Introdução à computação científica em economia e finanças em Python

Nelson Seixas dos Santos

Faculdade de Ciências Econômicas
Universidade Federal do Rio Grande do Sul

May 20, 2019

Sumário

INTRODUÇÃO

Introdução

Computação científica

Computação científica é a área da ciência da computação que se ocupa da solução de problemas computacionais surgidos na ciência com especial ênfase na implementação de métodos numéricos de solução de problemas matemáticos diversos que vão desde a determinação das raizes de uma equação até o cálculo diferencial e integral vetorial.

Os problemas numéricos mencionados surgem corriqueiramente em economia (em especial, em macroeconomia) e finanças

A biblioteca padrão

A biblioteca padrão de uma linguagem é o conjunto de funções e classes que são instaladas juntamente com o interpretador/compilador que fornecem funcionalidades não previstas na definição da linguagem.
Python conta com uma extensa biblioteca padrão que pode ser conhecida clicando aqui.
Para usar uma função da biblioteca padrão basta usar o comando *import*.

Pacotes matemáticos e estatísticos básicos

Pacotes matemáticos e estatísticos básicos

As funções matemáticas e estatísticas básicas costumeiramente usadas desde o ensino médio não fazem parte da linguagem Python, porém tais funções são disponibilizadas nos pacotes da biblioteca padrão mencionados a seguir:

1. math
2. cmath
3. fractions
4. random
5. statistics

O pacote Math

O pacote math fornece funções matemáticas básicas não disponíveis em Python tais como: log, exp, sin etc.
Para conhecer a lista de funções disponíveis no pacote math, clique aqui.

Função linear

```
def funcao_linear(coeficiente_angular, x):
  y = coeficiente_angular*x
  return y
```

Função afim

```
def funcao_afim(coeficiente_angular, coeficiente_linear, x)
  y = coeficiente_angular*x + coeficiente_linear
  return y
```

Função quadrática

```
def quadratica(c, b, a, x):
  y = c + b*x + a*x**2
  return y
```

Função logarítmica

```
from math import log
x = 0.01
lista = []
while x<=1:
  lista.append(log(x,2))
  x+=.01
print(lista)
```

Exemplo 2: cálculo do exponencial de um número

```
from math import exp

# Entrada de dados
numero = float(input("Digite o número:\n"))

# Cálculo do exponencial
expn = exp(numero)

# Saída de dados
print("O exponencial do número digita é igual a %f."%expn)
```

Exercício: capitalização contínua

Sabendo que, em capitalização contínua, a lei de movimento do capital é dada por $C_t = C_0.exp(r.t)$ onde r é a taxa de juros instantânea, escreva um código que calcule o valor do capital em qualquer instante.

O pacote cmath

Contém as mesmas funções que o pacote math, mas, no pacote cmath, as funções são aplicadas no domínio dos números complexos.

Veja os detalhes, clicando em cmath

O pacote fractions

Contém funções para manipulação de frações. Seu uso pode ser visto no código a seguir:

```
from fractions import Fraction

x = Fraction(3,2)
y = 1.5
x==y
```

Veja os detalhes, clicando em fractions

O pacote random

Contém funções para geração de números aleatórios. Seu uso pode ser visto no código a seguir:

```
from random import random

x = random()
print(x)
```

.

Veja os detalhes, clicando em random

O pacote statistics

Contém funções estatísticas básicas normalmente usadas em estatística descritiva. Seu uso pode ser visto no código a seguir:

```
from statistics import mean, mode, stdev
from random import sample
data = sample(range(10000), 100)
print(data)
print(mean(data), stdev(data))
```

.

Veja os detalhes, clicando em statistics

Cálculo Numérico em Python

Cálculo Numérico

Cálculo numérico é a parte da matemática que se ocupa de construir algoritmos para solução de problemas matemáticos numéricos que vão desde a solução de uma equação de primeiro grau, passando por cálculo de logaritmos, derivadas, integrais etc.

Computação científica em Python

Python é uma linguagem de uso geral e, por isso, não conta na sua biblioteca padrão de funções que implementem métodos numéricos.

