APÊNDICE F – Introdução aos módulos math, numpy, pandas e matplotlib.pyplot da linguagem Python.

Módulo math

Math é uma biblioteca que acompanha os interpretadores Python, não embutida no *core*, precisando apenas ser importada. Provê acesso, entre outras, às seguintes funções matemáticas:

- acos(x): Retorna o arco em radianos cujo cosseno é x.
- asin(x): Retorna o arco em radianos cujo seno é x.
- atan(x): Retorna o arco em radianos cuja tangente é x.
- ceil(x): Retorna o menor inteiro >= x
- cos(x): Retorna o cosseno de x radianos
- degrees(x): Converte um ângulo x de radianos para graus
- exp(x): Retorna e^x
- fabs(x): Retorna o valor absoluto de x
- factorial(x): Retorna $x!$ ou erro se x for negativo ou não inteiro
- floor(x): Retorna o maior valor inteiro <= x.
- gamma(x): Retorna a função Gamma de x.
- hypot(x, y): Retorna a distância euclidiana, $\sqrt{x^2 + y^2}$
- log10(x): Retorna $log_{10}x$
- pow(x, y): Retorna $x * y$ (x elevado a y).
- radians(x): Converte um ângulo x de graus para radianos.
- sin(x): Retorna o seno de x radianos.
- sqrt(x): Retorna $\sqrt{x}$
- tan(x): Retorna a tangente de x radianos
- e: Retorna 2.718281828459045
- pi: Retorna 3.141592653589793

Módulo numpy

NumPy é o pacote fundamental em computação científica em Python, contendo um objeto vetor n-dimensional (ndarray) de grande importância na construção de matrizes, além de operadores e transformadas sofisticados, métodos para integração com códigos em C/C++ e Fortran e métodos de álgebra linear sofisticados.

Não acompanha o interpretador Python básico da PSF, precisando ser instalado e posteriormente importado. Possui uma gama de funções, classes e variáveis enorme (aproximadamente 590) sendo as mais importantes:

- array(list, tipo): Converte uma lista em uma lista altamente manipulável chamada de ndarray, todos os elementos devem ser de um mesmo tipo.
- abs(list): Retorna um ndarray com os respectivos valores absolutos
- diag(list): Retorna um ndarray 2-D com a diagonal principal sendo os valores da lista passada como argumento.
- dot(listA, listB): Retorna o produto escalar entre duas listas.
- eye(n): Retorna a matriz indentidade nxn
- floor(list): Retorna um ndarray com os respectivos valores arredondados para menos.
- identity(n): O mesmo que eye(n)
- max(list): Retorna o valor máximo de uma lista.
- min(list): Retorna o valor mínimo de uma lista.
- ones([n,m]): Retorna um ndarray nxm preenchido com o nr 1.
- prod(list): Retorna o produto entre os elementos de uma lista.
- size(list, eixo): Retorna o tamanho da lista no eixo especificado.
- sqrt(list): Retorna um ndarray com os respectivos valores tirados a raiz quadrada.
- sum(list): Retorna o somatório dos elementos de uma lista .
- trace(list): Retorna o traço de uma lista 2-D (matriz) quadrada.
- transpose(list): Retorna a transposta da lista 1-D ou 2-D passada.
- zeros([n,m]): Retorna um ndarray nxm preenchido com o nr 0.

Do sub-módulo de algebra linear (*numpy.linalg),* alguns dos métodos mais importantes são:

- norm(list): Retorna a norma de uma lista 1-D ou 2-D

- inv(list): Retorna um ndarray com a inversa de uma lista 2-D quadrada.
- solve(listA, listB): Retorna um ndarray com a solução de um sistema linear na forma $A.x = B$, listA é 2-D quadrada e listB é 1-D ou 2-D.
- det(list): Retorna o determinante de uma matriz quadrada.
- lstsq(listA, listB): Retorna a solução de um problema MMQ na forma $A.x = B$, listA é 2-D e listB é 1-D.
- eig(list): Retorna os autovalores e auto vetores de uma matriz quadrada.

Interessante notar que NumPy tem uma função pronta para o problema MMQ (lstsq), a solução retorna um conjunto de ndarrays contendo o vetor dos parâmetros ajustado, a soma dos resíduos ao quadrado, posto da matriz de coeficientes e o um vetor contendo os valores singulares da matriz de coeficientes passada.

Dado o exemplo abaixo tem-se:

$$\begin{cases} x + y = 3 + v_1 \\ 2x - y = 1{,}5 + v_2 \\ x - y = 0{,}2 + v_3 \end{cases}$$

$$A = \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ 1 & -1 \end{bmatrix}, X = \begin{bmatrix} x \\ y \end{bmatrix}, b = \begin{bmatrix} 3 \\ 1{,}5 \\ 0{,}2 \end{bmatrix}, V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

A declaração dos vetores A e b como ndarrays seria:

```
>>> import numpy as np
>>> A=np.array([[1,1],[2,-1],[1,-1]], float)
>>> b=np.array([3,1.5,0.2],float)
```

A solução através das fórmulas de ajustamento em linguagem Python usando os métodos da inversa (inv), transposta (transpose), produto escalar (dot) e norma (norm) da biblioteca Numpy seria:

```
>>> Xa=np.linalg.inv(A.transpose().dot(A))
>>> Xa=Xa.dot(A.transpose()).dot(b)
>>> La= A.dot(Xa)
>>> V=La-b
>>> phi = np.linalg.norm(V)
>>> print Xa,'\n',La,'\n',V,'\n',phi*phi
[ 1.51428571  1.44285714]
[ 2.95714286  1.58571429  0.07142857]
[-0.04285714  0.08571429 -0.12857143]
0.0257142857143
```

A mesma solução melo método MMQ (lstsq) e produto escalar (dot) também de Numpy seria:

```
>>> Xa=np.linalg.lstsq(A,b)[0]
>>> La=A.dot(Xa)
>>> V=La-b
>>> phi=np.linalg.lstsq(A,b)[1]
>>> print Xa,'\n',La,'\n',V,'\n',phi
[ 1.51428571  1.44285714]
[ 2.95714286  1.58571429  0.07142857]
[-0.04285714  0.08571429 -0.12857143]
[ 0.02571429]
```

Não convém usar este último em ajustamento, já que não é possível incluir os pesos de cada observação. Usar o primeiro parece válido, com a ressalva do uso da função inversa (inv).

