Trabalho 1 - Computação Científica e Análise de Dados (COCADA)

Professor: João Paixão

Aluno: Diego Vasconcelos Schardosim de Matos

DRE: 120098723

In [185…
```
using LinearAlgebra
```

1) Algoritmo para realizar fatoração LU.

A fatoração LU consiste em pegarmos uma matriz quadrada nxn e fatora-la como o produto de uma matriz triangular inferior (L -> Lower) e uma matriz triangular superior (U -> Upper).

Usaremos o método visto em aula

In [186…
```
function my_LU(A)
    n, n=size(A)
    L=zeros(n,n)
    U=zeros(n,n)

    for i=1:n
        pivô=A[i,i] # Os pivos são os elementos da diagonal principal

        linha_upper=A[i,:] # A linha inteira do pivo 'vai para' matriz U
        coluna_lower=(1/pivô)*A[:,i] # Após multiplicar a coluna do pivo por 1/p

        L[:,i]=coluna_lower
        U[i,:]=linha_upper

        # Ao realizar o produto matricial entre a linha e coluna obtida anterior
        # temos uma matriz A 'aproximada', logo, realizamos a diferença para enc
        # o 'resto', e continuamos a conta a partir desse resto
        resto = A - coluna_lower*linha_upper'
        A = resto
    end
    return L,U
end
```

Out[186]:
```
my_LU (generic function with 1 method)
```

In [187…
```
# Teste da LU
A=randn(100,100)
L,U=my_LU(A)
norm(A-L*U)<0.00001
```

Out[187]:
```
true
```

2) Algoritmo para realizar decomposição QR pelo método de Gram–Schmidt

Consiste em escrever uma matriz quadrada A_{nxm} por uma nova base Q composta de vetorores ortogonais e ortonormais

A ideia é, iterativamente, escrevermos q_1 como o vetor a_1 normalizado e, r_{11} como sendo a norma de a_1 ('quanto precisa andar em q_1 para chegar em a_1'), o q_2 como sendo a diferença de projeção de a_2 em q_1, normalizado, r_{12} o tamanho da projeção em q_1 ($q_2^T q_1$) e r_{12} a norma de q_2, assim por adiante

$$q_1 = \frac{a_1}{||a_1||}$$

Para i de 2 até nº colunas

$$q_i = a_i - \sum_{l=1}^{i-1} (a_i^T q_l) q_l$$

$$q_i = \frac{q_i}{||q_i||}$$

In [188…

```
function my_QR(A)
    n, m = size(A)
    Q = zeros(n, m)
    R = zeros(n, m)

    r_11 = norm(A[:,1])
    q_1 = A[:,1] / r_11

    Q[:,1] = q_1
    R[1,1] = r_11

    for i=2:m
        q_i = A[:,i]

        for l=1:i-1
            q_l = Q[:,l] # Vetor q_(i-1)
            projecao = A[:,i]' * q_l # Projecao parcial de a_i em q_(i-1)
            R[l,i] = projecao # Quanto a_i andou em q_l
            projecao = projecao * q_l # Projecao de a_i em q_(i-1)

            q_i = q_i - projecao
        end

        R[i,i] = norm(q_i)
        Q[:,i] = q_i / norm(q_i)
    end

    return Q,R
end
```

Out[188]: my_QR (generic function with 1 method)

In [189…
```
# Teste da QR

A=randn(4,4)
Q,R=my_QR(A)
norm(A-Q*R)<0.00001
```

Out[189]: true

Resolvendo um sistema linear por LU

Para resolvermos um sistema linear usando decomposição LU precisamos reescrever

$$Ax = b$$

$$A = LU$$

$$(LU)x = b$$

E resolver o novo sistema lienar

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

sabendo que U é triangular superior, sendo necessário o método de substituição reversa, já L é triangular inferior, sendo o método de substituição direta suficiente.

In [190…
```
"
    Recebe uma matriz triangular inferior nxn, b é um vetor nx1, e retorna matri
"

function substituicao_direta(L, b)
    n, n = size(L)
    y = zeros(n, 1)

    y[1] = b[1] / L[1,1] # Elemento mais acima é direto

    for linha_atual=2:n # A partir da segunda linha
        y_i = b[linha_atual] # q_i é igual ao valor que procuramos

        for coluna_atual=1:linha_atual-1
            y_i -= y[coluna_atual] * L[linha_atual, coluna_atual] # Diferença do
        end

        y_i = y_i / L[linha_atual, linha_atual] # Na posicao coluna = linha deve

        y[linha_atual] = y_i
    end

    return y
end
```

Out[190]: substituicao_direta (generic function with 1 method)

In [191…
```
"
    Recebe uma matriz triangular superior nxm, y é um vetor mx1, e retorna matri
"
```

```
function substituicao_reversa(U, y)
    n, m = size(U)
    x = zeros(n, 1)

    x[m] = y[m] / U[m,m] # Último elemento é direto

    for linha_atual=n-1:-1:1 # Iteração reversa de n-1 até 1
        x_i = y[linha_atual] # q_i é igual ao valor que procuramos

        for coluna_atual=linha_atual+1:m # Começamos pelo elemento ao lado direi
            x_i -= x[coluna_atual] * U[linha_atual, coluna_atual]
        end

        x_i = x_i / U[linha_atual,linha_atual] # Elemento Uii que foi pulado é a
        x[linha_atual] = x_i
    end

    return x
end
```

Out[191]: substituicao_reversa (generic function with 1 method)

In [192…
```
function resolver_sistema_linear_LU(A, b)
    L, U = my_LU(A)

    y = substituicao_direta(L, b)
    x = substituicao_reversa(U, y)

    return x
end
```

Out[192]: resolver_sistema_linear_LU (generic function with 1 method)

In [193…
```
"
    Mesma matriz do exercicio passado em aula
"
A = [
    2 1 1
    4 3 3
    6 7 10
]
b = [
    5
    13
    33
]

x = resolver_sistema_linear_LU(A, b)
```

Out[193]:
```
3×1 Matrix{Float64}:
 1.0
 1.0
 2.0
```

In [194…
```
# teste
norm(b - A*x)
```

Out[194]: 0.0

Mínimos quadrados por qr

A solução por minimos quadrados é feito substituinod A = QR e resolvendo o novo sistema linear

$$R\tilde{x} = Q^T b$$

$$\begin{cases} R\tilde{x} = y \\ y = Q^T b \end{cases}$$

sabendo que R é triangular superior, sendo o método de substituição reversa suficiente a primeira equação

In [195…
```
function minimos_quadrados_por_qr(A, b)
    Q, R = my_QR(A)

    y = Q' * b
    x = substituicao_reversa(R, y)

    # erro = norm(b - R*x)
    return x
end
```

Out[195]:
```
minimos_quadrados_por_qr (generic function with 1 method)
```

In [196…
```
A = [
    1 0
    0 2
    0 3
]

b = [
    1
    1
    1
]

x = minimos_quadrados_por_qr(A, b)
```

Out[196]:
```
3×1 Matrix{Float64}:
 1.0
 0.3846153846153847
 0.0
```