Problemas, Algoritmos e Implementações

Cláudio Nogueira de Meneses
claudio.meneses@ufabc.edu.br

20 de abril de 2024

A problem defined, is a problem half solved.
Albert Einstein (1879-1955)

1 Problemas básicos

Iniciamos com a definição de algoritmo.

Definição 1.1 *Um* **algoritmo** *é uma sequência finita de instruções simples que realiza alguma tarefa.*

Definição 1.2 *Um* **programa de computador** *(ou simplesmente, programa) é uma sequência finita de instruções simples que realiza alguma tarefa em um computador.*

Definição 1.3 *Todo programa é um algoritmo.*

A seguir veremos descrições de problemas, algoritmos para resolvê-los e implementações na linguagem de programação Python destes algoritmos.

Problema. 1.1 *Encontre as raízes reais da equação* $ax^2+bx+c=0$*, onde* $a, b, c \in \mathbb{R}$.

Solução: É sabido que se $a \neq 0$, então podemos calcular as raízes da equação do segundo grau analisando o valor de $\Delta = b^2 - 4ac$, que é chamado de discriminante. Isto é,

Se $\Delta = 0$ então as duas raízes são reais e iguais;
Se $\Delta > 0$ então as duas raízes são reais e diferentes;
Se $\Delta < 0$ então as duas raízes são diferentes e imaginárias.

Por raízes imaginárias, quero dizer que elas pertencem ao conjunto dos números complexos.

Como estamos interessados em raízes reais, analisamos somente as situações onde $\Delta = 0$ e $\Delta > 0$. Nestes casos, usando a fórmula de Bhaskara, x_1 e x_2 são:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$
$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Algoritmo 1.1: Cálculo das raízes de uma equação do segundo grau.

read a, b, c
if $a = 0$ **then**
  print "O coeficiente de x^2 precisa ser diferente de 0."
else
  $delta \leftarrow b^2 - 4ac$
  if $delta < 0$ **then**
    print "Como $b^2 - 4ac < 0$ as raízes serão números complexos."
  else
    $x_1 \leftarrow (-b + \sqrt{delta})/2a$
    $x_2 \leftarrow (-b - \sqrt{delta})/2a$
    print "As raízes da equação são: " x_1 e x_2

```
1  # Formula de Bhaskara (implementacao em Python)
2  # Claudio N. Meneses, 08/02/24
3  import math
4
5  print('Problema 1.1. Encontrar as raizes de uma equacao ax^2 + bx + c = 0')
6  a = int(input('Coeficiente de x^2: '), 10)    #inteiro na base 10
7  b = int(input('Coeficiente de x: '), 10)      #inteiro na base 10
8  c = int(input('Termo constante: '), 10)       #inteiro na base 10
9  print(a,b,c)
10 if a == 0:
11     print('O valor de a precisa ser maior do que zero')
12 else:
13     delta = b*b - 4 * a * c
14     if delta < 0:
15         print('Como b^2 - 4ac < 0 entao as raizes serao numeros complexos')
16     else:
17         x1 = (-b + math.sqrt(delta))/(2*a)
18         x2 = (-b - math.sqrt(delta))/(2*a)
19         print('As raizes da equacao sao: ', x1, x2)
```

Para se tornar uma implementação mais geral, poderíamos trocas as linhas 6, 7 e 8 na implementação acima, de `int(...)` para `float(...)`. □

Problema. 1.2 *Encontre a média aritmética de quatro números reais a, b, c, d.*

Solução: A fórmula da média aritmética dos números reais a, b, c, d é dada por

$$\text{Média} = \frac{a + b + c + d}{4}$$

Algoritmo 1.2: Média de quatro números reais.

read a, b, c, d
Media $\leftarrow (a + b + c + d)/4$
print "A média é: " Media

```python
# Media aritmetica (implementacao em Python)
# Claudio N. Meneses, 08/02/24

print('Problema 1.2. Media aritmetica entre quatro numeros reais (a,b,c,d)')
a = float(input('Primeiro numero: '))
b = float(input('Segundo numero: '))
c = float(input('Terceiro numero: '))
d = float(input('Quarto numero: '))
print(a,b,c,d)
media = (a + b + c + d)/4
print('A media eh ', media)
```

Problema. 1.3 *Gere os n primeiros termos da sequência de Fibonacci:* $0, 1, 1, 2, 3, 5, 8, \ldots$.

Solução: Veja que a sequência de Fibonacci é definida pela fórmula recursiva

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \quad n = 2, 3, 4, \ldots \end{aligned}$$

A partir da fórmula acima, criamos os dois algoritmos abaixo. O segundo usa estruturas de controle de fluxo de decisão `if then else`. No primeiro, o valor de n é verificado sempre quatro vezes. Ou seja, testa se $n \leq 0$, se $n = 1$, se $n = 2$ e se $n \geq 3$. No segundo algoritmo, o teste é feito no máximo três vezes. Ou seja, se $n \leq 0$, se $n = 1$ e se $n = 2$.

Algoritmo 1.3: Sequência de Fibonacci.

read n
if $n \leq 0$ **then**
  print "O valor de n precisa ser um numero maior ou igual a um"
if $n = 1$ **then**
  print 0
if $n = 2$ **then**
  print 0,1
if $n \geq 3$ **then**
  $x \leftarrow 0$
  $y \leftarrow 1$
  print x, y
  for $t = 3$ **to** n **do**
    $z \leftarrow x$
    $x \leftarrow y$
    $y \leftarrow z + y$
    print y

Algoritmo 1.4: Sequência de Fibonacci.

```
read n
if n ≤ 0 then
    print "O valor de n precisa ser um numero maior ou igual a um"
else if n = 1 then
    print 0
else if n = 2 then
    print 0, 1
else
    x ← 0
    y ← 1
    print x, y
    for t = 3 to n do
        z ← x
        x ← y
        y ← z + y
        print y
```

```python
# Sequencia de Fibonacci (implementacao em Python)
# Claudio N. Meneses, 08/02/24

print('Problema 1.3. n primeiros termos da Sequencia de Fibonacci')
n = int(input('Digite o numero de termos a serem gerados: '), 10)        #inteiro na base 10
if n <= 0:
   print('O valor de n precisa ser maior ou igual a um')
elif n == 1:
   print('0')
elif n == 2:
   print('0 1')
else:
    x = 0
    y = 1
    # troca end de '\n' (newline) para um espaco.
    print(x, y, end = ' ')
    # range(r,s) cria uma sequencia de r a s-1
    for i in range(3, n+1, 1):
       z = x
       x = y
       y = z + y
       print(y, end = ' ')
```

Problema. 1.4 *Dado um inteiro positivo K, some todos os números pares da sequência de Fibonacci que são menores do que K.*

Solução: Escrevendo diretamente o programa para resolver o problema, temos para um dado $K = 4.000.000$:

```python
a=b=1
limite=4000000
soma = 0
while b < limite:
    if (b % 2 == 0):
        soma = soma +b
    print(a,' ',b,' ')
    a,b = b,a+b
print('a=%d   b=%d   soma= %d' %(a,b,soma))
```

Problema. 1.5 *Encontre uma solução do sistema de equações*

$$ax + by = u$$
$$cx + dy = v$$

resolvendo para x e y, isto é,

$$x = \frac{d}{ad-bc}u - \frac{b}{ad-bc}v$$
$$y = \frac{-c}{ad-bc}u + \frac{a}{ad-bc}v$$

onde a, b, c, d, u, v são números reais, $a \neq 0$ e $ad - bc \neq 0$.

Solução: Um algoritmo para resolver o sistema de equações é como segue:

Algoritmo 1.5: Solução de sistema de equações.

```
read a, b, c, d, u, v
D ← ad − bc
if D = 0 then
    print "ad − bc precisa ser diferente de 0."
else
    x ← (du − bv)/D
    y ← (av − uc)/D
    print x, y
```

Problema. 1.6 *Dada a função $f(x) = x^3 - 3x^2 + 1$, é possível encontrar qualquer das raízes reais de $f(x) = 0$ via aproximações sucessivas usando a seguinte fórmula:*

$$x_{n+1} = x_n - \frac{x_n^3 - 3x_n^2 + 1}{3x_n^2 - 6x_n} \quad n = 0, 1, 2, \ldots$$

Encontre a n-ésima aproximação de uma das três raízes da equação $f(x) = 0$ para algum x_0 apropriado.

Solução: Este método de aproximações sucessivas para aproximar as raízes de uma equação foi proposto, de maneira independente, por Isaac Newton (inglês, 1642-1727) e Joseph Raphson (inglês, 1668-1712). Veja que a fórmula acima pode ser reescrita como:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad n = 0, 1, 2, \ldots \tag{1.1}$$

onde $f'(x_n) \neq 0$. Se $f'(x_n)$ for igual a zero em alguma iteração n, o valor x_n encontrado pode não ser uma aproximação para uma raiz de $f(x) = 0$.

