Determinando a equação da reta usando Python

Mario Luiz Bernardinelli *

27 de Abril de 2012
v1.0

Resumo

Python é uma linguagem de programação de uso geral fantástica. Seu aprendizado é rápido, fácil e a grande quantidade de bibliotecas (ou módulos) ajuda muito no desenvolvimento de aplicações. Neste breve artigo, apresentarei um exemplo de uso do módulo Fraction para manipulação de números fracionários.

1 Introdução

Python é uma linguagem de script muito utilizada atualmente nas mais variadas áres. Podemos encontrar aplicações em Python em sistemas de automação, adminsitração de sistemas e de redes, hacking, criptografia, aplicações científicas e até mesmo em aplicações espaciais (a NASA que o diga).

Apesar de todo este poder, Python é uma linguagem de fácil aprendizado.

A motivação para a escrita deste artigo simples surgiu enquanto eu corrigia alguns exercícios de matemática de meu filho. Eram exercícios sobre a equação da reta. Os exercícios forneciam as coordenadas de dois pontos distintos de uma reta e, através deles, a equação da reta deveria ser determinada. Pensei então que seria uma oportunidade interessante para mostrar a facilidade em desenvolver um programa que determinasse a equação da reta.

É um exemplo simples, eu sei, mas pensei em utilizar uma biblioteca, que em Python chamamos módulo, chamada Fraction, que nos permite o uso de números fracionários de uma forma bastante simples.

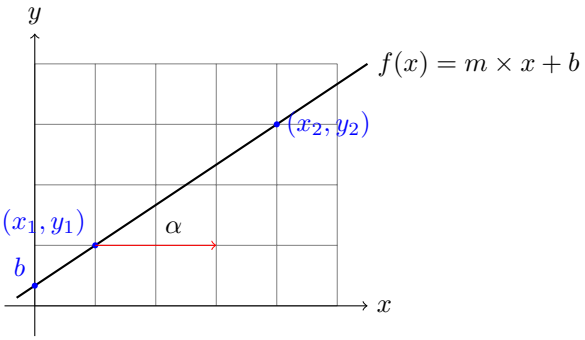
Tendo isto em mente, o objetivo deste artigo não é ensinar Python e muito menos matemática. Para ambos os temas há muitas excelentes referências bibliográficas nas bibliotecas, livrarias e Internet. Na verdade, este artigo nasceu de uma ideia de aprender a usar números fracionários em Python. Simples assim.

2 A reta

Considere o gráfico apresentado na Figura 1.

*Mario Luiz Bernardinelli é Tecnólogo em Processamento de Dados pela Faculdade de Tecnologia de Americana, possui os títulos de Especialista em Segurança da Informação, pela Veris/IBTA; Especialista em Redes de Computadores, pela Unicamp; Especialista em Engenharia de Software, pela Unicamp; e as certificações Linux LPI-C1, LPI-C2, MTCNA, MTCRE e NCLA.

Figura 1: *Gráfico de uma reta*



Conforme (SWOKOWSKI, 2008), a equação da reta é dada por:

$$f(x) = m \times x + b \tag{1}$$

O ponto onde a reta corta o eixo das ordenadas (eixo *y*) corresponde ao *b* da equação e é chamado coeficiente linear.

O coeficiente angular (*m*) da reta, considerando que ela não seja perpendicular ao eixo das abscissas, é o número real que expressa a tangente trigonométrica da sua inclinação em relação ao eixo das abscissas (eixo *x*).

Observe na Figura 1 que foram identificados dois pontos da reta: (*x1, y1*) e (*x2, y2*). Só precisamos de dois pontos para determinar uma reta, o que significa que podemos determinar a equação da mesma.

