**TRABALHO COMPUTACIONAL 3**

Nomes:

* Daiana Santos 120.357
* Isadora Muniz 120.431
* Luciana Bello 120.506
* Maria Victória Siqueira 120.529

# **Parte 1**

Os exercícios a seguir têm como abordagem a resolução de sistemas lineares de n equações e n incógnitas utilizando o método iterativo de Gauss-Jacobi.

**Exercício 1 - Parte 1**

Para a resolução deste problema, dividiu-se em duas partes: **interpretação do método** e sua **implementação em Python**.

Interpretação do método:

Dado o seguinte sistema:

x1 - x2 + 5x4 = 18

3x1 - 2x2 + x3 - x4 = 8

x1 + x2 + 9x3 + 4x4 = 47

x1 - 7x2 + 2x3 + 3x4 = 32

Para resolvê-lo utilizando o método, isola-se os valores de x1, x2, x3 e x4 em cada uma das equações, a fim de encontrar seus valores. Como exemplo, isolando o elemento x1 na primeira equação tem-se:

x1 = 18 + x2 – 5x4

Desejando realizar uma iteração com estes valores, a equação pode ser reescrita na forma a seguir:

O mesmo feito para x1 ocorre para x2 na segunda equação, e assim sucessivamente para as incógnitas seguintes. Como forma de compactar a equação de iteração, pode ser escrita como:

Uma análise de grande importância antes da resolução do método é quanto a convergência do sistema. Existem condições para a convergência do sistema, do qual pode ser analisada através do **Critério das Linhas** ou **Critério das Colunas**, que determina se a matriz é diagonalmente dominante. Se as condições de um dos critérios forem atendidas, significa que o sistema converge independente de valores iniciais, porém, se não forem atendidas, não há garantia de sua convergência.

Estes critérios são feitos da seguinte maneira:

*Critério das linhas*

Obtendo os valores da diagonal principal da matriz aumentada, a soma dos elementos da linha que não fazem parte da diagonal principal deve ser **maior** que o elemento pertencente a ela. Exemplo com base no sistema dado acima:

* 1 < (-1 + 5) Não atende critério
* -2 < (3 + 1 – 1) Não atende critério
* 9 > (4 + 1 + 1)Atende critério
* 3 > (1 – 7 + 2)Atende critério

*Critério das Colunas*

É realizado da mesma forma que o critério anterior, porém, com base nos valores da coluna da matriz aumentada. Exemplo com base no sistema dado acima:

* 1 < (3 + 1 + 1)Não atende critério
* -2 < (-1 + 1 – 7)Atende critério
* 9 > (0 + 1 + 2)Atende critério
* 3 > (5 + 4 - 1)Não atende critério

Sendo assim, de acordo com o exemplo acima, podemos concluir que o sistema não converge.

Quanto ao critério de parada adotado, é da distância relativa entre a solução atual e a anterior seja menor do que a tolerância (10^-6):

Já para a implementação do código a mesma ideia foi aplicada, tendo a facilitação de algumas iterações com funções fornecidas pela biblioteca **numpy** do Python. Seu resultado encontra-se nos anexos da parte 1.

**Exercício 2 - Parte 1**

De acordo com o problema, tem-se a seguinte matriz:

1 1 1

1 1 2

2 3 5

Analisando com base nos critérios das linhas e das colunas, temos:

*Critério das linhas:*

* 1 < (1 + 1) Não atende
* 1 < (1 + 2)Não atende
* 5 = 5Não atende

*Critério das colunas:*

* 1 < (1 + 2)Não atende
* 1 < (1 + 3)Não atende
* 5 > (1 + 2)Atende

Resposta: Logo, de acordo com as análises, não é possível garantir convergência neste sistema resolvendo pelo método de Gauss-Jacobi..

