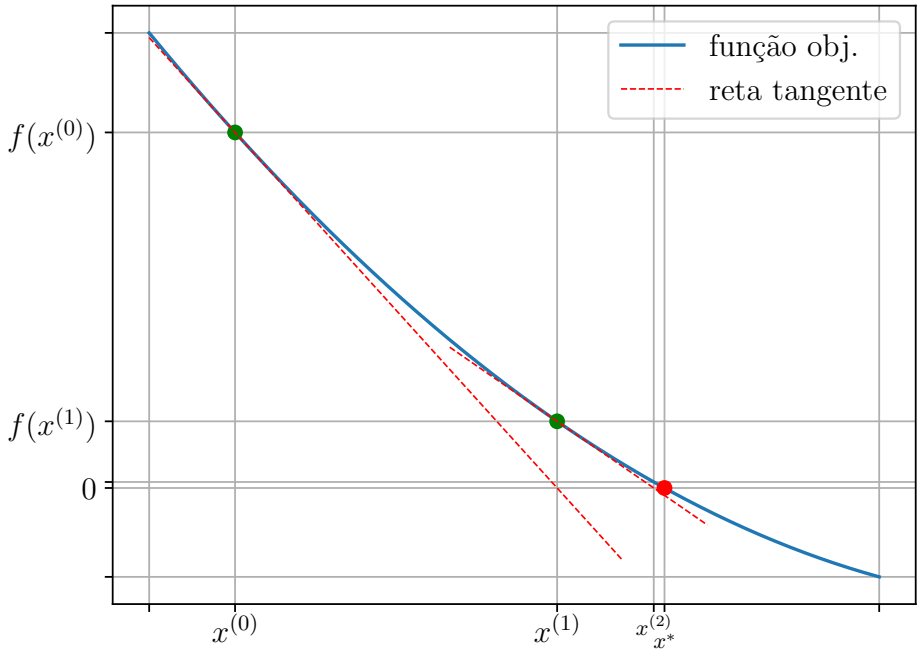

Figura 2.9: Interpretação geométrica das iterações de Newton.

Destas observações, concluímos que a iterada $x^{(k+1)}$ do método de Newton corresponde ao ponto de interseção da reta tangente ao gráfico da f pelo ponto $(x^k, f(x^k))$ com o eixo das abscissas[24]. Consulte a Figura 2.9.

Exemplo 2.5.2. Consideremos que o método de Newton seja usado para aproximarmos o zero de

$$f(x) = (x-1)e^{-x^2}. \tag{2.138}$$

Observemos que esta função tem $x = 1$ como seu único zero. Agora, se escolhermos $x^{(0)} = 0.5$ as iterações de Newton convergem para este zero, mas, se escolhermos $\tilde{x}^{(0)} = 1.5$ não (consulte a Tabela 2.7).

Tabela 2.7: Resultados referentes ao Exemplo 2.5.2

k	$x^{(k)}$	$\tilde{x}^{(k)}$
0	5.0000e−1	1.5000e+0
1	8.3333e−1	2.5000e+0
2	9.6377e−1	2.7308e+0
3	9.9763e−1	2.9355e+0
4	9.9999e−1	3.1223e+0
5	1.0000e+0	3.2955e+0

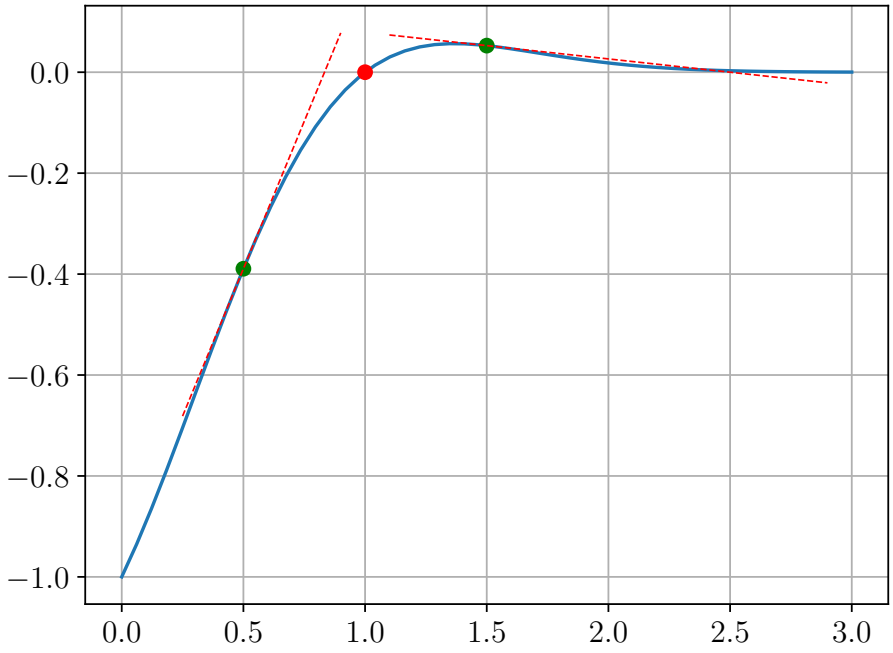

Figura 2.10: Escolha da aproximação inicial para o método de Newton.

Embora ambas aproximações iniciais estão a mesma distância da solução $x = 1$, quando tomamos $x^{(0)} = 1.5$ as iterações irão divergir, como podemos observar da interpretação geométrica dada na Figura 2.10.

2.5.2 Análise de Convergência

Seja x^* o zero de uma dada função f duas vezes continuamente diferenciável com $f'(x) \neq 0$ para todo $x \in [x^* - \varepsilon_0, x^* + \varepsilon_0]$ para algum $\varepsilon_0 > 0$. Seja, também, $(x^{(k)})_{k=0}^{\infty}$ a sequência das iteradas de Newton

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k = 0, 1, 2, \ldots, \tag{2.139}$$

com aproximação inicial $x^{(0)} \in (x^* - \varepsilon_0, x^* + \varepsilon_0)$. Então, do polinômio de Taylor de grau 1 de f em torno de $x^{(0)}$, temos

$$f(x^*) = f(x^{(0)}) + f'(x^{(0)})(x^* - x^{(0)}) + \frac{f''(\xi^{(0)})}{2!}(x^* - x^{(0)})^2, \tag{2.140}$$

onde $\xi^{(0)}$ está entre $x^{(0)}$ e $\xi^{(0)}$. Daí, rearranjamos os termos e notamos que $f(x^*) = 0$ para obtermos

$$x^* - \left[x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})}\right] = \frac{f''(\xi^{(0)})}{2f'(x^{(0)})}(x^* - x^{(0)})^2. \tag{2.141}$$

Então, da iteração de Newton (2.139), temos

$$x^* - x^{(1)} = \frac{f''(\xi^{(0)})}{2f'(x^{(0)})}(x^* - x^{(0)})^2 \tag{2.142}$$

Logo,

$$|x^* - x^{(1)}| \leq C|x^* - x^{(0)}|^2, \tag{2.143}$$

com

$$C = \sup_{x,y \in [x^* - \varepsilon, x^* + \varepsilon]} \left| \frac{f''(x)}{2f'(y)} \right|. \tag{2.144}$$

Segue, então, que se $x^{(0)} \in (x^* - \varepsilon, x^* + \varepsilon)$ para algum $\varepsilon > 0$ tal que

$$C|x^* - x^{(0)}|^2 < 1, \quad \forall x \in (x^* - \varepsilon, x^* + \varepsilon), \tag{2.145}$$

então $x^{(1)} \in (x^* - \varepsilon, x^* + \varepsilon)$.

Logo, por indução matemática[25], temos que o método de Newton tem ordem de convergência quadrática

$$|x^{(k+1)} - x^*| \leq C|x^{(k)} - x^*|^2, \tag{2.146}$$

para qualquer escolha de $x^{(0)}$ suficientemente próximo de x^*, i.e. $x^{(0)} \in (x^* - \varepsilon, x^* + \varepsilon)$.

Observação 2.5.2. O intervalo $(x^* - \varepsilon, x^* + \varepsilon)$ é chamado de **bacia de atração** do método de Newton.

Exemplo 2.5.3. Retornamos ao problema de encontrar o zero da função

$$\begin{aligned} f(x) = &\operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ &+ \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.147}$$

no intervalo $[2, 3]$. Este problema foi construído de forma que $x^* = 3\pi/4$ é um zero de f. Então, fazendo as iterações de Newton com aproximação inicial $x^{(0)} = 2.6$, obtemos os resultados apresentados na Tabela 2.8, os quais evidenciam a convergência quadrática das iterações computadas.

Tabela 2.8: Resultados referentes ao Exemplo 2.5.3.

k	$x^{(k)}$	$\lvert x^{(k)} - x^*\rvert$
0	2.6000	2.4e−01
1	2.3836	2.7e−02
2	2.3566	3.9e−04
3	2.3562	8.3e−08
4	2.3562	3.6e−15

2.5.3 Zeros Múltiplos

Na análise de convergência acima foi necessário assumir que $f'(x) \neq 0$ para todo x em uma vizinha do zero x^* da função f. Isto não é possível no caso de x^* ser um zero duplo pois, então, $f'(x^*) = 0$. Neste caso, podemos aplicar o método de Newton a $f'(x)$, a qual tem x^* como um zero simples.

Exemplo 2.5.4. Consideremos o problema de aproximar o zero da função

$$f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \tag{2.148}$$

no intervalo $[-1.0]$. Este problema foi construído de forma que $x^* = -\pi/4$ é um zero duplo de f. Então, aplicamos o método de Newton a

$$f'(x) = 2\operatorname{sen}\left(x + \frac{\pi}{4}\right)\cos\left(x + \frac{\pi}{4}\right) - 3x\check{s} + \frac{\pi}{2}x + \frac{5\pi^2}{16}. \tag{2.149}$$

Ou seja, as iterações de Newton são

$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}, \quad k = 0, 1, 2, \ldots, \tag{2.150}$$

sendo $x^{(1)}$ uma aproximação inicial. Na Tabela 2.9, temos os resultados obtidos da computação destas iterações com $x^{(1)} = -0.5$.

Tabela 2.9: Resultados referentes ao Exemplo **??**.

k	$x^{(k)}$	$\lvert x^{(k)} - x^*\rvert$
0	-0.5000	
1	-0.8341	3.3e-01
2	-0.7862	4.8e-02
3	-0.7854	7.9e-04
4	-0.7854	2.3e-07
5	-0.7854	1.9e-14

Observação 2.5.3. (Zeros múltiplos.) No caso de zeros de multiplicidade $m > 1$ de uma dada função f, podemos aplicar o método de Newton à derivada $m-1$ de f, o que requer o cálculo de m derivadas de f. Alternativamente, consideramos aplicar o método à função auxiliar

$$\mu(x) := \frac{f(x)}{f'(x)}. \tag{2.151}$$

De fato, se x^* é zero de multiplicidade $m \geq 1$ de f, então existe uma função g tal que $g(x^*) \neq 0$ e

$$f(x) = (x - x^*)^m \, g(x) \tag{2.152}$$

Com isso, temos

$$\begin{aligned} \mu(x) &= \frac{(x - x^*)^m \, g(x)}{m \, (x - x^*)^{m-1} \, g(x) + (x - x^*)^m \, g'(x)} \\ &= \frac{(x - x^*) \, g(x)}{mg(x) + (x - x^*) \, g'(x)} \end{aligned}$$

Como $g\,(x^*) \neq 0$, temos que

$$\frac{g(x^*)}{mg\,(x^*) + (x^* - x^*)\, g'\,(x^*)} = \frac{1}{m} \neq 0. \tag{2.153}$$

Ou seja, x^* é um zero simples de μ. A iteração do Método de Newton aplicado à μ fornece

$$x^{(k+1)} = x^{(k)} - \frac{\mu\left(x^{(k)}\right)}{\mu'\left(x^{k+1}\right)} \tag{2.154}$$

$$= x^{(k)} = \frac{\frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}}{\frac{\left[f'\left(x^{(k)}\right)\right]^2 - f\left(x^{(k)}\right) f''\left(x^{(k)}\right)}{\left[f'\left(x^{(k)}\right)\right]^2}} \tag{2.155}$$

Rearranjando os termos, obtemos a iteração modificada de Newton para zeros de multiplicidade maior que 1

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right) f'\left(x^{(k)}\right)}{\left[f'\left(x^{(k)}\right)\right]^2 - f\left(x^{(k)}\right) f''\left(x^{(k)}\right)}. \tag{2.156}$$

Para uma aplicação, consulte o exercício 2.5.3.

Exercícios

E.2.5.1. Use o método de Newton para obter uma aproximação do zero de $f(x) = x^3 \operatorname{sen}(x) - \cos(x)$ no intervalo $[0.5, 1]$ com precisão de 10^{-6}.

E.2.5.2. Use o método de Newton para obter uma aproximação do zero de

$$f(x) = (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 - 1.154x + 0.332929 \tag{2.157}$$

no intervalo $(0.55, 0.65)$ com precisão de 10^{-5}.

E.2.5.3. A função (2.148) tem um zero de multiplicidade par em $x = -\pi/4$. Assumindo a aproximação inicial $x^{(0)} = -0.5$, aplique a iteração modificada de Newton dada em (2.156) e compare os resultados com apresentados na Tabela 2.9.

E.2.5.4. Assumindo a aproximação inicial $x^{(0)} = 1$, aproxime o zero de

$$f(x) = e^x - x - 1 \tag{2.158}$$

usando:

a) a iteração de Newton para f.

b) a iteração de Newton para f'.

c) a iteração modificada de Newton[26] para f.

Qual a melhor abordagem? Justifique sua resposta.

E.2.5.5. Use o Método de Newton para obter a aproximação do zero de

$$p(x) = x^3 - 3\pi x^2 + 3\pi^2 x - \pi^3. \tag{2.159}$$

Análise Numérica

E.2.5.6. Complete a demonstração por indução matemática de que o método de Newton tem taxa de convergência quadrática.

Respostas

E.2.5.1. 9.15811×10^{-1}

E.2.5.2. 5.7700×10^{-1}

E.2.5.4. Dica: Analise a convergência das iteradas para cada abordagem.

E.2.5.5. Dica: $x = \pi$ é zero de multiplicidade 3.

2.6 Método da Secante

O Método da Secante é um método tipo de Newton. Observamos que para duas aproximações $x^{(k)}$ e $x^{(k-1)}$ suficientemente próximas, temos[27]

$$f'(x^{(k)}) \approx \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}. \tag{2.160}$$

Assim sendo, substituindo esta aproximação na iteração de Newton (Eq. (2.129)), obtemos a **iteração do Método da Secante**

$$x^{(0)}, x^{(1)} = \text{aprox. iniciais}, \tag{2.161a}$$

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}, \tag{2.161b}$$

para $k = 1, 2, 3, \ldots$.

Exemplo 2.6.1. Consideramos o problema de encontrar o zero da função

$$\begin{aligned} f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 \\ + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \end{aligned} \tag{2.162}$$

no intervalo $[2, 3]$. Fazendo as iterações do Método da Secante com aproximações iniciais $x^{(0)} = 2.6$ e $x^{(1)} = 2.5$, obtemos os resultados apresentados na Tabela 2.10.

```
1 import numpy as np
2
3 # fun obj
```

Tabela 2.10: Resultados referentes ao Exemplo **??**.

k	$x^{(k-1)}$	$x^{(k)}$	$\lvert x^{(k)} - x^{(k-1)}\rvert$
0	2.6000	2.5000	-x-
1	2.5000	2.3728	1.3e−1
2	2.3728	2.3574	1.5e−2
3	2.3574	2.3562	1.2e−3
4	2.3562	2.3562	1.1e−5
5	2.3562	2.3562	7.0e−9

```
4 f = lambda x: np.sin(x+np.pi/4)**2 \
5   - x**3 + np.pi/4*x**2 + 5*np.pi**2/16*x \
6   + 3*np.pi**3/64
7
8 # aprox. iniciais
9 x0 = 2.6
10 x1 = 2.5
11 print(f'\n0: {x0:.4f}, {x1:.4f}')
12
13 # iterações
14 for k in range(5):
15   x = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0))
16   x0 = x1
17   x1 = x
18   print(f'{k+1}: {x0:.4f}, {x1:.4f}, {np.fabs(x1-
x0):.1e}')
```

2.6.1 Interpretação Geométrica

A iteração do Método da Secante é

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})\frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}, \tag{2.163}$$

donde segue que

$$0 = x^{(k+1)} - x^{(k)} + f(x^{(k)})\frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}, \tag{2.164}$$

bem como que

$$0 = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}(x^{(k+1)} - x^{(k)}) + f(x^{(k)}). \tag{2.165}$$

Ou seja, $x^{(k+1)}$ é o ponto de interseção da reta

$$y = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x - x^{(k-1)}}(x^{(k+1)} - x^{(k)}) + f(x^{(k)}). \tag{2.166}$$

com o eixo x. Esta é a reta secante ao gráfico de f pelos pontos $(x^{(k)}, f(x^{(k)}))$ e $(x^{(k-1)}, f(x^{(k-1)}))$.

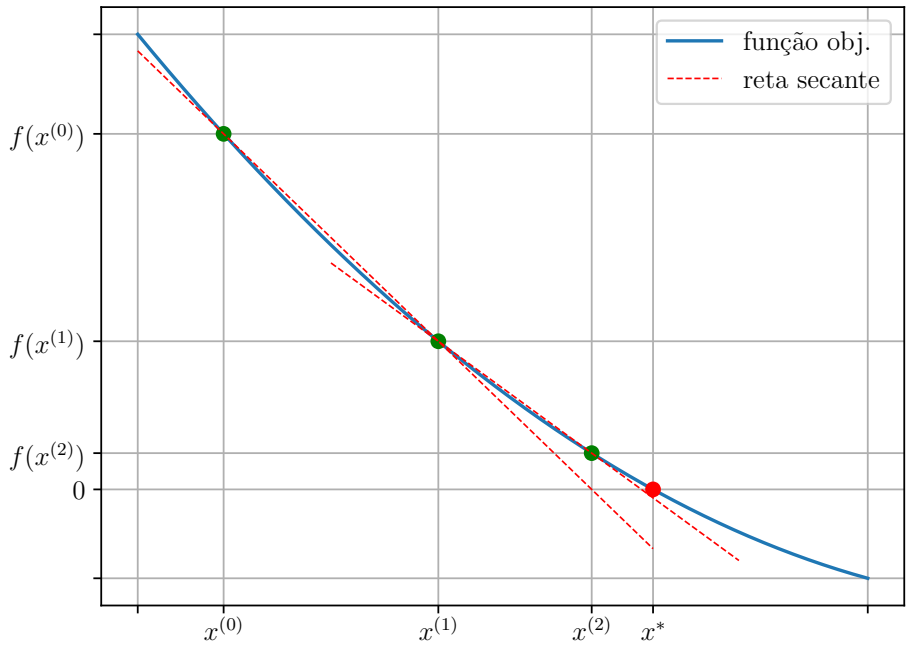

Figura 2.11: Interpretação geométrica do Método da Secante.

Observação 2.6.1. (Aproximações iniciais.) A interpretação geométrica do método da secante pode nos ajudar a escolher as aproximações iniciais $x^{(1)}$ e

$x^{(2)}$. Como uma boa prática, escolhemo-las próximas do zero (por inspeção gráfica), tomando $x^{(1)}$ como uma aproximação melhor que $x^{(0)}$.

Observação 2.6.2. (Ordem de convergência super-linear.) A ordem de convergência do Método da Secante é superlinear com

$$|x^{(k+1)} - x^*| \leq C|x^{(k)} - x^*|^{\varphi}, \tag{2.167}$$

onde $\varphi = (1 + \sqrt{5})/2 \approx 1.618$ (razão áurea) e x^* é o zero de f.

Observação 2.6.3. (Zeros de multiplicidade par.) A ordem de convergência super-linear do método da secante não se mantém para o caso de x^* ser um zero múltiplo. Para contornar este problema, pode-se aplicar o método à derivada $n-1$ de f, a fim de se aproximar um zero de multiplicidade n.

Observação 2.6.4. (Cancelamento catastrófico.) Conforme convergem as iterações do método da secante, o denominador $f(x^{(k)}) - f(x^{(k-1)})$ pode convergir rapidamente para zero, ocasionando uma divisão por zero.

Exercícios

E.2.6.1. Use o Método da Secante para obter uma aproximação do zero de

$$f(x) = x^3 \operatorname{sen}(x) - \cos(x) \tag{2.168}$$

no intervalo $[0.5, 1]$ com precisão de 10^{-5}.

E.2.6.2. Use o Método da Secante para computar a(s) solução(ões) das seguintes equações com precisão de 8 dígitos significativos.

a) $x = 2^{-x}$ para $0 \leq x \leq 2$.

b) $e^{-x^2} = 3x - x^2$ para $-1 \leq x \leq 4$.

E.2.6.3. Use o Método da Secante para obter uma aproximação do zero de

$$f(x) = \operatorname{sen}^2\left(x + \frac{\pi}{4}\right) - x^3 + \frac{\pi}{4}x^2 + \frac{5\pi^2}{16}x + \frac{3\pi^3}{64}. \tag{2.169}$$

no intervalo $[-1, 0]$ com precisão de 10^-5. Compare a convergência entre as seguintes abordagens:

a) aplicando a iteração (2.161) diretamente à f.

b) aplicando a iteração (2.161) diretamente à f'.

Qual das duas abordagens tem convergência mais rápida? Justifique sua resposta.

E.2.6.4. Use o Sétodo da Secante para obter uma aproximação do zero de

$$f(x) = (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 - 1.154x + 0.332929 \tag{2.170}$$

no intervalo $[0.55, 0.65]$ com precisão de 10^-5.

E.2.6.5. Use o Método da Secante para encontrar uma aproximação com precisão de 4 dígitos significativos do zero de

$$f(x) = (-x^2 + 1.154x - 0.332929)\cos(x) + x^2 - 1.154x + 0.332929 \tag{2.171}$$

no intervalo $[-1, 0]$.

bootstra

E.2.6.6. Use o Método da Secante para encontrar o ponto crítico[28] de

$$f(x) = (1 - x^2)e^{-x^2} \tag{2.172}$$

no intervalo $[0, 2]$. Obtenha o resultado com precisão de 5 dígitos significativos por arredondamento.

Respostas

E.2.6.1. 9.1581×10^{-1}

E.2.6.2. a) 6.4118574×10^{-1}; b) 3.3536470×10^{-1}; 2.9999589

E.2.6.3. Dica: f tem um zero de multiplicidade par no intervalo $[-1, 0]$.

E.2.6.4. 5.7700×10^{-1}

E.2.6.5. -7.861×10^{-1}

2.7 Raízes de Polinômios

Em revisão

Nesta seção, veremos como o método de Newton pode ser aplicado de forma robusta a polinômios com o auxílio do método de Horner[29]. A questão central é que dado um polinômio de grau n escrito na forma

$$p(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}, \tag{2.173}$$

o procedimento de avaliá-lo em um ponto requer $n!n$ multiplicações e n adições. Desta forma, a aplicação do método de Newton para obter as raízes de p torna-se computacionalmente custosa, por requer a avaliação de p e de sua derivada a cada iteração. Agora, o método de Horner nos permite avaliar p em um ponto qualquer usando apenas n multiplicações e n adições, reduzindo enormemente o custo computacional.

2.7.1 Método de Horner

Em revisão

Sejam dados um número x_0 e um polinômio de grau n

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1} s. \tag{2.174}$$

O método de Horner consiste em computar $p(x_0)$ pela iteração

$$q_1 = p_1, \tag{2.175}$$
$$q_k = p_k + q_{k-1}x_0, \quad k = 2, 3, \ldots, n+1, \tag{2.176}$$

sendo que, então, $p(x_0) = q_{n+1}$ e, além disso,

$$p(x) = (x - x_0)q(x) + q_{n+1}, \tag{2.177}$$

com

$$q(x) = q_1 x^{n-1} + q_2 x^{n-2} + \cdots + q_{n-1}x + q_n. \tag{2.178}$$

De fato, a verificação de (2.177) é direta, uma vez que

$$\begin{aligned}(x - x_0)q(x) + q_{n+1} &= (x - x_0)(q_1 x^{n-1} + q_2 x^{n-2} + \cdots + q_{n-1}x + q_n) \\ &\quad + q_{n+1} \qquad (2.179)\\ &= q_1 x^n + (q_2 - q_1 x_0)^{n-1} + \cdots + (q_{n+1} - q_n x_0). \qquad (2.180)\end{aligned}$$

E, então, igualando a $p(x)$ na forma (2.174), temos as equações (2.175)-(2.176).

Exemplo 2.7.1. Consideremos o polinômio

$$p(x) = x^3 - 3x^2 + 4. \tag{2.181}$$

Para computarmos $p(1)$ pelo método de Horner, tomamos $x_0 = 1$, $q_1 = p_1 = 1$ e

$$q_2 = p_2 + q_1 x_0 = -3 + 1 \cdot 1 = -2 \tag{2.182}$$
$$q_3 = p_3 + q_2 x_0 = 0 + (-2) \cdot 1 = -2 \tag{2.183}$$
$$q_4 = p_4 + q_3 x_0 = 4 + (-2) \cdot 1 = 2. \tag{2.184}$$

Com isso, temos $p(3) = q_4 = 4$ (verifique!).

Observação 2.7.1. Ao computarmos $p(x_0)$ pelo método de Horner, obtemos a decomposição

$$p(x) = (x - x_0)q(x) + b_{n+1}. \tag{2.185}$$

Desta forma, temos

$$p'(x) = q(x) + (x - x_0)q'(x), \tag{2.186}$$

donde temos que $p'(x_0) = q(x_0)$. Com isso, para computarmos $p'(x_0)$ podemos aplicar o método de Horner a $q(x)$.

2.7.2 Método de Newton-Horner

Em revisão

A implementação do método de Newton a polinômios pode ser feita de forma robusta com o auxílio do método de Horner. Dado um polinômio p e uma aproximação inicial $x^{(1)}$ para uma de suas raízes reais, a iteração de Newton consiste em

$$x^{(k+1)} = x^{(k)} - \frac{p(x^{(k)})}{p'(x^{(k)})}, \tag{2.187}$$

na qual podemos utilizar o método de Horner para computar $p(x^{(k)})$ e $p'(x^{(k)})$.

Exemplo 2.7.2. Consideremos o caso de aplicar o método de Newton para obter uma aproximação da raiz $x = -1$ de

$$p(x) = x^3 - 3x^2 + 4, \tag{2.188}$$

com aproximação inicial $x^{(1)} = -2$. Na Tabela 2.11 temos os resultados obtidos.