Porém, devido à sua facilidade e legibilidade ímpar, que acelera o desenvolvimento de código, muitos cientistas passaram a adotar Python como sua principal de linguagem de programação, relegando as mais tradicionais linguagens C e Fortran, comumente usadas em computação científica, às tarefas que necessitem de grande velocidade de execução.

Computação científica em Python (cont.)

Com o tempo, foram sendo criadas diversas bibliotecas para computação científica que vieram a formar um ecosistema denominador de SciPy.

O ecosistema Scipy

O Scipy é uma comunidade de desenvolvedores focados em computação científica que fornecem bibliotecas Python com os algoritmos mais usados em cálculo numérico e computação científica. As bibliotecas mais usadas são:

- numpy
- scipylib
- matplotlib
- pandas
- jupyter notebook

Os pacotes acima o NumPy e o SciPyLib formam o núcleo do que se chama ecosistema SciPy de computação científica.

O pacote numpy

- Fornece estruturas de dados e operações sobre estas úteis para cálculo numérico e, em especial, para álgebra linear.
- A estrutura de dados fundamental é o numpy ndarray, que basicamente implementa a noção de vetor da álgebra linear;
- Para mais detalhes, veja Tutorial Numpy

O pacote SciPyLib

- O pacote SciPyLib fornece módulos onde estão implementados os mais conhecidos métodos da literatura de cálculo numérico.
- Os métodos do SciPyLib são aplicados sobre a estrutura de dados numpy-array. Por isso, para usar o SciPyLib é preciso ter importado o numpy
- Conheça os módulos do SciPyLib, clicando aqui.

Distribuições de pacotes

- Os pacotes científicos estão reunidos no pacote Scipy estão disponíveis para diversos sistemas operacionais.
- No Linux, a instalação dos pacotes SciPy, por vezes, pode ser difícil e, em MS-Windows, não existe a possibilidade de instalação direta destes pacotes, sendo necessário que eles sejam compilados.
- Uma distribuição é um grande conjuntos de pacotes que são instalados simultaneamente quando da instalação da distribuição.

Desenvolvendo aplicações científicas: distribuição Anaconda Python

- A distribuição mais usada em economia e finanças é a Anaconda Python.
- Esta distribuição possui uma versão compacta, chamada Miniconda - que conta apenas com o gerenciador de pacotes Conda - e uma versão completa que inclui todos os pacotes científicos do SciPy e muitos outros.

Os ambientes de desenvolvimento integrado

- Anaconda Python vem com dois ambientes de desenvolvimento integrados: Jupyter Notebook e Spyder.
- O Jupyter Notebook é mais adequado para utilização em trabalhos interativos seguindo a metodologia e fluxo de trabalho típicos da pesquisa científica. Conheça o Jupyter Notebook.
- O Spyder é mais adequado para utilização em trabalhos de desenvolvimento de software, seguindo algum fluxo de trabalho típico da engenharia de software. Conheça o Spyder IDE.

Cálculo matricial computacional: o pacote Numpy

O pacote Numpy oferece ao Python funcionalidades para o cálculo matricial, fornecendo uma estrutura de dados denominada np.array, que atua como um vetor em álgebra linear, e métodos típicos da álgebra linear, dentro módulo linalg.

Definindo Vetores

```
import numpy as np
a = np.array([1,2,3])
print(a)
```

Lendo vetores

```
import numpy as np
a = np.array([0,0,0,0])
i=0
while i < 4:
  a[i] = float(input(Qual a coordenada %d"%(i+1)))
  i+=1
print(a)
```

Exemplo: cálculo da norma euclideana

```
import numpy as np
a = np.array([1,2,3])
print(np.linalg.norm(a))
```

Exercício: distância euclideana

Escreva um programa que solicite dois vetores, calcule a distância euclideana entre eles e coloque o resultado na tela.

