ção 222  ção 3ção 332 ção 8ção 863

LISTA 01

# TREINAMENTO PARA NOVOS BOLSISTAS 2019

Autor: JOÃO VICTOR FERRO

LCCV/UFAL
Maceió, ABRIL DE 2019

# 1 Introdução

## 1.1 Solução matricial pelo método de Gauss-Seidel

O método de Gauss-Seidel é um método iterativo para resolução de sistemas de equações lineares. O seu nome é uma homenagem aos matemáticos alemães Carl Friedrich Gauss e Philipp Ludwig von Seidel. É semelhante ao método de Jacobi (e como tal, obedece ao mesmo critério de convergência). É condição suficiente de convergência que a matriz seja estritamente diagonal dominante, i. e., fica garantida a convergência da sucessão de valores gerados para a solução exacta do sistema linear.

Procuramos a solução do conjunto de equações lineares, expressadas em termos de matriz como uma iteração. Por Gauss-Seidel é:

$$x^{k+1} = (D + L)^{-1} \times (-U \times x^k + b) \tag{1}$$

Onde A=D + L + U, as matrizes D, L, e U representam respectivamente os coeficientes da matriz A. Quando o método é implementado para fins computacionais. Temos a aproximação explícita da entrada por aproximação:

$$x_i^{(k+1)} = \frac{1}{a_{ij}}(b_i - \sum_{j<i} a_{ij}x_i^{(k+1)} - \sum_{j>i} a_{ij}x_i^{(k)}) \tag{2}$$

Em seguida os novos valores encontrados podem ser avaliados de modo que sejam comparados ao valor anterior da solução estejam num limite aceitável. O rigor do tamanho da tolerância depende de quem está implementando o código.

## 1.2 Interpolação pelo método de Langrange

Em análise numérica, polinômio de Lagrange (nomeado por razão de Joseph-Louis de Lagrange) é o polinômio de interpolação de um conjunto de pontos na forma de Lagrange. Se quiser determinar uma curva que passa por todos os pontos do conjunto, temos que:

Dado um conjunto (k+1) de pontos (x,y):

$$(x_0, y_0), ..., (x_n, y_n) \tag{3}$$

Com todos xj distintos, o polinômio de interpolação de um conjunto de pontos na forma de Lagrange é a combinação linear dos polinômios da base de Lagrange:

$$L(x) := \sum_{j=0}^{k} y_i l_j \tag{4}$$

E com os polinômios da base de Lagrange dados por:

$$l_j(x) := \prod_{i=a}^{b} \frac{x - x_i}{x_j - x_i} = \frac{x - x_0}{x_j - x_0} ... \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} ... \frac{x - x_k}{x_j - x_k} \tag{5}$$

Afim de resolver este problema, o matemático Joseph-Louis de Lagrange escolheu uma outra base que melhorasse o condicionamento da matriz. A ideia foi diagonalizá-la, obtendo uma matriz identidade cuja resolução do sistema linear é simples e direta. Dados n pontos de abscissas, o polinômio interpolador de Lagrange ,Pn(x), será obtido através de uma base de polinômios de grau menor ou igual n, satisfaça a condição de que em cada linha para o k= j , então 1. Mas se 0, temos que k=/= j.

Observe que vamos obter uma série de k polinômios de tal modo que cada um deles se anula em todos os pontos conhecidos com exceção de um em que k=j, de forma que cada polinômio ajuste o valor em um ponto, sendo funções independentes entre si.

$$L_0(x) = \frac{(x - x_1)(x - x_2)(x - x_3) \times ... \times (x - x_n)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3) \times ... \times (x_0 - x_n)} \tag{6}$$

De modo que seja feito até $L_n(x)$:

$$L_n(x) = \frac{(x - x_0)(x - x_1)(x - x_2) \times ... \times (x - x_{n-1})}{(x_n - x_0)(x_n - x_1)(x_0 - x_2) \times ... \times (x_n - x_{n-1})} \tag{7}$$

O Polinômio interpolador de Lagrange é dado pela combinação linear dos $L_k(x_k)$ polinômios base:

$$P_0 = a_0 L_0(x_0) + a_1 L_1(x_0) + ... + a_n L_n(x_0) \tag{8}$$

De modo que seja feito até $L_n(x_n)$:

$$P_0 = a_0 L_0(x_n) + a_1 L_1(x_n) + ... + a_n L_n(x_n) \tag{9}$$

Desta forma, podemos reduzir as equações anteriores em matrizes. Para que em seguida seja encontrados os coeficientes. $(a_0, a_1, ..., a_n)$

## 1.3 Método dos mínimos quadrados - MMQ

O Método dos Mínimos Quadrados, ou seja MMQ, é uma técnica de otimização matemática que procura encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados, tais diferenças são chamadas resíduos.