O processo de determinação da equação da reta é bastante simples. O coeficiente angular é dado pela fórmula:

$$m = \frac{\Delta y}{\Delta x} \tag{2}$$

Substituindo-se os membros da fórmula pelos pontos do gráfico, temos:

$$m = \frac{y2 - y1}{x2 - x1} \tag{3}$$

Uma vez que o coeficiente angular (m) tenha sido determinado, para determinar o coeficiente linear (b), basta utilizarmos um dos pontos para determiná-lo. Veja o exemplo para o ponto $(x1, y1)$:

$$y1 = m \times x1 + b \tag{4}$$
$$b = y1 - (m \times x1) \tag{5}$$

Simples, não é? Agora vamos implementar isso usando Python, num programa bem simples e direto.

3 Determinando a equação da reta usando Python

O código apresentado a seguir é bastante simples, já que o objetivo não é complicar, mas apresentar algumas características da linguagem Python, aliado ao uso de um módulo específico para tratamento de operações com números fracionários e tudo isso aplicado a uma aplicação prática.

O código da nossa aplicação é o seguinte:

```
#!/usr/bin/env python
"""
A equacao da reta e dada por y = mx + b

Onde:

m e o coeficiente angular da reta (determina a sua inclinacao)
b e o coeficiente linear da reta e determina o ponto onde a
  reta corta o eixo y (isto e, b determina o valor de y quando x for zero)
"""

from fractions import Fraction

if __name__ == "__main__":
  print("Determinacao da equacao da reta a partir de 2 pontos:")

  print(__doc__)

  p1 = raw_input("Ponto (x1,y1): ")
  p2 = raw_input("Ponto (x2,y2): ")

  print(p1, p2)

  x1 = p1.split(",")[0].strip()
  y1 = p1.split(",")[1].strip()

  x2 = p2.split(",")[0].strip()
  y2 = p2.split(",")[1].strip()

  print("(x1, y1) = (%s, %s)" % (x1, y1))
  print("(x2, y2) = (%s, %s)" % (x2, y2))

  x1Frac = Fraction(x1)
  y1Frac = Fraction(y1)

  x2Frac = Fraction(x2)
  y2Frac = Fraction(y2)

  mFrac = (y2Frac - y1Frac) / (x2Frac - x1Frac)

  bFrac = y1Frac - (mFrac*x1Frac)

  print("Equacao geral da reta: y = mx+b")

  # Determina o sinal a ser apresentado no texto da equacao da reta
  sign = ""
  if bFrac > 0:
    sign = "+"
  #

  # Monta m e b em formato string levando em consideracao se o
  # denominador for igual a 1 (se denominador=1, nao mostra fracao)
  if mFrac.denominator == 1:
    mStr = "%d" % mFrac.numerator
  else:
    mStr = "(%d/%d)" % (mFrac.numerator, mFrac.denominator)
  #
  if bFrac.denominator == 1:
```

```
59      bStr = "%d" % bFrac.numerator
60    else:
61      bStr = "(%d/%d)" % (bFrac.numerator, bFrac.denominator)
62    #
63    print("\nEquacao da reta proposta: y=%sx%s%s" % (
64          mStr,
65          sign,
66          bStr))
67
68    print("Coeficiente angular: %s" % mStr.strip("(").strip(")"))
69    print("Coeficiente linear:  %s" % bStr.strip("(").strip(")"))
```

Agora vamos comentar o código completamente. Para entender a descrição do código, estou assumindo que o leitor possui algum conhecimento básico de Python.

A primeira linha do programa contém uma diretiva para funcionamento em Linux. Acho que cabe aqui uma pequena introdução ao assunto. Em ambientes Windows, o tipo do arquivo é determinado pela sua extensão. Assim, quando instalamos o Python no Windows, o próprio *software* instalador já adiciona algumas regras ao sistema operacional associando os arquivos com extensão **py** ao interpretador da linguagem Python. O mesmo ocore quando instalamos, por exemplo, o Microsoft Office: os arquivos com extensões **.doc** e **.docx** são associados ao processador de textos (Word).