Exemplo: cálculo do produto interno

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])
c = a.np.dot(b)
print(c)
```

frame
Exercício

Escreva um programa que:

- leia dois vetores do teclado, calcule o produto interno entre os vetores sem usar a função np.dot e coloque o resultado na tela.
- teste o resultado no item anterior, usando a função np.dot, imprimindo na tela a frase: "código de saída = 0" se o resultado estiver correto e "código de saída = 1" em caso contrário.

Definindo matrizes

```
import numpy as np
a = np.array([[1,2,3],[7,8,9]])

print(a)
```

Lendo matrizes

```
import numpy as np

A = [[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]
i=0
j=0
while i < 4:
while j < 4:
a[i][j]=float(input("Termo linha %d, coluna %d\n"%(i,j)))
j+=1
i+=1
j=0

B = np.array(A)
print("A matriz é:\n", B)
```

Matriz Transposta

np.A.transpose()

Exercício

Escreva um código que recebe uma matriz 5×5, calcule sua transposta.

Matriz Inversa

np.linalg.inv(A)

Exercício

Escreva um código que recebe uma matriz 6×6, calcule sua inversa.

Exemplo: produto matricial

```
import numpy as np
a = np.array([[1,2,3],[7,8,9]])
b = np.array([[4,5], [0,1],[-1,0]])

print("O formato de a é: ",a.shape)
print("A dimensão de a é: ",a.ndim)

print("O formato de b é: ",b.shape)
print("A dimensão de b é: ",b.ndim)
c = a@b
print("O produto matricial de a por b é igual  a:\n", c)
```

Determinante

Para calcular o determinante de uma matriz quadrada basta usar a função numpy.linalg.det(A), onde A é uma matriz quadrada.

Autovetores e autovalores

Os autovalores de um operador linear (matriz quadrada) A são as raízes do polinômio característico. Formalmente:

$$|A - \lambda.I| = 0$$

onde I é a matriz identidade da mesma ordem de A.

Exercício

Escreva um código que recebe uma matriz 4×4, calcule seu determinante, seus autovalores e autovetores.

Fazendo gráficos: o pacote matplotlib

- O matplotlib é o pacote Python que constrói gráficos.
- O matplotlib tem uma estrutura hierárquica de níveis mostrada a seguir:
  1. matplotlib - primeiro nível (mais alto) - fornece funções para manipulação do motor de elaboração dos gráficos. Não costuma ser usado rotineiramente.
  2. matplotlib.pyplot - segundo nível - cria a figura e os objetos básicos nela constantes para elaboração de gráficos mais simples ou interativos.
  3. interface orientada a objetos - terceiro nível - manipula objetos individuais da figura, permitindo maior controle sobre os gráficos produzidos.

Fazendo gráficos: o pacote matplotlib II

Os principais objetos de uma figura são:

- figure - é a figura propriamente dita
- axes - é o sistema de eixos cartesianos onde o gráfico é desenhado
- axis - é a linha que representa cada um dos eixos cartesianos.
- artist - é tudo o que compõe a figura inclusive como, por exemplo, axes, axis e até mesmo figure.
- line plot - linha usada em gráficos de linha
- scatter plot - ponto usado em gráficos pontilhados

Fazendo gráficos: o pacote matplotlib III

Observação

Todos os métodos e funções do matplotlib são definidos no conjunto de numpy arrays. Objetos semelhantes podem ser aceitos, mas o resultado é imprevisível. A melhor prática é sempre converter seu objeto para numpy array antes de usar uma função matplotlib.

Exemplo: gráfico simples I

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

x = np.linspace(0, 2, 100)

# Gráfico função identidade
plt.plot(x, x, label = 'identidade')

# Gráfico função quadrática
plt.plot(x, x**2, label = 'quadrática')
```