## 

## **Anexos - parte 1**

**Código exercício 1 - Parte 1**

import numpy as np

from numpy import linalg

def CriterioLinhas(A):

continua = 1

for i in range(np.shape(A)[0]) :

vet = np.zeros(2)

for j in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das linhas")

continua = 0

break

if(continua):

print("Sistema atende o criterio das linhas")

return continua

def CriterioColunas(A):

continua = 1

j=0

while(j < (np.shape(A)[0])):

vet = np.zeros(2)

for i in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

#print("Vet[0]= " + str(vet[0]) + " Vet[1]= "+ str(vet[1]))

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das colunas")

continua = 0

break

j+=1

if(continua == 1):

print("Sistema atende o criterio das colunas")

return continua

def MetodoGaussJacobi(A,b,x0,tolerancia,N):

A = A.astype('double')

b = b.astype('double')

x0 = x0.astype('double')

n=np.shape(A)[0]

x = np.zeros(n)

it = 0

while (it < N):

it = it+1

#metodo de Gauss-Jacobi

for i in np.arange(n):

x[i] = b[i]

for j in np.concatenate((np.arange(0,i),np.arange(i+1,n))):

x[i] -= A[i,j]\*x0[j]

x[i] /= A[i,i]

#precisao

if (np.linalg.norm(x-x0,np.inf) < tolerancia):

return x

#nova iteracao

x0 = np.copy(x)

raise NameError('num. max. de iteracoes excedido.')

#\*\*\*\*\*\*\*\*\*\*\*\*FUNCAO PRINCIPAL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

n = int(input("Valor de n: "))

A = np.zeros((n,n))

b = np.zeros(n)

x0 = np.zeros(n)

for r in range(0,n):

for c in range(0,n):

A[(r),(c)] = float(input("Elemento a["+ str(r+1)+ str(c+1)+"] = "))

b[r] = float(input("b["+str(r+1)+"] = "))

if(CriterioLinhas(A) or CriterioColunas(A)):

x = MetodoGaussJacobi(A, b, x0, 10\*\*-6, 10)

print(x) #Imprime vetor x de resolucao

# **Parte 2**

**Exercício 1 - Parte 2**

O método iterativo de **Gauss-Seidel** é útil para resolver sistemas lineares a partir de uma aproximação inicial qualquer, ainda que longe da desejada. Para o exemplo, foi utilizado a forma de **Ax = b**, assim, há a matriz expandida para A e para B. Um valor inicial deve ser inserido, **x0**, como estimativa de solução, para que a análise comece. A diferença para o Método de **Gauss-Jacobi** é que neste método o resultado de é utilizado para calcular para n valores da matriz. Dessa forma, mesmo que x0 seja uma estimativa muito longe da ideal, o método iterativo irá realizar N iterações até atingir um valor próximo e que respeite a condição de erro com aproximação em .

Com a matriz expandida A, temos que: se o valor de cada componente da diagonal principal for maior que a soma dos valores da sua mesma linha, a matriz é dita **estritamente diagonal dominante**, ou seja, o método **converge** na situação aplicada. No código feito, este passo foi verificado pelo **critério das linhas e colunas** já utilizado em exercícios da Parte 1.

Para teste foram utilizados os valores a seguir:

**A** = 1 -1 0 5 **B =** 18

3 -2 1 -1 8

1 1 9 4 47

1 -7 2 3 32

Após aplicar o programa feito em anexo ao problema usandos os valores no enunciado para teste, podemos ver que o resultado **não atende** aos critérios estabelecidos para que o método convirja.

**Exercício 2 - Parte 2**

Usando o **Método de Gauss-Seidel** para resolver a este problema, foi possível notar que o método chega a resultados aproximados, mas não exatamente iguais, o que é contemplado pelo erro. Podemos notar que neste caso, esse não é o melhor método para resolver o problema devido a inexatidão dos resultados. Ao aplicar o problema no programa feito, foi visto que a equação **não converge**.

Valor de n: 3

Elemento a[11] = 1

Elemento a[12] = 1

Elemento a[13] = 1

b[1] = 400

Elemento a[21] = 1

Elemento a[22] = 1

Elemento a[23] = 2

b[2] = 600

Elemento a[31] = 2

Elemento a[32] = 3

Elemento a[33] = 5

b[3] = 1500

x[0] = 120

x[1]=120

x[2]=300

Sistema nao atende o criterio das linhas

Vet[0]= 1.0 Vet[1]= 3.0

Sistema nao atende o criterio das colunas

**Exercício 3 - Parte 2**

Para o **método de Gauss**, ou eliminação gaussiana, é feito o escalonamento da matriz, resultando em elementos nulos abaixo ou acima da diagonal principal. Assim, variáveis são eliminadas, possibilitando calcular diretamente o valor individual das mesmas a partir da última equação. Tanto para o método de **Gauss-Jacobi** e de **Gauss-Seidel** é inserido uma estimativa de valor inicial e através de sucessivas iterações, o resultado é apurado até chegar em um valor final com o erro estipulado(para esse caso, o utilizado foi de ). De acordo com os resultados, é possível encontrar valores próximos quanto ao desempenho dos métodos de Gauss-Jacobi e Gauss-Seidel, por serem resoluções parecidas.