Tabela 2.11: Resultados referentes ao Exemplo 2.7.2.

k	$x^{(k)}$	$\lvert x^{(k)} - x^{(k-1)}\rvert$
1	-2.0000	-x-
2	-1.3333	$6.7\text{e}{-1}$
3	-1.0556	$2.8\text{e}{-1}$
4	-1.0019	$5.4\text{e}{-2}$
5	-1.0000	$1.9\text{e}{-3}$

2.7.3 Exercícios

Em revisão

E.2.7.1. Use o método de Newton-Horner para computar a aproximação da raiz de $p(x) = x^3 - 3x^2 + 4$ no intervalo $[1, 3]$. Observe que p tem um zero duplo deste intervalo.

Respostas

E.2.7.1. 2

Capítulo 3

Métodos para Sistemas Lineares

Neste capítulo, discutimos sobre métodos diretos para a resolução de sistemas lineares de n-equações com n-incógnitas. Isto é, sistemas que podem ser escritos na seguinte **forma algébrica**

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 && \text{(3.1a)}\\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 && \text{(3.1b)}\\ &\vdots && \text{(3.1c)}\\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. && \text{(3.1d)} \end{aligned}$$

Equivalentemente, o sistema pode ser escrito na **forma matricial**

$$A\boldsymbol{x} = \boldsymbol{b} \tag{3.2}$$

onde, $A = [a_{i,j}]_{i,j=1}^{n,n}$ é a **matriz dos coeficientes**

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix}, \tag{3.3}$$

o **vetor das incógnitas** $\boldsymbol{x} = (x_i)_{i=1}^n$ é

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.4}$$

e o **vetor dos termos constantes** $\boldsymbol{b} = (b_i)_{i=1}^n$ é

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.5}$$

3.1 Método da Decomposição LU

O Método da Decomposição LU é uma forma eficiente de se resolver sistemas lineares de pequeno porte. Dado um sistema $A\boldsymbol{x} = \boldsymbol{b}$, a ideia é decompor a matriz A como o produto de uma **matriz triangular inferior** L (do inglês, *lower triangular matrix*) com uma **matriz triangular superior** U (do inglês, *upper triangular matrix*), i.e.

$$A = LU. \tag{3.6}$$

Com isso, o sistema pode ser reescrito na forma

$$A\boldsymbol{x} = \boldsymbol{b} \tag{3.7}$$

$$(LU)\boldsymbol{x} = \boldsymbol{b} \tag{3.8}$$

$$L(U\boldsymbol{x}) = \boldsymbol{b}. \tag{3.9}$$

Denotando,

$$U\boldsymbol{x} =: \mathbf{y} \tag{3.10}$$

podemos resolver o seguinte sistema triangular

$$L\boldsymbol{y} = \boldsymbol{b}. \tag{3.11}$$

Tendo resolvido este sistema, a solução do sistema $A\boldsymbol{x} = \boldsymbol{b}$ pode, então, ser computada como a solução do sistema triangular

$$U\boldsymbol{x} = \boldsymbol{y}. \tag{3.12}$$

Ou seja, a decomposição LU nos permite resolver uma sistema pela resolução de dois sistemas triangulares.

3.1.1 Sistemas Triangulares

Antes de estudarmos como podemos computar a decomposição LU de uma matriz, vamos discutir sobre a resolução de sistemas triangulares.

Sistema Triangular Inferior

Um **sistema linear triangular inferior** tem a forma algébrica

$$\begin{array}{llllll} a_{1,1}x_1 & & & & & = b_1 \\ a_{2,1}x_1 & +a_{2,2}x_2 & & & & = b_2 \\ \vdots & & & & & \vdots \\ a_{n-1,1}x_1 & +a_{n-1,2}x_2 & +\cdots & +a_{n-1,n-1}x_{n-1} & & = b_{n-1} \\ a_{n,1}x_1 & +a_{n,2}x_2 & +\cdots & +a_{n,n-1}x_{n-1} & +a_{n,n}x_n & = b_n \end{array} \tag{3.13}$$

Pode ser diretamente resolvido de cima para baixo, i.e.

$$x_1 = \frac{b_1}{a_{1,1}} \tag{3.14}$$

$$x_2 = \frac{b_2 - a_{2,1}x_1}{a_{2,2}} \tag{3.15}$$

$$\vdots \tag{3.16}$$

$$x_n = \frac{b_n - a_{n,1}x_1 - a_{n,2}x_2 - \ldots - a_{n,n-1}x_{n-1}}{a_{n,n}} \tag{3.17}$$

Exemplo 3.1.1. Vamos resolver o sistema triangular inferior

$$\begin{array}{llll} x_1 & & & = 2 \\ -3x_1 & +2x_2 & & = -8 \\ -x_1 & +x_2 & -x_3 & = 0 \end{array} \tag{3.18}$$

Na primeira equação, temos

$$x_1 = 2 \tag{3.19}$$

Então, da segunda equação do sistema

$$-3x_1 + 2x_2 = -8 \tag{3.20}$$

$$x_2 = \frac{-8 + 3x_1}{2} \tag{3.21}$$

$$x_2 = \frac{-8 + 3 \cdot 2}{2} \tag{3.22}$$

$$x_2 = -1 \tag{3.23}$$

E, por fim, da última equação

$$-x_1 + x_2 - x_3 = 0 \tag{3.24}$$

$$x_3 = \frac{x_1 - x_2}{-1} \tag{3.25}$$

$$x_3 = \frac{2 - (-1)}{-1} \tag{3.26}$$

$$x_3 = 3 \tag{3.27}$$

Concluímos que a solução do sistema é $\boldsymbol{x} = (2, -1, 3)$.

Código 3.1: solSisTriaInf.py

```
1  import numpy as np
2
3  def solSisTriaInf(A, b):
4    n = b.size
5    x = np.zeros_like(b)
6    for i in range(n):
7      x[i] = b[i]
8      for j in range(i):
9        x[i] -= A[i,j]*x[j]
10     x[i] /= A[i,i]
11   return x
12
13 # mat coefficientes
14 A = np.array([[1., 0., 0.],
15               [-3., 2., 0.],
16               [-1., 1., -1.]])
17
18 # vet termos constantes
19 b = np.array([2., -8., 0.])
20
21 # resol sis lin
```

```
x = solSisTriaInf(A, b)
print(x)
```

Observação 3.1.1. (Número de operações em ponto flutuante.) A computação da solução de um sistema $n \times n$ triangular inferior requer $O(n^2)$ operações em ponto flutuante (multiplicações/divisões e adições/subtrações).

Sistema Triangular Superior

Um **sistema linear triangular superior** tem a forma algébrica

$$\begin{array}{rrcrl} a_{1,1}x_1 & +a_{1,2}x_2 & +\cdots & +a_{1,n}x_n & = b_1 \\ & a_{2,2}x_2 & +\cdots & +a_{2,n}x_n & = b_2 \\ & & & \vdots & \vdots \\ & & & a_{n,n}x_n & = b_n \end{array} \tag{3.28}$$

Pode ser diretamente resolvido de baixo para cima, i.e.

$$x_n = \frac{b_n}{a_{n,n}} \tag{3.29}$$

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}} \tag{3.30}$$

$$\vdots \tag{3.31}$$

$$x_n = \frac{b_1 - a_{1,1}x_1 - a_{1,2}x_2 - \ldots - a_{1,n-1}x_{n-1}}{a_{1,1}} \tag{3.32}$$

Exemplo 3.1.2. Vamos resolver o sistema triangular superior

$$\begin{array}{rrrl} 2x_1 & -x_2 & +2x_3 & = 7 \\ & 2x_2 & -x_3 & = -3 \\ & & 3x_3 & = 3 \end{array} \tag{3.33}$$

Da última equação, temos

$$x_3 = \frac{3}{3} \tag{3.34}$$

$$x_3 = 1 \tag{3.35}$$

Então, da segunda equação do sistema, obtemos

$$2x_2 - x_3 = -3 \tag{3.36}$$
$$x_2 = \frac{x_3 - 3}{2} \tag{3.37}$$
$$x_2 = \frac{1 - 3}{2} \tag{3.38}$$
$$x_2 = -1 \tag{3.39}$$

Por fim, da primeira equação

$$2x_1 - x_2 + 2x_3 = 7 \tag{3.40}$$
$$x_1 = \frac{7 + x_2 - 2x_3}{2} \tag{3.41}$$
$$x_1 = \frac{7 - 1 - 2}{2} \tag{3.42}$$
$$x_1 = 2 \tag{3.43}$$

Concluímos que a solução do sistema é $\boldsymbol{x} = (2, -1, 1)$.

Código 3.2: solSisTriaSup.py

```
1 import numpy as np
2
3 def solSisTriaSup(A, b):
4   n = b.size
5   x = np.zeros_like(b)
6   for i in range(n-1,-1,-1):
7     x[i] = b[i]
8     for j in range(n-1,i,-1):
9       x[i] -= A[i,j]*x[j]
10    x[i] /= A[i,i]
11  return x
12
13
14 # mat coefficientes
15 A = np.array([[2., -1., 2.],
16               [0., 2., -1.],
17               [0., 0., 3.]])
18
```

```
19 # vet termos constantes
20 b = np.array([7., -3., 3.])
21
22 # sol sis lin
23 x = solSisTriaSup(A, b)
24 print(x)
```

Observação 3.1.2. (Número de operações em ponto flutuante.) A computação da solução um sistema $n \times n$ triangular superior requer $O(n^2)$ operações em ponto flutuante (multiplicações/divisões e adições/subtrações).

3.1.2 Decomposição LU

O procedimento da decomposição LU é equivalente ao método de eliminação gaussiana. Consideramos uma matriz $A = [a_{ij}]_{i,j=1}^{n,n}$, com $a_{1,1} \neq 0$, e começamos denotando esta matriz por $U^{(0)} = [u_{i,j}^{(0)}]_{i,j=1}^{n,n} = A$ e tomando $L^{(0)} = I_{n\times n}$. A eliminação abaixo do pivô $u_{1,1}^{(0)}$, pode ser computada com as seguintes operações equivalentes por linha

$$U^{(0)} = \begin{bmatrix} u_{1,1}^{(0)} & u_{1,2}^{(0)} & u_{1,3}^{(0)} & \dots & u_{1,n}^{(0)} \\ u_{2,1}^{(0)} & u_{2,2}^{(0)} & u_{2,3}^{(0)} & \dots & u_{2,n}^{(0)} \\ u_{3,1}^{(0)} & u_{3,2}^{(0)} & u_{3,3}^{(0)} & \dots & u_{3,n}^{(0)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ u_{n,1}^{(0)} & u_{n,2}^{(0)} & a_{n,3}^{(0)} & \dots & u_{n,n}^{(0)} \end{bmatrix} \begin{array}{l} U_1^{(1)} \leftarrow U_1^{(0)} \\ U_2^{(1)} \leftarrow U_2^{(0)} - l_{2,1}^{(1)} U_1^{(0)} \\ U_3^{(1)} \leftarrow U_3^{(0)} - l_{3,1}^{(1)} U_1^{(0)} \\ \vdots \\ U_n^{(1)} \leftarrow U_n^{(0)} - l_{n,1}^{(1)} U_1^{(0)} \end{array} \tag{3.44}$$

onde, $l_{i,1}^{(1)} = u_{i,1}^{(0)} / u_{1,1}^{(0)}$, $i = 2, 3, \dots, n$.

Destas computações, obtemos uma nova matriz da forma

$$U^{(1)} = \begin{bmatrix} u_{1,1}^{(1)} & u_{1,2}^{(1)} & u_{1,3}^{(1)} & \dots & u_{1,n}^{(1)} \\ 0 & u_{2,2}^{(1)} & u_{2,3}^{(1)} & \dots & u_{2,n}^{(1)} \\ 0 & u_{3,2}^{(1)} & u_{3,3}^{(1)} & \dots & u_{3,n}^{(1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & u_{n,2}^{(1)} & u_{n,3}^{(1)} & \dots & u_{n,n}^{(1)} \end{bmatrix} \tag{3.45}$$

E, denotando

$$L^{(1)} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{2,1}^{(1)} & 1 & 0 & \dots & 0 \\ l_{3,1}^{(1)} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ l_{n,1}^{(1)} & 0 & 0 & \dots & 1 \end{bmatrix} \tag{3.46}$$

temos

$$A = L^{(1)}U^{(1)}. \tag{3.47}$$

No caso de $u_{2,2}^{(1)} \neq 0$, podemos continuar com o procedimento de eliminação gaussiana com as seguintes operações equivalentes por linha

$$U^{(1)} = \begin{bmatrix} u_{1,1}^{(1)} & u_{1,2}^{(1)} & u_{1,3}^{(1)} & \dots & u_{1,n}^{(1)} \\ 0 & u_{2,2}^{(1)} & u_{2,3}^{(1)} & \dots & u_{2,n}^{(1)} \\ 0 & u_{3,2}^{(1)} & u_{3,3}^{(1)} & \dots & u_{3,n}^{(1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & u_{n,2}^{(1)} & u_{n,3}^{(1)} & \dots & u_{n,n}^{(1)} \end{bmatrix} \begin{array}{l} U_1^{(2)} \leftarrow U_1^{(1)} \\ U_2^{(2)} \leftarrow U_2^{(1)} \\ U_3^{(2)} \leftarrow U_3^{(1)} - l_{3,2}^{(2)}U_2^{(1)} \\ \vdots \\ U_n^{(2)} \leftarrow U_n^{(1)} - l_{n,2}^{(2)}U_2^{(1)} \end{array} \tag{3.48}$$

onde, $l_{i,2}^{(2)} = u_{i,2}^{(1)}/u_{2,2}^{(1)}$, $i = 3, 4, \dots, n$. Isto nos fornece o que nos fornece

$$U^{(2)} = \begin{bmatrix} u_{1,1}^{(2)} & u_{1,2}^{(2)} & u_{1,3}^{(2)} & \dots & u_{1,n}^{(2)} \\ 0 & u_{2,2}^{(2)} & u_{2,3}^{(2)} & \dots & u_{2,n}^{(2)} \\ 0 & 0 & u_{3,3}^{(2)} & \dots & u_{3,n}^{(2)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & u_{n,3}^{(2)} & \dots & u_{n,n}^{(2)} \end{bmatrix}. \tag{3.49}$$

Além disso, denotando

$$L^{(2)} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{2,1}^{(1)} & 1 & 0 & \dots & 0 \\ l_{3,1}^{(1)} & l_{3,2}^{(2)} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ l_{n,1}^{(1)} & l_{n,2}^{(2)} & 0 & \dots & 1 \end{bmatrix} \tag{3.50}$$

temos

$$A = L^{(2)}U^{(2)}. \tag{3.51}$$

Continuando com este procedimento, ao final de $n-1$ passos teremos obtido a decomposição

$$A = LU, \tag{3.52}$$

onde L é a matriz triangular inferior

$$L = L^{(n-1)} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{2,1}^{(1)} & 1 & 0 & \dots & 0 \\ l_{3,1}^{(1)} & l_{3,2}^{(2)} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ l_{n,1}^{(1)} & l_{n,2}^{(2)} & l_{n,3}^{(3)} & \dots & 1 \end{bmatrix} \tag{3.53}$$

e U é a matriz triangular superior

$$U = U^{(n-1)} = \begin{bmatrix} u_{1,1}^{(0)} & u_{1,2}^{(0)} & u_{1,3}^{(0)} & \dots & u_{1,n}^{(0)} \\ 0 & u_{2,2}^{(1)} & u_{2,3}^{(1)} & \dots & u_{2,n}^{(1)} \\ 0 & 0 & u_{3,3}^{(2)} & \dots & u_{3,n}^{(2)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & u_{n,n}^{(n-1)} \end{bmatrix}. \tag{3.54}$$

Exemplo 3.1.3. Consideramos a seguinte matriz

$$A = \begin{bmatrix} -1 & 2 & -2 \\ 3 & -4 & 1 \\ 1 & -5 & 3 \end{bmatrix}. \tag{3.55}$$

Para obtermos sua decomposição LU começamos com

$$L^{(0)} := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.56}$$

$$U^{(0)} := \begin{bmatrix} -1 & 2 & -2 \\ 3 & -4 & 1 \\ 1 & -5 & 3 \end{bmatrix}. \tag{3.57}$$

Então, observando que a eliminação abaixo do pivô $u_{1,1} = -1$ pode ser feita com as seguintes operações equivalentes por linha

$$U^{(0)} = \begin{bmatrix} -1 & 2 & -2 \\ 3 & -4 & 1 \\ 1 & -5 & 3 \end{bmatrix} \begin{array}{l} U_1^{(1)} \leftarrow U_1^{(0)} \\ U_2^1 \leftarrow U_2^{(0)} - \frac{3}{-1} U_1^{(0)} \\ U_3^1 \leftarrow U_3^{(0)} - \frac{1}{-1} U_1^{(0)} \end{array}, \tag{3.58}$$

temos

$$L^{(1)} := \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \tag{3.59}$$

$$U^{(1)} := \begin{bmatrix} -1 & 2 & -2 \\ 0 & 2 & -5 \\ 0 & -3 & 1 \end{bmatrix}. \tag{3.60}$$

Agora, para eliminarmos abaixo do pivô $u_{2,2} = 2$, usamos as operações

$$U^{(1)} := \begin{bmatrix} -1 & 2 & -2 \\ 0 & 2 & -5 \\ 0 & -3 & 1 \end{bmatrix} \begin{array}{l} U_1^{(2)} \leftarrow U_1^{(1)} \\ U_2^{(2)} \leftarrow U_2^{(1)} \\ U_3^{(2)} \leftarrow U_3^{(1)} - \frac{-3}{2} U_2^{(1)} \end{array}, \tag{3.61}$$

donde

$$L^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -1 & -1.5 & 1 \end{bmatrix}, \tag{3.62}$$

$$U^{(1)} = \begin{bmatrix} -1 & 2 & -2 \\ 0 & 2 & -5 \\ 0 & 0 & -6.5 \end{bmatrix}. \tag{3.63}$$

Isso completa a decomposição, sendo $L := L^{(3)}$ e $U := U^{(3)}$.

Código 3.3: lu.py

```
1 import numpy as np
2
3 def lu(A):
4   # num. de linhas
5   n = A.shape[0]
```

```
6
7   # inicialização
8   U = A.copy()
9   L = np.eye(n)
10
11  # decomposição
12  for i in range(n-1):
13    for j in range(i+1,n):
14        L[j,i] = U[j,i]/U[i,i]
15        U[j,i:] -= L[j,i]*U[i,i:]
16
17  return L, U
18
19# matriz
20A = np.array([[-1., 2., -2.],
21               [3., -4., 1.],
22               [1., -5., 3.]])
23
24L, U = lu(A)
```

Observação 3.1.3. (Número de operações em ponto flutuante.) A decomposição LU de um sistema $n \times n$ requer $O(n^3)$ operações em ponto flutuante (multiplicações/divisões e adições/subtrações).

3.1.3 Resolução do Sistema com Decomposição LU

Consideramos o sistema linear

$$A\boldsymbol{x} = \boldsymbol{b}. \tag{3.64}$$

Para resolvê-lo com o Método LU, fazemos

1. Computamos a decomposição LU

$$A = LU. \tag{3.65}$$

2. Resolvemos o sistema triangular inferior

$$L\boldsymbol{y} = \boldsymbol{b}. \tag{3.66}$$

3. Resolvemos o sistema triangular superior

$$U\boldsymbol{x} = \boldsymbol{y}. \tag{3.67}$$

Exemplo 3.1.4. Vamos resolver o seguinte sistema linear

$$\begin{aligned} -x_1 + 2x_2 - 2x_3 &= 6 \\ 3x_1 - 4x_2 + x_3 &= -11 \\ x_1 - 5x_2 + 3x_3 &= -10. \end{aligned} \tag{3.68}$$

No exemplo anterior (Exemplo 3.1.4), vimos que a matriz de coeficientes A deste sistema admite a seguinte decomposição LU

$$\underbrace{\begin{bmatrix} -1 & 2 & -2 \\ 3 & -4 & 1 \\ 1 & -5 & 3 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -1 & -1.5 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} -1 & 2 & -2 \\ 0 & 2 & -5 \\ 0 & 0 & -6.5 \end{bmatrix}}_{U} \tag{3.69}$$

Daí, iniciamos resolvendo o seguinte sistema triangular inferior $Ly = b$, i.e.

$$y_1 = 6 \tag{3.70}$$
$$-3y_1 + y_2 = -11 \tag{3.71}$$
$$\Rightarrow y_2 = 7 \tag{3.72}$$
$$-y_1 - 1.5y_2 + y_3 = -10 \tag{3.73}$$
$$\Rightarrow y_3 = 6.5. \tag{3.74}$$

Por fim, computamos a solução x resolvendo o sistema triangular superior $Ux = y$, i.e.

$$-6.5x_3 = 6.5 \tag{3.75}$$
$$\Rightarrow x_3 = -1, \tag{3.76}$$
$$2x_2 - 5x_3 = 7 \tag{3.77}$$
$$\Rightarrow x_2 = 1 \tag{3.78}$$
$$-x_1 + 2x_2 - 2x_3 = 6 \tag{3.79}$$
$$\Rightarrow x_1 = -2. \tag{3.80}$$

```python
import numpy as np

# mat coefs
A = np.array([[-1., 2., -2.],
              [3., -4., 1.],
              [1., -5., 3.]])
# vet termos const
b = np.array([6., -11., -10.])

# 1. LU
L, U = lu(A)

# 2. Ly = b
y = solSisTriaInf(L, b)

# 3. Ux = y
x = solSisTriaSup(U, y)
```

3.1.4 Fatoração LU com Pivotamento Parcial

O algoritmo estudando acima não é aplicável no caso de o pivô ser nulo, o que pode ser corrigido através de permutações de linhas. Na fatoração LU com pivotamento parcial fazemos permutações de linha na matriz de forma que o pivô seja sempre aquele de maior valor em módulo. Por exemplo, suponha que o elemento $a_{3,1}$ seja o maior valor em módulo na primeira coluna da matriz $A = U^{(0)}$ com

$$U^{(0)} = \begin{bmatrix} u_{1,1}^{(0)} & u_{1,2}^{(0)} & u_{1,3}^{(0)} & \dots & u_{1,n}^{(0)} \\ u_{2,1}^{(0)} & u_{2,2}^{(0)} & u_{2,3}^{(0)} & \dots & u_{2,n}^{(0)} \\ \mathbf{u_{3,1}^{(0)}} & u_{3,2}^{(0)} & u_{3,3}^{(0)} & \dots & u_{3,n}^{(0)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ u_{n,1}^{(0)} & u_{n,2}^{(0)} & a_{n,3}^{(0)} & \dots & u_{n,n}^{(0)} \end{bmatrix}. \tag{3.81}$$

Neste caso, o procedimento de eliminação na primeira coluna deve usar $u_{3,1}^{(0)}$ como pivô, o que requer a permutação entre as linhas 1 e 3 ($U_1^{(0)} \leftrightarrow U_3^{(0)}$).

Isto pode ser feito utilizando-se da seguinte **matriz de permutação**

$$P = \begin{bmatrix} 0 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}. \tag{3.82}$$

Com essa, iniciamos o procedimento de decomposição LU com $PA = L^{(0)}U^{(0)}$, onde $L^{(0)} = I_{n\times n}$ e $U^{(0)} = PA$. Caso sejam necessárias outras mudanças de linhas no decorrer do procedimento de decomposição, a matriz de permutação P deve ser atualizada apropriadamente.

Exemplo 3.1.5. Vamos fazer a decomposição LU com pivotamento parcial da seguinte matriz

$$A = \begin{bmatrix} -1 & 2 & -2 \\ 3 & -4 & 1 \\ 1 & -5 & 3 \end{bmatrix} \tag{3.83}$$

Começamos, tomando

$$P^{(0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.84}$$

$$L^{(0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.85}$$

$$U^{(0)} = \begin{bmatrix} -1 & 2 & -2 \\ 3 & -4 & 1 \\ 1 & -5 & 3 \end{bmatrix} \tag{3.86}$$

O candidato a pivô é o elemento $u^{(0)}_{2,1}$. Então, fazemos as permutações de linhas

$$P^{((0)}_1 \leftrightarrow P^{(0)}_2, \tag{3.87}$$

$$U^{(0)}_1 \leftrightarrow U^{(0)}_2 \tag{3.88}$$

e, na sequência, as operações elementares por linhas

$$U_{2:3}^{(1)} \leftarrow U_{2:3}^{(0)} - m_{2:3,1}^{(0)} U_1^{(0)}, \tag{3.89}$$

donde obtemos

$$P^{(1)} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.90}$$

$$L^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -0,\overline{3} & 1 & 0 \\ 0,\overline{3} & 0 & 1 \end{bmatrix}, \tag{3.91}$$

$$U^{(1)} = \begin{bmatrix} 3 & -4 & 1 \\ 0 & 0,\overline{6} & -1,\overline{6} \\ 0 & -3,\overline{6} & 2,\overline{6} \end{bmatrix} \tag{3.92}$$

Agora, o candidato a pivô é o elemento $u_{3,2}^{(1)}$. Assim, fazemos as permutações de linhas

$$P_2^{(1)} \leftrightarrow P_3^{(1)}, \tag{3.93}$$

$$U_2^{(1)} \leftrightarrow U_3^{(1)} \tag{3.94}$$

e análogo para os elementos da coluna 1 de L. Então, fazemos a operação elementar por linha

$$U_3^{(2)} \leftarrow U_3^{(1)} - m_{3,2}^{(1)} U_2^{(1)} \tag{3.95}$$

. Com isso, obtemos

$$P^{(2)} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \tag{3.96}$$

$$L^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0,\overline{3} & 1 & 0 \\ -0,\overline{3} & -0,\overline{18} & 1 \end{bmatrix}, \tag{3.97}$$

$$U^{(2)} = \begin{bmatrix} 3 & -4 & 1 \\ 0 & -3,\overline{6} & 2,\overline{6} \\ 0 & 0 & -1,\overline{18} \end{bmatrix} \tag{3.98}$$

Por fim, temos obtido a decomposição LU de A na forma

$$PA = LU, \tag{3.99}$$

com $P = P^{(2)}$, $L = L^{(2)}$ e $U = U^{(2)}$.

