Capítulo 11

Comandos de repetição

Neste capítulo vamos ver como executar comandos de forma repetida e controlada. No programa da bisseção, com dez iterações, tivemos que repetir as mesmas instruções, dez vezes. Para cada iteração, precisamos dividir o intervalo corrente, computar o erro e escolher o próximo intervalo. Nada muda, os comandos executados são sempre os mesmos.

Programa 11.1 Programa de bisseção, sem comandos de repetição

```
1  #iteração 1
2  c = (a+b)/2
3  if abs(( b - a ) / 2) < erro:
4      print('Achou raiz ', c, ' com erro ', (b-a)/2)
5      sys.exit();
6  if f(a) * f(c) < 0:
7      b = c
8  else:
9      a = c
10 ...
11 #iteração 10
12 c = (a+b)/2
13 if abs(( b - a ) / 2) < erro:
14     print('Achou raiz ', c, ' com erro ', (b-a)/2)
15     sys.exit();
16 if f(a) * f(c) < 0:
17     b = c
18 else:
19     a = c
```

Antes de mais nada, é extremamente deselegante escrevermos um programa desta maneira. Além disso, o programa é muito inflexível, ou seja, ele tenta a solução em dez iterações e pronto. Se quisermos mudar isso, temos que editar nosso programa e adicionar ou remover várias linhas de código. Por isso, nas próximas seções, vamos conhecer alguns comandos que permitem executar uma sequência de instruções repetidamente.

11.1 O comando `while`

No comando `while`, assim como no comando `if`, existe uma expressão booleana que é calculada e, se o resultado for `True`, então são executados os comandos que estão "dentro" do `while`. Se a expressão for falsa, então os comandos dentro do `while` são simplesmente ignorados e executa-se o próximo comando. A diferença, em relação ao `if` é que, terminada a execução dos comandos dentro do `while`, a execução volta para o início do comando, ou seja, a expressão booleana é avaliada novamente, e tudo se repete.

Por exemplo, no trecho de programa abaixo, a variável `i`, inicialmente recebe o valor 1. Em seguida, inicia-se a execução do `while`, calculando a expressão `i < 10`, cujo resultado é `True`. Por isso, são executados os dois comandos dentro do `while`. Note que o segundo comando faz com que `i` passe a valer 2. Depois disso, como não temos mais comandos dentro do `while`, a execução volta para o seu início, calculando a expressão booleana que, novamente é verdadeira e, novamente os comandos dentro do `while` são executados.

Programa 11.2 Exemplo do comando `while`

```
1 i = 1
2
3 while i < 10:
4     print('O valor de i é: ', i)
5     i = i + 1
6
7 print('O valor final de i é: ', i);
```

A cada vez que a execução do programa entra no comando `while`, uma mensagem é exibida e o valor da variável `i` aumenta. A execução só passa para o próximo comando, que segue o `while`, quando a expressão booleana for false, ou seja, quando `i` atingir o valor 10. A saída gerada por esse programa é a seguinte:

```
O valor de i é: 1
O valor de i é: 2
O valor de i é: 3
O valor de i é: 4
O valor de i é: 5
O valor de i é: 6
O valor de i é: 7
O valor de i é: 8
O valor de i é: 9
O valor final de i é: 10
```

Podemos visualizar a execução desse trecho de programa por meio do diagrama da Figura 11.1.

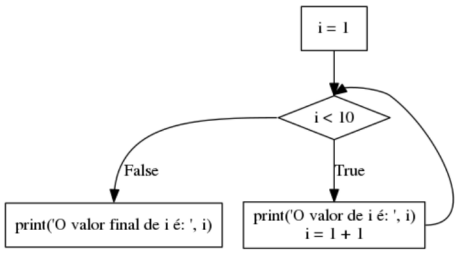

Figura 11.1: Fluxograma do comando `while`

Vamos, então, usar o comando `while` para implementar o algoritmo da bisseção. Mais precisamente, usamos esse comando para controlar o número de iterações do algoritmo. Se ele atingir um valor máximo, por exemplo 10, terminamos a execução e mostramos o resultado até aquele ponto. Inicialmente, fazemos as inicializações das variáveis que vamos utilizar no programa, ou seja, qual é a função, o intervalo, a tolerância, e o número máximo de iterações.