No Linux, e em outros sistemas operacionais similares, o tipo do arquivo não é determinado pela sua extensão, mas sim pelo seu conteúdo. Assim, quando excutamos um arquivo com conteúdo ASCII em um terminal, por exemplo, quando o interpretador de comandos (*shell*) carrega o arquivo, a primeira coisa que ele verifica é se o arquivo inicia-se com a sequência "#!", que é normalmente é chamada *shebang*. Se esta linha estiver presente, o interpretador de comandos (*shell*) executa o programa especificado nesta linha e passa para ele como parâmetro o arquivo do programa (aquele que contém o *shebang*).

A sequência **#!** também possui outros nomes, tais como, *hashbang*, *pound-bang*, *hash-exclam* ou *hash-pling*.

Em nosso programa, a linha:

```
1  #!/usr/bin/env python
```

Instrui o interpretador de comandos (*shell*) do Linux a executar o comando */usr/bin/env*, passando-lhe como parâmetros a palavra *python*, que está na própria *shebang* e o caminho completo do programa. Por exemplo, vamos supor que nosso programa em Python esteja gravado no arquivo *equacaoreta.py*, quando executamos este programa o sistema executará a seguinte linha:

```
1  python equacaoreta.py
```

É por isso que no Linux podemos remover a extensão do arquivo que ele ainda continuará sendo executado pelo interpretador correto, desde que ele tenha as permissões de execução, mas isto é assunto para um outro artigo.

As linhas 2 a 10 contém apenas comentários. Em Python, os comentários de uma linha podem começar com **#** e, os comentários de múltiplas linhas podem ser iniciados e finalizados com aspas triplas. Aliás, as aspas triplas também servem para fazermos atribuições de *strings* (textos) de múltiplas linhas à variáveis. Exemplo:

```
1  # Este é um comentário de uma única linha
2
3  """
4      Este comentário permite a inserção de várias linhas.
```

```
5       Este tipo de comentário é bastante utilizado, por exemplo,
6       para descrever o que uma função faz.
7   """
```

Na linha 12 começa efetivamente a brincadeira: aqui temos uma das formas de carregarmos módulos em nosso programa. Um módulo é mais ou menos como uma biblioteca. A forma geral de carregarmos um módulo é a seguinte:

```
1   import nome_do_modulo
```

Neste exemplo, estamos carregando todos os objetos, variáveis e funções contidas no módulo. Para acessar qualquer elemento deste módulos, devemos especificar o nome do módulo. Por exemplo, vamos supor que um determinado módulo tenha o nome **integridade** e que, dentro dele tenhamos uma função chamada CRC16. Para chamarmos esta função em nosso programa, deveríamos fazer o seguinte:

```
1   import integridade # Aqui informamos que o módulo integridade deve ser carregado
2
3   integridade.CRC16() # Aqui acessamos a função CRC16() do módulo integridade
```

Observe que o comando **import** é usado uma vez para cada módulo ou seja, só precisamos carregá-lo uma vez em nosso programa. Normalmente, declaramos (ou importamos) os módulos no início do programa.

Uma outra característica bastante interessante, e que foi usada no programa exemplo, é que a linguagem Python permite que sejam carregados apenas partes dos módulos. No nosso exemplo, o comando **from fractions import Fraction**, estamos solicitando ao Python que carregue apenas o objeto Fraction do módulo fractions. Simples e objetivo, como quase tudo em Python.

A linha 14 define o início do programa:

```
1   if __name__ == "__main__":
```

Nesta linha ocorre uma coisa interessante: ela diz ao interpretador Python que, se o módulo atual for o módulo principal do programa, então o bloco de comandos a seguir deve ser executado. Agora vamos entender como isso funciona.

No Python, qualquer programa (ou *script*, como alguns preferem chamar), pode ser tanto um programa como um módulo. Ou seja, podemos escrever códigos em Python que serão utilizados como módulos para outros programas. Quando o interpretador Python é chamado e recebe um arquivo para ser interpretado, aquele arquivo passa a ser o módulo principal do programa, pois ele está no topo da hierarquia de programas a serem executados. Este recurso é interessante porque podemos criar programas de teste e adicioná-lo diretamente no módulo, sem que isto afete o uso do módulo em outros programas.