Código 3.4: lup.py

```
import numpy as np

def lup(A):
  # num. de linhas
  n = A.shape[0]

  # inicialização
  U = A.copy()
  L = np.eye(n)
  P = np.eye(n)

  # decomposição
  for i in range(n-1):
    # permutação de linhas
    p = i + np.argmax(np.fabs(U[i:,i]))
    P[[i,p]] = P[[p,i]]
    U[[i,p]] = U[[p,i]]
    L[[i,p],:i] = L[[p,i],:i]
    # eliminação gaussiana
    for j in range(i+1,n):
      L[j,i] = U[j,i]/U[i,i]
      U[j,i:] -= L[j,i]*U[i,i:]

  return P, L, U

# matriz
A = np.array([[-1., 2, -2],
              [3, -4, 1],
              [1, -5, 3]])

P, L, U = lup(A)
```

Exemplo 3.1.6. Vamos computar a solução do seguinte sistema linear com o Método da Decomposição LU com Pivotamento Parcial.

$$-x_1 + 2x_2 - 2x_3 = 6 \tag{3.100}$$
$$3x_1 - 4x_2 + x_3 = -11 \tag{3.101}$$
$$x_1 - 5x_2 + 3x_3 = -10. \tag{3.102}$$

No exemplo anterior (Exemplo 3.1.5), vimos que a matriz de coeficientes A deste sistema admite a seguinte decomposição LU

$$PA = LU \tag{3.103}$$

com

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \tag{3.104}$$
$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0,\overline{3} & 1 & 0 \\ -0,\overline{3} & -0,\overline{18} & 1 \end{bmatrix}, \tag{3.105}$$
$$U = \begin{bmatrix} 3 & -4 & 1 \\ 0 & -3,\overline{6} & 2,\overline{6} \\ 0 & 0 & -1,\overline{18} \end{bmatrix} \tag{3.106}$$

Multiplicando o sistema a esquerda pela matriz P, obtemos

$$PA\boldsymbol{x} = P\boldsymbol{b} \tag{3.107}$$
$$LU\boldsymbol{x} = P\boldsymbol{b} \tag{3.108}$$
$$L(U\boldsymbol{x}) = P\boldsymbol{b} \tag{3.109}$$

Com isso, resolvemos

$$L\boldsymbol{y} = P\boldsymbol{b} \tag{3.110}$$

donde obtemos

$$\boldsymbol{y} = (-11, -6, \overline{3}, 1, \overline{18}) \tag{3.111}$$

Então, resolvemos

$$U\boldsymbol{x} = y \tag{3.112}$$

donde obtemos a solução

$$\boldsymbol{x} = (-2, 1, -1). \tag{3.113}$$

```
# matriz
A = np.array([[-1., 2., -2.],
              [3., -4., 1.],
              [1., -5., 3.]])
b = np.array([6,-11,-10])

P, L, U = lup(A)

y = solSisTriaInf(L, P@b)
x = solSisTriaSup(U, y)
```

3.1.5 Exercícios

E.3.1.1. Seja a matriz

$$A = \begin{bmatrix} -1 & 2 & -2 \\ 3 & 4 & 1 \\ -4 & -5 & 3 \end{bmatrix} \tag{3.114}$$

a) Compute sua decomposição LU sem pivotamento parcial.

b) Compute sua decomposição LU com pivotamento parcial.

E.3.1.2. Use o Método da Decomposição LU para resolver o sistema linear

$$-x_1 + 2x_2 - 2x_3 = -1 \tag{3.120}$$
$$3x_1 - 4x_2 + x_3 = -4 \tag{3.121}$$
$$-4x_1 - 5x_2 + 3x_3 = 20 \tag{3.122}$$

usando LU.

E.3.1.3. Compute a decomposição LU da matriz

$$A = \begin{bmatrix} -1 & 0 & -2 & 1 \\ 3 & 0 & 0 & -2 \\ 1 & -1 & 0 & -1 \\ 0 & 2 & -3 & 0 \end{bmatrix}. \tag{3.123}$$

E.3.1.4. Use o Método da Decomposição LU para resolver o sistema linear

$$-x_1 - 2x_3 + x_4 = 1 \tag{3.127}$$
$$3x_1 - 2x_4 = -7 \tag{3.128}$$
$$x_1 - x_2 - x_4 = -3 \tag{3.129}$$
$$2x_2 - 3x_3 = -3 \tag{3.130}$$

E.3.1.5. A matriz de Vandermonde[30] é definida por

$$V(x_1, \ldots, x_n) = \begin{bmatrix} x_1^{n-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^{n-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ x_n^{n-1} & \cdots & x_n^2 & x_n & 1 \end{bmatrix} \tag{3.131}$$

Compute a decomposição LU com pivotamento parcial das matrizes de Vandermonde[31].

a) $V(1, 2, 3)$

b) $V(-2, -1, 0, 1)$

c) $V(0.1, 0.25, 0.5, 0.75)$

d) $V(-0.1, 0.5, 1, 2, 10)$

E.3.1.6. Use o Método da Decomposição LU para computar a matriz inversa de cada uma das seguintes matrizes:

a)

$$A = \begin{bmatrix} -1 & 2 & -2 \\ 3 & 4 & 1 \\ -4 & -5 & 3 \end{bmatrix} \tag{3.132}$$

b)

$$B = \begin{bmatrix} -1 & 0 & -2 & 1 \\ 3 & 0 & 0 & -2 \\ 1 & -1 & 0 & -1 \\ 0 & 2 & -3 & 0 \end{bmatrix}. \tag{3.133}$$

Respostas

E.3.1.1. a)

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 4 & 0 & 1 \end{bmatrix} \tag{3.115}$$

$$U = \begin{bmatrix} -1 & 2 & 2 \\ 0 & 10 & -5 \\ 0 & 0 & 4.5 \end{bmatrix} \tag{3.116}$$

b)

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.117}$$

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 0.25 & 1 & 0 \\ -0.75 & 7.692\text{e}{-2} & 1 \end{bmatrix} \tag{3.118}$$

$$U = \begin{bmatrix} -4 & -5 & 3 \\ 0 & 3.25 & -2.75 \\ 0 & 0 & 3.462 \end{bmatrix} \tag{3.119}$$

E.3.1.2. $x_1 = -3$, $x_2 = -1$, $x_3 = 1$

E.3.1.3.

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.124}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -0.\overline{3} & 0 & 1 & 0 \\ 0.\overline{3} & -0.5 & 0.75 & 1 \end{bmatrix} \tag{3.125}$$

$$U = \begin{bmatrix} 3 & 0 & 0 & -2 \\ 0 & 2 & -3 & 0 \\ 0 & 0 & -2 & 0.\overline{3} \\ 0 & 0 & 0 & -0.58\overline{3} \end{bmatrix} \tag{3.126}$$

E.3.1.4. $x = (-1, 0, 1, 2)$

E.3.1.5. Dica: A matriz de Vandermonde pode ser alocada com o seguinte código:

```
import numpy as np
x = np.array([1.,2,3])
n = x.size
V = np.ones((n,n))
for i in range(n-1):
  V[:,i] = x**(n-1-i)
print(V)
```

E.3.1.6. Dica: $AA^{-1} = I$.

a)

$$A^{-1} = \begin{bmatrix} -0.3778 & -0.0889, & -0.2222 \\ 0.2889 & 0.2444 & 0.1111 \\ -0.0222 & 0.2889 & 0.2222 \end{bmatrix} \tag{3.134}$$

b)

$$B^{-1} = \begin{bmatrix} 0.8571 & 1 & -1.1429 & -0.5714 \\ -0.4286 & 0 & -0.4286 & 0.2857 \\ -0.2857 & 0 & -0.2857 & -0.1429 \\ 1.2857 & 1. & -1.7143 & -0.8571 \end{bmatrix} \tag{3.135}$$

3.2 Norma e Número de Condicionamento

Nesta seção, fazemos uma rápida discussão sobre normas de vetores e matrizes e sobre o condicionamento de uma matriz.

3.2.1 Norma L^2

Norma de Vetores

A **norma L^2** de um dado vetor $\boldsymbol{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ é definida por

$$\|\boldsymbol{v}\| := \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}. \tag{3.136}$$

Lembrando que o **produto interno** de dois vetores $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^2$ é definido por

$$\boldsymbol{u} \cdot \boldsymbol{v} := u_1 v_1 + u_2 v_2 + \cdots + u_n v_n, \tag{3.137}$$

temos que

$$\|\boldsymbol{v}\| = \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}. \tag{3.138}$$

Exemplo 3.2.1. Sejam os vetores

$$\boldsymbol{u} = (1, -2, 3, -4), \tag{3.139}$$
$$\boldsymbol{v} = (-1, 2, 0, 1). \tag{3.140}$$

a)

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{v} &= u_1 v_1 + u_2 v_2 + \cdots + u_n v_n && (3.141)\\ &= 1 \cdot (-1) + (-2) \cdot 2 + 3 \cdot 0 + (-4) \cdot 1 && (3.142)\\ &= -1 - 4 + 0 - 4 && (3.143)\\ &= -9. && (3.144) \end{aligned}$$

```python
import numpy as np
u = np.array([1., -2., 3., -4.])
v = np.array([-1., 2., 0., 1.])
udv = np.dot(u,v)
print(f'\nu.v = {udv}')
```

b)

$$\|\boldsymbol{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \tag{3.145}$$
$$= \sqrt{(-1)^2 + 2^2 + 0^2 + 1^2} \tag{3.146}$$
$$= \sqrt{1 + 4 + 1} \tag{3.147}$$
$$= \sqrt{6} = 2.4495\text{e+}0. \tag{3.148}$$

```python
import numpy as np
import numpy.linalg as npla
v = np.array([-1., 2., 0., 1.])
norm_v = npla.norm(v)
print(f'\n||v|| = {norm_v}')
```

Proposição 3.2.1. (Propriedades da Norma para Vetores.) Dados os vetores $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ e um escalar $\lambda \in \mathbb{R}$, temos:

a) **Positividade**

$$\|\boldsymbol{v}\| \geq 0. \tag{3.149}$$
$$\|\boldsymbol{v}\| = 0 \Leftrightarrow \boldsymbol{v} = \boldsymbol{0}. \tag{3.150}$$

b) **Multiplicação por escalar**

$$\|\lambda\boldsymbol{v}\| = |\lambda| \cdot \|\boldsymbol{v}\|. \tag{3.151}$$

c) **Desigualdade de Cauchy[32]-Schwarz[33]**

$$\boldsymbol{u} \cdot \boldsymbol{v} \leq \|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\| \tag{3.152}$$

d) **Desigualdade triangular**

$$\|\boldsymbol{u} + \boldsymbol{v}\| \leq \|\boldsymbol{u}\| + \|\boldsymbol{v}\| \tag{3.153}$$

Demonstração. Sejam dados $\lambda \in \mathbb{R}$ e $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$.

a) Positividade.

Observa-se diretamente que $v_1^2 + v_2^2 + \cdots + v_n^2 \geq 0$. Então, como a raiz quadrada é uma função não-negativa, concluímos que

$$\begin{aligned} \|\boldsymbol{v}\| &:= \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \\ &\geq 0 \end{aligned} \tag{3.154}$$

No caso de $\boldsymbol{v} = \boldsymbol{0}$, temos $v_i = 0$, $i = 1, 2, \ldots, n$, donde

$$\begin{aligned} \|\boldsymbol{v}\| &= \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \\ \|\boldsymbol{0}\| &= \sqrt{0} = 0. \end{aligned} \tag{3.155}$$

Ou seja, se $\boldsymbol{v} = \boldsymbol{0}$, então $\|\boldsymbol{v}\| = 0$. Agora, se $\boldsymbol{v} \neq \boldsymbol{0}$, então tem-se $v_i \neq 0$ para algum $i = 1, 2, \ldots, n$. Logo, pela monotonicidade da função raiz quadrada, temos

$$v_1^2 + v_2^2 + \cdots + v_n^2 > 0 \tag{3.156}$$

$$\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} > \sqrt{0} \tag{3.157}$$

$$\|\boldsymbol{v}\| > 0. \tag{3.158}$$

Portanto, concluímos que $\|\boldsymbol{v}\| = 0$ se, e somente se, $\boldsymbol{v} = \boldsymbol{0}$.

b) Multiplicação por escalar.

Observamos que

$$\lambda \boldsymbol{v} = (\lambda v_1, \lambda v_2, \ldots, \lambda v_n). \tag{3.159}$$

Então, segue por cálculo direto que

$$\|\lambda \boldsymbol{v}\| = \sqrt{(\lambda v_1)^2 + (\lambda v_2)^2 + \cdots + (\lambda v_n)^2} \tag{3.160}$$

$$= \sqrt{\lambda^2 (v_1^2 + v_2^2 + \cdots + v_n^2)} \tag{3.161}$$

$$= \sqrt{\lambda^2} \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \tag{3.162}$$

$$= |\lambda| \|\boldsymbol{v}\|. \tag{3.163}$$

c) Desigualdade de Cauchy-Schwarz.

Sem perda de generalidade, se $\boldsymbol{v} = \mathbf{0}$, então a desigualdade é imediatamente satisfeita. Suponhamos, agora, que $\boldsymbol{v} \neq \mathbf{0}$. Para qualquer $\alpha \in \mathbb{R}$, temos

$$(\boldsymbol{u} + \alpha\boldsymbol{v}) \cdot (\boldsymbol{u} + \alpha\boldsymbol{v}) \geq 0 \tag{3.164}$$
$$\boldsymbol{u} \cdot \boldsymbol{u} + 2\alpha\boldsymbol{u} \cdot \boldsymbol{v} + \alpha^2\boldsymbol{v} \cdot \boldsymbol{v} \geq 0 \tag{3.165}$$
$$\|\boldsymbol{u}\|^2 + 2\alpha\boldsymbol{u} \cdot \boldsymbol{v} + \alpha^2\|\boldsymbol{v}\|^2 \geq 0 \tag{3.166}$$

O lado direito desta desigualdade é um polinômio quadrático em α. Para ser não-negativo para todo α, seu discriminante precisa ser não-positivo, i.e.

$$(2\boldsymbol{u} \cdot \boldsymbol{v})^2 - 4\|\boldsymbol{u}\|^2\|\boldsymbol{v}\|^2 \leq 0 \tag{3.167}$$
$$(\boldsymbol{u} \cdot \boldsymbol{v})^2 \leq \|\boldsymbol{u}\|^2\|\boldsymbol{v}\|^2 \tag{3.168}$$
$$\sqrt{(\boldsymbol{u} \cdot \boldsymbol{v})^2} \leq \sqrt{\|\boldsymbol{u}\|^2\|\boldsymbol{v}\|^2} \tag{3.169}$$
$$|\boldsymbol{u} \cdot \boldsymbol{v}| \leq \|\boldsymbol{u}\|\|\boldsymbol{v}\| \tag{3.170}$$

Donde, concluímos que

$$\boldsymbol{u} \cdot \boldsymbol{v} \leq \|\boldsymbol{u}\|\|\boldsymbol{v}\|. \tag{3.171}$$

d) Desigualdade triangular

Consulte o Exercício 3.2.6 para a demonstração da desigualdade triangular.

□

Exemplo 3.2.2. Vamos verificar a Desigualdade Triangular (3.153) para os vetores

$$\boldsymbol{u} = (1, -2, 3, -4), \tag{3.172}$$
$$\boldsymbol{v} = (-1, 2, 0, 1). \tag{3.173}$$

De fato, temos

$$\|\boldsymbol{u} + \boldsymbol{v}\| = \sqrt{(u_1 + v_1)^2 + (u_2 + v_2)^2 + (u_3 + v_3)^2 + (u_n + v_n)^2} \tag{3.174}$$

$$= \sqrt{0^2 + 0^2 + 3^2 + 3^2} \tag{3.175}$$
$$= \sqrt{18} \tag{3.176}$$
$$= 4.2426\text{e+}0 \tag{3.177}$$

e

$$\begin{aligned} \|\boldsymbol{u}\| + \|\boldsymbol{v}\| &= \sqrt{u_1^2 + u_2^2 + u_3^2 + u_4^2} \\ &\quad + \sqrt{v_1^2 + v_2^2 + v_3^2 + v_4^2} && (3.178) \\ &= \sqrt{1^2 + (-2)^2 + 3^2 + (-4)^2} \\ &\quad + \sqrt{(-1)^2 + 2^2 + 0^2 + 1^2} && (3.179) \\ &= \sqrt{30} + \sqrt{6} && (3.180) \\ &= 7.9267\text{e+}0 && (3.181) \end{aligned}$$

Ou seja, temos que

$$\|\boldsymbol{u} + \boldsymbol{v}\| \leq \|\boldsymbol{u}\| + \|\boldsymbol{v}\| \tag{3.182}$$

como esperado.

```
import numpy as np
u = np.array([1., -2., 3., -4.])
v = np.array([-1., 2., 0., 1.])
nupv = npla.norm(u+v)
nupnv = npla.norm(u) + npla.norm(v)
print('\nu.v <= ||u||.||v||?')
print(nupv <= nupnv)
```

Norma de Matrizes

A norma L^2 induzida de uma dada matriz real $A = [a_{ij}]_{i,j=1}^n$ é definida por

$$\|A\|_2 := \sup_{\boldsymbol{x} \in \mathbb{R}^n, \|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\|. \tag{3.183}$$

Pode-se mostrar que[34]

$$\|A\|_2 = \sqrt{\lambda_{max}(A^T A)}, \tag{3.184}$$

onde $\lambda_{max}(A^T A) := \max\{|\lambda|;\ \lambda \text{ é autovalor de } A^T A\}$.

Tendo em vista o grande custo computacional em se calcular a norma induzida, vamos trabalhar com a **norma de Frobenius** (ou norma Euclidiana[35])

$$\|A\| = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{i,j}|^2 \right)^{\frac{1}{2}} \tag{3.185}$$

Esta é uma generalização da norma euclidiana para vetores e é equivalente a norma L^2 induzida, i.e.

$$\|A\|_2 \leq \|A\| \leq \sqrt{n}\|A\|_2. \tag{3.186}$$

Exemplo 3.2.3. A matriz

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -2 & \pi & 4 \\ 7 & -5 & \sqrt{2} \end{bmatrix} \tag{3.187}$$

tem norma

$$\|A\| = 10.5768\text{e}{+0}. \tag{3.188}$$

```
import numpy as np
import numpy.linalg as npla
A = np.array([[1., -1., 2.],
              [-2., np.pi, 4.],
              [7., -5., np.sqrt(2)]])
norm_A = npla.norm(A)
print(norm_A)
```

Proposição 3.2.2. (Propriedades da Norma para Matrizes.) Dadas as matrizes reais A, B $n \times n$, um vetor $v \in \mathbb{R}^n$ e um escalar λ, temos

a) **Positividade**

$$\|A\| \geq 0 \tag{3.189}$$
$$\|A\| = 0 \Leftrightarrow A = \mathbf{0} \tag{3.190}$$

b) **Multiplicação por escalar**

$$\|\lambda A\| = |\lambda| \cdot \|A\| \tag{3.191}$$

c) **Desigualdade triangular**

$$\|A + B\| \leq \|A\| + \|B\| \tag{3.192}$$

d) **Sub-multiplicatividade**

$$\|AB\| \leq \|A\|\|B\| \tag{3.193}$$

e) **Norma da aplicação**

$$\|Av\| \leq \|A\|\|v\| \tag{3.194}$$

Demonstração. Consulte o Exercício 3.2.7. □

Exemplo 3.2.4. Vamos ver exemplos das propriedades d) e e) da norma de matriz.

a) Sub-multiplicatividade

$$\|I_n^2\| = \|I_n\| = \sqrt{n} \tag{3.195}$$
$$\|I\| \cdot \|I\| = \|I\|^2 = n \tag{3.196}$$

b) Norma da aplicação

$$\|I_n \boldsymbol{v}\| = \|\boldsymbol{v}\| \tag{3.197}$$
$$\|I_n\|\|v\| = \sqrt{n}\|\boldsymbol{v}\| \tag{3.198}$$

3.2.2 Número de Condicionamento

`Em revisão`

O número de condicionamento de uma matriz é uma medida referente a propagação de erros que ocorre da sua aplicação. Mais especificamente, assumamos que seja dada uma matriz invertível $A = [a_{ij}]_{i,j=1}^{n,n}$, um vetor $\boldsymbol{x} \in \mathbb{R}^n$ e uma perturbação $\boldsymbol{\delta_x} \in \mathbb{R}^n$. Além disso, sejam

$$\boldsymbol{y} = A\boldsymbol{x} \tag{3.199}$$
$$\boldsymbol{y} + \boldsymbol{\delta_y} = A(\boldsymbol{x} + \boldsymbol{\delta_x}). \tag{3.200}$$

Ou seja, $\boldsymbol{\delta_y}$ é a perturbação em $\boldsymbol{y}$ propagada da aplicação de A em $\boldsymbol{x}$ com perturbação $\boldsymbol{\delta_x}$.

Agora, podemos estimar a razão entre os erros relativos

$$e_{rel}(\boldsymbol{y}) := \|\boldsymbol{\delta_y}\|/\|\boldsymbol{y}\| \tag{3.201}$$
$$e_{rel}(\boldsymbol{x}) := \|\boldsymbol{\delta_x}\|/\|\boldsymbol{x}\| \tag{3.202}$$

da seguinte forma

$$\frac{\|\boldsymbol{y}\|/\|\boldsymbol{\delta_y}\|}{\|\boldsymbol{x}\|/\|\boldsymbol{\delta_x}\|} = \frac{\|\boldsymbol{y}\|}{\|\boldsymbol{\delta_y}\|}\frac{\|\boldsymbol{\delta_x}\|}{\|\boldsymbol{x}\|} \tag{3.203}$$
$$= \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{\delta_y}\|}\frac{\|A^{-1}\boldsymbol{\delta_y}\|}{\|\boldsymbol{x}\|} \tag{3.204}$$
$$\leq \frac{\|A\|\|\boldsymbol{x}\|\|A^{-1}\|\|\boldsymbol{\delta_y}\|}{\|\boldsymbol{\delta_y}\|\|\boldsymbol{x}\|} \tag{3.205}$$
$$= \|A\|\|A^{-1}\|. \tag{3.206}$$

Logo, temos a seguinte estimativa de propagação de erro

$$e_{rel}(\boldsymbol{y}) \leq \|A\|\|A^{-1}\|e_{rel}(\boldsymbol{x}). \tag{3.207}$$

Isto nos motiva a definir o **número de condicionamento** da matriz A por

$$\kappa(A) := \|A\|\|A^{-1}\|. \tag{3.208}$$

A matriz identidade tem o menor número de condicionamento que é

$$\kappa(I) = 1. \tag{3.209}$$

temos que $\kappa(A) \geq 1$ e quanto maior, mais mal condicionada é a matriz. Ou seja, quando maior $\kappa(A)$, maior é a propagação dos erros ao se computar $A\boldsymbol{x}$.

Exemplo 3.2.5. Estudamos os seguintes exemplos:

a) Matriz bem condicionada.

$$A = \begin{bmatrix} 1 & -1 & 2 \\ -2 & \pi & 4 \\ 7 & -5 & \sqrt{2} \end{bmatrix}, \tag{3.210}$$

cujo número de condicionamento é $\kappa(A) = 13.997$.

```
import numpy as np
import numpy.linalg as npla
A = np.array([[1., -1., 2.],
              [-2., np.pi, 4.],
              [7., -5., np.sqrt(2)]])
cond_A = npla.cond(A, 'fro')
print(f'cond(A) = {cond_A}')
```

b) Matriz mal condicionada.

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -2 \\ 10^5 & 10^{-4} & 10^5 \end{bmatrix}, \tag{3.211}$$

tem número de condicionamento

$$\kappa(B) = 1.5811 \times 10^{14}, \tag{3.212}$$

o que indica que B é uma matriz mal condicionada.

```
import numpy as np
import numpy.linalg as npla
A = np.array([[1., 0., 0.],
              [0., 0., -2.],
              [1e5, 1e-4, 1e5]])
cond_A = npla.cond(A, 'fro')
print(f'cond(A) = {cond_A:.4e}')
```

3.2.3 Exercícios

E.3.2.1. Compute a norma de cada um dos seguintes vetores:

a) $\boldsymbol{u} = (1, 0, -1)$

b) $\boldsymbol{v} = (1, -2, -1, 2)$

c) $\boldsymbol{w} = (1, 0, -1, 2, -2)$

E.3.2.2. Compute a norma de cada uma das seguintes matrizes:

a)

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{3.213}$$

b)

$$B = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 0 & 2 \\ 3 & -2 & 0 \end{bmatrix} \tag{3.214}$$

c)

$$C = \begin{bmatrix} 2 & 1 & -1 & 0 \\ -1 & 0 & 2 & 0 \\ 3 & -2 & 0 & -1 \\ 0 & 1 & 0 & 2 \end{bmatrix} \tag{3.215}$$

E.3.2.3. Compute o número de condicionamento de cada uma das seguintes matrizes:

a)

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{3.216}$$

b)

$$B = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 0 & 2 \\ 3 & -2 & 0 \end{bmatrix} \tag{3.217}$$

c)

$$C = \begin{bmatrix} 2 & 1 & -1 & 0 \\ -1 & 0 & 2 & 0 \\ 3 & -2 & 0 & -1 \\ 0 & 1 & 0 & 2 \end{bmatrix} \tag{3.218}$$

d)

$$D = \begin{bmatrix} 1\text{e}{+5} & 1\text{e}{-1} & 0 & 3\text{e}{-3} \\ 0 & -1\text{e}{-1} & 0 & -1\text{e}{-3} \\ 1\text{e}{+5} & 1\text{e}{-1} & 1\text{e}{-2} & 2\text{e}{-3} \\ 0 & -1\text{e}{-1} & 0 & 0 \end{bmatrix} \tag{3.219}$$

e)

$$E = \begin{bmatrix} 1 & 1\text{e}{-6} & 0 & 3\text{e}{-8} \\ -1\text{e}{-6} & -2\text{e}{-16} & -2\text{e}{-1} & 0 \\ 1\text{e}{+7} & 1\text{e}{+1} & 1 & 2\text{e}{-1} \\ 0 & -1\text{e}2 & 0 & 0 \end{bmatrix} \tag{3.220}$$

Qual das matriz acima é a mais mal condicionada? Justifique sua resposta.

E.3.2.4. Considere o seguinte sistema linear

$$10^{-12}x_1 + 20x_2 + 3x_3 = -1, \tag{3.221}$$
$$2.001x_1 + 10^{-5}x_2 + -x_3 = -2, \tag{3.222}$$
$$x_1 - 2x_2 - 0.1x_3 = 0.1. \tag{3.223}$$

a) Compute a norma do vetor dos termos constantes deste sistema.

b) Compute a norma matriz dos coeficientes deste sistema.

c) Compute o número de condicionamento da matriz dos coeficientes deste sistema.

E.3.2.5. Considere

$$A = \begin{bmatrix} 1\text{e}{+8} & 1 & 2 & -1 \\ 1\text{e}{-2} & 1\text{e}{-1} & 0 & 1\text{e}{-3} \\ 0 & 1 & 1\text{e}2 & 0 \\ 1\text{e}{-5} & 0 & 0 & 1\text{e}{-4} \end{bmatrix} \tag{3.224}$$

a) Compute $\kappa(A)$.

b) Aloque a matriz diagonal M, cujos elementos da diagonal sejam iguais aos da matriz A.

c) Verifique que o número de condicionamento de $M^{-1}A$ é melhor que o de A.

Análise Numérica

E.3.2.6. Mostre que a Desigualdade triangular

$$\|\boldsymbol{u} + \boldsymbol{v}\| \leq \|\boldsymbol{u}\| + \|\boldsymbol{v}\| \tag{3.225}$$

vale para quaisquer $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$.

