MAC0300 - Métodos Numéricos para Álgebra Linear - 2015/S2

António Martins Miranda (Nº 7644342) {*amartmiranda@gmail.com*}
António Rui Castro Júnior (Nº 5984327) {*antonio.castro@usp.br*}

EP1 - Resolução de Sistemas de Equações Lineares - Relatório

1 Sistemas definidos positivos

Para a primeira parte do EP implementamos, de acordo com a exigências do enunciado e na linguagem de programação C, as seguintes funções:

- **int cholcol (int n, double A[][nmax])** - implementa a decomposição de Cholesky orientada a colunas;
- **int cholrow (int n, double A[][nmax])** - implementa a decomposição de Cholesky orientada a linhas;
- **int forwcol (int n, double A[][nmax], double b[])** - resolve um sistema do tipo $Ax = b$, orientado a colunas e usando substituição para frente;
- **int forwrow (int n, double A[][nmax], double b[])** - resolve um sistema do tipo $Ax = b$, orientado a linhas e usando substituição para frente;
- **int backcol (int n, double A[][nmax], double b[], int trans)** - resolve sistemas do tipo $Ax = b$ ou $A^T x = b$, orientado a colunas e usando a substituição para trás;
- **int backrow (int n, double A[][nmax], double b[], int trans)** - resolve sistemas do tipo $Ax = b$ ou $A^T x = b$, orientado a linhas e usando a substituição para trás.

As funções cujo nome terminam com *col* são orientadas a coluna e as que terminam com *row* são orientadas a linha. Para mais detalhes sobre as funções, consultar a primeira parte do enunciado do ep1.

1.1 Decomposição de Cholesky

Pseudocódigo da função *cholcol*:

```
for j=1 to n
        for k=1 to j-1
                for i=j to n /*conhecido por cmod(j,k)*/
                        a_ij = a_ij - a_ik * a_jk
                end
        end
        if a_jj <= 0
                return -1
        end
        a_jj = sqrt(a_jj) /*raiz quadrada de a_jj*/
        for k=j+1 to n /*conhecido por cdiv(j)*/
                a_kj = a_kj / a_jj
        end
end
return 0
```

Pseudocódigo da função *cholrow*:

```
for i=1 to n
        for j=1 to i-1
                for k=1 to j-1
                        a_ij = a_ij - a_ik * a_jk
                end
                a_ij = aij / a_jj
        end
        for j=1 to i-1
                a_ii = a_ii - a_ij * a_ij
        end
        if a_ii <= 0
                return -1
        end
        a_ii = sqrt(a_ii) /* raiz quadrada de a_ii */
end
return 0
```

1.2 Resolução de um sistema $Ax = b$ com substituição para frente

Pseudocódigo da função *forwcol*:

```
for j=1 to n
        if a_jj == 0
                return -1
        end
        b_j = b_j / a_jj
        for i=j+1 to n
                b_i = b_i - a_ij * b_j
        end
end
return 0
```

Pseudocódigo da função *forwrow*:

```
for i=1 to n
        for j=1 to i-1
                b_i = b_i - a_ij * b_j
        end
        if a_ii == 0
                return -1
        end
        b_i = b_i / a_ii
end
return 0
```

1.3 Resolução de um sistema $Ax = b$ (ou $A^T x = b$) com substituição para trás

Pseudocódigo da função *backcol*:

```
if trans == 1
        for i=n to 1 /*decremento*/
                for j=i+1 to n
                        b_i=b_i-a_ji*b_j
                end
                if a_ii == 0
                        return -1
                end
                b_i=b_i/a_ii
```

```
        end
else
        for j=n to 1 /*decremento*/
                if a_jj == 0
                        return -1
                end
                b_j=b_j/a_jj
                for i=j-1 to 1 /*decremento*/
                        b_i=b_i-a_ij*b_j
                end
        end
end
return 0
```

Pseudocódigo da função *backrow*:

```
if trans == 1
        for j=n to 1 /*decremento*/
                if a_jj == 0
                        return -1
                end
                b_j=b_j/a_jj
                for i=j-1 to 1 /*decremento*/
                        b_i=b_i-a_ji*b_j
                end
        end
else
        for i=n to 1 /*decremento*/
                for j=i+1 to n
                        b_i=b_i-a_ij*b_j
                end
                if a_ii == 0
                        return -1
                end
                b_i=b_i/a_ii
        end
end
return 0
```

1.4 Tempos de execução da Decomposição de Cholesky

PROBLEMA	DECOMPOSIÇÃO DE CHOLESKY					
	ORIENTADO A LINHA			ORIENTADO A COLUNA		
	$A = GG^T$	$Gy = b$	$G^Tx = y$	$A = GG^T$	$Gy = b$	$G^Tx = y$
1	0.000250	0.000008	0.000009	0.000324	0.000012	0.000010
2	0.001710	0.000030	0.000026	0.002558	0.000065	0.000029
3	0.005273	0.000052	0.000056	0.008853	0.000093	0.000061
4	0.012468	0.000091	0.000098	0.026486	0.000198	0.000141
5	0.025155	0.000146	0.000159	0.078277	0.000555	0.000417
6	0.043325	0.000210	0.000228	0.175645	0.000910	0.000690
7	0.069355	0.000299	0.000333	0.321707	0.001727	0.001053

Comentários:

Como usamos a linguagem C para programar as funções e nela as matrizes são armazenadas em memória por linha (e em sequência), era esperado que as funções orientadas a linha fossem mais eficiêntes que as funções orientadas a coluna, como podemos verificar pelos resultados da tabela. Calculamos os tempos no *cygwin*, instalado numa máquina (64 bits) com 8 Gb de memória RAM e um processador Intel Core 2 Duo de 2.80 GHz.