Como nosso programa é extremamente simples, todo o seu código foi escrito dentro da definição `if __name__ == "__main__"`.

Entendo que o programa é razoavelmente legível, então, vamos nos atentar apenas às linhas interessantes.

A linha 17 imprime o comentário inicial do arquivo. A diretiva __doc__ é conhecida como *docstring* e é usada para documentar módulos, classes, funções e métodos. Para documentar um objeto, deve-se incluir o objeto *docstring* (o texto, devidamente comentado) como primeiro comando após a definição do objeto. Em nosso programa, o primeiro comando em Python efetivamente executado foi a definição da *docstring*. Desta forma, este texto passou a ser a documentação do nosso programa.

As linhas 19 e 20 aguardam a entrada dos pontos da reta. As coordenadas do primeiro ponto são armazenadas na variável *p1* e as coordenadas do segundo ponto são armazenadas em *p2*. Em cada uma destas linhas, o processamento pára e fica aguardando o usuário digitar algum texto.

Observe que o programa, por ter o objetivo de ser simples, assume que o usuário irá digitar as coordenadas corretamente, no formato *x,y*. O tratamento das exceções fica por conta do leitor.

Nas linhas 24 a 28, separamos o texto digitado pelo usuário em valores isolados de cada uma das coordenadas. O programa assume que o usuário entrará as coordenadas dos pontos no formato *x,y*.

A função *split()* é utilizada para criar uma lista de elementos da *string* separadas pelo delimitador especificado no parâmetro da função. Já a função *strip()* elimina espaços em branco do início e final da *string*. Parece confuso? Nem tanto.

Tomemos como exemplo a linha 24:

```
1  x1 = p1.split(",")[0].strip()
```

Nesta linha ocorre o seguinte: a varável *p1* contém uma *string* (texto) digitado pelo usuário como, por exemplo, ”4,5”. Neste caso, queremos obter cara uma das coordenadas isoladamente. A função *split(”,”)* irá retornar uma lista de elementos contidos na variável *p1* e considerará como separador de elementos a vírgula (”,”). Vejamos um exemplo de como isso ocorre usando o interpretador interativamente:

```
1  >>> p1 = "3,5"
2  >>> x = p1.split(",")
3  >>> x
4  ['3', '5']
5  >>>
```

Observe que a função *split()* separou o texto contido em *p1* em elementos distintos, seperados por vírgula, o que resultou numa variável do tipo lista (*x*).

Só que a nossa linha de código do programa faz mais coisas além de separar os elementos da *string* em uma lista: aproveitamos que sabemos que o texto possui dois elementos e atribuímos à variável apenas o valor da coordenada *x* ou o valor da coordenada *y*. Fazemos isso usando o índice do elemento da lista gerada pela função *split()*. Veja o exemplo:

```
1  >>> p1 = "3,5"
2  >>> p1.split(",")[0]
3  '3'
4  >>> p1.split(",")[1]
5  '5'
```

ATENÇÂO Não se esqueça que o tratamento de exceções (erros) na entrada do usuário não é tratada neste código. Isto fica a cargo do leitor.

ATENÇÂO: NUNCA, NUNCA considere que a entrada de dados está correta. Um bom programador SEMPRE irá verificar o dado recebido antes de processá-lo. Todos dados provenientes de meios externos **devem** ser verificados quanto à limites e formato. A não verificação destes dados de entrada costumam ser uma das principais fontes de falhas de segurança em programas e sistemas. Em nosso caso, como o objetivo é manter as coisas mais simples, para facilitar o entendimento, estamos considerando que o usuário irá digitar os dados no formato correto.

Por fim, a função *strip()* remove os caracteres de espaço, se existirem, do início e do final de cada elemento das coordenadas.