## **Anexos - Parte 2**

**Código para o exercício 1 - Parte 2:**

import numpy as np

from numpy import linalg

def CriterioLinhas(A):

continua = 1

for i in range(np.shape(A)[0]) :

vet = np.zeros(2)

for j in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das linhas")

continua = 0

break

if(continua):

print("Sistema atende o criterio das linhas")

return continua

def CriterioColunas(A):

continua = 1

j=0

while(j < (np.shape(A)[0])):

vet = np.zeros(2)

for i in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

print("Vet[0]= " + str(vet[0]) + " Vet[1]= "+ str(vet[1]))

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das colunas")

continua = 0

break

j+=1

if(continua == 1):

print("Sistema atende o criterio das colunas")

return continua

def gauss\_seidel(A,b,x0,tol,N):

#preliminares

A = A.astype('double')

b = b.astype('double')

x0 = x0.astype('double')

n=np.shape(A)[0]

x = np.copy(x0)

it = 0

#iteracoes

while (it < N):

it = it+1

#iteracao de Jacobi

for i in np.arange(n):

x[i] = b[i]

for j in np.concatenate((np.arange(0,i),np.arange(i+1,n))):

x[i] -= A[i,j]\*x[j]

x[i] /= A[i,i]

print(x[i],A[i,i])

#tolerancia

if (np.linalg.norm(x-x0,np.inf) < tol):

return x

#prepara nova iteracao

x0 = np.copy(x)

raise NameError('num. max. de iteracoes excedido.')

n = int(input("Valor de n: "))

A = np.zeros((n,n))

b = np.zeros(n)

x0 = np.zeros(n)

for r in range(0,n):

for c in range(0,n):

A[(r),(c)] = float(input("Elemento a["+ str(r+1)+ str(c+1)+"] = "))

b[r] = float(input("b["+str(r+1)+"] = "))

if(CriterioLinhas(A) or CriterioColunas(A)):

x = gauss\_seidel(A, b, x0, 10\*\*-6, 10)

print(x) #Imprime vetor x de resolucao

**Código para o exercício 2 - Parte 2**

Neste exercício foi utilizado o código feito acima, porém com os valores referidos no enunciado. **O resultado foi explicitado no relatório da parte do Exercício 2 Parte 2.**

import numpy as np

from numpy import linalg

def CriterioLinhas(A):

continua = 1

for i in range(np.shape(A)[0]) :

vet = np.zeros(2)

for j in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das linhas")

continua = 0

break

if(continua):

print("Sistema atende o criterio das linhas")

return continua

def CriterioColunas(A):

continua = 1

j=0

while(j < (np.shape(A)[0])):

vet = np.zeros(2)

for i in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

print("Vet[0]= " + str(vet[0]) + " Vet[1]= "+ str(vet[1]))

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das colunas")

continua = 0

break

j+=1

if(continua == 1):

print("Sistema atende o criterio das colunas")

return continua

def gauss\_seidel(A,b,x0,tol,N):

#preliminares

A = A.astype('double') #valores utilizados para cada movel de cada material

b = b.astype('double') #estoque total de cada material

x0 = x0.astype('double')

n=np.shape(A)[0]

x = np.copy(x0)

it = 0

#iteracoes

while (it < N):

it = it+1

#iteracao de Jacobi

for i in np.arange(n):

x[i] = b[i]

for j in np.concatenate((np.arange(0,i),np.arange(i+1,n))):

x[i] -= A[i,j]\*x[j]

x[i] /= A[i,i]

print(x[i],A[i,i])

#tolerancia

if (np.linalg.norm(x-x0,np.inf) < tol):

return x

#prepara nova iteracao

x0 = np.copy(x)

raise NameError('num. max. de iteracoes excedido.')

