INTRODUÇÃO À COMPUTAÇÃO CIENTÍFICA

IIº Trabalho: Sistemas Lineares

Tobias J. D. E. Rosa

2022

1º Período

# 

# Resumo

O presente trabalho faz o uso dos métodos de resolução de sistema linear, de Gauss Jacobi, Gauss Seidel e pivoteamento do tipo LU para resolver sistemas com qualquer número de equações a partir da implementação computacional dos métodos, através de programação em linguagem Fortran. Por fim, é feita a validação do programa resolvendo o problema proposto com 30 equações e 30 incógnitas, que pode ser reescrito como uma matriz 30x30.

Apresentar de forma concisa como foi abordada a solução do problema enunciado, quais os métodos tentados para a solução e um adiantamento das conclusões mais relevantes.

# Solução do Problema

Sistemas lineares podem facilmente serem resolvidos, até mesmo de forma manual se for composto por um número pequeno de equações. Entretanto, muitas vezes é necessário trabalhar com elevados números de equações ao mesmo tempo, como no caso do cálculo de esforços em treliças, por exemplo. No problema apresentado, a matriz 30x30 possui 900 elementos que seguem um padrão de distribuição, o que viabiliza a criação de um programa especificamente para gravar os elementos dessa matriz, por ser na maior parte formada por zeros, exceto na diagonal principal a(i,i) e nos elementos sucessor a(i,i+1) e antecessor a(i,i-1).

O programa utilizado para gravar os valores na matriz utiliza dois comandos “do”, o primeiro incrementa valores em i e o segundo em j, associados aos comandos “if/elseif” que identificaram quando a referida posição é diagonal principal (i=j), antecessor à diagonal principal (i-j=1) ou sucessor à diagonal principal (j-i=1). O referido programa gera uma matriz em um arquivo do tipo “.txt” que será o próprio arquivo de entrada do programa principal de resolução de sistemas lineares.  
 A implementação computacional do método de Gauss Jacobi tem como ideia principal criar uma variável para armazenar o somatório dos elementos Aij\*Xj de toda a linha, exceto do elemento Xii a ser isolado e do elemento bi=Ain, pertencente à última coluna. Desta forma, pode ser criado um algoritmo que percorre todas as linhas e em cada linha, realiza o somatório acumulado dos elementos exceto quando i=j e j=n, em que n é o número total de colunas. Uma particularidade do método de Gauss Jacobi é que os elementos Xii são calculados e armazenados em uma variável de armazenamento temporário e são utilizados apenas na próxima iteração.

onde exceto quando

O método de Gauss Seidel, em geral utiliza a mesma estratégia do método de Gauss Jacobi, se distinguindo apenas pelo fato de que não haverá uma variável de armazenamento temporário, cada elemento Xii calculado é utilizado para definir o próximo, xi+1,i+1 que também é utilizado para calcular os demais elementos até o Xnn.

O método do PLU tem similaridades com o método da eliminação de Gauss, e de certa forma espera-se que utilize maior esforço computacional do que o próprio método da eliminação de Gauss para resolver problemas isolados. Entretanto a grande vantagem do método do PLU é que ele trabalha apenas com a Matriz dos coeficientes, não altera os elementos do vetor b durante as iterações, isso facilita porque para calcular o próximo vetor b não será mais necessário repetir o esforço computacional, diferentemente do método da eliminação de Gauss.

**Tabela 1** - Resultados obtidos

| **GAUSS JACOBI** | **GAUSS SEIDEL (mm)** | **PLU** |
| --- | --- | --- |
| 0.29411148559530315  0.58822085231169030  0.88233430116116873  1.1764421004960814  1.4705567319437447  1.7646639324852855  2.0587787370416066  2.3528862692702979  2.6470004421297708  2.9411088946459119  3.2352219897559387  3.5293317828058695  3.8234431172698100  4.1175561544646619  4.4116610382417552  4.7057900019561334  4.9998575348510030  5.2940805091738161  5.5879205049295040  5.8827072906613767  6.1751718535330360  6.4733392168077533  6.7575350671251222  7.0759695853074582  7.3105508462760351  7.7505038022855128  7.6875337337228595  8.8561458473444432  7.0088746904671035  12.546854700014146 | 0.29411764912033472  0.58823529831338284  0.88235294729250968  1.1764705968210598  1.4705882449784722  1.7647058965239300  2.0588235397358132  2.3529412033987045  2.6470588169313940  2.9411765532769505  3.2352939888229448  3.5294121610190672  3.8235285292658237  4.1176493150353837  4.4117592832317669  4.7058957412827622  4.9999673315856086  5.2941977684777015  5.5880392256983900  5.8828332068494564  6.1752946706731180  6.4734679459078590  6.7576542770113921  7.0760914949096749  7.3106559787031111  7.7506059629678727  7.6876129556446049  8.8562144484502401  7.0089174435809154  12.546878936800184 | 0.0000000000000000  -0.19943021355715970  -0.38245033043765964  2.7605986039173105E-002  -0.14211792402907819  -0.31141017821862049  0.10278403387087633  -6.6376897869712528E-002  -0.23553400540249808  -0.40469055702607498  9.5353916710531894E-003  -0.15962142175059893  -0.32877712245952906  8.5448548059418580E-002  -8.3707709005872544E-002  -0.25286396607116368  0.16136170444778397  -7.7945526175071580E-003  -0.17695080968279828  -0.34610706674808944  6.8118603770858238E-002  -0.10103765329443290  -0.27019446671608505  0.14403176015922362  0.55825798703453222  1.5558650287813578  2.5534731832409050  4.1344621525719694  7.4655980173684720  12.546878552204969 |