Programa 11.3 Inicialização de variáveis para método da bisseção

```
1 import sys
2
3 f = lambda x: x ** 3 - x ** 2 - 13 * x + 8
4 a = -4
5 b = -3
6 erro = 0.001
7 iteracoes = 10
```

Em seguida, precisamos calcular o valor médio entre `a` e `b` e verificar se o tamanho do intervalo é menor do que o erro. Se for, achamos já a solução e podemos terminar a execução. Se não for, calculamos um novo intervalo. Mas isso precisamos fazer várias vezes. No nosso caso, no máximo dez vezes, pois usamos `iteracoes = 10`. Temos, então, os seguintes comandos, depois da inicialização:

Programa 11.4 Implementação da bisseção usando o comando `while`

```
1  i = 1
2  while i <= iteracoes:
3      c = (a+b)/2
4      if abs(( b - a ) / 2) < erro:
5          print('Achou raiz ', c, ' com erro ', (b-a)/2)
6          sys.exit()
7      if f(a) * f(c) < 0:
8          b = c
9      else:
10         a = c
11     i = i + 1
12
13 print('Valor calculado ', c, ' com erro ', (b-a)/2)
```

A variável `i` é usada para controlar o número de iterações que foram executadas. Seu valor inicial é um pois quando o comando `while` é executado, o valor de `i` deve indicar qual é a iteração que está sendo iniciada. O comando `while` e os comandos que estão dentro dele serão executados enquanto o valor de `i` for menor ou igual ao número máximo de iterações, no caso, 10.

Note que caso o tamanho do intervalo seja menor do que o erro que determinamos no início, o programa termina de imediato, sem que seja preciso executar todas as iterações. Nesse caso, nosso programa mostra o valor calculado da variável `c` e termina a execução. Se o tamanho do intervalo não atingir um valor inferior ao erro nas dez iterações, o comando `while` termina e então o valor computado até aquela iteração é exibido, bem como o erro até aquele momento.

Para mudar o número de iterações desejadas não precisamos mudar substancialmente nosso programa. Basta mudar o comando que inicializa a variável `iteracoes`. Ou podemos fazer melhor do que isso. Podemos perguntar ao usuário quantas iterações ele deseja, qual o erro que ele quer usar e qual é o intervalo inicial.

Programa 11.5 Lendo o valor das variáveis para o método da bisseção

```
1  import sys
2
3  f = lambda x: x ** 3 - x ** 2 - 13 * x + 8
4  a = int(input('Forneça o valor inicial de a: '))
5  b = int(input('Forneça o valor inicial de b: '))
6  erro = float((input('Qual o valor da tolerância? '))
7  iteracoes = int(input('Número máximo de iterações: '))
```

Existem várias maneiras de implementar um mesmo algoritmo. Na listagem que segue, vemos uma forma alternativa de implementar o método da bisseção. A ideia aqui é que o programa termine quando atingimos o número máximo

de iterações ou quando o erro que temos for menor do que a tolerância que determinamos no início.

Mas como estamos utilizando o comando `while`, podemos dizer que o programa deve continuar enquanto não atingimos o número máximo de iterações e o erro for maior que a tolerância. A inicialização das variáveis continua a mesma mostrada anteriormente. O controle das iterações passa a ser:

Programa 11.6 Implementação da bisseção usando o comando `while` com uma condição composta

```
1  i = 1
2  c = (a+b)/2
3
4  while i <= iteracoes and abs(( b - a ) / 2) >= erro :
5      if f(a) * f(c) < 0:
6          b = c
7      else:
8          a = c
9      i = i + 1
10     c = (a+b)/2
11
12 print('Valor calculado ', c, ' com erro ', (b-a)/2)
```

Note que precisamos computar o valor de `c` antes do comando `while` pois é possível que a execução não entre laço nenhuma vez. Mesmo assim, o ponto médio entre `a` e `b` é a solução.