Nas linhas 33 a 37 são criados objetos do tipo *Fraction* a partir das coordenadas dos pontos. Observe que, na criação de uma instância de um objeto do tipo Fraction, passamos um texto que representa o número fracionário desejado. Por exemplo, para criamos um objeto Fraction chamado **fr1** que represente a fração 3/7 (três sétimos), usaríamos o seguinte comando:

```
1  fr1 = Fraction("3/7")
```

Na linha 39 calculamos o coeficiente angular (*m*), usando os objetos fracionários. Observe que o uso dos objetos do tipo Fraction nos permite executar operações matemáticas com número fracionários.

Na linha 41 determinamos o coeficiente linear, também usando objetos do tipo Fraction.

Nas linhas 46 a 48, criamos uma variável que irá armazenar o sinal do coeficiente angular. A esta variável (*sign*) atribuímos o texto ”+”se o valor do coeficiente linear for maior que zero.

Nas linhas 51 até 55 criamos uma variável em formato texto (*string*) e armazenamos nela o valor da primeira parte da função da reta, ou seja, o coeficiente angular multiplicado por x. A parte importante deste trecho de código é que, uma vez criado um objeto do tipo Fraction, podemos obter os valores do numerador e denominador através dos atributos *numerator* e *denominator*, respectivamente.

A linha 53 verifica se o denominador é 1 e, se for 1, simplesmente o ignoramos, ou seja, usamos aprenas o denominador, caso contrário, consideramos que o valor é um número fracionário e usamos o numerador e o denominador.

Nas linhas 58 a 61, fazemos o mesmo, só que para o coeficiente linear (*b*).

Na linha 63 imprimimos a função da reta e, nas linhas 68 e 69, imprimimos os coeficientes angular e linear da reta.

Vejamos um exemplo, considerando-se os valores apresentados no gráfico da Figura 1:

```
1  ./equacaoreta.py
2  Determinacao da equacao da reta a partir de 2 pontos:
3
4  A equacao da reta e dada por y = mx + b
5
6  Onde:
7
8  m e o coeficiente angular da reta (determina a sua inclinacao)
9  b e o coeficiente linear da reta e determina o ponto onde a
10   reta corta o eixo y (isto e, b determina o valor de y quando x for zero)
11
12 Ponto (x1,y1): 1,1
13 Ponto (x2,y2): 4,3
14 ('1,1', '4,3')
15 (x1, y1) = (1, 1)
16 (x2, y2) = (4, 3)
17 Equacao geral da reta: y = mx+b
18
19 Equacao da reta proposta: y=(2/3)x+(1/3)
20 Coeficiente angular: 2/3
21 Coeficiente linear:  1/3
```

4 Conclusão

Através de simples exemplo, pudemos ver várias características de um programa escrito em Python, das quais podemos destacar:

- Fácil entendimento. A linguagem Python possui uma sintaxe bastante amigável.
- Altíssimo nível. Por ser uma linguagem de *script*, Python é mais uma linguagem de alto nível, ou seja, que possui objetos complexos, como listas, que facilitam o desenvolvimento rápido de aplicações.

- Orientação a objetos. Apesar do programa apresentado ser simples e não ser orientado a objetos, ele fez uso intensivo de objetos, como *strings* e o próprio objeto Fraction, que permite o uso de números fracionários de uma forma bastante simples.
- Uso de módulos. Usamos apenas um módulo, o *fractions*, e aprendemos que, em Python, podemos carregar e utilizar apenas partes específicas de módulos.
- Módulo principal. Aprendemos, ainda que basicamente, que apenas o programa principal tem o nome "__main__".

Python é uma linguagem de *script* orientada a objetos com uma infinidade de módulos para as mais diversas aplicações. Tenho usado esta linguagem no desenvolvimento de aplicações profissionais em sistemas com Linux embarcado, programas de acesso a bancos de dados, coleta de dados de automação, comunicação em rede e auxílio na administração de servidores e de rede. Python é, sem dúvida nenhuma, uma linguagem poderosa, de fácil aprendizado e que deve crescer muito nos próximos anos.

Referências

SWOKOWSKI, E. W. *Algebra and Trigonometry with Analytic Geometry*. Classic twelfth edition. [S.l.]: Books/Cole, 2008. ISBN 978-0-495-55971-9.