Usando fórmula 1.1, construímos o Algoritmo 1.6. Observando o gráfico mostrado na Figura 1.1, é recomendável que escolhamos o valor de x_0 em um dos intervalos $[-1, 0]$, $[\frac{1}{2}, 1]$, $[\frac{5}{2}, 3]$, pois as três raízes encontram-se nestes intervalos. É sabido que se x_0 for "distante" de uma raiz, o método pode não convergir para uma raiz da função.

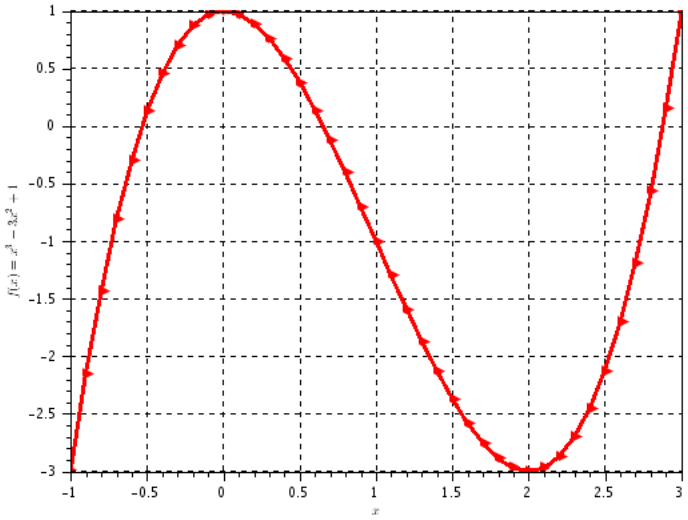

Figura 1.1: Função no Problema 1.6. Desenho feito no Scilab. com o código mostrado no Apêndice A

.

Algoritmo 1.6: Raiz de $x^3 - 3x^2 + 1 = 0$ próxima ao ponto x_0.

read x, num_iter // x_0 e número de iterações do método
for $n \leftarrow 1$ **to** num_iter **do**
  $dfx \leftarrow 3x^2 - 6x$
  if $dfx = 0$ **then**
    print "A primeira derivada de $f(x)$ no ponto ", x, "precisa ser diferente de 0."
    break
  else
    $f_x \leftarrow x^3 - 3x^2 + 1$
    $x \leftarrow x - f_x/d$
if $d = 0$ **then** $f_x \leftarrow x^3 - 3x + 1$
if $n = num\_iter + 1$ **then** $n \leftarrow n - 1$
print "Na iteração ", n
print "valor de x: ", x
print "valor de $f(x)$:", f_x

```
# Raiz aproximada de um dado polinomio do terceiro grau - Incluindo tratamento de excecoes
# (implementacao em Python)
# Claudio N. Meneses, 08/02/24
EPSILON = 1.0e-6    # constante para testar se um numero eh pequeno

while True:
    try:
        print('Problema 1.5. Raiz de um dado polinomio do terceiro grau.')
        x = float(input('Raiz aproximada inicial: '))
        num_iter = int(input('Numero de iteracoes do metodo: '), 10)
```

```
        if (num_iter <= 0):
            print("\nNumero de iteracoes precisa ser um numero inteiro positivo.\n")
            continue
        break
    except:
        print("\nErro: valor a ser digitado precisa ser um numero.\n")

print('n            x_n                    f(x_n)')
print('-------------------------------------------------------')
n=0
f_x = pow(x,3) - 3*x*x +1
print(n,'    ', x,'                  ',f_x)
for n in range(1,num_iter+1):
    dfx = 3*x*x - 6*x           # derivada de f no ponto x
    if abs(dfx) < EPSILON:      # valor da derivada em x eh muito pequeno?
        print('O valor da primeira derivada de f(x) em ',x,' eh muito pequeno')
        break
    else:
        x = x - f_x/dx
        f_x = pow(x,3) - 3*x*x +1
        print(n,'    ', x,'    ',f_x)
```

Resumo do que foi aprendido na SEMANA 1

O Algoritmo 1.6 tem os três "ingredientes" básicos que qualquer algoritmo pode possuir. São eles:

- **variáveis** (neste caso, $iter, d, n, x$);
- **estruturas de controle** do fluxo do que é feito durante a execução (neste caso, `if then` e `if then else`);
- **estrutura de repetição** (neste caso, `for loop`);
- início do aprendizado sobre tratamento de exceções em Python.

```
    while True:
        try:
                = input( )
            if ( ):
                print( )
                continue
            break
        except:
            print( )
```

SEMANA 2

Problema. 1.7 *Encontre os fatores de um dado número inteiro positivo x.*

Solução: Um algoritmo é o seguinte:

Algoritmo 1.7: Fatores de um dado número inteiro positivo.

read x
if $x \leq 0$ **then**
  print "x precisa ser um número inteiro maior ou igual a que 1."
else
  for $i = 1$ **to** $\frac{x}{2}$ **do**
    if $\mod(x, i) = 0$ `// i é divisor inteiro de x?`
    then print i
  print x

```
# Fatores de um numero inteiro positivo - Incluindo tratamento de excecoes
# (implementacao em Python)
# Claudio N. Meneses, 08/02/24

while True:
    try:
        print('Problema 1.6. Fatores de um numero inteiro positivo.')
        numero = int(input('Numero a ser verificado: '))
        if (numero <= 0):
            print("\nErro: valor a ser fatorado precisa ser um numero inteiro positivo.\n")
            continue
        break
    except:
        print("\n\nErro: valor a ser digitado precisa ser um numero.\n")

print('Os fatores do numero %d sao: ' % (numero))
for i in range(1,int(numero/2) + 1):
    if numero % i == 0:        # numero eh divisivel por i
        print(i, end= ' ')
print(numero,end='')
```

Problema. 1.8 *Dado um número real x, compute e^x usando os primeiros n termos da série*

$$e^x = \frac{x^0}{0!} + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{1.2}$$

Solução: Abaixo descrevemos dois possíveis algoritmos para resolver o problema. O primeiro usa a ideia de sub-rotina.

Denotamos por $\mathbb{Z}_{\geq 0} = \{0, 1, 2, 3, \ldots\}$ o conjunto dos números inteiros não negativos. Lembre-se que: se $n \in \mathbb{Z}_+$, o fatorial de n (isto é, $n!$) é definido como $0! = 1$ e $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$. Ainda melhor, $n! = n \times (n-1)!$. Usarei este fato para projetar o segundo algoritmo.

Algoritmo 1.8: Valor aproximado de e^x.

Function fat(x)
$a \leftarrow 1$
for $i = 1$ **to** x **do**
  $a \leftarrow ia$
return a
end

Algoritmo principal:
read n, x
$soma \leftarrow 0$
$y \leftarrow 1$
for $i = 0$ **to** n **do**
  $soma \leftarrow soma + y/fat(i)$
  $y \leftarrow yx$
print $soma$

```
# Valor aproxima de e^x - Incluindo tratamento de excecoes
# (implementacao em Python)
# Claudio N. Meneses, 08/02/24

import math

def fat(n):
    return 1 if (n == 1 or n == 0) else n * fat(n - 1)

% def fat(x):
%     a = 1
%     for i in range(1,x+1):
%         a = i * a
%     return a

while True:
    try:
        print('Problema 1.8. Aproximacao de e^x por meio de uma serie.')
        x = float(input('Digite o valor do expoente em e^x: '))
        num_termos = int(input('Digite o numero de termos da serie: '),10)
        if (num_termos < 0):
            print('\nErro: o numero de termos da serie precisa ')
            print('ser um numero inteiro nao negativo.\n')
            continue
        break
    except:
        print('\n\nErro: valor a ser digitado precisa ser um numero.\n')

print('\nValor aproximado de e^%.8f em diferentes quantidades de termos' % (x))
print('Numero de termos da serie      valor da serie')

soma = 0
y = 1
for i in range(num_termos):
    soma = soma + y/fat(i)
    y = y * x
    print('%d    %.30f' % (i+1, soma))

print('Na funcao da bibliotaca em Python: e^%.8f = %.20f ' % (x,math.exp(x)))
print('Na nossa implementacao:            e^%.8f = %.20f ' % (x,soma))
dif = math.exp(x) - soma
```

```
print('Diferenca = %.20f ' % (dif))
print(f"Em notacao cientifica, a diferenca eh {dif:e}")
```

Algoritmo 1.9: Valor aproximado de e^x.

read n, x
if $n < 0$ **then**
  print n "precisa ser um número inteiro maior ou igual a 0."
else
  $soma \leftarrow 0$
  $fat \leftarrow 1$
  $y \leftarrow 1$
  for $i = 0$ **to** n **do**
    $soma \leftarrow soma + y/fat$
    $fat \leftarrow (i+1)fat$
    $y \leftarrow yx$
  print $soma$

```
# Valor aproxima de e^x - Incluindo tratamento de excecoes
# (implementacao em Python)
# Claudio N. Meneses, 08/02/24

import math

while True:
    try:
        print('Problema 1.8. Aproximacao de e^x por meio de uma serie.')
        x = float(input('Digite o valor do expoente em e^x: '))
        num_termos = int(input('Digite o numeto de termos da serie: '),10)
        if (num_termos < 0):
            print('\nErro: o numero de termos da serie precisa ')
            print('ser um numero inteiro nao negativo.\n')
            continue
        break
    except:
        print('\n\nErro: valor a ser digitado precisa ser um numero.\n')

print('\nValor aproximado de e^%.8f em diferentes quantidades de termos' % (x))
print('Numero de termos da serie, valor da serie')
soma = 0
fat = 1
y = 1
for i in range(num_termos):
    soma = soma + y/fat
    fat = (i+1) * fat
    y = y * x
    print('%d   %.30f' % (i+1, soma))

print('\nNa funcao da bibliotaca em Python: e^%.8f = %.20f ' % (x,math.exp(x)))
print('Na nossa implementacao:            e^%.8f = %.20f ' % (x,soma))
dif = math.exp(x) - soma
print('Diferenca = %.20f ' % (dif))
print(f"Em notacao cientifica, a diferenca eh {dif:e}")
```

PERGUNTA: No seu computador, qual foi o máximo valor de termos para o qual foi possível executar o programa associado ao primeiro código-fonte que aparece na solução do Problema 1.8?