E.3.2.7. Mostre a Proposição 3.2.2.

Respostas

E.3.2.1. a) $\|u\| = \sqrt{2}$; b) $\|v\| = 3.1623\text{e}{+0}$; c) $\|w\| = 3.1623\text{e}{+0}$;

E.3.2.2. a) $\|A\| = \sqrt{2}$; b) $\|B\| = 4.8990\text{e}{+0}$; c) $\|C\| = 5.4772\text{e}{+0}$

E.3.2.3. a) $\kappa(A) = 2$; b) $\kappa(B) = 4.6904\text{e}+0$; c) $\kappa(C) = 6.6291\text{e}+0$; d) $\kappa(D) = 4.6314\text{e}{+8}$ e) $\kappa(E) = 1.0000\text{e}{+15}$, a mais mal condicionada.

E.3.2.4. a) $2.2383\text{e}{+0}$; b) $2.0323\text{e}{+1}$; c) $3.5128\text{e}{+1}$

E.3.2.5. a) $\kappa(A) = 1.0\text{e+}12$; b) `M = np.diag(np.diag(A))`; c) $\kappa(M^{-1}A) = 4.0201\text{e+}0$

E.3.2.6. Dica: use a Desigualdade de Cauchy-Schwarz (3.152).

E.3.2.7. Dica: use as propriedades da norma de vetores.

3.3 Métodos de Jacobi e de Gauss-Seidel

Nesta seção, estudamos os métodos iterativos de Jacobi[36] e de Gauss[37]-Seidel[38] para a aproximação da solução de sistemas lineares

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{3.226}$$

onde $A = [a_{i,j}]_{i,j=1}^{n,n}$, $n \geq 1$, é uma dada matriz dos coeficientes, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ é o vetor das incógnitas e $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$ é um dado vetor dos termos constantes.

Métodos iterativos para sistemas lineares têm a forma

$$\boldsymbol{x}^{(0)} = \text{aprox. inicial}, \tag{3.227}$$

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}, \tag{3.228}$$

onde $T = [t_{i,j}]_{i,j=1}^{n,n}$ é a **matriz de iteração** e $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$ é o **vetor de iteração**.

3.3.1 Método de Jacobi

Consideramos a seguinte decomposição da matriz $A = L + D + U$:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \ldots & a_{3n} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \end{bmatrix} \tag{3.229}$$

$$= \underbrace{\begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & 0 \end{bmatrix}}_{L} \tag{3.230}$$

$$+ \underbrace{\begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ 0 & 0 & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}}_{D} \tag{3.231}$$

$$+ \underbrace{\begin{bmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}}_{U}. \tag{3.232}$$

Isto é, a matriz A decomposta como a soma de sua parte triangular inferior L, de sua diagonal D e de sua parte triangular superior U.

Desta forma, podemos reescrever o sistema como segue:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{3.233}$$

$$(L + D + U)\boldsymbol{x} = \boldsymbol{b} \tag{3.234}$$

$$(L + U)\boldsymbol{x} + D\boldsymbol{x} = \boldsymbol{b} \tag{3.235}$$

$$D\boldsymbol{x} = -(L + U)\boldsymbol{x} + \boldsymbol{b} \tag{3.236}$$

$$\boldsymbol{x} = -D^{-1}(L + U)\boldsymbol{x} + D^{-1}\boldsymbol{b}. \tag{3.237}$$

Ou seja, resolver o sistema $A\boldsymbol{x} = \boldsymbol{b}$ é equivalente a resolver o problema de ponto fixo

$$\boldsymbol{x} = T_J\boldsymbol{x} + \boldsymbol{c}_J, \tag{3.238}$$

onde $T_J = -D^{-1}(L + U)$ é chamada de **matriz de Jacobi** e $\boldsymbol{c}_J = D^{-1}\boldsymbol{b}$ é chamado de **vetor de Jacobi**.

Exemplo 3.3.1. Consideramos o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$\begin{aligned} A &= \begin{bmatrix} -4 & 2 & -1 \\ -2 & 5 & 2 \\ 1 & -1 & -3 \end{bmatrix}, \\ \boldsymbol{b} &= \begin{bmatrix} -11 \\ -7 \\ 0 \end{bmatrix}. \end{aligned} \tag{3.239}$$

Este sistema tem solução $\boldsymbol{x} = (2, -1, 1)$. Neste caso, temos a decomposição $A = L + D + U$ com

$$L = \begin{bmatrix} 0 & 0 & 0 \\ -2 & 0 & 0 \\ 1 & -1 & 0 \end{bmatrix}, \tag{3.240}$$

$$D = \begin{bmatrix} -4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & -3 \end{bmatrix}, \tag{3.241}$$

$$U = \begin{bmatrix} 0 & 2 & -1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.242}$$

Ainda, observamos que

$$T_J\boldsymbol{x} + \boldsymbol{c}_J = -D^{-1}(L+U)\boldsymbol{x} + D^{-1}\boldsymbol{b} \tag{3.243}$$

$$= \underbrace{\begin{bmatrix} 0 & 1/2 & 1/4 \\ 2/5 & 0 & -2/5 \\ 1/3 & -1/3 & 0 \end{bmatrix}}_{T_J} \underbrace{\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{bmatrix} 11/4 \\ -7/5 \\ 0 \end{bmatrix}}_{\boldsymbol{c}_J} \tag{3.244}$$

$$= \underbrace{\begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}}_{\boldsymbol{x}}. \tag{3.245}$$

conforme esperado.

```
1 import numpy as np
2 import numpy.linalg as npla
3
```

```
# sistema
A = np.array([[-4., 2., -1.],
              [-2., 5., 2.],
              [1., -1., -3.]])
b = np.array([-11., -7., 0.])

# A = L + D + U
L = np.tril(A, -1)
D = np.diag(np.diag(A))
U = np.triu(A, 1)

# matriz de Jacobi
T = -npla.inv(D) @ (L + U)
# vetor de Jacobi
c = npla.inv(D) @ b
```

A forma matricial das iterações de Jacobi consiste em

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \text{aprox. inicial}, \\ \boldsymbol{x}^{(k+1)} &= T_J \boldsymbol{x}^{(k)} + \boldsymbol{c}_J, \end{aligned} \tag{3.246}$$

onde $\boldsymbol{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ é a k-ésima aproximação da solução do sistema, $k = 0, 1, 2, \ldots$.

Equivalentemente, tem-se a forma algébrica das iterações de Jacobi

$$x_i^{(k+1)} = \frac{b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)}}{a_{ii}}, \tag{3.247}$$

com $i = 1, 2, \ldots, n$. Por não requerer as computações da matriz T_J e vetor $\boldsymbol{c}_J$, esta é a forma mais usada em implementações computacionais.

Exemplo 3.3.2. Consideramos o sistema $A\boldsymbol{x} = \boldsymbol{b}$ com

$$\begin{aligned} A &= \begin{bmatrix} -4 & 2 & -1 \\ -2 & 5 & 2 \\ 1 & -1 & -3 \end{bmatrix}, \\ \boldsymbol{b} &= \begin{bmatrix} -11 \\ -7 \\ 0 \end{bmatrix}. \end{aligned} \tag{3.248}$$

Aplicando o método de Jacobi com aproximação inicial $\boldsymbol{x}^{(1)} = (0, 0, 0)$ obtemos os resultados da Tabela 3.1.

k	$\boldsymbol{x}^{(k)}$	$\|A\boldsymbol{x}^{(k)} - \boldsymbol{b}\|$
0	$(0, 0, 0)$	2.4e+1
1	$(2.75, -1.4, 0)$	7.4e+0
2	$(2.05, -0.3, 1.38)$	4.6e+0
3	$(2.25, -1.13, 0.78)$	2.2e+0
4	$(1.99, -0.81, 1.13)$	1.4e+0
5	$(2.06, -1.06, 0.93)$	6.9e−1
⋮	⋮	⋮
29	$(2, -1, 1)$	5.7e−7

Tabela 3.1: Resultados referentes ao Exemplo 3.3.2.

Código 3.5: jacobi.py

```
import numpy as np
import numpy.linalg as npla

def jacobi(A, b, x0, maxiter = 100,
           tol=4.9e-8, atol=4.9e-8):

  n = b.size
  info = -1

  x = np.empty_like(x0)
  nres = npla.norm(A@x - b)
  print(f'\n{0}: {x0}, {nres:.1e}')

  # iterações
  for k in range(maxiter):
    for i in  range(n):
      x[i] = b[i]
      # x[i] -= np.dot(A[i,:i], x0[:i])
      for j in range(i):
          x[i] -= A[i,j]*x0[j]
      # x[i] -= np.dot(A[i,i+1:], x0[i+1:])
      for j in range(i+1,n):
```

```
23            x[i] -= A[i,j]*x0[j]
24        x[i] /= A[i,i]
25      # critério de parada
26      nres = npla.norm(A@x - b)
27      print(f'{k+1}: {x}, {nres:.1e}')
28      if (nres <= max(tol*npla.norm(b), atol)):
29        info = 0
30        break
31      x0 = x.copy()
32
33    return x, info
```

A aplicação de Jacobi pode, então, ser feita como segue:

```
1 # sistema
2 A = np.array([[-4., 2., -1.],
3                [-2., 5., 2.],
4                [1., -1., -3.]])
5
6 b = np.array([-11., -7., 0.])
7
8 x0 = np.zeros_like(b)
9 x, info = jacobi(A, b, x0)
```

3.3.2 Método de Gauss-Seidel

Como acima, começamos considerando um sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ e a decomposição $A = L + D + U$, onde L é a parte triangular inferior de A, D é sua parte diagonal e U sua parte triangular superior. Então, observamos que

$$A\boldsymbol{x} = \boldsymbol{b} \tag{3.249}$$
$$(L + D + U)\boldsymbol{x} = \boldsymbol{b} \tag{3.250}$$
$$(L + D)\boldsymbol{x} = -U\boldsymbol{x} + \boldsymbol{b} \tag{3.251}$$
$$\boldsymbol{x} = -(L + D)^{-1}U\boldsymbol{x} + (L + D)^{-1}\boldsymbol{b}. \tag{3.252}$$

Isto nos leva a **forma matricial iteração de Gauss-Seidel**

$$\begin{aligned} \boldsymbol{x}^{(1)} &= \text{aprox. inicial}, \\ \boldsymbol{x}^{(k+1)} &= T_G\boldsymbol{x}^{(k)} + \boldsymbol{c}_G, \end{aligned} \tag{3.253}$$

onde

$$T_G = -(L+D)^{-1}U, \tag{3.254}$$
$$\boldsymbol{c}_G = (L+D)^{-1}\boldsymbol{b}, \tag{3.255}$$

são a matriz e o vetor de Gauss-Seidel.

Equivalentemente e mais adequada para implementação computacional, temos a **forma algébrica da iteração de Gauss-Seidel**

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}, \tag{3.256}$$

para $i = 1, 2, \ldots, n$.

Exemplo 3.3.3. Consideramos o sistema $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} -4 & 2 & -1 \\ -2 & 5 & 2 \\ 1 & -1 & -3 \end{bmatrix}, \tag{3.257}$$

$$\boldsymbol{b} = \begin{bmatrix} -11 \\ -7 \\ 0 \end{bmatrix}. \tag{3.258}$$

Aplicando o método de Gauss-Seidel com aproximação inicial $\boldsymbol{x}^{(1)} = (0, 0, 0)$ obtemos os resultados da Tabela 3.2.

Tabela 3.2: Resultados referentes ao Exemplo 3.3.3.

k	$\boldsymbol{x}^{(k)}$	$\|A\boldsymbol{x}^{(k)} - \boldsymbol{b}\|$
0	$(0, 0, 0)$	2.4e+1
1	$(2.75, -0.3, 1.02)$	2.6e+0
2	$(2.35, -0.87, 1.07)$	1.2e+0
3	$(2.05, -1.01, 1.02)$	2.5e−1
4	$(1.99, -1.01, 1)$	4.0e−2
5	$(1.99, -1, 1)$	2.0e−2
⋮	⋮	⋮
13	$(2, -1, 1)$	2.3e−7

Código 3.6: gs.py

```
import numpy as np
import numpy.linalg as npla

def gs(A, b, x0, maxiter = 100,
       tol=4.9e-8, atol=4.9e-8):

  n = b.size
  info = -1

  x = np.empty_like(x0)
  nres = npla.norm(A@x - b)
  print(f'\n{0}: {x0}, {nres:.1e}')

  # iterações
  for k in range(maxiter):
    for i in  range(n):
      x[i] = b[i]
      # x[i] -= np.dot(A[i,:i], x[:i])
      for j in range(i):
          x[i] -= A[i,j]*x[j]
      # x[i] -= np.dot(A[i,i+1:], x0[i+1:])
      for j in range(i+1,n):
          x[i] -= A[i,j]*x0[j]
      x[i] /= A[i,i]
    # critério de parada
    nres = npla.norm(A@x - b)
    print(f'{k+1}: {x}, {nres:.1e}')
    if (nres <= max(tol*npla.norm(b), atol)):
      info = 0
      break
    x0 = x.copy()

  return x, info
```

3.3.3 Análise Numérica

Observamos que ambos os métodos de Jacobi e de Gauss-Seidel consistem de iterações da forma

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}, \tag{3.259}$$

para $k = 1, 2, \ldots$, com $x^{(0)}$ uma aproximação inicial dada, T e c a matriz e o vetor de iteração, respectivamente. O seguinte teorema nos fornece uma condição suficiente e necessária para a convergência de tais métodos.

Teorema 3.3.1. *Para qualquer $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$, temos que a sequência $\{\boldsymbol{x}^{(k)}\}_{k=0}^{\infty}$ dada por*

$$\boldsymbol{x}^{(k+1)} = T\boldsymbol{x}^{(k)} + \boldsymbol{c}, \tag{3.260}$$

converge para a solução única de $\boldsymbol{x} = T\boldsymbol{x} + \boldsymbol{c}$ se, e somente se, $\rho(T) < 1$[39].

Demonstração. Veja [2, Cap. 7, Sec. 7.3]. □

Observação 3.3.1. ((Taxa de convergência.)) Para uma iteração da forma (3.259), temos a seguinte estimativa

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}\| \approx \rho(T)^{k+1}\|\boldsymbol{x}^{(0)} - \boldsymbol{x}\|^{\mathbf{1}}, \tag{3.261}$$

onde $\boldsymbol{x}$ é a solução de $\boldsymbol{x} = T\boldsymbol{x} + \boldsymbol{c}$.

Exemplo 3.3.4. Consideramos o sistema $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} -4 & 2 & -1 \\ -2 & 5 & 2 \\ 1 & -1 & -3 \end{bmatrix}, \tag{3.262}$$

$$\boldsymbol{b} = \begin{bmatrix} -11 \\ -7 \\ 0 \end{bmatrix}. \tag{3.263}$$

Nos Exemplos 3.3.2 e 3.3.3 vimos que ambos os métodos de Jacobi e de Gauss-Seidel são convergentes para este sistema. Este convergiu aproximadamente duas vezes mais rápido que esse. Isto é confirmado pelos raios espectrais das respectivas matrizes de iteração

$$\rho(T_J) \approx 0.56, \tag{3.264}$$

$$\rho(T_G) \approx 0.26. \tag{3.265}$$

```python
import numpy as np
import numpy.linalg as npla

# matriz dos coefs
A = np.array([[-4., 2., -1.],
              [-2., 5., 2.],
              [1., -1., -3.]])

# A = L + D + U
L = np.tril(A, -1)
D = np.diag(np.diag(A))
U = np.triu(A, 1)

# matriz de Jacobi
TJ = -npla.inv(D) @ (L + U)
rho_TJ = max(np.abs(npla.eigvals(TJ)))
print(f'rho(T_J) = {rho_TJ:.2f}')

# matriz de Gauss-Seidel
TG = -npla.inv(L+D) @ U
rho_TG = max(np.abs(npla.eigvals(TG)))
print(f'rho(T_G) = {rho_TG:.2f}')
```

Observação 3.3.2. (Matriz estritamente diagonal dominante.) Pode-se mostrar que se A é uma matriz estritamente diagonal dominante, i.e. se

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|, \tag{3.266}$$

para todo $i = 1, 2, \ldots, n$, então ambos os métodos de Jacobi e de Gauss-Seidel são convergentes.

3.3.4 Exercícios

E.3.3.1. Considere o seguinte sistema linear

$$-4x_1 + x_2 + x_3 - x_4 = -1 \tag{3.267}$$

$$5x_2 - x_3 + 2x_4 = 3 \tag{3.268}$$
$$-x_1 + 4x_3 - 2x_4 = -2 \tag{3.269}$$
$$x_1 - x_2 - 5x_4 = 1 \tag{3.270}$$

Compute a aproximação $\boldsymbol{x}^{(5)}$ obtida da aplicação do Método de Jacobi com aproximação inicial $\boldsymbol{x}^{(0)} = (1, 1, -1, -1)$. Também, compute $\|A\boldsymbol{x}^{(5)} - b\|$.

E.3.3.2. Considere o sistema linear do Exercício . Compute a aproximação $\boldsymbol{x}^{(5)}$ obtida da aplicação do Método de Gauss-Seidel com aproximação inicial $\boldsymbol{x}^{(0)} = (1, 1, -1, -1)$. Também, compute $\|A\boldsymbol{x}^{(5)} - b\|$.

E.3.3.3. Verifique que o Método de Jacobi, com $\boldsymbol{x}^{(0)} = \boldsymbol{0}$, é divergente para o sistema $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} -3 & 0 & 1 & -1 \\ 1 & -1 & 4 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & -1 & 1 & -5 \end{bmatrix}, \tag{3.271}$$

$$b = \begin{bmatrix} -2 \\ -11 \\ 2 \\ -19 \end{bmatrix} \tag{3.272}$$

Então, escreva um sistema equivalente para o qual o Método de Jacobi seja convergente para qualquer escolha de aproximação inicial.

E.3.3.4. Considere o sistema linear

$$\begin{aligned} -x_1 + 2x_2 - 2x_3 &= 6 \\ 3x_1 - 4x_2 + x_3 &= -11 \\ x_1 - 5x_2 + 3x_3 &= -10. \end{aligned} \tag{3.275}$$

Empregando a aproximação inicial $\boldsymbol{x}^{(0)} = \boldsymbol{0}$, compute a solução com o:

a) Método de Jacobi.

b) Método de Gauss-Seidel.

E.3.3.5. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.276}$$

$$b = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0 \end{bmatrix}. \tag{3.277}$$

Compute a solução empregando o:

a) Método de Jacobi.

b) Método de Gauss-Seidel.

E.3.3.6. Considere o seguinte sistema de equações

$$\begin{aligned} 2.1x_1 - x_2 + 0.9x_3 &= -1.3 \\ 2x_1 + 2x_2 + 2.1x_3 &= 2.1 \\ -1.2x_1 - x_2 + 2x_3 &= -\pi \end{aligned} \tag{3.278}$$

Usando a aproximação inicial $\boldsymbol{x}^{(0)} = \mathbf{0}$, verifique que o método de Jacobi não converge para sua solução, enquanto que o método de Gauss-Seidel converge. Por quê?

E.3.3.7. Considere o seguinte sistema de equações

$$\begin{aligned} 1.1x_1 + 2x_2 - 1.9x_3 &= -1.3 \\ x_1 + x_2 + x_3 &= 2.1 \\ 2.1x_1 + 1.9x_2 + x_3 &= -\pi \end{aligned} \tag{3.279}$$

Usando a aproximação inicial $\boldsymbol{x}^{(0)} = \mathbf{0}$, verifique que o método de Jacobi converge para sua solução, enquanto que o método de Gauss-Seidel não converge. Por quê?

Respostas

E.3.3.1. $\boldsymbol{x}^{(5)} = (0.325164, 0.593096, -0.546128, -0.253043)$; $\|A\boldsymbol{x}^{(5)} - b\| = 7.2\text{e}{-3}$

E.3.3.2. $\boldsymbol{x}^{(5)} = (0.325208, 0.592417, -0.545397, -0.253442)$; $\|A\boldsymbol{x}^{(5)} - b\| = 7.1\text{e}{-4}$

E.3.3.3.

$$A = \begin{bmatrix} -3 & 0 & 1 & -1 \\ 0 & 2 & 1 & 0 \\ 1 & -1 & 4 & 0 \\ 0 & -1 & 1 & -5 \end{bmatrix}, \tag{3.273}$$

$$b = \begin{bmatrix} -2 \\ 2 \\ -11 \\ -19 \end{bmatrix} \tag{3.274}$$

E.3.3.4. a) divergente. b) $(-2, 1, -1)$

E.3.3.5. $\boldsymbol{x} = (0, 0.75, 1, 0.75, 0)$

E.3.3.6. $\rho(T_J) = 1.12\text{e}{+0} > 1$, $\rho(T_G) = 5.48\text{e}{-1} < 1$.

E.3.3.7. $\rho(T_J) = 8.5\text{e}{-1} < 1$, $\rho(T_G) = 1.95\text{e}{+0} > 1$.

3.4 Método do Gradiente

Começamos observando que se A é uma matriz $n \times n$ positiva definida[40], temos que $\boldsymbol{x} \in \mathbb{R}^n$ é solução de

$$A\boldsymbol{x} = \boldsymbol{b} \tag{3.280}$$

se, e somente se, $\boldsymbol{x}$ é solução do seguinte problema de minimização

$$\min_{\boldsymbol{y}\in\mathbb{R}^n}\left\{f(\boldsymbol{y}) := \frac{1}{2}\boldsymbol{y}^T A\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{b}\right\}. \tag{3.281}$$

De fato, sejam $\boldsymbol{x}$ a solução de (3.280) e $\boldsymbol{y}$ a solução de (3.281), então

$$f(\boldsymbol{y}) := \frac{1}{2}\boldsymbol{y}^T A\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{b} \tag{3.282}$$

$$= \frac{1}{2}\boldsymbol{y}^T A\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{b} + \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - +\frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} \tag{3.283}$$

$$= \frac{1}{2}(\boldsymbol{x}-\boldsymbol{y})^T A(\boldsymbol{x}-\boldsymbol{y}) - \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} \tag{3.284}$$

O segundo termo é independente de $\boldsymbol{y}$ e, como A é positiva definida, temos

$$\frac{1}{2}(\boldsymbol{x}-\boldsymbol{y})^T A(\boldsymbol{x}-\boldsymbol{y}) \geq 0. \tag{3.285}$$

Logo, o mínimo de f ocorre quando $\boldsymbol{x}-\boldsymbol{y}=\boldsymbol{0}$, i.e. $\boldsymbol{y}=\boldsymbol{x}$.

A iteração do Método do Gradiente tem a forma

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \text{aprox. inicial},\\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{d}^{(k)}, \end{aligned} \tag{3.286}$$

para $k = 0, 1, 2, \ldots$, onde $\alpha_k > 0$ é o **tamanho do passo** e $\boldsymbol{d}^{(k)}$ é o **vetor direção de busca**.

Para escolhermos a direção $\boldsymbol{d}^{(k)}$, tomamos a fórmula de Taylor de f em torno da aproximação $\boldsymbol{x}^{(k)}$

$$f\left(\boldsymbol{x}^{(k+1)}\right) = f\left(\boldsymbol{x}^{(k)}\right) + \alpha^{(k)}\nabla f\left(\boldsymbol{x}^{(k)}\right)\cdot\boldsymbol{d}^{(k)} + \epsilon, \tag{3.287}$$

onde ∇f denota o gradiente de f, i.e.

$$\nabla f(\boldsymbol{x}) := \left(\frac{\partial f}{\partial x_1}(\boldsymbol{x}), \frac{\partial f}{\partial x_2}(\boldsymbol{x}), \ldots, \frac{\partial f}{\partial x_n}(\boldsymbol{x})\right) \tag{3.288}$$

$$\nabla f(\boldsymbol{x}) = A\boldsymbol{x}^{(k)} - \boldsymbol{b}. \tag{3.289}$$

De , segue que se

$$\nabla f\left(\boldsymbol{x}^{(k)}\right)\cdot\boldsymbol{d}^{(k)} < 0, \tag{3.290}$$

então $f\left(\boldsymbol{x}^{(k+1)}\right) < f\left(\boldsymbol{x}^{(k)}\right)$, para $\alpha^{(k)}$ suficientemente pequeno. Em particular, podemos escolher

$$\boldsymbol{d}^{(k)} = -\nabla f\left(\boldsymbol{x}^{(k)}\right), \tag{3.291}$$

se $\nabla f(\boldsymbol{x}^{(k)}) \neq 0$.

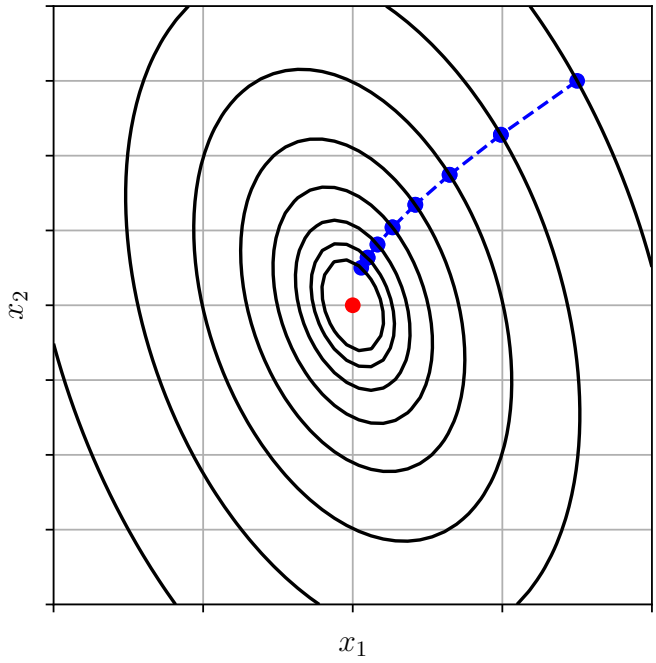

Figura 3.1: Iterações do Método do Gradiente para sistemas lineares. Linha: $\|b - Ax\|$. Pontos: $\boldsymbol{x}^{(k)}$.

Do exposto acima, temos a **iteração do Método do Gradiente**

$$\begin{aligned} \boldsymbol{x}^{(0)} &= \text{aprox. inicial} \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{r}^{(k)} \end{aligned} \tag{3.292}$$

com $k = 0, 1, 2, \ldots$, onde $\boldsymbol{r}^{(k)}$ é o **resíduo**

$$\boldsymbol{r}^{(k)} = \boldsymbol{b} - A\boldsymbol{x}^{(k)}. \tag{3.293}$$

Exemplo 3.4.1. Consideramos o sistema $Ax = b$ com

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.294}$$

$$b = \begin{bmatrix} -3 \\ 2 \\ 2 \\ -3 \end{bmatrix}. \tag{3.295}$$

Na Tabela 3.3 temos os resultados do emprego do método do gradiente com $\boldsymbol{x}^{(1)} = (0, 0, 0, 0)$ e com passo constante $\alpha^{(k)} \equiv 0.5$.

