MÓDULOS PYTHON

Clerisvaldo Holanda dos Santos Junior

As atividades a seguir foram feitas a fim de mostrar funções de alguns módulos do python como ***Numpy***, ***Scipy***, ***Timeit*** e ***Matplotlib***, e compará-las com uma forma equivalente programada no *python* para alguns tipos de problemas.

1 Sistema de Equações Lineares

A priori, foi importado os módulos e definida a função do método de resolução por *Gauss-Seidel*. Após isso, foi implementado ao código uma geração de *matriz 20x20* aleatória de forma que ela fosse *estritamente diagonal dominante*.

A partir daí, foi utilizada a função *Gauss-Seidel*, onde são dados os valores das matrizes com os coeficientes e resultados das equações e ela retorna os valores das variáveis em uma matriz. E para a forma de resolução utilizando o **Numpy** foi utilizada a função *np.linalg.solve()*, onde são utilizados os mesmos argumentos da função programada e também retorna os valores das variáveis como uma matriz.

Finalmente, foi possível comparar os dois métodos utilizados e ver que a função direta do **Numpy** foi demasiadamente rápida em relação ao método programado no *python* e com uma diferença de precisão insignificante.

2 Interpolação

Não foi possível concluir.

3 Ajustes

Para esta atividade, primeiro foi importado os módulos, definida uma função que determinava um *polinômio de até 4° grau* que recebe o valor de x e seus

coeficientes em ordem decrescente.

Após isso, foram criados *pontos* utilizando a função definida acima, adicionada a um valor denominado ruído, o qual adiciona um valor ao retornado pela função, e que após isso foram plotado esses pontos.

Para a forma programada no *python* para o *método dos mínimos quadrados*, foi utilizada a relação a seguir:

$$g = A \cdot B \longrightarrow B = \left[A^T \cdot A\right]^{-1} \cdot A^T \cdot g$$

onde g são os valores de y dos pontos, A é a matriz com os valores atribuídos de x para uma função de 4°distribuída em colunas, e B a matriz de coeficientes.

Para a forma utilizando o módulo **Scipy**, foi utilizada a função *curve_fit()*, retornando os valores para os coeficientes também em uma matriz.

Ao final disso, foram plotados os gráficos para as duas formas de resoluções e foi percebido que quando adicionado o ruído os polinômios eram idênticos, e ao retirar este, os polinômios divergiam de maneira irrelevante. Além disso, foi possível observar também que o tempo de duração da forma de resolução programada no *python* foi mais rápida que a forma de resolução utilizando o **Scipy**.

4 Integração e Diferenciação

Primeiramente foi feita a importação dos módulos, definida uma função de sexto grau e uma função para o cálculo da integral pelo método do trapézio. Esta função consiste em:

1. Receber os valores de x, y e o número de intervalos;
2. Cria um vetor com índices nulos do tamanho do número de intervalos subtraindo 1;
3. Calcula a área de cada trapézio formado;
4. Retorna o somatório das áreas calculadas anteriormente armazenadas no vetor.

Como valor real para a integral, foi utilizada a função *integrate.quad()* do **Scipy** que tinha uma precisão de *1e-5*. Após isso foi feito um estudo

de valores variando o tamanho do intervalo, e quanto menor o tamanho do intervalo, mais a integral numérica convergia para o valor analítico com um porém; quando utilizado a função descrita anteriormente, o tempo para cálculo da integral aumentava proporcionalmente ao número de intervalos, enquanto a função *integrate.trapz()* para o mesmo método utilizando o **Scipy** mostrava pouca variação no tempo com o aumento do número de intervalos.

Foi feita uma tabela mostrando os dados obtidos e um gráfico mostrando a variação do tempo em função do número de intervalos.