11.1.1 Exercícios

1. Escreva um programa para calcular o valor total de uma compra de supermercado. O valor dos produtos deve ser informado pelo usuário. O programa deve mostrar o valor final após o usuário informar um número negativo (o número negativo não deve fazer parte do cálculo do resultado final).

2. Incremente, usando o comando `while`, o Exercício 2 da Seção 10.4.1 e permita que o usuário informe o número de alunos que deverão ter suas notas calculadas. O resultado e a nota de cada aluno deverá ser exibido imediatamente após o usuário informar a nota. Exemplo: Caso o usuário informe que será calculada a nota de 3 alunos:

   (a) 1ª nota do 1º Aluno: 7. 2ª nota do 1º aluno: 8. O programa deve exibir "1o. Aluno: Nota Final: 7.75 | Aprovado".

   (b) 1ª nota do 2º Aluno: 7,5. 2ª nota do 2º aluno: 5,5. O programa deve exibir "2o. Aluno: Nota Final: 6,5 | Recuperação".

   (c) 1ª nota do 3º Aluno: 2,5. 2ª nota do 3º aluno: 4,5. O sistema deve exibir "3o. Aluno: Nota Final: 4.5 | Reprovado"

3. Incremente o exercício anterior, que até então calculava a média aritmética só de 2 notas e permita que o usuário informe o número de notas que serão utilizadas para calcular a nota final, ou seja, caso o usuário informe 1, deverá ser digitada 1 nota para cada aluno, caso o usuário informe 5, deverão ser digitadas 5 notas para cada aluno. Exemplo: Caso o usuário informe que serão calculadas 3 notas para cada aluno e 5 alunos:

   (a) 1ª nota do 1º Aluno: 7. 2ª nota do 1º aluno: 8. 3ª nota do 1º aluno: 6,85. O programa deve exibir "1o. Aluno: Nota Final: 7.28 | Aprovado".

   (b) 1ª nota do 2º Aluno: 7,5. 2ª nota do 2º aluno: 5,5. 3ª nota do 2º aluno: 7,5. O programa deve exibir "2o. Aluno: Nota Final: 6,83 | Recuperação".

   (c) 1ª nota do 3º Aluno: 2,5. 2ª nota do 3º aluno: 4,06. 3ª nota do 2º aluno: 5,2. O sistema deve exibir "3o. Aluno: Nota Final: 3.92 | Reprovado".

   (d) 1ª nota do 4º Aluno: 10. 2ª nota do 2º aluno: 8,9. 3ª nota do 2º aluno: 9,5. O programa deve exibir "4o. Aluno: Nota Final: 9.46 | Aprovado".

   (e) 1ª nota do 5º Aluno: 5. 2ª nota do 3º aluno: 7,25. 3ª nota do 2º aluno: 6,15. O sistema deve exibir "5o. Aluno: Nota Final: 6.13 | Recuperação".

11.2 O comando `for`

Outro comando de repetição é o `for`. Ele requer uma variável de controle e um objeto composto por diversas "partes". A cada iteração do comando `for`, uma dessas partes é atribuída à variável de controle. Esse comando é muito utilizado com listas, que veremos em um capítulo mais adiante mas também pode ser usado com outros objetos.

Por exemplo, como vimos anteriormente, um string é composto por vários caracteres, que podem ser acessados por um índice. Podemos utilizar o `for` para pegar esses caracteres, um de cada vez.

Programa 11.7 Uso do comando `for` para pegar os caracteres de um string

```
1 s = 'Python'
2 for c in s:
3     print(c)
```

A execução desse trecho de programa faz, inicialmente, que a letra 'P' seja atribuída à variável `c` e o comando `print` executado com esse valor. Retorna-se ao `for` e o segundo caractere é atribuído à variável `c` e o `print` é executado. E assim por diante, até o último caractere do string armazenado na variável `s`. Dizemos que a variável `c` é usada para "percorrer" os elementos do string.