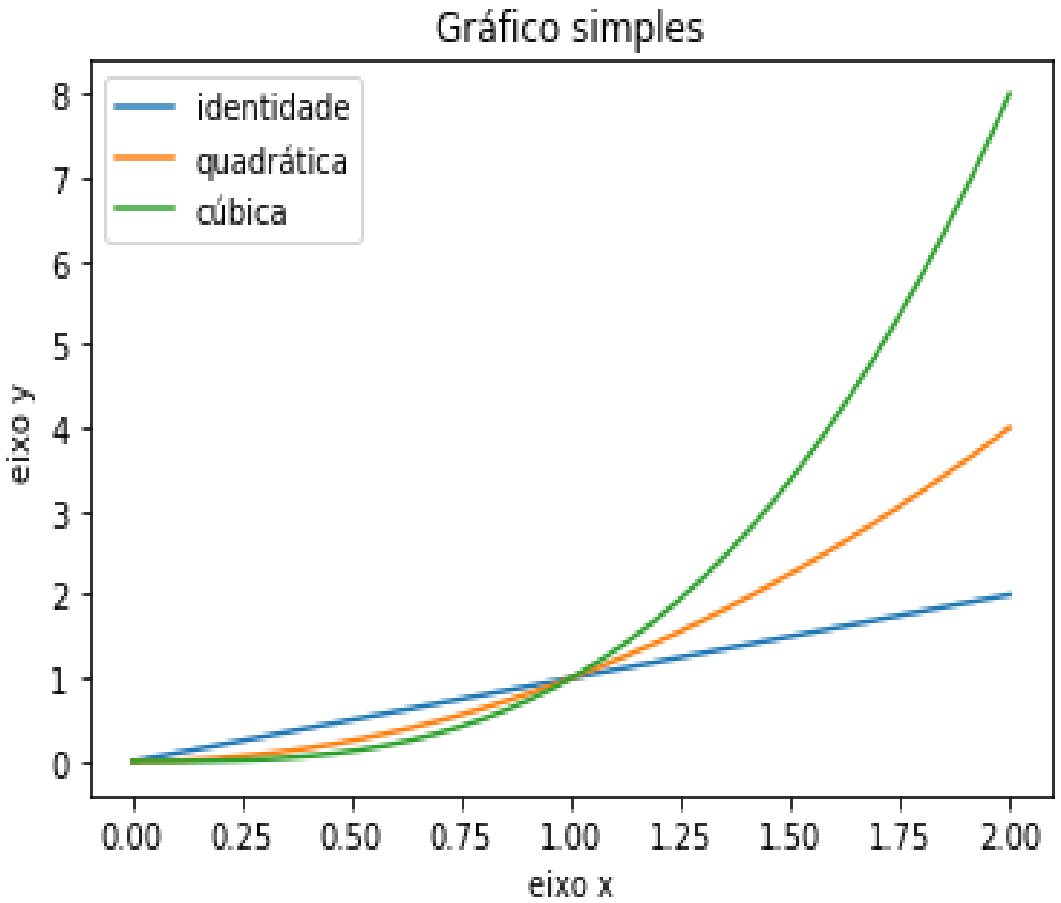
Exemplo: gráfico simples II

```
# Gráfico função cública
plt.plot(x, x**3, label = 'cúbica')

# Nomeando os eixos
plt.xlabel('eixo x')
plt.ylabel('eixo y')

# Colocando título no gráfico
plt.title('Gráfico simples')
```

Exemplo: gráfico simples III

```
# Apresentando as legendas do gráfico
plt.legend()

# Apresentando o gráfico
plt.show()
```

Exemplo: gráfico simples IV



Sistemas lineares

Sistemas Lineares

Determine a solução do sistema linear a seguir:

$$A.x = b \tag{1}$$

Onde $A_{n\times n}$, $x_{n\times 1}$ e $b_{n\times 1}$

Sistemas Lineares: exemplo

$$\begin{cases} x + y + z & = 2 \\ 9x + 3y + z & = 4 \\ 25x + 5y + z & = 6 \end{cases}$$

Sistemas lineares: soluções clássicas

- $x = A^{-}1.b$
- triangularizar A por eliminação gaussiana e resolver o sistema de baixo para cima.

Sistemas lineares: solução com módulo numpy.linalg

- A primeira solução pode ser obtida fazendo numpy.linalg.inv(A).b
- A segunda pode ser obtida com numpy.linalg.solve(A,b).

Solução do exemplo

```
# Importação do pacote linalg
import numpy as np

# Entrada de dados
A = np.array([[1,1,1],[9,3,1],[25,5,1]])
b = np.array([2,4,6])

# Processamento de dados
x = np.linalg.solve(A,b)

# Saida de dados
print(x)
```

Equações não lineares

Equações não lineares: o módulo SciPyLib

Problema

Seja $f : \mathbb{R} \to \mathbb{R}$ uma função não linear, determine a solução da equação $f(x) = 0$.