Tabela 3.3: Resultados referentes ao Exemplo 3.4.1.

k	$\boldsymbol{x}^{(k)}$	$\|A\boldsymbol{x}^{(k)} - \boldsymbol{b}\|$
0	$(0.0, 0.0, 0.0, 0.0)$	5.1e+0
1	$(-1.5, 1.0, 1.0, -1.5)$	1.6e+0
2	$(-1.0, 0.8, 0.8, -1.0)$	5.0e−1
3	$(-1.1, 0.9, 0.9, -1.1)$	1.8e−1
4	$(-1.1, 0.9, 0.9, -1.1)$	8.8e−2
5	$(-1.1, 0.9, 0.9, -1.1)$	6.2e−2
6	$(-1.0, 0.9, 0.9, -1.0)$	4.9e−2
7	$(-1.0, 0.9, 0.9, -1.0)$	4.0e−2
8	$(-1.0, 0.9, 0.9, -1.0)$	3.2e−2
9	$(-1.0, 1.0, 1.0, -1.0)$	2.6e−2
10	$(-1.0, 1.0, 1.0, -1.0)$	2.1e−2

Código 3.7: mg.py

```
1 import numpy as np
2 import numpy.linalg as npla
3
4 def mg(A, b, x0, alpha=1e-2,
5        maxiter=100, atol=1.49e-8, rtol=1.49e-8):
6
7   n = b.size
8   x = x0.copy()
```

```
  res = b - A@x
  nres = npla.norm(res)
  print(f'0: {x}, nres = {nres:.1e}')
  info = -1
  for k in range(maxiter):
    x = x + alpha*res

    res = b - A@x
    nres = npla.norm(res)

    print(f'{k+1}: {x}, nres = {nres:.1e}')

    if (nres <= max(atol, rtol*npla.norm(b))):
        info = 0
        break

  return x, info

# matriz coefs
A = np.array([[2., -1., 0., 0.],
              [-1., 2., -1., 0.],
              [0., -1., 2., -1.],
              [0., 0., -1., 2.]])
# vetor constante
b = np.array([-3., 2., 2., -3.])

# aprox. inicial
x0 = np.zeros_like(b)

x, info = mg(A, b, x0, alpha=0.5)
```

3.4.1 Escolha do Passo

Para a escolha do passo, podemos usar o **Método da Pesquisa Linear**. A ideia é escolher o passo $\alpha^{(k)}$ tal que

$$f\left(\boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{r}^{(k)}\right) = \min_{\alpha>0} f\left(\boldsymbol{x}^{(k)} + \alpha\boldsymbol{r}^{(k)}\right). \tag{3.296}$$

Observando que $f(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{r}^{(k)})$ é função apenas de α, temos que seu mínimo ocorre em seu ponto crítico, i.e.

$$\frac{d}{d\alpha} f\left(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{r}^{(k)}\right) = 0 \tag{3.297}$$

$$\nabla f\left(\boldsymbol{x}^{(k+1)}\right) \cdot \frac{d}{d\alpha}\left(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{r}^{(k)}\right) = 0 \tag{3.298}$$

$$\nabla f(\boldsymbol{x}^{(k+1)}) \cdot \boldsymbol{r}^{(k)} = 0, \tag{3.299}$$

$$\left[A\left(\boldsymbol{x}^{(k)} + \alpha^{(k)} \boldsymbol{r}^{(k)}\right) - b\right] \cdot \boldsymbol{r}^{(k)} = 0, \tag{3.300}$$

$$\left(A\boldsymbol{x}^{(k)} - \boldsymbol{b}\right) \cdot \boldsymbol{r}^{(k)} + \alpha^{(k)} \boldsymbol{r}^{(k)} \cdot A\boldsymbol{r}^{(k)} = 0, \tag{3.301}$$

donde

$$\alpha^{(k)} = \frac{\boldsymbol{r}^{(k)} \cdot \boldsymbol{r}^{(k)}}{\boldsymbol{r}^{(k)} \cdot A\boldsymbol{r}^{(k)}}. \tag{3.302}$$

Exemplo 3.4.2. Consideramos o sistema $Ax = b$ com

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.303}$$

$$b = \begin{bmatrix} -3 \\ 2 \\ 2 \\ -3 \end{bmatrix}. \tag{3.304}$$

Na Tabela 3.4 temos os resultados do emprego do método do gradiente com $\boldsymbol{x}^{(1)} = (0, 0, 0, 0)$ e com passo escolhido conforme (3.302).

Tabela 3.4: Resultados referentes ao Exemplo **??**.

k	$\boldsymbol{x}^{(k)}$	$\alpha^{(k)}$	$\|A\boldsymbol{x}^{(k)} - \boldsymbol{b}\|$
0	$(0.0, 0.0, 0.0, 0.0)$	3.8e−1	5.1e+0
1	$(-1.1, 0.8, 0.8, -1.1)$	2.6e+0	1.5e−1
2	$(-1.0, 1.0, 1.0, -1.0)$	3.8e−1	3.0e−2
3	$(-1.0, 1.0, 1.0, -1.0)$	2.6e+0	8.8e−4
4	$(-1.0, 1.0, 1.0, -1.0)$	3.8e+0	1.8e−4
5	$(-1.0, 1.0, 1.0, -1.0)$	2.6e+0	5.2e−6

```
import numpy as np
import numpy.linalg as npla

def mg_pl(A, b, x0,
          maxiter=100, atol=1.49e-8, rtol=1.49e-8):

  n = b.size
  x = x0.copy()
  res = b - A@x
  alpha = np.dot(res, res)/np.dot(res, A@res)
  nres = npla.norm(res)
  print(f'0: {x}, alpha = {alpha}, nres = {nres:.1
e}')
  info = -1
  for k in range(maxiter):
    x = x + alpha*res

    res = b - A@x
    alpha = np.dot(res, res)/np.dot(res, A@res)
    nres = npla.norm(res)

    print(f'{k+1}: {x}, alpha = {alpha}, nres = {
nres:.1e}')

    if (nres <= max(atol, rtol*npla.norm(b))):
      info = 0
      break

  return x, info

# matriz coefs
A = np.array([[2., -1., 0., 0.],
              [-1., 2., -1., 0.],
              [0., -1., 2., -1.],
              [0., 0., -1., 2.]])
# vetor constante
b = np.array([-3., 2., 2., -3.])

```

```
# aprox. inicial
x0 = np.zeros_like(b)

x, info = mg_pl(A, b, x0)
```

3.4.2 Exercícios

E.3.4.1. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.305}$$

$$b = \begin{bmatrix} 5 \\ -7 \\ 6 \\ -1 \end{bmatrix}. \tag{3.306}$$

Por tentativa e erro, encontre um valor para α tal que o Método do Gradiente converge para solução do sistema em menos de 80 iterações. Use

$$\boldsymbol{x}^{(0)} = \boldsymbol{0} \tag{3.307}$$

como aproximação inicial e assuma o critério de parada

$$\|\boldsymbol{r}\| \leq \max\{\texttt{tol}, \texttt{tol}\|\boldsymbol{b}\|\}, \tag{3.308}$$

onde $\boldsymbol{r}$ é o resíduo do sistema e $\texttt{tol} = 1.49\text{e}{-8}$.

E.3.4.2. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 2 & -1 & 1 & 0 \\ -1 & 2 & -1 & 0 \\ 1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.309}$$

$$b = \begin{bmatrix} -8 \\ 9 \\ -10 \\ 5 \end{bmatrix}. \tag{3.310}$$

Por tentativa e erro, encontre um valor para α tal que o Método do Gradiente converge para solução do sistema em menos de 50 iterações. Use

$$\boldsymbol{x}^{(0)} = \mathbf{0} \tag{3.311}$$

como aproximação inicial e assuma o critério de parada

$$\|\boldsymbol{r}\| \leq \max\{\texttt{tol}, \texttt{tol}\|\boldsymbol{b}\|\}, \tag{3.312}$$

onde $\boldsymbol{r}$ é o resíduo do sistema e $\texttt{tol} = 1.49\text{e}{-8}$.

E.3.4.3. Considere o sistema linear dado no Exercício 3.4.1. Utilizando a mesma aproximação inicial e tolerância, aplique o Método do Gradiente com Pesquisa Linear. Quantas iterações são necessárias até a convergência e qual o valor médio de $\alpha^{(k)}$ utilizado durante as iterações?

E.3.4.4. Considere o sistema linear dado no Exercício 3.4.2. Utilizando a mesma aproximação inicial e tolerância, aplique o Método do Gradiente com Pesquisa Linear. Quantas iterações são necessárias até a convergência e qual o valor médio de $\alpha^{(k)}$ utilizado durante as iterações?

E.3.4.5. Considere o problema de Laplace

$$-u_{xx} = 2, \quad 0 < x < 1, \tag{3.313}$$

$$u(0) = u(1) = 0. \tag{3.314}$$

A discretização pelo Método das Diferenças Finitas em uma malha uniforme $x_i = ih$, $i = 0.1, \ldots, n$, $h = 1/(n-1)$, leva ao seguinte sistema linear

$$u_1 = 0 \tag{3.315}$$

$$-\frac{1}{h^2}u_{i-1} + \frac{2}{h^2}u_i - \frac{1}{h^2}u_{i+1} = 2, \tag{3.316}$$

$$u_n = 0 \tag{3.317}$$

onde $u_i \approx u(x_i)$. Com $u^{(0)} = \mathbf{0}$, aplique o Método do Gradiente com Pesquisa Linear para computar a solução deste sistema quando $n = 10, 20, 40, 80$. Quantas iterações são necessárias para obter-se a convergência do método com critério de convergência

$$\|\boldsymbol{r}\| \leq \texttt{tol}, \tag{3.318}$$

onde, $\boldsymbol{r} := \boldsymbol{b} - A\boldsymbol{x}$ é o resíduo e $\texttt{tol} = 1e-4$.

Análise Numérica

E.3.4.6. Sendo f dada em (3.281), mostre que

$$\nabla f(\boldsymbol{x}) = -\boldsymbol{r}, \tag{3.319}$$

com, $\boldsymbol{r} := b - A\boldsymbol{x}$ o resíduo do sistema $A\boldsymbol{x} = \boldsymbol{b}$.

Respostas

E.3.4.1. $alpha = 4.9\text{e}{-1}$, $\boldsymbol{x} = (2.0, -1.0, 3.0, 1.0)$

E.3.4.2. $alpha = 3.6\text{e}{-1}$, $\boldsymbol{x} = (-2.0, 3.0, -1.0, 2.0)$

E.3.4.3. 75, $\overline{\alpha} = 5.0\text{e}{-1}$

E.3.4.4. 35, $\overline{\alpha} = 3.7\text{e}{-1}$

E.3.4.5.

n	k
10	214
20	918
40	3840
80	15910

3.5 Método do Gradiente Conjugado

O Método do Gradiente Conjugado é uma variação do Método do Gradiente (consulte Seção 3.4). Ambos são aplicáveis a sistemas lineares $A\boldsymbol{x} = \boldsymbol{b}$, A matriz positiva definida, e as iterações têm a forma

$$\boldsymbol{x}^{(0)} = \text{aprox. inicial}, \tag{3.320}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}. \tag{3.321}$$

onde $\boldsymbol{d}^{(k)}$ é o vetor direção na k-ésima iterada. No Método do Gradiente, o vetor direção é $\boldsymbol{d}^{(k)} = \boldsymbol{r}^{(k)} := \boldsymbol{b} - A\boldsymbol{x}^{(k)}$. Aqui, as direções escolhidas são

tomadas conjugadas duas-a-duas[41]. Isto nos leva a **iteração do Método do Gradiente Conjugado**:

$$\boldsymbol{x}^{(0)} = \text{aprox. inicial}, \tag{3.322}$$
$$\boldsymbol{d}^{(0)} = \boldsymbol{r}^{(0)}, \tag{3.323}$$

$$\alpha^{(k)} = \frac{\boldsymbol{r}^{(k)} \cdot \boldsymbol{r}^{(k)}}{\boldsymbol{d}^{(k)} \cdot A\boldsymbol{d}^{(k)}}, \tag{3.324}$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}, \tag{3.325}$$

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \alpha^{(k)} A\boldsymbol{d}^{(k)}, \tag{3.326}$$
$$\beta^{(k)} = \frac{\boldsymbol{r}^{(k+1)} \cdot \boldsymbol{r}^{(k+1)}}{\boldsymbol{r}^{(k)}\boldsymbol{r}^{(k)}}, \tag{3.327}$$
$$\boldsymbol{d}^{(k+1)} = \boldsymbol{r}^{(k+1)} + \beta^{(k)}\boldsymbol{d}^{(k)}, \tag{3.328}$$
$$\tag{3.329}$$

para $k = 0, 1, \ldots$, e $\boldsymbol{r}^{(k)} := \boldsymbol{b} - A\boldsymbol{x}^{(k)}$.

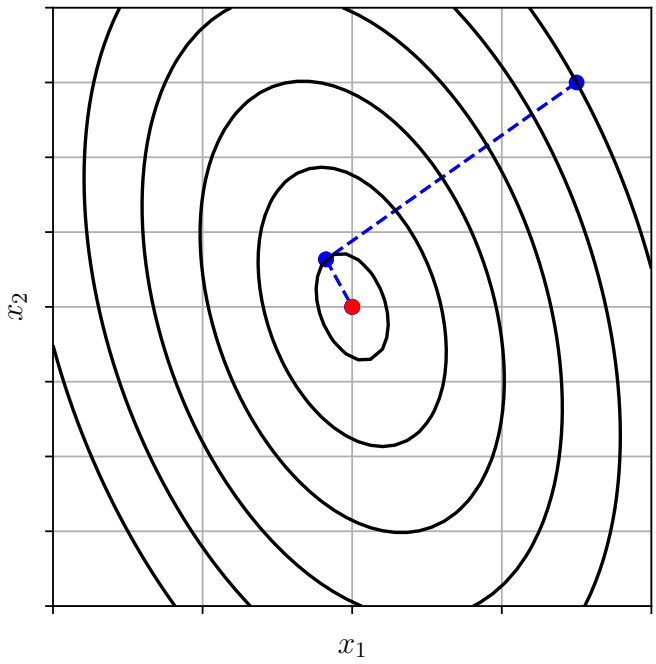

Figura 3.2: Iterações do Método do Gradiente Conjugado para sistemas lineares. Linha: $\|b - Ax\|$. Pontos: $\boldsymbol{x}^{(k)}$.

Observação 3.5.1. (Convergência.) A menos de erros de arredondamento, o Método do Gradiente Conjugado converge em no máximo n passos para a solução do sistema $A\boldsymbol{x} = \boldsymbol{b}$, com A matriz positiva definida $n \times n$.

Exemplo 3.5.1. Consideremos o sistema $Ax = b$ com

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.330}$$

$$b = \begin{bmatrix} -3 \\ 2 \\ 2 \\ -3 \end{bmatrix}. \tag{3.331}$$

Na Tabela 3.5 temos os resultados do emprego do método do gradiente conjugado com $\boldsymbol{x}^{(0)} = (0, 0, 0, 0)$.

Tabela 3.5: Resultados referentes ao Exemplo 3.5.1.

k	$\boldsymbol{x}^{(k)}$	$\|\boldsymbol{b} - A\boldsymbol{x}^{(k)}\|$
1	$(0, 0, 0, 0)$	5.1e+00
2	$(-1.1, 0.8, 0.8, -1.1)$	1.5e−01
3	$(-1.0, 1.0, 1.0, -1.0)$	2.0e−15

Código 3.8: mgc.py

```
1 import numpy as np
2 import numpy.linalg as npla
3
4 def mgc(A, b, x0,
5         maxiter=100, atol=1.49e-8, rtol=1.49e-8):
6
7   n = b.size
8   x = x0.copy()
9   r = b - A@x
10  d = r.copy()
11  nr = npla.norm(r)
12  print(f'0: {x}, nr = {nr:.1e}')
13  info = -1
14  for k in range(maxiter):
15    rdr = np.dot(r, r)
16    Ad = A@d
17    alpha = rdr/np.dot(d,Ad)
18    x = x + alpha*d
19
20    r = r - alpha*Ad
21    beta = np.dot(r,r)/rdr
22    d = r + beta*d
23
24    nr = npla.norm(r)
25    print(f'{k+1}: {x}, nr = {nr:.1e}')
26
```

```
27      if (nr <= max(atol, rtol*npla.norm(b))):
28        info = 0
29        break
30
31    return x, info
32
33 # matriz coefs
34 A = np.array([[2., -1., 0., 0.],
35               [-1., 2., -1., 0.],
36               [0., -1., 2., -1.],
37               [0., 0., -1., 2.]])
38 # vetor constante
39 b = np.array([-3., 2., 2., -3.])
40
41 # aprox. inicial
42 x0 = np.zeros_like(b)
43
44 x, info = mgc(A, b, x0)
```

3.5.1 Exercícios

E.3.5.1. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \tag{3.332}$$

$$b = \begin{bmatrix} -7 \\ 8 \end{bmatrix}. \tag{3.333}$$

Use o Método do Gradiente Conjugado com $\boldsymbol{x}^{(0)} = \mathbf{0}$ para computar a solução com tolerância

$$\|\boldsymbol{r}\| \leq \max\{\texttt{tol}, \texttt{tol}\|\boldsymbol{b}\|\}, \tag{3.334}$$

onde $\boldsymbol{r}$ é o resíduo do sistema e $\texttt{tol} = 1.49\text{e}-8$. Quantas iterações são requeridas até a convergência?

E.3.5.2. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 3 & 1 \\ 1 & 1 & 4 \end{bmatrix}, \tag{3.335}$$

$$b = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}. \tag{3.336}$$

Use o Método do Gradiente Conjugado com $\boldsymbol{x}^{(0)} = \boldsymbol{0}$ para computar a solução com tolerância

$$\|\boldsymbol{r}\| \leq \max\{\texttt{tol}, \texttt{tol}\|\boldsymbol{b}\|\}, \tag{3.337}$$

onde $\boldsymbol{r}$ é o resíduo do sistema e $\texttt{tol} = 1.49\text{e}-8$. Quantas iterações são requeridas até a convergência?

E.3.5.3. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.338}$$

$$b = \begin{bmatrix} 5 \\ -7 \\ 6 \\ -1 \end{bmatrix}. \tag{3.339}$$

Use o Método do Gradiente Conjugado com $\boldsymbol{x}^{(0)} = \boldsymbol{0}$ para computar a solução com tolerância

$$\|\boldsymbol{r}\| \leq \max\{\texttt{tol}, \texttt{tol}\|\boldsymbol{b}\|\}, \tag{3.340}$$

onde $\boldsymbol{r}$ é o resíduo do sistema e $\texttt{tol} = 1.49\text{e}-8$.

E.3.5.4. Considere o sistema linear $A\boldsymbol{x} = \boldsymbol{b}$ com

$$A = \begin{bmatrix} 2 & -1 & 1 & 0 \\ -1 & 2 & -1 & 0 \\ 1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \tag{3.341}$$

$$b = \begin{bmatrix} -8 \\ 9 \\ -10 \\ 5 \end{bmatrix}. \tag{3.342}$$

Use o Método do Gradiente Conjugado com $\boldsymbol{x}^{(0)} = \mathbf{0}$ para computar a solução com tolerância

$$\|\boldsymbol{r}\| \leq \max\{\texttt{tol}, \texttt{tol}\|\boldsymbol{b}\|\}, \tag{3.343}$$

onde $\boldsymbol{r}$ é o resíduo do sistema e $\texttt{tol} = 1.49\text{e}{-8}$.

E.3.5.5. Considere o problema de Laplace

$$-u_{xx} = 2, \quad 0 < x < 1, \tag{3.344}$$
$$u(0) = u(1) = 0. \tag{3.345}$$

A discretização pelo Método das Diferenças Finitas em uma malha uniforme $x_i = ih$, $i = 0.1, \ldots, n$, $h = 1/(n-1)$, leva ao seguinte sistema linear

$$u_1 = 0 \tag{3.346}$$
$$-\frac{1}{h^2}u_{i-1} + \frac{2}{h^2}u_i - \frac{1}{h^2}u_{i+1} = 2, \tag{3.347}$$
$$u_n = 0 \tag{3.348}$$

onde $u_i \approx u(x_i)$. Com $u^{(0)} = \mathbf{0}$, aplique o Método do Gradiente Conjugado para computar a solução deste sistema quando $n = 10, 20, 40, 80$. Quantas iterações são necessárias para obter-se a convergência do método com critério de convergência

$$\|\boldsymbol{r}\| \leq \texttt{tol}, \tag{3.349}$$

onde, $\boldsymbol{r} := \boldsymbol{b} - A\boldsymbol{x}$ é o resíduo e $\texttt{tol} = 1e-4$.

Respostas

E.3.5.1. Iterações: 2, $\boldsymbol{x} = (-2, 3)$.

E.3.5.2. Iterações: 3, $\boldsymbol{x} = (2, 0, -1)$.

E.3.5.3. $\boldsymbol{x} = (2.0, -1.0, 3.0, 1.0)$

E.3.5.4. $\boldsymbol{x} = (-2.0, 3.0, -1.0, 2.0)$

E.3.5.5.

n	k
10	4
20	9
40	19
80	39
160	79

Capítulo 4

Métodos para Sistemas Não Lineares

Neste capítulo, estudamos sobre métodos para a resolução de sistemas de equações não lineares. Vamos tratar o caso de problemas da forma: encontrar $\boldsymbol{x} \in \mathbb{R}^n$ tal que

$$F(\boldsymbol{x}) = \mathbf{0}, \tag{4.1}$$

onde $F : \mathbb{R}^n \to \mathbb{R}^n$ é uma dada função vetorial.

4.1 Método de Newton

Consideramos o problema de encontrar

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \tag{4.2}$$

tal que

$$F(\boldsymbol{x}) = \mathbf{0}, \tag{4.3}$$

onde $F : \mathbb{R}^n \to \mathbb{R}^n$ é uma dada função vetorial com

$$F(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})) \in \mathbb{R}^n. \tag{4.4}$$

Sejam $\boldsymbol{x}^*$ a solução exata de (4.3) e $\boldsymbol{x}^{(0)}$ uma dada aproximação de $\boldsymbol{x}^*$. Assim sendo, tomamos a seguinte expansão de F em polinômio de Taylor[42]:

$$F(\boldsymbol{x}^*) = F(\boldsymbol{x}^{(0)}) + J_F(\boldsymbol{x}^{(0)})(\boldsymbol{x}^* - \boldsymbol{x}^{(0)}) + \boldsymbol{r}, \tag{4.5}$$

onde J_F é a **matriz jacobiana**[47] de F

$$J_F(\boldsymbol{x}) := \frac{\partial(f_1, f_2, \ldots, f_n)}{\partial(x_1, x_2, \ldots, x_n)} \tag{4.6}$$

$$:= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{4.7}$$

e $\|\boldsymbol{r}\|^2 \to 0$ quando $\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\| \to 0$.

Daí, como $F(\boldsymbol{x}^*) = \boldsymbol{0}$, segue que

$$J_F(\boldsymbol{x}^{(0)})(\boldsymbol{x}^* - \boldsymbol{x}^{(0)}) \approx -F(\boldsymbol{x}^{(0)}). \tag{4.8}$$

Então, multiplicando a inversa da jacobiana à esquerda, obtemos

$$\boldsymbol{x}^* - \boldsymbol{x}^{(0)} \approx -J_F^{-1}(\boldsymbol{x}^{(0)})F(\boldsymbol{x}^{(0)}) \tag{4.9}$$

e, também,

$$\boldsymbol{x}^* \approx \boldsymbol{x}^{(0)} - J_F^{-1}(\boldsymbol{x}^{(0)})F(\boldsymbol{x}^{(0)}). \tag{4.10}$$

O exposto acima nos motiva a **iteração de Newton**[48]:

$$\boldsymbol{x}^{(0)} = \text{aprox. inicial}, \tag{4.11a}$$
$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - J_F^{-1}(\boldsymbol{x}^{(k)})F(\boldsymbol{x}^{(k)}), \tag{4.11b}$$

com $k = 0, 1, 2, \ldots$.

Exemplo 4.1.1. Seja o sistema de equações não lineares

$$x_1 x_2^2 = x_1^2 x_2 - 6, \tag{4.12a}$$
$$x_1^2 x_2^3 - 7 = -x_1. \tag{4.12b}$$

Para usarmos o método de Newton, reescrevemos o sistema na seguinte forma

$$x_1 x_2^2 - x_1^2 x_2 + 6 = 0, \tag{4.13a}$$
$$x_1 + x_1^2 x_2^3 - 7 = 0. \tag{4.13b}$$

Com isso, identificamos a função objetivo

$$F(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} \tag{4.14}$$
$$= \begin{bmatrix} x_1 x_2^2 - x_1^2 x_2 + 6 \\ x_1 + x_1^2 x_2^3 - 7 \end{bmatrix} \tag{4.15}$$

e calculamos sua matriz jacobiana

$$J_F(\boldsymbol{x}) = \frac{\partial(f_1, f_2)}{\partial(x_1, x_2)} \tag{4.16}$$
$$= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \tag{4.17}$$
$$= \begin{bmatrix} x_2^2 - 2x_1 x_2 & 2x_1 x_2 - x_1^2 \\ 1 + 2x_1 x_2^3 & 3x_1^2 x_2^2 \end{bmatrix} \tag{4.18}$$

Definidas F e J_F e tomando a aproximação inicial

$$\boldsymbol{x}^{(0)} = (-1.5, 1.5) \tag{4.19}$$

computamos as iterações de Newton e obtemos os resultados apresentados na Tabela 4.1.