A saída produzida por esse trecho de programa é o seguinte:

```
P
y
t
h
o
n
```

Dentro do comando `for` podemos ter vários comandos, e quaisquer comandos, como, por exemplo, um comando `if`. No programa abaixo, mostramos na saída somente as letras que forem minúsculas.

Programa 11.8 Uso do comando `for` para pegar os caracteres minúsculos de um string

```
1 s = 'Python'
2 for c in s:
3     if c.islower():
4         print(c)
```

A saída produzida por esse trecho de programa é o seguinte:

```
y
t
h
o
n
```

Outro objeto usado com o comando `for` é do tipo `range`. Ele pode ser obtido com a chamada da função `range`. Se passarmos como parâmetro um número inteiro n, ela produz um objeto que ao ser percorrido pelo `for` vai produzir números inteiros de zero até $n - 1$. Por exemplo, os valores 0, 1, 2, 3 e 4 são atribuídos, em sequência à variável `j` no programa a seguir.

Programa 11.9 Uso do comando `for` com a função `range`

```
1 for j in range(5):
2     print(j)
```

A saída produzida por esse trecho de programa é o seguinte:

```
0
1
2
3
4
```

Se usarmos a função `range` com dois parâmetros, por exemplo `range(3,10)`, queremos atribuir à variável de controle do `for` os valores 3, 4, 5, 6, 7, 8 e 9, ou seja, o primeiro parâmetro indica o valor inicial e o último parâmetro indica o valor final.

Programa 11.10 Uso do comando `for` com a função `range` com 2 parâmetros

```
1 for j in range(3, 10):
2     print(j)
```

A saída produzida por esse trecho de programa é o seguinte:

```
3
4
5
6
7
8
9
```

E se usarmos três parâmetros, o último indica qual é o salto entre uma atribuição e outra. Por exemplo, com `range(-5, 10, 3)`, vamos utilizar os valores -5, -2, 1, 4, 7 e 10.

Programa 11.11 Uso do comando `for` com a função `range` com 3 parâmetros

```
1 for j in range(-5, 10, 3):
2     print(j)
```

A saída produzida por esse trecho de programa é o seguinte:

```
-5
-2
1
4
7
```

11.2.1 Exercícios

- Implemente o método da bisseção usando o comando `for`
- Refaça os exercícios da Seção 11.1.1 utilizando o comando `for`.

11.3 Aplicação: integração numérica

Com o que aprendemos neste capítulo podemos criar um programa que implementa os métodos de integração numérica que vimos na primeira parte do livro. Vamos começar com o método dos trapézio. Nesse método, vamos dividir o intervalo em subintervalos e, então, computar a área de cada um deles, como já discutimos anteriormente.

Programa 11.12 Método dos trapézios usando o comando `while`

```
1  import math
2  f = lambda x:  x ** 3  + 8
3
4  n = 40 # intervalos
5  a = 0 #início do intervalo
6  b = 1 # fim do intervalo
7  h = (b - a) / n # tamanho de cada intervalo
8
9  s = 0.0 # variável para somar a integral
10 while a < b :
11     s += (f(a) + f(a+h)) / 2 * h
12     a += h
13
14 print('O valor da integral é {:.7f}'.format(s))
```

Assim como mostramos antes, a inicialização das variáveis `a`, `b` e `n` pode ser feita perguntando-se ao usuário quais valores deseja utilizar. Aqui mantivemos fixos, o intervalo $(0, 1)$ e dividimo-lo em 40 subintervalos. A função que estamos usando é $f(x) = x^3 + 8$.

Uma vez inicializados os parâmetros básicos do método, calculamos o tamanho de cada intervalo, guardando seu valor na variável `h`. A variável `s` guarda

o valor da integral. A cada iteração do método, soma-se a essa variável a área do subintervalo.