Problema. 1.9 *O número inteiro 3025 tem a seguinte propriedade: 30 + 25 = 55 e* $55^2 = 3025$*. Encontre todos os números inteiros positivos com 4 dígitos que têm esta propriedade.*

Solução:

Algoritmo 1.10: Identifica os números inteiros com a dada propriedade.

```
for x = 1000 to 9999 do
    y ← truncate(x/100)
    z ← x − 100y
    if (y + z)^2 = x then
        print x "tem a propriedade"
```

Problema. 1.10 *Seja* $P(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + ax + a_0$*. Projete um algoritmo que leia* n*, os coeficientes* a_i*,* $i = 0, 1, \ldots, n$*, e* x*. Dados estes valores compute e imprima* $P(x)$*.*

Solução: Um algoritmo para resolver o problema é como segue:

Algoritmo 1.11: Valor de um polinômio.

```
read n, x
soma ← 0
for i = 0 to n do
    read a
    soma ← soma + a x^i
print "O valor de P(x) é " soma
```

Problema. 1.11 *Crie um algoritmo que computa e imprime o valor de* S *utilizando os primeiros* n *termos da série:*

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \cdots$$

Solução: Um algoritmo para o problema é como segue:

Algoritmo 1.12: Cálculo de série.

```
read n
S ← 0
x ← 1
y ← 1
while y ≤ n do
    S ← S + x/y
    x ← x + 2
    y ← y + 1
print S
```

```python
# Valor aproxima de e^x - Incluindo tratamento de excecoes
# (implementacao em Python)
# Claudio N. Meneses, 08/02/24

def InputData():
    while True:
        try:
            nt = int(input('Numero de termos na serie: '),10)
            if (nt < 0):
                print('\nErro: o numero de termos da serie precisa ')
                print('ser um numero inteiro positivo.\n')
                continue
            break
        except:
            print('\n\nErro: valor a ser digitado precisa ser um numero.\n')
    return nt

if __name__ == '__main__':
    print('Problema 1.11. Aproximacao do valor de uma serie.')
    num_terms = InputData()
    print(num_terms)

    Sum = 0
    x = 1
    y = 1
    print(x,y)
    for iterations in range(1,num_terms+1):
        Sum = Sum + x / y
        x = x + 2
        y = y + 1
        print('Ao final da iteracao %d:  x= %d  y= %d   S= %.8f' %(iterations, x, y, Sum)
```

```
Problema 1.11. Aproximacao do valor de uma serie.
Numero de termos na serie: 5
5
1 1
Ao final da iteracao 1:  x= 3  y= 2   S= 1.00000000
Ao final da iteracao 2:  x= 5  y= 3   S= 2.50000000
Ao final da iteracao 3:  x= 7  y= 4   S= 4.16666667
Ao final da iteracao 4:  x= 9  y= 5   S= 5.91666667
Ao final da iteracao 5:  x= 11  y= 6   S= 7.71666667
```

SEMANA 3

Problema. 1.12 *Crie um algoritmo que computa e imprime o valor de S:*

$$S = \frac{2^1}{20} + \frac{2^2}{19} + \frac{2^3}{18} + \cdots + \frac{2^{20}}{1}.$$

Solução: Um algoritmo para resolver o problema é o seguinte:

Algoritmo 1.13: Cálculo de série.

$S \leftarrow 0$
$x \leftarrow 2$
$y \leftarrow 20$
$t \leftarrow 2$
while $y \geq 1$ **do**
  $S \leftarrow S + x/y$
  $x \leftarrow tx$
  $y \leftarrow y - 1$
print S

Problema. 1.13 *Crie um algoritmo que computa e imprime o valor de S:*

$$S = \frac{37 \times 38}{1} + \frac{36 \times 37}{2} + \frac{35 \times 36}{3} + \cdots + \frac{1 \times 2}{37}.$$

Solução: Um algoritmo é o seguinte:

Algoritmo 1.14: Cálculo de S.

$S \leftarrow 0$
$x \leftarrow 37$
$y \leftarrow 1$
while $y \leq 37$ **do**
  $S \leftarrow S + (x(x+1))/y$
  $x \leftarrow x - 1$
  $y \leftarrow y + 1$
print S

Problema. 1.14 *Crie um algoritmo que computa e imprime o valor de S:*

$$S = \frac{1}{1} - \frac{2}{4} + \frac{3}{9} - \frac{4}{16} + \frac{5}{25} - \frac{6}{36} + \cdots - \frac{10}{100}.$$

Solução: Um algoritmo para resolver o problema é o seguinte:

Algoritmo 1.15: Cálculo de S.

$S \leftarrow 0$
$x \leftarrow 1$
$y \leftarrow 1$
$t \leftarrow 1$
while $x \leq 10$ **do**
  $S \leftarrow S + (tx)/y$
  $x \leftarrow x + 1$
  $y \leftarrow y + 1$
  $y \leftarrow y^2$
  $t \leftarrow -1t$
print S

Problema. 1.15 *Compute o valor aproximado de π utilizando os n primeiros termos da seguinte fórmula:*

$$\pi \cong 4\sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1} = 4(1 - \frac{1}{3} + \frac{1}{5} - \cdots)$$

Solução: Um algoritmo para calcular o valor aproximado da série é o seguinte:

Algoritmo 1.16: Cálculo do valor aproximado de π.

read n
$S \leftarrow 0$
$y \leftarrow t \leftarrow 1$
for $i = 1$ **to** n **do**
  $S \leftarrow S + t/y$
  $y \leftarrow y + 2$
  $t \leftarrow -t$
$S \leftarrow 4S$
print S

```
# Valor aproximado de pi (implementacao em Python)
# Claudio N. Meneses, 18/02/24

import numpy as np

def InputData():
    while True:
        try:
            print('Prob 1.15. Aproximacao de pi por meio de uma serie.')
            num_termos = int(input('Numero de termos da serie: '),10)
            if (num_termos < 0):
                print('\nErro: o numero de termos da serie precisa ')
                print('ser um numero inteiro positivo.\n')
                continue
            break
        except:
            print('\n\nErro: valor a ser digitado precisa ser um numero.\n')
    return num_termos

if __name__ == '__main__':
    num_termos = InputData()
    print('\nValor aproximado em diferentes quantidades de termos')
    soma = 0
    y = t = 1
    for i in range(1,num_termos+1):
        soma = soma + t/y
        y = y +2
        t = -t
        print('%d   %.20f' % (i, 4 * soma))
    soma = 4*soma

    print('\nNa versao 3.12.3 do Python, pi eh %0.20f' %(np.pi))
    print('Na nossa implementacao, pi eh %.20f ' % (soma))
    dif = np.pi - soma
    print('A diferenca eh %.20f ' % (dif))
    print(f"Em notacao cientifica, a diferenca eh {dif:e}")
```

```
Prob 1.15. Aproximacao de pi por meio de uma serie.
Numero de termos da serie: 10
Valor aproximado em diferentes quantidades de termos
1   4.0000000000000000000
2   2.6666666666666696273
3   3.4666666666666678509
4   2.8952380952380961028
5   3.3396825396825401021
6   2.9760461760461764946
7   3.2837384837384844260
8   3.0170718170718178186
9   3.2523659347188766943
10   3.0418396189294032439

Na versao 3.12.3 do Python, pi eh 3.1415926535897931160
Na nossa implementacao, pi eh 3.0418396189294032439
A diferenca eh 0.0997530346603898721
Em notacao cientifica, a diferenca eh 9.975303e-02
```

SEMANA 4

Problema. 1.16 *Crie um algoritmo que calcula e imprime o valor de S:*

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \cdots + \frac{99}{50}.$$

Solução:

Algoritmo 1.17: Calcula S.

```
S ← 0
x ← 1
y ← 1
while y ≤ 50 do
    S ← S + x/y
    x ← x + 2
    y ← y + 1
print S
```