2 Sistemas Gerais

De acordo com o enunciado e usando e também usando a linguagem de programação C, implementamos para essa segunda parte as seguintes funções:

- **int lucol (int n, double A[][nmax], int p[])** - implementa a decomposição LU orientada a colunas;
- **int lurow (int n, double A[][nmax], int p[])** - implementa a decomposição LU orientada a linhas;
- **int sscol (int n, double A[][nmax], int p[], double b[])** - resolve um sistema do tipo $LUx = Pb$, orientado a colunas;
- **int ssrow (int n, double A[][nmax], int p[], double b[])** - resolve um sistema do tipo $LUx = Pb$, orientado a linhas.

Como na primeira parte, as funções cujo nome terminam com *col* são orientadas a coluna e as que terminam com *row* são orientadas a linha. Para mais detalhes sobre as funções, consultar a segunda parte do enunciado do ep1.

2.1 Decomposição LU

Pseudocódigo da função *lucol*:

```
for k=1 to n-1
        imax=k
        for i=k+1 to n
                if (|a_ik| > |a_imax,k|)
                        imax=i
                end
        end
        p(k)=imax /*p(k) vetor permutacao*/
        if p(k)!=k
                for j=1 to n
                        tmp=a_kj
                        a_kj=a_p(k),j
                        a_p(k),j=tmp
                end
        end
        if a_kk == 0
                return -1
        end
        for i=k+1 to n
                a_ik=a_ik / a_kk
        end
        for j=k+1 to n
                for i=k+1 to n
                        a_ij=a_ij-a_kj*a_ik
                end
        end
        if a_nn == 0
                return -1
        end
end
return 0
```

Pseudocódigo da função *lurow*:

```
for k=1 to n-1
        imax=k
        for i=k+1 to n
                if (|a_ik| > |a_imax,k|)
```

```
                imax=i
            end
    end
    p(k)=imax /*p(k) vetor permutacao*/
    if p(k)!=k
            for j=1 to n
                    tmp=a_kj
                    a_kj=a_p(k),j
                    a_p(k),j=tmp
            end
    end
    if a_kk == 0
            return -1
    end
    for i=k+1 to n
            a_ik=a_ik / a_kk
            for j=k+1 to n
                    a_ij=a_ij-a_kj*a_ik
            end
    end
    if a_nn == 0
            return -1
    end
end
return 0
```

2.2 Resolução de um sistema $LUx = Pb$

Pseudocódigo da função *sscol*:

```
for i=1 to n-1
        tmp=b_i
        b_i=b_p(i) /*p(i) vetor permutacao*/
        b_p(i)=tmp
end
for j=1 to n
        for i=j+1 to n
                b_i=b_i-a_ij*b_j
        end
end
for j=n to 1 /*decremento*/
        if a_jj == 0
                return -1
        end
        b_j=b_j/a_jj
        for i=j-1 to 1 /*decremento*/
                b_i=b_i-a_ij*b_j
        end
end
return 0
```

Pseudocódigo da função *ssrow*:

```
for i=1 to n-1
        tmp=b_i
        b_i=b_p(i) /*p(i) vetor permutacao*/
        b_p(i)=tmp
end
for i=1 to n
        for j=1 to i-1
```

```
            b_i=b_i-a_ij*b_j
        end
end
for i=n to 1 /*decremento*/
        if a_ii == 0
            return -1
        end
        for j=i+1 to n
            b_i=b_i-a_ij*b_j
        end
        b_i=b_i/a_ii
end
return 0
```

2.3 Tempos de execução da Decomposição LU

PROBLEMA	DECOMPOSIÇÃO LU			
	ORIENTADO A LINHA		ORIENTADO A COLUNA	
	$PA = LU$	$LUx = Pb$	$PA = LU$	$LUx = Pb$
1	0.000478	0.000020	0.000648	0.000023
2	0.003653	0.000072	0.004695	0.000077
3	0.011509	0.000168	0.024716	0.000188
4	0.027829	0.000418	0.087417	0.000441
5	0.055414	0.001359	0.229434	0.001574
6	0.093878	0.002135	0.449888	0.002238
7	0.152274	0.003061	0.781473	0.003559

Comentários:

Como podemos verificar, as funções orientadas a linha são mais eficiêntes que as funções orientadas a coluna, visto que C é uma linguagem que armazena matrizes na memória por linha (e em sequência). Para esta parte também calculamos os tempos no *cygwin*, instalado numa máquina (64 bits) com 8 Gb de memória RAM e um processador Intel Core 2 Duo de 2.80 GHz.

Pelo pseudocódigo da decomposição de Cholesky verificamos que a ela é constituída por uma operação de custo $O(\frac{n^2}{2})$ (*cdiv*) e outra de custo $O(n^3)$ (*cmod*), enquanto que a decomposição LU é constituída por uma operação de custo $O(n^2)$ (permutação da linha que contém o elemento máximo em módulo com a linha do pivot), uma operação de custo $O(\frac{n^2}{2})$ (escolha do elemento máximo em módulo na coluna do pivot e abiaxo dele) e uma operação de custo $O(n^3)$ (zerar os elementos abaixo do pivot usando operações elementares e armazenar essas operações nas posições zeradas). Logo, é natural que a decomposição LU seja mais custosa e aproximadamente o dobro da decomposição de Cholesky($2 \times O(n^3 + \frac{n^2}{2}) \approx O(n^3 + n^2) \approx O(n^3 + \frac{3n^2}{2})$).