Tabela 4.1: Resultados referentes ao Exemplo 4.1.1.

k	$\boldsymbol{x}^{(k)}$	$\|F(\boldsymbol{x}^{(k)})\|$
0	$(-1.50, 1.50)$	1.2e+0
1	$(-1.07, 1.82)$	1.2e+0
2	$(-9.95\text{e}{-1}, 2.00)$	7.6e−2
3	$(-1.00, 2.00)$	1.2e−4
4	$(-1.00, 2.00)$	2.1e−9

```
import numpy as np
import numpy.linalg as npla

def newton(F, J, x0,
           maxiter=100, tol=1.49e-8):
  print(f'\n{0}: x = {x0}, ' + \
        f'norm = {npla.norm(F(x0)):.1e}')
  info = -1
  for k in range(maxiter):
    x = x0 - npla.inv(J(x0))@F(x0)
    print(f'{k+1}: x = {x}, ' + \
          f'norm = {npla.norm(F(x)):.1e}')
    if (npla.norm(x - x0) < tol):
      info = 0
      break
    x0 = x.copy()
  return x, info

def F(x):
  n = x.size
  y = np.empty(n)
  y[0] = x[0]*x[1]**2 - x[0]**2*x[1] + 6
  y[1] = x[0] + x[0]**2*x[1]**3 - 7
  return y

def J(x):
  n = x.size
  y = np.empty((n,n))
  y[0,0] = x[1]**2 - 2*x[0]*x[1]
  y[0,1] = 2*x[0]*x[1] - x[0]**2
  y[1,0] = 1 + 2*x[0]*x[1]**3
  y[1,1] = 3*x[0]**2*x[1]**2
  return y

x0 = np.array([-1.5, 1.5])
x, info = newton(F, J, x0)
```

4.1.1 Análise Numérica

Para uma função F suficientemente suave e com uma escolha apropriada da aproximação inicial $\boldsymbol{x}^{(0)}$, temos que as iterações de Newton

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - J_F^{-1}(\boldsymbol{x}^{(k)})F(\boldsymbol{x}^{(k)}), \tag{4.20}$$

com $k = 0, 1, 2, \ldots$, são quadraticamente convergentes[49], i.e.

$$\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^*\| \leq C\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|^2, \tag{4.21}$$

onde $\boldsymbol{x}^*$ é a solução exata, i.e. $F(\boldsymbol{x}^*) = \mathbf{0}$.

Exemplo 4.1.2. Consideremos o seguinte sistema de equações não lineares

$$x_1 x_2^2 - x_1^2 x_2 + 6 = 0, \tag{4.22}$$
$$x_1 + x_1^2 x_2^3 - 7 = 0. \tag{4.23}$$

A Figura 4.1 é um esboço do gráfico da $\|F(\cdot)\|$. Este problema foi confeccionado de forma que $\boldsymbol{x}^* = (-1, 2)$. Então, tomando $\boldsymbol{x}^{(0)} = (1.5, 1.5)$ como aproximação inicial, computamos as iterações de Newton para este problema, donde obtemos os resultados reportados na Tabela 4.2.

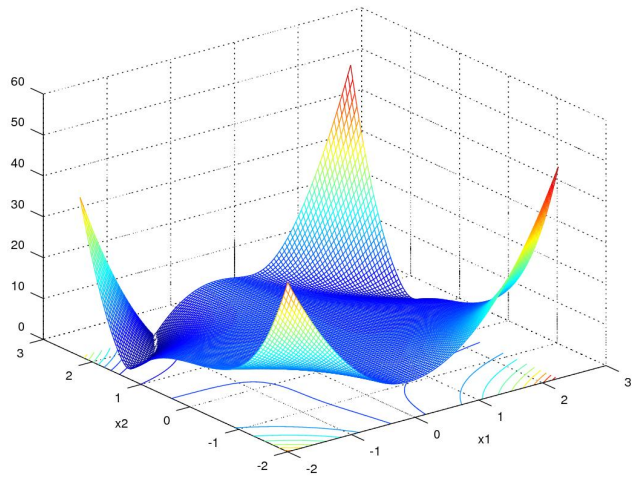

Figura 4.1: Esboço do gráfico de $\|F(\cdot)\|$ referente ao Exemplo 4.1.2.

k	$\boldsymbol{x}^{(k)}$	$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|$
0	$(-1.50, 1.50)$	7.1e−01
1	$(-1.07, 1.82)$	2.0e−01
2	$(-9.95\text{e}{-1}, 2.00)$	5.1e−03
3	$(-1.00, 2.00)$	2.6e−05
4	$(-1.00, 2.00)$	2.0e−10
5	$(-1.00, 2.00)$	3.1e−16

Tabela 4.2: Resultados referentes ao Exemplo 4.1.2.

4.1.2 Exercícios

E.4.1.1. Use o Método de Newton para computar uma solução aproximada para o sistema de equações

$$\frac{x_1^2}{3} + x_2^2 = 1 \tag{4.24a}$$
$$x_1^2 + \frac{x_2^2}{4} = 1 \tag{4.24b}$$

E.4.1.2. Use o Método de Newton, com aproximação inicial $\boldsymbol{x}^{(0)} = (1.5, 0.5)$ para computar uma solução aproximada para o sistema de equações

$$x_1^2 = \cos(x_1 x_2) + 1 \tag{4.25a}$$
$$\operatorname{sen}(x_2) = 2\cos(x_1) \tag{4.25b}$$

E.4.1.3. Use o Método de Newton, com aproximação inicial $\boldsymbol{x}^{(0)} = (1, -1)$ para computar uma solução aproximada para o sistema de equações

$$3x_1 = \cos(x_1 x_2) + \frac{1}{2} \tag{4.26a}$$
$$4x_1^2 + 2x_2 x_1 = 0 \tag{4.26b}$$

E.4.1.4. Use o método de Newton para obter uma aproximação de uma solução de

$$x_2 \operatorname{sen}(x_3) + x_1 - 2 = 0, \tag{4.27}$$
$$x_1 x_2 - \operatorname{sen}(x_2) + 0.2 = 0, \tag{4.28}$$
$$x_3^2 + \cos(x_1 x_2) - 4.5 = 0. \tag{4.29}$$

Para tanto, use $\boldsymbol{x}^{(1)} = (1, -1, -1)$.

E.4.1.5. Considere o problema de encontrar os pontos de interseção no plano $x - y$ da elipse

$$\frac{x^2}{4} + \frac{y^2}{9} = 1 \tag{4.30}$$

com a curva

$$x = y^2\sqrt{x}. \tag{4.31}$$

Escreva o problema na forma $F(\boldsymbol{x}) = \boldsymbol{0}$ e use o Método de Newton para encontrar o ponto de interseção próximo de $(x, y) = (1.5, 1.5)$.

Respostas

E.4.1.1. Soluções exatas: $\boldsymbol{x} = \pm\left(\sqrt{\frac{9}{11}}, \sqrt{\frac{8}{11}}\right)$.

E.4.1.2. $\boldsymbol{x} = (1.3468109, 0.4603195)$

E.4.1.3. $\boldsymbol{x} = (4.668417\text{e}{-1}, -9.368334\text{e}{-1})$

E.4.1.4. $\boldsymbol{x} = (1.7519\text{e}{+0}, -2.6202\text{e}{-1}, -1.8983\text{e}{+0})$

E.4.1.5. $x = 1.842996\text{e}{+0}$, $y = 1.165148\text{e}{+0}$

4.2 Métodos *Quasi*-Newton

Em revisão

4.2.1 Método do Acorde

Em revisão

O método do acorde consiste na seguinte iteração

$$\boldsymbol{x}^{(1)} = \text{aprox. inicial}, \tag{4.32}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - J_F^{-1}(\boldsymbol{x}^{(1)})F(\boldsymbol{x}^{(k)}). \tag{4.33}$$

Ou seja, é a iteração de Newton com jacobina constante.

Exemplo 4.2.1. Consideremos o seguinte sistema de equações não lineares

$$x_1 x_2^2 - x_1^2 x_2 + 6 = 0, \tag{4.34}$$

$$x_1 + x_1^2 x_2^3 - 7 = 0. \tag{4.35}$$

Definidas F e J_F e tomando $\boldsymbol{x}^{(1)} = (1.5, 1.5)$ como aproximação inicial, computamos as iterações do método do acorde de forma a obtermos os resultados apresentados na Tabela 4.3.

k	$\boldsymbol{x}^{(k)}$	$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|$
1	$(-1.50, 1.50)$	-x-
2	$(-1.07, 1.82)$	$5.3\text{e}{-1}$
3	$(-1.02, 1.93)$	$1.2\text{e}{-1}$
4	$(-1.00, 1.98)$	$5.2\text{e}{-2}$
5	$(-9.98\text{e}{-1}, 2.00)$	$1.8\text{e}{-2}$
6	$(-9.98\text{e}{-1}, 2.00)$	$4.7\text{e}{-3}$
7	$(-9.99\text{e}{-1}, 2.00)$	$9.0\text{e}{-4}$
8	$(-1.00, 2.00)$	$7.4\text{e}{-4}$
9	$(-1.00, 2.00)$	$4.3\text{e}{-4}$

Tabela 4.3: Resultados referentes ao Exemplo 4.2.1.

4.2.2 Jacobiana Aproximada

Em revisão

A jacobiana $J_F(\boldsymbol{x})$ de uma dada função $F(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ é a matriz cujo elemento da i-ésima linha e j-ésima coluna é

$$\frac{\partial f_i}{\partial x_j} = \lim_{h \to 0} \frac{f_i(\boldsymbol{x} + \boldsymbol{e}_j h) - f_i(\boldsymbol{x})}{h}, \tag{4.36}$$

onde $\boldsymbol{e}_j$ é o j-ésimo vetor da base canônica de $\mathbb{R}^n$, i.e. $\boldsymbol{e}_j = (0, \ldots, 0, 1, 0, \ldots, 0)$ com 1 na j-ésima posição.

Com isso, podemos computar uma jacobiana aproximada tomando

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(\boldsymbol{x} + \boldsymbol{e}_j h) - f_i(\boldsymbol{x})}{h}, \tag{4.37}$$

com h suficientemente pequeno.

Exemplo 4.2.2. Consideremos o seguinte sistema de equações não lineares

$$x_1 x_2^2 - x_1^2 x_2 + 6 = 0, \tag{4.38}$$
$$x_1 + x_1^2 x_2^3 - 7 = 0. \tag{4.39}$$

Definida F, sua jacobina aproximada $\tilde{J}_F$ com $h = 10^{-7}$ e tomando $\boldsymbol{x}^{(1)} = (1.5, 1.5)$ como aproximação inicial, computamos as iterações do *quasi*-método de forma a obtermos os resultados apresentados na Tabela 4.4.

k	$\boldsymbol{x}^{(k)}$	$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|$
1	$(-1.50, 1.50)$	-x-
2	$(-1.07, 1.82)$	5.3e−1
3	$(-9.95\text{e}{-1}, 2.00)$	2.0e−1
4	$(-1.00, 2.00)$	5.1e−3
5	$(-1.00, 2.00)$	2.6e−5

Tabela 4.4: Resultados referentes ao Exemplo 4.2.2.

4.2.3 Exercícios

`Em construção`

Respostas

Capítulo 5

Interpolação

Neste capítulo, estudamos a resolução de problemas de interpolação da forma: dados uma família de n funções reais

$$\mathcal{F} = \{f_1(x), f_2(x), \ldots, f_n(x)\} \tag{5.1}$$

e um conjunto de n pontos $\{(x_i, y_i)\}_{i=1}^n$, com $x_i \neq x_j$ se $i \neq j$, encontrar a **função interpoladora**

$$f(x) = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x), \tag{5.2}$$

tal que

$$y_i = f(x_i), \quad i = 1, 2, \ldots, n. \tag{5.3}$$

5.1 Interpolação Polinomial

Dado um conjunto de n pontos $\{(x_i, y_i)\}_{i=1}^n$, o problema de interpolação consiste em encontrar o polinômio[50] de grau $n-1$

$$\begin{aligned} p(x) = {} & p_1 x^{n-1} + p_2 x^{n-2} \\ & + \cdots + p_{n-1} x + p_n \end{aligned} \tag{5.4}$$

tal que

$$y_i = p(x_i), \tag{5.5}$$

para todo $i = 1, 2, \ldots, n$.

Das condições (5.4), temos

$$\begin{aligned} p_1 x_1^{n-1} + p_2 x_1^{n-2} + \cdots + p_n &= y_1 \\ p_1 x_2^{n-1} + p_2 x_2^{n-2} + \cdots + p_n &= y_2 \\ &\vdots \\ p_1 x_n^{n-1} + p_2 x_n^{n-2} + \cdots + p_n &= y_n. \end{aligned} \tag{5.6}$$

Isto é, os coeficientes do **polinômio interpolador** (5.4) satisfazem o sistema linear

$$A\boldsymbol{p} = \boldsymbol{y}, \tag{5.7}$$

onde A é a **matriz de Vandermonde**[51]

$$A = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix}, \tag{5.8}$$

$\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$ é o **vetor das incógnitas** e $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ é o **vetor dos termos constantes**.

Exemplo 5.1.1. Consideramos o problema de encontrar o polinômio interpolador do conjunto de pontos $\{(-1, -1), (0, 1), (1, 1/2)\}$. Como temos 3 pontos, o polinômio tem grau 2 e pode ser escrito na forma

$$p(x) = p_1 x^2 + p_2 x + p_3. \tag{5.9}$$

Seguindo a abordagem acima, temos $\boldsymbol{p} = (p_1, p_2, p_3)$, $\boldsymbol{x} = (-1, 0, 1)$, $\boldsymbol{y} = (-1, 1, 1/2)$ e

$$A = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix}. \tag{5.10}$$

Então, resolvendo $A\boldsymbol{p} = \boldsymbol{y}$, obtemos o polinômio interpolador

$$p(x) = -1.25x^2 + 0.75x + 1. \tag{5.11}$$

A Figura 5.1 mostra os esboços do polinômio interpolador $p(x)$ e dos pontos dados.

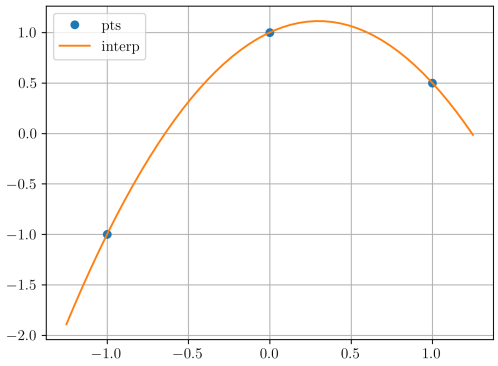

Figura 5.1: Esboço do polinômio interpolador referente ao Exemplo 5.1.1.

Código 5.1: poliInterp.py

```
1 import numpy as np
2 import numpy.linalg as npla
3
4 def poliInterp(x, y):
5   # num. pts
6   n = x.size
7   # Vandermonde
8   A = np.empty((n,n))
9   for j in range(n):
10    A[:,j] = x**(n-1-j)
11  # coefs
12  p = npla.solve(A, y)
13  return p
14
15 # exemplo
16 x = np.array([-1., 0, 1])
17 y = np.array([-1., 1, 1/2])
18
19 # poli interp
```

```
20 p = poliInterp(x, y)
21
22 # verificação
23 print(np.polyval(p, x))
```

5.1.1 Exercícios

E.5.1.1. Obtenha o polinômio interpolador do conjunto de pontos $\{(-1,-1), (-0.5,1), (1,2)\}$.

E.5.1.2. Obtenha o polinômio interpolador do conjunto de pontos $\{(-1,-1), (0,1), (1,1/2), (2,1)\}$.

E.5.1.3. Obtenha o polinômio interpolador do conjunto de pontos $\{(-1,\ -1), (0,1), (1,1/2), (2,1), (2.5,1)\}$.

E.5.1.4. Considere a matriz de Vandermonde $V = [\boldsymbol{x}^{n-j}]_{j=1}^{n}$, com $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, sendo $x_i = (i-1)h$, $h = 0.1$ e $i = 1, 2, \ldots, n$. Compute o número de condicionamento de V para $n = 5, 10, 100$. De que forma os resultados obtidos impactam no problema de interpolação polinomial?

E.5.1.5. Aproxime a função $f(x) = \cos(x)$ por um polinômio interpolador p no intervalo $[0, \pi]$. Escolhas pontos nesse intervalo de forma a obter p que aproxime f com boa precisão.

Respostas

E.5.1.1.

$$-1,\bar{6}x^2 + 1.5x + 2.1\bar{6}$$

E.5.1.2.

$$0.58\bar{3}x^3 - 1.25x^2 + 0.1\bar{6}x + 1$$

.

E.5.1.3.

$$-0.26190476x^4 + 1.10714286x^3 - 0.98809524x^2 - 0.35714286x + 1$$

.

E.5.1.4.

n	$\kappa(V)$
5	1.03e+4
10	2.57e+7
100	9.11e+109

E.5.1.5. Dica: use os pontos $x_i = (i-1)\frac{\pi}{4}$, $i = 1, 2, 3, 4$.

5.2 Interpolação de Lagrange

Interpolação de Lagrange[58] é uma técnica para a computação do polinômio interpolador $p(x)$ de um conjunto de pontos $\{(x_i, y_i)\}_{i=1}^n$ dados. A ideia consiste em escrever o polinômio interpolador na forma

$$p(x) = \sum_{i=1}^{n} y_i L_i(x) \tag{5.12a}$$

$$= y_1 L_1(x) + y_2 L_2(x) + \cdots + y_n L_n(x), \tag{5.12b}$$

onde $L_i(x)$ é chamado de i-ésimo polinômio de Lagrange e é definido como o polinômio de grau $n-1$ que satisfaz

$$L_i(x_j) = \begin{cases} 1 & , i = j \\ 0 & , i \neq j \end{cases} \tag{5.13}$$

Mais especificamente, temos que $L_i(x)$ tem raízes $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$ e, portanto, pode ser decomposto na forma

$$L_i(x) = c_i \prod_{\substack{j=1 \\ j \neq i}}^{n} (x - x_j) \tag{5.14a}$$

$$= c_i (x - x_1) \cdots (x - x_{i-1})$$

$$\times (x - x_i) \cdots (x - x_n). \tag{5.14b}$$

Além disso, como $L_i(x_i) = 1$, temos

$$c_i = \frac{1}{\prod\limits_{\substack{j=1 \\ j \neq i}}^{n} (x_i - x_j)}. \tag{5.15}$$

Assim sendo, podemos concluir que

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{5.16a}$$

$$= \frac{(x - x_1) \cdots (x - x_{i-1})}{(x_i - x_1) \cdots (x_i - x_{i-1})} \times \frac{(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_{i+1}) \cdots (x_i - x_n)}. \tag{5.16b}$$

Exemplo 5.2.1. Consideramos o problema de encontrar o polinômio interpolador do conjunto de pontos $\{(-1, -1), (0, 1), (1, 1/2)\}$. Como temos 3 pontos, o polinômio tem grau 2 e pode ser escrito na seguinte forma de Lagrange

$$p(x) = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x), \tag{5.17}$$

onde $y_1 = -1$, $y_2 = 1$ e $y_3 = 1/2$. Os polinômios de Lagrange são dados por

$$L_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \tag{5.18}$$

$$= \frac{1}{2}x^2 - \frac{1}{2}x, \tag{5.19}$$

$$L_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} \tag{5.20}$$

$$= -x^2 + 1, \tag{5.21}$$

$$L_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} \tag{5.22}$$

$$= \frac{1}{2}x^2 + \frac{1}{2}x. \tag{5.23}$$

(5.24)

E, então, temos o polinômio interpolador

$$p(x) = -1.25x^2 + 0.75x + 1. \tag{5.25}$$

Código 5.2: poliLagrange.py

```
import numpy as np

def poliLagrange(xpts, ypts):
  # num. pts
  n = xpts.size
  # interp poli
  p = np.poly1d(0)
  for i in range(n):
    # Lagrange poli
    L = np.poly1d(1)
    for j in range(n):
      if (i != j):
        L *= np.poly1d([1, -xpts[j]])/(xpts[i]-
xpts[j])
    p += ypts[i] * L
  return p



# exemplo
xpts = np.array([-1., 0, 1])
ypts = np.array([-1., 1, 1/2])


# interp poli
p = poliLagrange(xpts, ypts)
print(p)

# verificação
print(p(x))
```

5.2.1 Aproximação de Funções

Polinômio interpoladores podem ser usados para a aproximação de funções. Podemos aproximar uma dada função f pelo polinômio interpolador de um conjunto de pontos selecionados $\{(x_i, y_i = f(x_i))\}_{i=1}^n$. De fato, o seguinte teorema nos fornece uma estimativa para o erro de uma tal interpolação.

Teorema 5.2.1 (Teorema de Lagrange)**.** *Sejam dados uma função $f \in C^{n+1}([a,b])$ e n pontos $\{x_i\}_{i=1}^n \subset [a,b]$. Então, o polinômio interpolador do conjunto de pontos $\{x_i, y_i = f(x_i)\}_{i=1}^n$ satisfaz*

$$f(x) = p(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=1}^{n} (x - x_i). \tag{5.26}$$

Exemplo 5.2.2. Consideramos o problema de aproximar $f(x) = \text{sen}(x)$ pelo polinômio interpolador do conjunto de pontos $x_1 = 0$, $x_2 = \pi/2$ e $x_3 = \pi$. I.e., queremos determinar o polinômio $p(x)$ de grau 2 que interpola os pontos $\{(0,0),\ (\pi/2, 1),\ (\pi, 0)\}$. Usando a técnica de Lagrange, obtemos

$$p(x) = -0.41x^2 + 1.3x, \tag{5.27}$$

com seus coeficientes arredondados para dois dígitos significativos. A Figura 5.2 mostra os esboços da função $f(x) = \text{sen}(x)$, dos pontos dados e do polinômio interpolador $p(x)$.

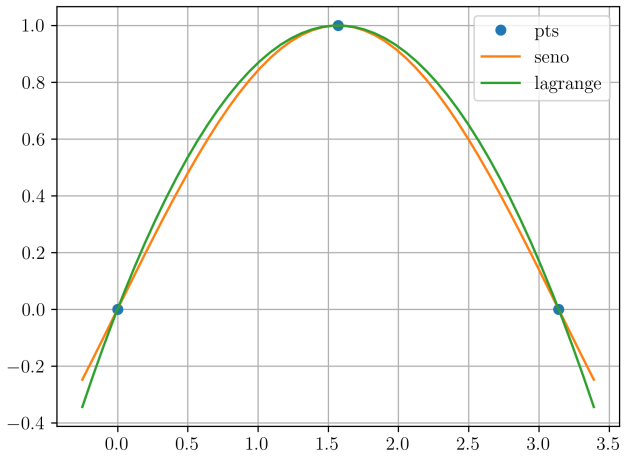

Figura 5.2: Esboços dos gráficos da função, dos pontos e do polinômio interpolador computado no Exemplo 5.2.2.

```
x = np.array([0, np.pi/2, np.pi])
f = lambda x: np.sin(x)
poli = poliLagrange(x, f(x))
```

5.2.2 Exercícios

E.5.2.1. Use a técnica de Lagrange para obter o polinômio interpolador do conjunto de pontos $\{(-1,-1), (-0.5,1), (1,2)\}$.

E.5.2.2. Use a técnica de Lagrange para obter o polinômio interpolador do conjunto de pontos $\{(-1,-1), (0,1), (1,1/2), (2,1)\}$.

E.5.2.3. Use a técnica de Lagrange para obter o polinômio interpolador do conjunto de pontos $\{(-1,-1), (0,1), (1,1/2), (2,1), (2.5,1)\}$.

E.5.2.4. Use a técnica de interpolação de Lagrange para encontrar o polinômio interpolador que aproxima a função$f(x) = e^x$ pelos pontos $x_1 = 0$,

$x_2 = 1$, $x_3 = 1.5$ e $x_4 = 2$.

E.5.2.5. Use a técnica de Lagrange para aproximar a função $f(x) = \cos(x)$ por um polinômio interpolador p no intervalo $[0, \pi]$. Escolha pontos de forma a obter p que aproxime f com boa precisão.

Respostas

E.5.2.1. $-1,\bar{6}x^2 + 1.5x + 2.1\bar{6}$

E.5.2.2. $0.58\bar{3}x^3 - 1.25x^2 + 0.1\bar{6}x + 1.$

E.5.2.3. $-0.26190476x^4 1.10714286x^3 - 0.98809524x^2 - 0.35714286x 1.$

E.5.2.4. $0.54x^3 - 0.15x^2 + 1.3x + 1.$

E.5.2.5. Dica: use os pontos $x_i = (i-1)\frac{\pi}{4}$, $i = 1, 2, 3, 4$.

5.3 Diferenças Divididas de Newton

Dado um conjunto de pontos $\{(x_i, y_i)\}_{i=1}^n$, o **Método das Diferenças Divididas de Newton**[59] busca determinar o polinômio interpolador da forma

$$\begin{aligned} p(x) &= a_1 + a_2(x - x_1) \\ &\quad + a_3(x - x_1)(x - x_2) \\ &\quad + \cdots + a_n(x - x_1)\cdots(x - x_{n-1}). \end{aligned} \tag{5.28}$$

Por uma abordagem direta, temos que $p(x_i) = y_i$, $i = 1, 2, \ldots, n$, o que nos leva ao seguinte sistema triangular inferior

$$a_1 = y_1, \tag{5.29a}$$
$$a_1 + a_2(x_2 - x_1) = y_2, \tag{5.29b}$$
$$a_1 + a_2(x_3 - x_1) + a_3(x_3 - x_1)(x_3 - x_2) = y_3, \tag{5.29c}$$

$$\vdots \tag{5.29d}$$

$$a_1 + a_2(x_n - x_1) + \cdots + a_n(x_n - x_1) \cdot \cdots \cdot (x_n - x_{n-1}) = y_n. \tag{5.29e}$$

Entretanto, existe uma forma mais eficiente de se determinar os coeficientes a_i, $i = 1, 2, \ldots, n$.

Denotemos por $p[x_j, x_{j+1}, \ldots, x_k](x)$ o polinômio interpolador do conjunto de pontos $\{(x_i, y_i)\}_{i=j}^{k}$. Então, temos a seguinte recursão

$$p[x_j] = y_j, \tag{5.30}$$

para $j = 1, 2, \ldots, n$ e

$$\begin{aligned} &p[x_j, x_{j+1}, \ldots, x_k](x) \\ &\quad = \frac{(x - x_j)p[x_{j+1}, \ldots, x_k](x) - (x - x_k)p[x_j, \ldots, x_{k-1}](x)}{x_k - x_j}, \end{aligned} \tag{5.31}$$

para todo $n \geq k > j \geq 1$.

De fato, (5.30) é trivial. Agora, denotando por $r(x)$ o lado direito da equação (5.31), vemos que $r(x)$ tem grau menor ou igual a $k - j$, o mesmo de $p[x_j, x_{j+1}, \ldots, x_k](x)$. Desta forma, para mostrar (5.31), basta verificarmos que $r(x)$ interpola o conjunto de pontos $\{(x_i, y_i)\}_{i=j}^{k}$. O que de fato ocorre

$$r(x_j) = \frac{-(x_j - x_k)y_j}{x_k - x_j} = y_j, \tag{5.32a}$$

$$r(x_l) = \frac{(x_l - x_j)y_l - (x_l - x_k)y_l}{x_k - x_j} = y_l,$$

$$l = j + 1, \ldots, k - 1, \tag{5.32b}$$

$$r(x_k) = \frac{(x_k - x_j)y_k}{x_k - x_j} = y_k. \tag{5.32c}$$

Logo, pela unicidade do polinômio interpolador[60], temos demonstrado (5.31).

