Universidade Federal de Pelotas

Trabalho de Cálculo Numérico

Mateus Brugnaroto

Relatório sobre os Sistemas de Equações Lineares e Sistemas de Equações Não Lineares

Pelotas – Rio Grande do Sul - 06/05/2019

Sumário

1. Introdução

Este trabalho tem por finalidade abordar o conteúdo sobre a Solução de Equações Lineares e Não Lineares. Será tratado técnicas numéricas para obtenção de solução desses sistemas. Essas técnicas (métodos) serão implementadas computacionalmente através da linguagem Python.

Nesse artigo, irei abordar algumas características de cada implementação de método, como:

- Entrada e saída do algoritmo implementado
- Critérios utilizados
- Comparações entre os métodos iterativos e diretos

2. Solução de Sistemas de Equações Lineares

Nesse tópico, irei descrever um pouco sobre o funcionamento de cada método implementado para obtenção de soluções de sistemas lineares.

2.1. Métodos Diretos

2.1.1. Funções em Comum

1. Equações fundamentais para Gauss e LU:
   - def $calcM\ (Matriz,\ linha,\ base)$:
     - Define a função:

$$M[n] = \frac{Matriz[linha][base]}{pivo[base]}$$

   - def $calcMatriz\ (Matriz,\ linha,\ base)$:
     - Define os valores novos que serão retornados para cada posição (n) em uma linha (L):

$$L[n] = L[linha] - M[n] \times L[base]$$

2. Equações fundamentais para Cholesky:
   - def $raizG\ (MatrizOriginal)$:
     - Define a função que retornará o valor para a primeira posição da matriz G:

$$G[0][0] = raiz(MatrizOriginal[0][0])$$

   - def $divisaoG\ (MatrizOriginal, g, pivo)$:
     - Define a função que retornará o valor para a linha denominada pela variável *pivo* na primeira coluna:

$$r = (MatrizOriginal[pivo][0])/G[0][0]$$

- def $diagonalG\ (MatrizOriginal, g, pivo)$:
  - Define a função que calcula o valor em uma posição *pivo* na diagonal principal e o retorna:

$$r = MatrizOriginal[pivo][pivo] - \sum_{i=1}^{pivo-1} (g[pivo][i])^2 \quad ^{1/2}$$

- def $restanteG\ (MatrizOriginal, g, linha, coluna)$:
  - Define a função que calcula o valor que falta para completar a matriz no triangulo inferior:

$$r = (MatrizOriginal[linha][coluna] - \sum_{i=1}^{pivo-1} g[linha][i] * g[coluna][i])/(g[coluna][coluna])$$

2.1.2. Eliminação de Gauss

1. <u>Entrada de dados e inicialização da Matriz:</u>

O método utiliza-se de uma matriz de dimensão $linhas\ x\ colunas$, onde as condições $linhas < colunas$ e $linhas + 1 = colunas$ devem ser satisfeitas. Cada linha da matriz corresponde a uma única equação.

Exemplo:

$$\begin{cases} 2x + 1y - 1z + 8w = 6 \\ -3x - 1y + 2z - 11w = 12 \\ -2x + 1 + 2 - 3w = -4 \\ 3x + 5 + 12 - 5w = 2 \end{cases}$$

$$Matriz = ([[2, 1, -1, 8, 6], [-3, -1, 2, -11, 12], [-2, 1, 2, -3, -4], [3, 5, 12, -5, 2]])$$

O tipo dos dados e o arredondamento é definido na própria inicialização da matriz ($dtype = np.float32$), ou seja, o tipo dos dados visualizados na entrada será o mesmo na saída.

2. <u>Método:</u>

Em sua base, usa-se de dois laços de repetição (*while*) para fazer o acesso a cada posição da matriz. O primeiro *while* garante que será percorrido cada coluna da matriz e o segundo cada linha. A equação a seguir (juntamente com os lanços) garante que somente os dados abaixo da diagonal principal serão alterados para satisfazer as condições de Gauss. Termina quando todas as posições da matriz forem percorridas.

$$Matriz[n] = calcMatriz(Matriz, linha, base)$$

3. Chamada do Método e saída dos dados (visualização da resposta):

O método de Gauss é chamado através da função $def\ Gauss(Matriz)$, sendo necessário somente a matriz de entrada como parâmetro para resolução do sistema. A resposta será mostrada com o escalonamento da matriz.

$$\begin{Bmatrix} 2 & 1 & -1 & 8 & 6 \\ 0 & 0.5 & 0.5 & 1 & 21 \\ 0 & 0 & -1 & 1 & -82 \\ 0 & 0 & 0 & -14 & -974 \end{Bmatrix}$$

2.1.3. Fatoração LU

1. Entrada de dados e inicialização da Matriz:

O método utiliza-se de uma matriz de dimensão $linhas\ x\ colunas - 1$, onde as condições $linhas < colunas$ e $linhas + 1 = colunas$ devem ser satisfeitas. O que será passado para a matriz de entrada será somente os valores que estarão acompanhando as incógnitas. Cada linha da matriz corresponde a uma única equação.