O manual de referência NumPy explica a obtenção da inversa. Na verdade ela é consequência da função solve no problema $A_n^n.X_n^n = I_n^n$, onde X assume o valor da inversa da matriz quadrada A.

No método solve, este manual faz referência ao método de solução de sistema lineares por redução de linhas (eliminação Gaussiana ou de Gauss). De fato, este método é o mais utilizado em sistemas. Ele consiste em mudar a forma do problema passando de $A.x = b$ para $U.x = c$, onde U é uma matriz triangular superior (upper), em seguida para $D.x=d$, onde D é uma matriz diagonal e então para $I.x=s$, onde I é a matriz identidade e s é a solução do problema.

A inversa por este método ($A_n^n.X_n^n = I_n^n$) possui custo computacional proporcional a n^3 na fase de decomposição e proprocional a n^2 para cada vetor na fase de solução sendo portanto consideravelmente mais custoso que uma solução $A_n^n.x_n^1 = b_n^1$ (KIUSALAAS 2013)

Abaixo, um exemplo de como utilizar a função solve do módulo NumPy, para o problema da inversa:

```
>>> A=[[1,2],[2,-1]]
>>> print numpy.linalg.inv(A)
>>> print numpy.linalg.solve(A,np.eye(2))
[[ 0.2  0.4]
 [ 0.4 -0.2]]
[[ 0.2  0.4]
 [ 0.4 -0.2]]
```

Em problemas de ajustamento, executa-se a inversa nas fórmulas:

$$Xa = (A^T.P.A)^{-1}A^T.P.Lb$$

$$\sum Xa = {\sigma_0}^2(A^T.P.A)^{-1}$$

Sabendo que A^T é (uxn), P é (nxn) e A é (nxu), onde n é o número de observações e u é o número de parâmetros, o termo da inversa fica:

$$A^{T\,n}_{u}.P^n_n.A^u_n = N^u_u$$

Decorrendo que a inversa possuirá custo computacional proporcional a (u^3 + u^2), consequentemente limitando a quantidade de parâmetros a serem ajustados.

Módulo pandas

Pandas é um rápido e eficiente organizador e manipulador de dados. Possui um objeto sofisticado capaz de organizar e incluir rótulos para a linhas e colunas de uma tabela (matriz, lista ou ndarray).

No escopo de ajustamento sua classe mais importante é a DataFrame, não sendo necessários outras funções. Ele contribui na fase de desenvolvimento com o objetivo de melhorar a visualização das matrizes.

Tomando como exemplo a matriz dos coeficientes A de uma rede de nivelamento, exibindo diretamente em comparação com a exibição após ser convertido em objeto da classe DataFrame de Pandas fica:

```
>>> import pandas
>>> A=[[1,0,0],[0,1,0],[-1,1,0],[0,0,1],[0,-1,1],[1,0,-1]]
>>> print 'Matriz A=\n',A
>>> lin=['Eq1:','Eq2:','Eq3:','Eq4:','Eq5:','Eq6:']
>>> col=['hI','hII','hIII']
>>> print u'Matriz A com r\u00F3tulos='
>>> print pandas.DataFrame(A,index=lin,columns=col)
Matriz A=
[[1,0,0],[0,1,0],[-1,1,0],[0,0,1],[0,-1,1],[1,0,-1]]
Matriz A com rótulos=
      hI  hII  hIII
Eq1:   1    0     0
Eq2:   0    1     0
Eq3:  -1    1     0
Eq4:   0    0     1
Eq5:   0   -1     1
Eq6:   1    0    -1
Eq6:   1    0    -1
```

Módulo matplotlib.pyplot

Provê um framework para plotagem muito parecido com o de MatLab®.

Algumas de suas funções importantes são:

- imread(‘arquivo.png'): importa imagens PNG
- figure(n): invoca uma nova janela onde poderá ser plotada a n-ésima figura
- title(‘string’): define o título da figura atual em que se está trabalhando
- imshow(img, cmap=’mapa_de_cor’): plota a matriz img na figura atual, cmap é opcional e define um mapa de cores caso a matriz não seja do tipo byte.
- Savefig(‘nome_da_figura.jpg’): Salva a figura atual com o nome desejado
- Show(): permite a visualização das janelas de figuras criadas
- subplot(nrows, ncols, plot_number): define um grid de subplotagens [nrows ncol] e onde deve ser posicionada a plotagem atual plot_number.
- subplots(): permite várias plotagens em uma única figura
- plot(X, Y): plota os pares ordenados (x,y) dos vetores X e Y respectivamente
- fill_between(X, y1, ymax, facecolor=’nome_da_cor’): preenche de cor os pontos do vetor X desde y1 até ymax.
- axvline(pos_X, color=’nome_da_cor’, linewidth='n'): cria uma linha reta vertical na plotagem atual na posição indicada por pos_X de cor e espessura indicada
- xlabel(‘rotulo do eixo X’): define o rotulo das abcissas
- ylabel(‘rotulo do eixo Y’): define o rótulo das ordenadas