Problema. 1.17 *Crie um algoritmo que calcula e imprime o valor de S:*

$$S = \frac{2^1}{20} + \frac{2^2}{19} + \frac{2^3}{18} + \cdots + \frac{2^{20}}{1}.$$

Solução:

Algoritmo 1.18: Calcula S.

```
S ← 0
x ← 2
y ← 20
t ← 2
while y ≥ 1 do
    S ← S + x/y
    x ← 2^t
    t ← t + 1
    y ← y − 1
print S
```

Problema. 1.18 *Crie um algoritmo que calcula e imprime o valor de S:*

$$S = \frac{37 \times 38}{1} + \frac{36 \times 37}{2} + \frac{35 \times 36}{3} + \cdots + \frac{1 \times 2}{37}.$$

Solução:

Algoritmo 1.19: Calcula S.

```
S ← 0;
x ← 37;
y ← 1;
while y ≤ 37 do
    S ← S + (x * (x + 1))/y;
    x ← x − 1;
    y ← y + 1;
print S;
```

Problema. 1.19 *Implemente o seguinte algoritmo que calcula e imprime o valor de S:*

$$S = \frac{1}{1} - \frac{2}{4} + \frac{3}{9} - \frac{4}{16} + \frac{5}{25} - \frac{6}{36} + \cdots - \frac{10}{100}.$$

Solução:

Algoritmo 1.20: Calcula o valor de S.

```
S ← 0
x ← 1
y ← 1
d ← 1
t ← 1
while d ≤ 100 do
    S ← S + (t * x)/d
    x ← x + 1
    d ← (y + 1)^2
    y ← y + 1
    if mod (y, 2) = 0 then
        t ← −1
    else
        t ← 1
print S
```

SEMANA 5

Problema. 1.20 *Um número inteiro x é perfeito se a soma de seus fatores, exceto ele mesmo, é igual a x. Por exemplo, 6 é perfeito visto que $1 + 2 + 3 = 6$. Compute todos os números perfeitos no intervalo $[1, 10^6]$.*

Solução:

Um algoritmo é o seguinte:

Algoritmo 1.21: Números perfeitos.

```
read i, s
for x = i to s do
    soma ← 1
    for j = 2 to x/2 do
        if mod (x, j) = 0 then
            soma ← soma + j
    if x = soma then
        print x " é um número perfeito."
```

Definição 1.4 *Um fator próprio (também chamado divisor próprio) de um inteiro positivo n é um divisor de n que é diferente de 1 e n. Por exemplo, os fatores positivos de 10 são 1, 2, 5 e 10, onde os fatores 2 e 5 são próprios, e 1 e 10 não são próprios.*

Definição 1.5 *Um número inteiro positivo n qualquer é composto se existem dois números inteiros positivos a e b tal que* $n = a \times b$ *e* $1 < a \leq b < n$. *Por exemplo, o número* 15 *é composto, pois* $15 = 3 \times 5$ *e* $1 < 3 \leq 5 < 15$. *Equivalentemente, n é composto se ele tem no mínimo um divisor positivo diferente de 1 e n. Por exemplo, 18 é um número composto, pois ele tem os divisores 2 e 9 além de 1 e 18. O número 17 não é composto, pois os únicos divisores positivos de 17 são 1 e 17.*

Definição 1.6 *Um fator primo de um dado número inteiro positivo é um fator que é primo. Por exemplo, os fatores primos do número 15 são 3 e 5, porque 3 e 5 são divisores de 15 e são números primos.*

Problema. 1.21 *Todo número composto tem um fator próprio menor que ou igual a sua raiz quadrada.*

Prova: A prova é por contradição. Suponha que n é um número composto qualquer. Então, podemos escrever $n = a \times b$, onde a e b são ambos entre 1 e n. Se $a > \sqrt{n}$ e $b > \sqrt{n}$, então $(a)(b) > (\sqrt{n})(\sqrt{n})$ que significa que $a \times b > n$. Isto contradiz a suposição que $a \times b = n$. Portanto, $a \leq \sqrt{n}$ ou $b \leq \sqrt{n}$. Isto é, se n é composto, então n tem um fator primo $p \leq \sqrt{n}$. □

Com esta prova, demostramos que um número inteiro positivo que não tem divisor maior que 1 e menor que a sua raiz é primo. Este resultado nos ajuda a projetar o segundo algoritmo no Problema 1.22.

Problema. 1.22 *Compute os números primos no intervalo* $[1, 10000]$. *Um número inteiro positivo é primo se ele tem somente dois fatores, 1 e ele mesmo.*

Solução: Propomos três **algoritmos básicos** para gerar os números primos no intervalo de números inteiros $[i, s]$, com $2 \leq i \leq s$. A diferença do primeiro para os outros dois algoritmos está no limite superior no *for loop* mais interno, onde trocamos $\frac{x}{2}$ por $\sqrt{x}$.

Algoritmo 1.22: Números primos.

```
read i, s
if i > 1 and s > 1 and i ≤ s then
    for x = i to s do
        soma ← 1
        for t = 2 to x/2 do
            if mod(x, t) = 0 then
                soma ← soma + 1
        if soma = 1 then
            print x "é primo"
```

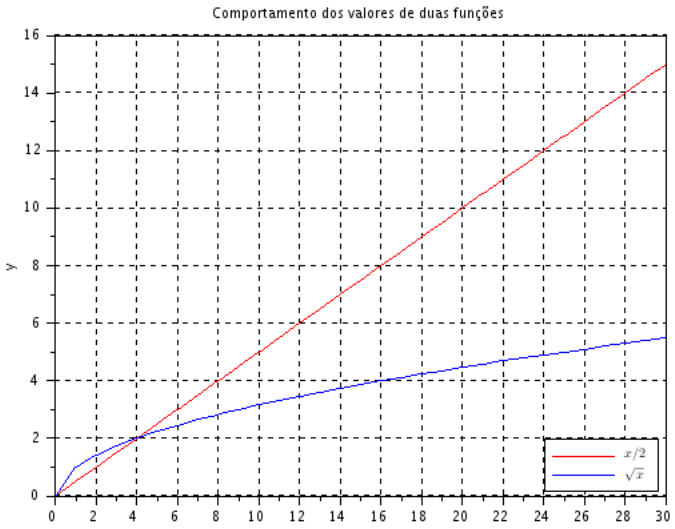

Figura 1.2: Comportamentos das funções $\frac{1}{2}x$ e $\sqrt{x}$.

Algoritmo 1.23: Números primos.

```
read i, s
if i > 1 and s > 1 and i ≤ s then
    for x = i to s do
        soma ← 1
        for t = 2 to √x do
            if mod(x, t) = 0 then
                soma ← soma + 1
        if soma = 1 then
            print x "é primo"
```

O algoritmo anterior pode ser "melhorado" assim:

Algoritmo 1.24: Números primos.

read i, s
if $i > 0$ **and** $s > 0$ **and** $i \leq s$ **then**
  for $x = i$ **to** s **do**
    $flag \leftarrow true$
    $target \leftarrow \lfloor \sqrt{x} \rfloor + 1$
    for $t = 2$ **to** $target$ **do**
      if $mod(x, t) = 0$ **then**
        $flag \leftarrow false$
        break
    if $flag = true$ **then**
      print x "é primo"

```
import math

i, s = input().split(' ')
i = int(i)
s = int(s)
if (i == 1): i = 2
Lista_primos = []
if (i >= 2 and s >= 2 and i <= s):
    k = i
    while (k >= i and k <= s):
        soma =1
        t = 2
        target = int(math.sqrt(k))
        while (t <= target):
            if (k % t == 0): soma +=1
            t += 1
        if (soma == 1): Lista_primos.append(k)
        k += 1
    print('Lista de primos=', Lista_primos)
    print('Numero de primos no intervalo [%d,%d] eh %d' %(i,s,len(Lista_primos)))
```

Problema. 1.23 *Compute a raiz quadrada de um número real positivo usando o método de aproximações sucessivas de Isaac Newton (inglês, 1643-1727). O método funciona da seguinte maneira. Dado um número real positivo x:*

- *A primeira aproximação para a raiz quadrada de x é $x_0 > 0$, onde $x_0 < x$.*
- *As aproximações sucessivas são: $x_{n+1} = \frac{\frac{x}{x_n} + x_n}{2}$ para $n = 0, 1, 2, \ldots$.*

Gere as n primeiras aproximações para um dado x_0.