Exemplo:

$$\begin{Bmatrix} 3x + 2y + 4z = 0 \\ 1x + 1y + 2z = 12 \\ 4x + 3y - 2z = -4 \end{Bmatrix}$$

$$Matriz = ([[3, 2, 4], [1, 1, 2,], [4, 3, -2,]])$$

O tipo dos dados e o arredondamento é definido na própria inicialização da matriz ($dtype = np.float32$), ou seja, o tipo dos dados visualizados na entrada será o mesmo na saída.

2. Método:

Em sua base, usa-se de dois laços de repetição (*while*) para fazer o acesso a cada posição da matriz. O primeiro *while* garante que será percorrido cada coluna da matriz e o segundo cada linha. As duas equações seguintes formam a fatoração LU:

$$1)\ Matriz[n] = calcMatriz(Matriz, linha, base)$$

$$2)\ L[n][coluna] = calcM(MLU, linha, base)$$

A equação 1) garante que somente os dados abaixo da diagonal principal serão alterados para satisfazer as condições de U (*upper*). A equação 2) calcula os valores para satisfazer a condição de L (*lower*). Termina quando todas as posições da matriz forem percorridas e os valores, em cada uma das matrizes, atualizados.Observação: é criada uma nova matriz L para representação dos dados.

3. Chamada do Método e saída dos dados (visualização da resposta):

O método de LU é chamado através da função $def\ LU(Matriz)$, sendo necessário somente a matriz de entrada como parâmetro para resolução do sistema. A resposta será mostrada em duas matrizes:

- A primeira com o escalonamento satisfazendo a condição do L.
- A segunda terá a diagonal principal com valor unitário e a parte triangular inferior os valores calculados pela equação 2.

Exemplo:

Matriz *U*:

$$\begin{Bmatrix} 3 & 2 & 4 \\ 0 & 0.3 & 0.6 \\ 0 & 0 & -8 \end{Bmatrix}$$

Matriz *L:*

$$\begin{Bmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ 1.3 & 0.9 & 1 \end{Bmatrix}$$

2.1.4. Fatoração Cholesky

1. Entrada de dados e inicialização da Matriz:

O método utiliza-se de uma matriz de dimensão $linhas\ x\ colunas - 1$, onde as condições $Matriz[linha][coluna] = Matriz[coluna][linha]$, com a diagonal principal contendo apenas valores positivos, devem ser satisfeitas. O que será passado para a matriz de entrada será somente os valores que estarão acompanhando as incógnitas. Cada linha da matriz corresponde a uma única equação.

Exemplo:

$$\begin{cases} 4x + 12y - 16z = 0 \\ 12x + 37y - 43z = 12 \\ -16x - 43y + 98z = -4 \end{cases}$$

$$Matriz = ([[4, 12, -16], [12, 37, -43], [-16, -43, 98]])$$

O tipo dos dados e o arredondamento é definido na própria inicialização da matriz ($dtype = np.float32$), ou seja, o tipo dos dados visualizados na entrada será o mesmo na saída.

3. Método:

Inicialmente, chama-se a função $raizG()$ para fazer o cálculo do valor para a primeira posição da matriz G. Após calculado, utiliza-se de um lanço *while,* chamando a função $divisaoG()$, para calcular todos os valores da primeira coluna da matriz G. O segundo *while,* que chama a função $diagonalG()$, é usado para calcular os valores de cada posição na diagonal principal. Porém, quando é calculado um valor da diagonal

principal, entra-se em outro laço *while* que calcula o restante das posições daquela coluna através da função $restanteG()$. Termina após percorrer cada posição do triangulo inferior.

4. Chamada do Método e saída dos dados (visualização da resposta):

O método de Cholesky é chamado através da função $def\ Cholesky\ (MC)$, sendo necessário somente a matriz de entrada como parâmetro para resolução do sistema. A resposta será mostrada em duas matrizes:

- A primeira mostrará a matriz G com os valores atualizados após todos os cálculos necessários.
- A segunda será a transposta da matriz G.