Algumas soluções clássicas

- método de bissecção
- método da iteração linear
- método de Newton
- método das secantes

Exemplo

As equações abaixo são não lineares

1. $2x^2 + 5x + 6 = 0$
2. $x^3 + 3x^2 + 3x + 1 = 0$
3. $sen(x) = cos(x + \frac{\pi}{6})$
4. $x^3 + 3x - 1$

O módulo SciPyLib

O SciPyLib é um pacote do ecosistema SciPy que provê funções de cálculo numérico, sobrescrevendo, inclusive, algumas funções disponibilizadas pelo Numpy.

O SciPyLib é estruturado como uma árvore de diretórios onde na raiz há apenas funções provenientes do Numpy. Nos subdiretórios, encontram-se as funções específicas disponíveis apenas no SciPyLib.

O módulo SciPyLib

Os submódulos SciPyLib que mais nos interessam são:

- constants - constantes matemáticas importantes
- fftpack - transformadas rápidas de Fourier
- integrate - integração
- interpolate - interpolação
- io - entrada e saída de dados
- linalg - álgebra linear
- optimize - otimização
- signal - sinais
- sparse - matrizes esparsas
- stats - distribuições de probabilidade

Solução da equação não linear: o módulo scipy.optimize

- O módulo optimize oferece funções de otimização
- A solução de uma equação não linear usa métodos de minimização do erro de aproximação.
- Estão disponíveis vários métodos, além de bisseção, Newton e secante.

Exemplo

Exercício

Resolva a equação 4 acima.

Apresentamos as soluções pelos métodos de bisseção e Brentq, pois ambos apresentam convergência matematicamente garantida, tendo o método de Brentq maior velocidade velocidade de convergência.

Solução do exemplo

```
from scipy import optimize

# entrada da função a ser otimizada
def funcao(x):
y = x**2 - 3*x - 1
return y

# processamento
solucaoBrentq = optimize.brentq(funcao, -2, 2)
solucaoBisect = optimize.bisect(funcao, -2, 2)

# saída
print("solucaoBrentq = ", solucaoBrentq)
print("solucaoBisect = ", solucaoBisect)
```

Mínimos quadrados ordinários

scipy.linalg.lstsq()

Interpolação polinomial

Problema

Determinar o polinômio P(x) que passa pelos n+1 pontos $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_{n-1}, y_{n-1}), (x_n, y_n)$.

Solução

Para resolver o problema posto, lembre que a equação geral de um polinômio de n-ésimo grau é dada por:

$$P(x) = a_0 + a_1.x + a_2.x^2 + ... + a_{n-1}.x^{n-1} + a_n.x^n$$

Como a equação acima tem n+1 coeficientes, basta-nos resolver o sistema linear a seguir:

1. $y_0 = P(x_0)$
2. $y_1 = P(x_1)$
3. ...
4. $y_{n-1} = P(x_{n-1})$
5. $y_n = P(x_n)$

Exemplo

Encontre o polinômio que passa pelos pontos abaixo:

1. $(x_0, y_0) = (1, 2)$
2. $(x_1, y_1) = (3, 4)$
3. $(x_2, y_2) = (5, 6)$

Solução

$$y = a.x^2 + b.x + c \tag{2}$$

Logo, substituindo as coordenadas dos pontos dados na equação (2) acima, temos:

$$\begin{cases} a + b + c & = 2 \\ 9a + 3b + c & = 4 \\ 25a + 5b + c & = 6 \end{cases}$$

Scipy

Alternativamente, podemos usar a função de interpolação do pacote Python Scipy

Exemplo: determinação da estrutura a termo das taxas de juros de uma economia

Problema

Determinar o polinômio que que liga os pontos $(t - t, r_{t,T})$ no plano tempo-taxa.

Referências

- Python Software Foundation
- **FRANCO**, Neide Bertoldi. *Cálculo Numérico* São Paulo: Prentice-Hall, 2006.
- **MENEZES**, Nilo Ney C. *Introdução à programação com PYTHON: algoritmos e lógica de programação para iniciantes, 2a. Ed.* São Paulo: NOVATEC, 2014