Solução: Um algoritmo é o seguinte:

Algoritmo 1.25: Valor aproximado da raiz quadrada de x.

```
read x, y, n
if n < 1 or y ≤ 0 or y ≥ x then
    print ("n > 1 ou y > 0  ou  y < x")
else
    x ← y
    for i = 0 to n − 1 do
        y ← (x/y + y)/2
    print y
```

Problema. 1.24 *Crie um algoritmo que computa e imprime o valor de S utilizando os primeiros n termos da série:*

$$S = \frac{1000}{1} - \frac{997}{2} + \frac{994}{3} - \frac{991}{4} + \cdots$$

Solução:

Algoritmo 1.26: Cálculo de série.

```
read n
S ← 0
x ← 1000
y ← 1
t ← 1
while y ≤ n do
    S ← S + (tx)/y
    x ← x − 3
    y ← y + 1
    t ← −1t
print S
```

Problema. 1.25 *Crie um algoritmo que computa e imprime o valor de S utilizando os primeiros n termos da série:*

$$S = \frac{480}{10} - \frac{475}{11} + \frac{470}{12} - \frac{465}{13} + \cdots$$

Solução:

Algoritmo 1.27: Cálculo de série.

```
read n
S ← 0
x ← 480
y ← 10
t ← 1
d ← 1
while d ≤ n do
    S ← S + (tx)/y
    x ← x − 5
    y ← y + 1
    d ← d + 1
    t ← −1t
print S
```

Problema. 1.26 *Crie um algoritmo que computa e imprime o valor aproximado de π utilizando os primeiros n termos da série:*

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Solução:

Algoritmo 1.28: Cálculo de série.

```
read n
S ← 0
x ← 4
y ← 1
d ← 1
t ← 1
while d ≤ n do
    S ← S + (tx)/y
    y ← y + 2
    d ← d + 1
    t ← −1t
print S
```

Problema. 1.27 *Crie um algoritmo que calcula o valor aproximado de $\pi = \sqrt[3]{32S}$ utilizando os primeiros n termos da série abaixo, onde S é*

$$S = \frac{1}{1^3} - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \frac{1}{9^3} - \frac{1}{11^3} + \cdots$$

Solução:

Algoritmo 1.29: Cálculo de série.

```
read n
S ← 0
x ← 1
y ← 1
d ← 1
t ← 1
while d ≤ n do
    S ← S + t/x
    y ← y + 2
    x ← y^3
    t ← −1t
    d ← d + 1
print ∛(32S)
```

Problema. 1.28 *Crie um algoritmo que computa e imprime o valor de S:*

$$S = \frac{x^{25}}{1} - \frac{x^{24}}{2} + \frac{x^{23}}{3} - \frac{x^{22}}{4} + \cdots + \frac{x}{25}.$$

Solução:

Algoritmo 1.30: Cálculo de série.

```
read x
S ← 0
y ← 1
t ← 1
while y ≤ 25 do
    S ← S + x^(26−y)/y
    y ← y + 1
    t ← −1t
print S
```

Problema. 1.29 *Crie um algoritmo que computa e imprime o valor de S:*

$$S = \frac{1}{225} - \frac{2}{196} + \frac{4}{169} - \frac{8}{144} + \cdots + \frac{16384}{1}.$$

Solução:

Algoritmo 1.31: Cálculo de série.

```
S ← 0
x ← 1
y ← 15
t ← 1
d ← y²
while d ≥ 1 do
    S ← S + x/d
    x ← 2x
    y ← y − 1
    d ← y²
    t ← −1
print S
```

SEMANA 8

Problema. 1.30 *Calcule o valor de π usando a integral*

$$\pi = \int_0^1 \frac{1}{1+x^2} dx.$$

Use o método trapezoidal:

$$F = \int_a^b f(x)dx.$$

$$A_i = \frac{(y_i + y_{i+1})h}{2}$$

$$h = x_{i+1} - x_i = \frac{b-a}{n}$$

$$y_i = f(x_i)$$

$$F \cong \sum_{i=1}^{n} A_i$$

Solução: Ao invés de calcular algebricamente a integral acima, podemos determinar o valor aproximado desta integral por meio do cálculo da área sob a curva definida pela função $\frac{1}{1+x^2}$ para $x \in [0, 1]$. A Figura 1.3.

Para calcular a aproximação, usamos o método trapezoidal que está expresso no algoritmo 1.32.

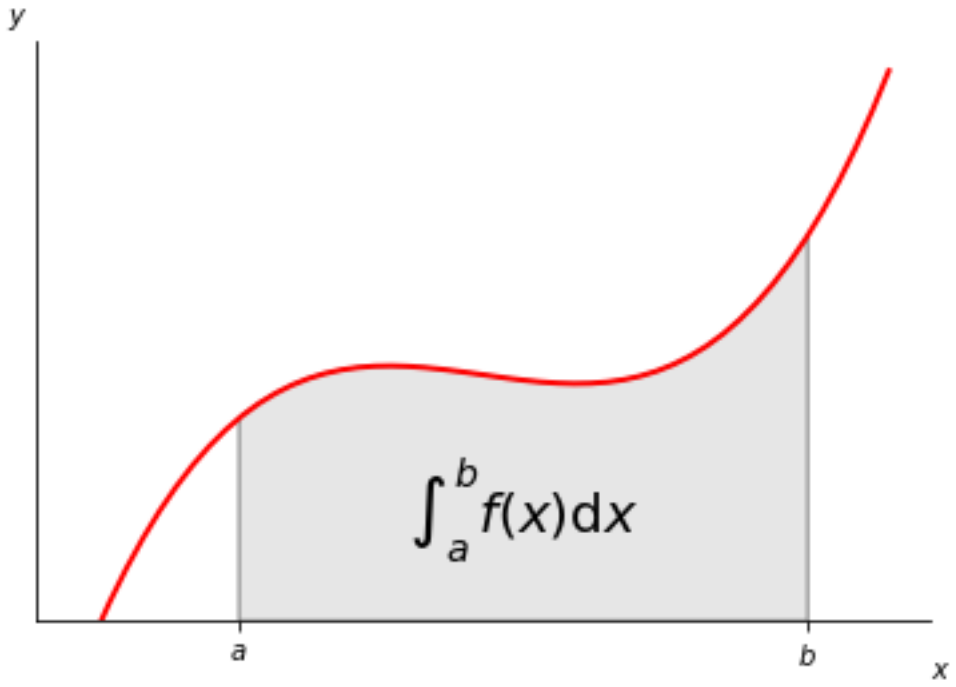

Figura 1.3: Exemplo da equivalência entre os valores $\int_a^b f(x)dx$ e a área sob a curva definida por uma função $f(x)$ no intervalo real $[a, b]$.

Algoritmo 1.32: Cálculo de integral.

read n, a, b
$x \leftarrow a$
$h \leftarrow (b - a)/n$
$soma \leftarrow 0$
while $x \leq b$ **do**
  $y \leftarrow 1/(1 + x^2)$
  $z \leftarrow 1/((1 + (x + h)^2)$
  $soma \leftarrow soma + ((y + z)h)/2$
  $x \leftarrow x + h$
print *soma*

Problema. 1.31 *O algoritmo de Euclides determina o máximo divisor comum entre dois números inteiros* a *e* b*, onde* $a \geq b$*.*

Solução:

Algoritmo 1.33: algoritmo de Euclides para o $mdc(a, b)$.

```
read a, b
while b ≠ 0 do
    t ← b
    b ← mod(a, b)
    a ← t
print a
```

Problema. 1.32 *Dado um número decimal x, encontre sua representação binária. Por exemplo, $8_{10} = 1000_2$.*

Solução: Um algoritmo para resolver o problema é o seguinte:

Algoritmo 1.34: Conversão de decimal para binário.

```
read x
comprimento ← 1 + ⌊log2(x)⌋
for i = 1 to comprimento do
    binaria(i) ← 0
y ← x
i ← comprimento
while y ≥ 2 do
    binaria(i) ← y mod 2
    y ← y div 2
    i ← i − 1
binaria(i) ← y
for i = 1 to comprimento do
    print binaria(i)
```

Definição 1.7 (Notação Big O) *Dadas as funções $f(x)$ e $g(x)$, dizemos que $f(n)$ é $O(g(n))$ se existem constantes $K > 0$ $n_0 > 0$ tal que $f(n) \leq K\ g(n)$ para todo $n \geq n_0$. Por exemplo, se $f(n) = n^2$, então $f(n) = O(n^2)$.*

Uma maneira equivalente de definir a notação Big O é a seguinte:

Definição 1.8 *$O(g(n)) = \{f(n)$: existem constantes positivas c e n_0 tal que $0 \leq f(n) \leq c\ g(n)$ para todo $n \geq n_0$.$\}$*

A notação Big O representa um limite superior do tempo de execução de um algoritmo. Assim, ela fornece a complexidade de tempo de pior caso do algoritmo.

Definição 1.9 (Notação Ω) *Dadas as funções $f(x)$ e $g(x)$, dizemos que $f(n)$ é $\Omega(g(n))$ se existem constantes $c > 0$ $n_0 > 0$ tal que $0 \leq c\ g(n) \leq f(n)$ para todo $n \geq n_0$. Por exemplo, se $f(n) = n^2$, então $f(n) = \Omega(n)$.*

Uma maneira equivalente de definir a notação Ω é a seguinte:

Definição 1.10 $\Omega(g(n)) = \{f(n)$: *existem constantes positivas c e n_0 tal que* $0 \leq c\ g(n) \leq f(n)$ *para todo* $n \geq n_0.\}$

A notação Ω representa um limite inferior do tempo de execução de um algoritmo. Assim, ela fornece a complexidade de tempo de melhor caso do algoritmo.