Um requisito para o método dos mínimos quadrados é que o fator imprevisível (erro) seja distribuído aleatoriamente e essa distribuição seja normal. O Teorema Gauss-Markov garante (embora indiretamente) que o estimador de mínimos quadrados é o estimador não-enviesado de mínima variância linear na variável resposta.

Outro requisito é que o modelo é linear nos parâmetros, ou seja, as variáveis apresentam uma relação linear entre si. Caso contrário, deveria ser usado um modelo de regressão não-linear. Para determinar os coeficientes da regressão, seja ela um polinômio de grau n. Neste caso, para uma regressão de grau 2, temos que:

Seja a melhor aproximação:

$$y = a_0 + a_1 x + a_2 x^2 \tag{10}$$

Então os "melhores"valores dos coeficientes são:

$$S_r = \sum_{i=1}^{n} (y_i - a_0 + a_1 x_i + a_2 x_i^2)^2 \tag{11}$$

São aqueles em que o resíduo $S_r$ seja o mínimo, ou seja, podemos estipular que as derivadas parciais dos coeficientes em relação ao resíduo seja mínimo.E, Desta maneira sejam iguais a zero.

$$\frac{\partial S_r}{\partial a_0} = \sum_{i=1}^{n} -2(y_i - a_0 + a_1 x_i + a_2 x_i^2) = 0 \tag{12}$$

$$\frac{\partial S_r}{\partial a_1} = \sum_{i=1}^{n} -2x(y_i - a_0 + a_1 x_i + a_2 x_i^2) = 0 \tag{13}$$

$$\frac{\partial S_r}{\partial a_2} = \sum_{i=1}^{n} -2x_i^2(y_i - a_0 + a_1 x_i + a_2 x_i^2) = 0 \tag{14}$$

Ao unir essas equações em um matriz, podemos obter os valores que melhor definem uma regresão, os valores podem ser obtidos utilizando o próprio método de Gauss - Seidel, que foi apresentado anteriormente.

## 1.4 Integração Pelo método do trapézio

A integral de uma função pode ser interpretado como a área entre a abcissa e o gráfico da função que está integrando. Deste modo, para uma dada função f (x), queremos aproximar a integral por n trapezóides.

$$\int_a^b f(x)\,dx = \int_{x_0}^{x_1} f(x)\,dx + ... + \int_{x_n-1}^{x_n} f(x)\,dx \tag{15}$$

Mas, ao aproximar por trapézios, temos que:

$$\int_a^b f(x)\,dx \approx h\frac{f(x_0)+f(x_1)}{2} + ... + h\frac{f(x_{n-1})+f(x_n)}{2} \tag{16}$$

Ao simplificar a equação anterior temos que:

$$\int_a^b f(x)\,dx \approx h[\frac{f(x_0)}{2} + \sum_{i=1}^{n-1}[f(x_i)] + \frac{f(x_0)}{2}] \tag{17}$$

Deste modo facilitando a integração numérica.

# 2 Resultados e discussões

## 2.1 Gauss - Seidel

Primeiramente foi deselvolvido o algoritmo que não utiliza nenhum pacote extra, apenas as ferramentas comuns do *Python*. No primeiro momento o código funciona basicamente seguindo a cção 2Equação 2 que é basicamente receber uma matriz, um vetor com os valores de Y correspondentes e um número n que vai determinar o tamanho desta matriz quadrada, consequentemente o vetor também.

```
def Gauss_seidel(matriz,vetor,n):
        solucao_ant = [0 for _ in range(n)]
        solucao_pos = [0 for _ in range(n)]
        erro = [10 for _ in range(n)]
        tolerancia = 0.1
        flag = 0
        while flag != n:
                flag = 0
                for i in range (0,n,1):
                        for j in range (0,n,1):
                                if i != j:
                                        solucao_pos[i] +=
                                           matriz[i][j]*
                                           solucao_ant[j]
                                elif matriz[i][i]  == 0:
                                         print('Erro
                                            encontrado! A
                                            diagonal
                                            principal
                                            cont m zeros'
                                            )
                                         break
```

```
                        solucao_pos[i] = (1/matriz[i][i])*(
                            vetor[i] - solucao_pos[i])
                        erro[i] = abs(solucao_pos[i] -
                            solucao_ant[i])
                        solucao_ant[i] = solucao_pos[i]
                        solucao_pos = [0 for _ in range(n)]
                        if erro[i] < tolerancia:
                                flag= flag + 1
        return solucao_ant
```

O restante do código vai trabalhar utilizando duas partes que trabalham em conjunto. A primeira parte consiste na atualização sequencial dos valores para cada elemento do vetor soluçao. E, ao mesmo tempo temos temos o elemento avaliativo que é o erro, ele irá determinar quando o processo recursivo irá parar, fazendo com que o comando superior *while* pare e volte para a linha superior e retorne o resultado.