O comando `while` vai ser executado enquanto o valor de `a` for menor do que o valor de `b`. A cada execução do `while`, vamos adicionando o tamanho do intervalo à variável `a`. Assim, essa variável tem sempre o valor de início do próximo intervalo do qual vamos calcular a área. No primeiro comando dentro do `while`, calculamos a área do intervalo $(a, a + h)$ e somamos esse valor na variável `s` [1].

Ao final da execução do `while`, a variável `s` é exibida, indicando o valor da integral calculada.

Com o comando `for` nós utilizamos uma variável `i` que "conta" qual é o intervalo que vamos calcular a área. Para saber onde é o início do intervalo, soma-se $i * h$ ao valor inicial do intervalo todo, ou seja, a variável `a`. Esse valor é calculado a cada execução do `for` e atribuído à variável `x0`. Dessa forma não é necessário, ou melhor, seria incorreto ir incrementando o valor da variável `a`, como fizemos anteriormente.

Programa 11.13 Método dos trapézios usando o comando `for`

```
1  import math
2  f = lambda x:  x ** 3  + 8
3
4  n = 40 # intervalos
5  a = 0 #início do intervalo
6  b = 1 # fim do intervalo
7  h = (b - a) / n # tamanho de cada intervalo
8
9  s = 0.0 # variável para somar a integral
10 for i in range(n) :
11     x0 = a + i * h
12     s += (f(x0) + f(x0+h)) / 2 * h
13
14 print('O valor da integral é {:.7f}'.format(s))
```

Embora funcione bem para a função $f(x) = x^3 + 8$, esse nosso programa, em particular o que usa o comando `while` tem um problema. Se o executarmos com a função $f(x) = \sqrt{1 - x^2}$ veremos que um erro ocorre.

Isso acontece por um problema de precisão. Um número `float`, quando representado digitalmente, pode apresentar um erro que, embora muito pequeno, algumas vezes nos atrapalha. Nesse caso, o valor de `a` vai sendo incrementado e calculamos a área do trapézio definido entre esse valor e o valor de `a + h`. No último intervalo, o valor de `a + h` coincide com `b`. Mas, por causa desses erros minúsculos, é possível que o valor de `a + h` ultrapasse o valor de `b`, em uma quantidade muito, muito pequena (algo próximo de $10E - 16$). Isso não seria

[1] Quando usamos a notação "`x += <expressão>`" estamos dizendo que a expressão à direita é calculada e somada ao valor que já existia na variável `x`. É o mesmo que escrever `x = x + <expressão>`. Existem operadores de atribuição como esse também para os demais operadores aritméticos como `-=`, `*=` e `/=`

problema, em geral, pois faria com que a área do último trapézio aumentasse de um valor muito pequeno, e nosso erro seria, também, muito pequeno.

Porém, em alguns casos como o da função mencionada, acontece um erro pois a função não é definida além do intervalo de integração. Ou seja, $f(x)$ é indefinido para $x > 1$ e ao tentarmos computar, no nosso programa, o valor de `f(a + h)` obtemos um erro pois para o interpretador Python não se pode computar a raiz quadrada de um número negativo.

Para resolver esse problema, podemos tentar lidar com o erro de precisão dos `float`s ou podemos alterar um pouco nosso programa, que é o que faremos. Antes de computar a área do trapézio, verificamos se `a + h` ainda está dentro do intervalo. Se não estiver, usamos `b` como limite do intervalo e não `a + h`. Fica assim, então nosso programa:

Programa 11.14 Método dos trapézios usando o comando `while`, corrigido

```
1  import math
2  f = lambda x:  math.sqrt(1 - x**2)
3
4  n = 40 # intervalos
5  a = 0 #início do intervalo
6  b = 1 # fim do intervalo
7  h = (b - a) / n # tamanho de cada intervalo
8
9  s = 0.0 # variável para somar a integral
10 while a < b :
11     if ( a + h ) > b :
12         s += (f(a) + f(b)) / 2 * (b-a)
13     else:
14         s += (f(a) + f(a+h)) / 2 * h
15     a += h
16
17 print('O valor da integral é {:.7f}'.format(s))
```