Problema. 1.33 *Dados n números reais $(a_1, a_2, \ldots, a_n)$ encontre o máximo e o mínimo números. Por exemplo, considere os seguintes dez números 4, 6, -1, 9, 10, 45, 23, 8, 9, 67. O máximo é 67 e o mínimo é -1.*

Solução: Um algoritmo para o problema é como segue:

Algoritmo 1.35: Máximo e mínimo números.

```
read n
min ← +∞
max ← −∞
for i = 1 to n do
    read x
    if x < min then
        min ← x
    if x > max then
        max ← x
print "o número mínimo é " min
print "o número máximo é " max
```

A Tabela 1.1 mostra os números de operações básicas no Algoritmo 1.35. Dentro do *loop for*, o número máximo de atribuições realizadas é n. Assim, $n = k$. No total, o algoritmo realiza no máximo $n + 2 + n + n + 1 + 2n = 5n + 3$ operações básicas.

A complexidade de tempo de um algoritmo é diretamente proporcional ao número de operações básicas realizadas pelo algoritmo. Portanto, a complexidade de tempo do Algoritmo 1.35 é $O(n)$, pois fazendo, por exemplo, $c = 9$ e $n_0 = 1$ concluímos que $0 \leq 5n+3 \leq 9n$.

Problema. 1.34 *Dadas duas matrizes $A_{m\times n}$ e $B_{m\times n}$, compute $C = A + B$.*

Solução:

Operações básicas		
Atribuições	Aritméticas $(+, -, *, /)$	Comparações
$n + 2 + k$	$n + 1$	$2n$

Tabela 1.1: Números de operações no Algoritmo 1.35.

Algoritmo 1.36: Soma de duas matrizes.

```
read m, n
for i = 1 to m do
    for j = 1 to n do
        read A(i, j)
for i = 1 to m do
    for j = 1 to n do
        read B(i, j)
        C(i, j) = A(i, j) + B(i, j)
for i = 1 to m do
    for j = 1 to n do
        print C(i, j)
```

Problema. 1.35 *Dada uma matriz $A_{m\times n}$, encontre sua transposta. Imprima A e A^T. Examplo:*

$$A = \begin{bmatrix} 9 & 16 & 34 \\ 32 & 17 & 12 \end{bmatrix}, A^T = \begin{bmatrix} 9 & 32 \\ 16 & 17 \\ 34 & 12 \end{bmatrix}.$$

Problema. 1.36 *Dadas duas matrizes $A_{m\times n}$ e $B_{n\times k}$, compute $C = AB$, onde $C_{i,j}$ é dado por*

$$C_{i,j} = \sum_{l=1}^{n} A_{i,l} B_{l,j}.$$

Solução:

Algoritmo 1.37: Multiplicação entre duas matrizes.

```
read m, n, k
for i = 1 to m do
    for j = 1 to n do
        read A(i, j)
for i = 1 to n do
    for j = 1 to k do
        read B(i, j)
for i = 1 to n do
    for j = 1 to k do
        S ← 0
        for l = 1 to n do
            S ← S + A(i, l).B(l, j)
        C(i, j) ← S
        print C(i, j)
```

Problema. 1.37 *Dado o seguinte sistema de equações lineares, compute suas n raízes.*

$$\begin{aligned} a_{1,1}x_1 &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 &= b_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 &= b_3 \\ \vdots &= \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + a_{n,3}x_3 + \cdots + a_{n,n}x_n &= b_m \end{aligned}$$

A matriz A contém os coeficientes $a_{i,j}$ e o vetor b contém os termos independentes. Assuma que $a_{ii} \neq 0$ para $i = 1, 2, \ldots, n$.

Solução: Veja que

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{i,j}x_j}{a_{i,j}} \qquad i = 1, 2, \ldots, n$$

Assim, podemos expressar esses cálculos por meio do Algoritmo 1.38.

Algoritmo 1.38: Resolução de um sistema de equações lineares $Ax = b$.

```
read n
for i = 1 to n do
    for j = 1 to n do
        A(i, j) ← 0
i ← 1
while i ≤ n do
    j ← 1
    while j ≤ i do
        read A(i, j)
        j ← j + 1
    i ← i + 1
for i = 1 to n do
    soma ← 0
    for j = 1 to i − 1 do
        soma ← soma + a_{i,j} x_j
    x_i ← (b_i − soma)/a_{i,i}
```

SEMANA 9

Problema. 1.38 *Seja* $P = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + ax + a_0$. *Escreva um algoritmo que leia* n, *os coeficientes* a_i, $i = 0, 1, \ldots, n$, *e* x. *Dados estes valores, calcule e imprima* P.

Solução:

Algoritmo 1.39: Valor de um polinômio.

```
read n, x
sum ← 0
for i = 0 to n do
    read a
    sum ← sum + a x^i
print "O valor de P é" sum
```

Problema. 1.39 *Dados n números, ordene-os em ordem crescente. Por exemplo, suponha que temos os seguintes números 4, -1, 7, 3, 2, 10, 5, 8, 1. A saída do algoritmo deve ser -1,1,2,3,4,5,7,8,10.*

Solução:

Algoritmo 1.40: Ordena n números em ordem crescente.

```
read n
for i = 1 to n do
    read L(i)
for i = 1 to n − 1 do
    min ← L(i)
    index ← i
    for j = i + 1 to n do
        if min > L(j) then
            min ← L(j)
            index ← j
    temp ← L(i)
    L(i) ← L(index)
    L(index) ← temp
for i = 1 to n do
    print L(i)
```

Problema. 1.40 *O problema de string-matching consiste em encontrar todas as ocorrências de um padrão em um texto. Assuma que o texto é um vetor $T[1...n]$ de comprimento n e que o padrão é um vetor $P[1...m]$ de comprimento m. Assuma que elementos de P e T são caracteres retirados do alfabeto $\{a, b, c, \ldots, z, A, B, \ldots, Z\}$. O seguinte algoritmo resolve o problema de string-matching.*

Solução:

Algoritmo 1.41: Algoritmo para o problema de string matching.

```
read T, P
n ← comprimento(T)
m ← comprimento(P)
s ← 0
while s ≤ n − m do
    j ← m
    while j > 0 and P(j) = T(s + j) do
        j ← j − 1
    if j = 0 then
        print "Padrão ocorre na posição" s
    s ← s + 1
```

Problema. 1.41 *Encontre os números inteiros $x, y \in [1, 10^6]$ que satisfazem a equação $xy = 5x + 11y$.*

Problema. 1.42 *Encontre os números inteiros que satisfazem a equação $(x + 1)^2 - x^3 = 1$, onde $x \in [2, 10^6]$.*

Problema. 1.43 *Encontre todos os números inteiros* $n \in [1, 10]$ *tal que* 7^n+4^n+1 *é divisível por 6.*

Problema. 1.44 *Encontre valores inteiros* x, y, z *tal que* $x^n + y^n = z^n$ *para* $x, y, z \in [1, 10^6]$ *e* $n = 2$ *e depois* $n = 3$*.*

Problema. 1.45 *Calcule a soma dos n primeiros termos da série*

$$1, 2, 4, 7, 11, 16, \ldots$$

Problema. 1.46 (Sequência da Lucas) *Gere os n primeiros termos da sequência definida por* $L_n = L_{n-1} + L_{n-2}$ *para* $n \geq 2$*, com* $L_0 = 2$ *e* $L_1 = 1$*. Como exemplo, os 10 primeiros números desta sequência, são:*

$$2, 1, 3, 4, 7, 11, 18, 29, 47, 76.$$

Problema. 1.47 (Sequência de Tribonacci) *Gere os n primeiros termos da sequência definida por* $T_n = T_{n-1}+T_{n-2}+T_{n-3}$ *para* $n \geq 3$*, com* $T_0 = 0$ *e* $T_1 = T_2 = 1$*. Como exemplo, os 10 primeiros números desta sequência, são:*

$$0, 1, 1, 2, 4, 7, 13, 24, 44, 81.$$

Problema. 1.48 (Sequência de Padovan) *Gere os n primeiros termos da sequência definida por* $P_n = P_{n-2} + P_{n-3}$ *para* $n \geq 3$*, com* $P_0 = P_1 = P_2 = 1$*. Como exemplo, os 10 primeiros números desta sequência, são:*

$$1, 1, 1, 2, 2, 3, 4, 5, 7, 9.$$

Problema. 1.49 (Sequência de Fermat) *Gere os n primeiros termos da sequência definida por* $F_n = 2^{2^n} + 1$ *para* $n \geq \mathbb{Z}_{\geq 0}$*. Como exemplo, os oito primeiros números desta sequência, são:*

$$3, 5, 17, 257, 65537, 4294967297, 18446744073709551617, 340282366920938463463374607431768211457.$$

Problema. 1.50 *Os números da forma* $F_n = 2^{2^n} + 1$ *para* $n \geq \mathbb{Z}_{\geq 0}$*, definidos no problema 1.49, são chamados de* `números de Fermat`*. Verifique quais destes números são primos para* $n \in [0, 20]$*.*