n = int(input("Valor de n: "))

A = np.zeros((n,n))

b = np.zeros(n)

x0 = np.zeros(n)

for r in range(0,n):

for c in range(0,n):

A[(r),(c)] = float(input("Elemento a["+ str(r+1)+ str(c+1)+"] = "))

b[r] = float(input("b["+str(r+1)+"] = "))

if(CriterioLinhas(A) or CriterioColunas(A)):

x = gauss\_seidel(A, b, x0, 10\*\*-6, 10)

print(x) #Imprime vetor x de resolucao

# 

**Código para o exercício 3 - Parte 2**

from random import randrange

import numpy as np

from sklearn.feature\_extraction import DictVectorizer

def CriterioLinhas(A):

continua = 1

for i in range(np.shape(A)[0]) :

vet = np.zeros(2)

for j in range(np.shape(A)[0]):

if(j != i):

vet[1] += A[i,j]

else:

vet[0] = A[i,j]

if(vet[0] <= vet[1]):

print("Sistema nao atende o criterio das linhas")

continua = 0

break

if(continua):

print("Sistema atende o criterio das linhas")

return continua

def MetodoGaussJacobi(A,b,x0,tolerancia,N):

A = A.astype('double')

b = b.astype('double')

x0 = x0.astype('double')

n=np.shape(A)[0]

x = np.zeros(n)

it = 0

while (it < N):

it = it+1

#metodo de Gauss-Jacobi

for i in np.arange(n):

x[i] = b[i]

for j in np.concatenate((np.arange(0,i),np.arange(i+1,n))):

x[i] -= A[i,j]\*x0[j]

x[i] /= A[i,i]

#precisao

if (np.linalg.norm(x-x0,np.inf) < tolerancia):

print("Gauss-Jacobi = " + str(x) + " Iteracoes = " + str(it)) #Imprime vetor x de resolucao

return x

#nova iteracao

x0 = np.copy(x)

raise NameError('num. max. de iteracoes excedido.')

def gauss\_seidel(A,b,x0,tol,N):

#preliminares

A = A.astype('double')

b = b.astype('double')

x0 = x0.astype('double')

n=np.shape(A)[0]

x = np.copy(x0)

it = 0

#iteracoes

while (it < N):

it = it+1

#iteracao de Jacobi

for i in np.arange(n):

x[i] = b[i]

for j in np.concatenate((np.arange(0,i),np.arange(i+1,n))):

x[i] -= A[i,j]\*x[j]

x[i] /= A[i,i]

#print(x[i],A[i,i])

#tolerancia

if (np.linalg.norm(x-x0,np.inf) < tol):

print("Gauss-Seidel = " + str(x) + " Iteracoes = " + str(it)) #Imprime vetor x de resolucao

return x

#prepara nova iteracao

x0 = np.copy(x)

raise NameError('num. max. de iteracoes excedido.')

def Gauss(matriz,b):

n=np.shape(matriz)[0]

for i in range(0, n-1):

for j in range(i+1, n):

m = matriz[j][i]/matriz[i][i]

matriz[j][i] = 0

for k in range(i+1, n):

matriz[j][k] = matriz[j][k] - (m \* matriz[i][k])

b[j] = b[j] - m\*b[i]

#print("A Matriz final ficou:")

#print(matriz) #MATRIZ TRIANGULAR SUPERIOR

x = np.linalg.solve(matriz, b)

#print("O vetor solucao eh:")

#print(x) #VETOR SOLUCAO

dim = 1000, 1000

mat = {}

# Tuplas são imutáveis

# Cada tupla representa

# uma posição na matriz

k=0

for lin in range(dim[0]):

for col in range(dim[1]):

k+=1

if(col== lin):

mat[col, lin] = randrange(20, 40) #faixa de inteiro

if(k == 300):

mat[col, lin] = randrange(0, 9)

k=0

l=0

for lin in range(dim[0]):

for col in range(dim[1]):

# Método get(chave, valor)

# retorna o valor da chave

# no dicionário ou se a chave

# não existir, retorna o

# segundo argumento

if(col== dim[1] - 1):

print(mat.get((lin, col), 0), )

else:

print(mat.get((lin, col), 0), end=" ")