O segundo método que vimos para computar a integral é pelo método de Simpson. Nesse caso precisamos apenas dividir o intervalo desejado em subintervalos e adicionar o valor da função em cada um dos pontos que limitam os subintervalos, incluindo `a` e `b`. O único detalhe é que cada um desses valores é multiplicado por uma constante. Para `a` e `b` essa constante é um. Para os demais, é dois para os pontos pares (o segundo, quarto etc) e quatro para os ímpares (o primeiro, terceiro etc). Ao final, a soma é multiplicada pelo tamanho do intervalo, dividido por três.

Como precisamos, em cada iteração, saber qual é o intervalo que estamos trabalhando, usaremos o comando `for` para implementar este programa.

Programa 11.15 Método de Simpson usando o comando `for`

```
1  import math
2  f = lambda x: math.sqrt(1 - x**2)
3
4  n = 40 # intervalos
5  a = 0 #início do intervalo
6  b = 1 # fim do intervalo
7  h = (b - a) / n # tamanho de cada intervalo
8
9  s = 0.0 # variável para somar a integral
10 for i  in range(1,n):
11     if i % 2 != 0: # verifica se i é impar
12         c = 4
13     else:
14         c = 2
15     x0 = a + i * h
16     s += c * f(x0)
17
18 s += f(a) + f(b)
19 s *= h / 3
20
21 print('O valor da integral é {:.7f}'.format(s))
```

11.4 Comandos `break` e `continue`

Existem dois comandos que podemos usar "dentro" de um comando de repetição e que ajudam a controlar sua execução. O primeiro deles é o `break`. Se esse comando é executado dentro de um `while` ou `for`, sua execução faz com que seja abortada a repetição e o próximo comando a ser executado é o que vem depois do `while` ou do `for`.

Por exemplo, vamos supor que temos dois números inteiros, armazenados nas variáveis `a` e `b` e `a` $<$ `b`. Queremos achar o menor valor entre esses dois que seja um divisor de `b`. Para isso, vamos incrementando o valor de `a` até acharmos um divisor ou até que seu valor chegue em `b`. Para isso, podemos usar um comando `break`.

Programa 11.16 Exemplo de uso do comando `break`

```
1 a = int(input('Digite o valor de a: '))
2 b = int(input('Digite o valor de b: '))
3 while a < b:
4     if b % a == 0: # verifica se a divide b
5         break
6     a += 1
7 print('O valor do divisor é: ', a)
```

Nesse exemplo, quando um divisor é encontrado, a condição do `if` é verdadeira e o comando `break` é executado. Isso faz a execução do programa "saltar" diretamente para o fim do comando `while`, ou seja, para o `print`. Assim, o laço de repetição é abortado no momento correto.

Podemos reescrever o programa 11.4 usando um `break` quando chegarmos a um erro menor do que a tolerância que estabelecemos. Nesse caso, a execução do `while` é interrompida e o resultado é mostrado no `print` que está no final, depois do `while`.

Programa 11.17 Implementação da bisseção usando os comandos `while` e `break`

```
1  i = 1
2  while i <= iteracoes:
3      c = (a+b)/2
4      if abs(( b - a ) / 2) < erro:
5          break
6      if f(a) * f(c) < 0:
7          b = c
8      else:
9          a = c
10     i = i + 1
11
12 print('Valor calculado ', c, ' com erro ', (b-a)/2)
```

O comando `continue` faz com que a a execução do laço seja interrompida mas não abandonada. A execução volta para o início do comando de repetição, ou seja, a condição vai ser testada novamente e, se for verdadeira, uma nova iteração do comando acontece. Se for falsa, o comando de repetição termina normalmente. Isso significa que em uma execução de um comando de repetição o `continue`, ao contrário do `break`, pode ser executado várias vezes.