Exemplo:

Matriz *G*: Transporta *G:*

$$\begin{Bmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{Bmatrix} \qquad \begin{Bmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{Bmatrix}$$

2.2. Métodos Iterativos

2.2.1 Funções em Comum

1. Equações fundamentais para Gauss e LU:
   - def $valoresB\ (a, t)$:
     - Seleciona os valores da diagonal principal para o cálculo do x0.
   - def $calc\_x0\ (x, b, t)$:
     - Calcula o valo de x0:

$$calc\ =\ b[i]/valoresX[i]$$

2.2.2. Gauss-Jacobi

1. Entrada de dados e inicialização da Matriz:

O método utiliza-se de uma matriz de dimensão $linhas\ x\ colunas - 1$. O que será passado para a matriz de entrada será os valores que estarão acompanhando as incógnitas. Os valores que não acompanham uma incógnita devem ser passados ao vetor chamado de *b,* o número de iterações na variável *it* e a precisão na variável *precisao*. Cada linha da matriz corresponde a uma única equação.

Exemplo:

$$\begin{cases} 10x + 2y + 1z = 7 \\ 1x + 5y + 1z = -8 \\ 2x + 3y + 10z = 6 \end{cases}$$

$$Matriz = ([[2, 1, -1], [-3, -1, 2,], [-2, 1, 2,]])$$

$$b = [7, -8, 6]$$

$$it = 10$$

$$precisao = 0.05$$

O tipo dos dados e o arredondamento é definido na própria inicialização da matriz ($dtype = np.float32$), ou seja, o tipo dos dados visualizados na entrada será o mesmo na saída.

2. Método:

Inicialmente, calcula-se o valor de x0 através das funções def $valoresB()$ e def $calc\_x0()$. O primeiro laço de repetição *while* (k < it) garante que o método calculará os valores das incógnitas no máximo o número de iterações passada pelo usuário, contudo, se outra condição de parada for satisfeita, o método termina. O segundo (w < t) e terceiro (l < t) *while* efetivarão a execução do método de Jacobi através da fórmula a seguir:

$$y = \frac{1}{matriz[w][w]} * b[w] + \sum_{l=0}^{l} -matriz[w][l] * x0[l]$$

Onde o *x0*, ao fim do método, será atualizado com os novos valores de *y*.

$$x0 = x1 = y$$

O restante do código contém os facilitadores para os cálculos e a exibição do resultado final. O método termina quando o número de iterações chegar ao valor passado pelo usuário ou a variável *dy* ser menor que a precisão:

$$dy = parada < precisao:$$

3. Chamada do Método e saída dos dados (visualização da resposta):

O método de Jacobi é chamado através da função $def\ gauss\_jacobi(a, b, it, precisao)$, sendo necessário a matriz de entrada, o valores que não seguem nenhuma incógnita, número de iterações e a precisao como parâmetro para resolução do sistema. A resposta será exibida através dos valores das incógnitas em sequência, o número de iterações para chegar naquele resultado e o valor da precisão alcançada.

Exemplo:

Iteração: 0 [0.7, -1.6, 0.6]	Iteração: 1 [0.96, -1.86, 0.94] dy: 0.1828
Iteração: 2 [0.978, -1.98, 0.96] dy: 0.0606	Iteração: 3 [0.99, -1.98, 0.99] dy: 0.0163

2.2.3. Gauss-Seidel

1. Entrada de dados e inicialização da Matriz:

O método utiliza-se de uma matriz de dimensão $linhas\ x\ colunas - 1$. O que será passado para a matriz de entrada será os valores que estarão acompanhando as incógnitas. Os valores que não acompanham uma incógnita devem ser passados ao vetor chamado de *b,* o número de iterações na variável *it* e a precisão na variável *precisao*. Cada linha da matriz corresponde a uma única equação.

Exemplo:

$$\begin{cases} 10x + 2y + 1z = 7 \\ 1x + 5y + 1z = -8 \\ 2x + 3y + 10z = 6 \end{cases}$$

$$Matriz = ([[2, 1, -1], [-3, -1, 2,], [-2, 1, 2,]])$$

$$b = [7, -8, 6]$$

$$it = 10$$

$$precisao = 0.05$$

O tipo dos dados e o arredondamento é definido na própria inicialização da matriz ($dtype = np.float32$), ou seja, o tipo dos dados visualizados na entrada será o mesmo na saída.