Observando que o polinômio interpolador $p(x)$ é igual a $p[x_1, \ldots, x_n](x)$, temos que (5.30)-(5.31) nos fornece uma forma de computar $p(x)$ de forma recursiva. Além disso, observemos que $p[x_j, \ldots, x_{k-1}](x)$ e $p[x_j, \ldots, x_k]$ diferem por um polinômio de grau $k - j$ com zeros x_j, x_{j+1}, ..., x_{k-1}. Logo,

temos

$$\begin{aligned} p[x_j,\ldots,x_k](x) &= p[x_j,\ldots,x_{k-1}](x) \\ &\quad + f[x_j,\ldots,x_k](x-x_j)\cdots(x-x_{k-1}), \end{aligned} \tag{5.33}$$

onde $f[x_j,\ldots,x_k]$ são coeficientes a determinar. Ainda, tomando $p[x_i] = f[x_i]$, temos

$$\begin{aligned} p[x_j,\ldots,x_k](x) &= f[x_j] + f[x_j,x_{j+1}](x-x_j) \\ &\quad + f[x_j,\ldots,x_k](x-x_j)\cdots(x-x_{k-1}). \end{aligned} \tag{5.34}$$

Por fim, a recursão (5.30)-(5.31) nos mostra que as Diferenças Divididas Newton podem ser obtidas de

$$f[x_j] = y_j, \quad j = 1,2,\ldots,n, \tag{5.35a}$$

$$f[x_j,\ldots,x_k] = \frac{f[x_{j+1},\ldots,x_k] - f[x_j,\ldots,x_{k-1}]}{x_k - x_j}, \tag{5.35b}$$

para todo $n \geq k > j \geq 1$. E, temos o polinômio interpolador do conjunto de pontos $\{(x_i,y_i)\}_{i=1}^n$ dado por

$$\begin{aligned} p[x_1,\ldots,x_n](x) &= f[x_1] + f[x_1,x_2](x-x_1) \\ &\quad + \cdots + f[x_1,\ldots,x_n](x-x_1)\cdots(x-x_n). \end{aligned} \tag{5.36}$$

Observação 5.3.1. A recursão (5.35) pode ser adequadamente organizada em uma matriz da forma

$$\begin{bmatrix} \boldsymbol{f[x_1]} & 0 & 0 & \ldots & 0 \\ f[x_2] & \boldsymbol{f[x_1,x_2]} & 0 & \ldots & 0 \\ f[x_3] & f[x_2,x_3] & \boldsymbol{f[x_1,x_2,x_3]} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ f[x_n] & f[x_{n-1},x_n] & f[x_{n-2},x_{n-1},x_n] & \ldots & \boldsymbol{f[x_1,x_2,\ldots,x_n]} \end{bmatrix} \tag{5.37}$$

onde os elementos da diagonal correspondem aos coeficientes do polinômio interpolador na forma (5.36).

Exemplo 5.3.1. Consideramos o problema de encontrar o polinômio interpolador do conjunto de pontos $\{(-1,-1),(0,1),(1,1/2)\}$. Usando o Método das Diferenças Divididas de Newton, escrevemos o polinômio na forma

$$p(x) = f[x_1] + f[x_1,x_2](x-x_1) + f[x_1,x_2,x_3](x-x_1)(x-x_2). \tag{5.38}$$

Então, computamos seus coeficientes pela recursão (5.35). Ou seja, temos

$$f[x_1] = -1, \tag{5.39a}$$
$$f[x_2] = 1, \tag{5.39b}$$
$$f[x_3] = 1/2. \tag{5.39c}$$

Daí, segue

$$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1} = 2 \tag{5.40a}$$
$$f[x_2, x_3] = \frac{f[x_3] - f[x_2]}{x_3 - x_2} = -\frac{1}{2} \tag{5.40b}$$
$$\tag{5.40c}$$

e, por fim, que

$$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} \tag{5.41a}$$
$$= -1.25. \tag{5.41b}$$

Logo, o polinômio interpolador é

$$p(x) = 0.5 + 2(x + 1) - 1.25(x + 1)(x - 1), \tag{5.42}$$

ou, equivalentemente,

$$p(x) = -1.25x^2 + 0.75x + 1. \tag{5.43}$$

```
1  import numpy as np
2
3  def interpDDF(x, y):
4    n = x.size
5    M = np.empty((n,n))
6    M[:,0] = y
7    for j in range(1,n):
8      for i in range(j,n):
9        M[i,j] = (M[i,j-1] - M[i-1,j-1]) \
10           / (x[i]-x[i-j])
11   return np.diag(M)
```

```
12
13 def poliDDF(x, p, xpts):
14   n = p.size
15   pval = p[0]
16   aux = 1.
17   for i in range(1,n):
18     aux *= (x-xpts[i-1])
19     pval += p[i]*aux
20   return pval
21
22 xpts = np.array([-1., 0, 1])
23 ypts = np.array([-1., 1, 1/2])
24 p = interpDDF(xpts, ypts)
25 print(poliDDF(xpts, p, xpts))
```

5.3.1 Exercícios

E.5.3.1. Use o Método das Diferenças Divididas de Newton para obter o polinômio interpolador do conjunto de pontos $\{(-1,-1),(-0.5,1),(1,2)\}$.

E.5.3.2. Use o Método das Diferenças Divididas de Newton para obter o polinômio interpolador do conjunto de pontos $\{(-1,-1),(0,1),(1,1/2),(2,1)\}$.

E.5.3.3. Use o Método das Diferenças Divididas de Newton para obter o polinômio interpolador do conjunto de pontos $\{(-1,-1),(0,1),(1,1/2),(2,1),(2.5,1)\}$.

E.5.3.4. Use o método das diferenças divididas de Newton para encontrar o polinômio interpolador que aproxima a função $f(x) = e^x$ pelos pontos $x_1 = 0$, $x_2 = 1$, $x_3 = 1.5$ e $x_4 = 2$.

Análise Numérica

E.5.3.5. Dado um conjunto de pontos distintos $\{(x_i, y_i)\}_{i=1}^{n}$, mostre que é único o polinômio interpolador do conjunto.

Respostas

E.5.3.1. $-1,\bar{6}x^2 + 1.5x + 2.1\bar{6}$

E.5.3.2. $0.58\bar{3}x^3 - 1.25x^2 + 0.1\bar{6}x + 1.$

E.5.3.3. $-0.26190476x^4 1.10714286x^3 - 0.98809524x^2 - 0.35714286x 1.$

E.5.3.4. $p(x) = 0.54x^3 - 0.15x^2 + 1.3x + 1.$

5.4 Spline Cúbico

Dado um conjunto de pontos $\{(x_i, y_i)\}_{i=1}^n$, um **spline cúbico** é uma função duas vezes continuamente diferenciável da forma

$$s(x) = \begin{cases} s_{11}(x-x_1)^3 + s_{12}(x-x_1)^2 + s_{13}(x-x_1) + s_{14} & , x_1 \leq x < x_2, \\ s_{21}(x-x_2)^3 + s_{22}(x-x_2)^2 + s_{23}(x-x_2) + s_{24} & , x_2 \leq x < x_3, \\ & \vdots \\ s_{n-1,1}(x-x_2)^3 + s_{n-1,2}(x-x_2)^2 + s_{n-1,3}(x-x_2) + s_{n-1,4} & , x_{n-1} \leq x \leq x_n. \end{cases} \tag{5.44}$$

que satisfaz as seguintes propriedades

1. $s(x_i) = y_i$ para $i = 1, 2, \ldots, n$,
2. $s_j(x_j) = s_{j+1}(x_j)$ para todo $j = 1, 2, \ldots, n-2$,
3. $s'_j(x_j) = s'_{j+1}(x_j)$ para todo $j = 1, 2, \ldots, n-2$,
4. $s''_j(x_j) = s''_{j+1}(x_j)$ para todo $j = 1, 2, \ldots, n-2$.

Observemos que o spline tem $4(n-1)$ coeficientes a determinar, enquanto que as condições acima nos fornecem $4n-6$ equações. Assim sendo, notamos que a determinação de um spline requer ainda 2 condições de fechamento. Conforme a escolha destas condições, diferentes splines cúbicos são computados.

5.4.1 Spline *Not-a-Knot*

A condição *not-a-knot* exige que o spline cúbico tenha derivada terceira contínua nos pontos x_2 e x_{n-1}, i.e.

$$s_1'''(x_2) = s_2'''(x_2), \tag{5.45a}$$
$$s_{n-2}'''(x_{n-1}) = s_{n-1}'''(x_{n-1}). \tag{5.45b}$$

Exemplo 5.4.1. Consideremos o problema de aproximar a função $f(x) = \text{sen}(x)$ pelo spline cúbico *not-a-knot* com pontos $x_1 = 0$, $x_2 = \pi/6$, $x_3 = \pi/3$ e $x_4 = \pi/2$. Na Figura 5.3 temos os esboços de f e do spline cúbico computado. O spline computado é aproximadamente

$$s(x) = \begin{cases} -0.11x^3 - 0.11x^2 - 0.11x & , 0 \leq x < \frac{\pi}{6} \\ -0.07(x-\frac{\pi}{6})^3 - 0.24(x-\frac{\pi}{6})^2 - 0.42(x-\frac{\pi}{6}) + \frac{1}{2} & , \frac{\pi}{6} \leq x < \frac{\pi}{3}, \\ 1.02(x-\frac{\pi}{3})^3 + 0.86(x-\frac{\pi}{3})^2 + 0.51(x-\frac{\pi}{3}) + \frac{\sqrt{3}}{2} & , \frac{\pi}{3} \leq x < \frac{\pi}{2} x \end{cases} \tag{5.46}$$

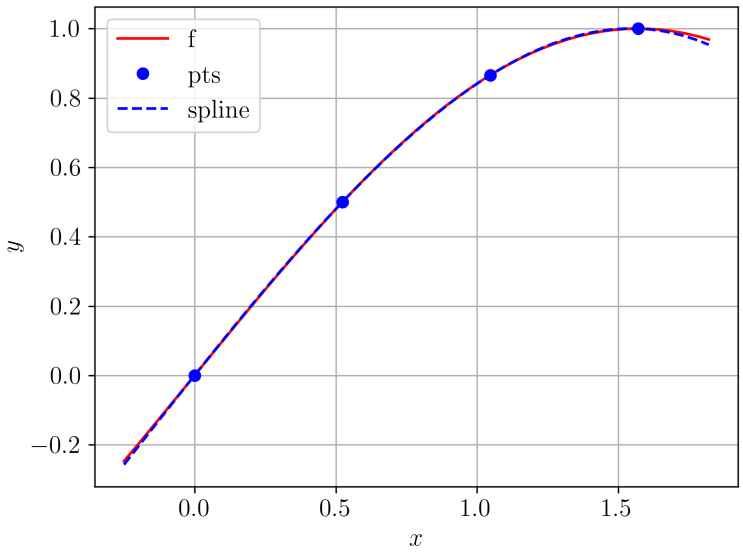

Figura 5.3: Esboço dos gráficos da função $f(x) = \text{sen}(x)$ e do spline cúbico computado no Exemplo 5.4.1.

Código 5.3: splineNotAKnot.py

```
1 import numpy as np
2 from scipy.interpolate import CubicSpline
3
4 # dados
5 f = lambda x: np.sin(x)
6 xx = np.array([0.,
7                np.pi/6,
8                np.pi/3,
9                np.pi/2])
10 yy = f(xx)
11
12 # spline
13 cs = CubicSpline(xx, yy)
14
15 # coefs
16 print(cs.c)
```

5.4.2 Spline Fixado

Os splines cúbicos fixados são obtidos impondo os valores das derivadas na fronteira, i.e.

$$s'(x_1) = y'_1, \tag{5.47a}$$
$$s'(x_n) = y'_n, \tag{5.47b}$$

onde y'_1 e y'_n são escalares dados. Quando usamos splines para aproximarmos uma dada função f, usualmente, escolhemos $y'_1 = f'(x_1)$ e $y'_n = f'(x_n)$.

Exemplo 5.4.2. Consideremos o problema de aproximar a função $f(x) = \text{sen}(x)$ pelo spline cúbico fixado com pontos $x_1 = 0$, $x_2 = \pi/6$, $x_3 = \pi/3$ e $x_4 = \pi/2$. Na Figura 5.4 temos os esboços de f e do spline cúbico computado

$$s(x) = \begin{cases} -0.16x^3 - 0.12x^2 - 0.04x & ,0 \leq x < \frac{\pi}{6} \\ -0.001(x - \frac{\pi}{6})^3 - 0.26(x - \frac{\pi}{6})^2 - 0.44(x - \frac{\pi}{6}) + \frac{1}{2} & ,\frac{\pi}{6} \leq x < \frac{\pi}{3}, \\ 0.0(x - \frac{\pi}{3})^3 + 0.5(x - \frac{\pi}{3})^2 + 0.87(x - \frac{\pi}{3}) + \frac{\sqrt{3}}{2} & ,\frac{\pi}{3} \leq x < \frac{\pi}{2}x \end{cases} \tag{5.48}$$

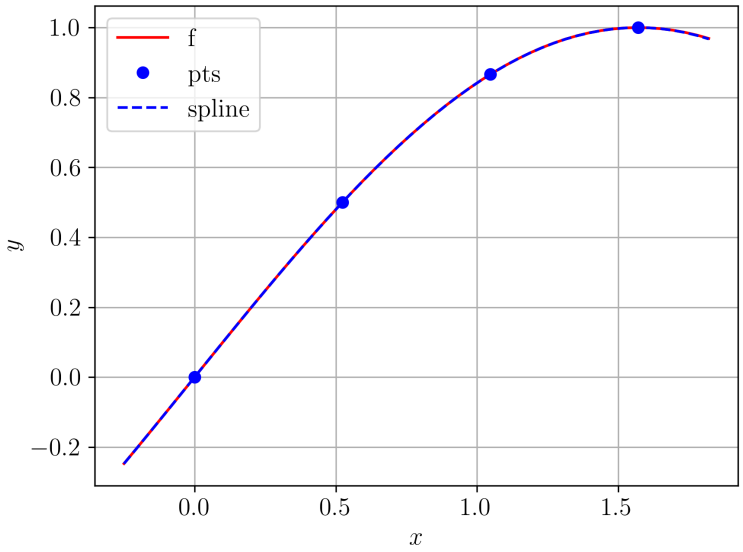

Figura 5.4: Esboço dos gráficos da função $f(x) = \text{sen}(x)$ e do spline cúbico fixado computado no Exemplo 5.4.2.

```
import numpy as np
from scipy.interpolate import CubicSpline

# dados
f = lambda x: np.sin(x)
xx = np.array([0.,
               np.pi/6,
               np.pi/3,
               np.pi/2])
yy = f(xx)

# spline
cs = CubicSpline(xx, yy,
                 bc_type=((1, 1.),
                          (1, 0.)))

# coefs
print(cs.c)
```

5.4.3 Exercícios

E.5.4.1. Dado o conjunto de pontos $\{(-1,-1),(-0.5,1),(1,2)\}$, obtenha o spline cúbico associado com condição de controno:

a) Not-a-Knot.

b) Fixado.

E.5.4.2. Dado o conjunto de pontos $\{(-1,-1),(0,1),(1,1/2),(2,1)\}$, obtenha o spline cúbico associado com condição de controno:

a) Not-a-Knot.

b) Fixado.

E.5.4.3. Dado o conjunto de pontos $\{(-1,\ -1),(0,1),(1,1/2),(2,1),(2.5,1)\}$, obtenha o spline cúbico associado com condição de controno:

a) Not-a-Knot.

b) Fixado.

E.5.4.4. Aproxime a função $f(x)=e^x$ por um spline cúbico que passa pelos pontos $x_1=0$, $x_2=1$, $x_3=1.5$ e $x_4=2$.

E.5.4.5. Considere o problema de aproximar a função $f(x)=\cos(x)$ por um spline cúbico $s=s(x)$ no intervalo $[0,\pi]$. Escolha os pontos e a condição de fronteira de forma a obter s que aproxime f com boa precisão gráfica.

Respostas

E.5.4.4. Dica: use um spline fixado.

E.5.4.5. Dica: use um spline fixado.

Capítulo 6

Aproximação por Mínimos Quadrados

6.1 Problemas Lineares

Dado um conjunto de n pontos $\{(x_i, y_i)\}_{i=1}^n$, $x_i \neq x_j$ para $i \neq j$, e uma família de $m \leq n$ funções $\{f_i(x)\}_{i=1}^m$, o problema linear de **aproximação por mínimos quadrados** consiste em determinar os m coeficientes $\{c_i\}_{i=1}^m$ tal que a função

$$f(x; \boldsymbol{c}) = \sum_{j=1}^{m} c_j f_j(x) \tag{6.1a}$$

$$= c_1 f_1(x) + c_2 f_2(x) + c_3 f_3(x) + \cdots + c_m f_m(x) \tag{6.1b}$$

aproxime o dado conjunto de pontos no sentido de mínimos quadrados, i.e. o **vetor dos coeficientes** $\boldsymbol{c} = (c_1, c_2, \ldots, c_m)$ é solução do seguinte problema linear de minimização

$$\min_{\boldsymbol{c}} \left\{ E := \sum_{i=1}^{n} |y_i - f(x_i; \boldsymbol{c})|^2 \right\}. \tag{6.2}$$

A fim de trabalharmos com uma notação mais compacta, definimos o **resíduo** $\boldsymbol{r}(\boldsymbol{c}) = (r_1(\boldsymbol{c}), r_2(\boldsymbol{c}), \ldots, r_n(\boldsymbol{c}))$, onde $r_i(\boldsymbol{c}) := y_i - f(x_i; \boldsymbol{c})$. Com esta notação,

o **problema de mínimos quadrados** se resume a resolver

$$\min_{\boldsymbol{c}}\{E := \|r(\boldsymbol{c})\|_2^2\}. \tag{6.3}$$

6.1.1 Método das Equações Normais

A fim de resolver o problema de mínimos quadrados (6.3), observamos que o **erro quadrático**

$$E := \|\boldsymbol{r}(\boldsymbol{c})\|_2^2 \tag{6.4a}$$
$$= \sum_{i=1}^{n} r_i^2(\boldsymbol{c}) \tag{6.4b}$$
$$= \sum_{i=1}^{n} \left(y_i - f(x_i;\boldsymbol{c})\right)^2 \tag{6.4c}$$
$$= \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} c_j f_j(x_i)\right)^2 \tag{6.4d}$$
$$E = \|\boldsymbol{y} - A\boldsymbol{c}\|_2^2, \tag{6.4e}$$

onde $\boldsymbol{y} := (y_1, y_2, \ldots, y_n)$ e

$$A := \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_m(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_m(x_n) \end{bmatrix}. \tag{6.5}$$

Os parâmetros c_j que minimizam o erro E são solução do seguinte sistema de equações

$$\frac{\partial E}{\partial c_j} = 2\sum_{i=0}^{n} r_i(c)\frac{\partial}{\partial c_j} r_i(c) = 0, \tag{6.6}$$

onde $j = 1, 2, \ldots, m$. Ou, em uma notação mais apropriada,

$$\nabla_{\boldsymbol{c}} E = 0 \tag{6.7}$$
$$A^T \boldsymbol{r}(\boldsymbol{c}) = 0 \tag{6.8}$$
$$A^T(\boldsymbol{y} - A\boldsymbol{c}) = 0 \tag{6.9}$$

$$A^T A\boldsymbol{c} = A^T\boldsymbol{y}. \tag{6.10}$$

Portanto, o problema linear de mínimos quadrados se resume em resolver as chamadas **equações normais**

$$A^T A\boldsymbol{c} = A^T\boldsymbol{y}. \tag{6.11}$$

Concluímos a solução é dada por

$$\boldsymbol{c} = (A^T A)^{-1} A^T\boldsymbol{y}, \tag{6.12}$$

quando $A^T A$ é inversível.

Exemplo 6.1.1. (Ajuste de polinômios.) Considere o problema de ajustar o conjunto de pontos

i	x_i	y_i
1	-1.0	1.2
2	0.0	-0.1
3	1.0	0.7
4	1.5	2.4

por um polinômio quadrático da forma

$$p(x) = p_1 x^2 + p_2 x + p_3 \tag{6.13}$$

no sentido de mínimos quadrados.

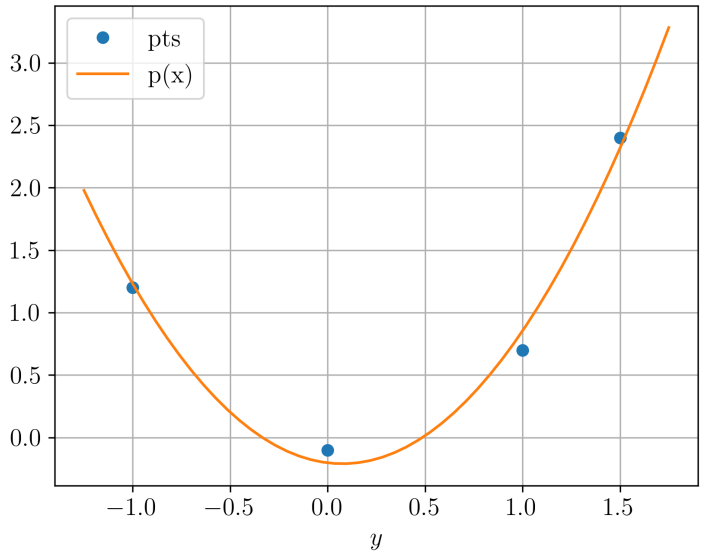

Figura 6.1: Esboço do polinômio ajustado no Exemplo 6.1.1.

Neste caso, a família de funções do problema de mínimos quadrados é $f_1(x) = x^2$, $f_2(x) = x$ e $f_3(x) = 1$. Assim sendo, os coeficientes $p = (p_1, p_2, p_3)$ são solução do seguinte sistema linear

$$A^T A\boldsymbol{p} = A^T\boldsymbol{y}, \tag{6.14}$$

onde $\boldsymbol{y} = (y_1, y_2, y_3)$ e

$$A := \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \\ x_4^2 & x_4 & 1 \end{bmatrix}. \tag{6.15}$$

Emfim, resolvendo as equações normais (6.14), obtemos

$$p(x) = 1.25x^2 - 0.188x - 0.203. \tag{6.16}$$

A Figura 6.1 mostra um esboço dos pontos e do polinômio ajustado.

Código 6.1: mqPoli.py

```
import numpy as np
import numpy.linalg as npla

# dados
xx = np.array([-1.0,
               0.0,
               1.0,
               1.5])
yy = np.array([1.2,
                -0.1,
                0.7,
                2.4])

# matriz
m = 3
n = xx.size
A = np.empty((n,m))
A[:,0] = xx**2
A[:,1] = xx
A[:,2] = np.ones_like(xx)

# sol de mq
p = npla.solve(A.T@A, A.T@yy)
print(p)
```

Exemplo 6.1.2. (Ajuste de curvas.) Consideremos o mesmo conjunto de pontos do exemplo anterior (Exemplo 6.1.1). Aqui, vamos ajustar uma curva da forma

$$f(x;\boldsymbol{c}) = c_1 \operatorname{sen}(x) + c_2 \cos(x) + c_3 \tag{6.17}$$

no sentido de mínimos quadrados. Para tanto, formamos a matriz

$$A := \begin{bmatrix} \operatorname{sen}(x_1) & \cos(x_1) & 1 \\ \operatorname{sen}(x_2) & \cos(x_2) & 1 \\ \operatorname{sen}(x_3) & \cos(x_3) & 1 \\ \operatorname{sen}(x_4) & \cos(x_4) & 1 \end{bmatrix} \tag{6.18}$$

e, então, resolvemos as equações normais $A^T A\boldsymbol{c} = A^T\boldsymbol{y}$ para o vetor de coeficientes $\boldsymbol{c} = (c_1, c_2)$. Fazendo isso, obtemos $c_1 = -0,198$, $c_2 = -2.906$ e

$c_3 = 2.662$. A Figura 6.2 mostra um esboço da curva ajustada aos pontos dados.

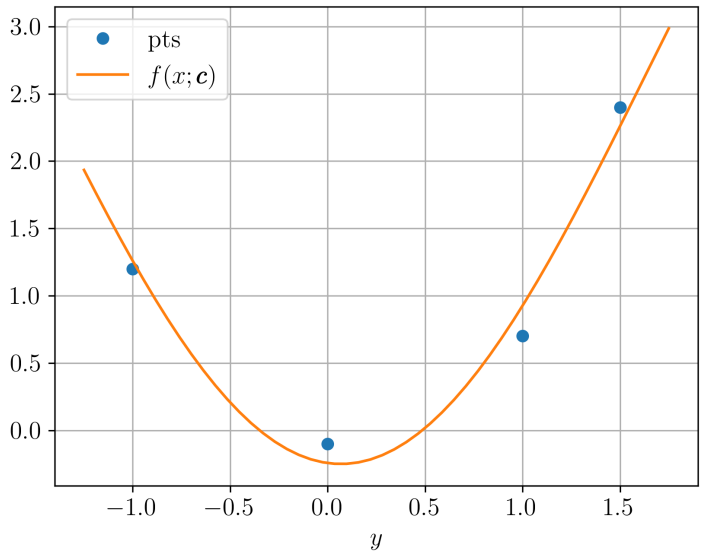

Figura 6.2: Esboço da curva ajustada no Exemplo 6.1.2.

Código 6.2: mqCurva.py

```
1 import numpy as np
2 import numpy.linalg as npla
3
4 # dados
5 xx = np.array([-1.0,
6                0.0,
7                1.0,
8                1.5])
9 yy = np.array([1.2,
10                -0.1,
11                0.7,
12                2.4])
13
14 # matriz
15 m = 3
```

```
16 n = xx.size
17 A = np.empty((n,m))
18 A[:,0] = np.sin(xx)
19 A[:,1] = np.cos(xx)
20 A[:,2] = np.ones_like(xx)
21
22 # sol de mq
23 c = npla.solve(A.T@A, A.T@yy)
24
25 def f(x, c=c):
26     y = c[0]*np.sin(x) \
27         + c[1]*np.cos(x) \
28         + c[2]
29     return y
```

Exemplo 6.1.3. (Um problema não linear.) Consideremos o problema de ajustar, no sentido de mínimos quadrados, a função

$$f(x;\boldsymbol{c}) = c_1 e^{c_2 x} \tag{6.19}$$

ao seguinte conjunto de pontos

i	x_i	y_i
1	-1.0	8.0
2	0.0	1.5
3	1.0	0.2
4	1.5	0.1

Aqui, temos um problema não linear de mínimos quadrados que pode ser transformado em um problema linear fazendo-se

$$y = c_1 e^{c_2 x} \tag{6.20}$$

$$\ln y = \ln c_1 e^{c_2 x} \tag{6.21}$$

$$\ln y = \ln c_1 + c_2 x. \tag{6.22}$$

Isto é, denotando $d_1 := \ln c_1$ e $d_2 := c_2$, o problema se resume a ajustar uma reta $r(x) = d_1 + d_2 x$ ao conjunto de pontos $\{(x_i, \ln y_i)\}_{i=1}^4$.