Voltando ao exemplo do Programa 11.18, vamos supor que queremos achar o menor divisor mas ele não pode ser múltiplo de 11. Então, cada vez que um múltiplo de 11 aparecer nosso programa vai fazer a execução voltar ao comando de repetição. Note, também, que aqui vamos usar o comando `for` porque ele incrementa automaticamente o valor da variável de controle do laço. Se usássemos o comando `while` teríamos que incrementar essa variável antes de executar o `continue`.

Programa 11.18 Exemplo de uso do comando `continue`

```
1 a = int(input('Digite o valor de a: '))
2 b = int(input('Digite o valor de b: '))
3
4 for k in range(a,b+1):
5     if k % 11 == 0:
6         continue
7     if b % k == 0:
8         break
9 print('O valor do divisor é: ', k)
```

11.4.1 Exercícios

1. Implemente o Programa 11.18 usando o comando `while`.

2. Escreva um programa que gere aleatoriamente um número entre 0 e 100. Depois, o programa deve dar até 10 chances para o usuário adivinhar qual é o numero secreto. A cada palpite, o programa diz ao usuário se seu palpite é maior, menor, ou se ele acertou o valor. Se o usuário acertar o valor, o programa termina. Para gerar um número aleatório use a função *random.randint*.

3. Implemente o método de Simpson utilizando o comando de repetição `while`.

4. Escreva um programa que recebe como entrada um string que representa um número em algarismos romanos e apresenta como saída o valor decimal desse número. Seu programa não precisa verificar se o número fornecido é valido ou não. Apenas assuma que é.

5. Implemente o método de Newton Raphson para o cálculo das raízes de equações.

6. Uma outra forma de computar as raízes de uma função $f(x)$ no intervalo (a, b) é o seguinte:

   a) a partir de a, use uma variável t e vá incrementando essa variável com o valor $0,1$, até que o valor da função mude de sinal;

   b) quando isso acontecer, você terá um intervalo de tamanho $0,1$ no qual a raiz está localizada;

   c) repita a busca nesse intervalo, usando um incremento menor, $0,01$;

   d) repita esse processo, sempre diminuindo o incremento, até que seu intervalo seja menor que a tolerância desejada;

   e) quando isso acontecer, a solução pode ser dada pelo valor inicial do intervalo.

   Implemente esse algoritmo e verifique se ele realmente funciona. Qual é a desvantagem desse método em relação aos outros que estudamos?

7. Implemente a função de seno, que usa a série de Taylor até que o termo calculado seja menor do que 0,0001.

8. Escreva programas que mostrem as seguintes árvores, com qualquer número de linhas (cada programa mostra uma árvore diferente). Ou seja, o usuário escolhe quantas linhas quer. Se escolher 5, os resultados apresentados são os seguintes:

```
a)
     *
    **
   ***
  ****
 *****

b)
    *
   ***
  *****
 *******
*********
```

9. Escreva um programa que leia um número inteiro N. Depois ele deve ler N números inteiro e dizer quantos são ímpares.

10. Escreva um programa que leia um número inteiro e verifique se ele é primo.

11. Escreva um programa que leia uma sequência de números de ponto flutuante, um de cada vez, até que seja digitado o valor zero. Seu programa deve identificar e mostrar qual é o maior e qual é o menor de todos.

12. A série de Fibonacci é definida da seguinte maneira: o primeiro elemento da série é 1, o segundo também. Os demais são a soma dos dois elementos anteriores. Temos então: 1, 1, 2, 3, 5, 8, 13... Escreva um programa que lê um número inteiro N e mostra os N primeiros elementos da série de Fibonacci.

13. Escreva um programa que lê o primeiro valor, o último e a razão de uma PG e que exibe: todos os elementos da PG nesse intervalo e a soma dos elementos nesse intervalo.