#Create DictVectorizer object

dictvectorizer = DictVectorizer(sparse=False)

# Convert dictionary into feature matrix

features = dictvectorizer.fit\_transform(mat)

CriterioLinhas(features)

b = np.random.randint(1000, size = 1000)

x0 = np.random.randint(1000, size = 1000)

x = MetodoGaussJacobi(features, b, x0, 10\*\*-6, 100)

x1 = gauss\_seidel(features, b, x0, 10\*\*-6, 100)

x2 = Gauss(features, b)

# **Parte 3**

**Exercício 1 - Parte 3**

No exercício, nos foi fornecido alguns pontos de x (porcentual de álcool anidro) e seus respectivos valores de f(x) (Viscosidade). Utilizando a interpolação polinomial, consideramos o conjunto de nós de interpolação *x0 , ... , xn* , a que estão associados os valores de uma função *f0 , ... , fn*, respectivamente.

Pretendemos encontrar um polinômio p tal que *p( x ) = a0 + a1 x + ... + am xm*, em que

| *a0 + a1 x0 + ... + am x0m = f0* |
| --- |
| ... |
| *a0 + a1 xn + ... + am xnm = fn* |

Os pontos passados no exercício foram:

x*0* = 5 f(x*0*) = 1226

x*1* = 10 f(x*1*) = 1498

x*2* = 15 f(x*2*) = 1822

x*3* = 20 f(x*3*) = 2138

x*4* = 30 f(x*4*) = 2662

x*5* = 40 f(x*5*) = 2840

sendo assim, o modelo do polinômio segue:

p(x) = *a0 + a1 x1 + a2 x2 + a3 x3 + a4 x4 + a5 x5*

Para descobrir o valor de a*0*, … a*5  montou o sistema:*

*a0 + a1 51 + a2 52 + a3 53 + a4 54 + a5 55 = 1226*

*a0 + a1 101 + a2 102 + a3 103 + a4 104 + a5 105 = 1498*

*a0 + a1 151 + a2 152 + a3 153 + a4 154 + a5 155 = 1822*

*a0 + a1 201 + a2 202 + a3 203 + a4 204 + a5 205 = 2138*

*a0 + a1 301 + a2 302 + a3 303 + a4 304 + a5 305 = 2662*

*a0 + a1 401 + a2 402 + a3 403 + a4 404 + a5 405 = 2840*

Calculando a solução, obteve-se os resultados:

*a0=*

*a1=−*

*a2=*

*a3=−*

*a4=*

*a5=−*

*Substituindo no modelo, o polinômio de interpolação polinomial resultou em:*

*p(x) = - x1 + x2 - x3 + x4 - x5*

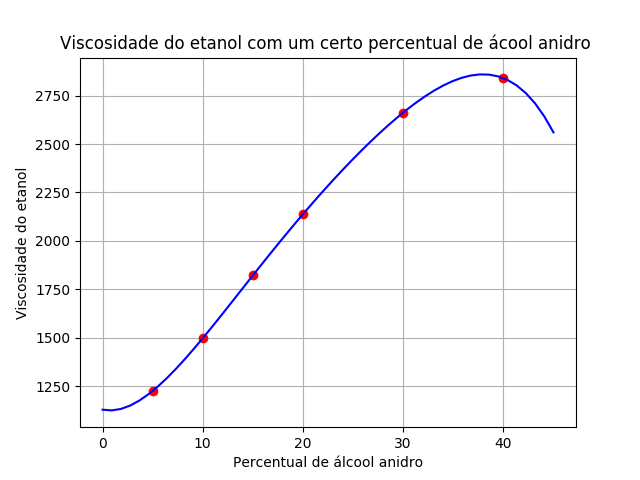
*No problema, foi pedido para que determinasse a viscosidade para os valores de porcentagem de 12, 25 e 38, substituindo no polinômio encontrado ficará:*

*p(12) = - 121 + 122 - 123 + 124 - 125 = 1625,849*

*p(25) = - 251 + 252 - 253 + 254 - 255 = 2422,5428*

*p(38) = - 381 + 382 - 383 + 384 - 385 = 2859,66*

O gráfico que representa o polinômio encontrado está abaixo, com os pontos dados no problema, gerado através do código que está em anexo:



**Exercício 2 - Parte 3**

O polinômio de Lagrange possui a forma:

P(x) = f(x*0*) L*n,0*(x) + ... + f (x*n*)L*n,n*(x) = f (x*k* )L*n,k* (x),

no qual os pontos são:

x*0* = 1 f(x*0*) = 14,5

x*1* = 2 f(x*1*) = 19,5

x*2* = 3 f(x*2*) = 30,5

x*3* = 4 f(x*3*) = 53,5

x*4* = 5 f(x*4*) = 94,5

x*5* = 6 f(x*5*) = 159,5

Polinômio deve ficar na forma:

P(x) = f(x*0*) . L*1*(x) + f(x*1*) . L*2*(x) + f(x*2*) . L*3*(x) + f(x*3*) . L*4*(x) + f(x*4*) . L*5*(x) + f(x*5*) . L*6*(x)

Onde L*k*(x) é calculado da seguinte maneira:

L*1*(x) =

L*2*(x) =

L*3*(x) =

L*4*(x) =

L*5*(x) =

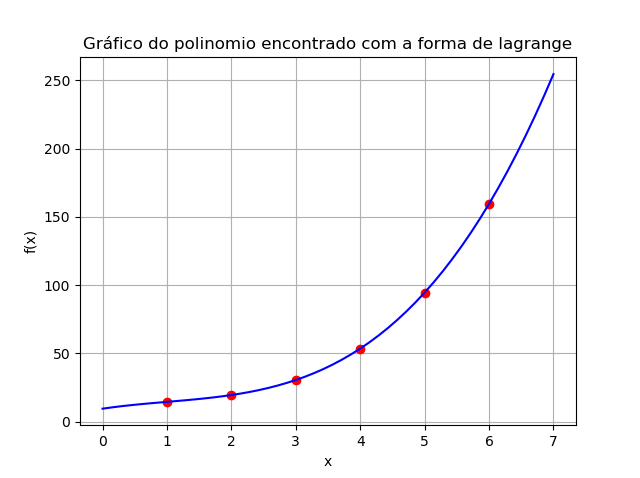
L*6*(x) =

Sendo assim, o polinômio é:

P(x) = 14,5 . L*1*(x) + 19,5 . L*2*(x) + 30,5 . L*3*(x) + 53,5 . L*4*(x) + 94,5 . L*5*(x) + 159,5 . L*6*(x) =

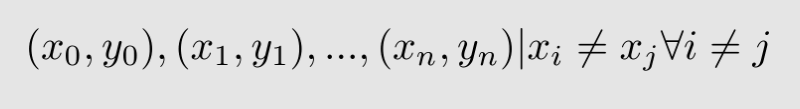
Para calcular o valor de 4,5 é somente substituir no polinômio, ficando:

Concluindo que pode-se obter o mesmo resultado que o problema sugeriu.

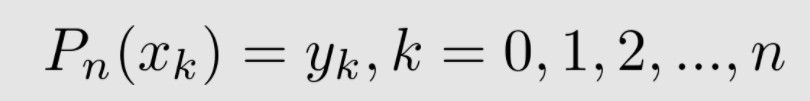


**Exercício 3 - Parte 3**

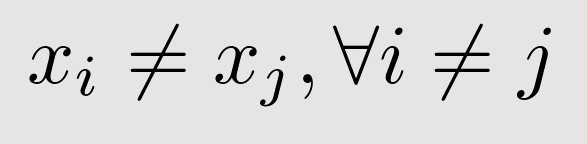
Para a resolução do problema, foi necessário a utilização do método de **Interpolação Polinomial** através da **Fórmula de Lagrange**. A **interpolação polinomial** consiste em dado diversos pontos, tais que:

,

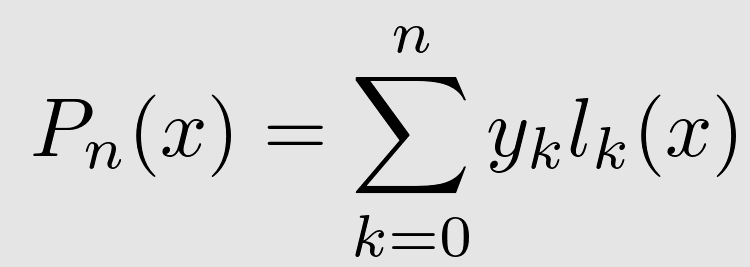
dessa forma, precisa-se obter um polinómio de grau n dado por:

,

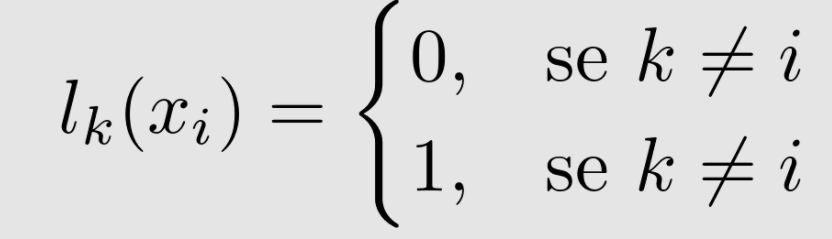
logo, dados diversos pontos de (x,y) só existe um polinômio de grau em que:



Para o interpolador polinomial de Lagrange temos que:

,

para que P(x) seja o polinômio interpolador desejado, os polinômios l(x) devem ter grau n ( ou menor) e satisfazer :

 ou seja, a condição l(x)=0 com k diferente de i, diz que x, com i diferente de k são n raízes do polinômio l(x). Para a implementação do código e resolução do problema, o polinômio de Lagrange de primeira ordem foi aplicado. Portanto, **para x= 4** o **valor de y encontrado** foi de: **7,47496597102**

# **Anexos parte 3**

**Código para o exercício 1 - Parte 3**

import matplotlib.pyplot as plt

import numpy as np

xi = np.array([5,10,15,20,30,40], dtype ='double')

yi = np.array([1226,1498,1822,2138,2662,2840], dtype ='double')

A = np.array([xi\*\*5,xi\*\*4,xi\*\*3,xi\*\*2,xi\*\*1,xi\*\*0]).transpose()

a = np.linalg.inv(A).dot(yi)

c = (a[5] + a[4]\*(12\*\*1) + a[3]\*(12\*\*2) + a[2]\*(12\*\*3) + a[1]\*(12\*\*4) + a[0]\*(12\*\*5))

print('Valor da Viscosidade com a porcentagem de alcool igual a 12: ' + str(c))

c = (a[5] + a[4]\*(25\*\*1) + a[3]\*(25\*\*2) + a[2]\*(25\*\*3) + a[1]\*(25\*\*4) + a[0]\*(25\*\*5))

print('Valor da Viscosidade com a porcentagem de alcool igual a 25: ' + str(c))

c = (a[5] + a[4]\*(38\*\*1) + a[3]\*(38\*\*2) + a[2]\*(38\*\*3) + a[1]\*(38\*\*4) + a[0]\*(38\*\*5))

print('Valor da Viscosidade com a porcentagem de alcool igual a 38: ' + str(c))

xx = np.linspace(0,45)

plt.plot(xi,yi,'ro',xx,np.polyval(a,xx),'b-')

plt.grid()

plt.title('Viscosidade do etanol com um certo percentual de ácool anidro ')

plt.xlabel('Percentual de álcool anidro')

plt.ylabel('Viscosidade do etanol')

plt.show()

**Código para o exercício 2 - Parte 3**

from scipy.interpolate import lagrange

import matplotlib.pyplot as plt

import numpy as np

x=[1,2,3,4,5,6]

y=[14.5, 19.5, 30.5, 53.5, 94.5, 159.5]

p=lagrange(x,y)

xx = np.linspace(0,7)

plt.plot(x,y,'ro',xx,np.polyval(p,xx),'b-')

plt.grid()

plt.title('Gráfico do polinomio encontrado com a forma de lagrange')

plt.xlabel('x')

plt.ylabel('f(x)')

plt.show()

print(p(4.5))

**Código para o exercício 3 - Parte 3**

from scipy.interpolate import lagrange

import numpy as np

x=[2.00,4.25,5.25,7.81,9.20,10.60]

y=[7.2, 7.1, 6.0, 5.0, 3.5, 5.0]

p=lagrange(x,y)

print(p(4.0))