Para avaliar o primeiro código, foi criada uma função *main*, ela tem o objetivo de fornecer os dados necessários para que que possa ser calculada. Vale ressaltar que esse programa não avalia a matriz em sua totalidade. Para que ele seja eficiente, deve ser fornecida uma matriz dominante, ou seja, os elementos da diagonal principal representem, em sua maioria, os maiores entre os outros de sua respectiva linha da matriz.

```
import numpy as np
import time
import random
from Matriz_aleatoria_nxn import Matriz_aleatoria_nxn
from Gauss_seidel import Gauss_seidel

def main(n):
    start = time.time()
    matriz, vetor,n = Matriz_aleatoria_nxn(n)
    ans = Gauss_seidel(matriz,vetor,n)
    end = time.time()
    tempo = (end - start)
    return ans, tempo
```

A resposta que era recebida, geralmente, quando não havia divergência e este tendia ao infito tinha certa demora em sua execução se comparado com o algoritmo que utiliza os pacotes *numpy*, pois, a medida que este código exige de 2 a 3 segundos para solucionar o problema, o outro com pacote. *numpy*

resolve a mesma questão em aproximadamente 0.05 segundos.

```
#!/usr/bin/env python
# coding: utf-8
# ## Importando M dulo ## #
#C:\Users\User\Documents\UFAL\LCCV\Projeto 003\Atividade 1\
    Solu   o com pacotes
import numpy as np
def Gauss_seidel_num (matriz,vetor):
        ans = np.linalg.solve(matriz,vetor)
        return ans
```

O código implementado com o pacote surpreende com sua eficácia, resolve as matrizes de maneira rápida e exige muito menos do computador. Essa observação se baseia na diferença gigantesca entre o que não utiliza qualquer pacote extra.

## 2.2 Lagrange

Do mesmo modo que foi para resolver com o método de Gauss - Seidel, para encontrar os valores de um polinômio de Lagrange, vai ser utilizado o processo iterativo de Gauss-Seidel para determiná-los. Desta forma o custo computacional que o algoritmo vai ser muito maior que quando vai simplesmente calcular a matriz. Pois neste algoritmo ainda será preciso ler os valores para cada ponto,cção 3Equação 3. E Calcular cada valor de x em uma função, cção 8Equação 8.

```
from Gauss_seidel import Gauss_seidel
def Interpol_lagrange (x_data,y_data,grau):
        matriz  = []
        for i in range (0,grau+1,1):
                vetor = []
                for j in range (0,grau+1,1):
                        vetor.append(x_data[i]**j)
                matriz.append(vetor)
        print(matriz)
        print(y_data)
        solucao = Gauss_seidel(matriz,y_data,(grau+1))
        return solucao
```

Em comparação com o código com pacotes, além de ser muito maior, executa muito mais devagar que o código sem pacotes anterior. A solução com pacotes é mais explícita e objetiva. Além do que, executa o código na faixa de 0.06 segundos, enquanto que o algoritmo sem pacotes tem oscilasções grandes de tempo, cerca de 0.09 até 5 segundos.

```
#!/usr/bin/env python
# coding: utf-8
get_ipython().run_line_magic('matplotlib', 'inline')
import numpy as np
from scipy.interpolate import lagrange
import matplotlib.pyplot as plt
from polinomio import polinomio

def Lagrange_num(x_data):
        x_coord = np.array(x_data)
        y_coord = polinomio(x_data)
        poly_lag = lagrange(x_pt,y_pt)
        ans = (np.polynomial.polynomial.Polynomial(poly_lag
            ).coef)
        return ans



print(x_pt)
plt.plot(x_pt,y_pt,'o', color = 'red')
print(poly_lag)
print(np.polynomial.polynomial.Polynomial(poly_lag).coef)
x_tst = np.linspace(0,8,20)
y_tst = poly_lag(x_tst)
plt.plot(x_tst,y_tst,'b-')
plt.plot(x_pt,y_pt,'ro')
plt.title('Grafico')
```

# 3 Conclusão

Esse tipo de criptografia é muito eficiente sua implementação é simples, seu manuseio também é. No entanto deve se conter a alguns cuidados. Primeiramente, o código tem que gerar números primos aleatórios. E a depender de como isso for feito o processo pode ficar mais extendido. Ao passo que, se for gerada uma chave pública proveniente de um numero primo muito grande, vai levar muito tempo tempo para ser descoberta a chave privada. Por fim

concluí-se que essa implementação possui alguns erros a serem corrigidos e que ainda é muito ineficiente.