Para resolver o problema transformado, formamos a matriz

$$A := \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{bmatrix} \tag{6.23}$$

e, então, resolvemos as equações normais $A^T A\boldsymbol{d} = A^T \ln \boldsymbol{y}$, com $\ln \boldsymbol{y} = (\ln y_1, \ln y_2, \ln y_3, \ln y_4)$, donde obtemos $d_1 = 0.315$ e $d_2 = -1.792$. Das definições de d_1 e d_2, temos

$$c_2 = d_2 = -1.792 \tag{6.24}$$

$$c_1 = e^{d_1} = 1.371. \tag{6.25}$$

A Figura 6.3 mostra um esboço da curva $f(x; \boldsymbol{c}) = c_1 e^{c_2 x}$ ajustada aos pontos dados.

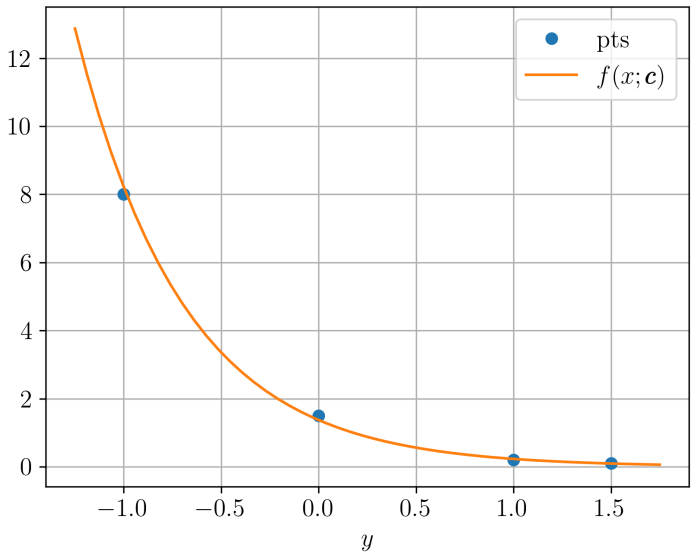

Figura 6.3: Esboço da curva ajustada no Exemplo 6.1.3.

6.1.2 Análise Numérica

O problema linear de mínimos quadrados (6.3) reduz-se a resolver o sistema linear (6.11) para $\boldsymbol{c}$. Isto nos leva a questão de verificar se $A^T A$ é invertível.

Teorema 6.1.1. *A matriz A^TA é positiva definida se, e somente se, as colunas de A são linearmente independentes.*

Demonstração. Se as colunas de A são linearmente independentes, então $x \neq 0$ implica $Ax \neq 0$ e, equivalentemente, $x^TA^T \neq 0$. Portanto, $x \neq 0$ implica $x^TA^TAx = \|Ax\|_2^2 > 0$, o que mostra que A^TA é positiva definida.

Suponhamos, agora, que as colunas de A não são linearmente independentes. Então, existe $x_0 \neq 0$ tal que $Ax_0 = 0$. Mas, então, $x_0^TA^TAx_0 = 0$, o que mostra que A^TA não é positiva definida. □

Este teorema nos fornece uma condição suficiente para a existência (e unicidade) de solução do problema linear de mínimos quadrados. Mais especificamente, se as colunas da matriz A são linearmente independentes, então os coeficientes da função $f(x)$ que melhor ajustam os pontos dados são

$$c = (A^TA)^{-1}A^Ty. \tag{6.26}$$

6.1.3 Exercícios

E.6.1.1. Determine a reta $y = c_1x + c_2$ que melhor se ajusta, no sentido de mínimos quadrados, aos pontos

i	1	2	3	4	5
x_i	-2.5	-1.3	0.2	1.7	2.3
y_i	3.8	1.5	-0.7	-1.5	-3.2

Por fim, compute a norma L^2 do resíduo, i.e. $\|r(c)\|_2 = \|y - (c_1x + c_2)\|_2$ para os pontos dados.

E.6.1.2. Determine o polinômio $y = c_1x^3 + c_2x^2 + c_3x + c_4$ que melhor se ajusta, no sentido de mínimos quadrados, aos pontos

i	1	2	3	4	5
x_i	-2.5	-1.3	0.2	1.7	2.3
y_i	3.8	0.5	2.7	1.2	-1.3

Por fim, compute a norma L^2 do resíduo, i.e. $\|r(c)\|_2$.

E.6.1.3. Determine a curva $y = c_1 \operatorname{sen} x + c_2 \cos x + c_3$ que melhor se ajusta, no sentido de mínimos quadrados, aos pontos

i	1	2	3	4	5
x_i	-2.5	-1.3	0.2	1.7	2.3
y_i	3.8	0.5	2.7	1.2	-1.3

Por fim, compute a norma L^2 do resíduo, i.e. $\|r(c)\|_2$.

E.6.1.4. Use a transformação $z = \ln y$ para ajustar, no sentido de mínimos quadrados, a curva $y = c_1 e^{c_2(x-c_3)^2}$ aos pontos

i	1	2	3	4	5	6
x_i	-0.5	0.5	1.3	2,1	2.7	3,1
y_i	0,1	1.2	2.7	0.9	0.2	0,1

Respostas

E.6.1.1. $c_1 = -1.3259$, $c_2 = 8.66071\text{e}{-2}$, $\|r(c)\|_2 = 1.01390$.

E.6.1.2. $c_1 = -4.50361\text{e}{-1}$, $c_2 = -2.78350\text{e}{-1}$, $c_3 = 1.46291$, $c_4 = 2.09648$, $\|r(c)\|_2 = 5.71346\text{e}{-1}$

E.6.1.3. $c_1 = -0.86290414$, $c_2 = 0.36547042$, $c_3 = 1.4700346$, $\|r(c)\|_2 = 3.616409$

E.6.1.4. $c_1 = 2.10131\text{e}{+0}$, $c_2 = -9.73859\text{e}{-1}$, $c_3 = 1.25521\text{e}{+0}$

6.2 Problemas Não Lineares

Em revisão

Um problema não linear de mínimos quadrados consiste em ajustar uma dada

função $f(x;c)$ que dependa não linearmente dos parâmetros $c = (c_1, c_2, \ldots, c_m)$, $m \geq 1$, a um dado conjunto de $n \geq m$ pontos $\{(x_i, y_i)\}_{i=1}^n$. Mais especificamente, buscamos resolver o seguinte problema de minimização

$$\min_{\{c_1,c_2,\ldots,c_m\}} \left[E := \sum_{i=1}^{n} \left(y_i - f(x_i;c)\right)^2 \right]. \tag{6.27}$$

Aqui, denotaremos por $r(c) := (r_1(c), r_2(c), \ldots, r_n(c))$ o vetor dos resíduos $r_i(c) := y_i - f(x_i, c)$. Com isso, o problema se resume a encontrar o vetor de parâmetros c que minimiza

$$E = \|r(c)\|_2^2. \tag{6.28}$$

Tais parâmetros são solução do seguinte sistema de equações

$$\frac{\partial E}{\partial c_j} = 2\sum_{i=1}^{n} r_i(c)\frac{\partial}{\partial c_j} r_i(c) = 0 \tag{6.29}$$

ou, equivalentemente, da equação

$$\nabla E = 0 \Leftrightarrow J_R^T(c) r(c) = 0, \tag{6.30}$$

onde

$$J_R(c) := \begin{bmatrix} \frac{\partial r_1}{\partial c_1} & \frac{\partial r_1}{\partial c_2} & \cdots & \frac{\partial r_1}{\partial c_m} \\ \frac{\partial r_2}{\partial c_1} & \frac{\partial r_2}{\partial c_2} & \cdots & \frac{\partial r_2}{\partial c_m} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_n}{\partial c_1} & \frac{\partial r_n}{\partial c_2} & \cdots & \frac{\partial r_n}{\partial c_m} \end{bmatrix} \tag{6.31}$$

é a jacobiana do resíduo r em relação aos parâmetros c.

Podemos usar o método de Newton para resolver (6.30). Para tanto, escolhemos uma aproximação inicial para $c^{(1)} = (c_1^{(1)}, c_2^{(1)}, \ldots, c_m^{(1)})$ e iteramos

$$H_R(c^{(k)})\delta^{(k)} = -J_R^T(c) r(c) \tag{6.32}$$

$$c^{(k+1)} = c^{(k)} + \delta^{(k)}, \tag{6.33}$$

onde $\delta^{(k)} = (\delta_1^{(k)}, \delta_2^{(k)}, \delta_m^{(k)})$ é a atualização de Newton (ou direção de busca) e $H_R(c) := [h_{p,q}(c)]_{p,q=1}^{m,m}$ é a matriz hessiana, cujos elementos são

$$h_{p,q} := \sum_{i=1}^{n} \left\{ \frac{\partial r_i}{\partial c_q}\frac{\partial r_i}{\partial c_p} + r_i \frac{\partial^2 r_i}{\partial c_q \partial c_p} \right\}. \tag{6.34}$$

Exemplo 6.2.1. Consideremos o problema de ajustar, no sentido de mínimos quadrados, a função

$$f(x;c) = c_1 e^{c_2 x} \tag{6.35}$$

ao seguinte conjunto de pontos

i	1	2	3	4
x_i	-1	0	1	1.5
y_i	8.0	1.5	0.2	0.1

Aqui, vamos utilizar a iteração de Newton para o problema de mínimos quadrados, i.e. a iteração dada em (6.32)-(6.33). Para tanto, para cada $i = 1, 2, 3, 4$, precisamos das seguintes derivadas parciais do resíduo $r_i(c) := y_i - c_1 e^{c_2 x_i}$:

$$\frac{\partial}{\partial c_1} r_i(c) = -e^{c_2 x_i}, \tag{6.36}$$

$$\frac{\partial}{\partial c_2} r_i(c) = -c_1 x_i e^{c_2 x_i}, \tag{6.37}$$

$$\frac{\partial^2}{\partial c_1^2} r_i(c) = 0, \tag{6.38}$$

$$\frac{\partial^2}{\partial c_1 \partial c_2} r_i(c) = \frac{\partial^2}{\partial c_2 \partial c_1} r_i(c) = -x_i e^{c_2 x_i}, \tag{6.39}$$

$$\frac{\partial^2}{\partial c_2^2} r_i(c) = -c_1 x_i^2 e^{c_2 x_i}. \tag{6.40}$$

Com isso e tomando $c^{(1)} = (1.4, -1.8)$ (motivado do Exemplo **??**), computamos as iterações de Newton (6.32)-(6.33). Iterando até a precisão de $TOL = 10^{-4}$, obtemos a solução $c_1 = 1.471$ e $c_2 = -1.6938$. Na Figura 6.4 vemos uma comparação entre a curva aqui ajustada ($-$) e aquela obtida no Exemplo **??** ($--$).

Observamos que a solução obtida no exemplo anterior (Exemplo 6.2.1) difere da previamente encontrada no Exemplo **??**. Naquele exemplo, os parâmetros obtidos nos fornecem $E = 6.8\text{e}{-2}$, enquanto que a solução do exemplo anterior fornece $E = 6.1\text{e}-3$. Isto é esperado, pois naquele exemplo resolvemos um problema aproximado, enquanto no exemplo anterior resolvemos o problema por si.

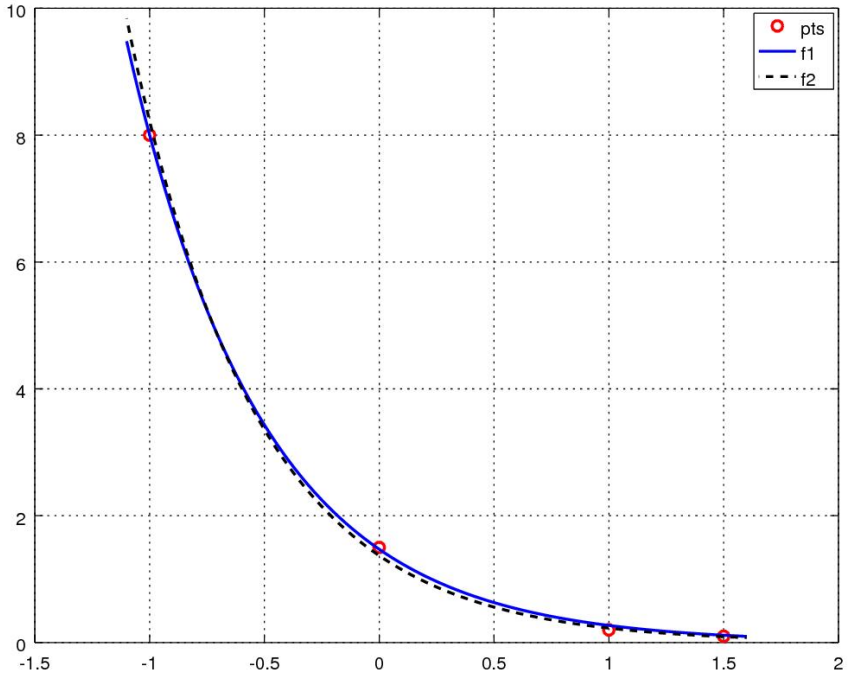

Figura 6.4: Esboço da curva ajustada no Exemplo 6.2.1.

O emprego do método de Newton para o problema de mínimos quadrados tem a vantagem da taxa de convergência quadrática, entretanto requer a computação das derivadas parciais de segunda ordem do resíduo. Na sequência discutimos alternativas comumente empregadas.

6.2.1 Método de Gauss-Newton

Em revisão

O método de Gauss-Newton é uma técnica iterativa que aproxima o problema não linear de mínimos quadrados (6.27) por uma sequência de problemas lineares. Para seu desenvolvimento, começamos de uma aproximação inicial $c^{(1)} = (c_1^{(1)}, c_2^{(1)}, \ldots, c_m^{(1)})$ dos parâmetros que queremos ajustar. Também, assumindo que a n-ésima iterada $c^{(k)}$ é conhecida, faremos uso da aproximação

de primeira ordem de $f(x, c)$ por polinômio de Taylor, i.e.

$$f(x; c^{(k+1)}) \approx f(x; c^{(k)}) + \nabla_c f(x; c^{(k)})(c^{(k+1)} - c^{(k)}), \tag{6.41}$$

onde

$$\nabla_c f(x; c) = \left[\frac{\partial}{\partial c_1} f(x; c) \ \frac{\partial}{\partial c_2} f(x; c) \ \cdots \ \frac{\partial}{\partial c_m} f(x; c) \right]. \tag{6.42}$$

O método consiste em obter a solução do problema não linear (6.27) pelo limite dos seguintes problemas lineares de mínimos quadrados

$$\min_{\delta^{(k)}} \left[\tilde{E} := \sum_{i=1}^{n} (y_i - f(x_i, c^{(k)}) - \nabla_c f(x_i; c^{(k)}) \delta^{(k)})^2 \right] \tag{6.43}$$

$$c^{(k+1)} = c^{(k)} + \delta^{(k)}. \tag{6.44}$$

Agora, usando a notação de resíduo $r(c) = y - f(x; c)$, observamos que (6.50) consiste no problema linear de mínimos quadrados

$$\min_{\delta^{(k)}} \| r(c^{(k)}) + J_R(c^{(k)}) \delta^{(k)} \|_2^2, \tag{6.45}$$

o qual é equivalente a resolver as equações normais

$$J_R^T(c^{(n)}) J_R(c^{(n)}) \delta^{(n)} = -J_R^T(c) r(c). \tag{6.46}$$

Com isso, dada uma aproximação inicial $c^{(1)}$, a **iteração do método de Gauss-Newton** consiste em

$$J_R^T(c^{(k)}) J_R(c^{(k)}) \delta^{(k)} = -J_R^T(c) r(c) \tag{6.47}$$

$$c^{(k+1)} = c^{(k)} + \delta^{(k)}. \tag{6.48}$$

Exemplo 6.2.2. A aplicação da iteração de Gauss-Newton ao problema de mínimos quadrados discutido no Exemplo 6.2.1 nos fornece a mesma solução obtida naquele exemplo (preservadas a aproximação inicial e a tolerância de precisão).

O método de Gauss-Newton pode ser lentamente convergente para problemas muito não lineares ou com resíduos grandes. Nesse caso, métodos de Gauss-Newton com amortecimento são alternativas robustas [1, 5]. Na sequência, introduziremos um destes métodos, conhecido como método de Levenberg-Marquardt.

6.2.2 Método de Levenberg-Marquardt

Em revisão

O método de Levenberg-Marquardt é uma variação do método de Gauss-Newton no qual a direção de busca $\delta^{(n)}$ é obtida da solução do seguinte problema regularizado

$$\min_{\delta^{(k)}}\{\|r(c^{(k)}) + J_R(c^{(k)})\delta^{(k)}\|_2^2 + \mu^{(k)}\|\delta^{(k)}\|_2^2\} \tag{6.49}$$

ou, equivalentemente,

$$\min_{\delta^{(k)}} \left\| \begin{bmatrix} r(c^{(k)}) \\ 0 \end{bmatrix} + \begin{bmatrix} J_R(c^{(k)}) \\ \mu^{(k)} I \end{bmatrix} \delta^{(k)} \right\|_2^2 \tag{6.50}$$

A taxa de convergência das iterações de Levenberg-Marquardt é sensível a escolha do parâmetro $\mu^{(k)} \geq 0$. Aqui, faremos esta escolha por tentativa e erro. O leitor pode aprofundar-se mais sobre esta questão na literatura especializada (veja, por exemplo, [1, 5]).

Observação 6.2.1. Quando $\mu^{(k)} \equiv 0$ para todo n, o método de Levenberg-Marquardt é equivalente ao método de Gauss-Newton.

Exemplo 6.2.3. Consideremos o problema de mínimos quadrados discutido no Exemplo 6.2.1. O método de Gauss-Newton falha para este problema se escolhermos, por exemplo, $c^{(1)} = (0, 0)$. Isto ocorre pois, para esta escolha de $c^{(1)}$, a jacobiana $J(c^{(1)})$ não tem posto completo. Entretanto, o método de Levenberg-Marquardt com $\mu^{(k)} = 0.1$ é convergente, mesmo para esta escolha de $c^{(1)}$.

6.2.3 Exercícios

Em revisão

E.6.2.1. Use o método de Gauss-Newton para ajustar, no sentido de mínimos quadrados e com precisão de 10^{-4}, a curva $y = c_1 e^{c_2(x-c_3)^2}$ aos pontos

i	1	2	3	4	5	6
x_i	-0.5	0.5	1.3	2.1	2.7	3.1
y_i	0.1	1.2	2.7	0.9	0.2	0.1

Use as condições iniciais:

a) $c_1 = 2.1$, $c_2 = -1$ e $c_3 = 1.3$.

b) $c_1 = 1$, $c_2 = -1$ e $c_3 = -1$.

E.6.2.2. Resolva o exercício anterior (Exercício 6.2.1) usando o método de Levenberg-Marquardt com amortecimento constante $\mu = 0.2$.

Respostas

E.6.2.1. a) $c_1 = 2.69971\text{e}+0$, $c_2 = -1.44723\text{e}+0$, $c_3 = 1.24333\text{e}+0$; b) divergente.

E.6.2.2. a) $c_1 = 2.69971\text{e}+0$, $c_2 = -1.44723\text{e}+0$, $c_3 = 1.24333\text{e}+0$; b) $c_1 = 2.69971\text{e}+0$, $c_2 = -1.44723\text{e}+0$, $c_3 = 1.24333\text{e}+0$

Notas

[1]Para bases $b \geq 11$, usamos a representação dos dígitos maiores ou iguais a 10 por letras maiúsculas do alfabeto latino, i.e. $A = 10$, $B = 11$, $C = 12$ e assim por diante.

[2]8 *bits* = 1 *byte* [B].

[3]Padrão IEEE 754.

[4]Esta função não é precisa e pode fornecer registros errados devido a erros de arredondamento. Uma alternativa melhor é apresentada na Observação 1.2.2.

[5]No caso do número zero, temos $d_0 = 0$.

[6]Consulte na *web* por Python Docs:String: Format Specification Mini-Language para uma lista completa.

[7]Para mais detalhes, consulte IEEE 754: Wikipedia.

[8]Assumindo o arredondamento por proximidade com desempate par.

[9]Esta é uma consequência imediata do Teorema do Valor Intermediário.

[10]O problema foi construído para que tivesse estas soluções.

[11]De fato, f tem três zeros no intervalo $(-2, 3)$.

[12]Caso, $f\left(a^{(k)}\right) = 0$ ou $f\left(a^{(b)}\right) = 0$, então assumimos que $a^{(k+i)} = a^{(k)}$ ou $b^{(k+i)} = b^{(k)}$, conforme o caso, para $i = 1, 2, 3, \ldots$.

[13]Função estritamente crescente ou estritamente decrescente, exclusivamente.

[14]Caso, $f\left(a^{(k)}\right) = 0$ ou $f\left(a^{(b)}\right) = 0$, então assumimos que $a^{(k+i)} = a^{(k)}$ ou $b^{(k+i)} = b^{(k)}$, conforme o caso, para $i = 1, 2, 3, \ldots$.

[15]Definimos que x é ponto crítico de uma dada f, quando $f'(x) = 0$ ou $\nexists f'(x)$.

[16]Definimos que x é ponto crítico de uma dada f, quando $f'(x) = 0$ ou $\nexists f'(x)$.

[17]Definimos que x é ponto crítico de uma dada f, quando $f'(x) = 0$ ou $\nexists f'(x)$.

[18]Johan Frederik Steffensen, 1873 - 1961, matemático e estatístico dinamarquês. Fonte: Wikipédia.

[19]Alexander Aitken, 1895 - 1967, matemático neozelandês. Fonte: Wikipédia.

[20]Definimos que x é ponto crítico de uma dada f, quando $f'(x) = 0$ ou $\nexists f'(x)$.

[21]Brook Taylor, 1685 - 1731, matemático britânico. Fonte: Wikipédia:Brook Taylor.

[22]Sir Isaac Newton, matemático e físico inglês, 1642 - 1726/27. Fonte: Wikipedia.

[23]Supondo verdadeiras as demais hipóteses do Teorema 2.3.1.

[24]Eixo x.

[25]Veja o exercício 2.5.6.

[26]Equação (2.156).

[27]Razão fundamental do Cálculo.

[28]Definimos que x é ponto crítico de uma dada f, quando $f'(x) = 0$ ou $\nexists f'(x)$.

[29]William George Horner, matemático britânico, 1786 - 1837. Fonte: Wikipedia

[30]Alexandre-Théophile Vandermonde, 1735 - 1796, matemático francês. Fonte: Wikipédia: Alexandre-Théophile Vandermonde.

[31]Alexandre-Théophile Vandermonde, 1735 - 1796, matemático francês. Fonte: Wikipédia: Alexandre-Théophile Vandermonde.

[32]Augustin-Louis Cauchy, 1789-1857, matemático francês. Fonte: Wikipédia: Augustin-Louis Cauchy.

[33]Karl Hermann Amandus Schwarz, 1843-1921, matemático alemão. Fonte: Wikipédia: Hermann Amandus Schwarz.

[34]Consulte [7, Section 1.3-3] para informações sobre a demonstração.

[35]Euclides de Alexandria, 300 a.C., matemático grego. Fonte: Wikipédia: Euclides.

[36]Carl Gustav Jakob Jacobi, 1804 - 1851, matemático alemão. Fonte: Wikipédia: Carl Gustav Jakob Jacobi.

[37]Johann Carl Friedrich Gauss, 1777 - 1855, matemático alemão. Fonte: Wikipédia: Carl Friedrich Gauss.

[38]Philipp Ludwig von Seidel, 1821 - 1896, matemático alemão. Fonte: Wikipédia: Philipp Ludwig von Seidel.

[39]$\rho(T)$ é o raio espectral da matriz T, i.e. o máximo dos módulos dos autovalores de T.

[40] A é simétrica e $\boldsymbol{x}^T A\boldsymbol{x} > 0$ para todo $\boldsymbol{x} \neq 0$.

[41] São ortogonais em relação ao produto interno induzido por A, $< \boldsymbol{u}, \boldsymbol{v} >_A := \boldsymbol{u} \cdot A\boldsymbol{v}$.

[42] Brook Taylor, 1685 - 1731, matemático britânico. Fonte: Wikipédia:Brook Taylor.

[43] Carl Gustav Jakob Jacobi, 1804 - 1851, matemático alemão. Fonte: Wikipédia: Carl Gustav Jakob Jacobi.

[44] Carl Gustav Jakob Jacobi, 1804 - 1851, matemático alemão. Fonte: Wikipédia: Carl Gustav Jakob Jacobi.

[45] Carl Gustav Jakob Jacobi, 1804 - 1851, matemático alemão. Fonte: Wikipédia: Carl Gustav Jakob Jacobi.

[46] Carl Gustav Jakob Jacobi, 1804 - 1851, matemático alemão. Fonte: Wikipédia: Carl Gustav Jakob Jacobi.

[47] Carl Gustav Jakob Jacobi, 1804 - 1851, matemático alemão. Fonte: Wikipédia: Carl Gustav Jakob Jacobi.

[48] Isaac Newton, 1642 - 1727, matemático, físico, astrônomo, teólogo e autor inglês. Fonte: Wikipédia: Isaac Newton.

[49] Para informações mais precisas sobre a convergência do Método de Newton, consulte [8, Seção 5.3].

[50] Chamado de **polinômio interpolador**.

[51] Alexandre-Théophile Vandermonde, 1735 - 1796, matemático francês. Fonte: Wikipédia: Alexandre-Théophile Vandermonde.

[52] Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[53] Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[54] Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[55] Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[56] Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[57]Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[58]Joseph-Louis Lagrange, 1736 - 1813, matemático italiano. Fonte: Wikipédia: Joseph-Louis Lagrange.

[59]Isaac Newton, 1642 - 1727, matemático, físico, astrônomo, teólogo e autor inglês. Fonte: Wikipédia: Isaac Newton.

[60]Consulte o Exercício 5.3.5

Referências

[1] Björk, A.. Numerical Methods for Least Squares Problems. SIAM, 1996.

[2] Burden, R.L.; Faires, J.D.; Burden, A.M.. Análise Numérica. 3. ed., Cengage Learning, 2016. ISBN: 978-8522123414. Acesso SABI+UFRGS.

[3] Isaacson, E.; Keller H.B.. Analysis of Numerical Methods. Dover, 1994.

[4] Lemire, D.. Number Parsing at a Gigabyte per Second. Software: Practice and Experience, 51(8), 2021, 1700-1727. DOI: 10.1002/spe.2984.

[5] Nocedal, J.; Wright, S.J.. Numerical Optimization. Springer, 2006.

[6] Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P.. Numerical Recipes. 3. ed., Cambridge University Press, 2007.

[7] Ralston, A.; Rabinowitz, P.. A First Course in Numerical Analysis. 2. ed., Dover: New York, 2021. ISBN 048641454X.

[8] Stoer, J.; Bulirsch, R.. Introduction to numerical analysis. 2. ed., Springer-Verlag, 1993.

Índice de Comandos