2. Método:

Inicialmente, calcula-se o valor de x0 através das funções $\text{def}\, valoresB()$ e $\text{def}\, calc\_x0()$. O primeiro laço de repetição *while* (k < it) garante que o método terá no máximo o número de iterações passada pelo usuário, contudo, se outra condição de parada for satisfeita, o método termina. O segundo (w < t) e terceiro (l < t) *while* efetivarão a execução do método de Seidel através da fórmula a seguir:

$$y = \frac{1}{matriz[w][w]} * b[w] + \sum_{l=0}^{l} -matriz[w][l] * xj[l]$$

Onde o *xj* tem como função receber o valor atualizado da incógnita calculada e já ser utilizada para o cálculo da próximo variável em uma mesma iteração.

$$xj[w] = y$$

O restante do código contém os facilitadores para os cálculos e a exibição do resultado final. O método termina quando o número de iterações chegar ao valor passado pelo usuário ou a variável *dy* ser menor que a precisão:

$$dy = parada < precisao$$

3. Chamada do Método e saída dos dados (visualização da resposta):

O método de Seidel é chamado através da função $def\ gauss\_seidel(a, b, it, precisao)$, sendo necessário a matriz de entrada, o valores que não seguem nenhuma incógnita, número de iterações e a precisão como parâmetros para resolução do sistema. A resposta será exibida através dos valores das incógnitas em sequência, o número de iterações para chegar naquele resultado e o valor da precisão alcançada.

Exemplo:

Iteração: 0
[0.7, -1.6, 0.6]

Iteração: 1
[0.96, -1.912, 0.9816]
dy: 0.1996

Iteração: 2
[0.98424, -1.9931, 1.00011]
dy: 0.0606

2.3. Comparação entre os métodos diretos e iterativos

Através de testes de esparsidade, arredondamento e convergência, pude concluir que:

1. Os métodos iterativos, quando há convergência, são mais eficazes para a solução de sistemas grandes. Tem vantagem por evitar os problemas de instabilidade numérica, que podem ocorrer em um método direto, e são menos suscetíveis ao acúmulo de erros de arredondamento. Se houver uma grande quantidade de elementos iguais à 0, se torna ainda mais eficiente.
2. Os Métodos diretos mostram-se mais eficientes para solução de sistemas pequenos, sendo ainda mais eficientes se a matriz tiver muitos elementos diferentes de 0.

3. Solução de Sistemas Não Lineares

Nesse tópico, irei descrever um pouco sobre o funcionamento do método de Newton implementado para obtenção de soluções de sistemas não lineares.

3.1. Funções

1. Equações fundamentais para Gauss e LU:
    - def $f(x1)$:
        - Tem como função resolver o sistema de entrada, retornando um vetor.
    - def $Jac(x1)$:
        - Tem como função substituir as incógnitas do sistema pelo vetor resultado.

3.2. Newton para Sistemas

1. Entrada de dados e inicialização da Matriz:

Os dados de entrada devem ser passados em uma matriz (*matriz*), para o número de iterações (*it*) será reservado uma variável, como também para a precisão (*precisao*) escolhida pelo usuário.

Exemplo:

matriz = [1,5]

it = 10

precisao = 0.00001

2. Método:

Em sua base, usa-se de um laço de repetição (*while*) que garante que o método fará no máximo o número de iterações passada pelo usuário, contudo, se outra condição de parada for satisfeita, o método termina. O código a seguir funciona como divisor de matriz (*s*). Para encontrá-la, é divido a *f(x)* pela *Jac(x),* assim, satisfazendo a primeira condição para resolução de Newton. Após isso, é utilizado a variável *num* para armazenar o valor do primeiro elemento da matriz (*s*) que será usado para critério de parada por precisão.

3. Chamada do Método e saída dos dados (visualização da resposta):

O método de Newton é chamado através da função $def\ Newton(Matriz, it, precisao)$, sendo necessário a matriz de entrada, número de iterações e a precisão como parâmetros para resolução do sistema. A resposta será mostrada da seguinte forma:

Exemplo:

Iteração: 0
s:
[-1.625 -1.375]
Solução:
[-0.625 3.625]

Iteração: 1
s:
[0.53308824 -0.53308824]
Solução:
[-0.09191176 3.09191176]

Iteração: 2
s:
[0.08925842 -0.0892]
Solução:
[-2.653e-03 3.00e+00]

Iteração: 3
s:
[0.002651 0.002651]
Solução:
[-2.34259734e-06 3.00000234e+00]

Iteração: 4
s:
[2.3421e-06 -2.341e-06]
Solução:
[-1.8292e-12 3.00e+00]