A definição de **triângulo de Tartaglia** [Nicolo Tartaglia, italiano, 1499/1500 - 1557] é bastante conhecida. Este triângulo é também conhecido como **triângulo de Pascal** [Blaise Pascal, francês, 1623-1662]). Para um dado número inteiro n, as primeiras quatro linhas deste triângulo pode ser representada como

$$
\begin{array}{lccccccccc}
n=0 & & & & & \binom{n}{0} & & & & \\
n=1 & & & & \binom{n}{0} & & \binom{n}{1} & & & \\
n=2 & & & \binom{n}{0} & & \binom{n}{1} & & \binom{n}{2} & & \\
n=3 & & \binom{n}{0} & & \binom{n}{1} & & \binom{n}{2} & & \binom{3}{3} & \\
n=4 & \binom{n}{0} & & \binom{n}{1} & & \binom{n}{2} & & \binom{n}{3} & & \binom{n}{4}
\end{array}
$$

onde $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$, com $n, k \in \mathbb{Z}_{\geq 0}$ e $n \geq k$. Se $n = 4$, podemos representar as primeiras quatro linhas deste triângulo por

$$
\begin{array}{ccccccccc}
 & & & \binom{0}{0} & & & & & \\
 & & \binom{1}{0} & & \binom{1}{1} & & & & \\
 & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & & \\
\binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & & \\
\binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4}
\end{array}
$$

Fazendo todos os cálculos necessários, o triângulo acima torna-se

$$
\begin{array}{ccccccccc}
 & & & 1 & & & & & \\
 & & 1 & & 1 & & & & \\
 & 1 & & 2 & & 1 & & & \\
1 & & 3 & & 3 & & 1 & & \\
 & 4 & & 6 & & 4 & & 1 &
\end{array}
$$

Os coeficientes no triângulo de Tartaglia/Pascal podem nos ajudar a calcular os coeficientes das expressões:

$$
\begin{aligned}
(a)^0 &= 1 \\
(a+b)^1 &= 1a + 1b \\
(a+b)^2 &= (a+b)(a+b) = 1a^2 + 2ab + 1b^2 \\
(a+b)^3 &= (1a^2 + 2ab + 1b^2)(a+b) = 1a^3 + 3a^2b + 3ab^2 + 1b^2 \\
(a+b)^4 &= (1a^3 + 3a^2b + 3ab^2 + 1b^2)(a+b) \\
&= 1a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + 1b^4
\end{aligned}
$$

onde $a, b \in \mathbb{R}$ (conjunto dos números reais).

Problema. 1.51 (Sequência de Gould) *Gere os primeiros n termos da sequência obtida a partir da contagem do número de números ímpares nas n primeiras linhas do triângulo de Pascal. De maneira formal, seja g_i o número de números ímpares na linha i do triângulo de Pascal. Gere os valores de g_i para $i = 1, 2, \ldots, n$. Como exemplo, os primeiros 15 termos da sequência são*

$$1, 2, 2, 4, 2, 4, 4, 8, 2, 4, 4, 8, 4, 8, 8.$$

Número de Mersenne é um número da forma $M_n = 2^n - 1$, onde $n \in \mathbb{Z}_{\geq 0}$. **Primo de Mersenne** é um número de Mersenne que também é um número primo. Nem todo número de Mersenne é primo. Como exemplo, os primeiros oito números de Mersenne são:

$$M_0 = 0, M_1 = 1, M_2 = 3, M_3 = 7, M_4 = 15, M_5 = 31, M_6 = 63, M_7 = 127.$$

Problema. 1.52 *Gere os primeiros n expoentes primos de primos de Mersenne. Ou seja, os n primeiros valores de p que são primos tal que $2^p - 1$ é primo. Como exemplo, os primeiros 10 termos desta sequências são*

$$2, 3, 5, 7, 13, 17, 19, 31, 61, 89.$$

Problema. 1.53 *Gere os n primeiros primos de Mersenne. Ou seja, os n primeiros números primos da forma $2^p - 1$, onde p é primo. Como exemplo, os primeiros 10 termos desta sequências são*

$$3, 7, 31, 127, 8191, 131071, 524287, 2147483647, 2305843009213693951, 618970019642690137449562111.$$

Problema. 1.54 *Compute uma aproximação para o valor de $cos(\theta)$, θ em graus, utilizando os primeiros n termos da série*

$$cos(\theta) = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \ldots$$

usando os primeiros n termos desta soma, onde n é um número inteiro positivo.

Problema. 1.55 *Dado um número binário x, encontre sua representação decimal. Por exemplo, $1011_2 = 11_{10}$.*

Problema. 1.56 *Sejam P e A dois conjuntos não vazios de números reais, onde: $A \subset P$, $n = |P|$ e $m = |A|$. Assuma que $m > 1$. Segue que $n > m > 1$. Considere que os elementos em P são armazenados no vetor x. Compute os desvios padrão de A (população) e de A (amostra). Use as fórmulas:*

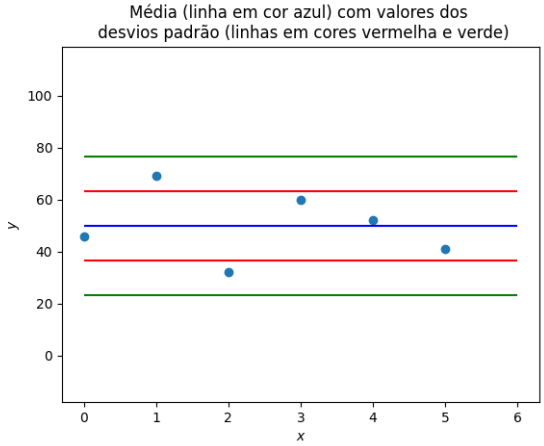

Figura 1.4: Figura com os dados no exemplo no problema 1.56.

$$\mu_P = \frac{\sum_{i=1}^{n} x_i}{n} \qquad [\textit{média da população}]$$

$$DP_P = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}} \qquad [\textit{desvio padrão da população}]$$

$$\mu_A = \frac{\sum_{i=1}^{m} x_i}{m} \qquad [\textit{média da amostra}]$$

$$DP_A = \sqrt{\frac{\sum_{i=1}^{m}(x_i - \mu)^2}{m-1}} \qquad [\textit{desvio padrão da amostra}]$$

Exemplo: *Suponha que* $A = \{46, 69, 32, 60, 52, 41\}$. *Assim,*

$$\mu_A = \frac{46 + 69 + 32 + 60 + 52 + 41}{6} = 50$$

$$\begin{aligned} DP_A &= \sqrt{\frac{(46-50)^2 + (69-50)^2 + (32-50)^2 + (60-50)^2 + (52-50)^2 + (41-50)^2}{5}} \\ &= \sqrt{\frac{886}{5}} \\ &\cong 13.31 \end{aligned}$$

É costumeiramente útil determinar em quais faixas os valores na amostra A estão. Podemos fazer isto, calculando $\mu_A \pm k * DP_A$ *para alguns valores de* k. *Na Figura 1.4, fazendo* $k = 1$ *e* $k = 2$ *definimos as linhas em cores vermelha e verde, respectivamente.*

Problema. 1.57 *Dados dois números inteiros* x *e* y, *encontre o* máximo divisor comum *(mdc) entre* x *e* y. *(O máximo divisor comum entre dois números inteiros é o maior inteiro que divide ambos os números.) Por exemplo,* $mdc(3, 8) = 1$ *e* $mdc(4, 10) = 2$.

Problema. 1.58 *Dois números inteiros* x *e* y *são relativamente primos (ou coprimos) se* $mdc(x, y) = 1$. *Isto é,* x *e* y *são relativamente primos se eles não compartilham divisores*

exceto 1. Por exemplo, 2 e 3 são coprimos pois $mdc(2,3) = 1$*; 3 e 4 são coprimos porque* $mdc(3,4) = 1$*; 2 e 4 não são coprimos pois* $mdc(2,4) = 2$.

Problema. 1.59 *Dados dois pontos* $x = (x_1, x_2)$ *e* $y = (y_1, y_2)$*, onde* $x_i, y_i \in \mathbb{R}$*, compute a distância entre* x *e* y*. A distância entre* x *e* y *é dada por* $d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.

Problema. 1.60 *Encontre todas as raízes inteiras do polinômio* $f(x) = 2x^4+2x^3-7x^2+x-7$ *para* $x \in [0, 1000]$*. (Dica: Encontre valores para* x *tal que* $f(x) = 0$*.)*

Problema. 1.61 *Dada uma matriz quadrada* $A_{n\times n}$*, compute o seu traço. (Dica:* $tr(A) = \sum_{i=1}^{n} A(i,i)$*.)*

Problema. 1.62 *Compute* $y = f(x) + g(x)$ *para* $x = 1, 2, \ldots, 10$ *onde*

$$h(x) = x^2 - 16$$

$$f(x) = \begin{cases} h(x) & se\ h(x) \geq 0 \\ 1 & se\ h(x) < 0 \end{cases}$$

$$g(x) = \begin{cases} x^2 + 16 & se\ f(x) = 0 \\ 0 & se\ f(x) > 0 \end{cases}$$

Problema. 1.63 *Dados os valores* a, b, c *dos comprimentos dos três lados de um triângulo, compute a sua área usando a fórmula de Herão, que é dada por Área*$= \sqrt{p(p-a)(p-b)(p-c)}$*, onde* $p = \frac{a+b+c}{2}$.