# Apresentação e Discussão de Resultados

Os programas apresentaram valores muito próximos, exceto o método PLU. Faltou detalhes a respeito do tempo de processamento, erro e número de iterações que pode ser apresentado posteriormente se houver a oportunidade.

**Conclusões**

Os métodos de gauss jacobi e seidel são mais simples de serem implementados, sendo o método de gauss seidel de convergência mais rápida por armazenar simultaneamente o valor de x anterior e já o utilizar para o próximo valor de x. O método do PLU é mais trabalhoso de se implementar, entretanto é mais vantajoso que o método da eliminação de gauss pois permite utilizar as matrizes L e U encontradas para encontrar infinitas soluções para infinitos vetores b conhecidos.

# Referências

[RUGGIERO, Marcia A. Gomes; LOPES, Vera Lucia da Rocha. *Cálculo numérico: aspectos teóricos e computacionais*. São Paulo: Pearson Makron Books, 2008.](https://www.zotero.org/google-docs/?0BCUCt)

CÓDIGOS-FONTE

# matriz.f90

!gfortran matriz.f90

!./a.out

!esse programa gera a matriz 30x30 do problema proposto e o vetor b em uma coluna adicional, em…

! ..um arquivo com nome “matriz.txt”

!Tobias / IME 2022

program matriz

implicit none

real, dimension (:,:), allocatable :: darray

character(20)::argument

integer :: s1, s2

integer :: i, j

open(2,file="matriz.txt",status='unknown')

s1=30

s2=31

!CRIA UMA COLUNA ADICIONAL PARA ARMAZENAR O VETOR B

write(2,\*)s1,s2

allocate ( darray(s1,s2) )

do i=1,s1

do j=1,s2

if (i==j) then

darray(i,j)=2

elseif (i-j==1) then

darray(i,j)=0.7

elseif (j-i==1) then

darray(i,j)=0.7

end if

end do

darray(i,s2)=i

end do

call writeMatrix(darray,s1,s2)

deallocate (darray)

end program matriz

subroutine writeMatrix(array, n, m)

implicit none

real, intent(in) :: array(n,m)

integer, intent(in) :: n,m

integer :: i

do i = 1,n

write(2,\*) array(i,:) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i,:) !IMPRIME NA TELA

end do

end subroutine writeMatrix

# jacobi.f90

!gfortran jacobi.f90

!./a.out matriz

!esse programa encontra a solução do sistema e grava em um arquivo de nome “jacobi.txt”

!Tobias / IME 2022

program jacobi

implicit none

real\*8::s,e

real\*8, dimension (:,:), allocatable :: A

real\*8, dimension (:), allocatable:: x,xx

character(20)::argument

integer :: s1, s2,i,j,n

call get\_command\_argument(1, argument)

open(1,file=trim(argument)//".txt",status='old')

open(2,file="jacobi.txt",status='unknown')

read(1,\*)s1,s2

allocate ( A(s1,s2),x(s1),xx(s1))

do i=1,s1

x(i)=0

end do

n=0

e=5

do i=1,s1

read(1,\*) A(i, :)

end do

do while (n<30)

do i=1,s1

s=0

do j=1,s2-1

if (i/=j) then

s=s+A(i,j)\*x(j)

end if

end do

xx(i)=(A(i,s2)-s)/(A(i,i))

!print\*,"X",i,"=",xx(i),"erro =",e

end do

n=n+1

!print\*, "iteração",n

x=xx

end do

!print\*,"O programa terminou!"

call writevetor(x,s1)

deallocate (A)

end program jacobi

subroutine writeMatrix(array, n, m)

implicit none

real, intent(in) :: array(n,m)

integer, intent(in) :: n,m

integer :: i

do i = 1,n

write(2,\*) array(i,:) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i,:) !IMPRIME NA TELA

end do

end subroutine writeMatrix

subroutine writevetor(array, n)

implicit none

real\*8, intent(in) :: array(n)

integer, intent(in) :: n

integer :: i

do i = 1,n

write(2,\*) array(i) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i) !IMPRIME NA TELA

end do

end subroutine writevetor

# seidel.f90

!gfortran seidel.f90

!./a.out matriz

!esse programa encontra a solução do sistema e grava em um arquivo de nome “seidel.txt”

!Tobias / IME 2022

program seidel

implicit none

real\*8::s,e,x\_anterior

real\*8, dimension (:,:), allocatable :: A

real\*8, dimension (:), allocatable:: x

character(20)::argument

integer :: s1, s2,i,j,n

call get\_command\_argument(1, argument)

open(1,file=trim(argument)//".txt",status='old')

open(2,file="seidel.txt",status='unknown')

read(1,\*)s1,s2

allocate ( A(s1,s2),x(s1))

do i=1,s2-1

x(i)=0

end do

n=0

e=5

do i=1,s1

read(1,\*) A(i, :)

end do

do while (e>0.00000000000001)

do i=1,s1

s=0

do j=1,s2-1

if (i/=j) then

s=s+A(i,j)\*x(j)

end if

end do

x\_anterior=x(i)

x(i)=(A(i,s2)-s)/(A(i,i))

e=((x(i)-x\_anterior)\*\*2)\*\*0.5

!print\*,"X",i,"=",x(i),"erro =",e

end do

n=n+1

!print\*, "iteração",n

end do

!print\*,"O programa terminou!"

call writevetor(x,s1)

deallocate (A)

end program seidel

subroutine writeMatrix(array, n, m)

implicit none

real, intent(in) :: array(n,m)

integer, intent(in) :: n,m

integer :: i

do i = 1,n

write(2,\*) array(i,:) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i,:) !IMPRIME NA TELA

end do

end subroutine writeMatrix

subroutine writevetor(array, n)

implicit none

real\*8, intent(in) :: array(n)

integer, intent(in) :: n

integer :: i

do i = 1,n

write(2,\*) array(i) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i) !IMPRIME NA TELA

end do

end subroutine writevetor

# plu.f90

!gfortran plu.f90

!./a.out matriz

!esse programa encontra a solução do sistema e grava em um arquivo de nome “plu.txt”

!Tobias / IME 2022

program plu

implicit none

real\*8::e,pivo,mm

real, dimension (:,:), allocatable :: A,M

real\*8, dimension (:), allocatable:: x,b

character(20)::argument

integer ::s, s1, s2,i,j,n,c

call get\_command\_argument(1, argument)

open(1,file=trim(argument)//".txt",status='old')

open(2,file="plu.txt",status='unknown')

read(1,\*)s1,s2

allocate ( A(s1,s2),M(s1,s2),x(s1),b(s1))

e=5

!LÊ A MATRIZ

do i=1,s1

read(1,\*) A(i, :)

end do

!LÊ O VETOR B (ULTIMA COLUNA DO TXT)

do i=1,s1

b(i)=A(i,s2)

end do

c=1

do while (c<s2)

!print\*,"c=",c

do j=1,s2

do i=1,s1

if (i==j) then

pivo=A(i,j)

!print\*,"pivo","A[",i,"][",j,"]=",A(i,j)

elseif (i>j-1.AND.j==c) then ! # Condição para só acessar a parte triangular inferior

M(i,j)=A(i,j)/pivo

!print\*,"multiplicador M[",i,"][",j,"]=",M(i,j)

end if

end do

end do

!MULTIPLICA Linha

do i=1,s2

mm=M(i,c)

do j=c,s2

A(i,j)=A(i,j)-mm\*A(c,j)

!print\*,"A(",i,",",j,")=",A(i,j)

end do

end do

!call writeMatrix(A,s1,s2)

c=c+1 !Onde está o pivô

end do

!print\*,"A matriz L é"

!call writeMatrix(A,s1,s2-1)

!print\*,"A matriz U é"

!call writeMatrix(M,s1,s2-1)

! AQUI FALTA UM PROGRMA QUE RESOLVE SISTEMA TRIANGULAR INFERIOR

! O PROGRAMA ESTÁ MULTIPLICANDO O VETOR B JUNTAMENTE COM AS MATRIZES

do i=1,s1

b(i)=A(i,s2)

end do

!print\*,"O vetor b é",b

do i=1,s1

x(i)=0

end do

! RESOLVE SISTEMA TRIANGULAR SUPERIOR

x(s1)=b(s1)/A(s1,s1)

print\*,"i=",s1,"j=",s1,"x[",s1,"]",x(s1)

n=s1

s=0

do while (n > 1)

n=n-1

i=n

j=i+1

s=s+A(i,j)\*x(j)

x(i)=(b(i)-s)/A(i,i)

!print\*,"i=",i,"j=",j,"x[",i,"]",x(i)

j=j+1

end do

call writevetor(x,s1)

deallocate (A,M,x)

end program plu

subroutine writeMatrix(array, n, m)

implicit none

real, intent(in) :: array(n,m)

integer, intent(in) :: n,m

integer :: i

do i = 1,n

write(2,\*) array(i,:) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i,:) !IMPRIME NA TELA

end do

end subroutine writeMatrix

subroutine writevetor(array, n)

implicit none

real\*8, intent(in) :: array(n)

integer, intent(in) :: n

integer :: i

do i = 1,n

write(2,\*) array(i) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i) !IMPRIME NA TELA

end do

end subroutine writevetor