Problema. 1.64 *Armazene os números inteiros* $\pm 1, \pm 2, \ldots, \pm 2004$ *em um vetor* A*. Escolha aleatoriamente dois números* x *e* y *e remova-os de* A*, armazene o seu produto,* xy*, em* A*. Continue este processo até restar somente um número. Qual é o sinal do último número em* A*? Isto é, ele é positivo ou negativo? Implemente um algoritmo para resolver este problema, execute-o 100 vezes e conte o número de vezes cujo sinal do último número é positivo.*

Problema. 1.65 *Compute a raiz cúbica de um número positivo* α *por meio do método de Newton aplicando iterativamente*

$$x_n = \frac{1}{3}(\frac{\alpha}{x_{n-1}^2} + 2x_{n-1})$$

para algum valor real inicial x_0*. Encontre* $\sqrt[3]{\alpha}$ *para (a)* $\alpha = 27$ *e* $x_0 = 2$*; (b)* $\alpha = 127$ *e* $x_0 = 5$.

Problema. 1.66 *Um número inteiro com* n *dígitos que é igual a soma dos seus dígitos elevados a* n *é chamado* n*-narcisista. Por exemplo,* $153 = 1^3 + 5^3 + 3^3$ *é 3-narcisista. Encontre todos os números narcisistas no intervalo de números inteiros* $[1, 10^7]$.

Problema. 1.67 *Encontre todo* p_n*,* n*-ésimo número primo, tal que* $A_n = \sqrt{p_{n+1}} - \sqrt{p_n} \geq 1$ *para* $p_n \in [2, 10^7]$.

Problema. 1.68 *A função de contagem de primos $c(x)$ fornece o número de números primos menores ou iguais a um dado número inteiro x. Por exemplo, como não existem números primos menores ou iguais a um, então $c(1) = 0$. Como existe um único número primo menor ou igual a 2, então $c(2) = 1$. Existem dois números primos menores ou iguais a 3, assim $c(3) = 2$. Compute $c(x)$ para todo $x \in [1, 10^6]$.*

Problema. 1.69 *Conte o número de vezes a inequação $c(p_{n+1}^2) - c(p_n^2) < 4$ é satisfeita, para $n \in [2, 10^6]$, onde $c(n)$ é a função de contagem de primos e p_n é o n-ésimo primo.*

Problema. 1.70 *O que $\sum_{i=1}^{\infty} \frac{1}{4^i}$ e $3\sum_{i=1}^{\infty} \frac{1}{10^i}$ têm em comum? (Dica: implemente um algoritmo para computar estas somas.)*

Problema. 1.71 *Compute um valor aproximado para a seguinte soma:*

$$\sum_{n=0}^{\infty} (-1)^n \frac{(3n)!}{[n!(3n)!]^3} x^{2n}.$$

Problema. 1.72 *Compute um valor aproximado para a seguinte soma:*

$$\sum_{k=1,3,5,\ldots} \frac{e^{-kx} sin(ky)}{k}.$$

Problema. 1.73 *Dado o intervalo aberto $(a, b) \subset (0, \pi)$, compute*

$$S = \int_a^b sin(x)\, dx \cong \frac{h}{2}(y_0 + 2y_1 + 2y_2 + \cdots + 2y_{n-1} + y_n),$$

onde h é constante e definida como $h = x_{i+1} - x_i = \frac{b-a}{n}$, n é o número de subintervalos desejado em (a, b), $y_i = sin(x_i)$ e $x_{i+1} = x_i + h$ para $i = 0, 1, \ldots, n-1$.

Dica: digite `integral_0^1 sin(x)` *no wolframalpha para $(a, b) = (0, 1)$.*

Problema. 1.74 *Crie um algoritmo que calcula o valor de S, sendo*

$$S = \sum_{k=1}^{\infty} \frac{1}{(k-1)!}$$

Para qual valor a soma S parece convergir?

A

Códigos-fontes dos gráficos

Os gráficos que aparecem no capítulo anterior foram criados por códigos no Scilab ou utilizando bibliotecas do Python. Os códigos estão listados abaixo. Para o Scilab, O leitor pode encontrar vários tutoriais do Scilab na Web.

Código na linguagem do Scilab para a Figura 1.1:

```
function f = myfunc(x)
   f= x.^3 -3*x.^2 +1;
endfunction

clf
x = [-1:0.1:3];
y = myfunc(x)
xlabel("$x$")
ylabel("$f(x)=x^3 -3x^2 +1$")
xgrid
plot(x,y,'r.->','LineWidth',3)
```

Código na linguagem do Scilab para a Figura 1.2:

```
function f = myfunc_1 ( x )
  f = x./2
endfunction

function f = myfunc_2 ( x )
  f = sqrt(x)
endfunction

xdata = [1:1:30];
ydata = myfunc_1 ( xdata );
```

```
plot ( xdata , ydata, "r" )
ydata2 = myfunc_2 ( xdata );
plot ( xdata , ydata2 , "b" )
xtitle ("Comportamento dos valores de duas funcoes","x","y");
legend ( "$x/2$" , "$\sqrt{x}$",4 );
```

Código em Python para a Figura A.1 do Problema 1.22.

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

#plt.style.use('_mpl-gallery')

# define os dados
x = np.linspace(0, 100,200)     # 200 valores espacados em [0,100]
y1 = x/2
y2 = np.sqrt(x)

fig, ax = plt.subplots()

ax.plot(x,y1,'r',label='$x/2$',linewidth=1.0)         # cor r='red'
ax.plot(x,y2,'b',label='$\sqrt{x}$',linewidth=1.0)   # cor b='blue'
ax.set_xlabel('$x$')
ax.legend(loc='best')

plt.show()
```

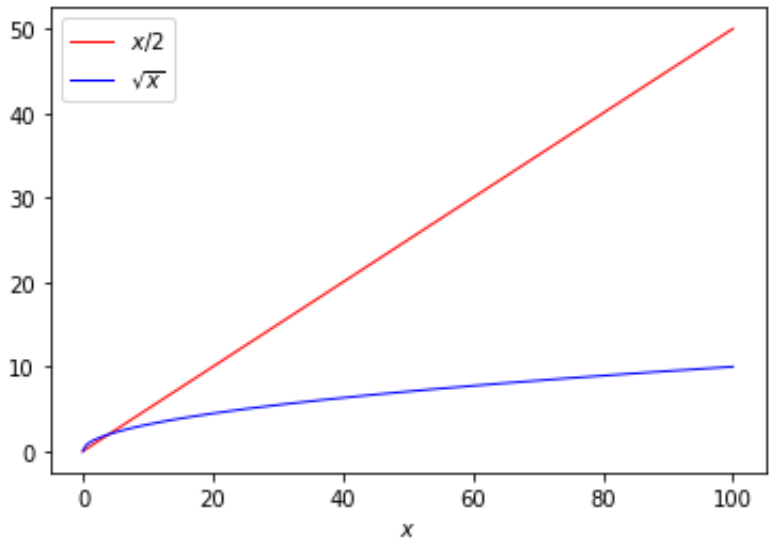

Figura A.1: Comportamentos das funções $x/2$ e $\sqrt{x}$ considerando 200 valores de x igualmente espaçados no intervalo real $[0, 100]$. Esta figura foi desenhada utilizando as bibliotecas `matplotlib` e `numpy` da linguagem Python.

Código em Python para a Figura 1.4 do Problema 1.56.

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

def PlotStd(data):
    print(data)
    mean = np.mean(data)
    std = np.std(data,ddof=1)
    min_value = np.min(data)
    max_value = np.max(data)
    print('media= {}, desvio padrao= {:0.2f}, minimo valor = {},
       maximo valor = {}'.format(mean,std,min_value,max_value))
    x = [k for k in range(len(data))]
    print(x, ' ',data)
    plt.title('Media (linha em cor azul) com valores dos \n desvios
       padrao (linhas em cores vermelha e verde)')
    plt.xlabel('$x$')
    plt.ylabel('$y$')
    plt.ylim(min_value - 50, max_value + 50)
    plt.scatter(x, y=data)
    plt.hlines(y=mean, xmin=0, xmax=len(data),colors='b')

    plt.hlines(y=mean - std, xmin=0, xmax=len(data), colors='r')
    plt.hlines(y=mean + std, xmin=0, xmax=len(data), colors='r')
    plt.hlines(y=mean - 2*std, xmin=0, xmax=len(data), colors='g')
    plt.hlines(y=mean + 2*std, xmin=0, xmax=len(data), colors='g')
    plt.show()

def main():
    data = [46,69,32,60,52,41]
    PlotStd(data)

if __name__ == '__main__':
    main()
```