## Name: Łukasz Skrzyński, Vickers Zhu 15 April 2019

## Group: FA2 Computer Science MINI

Solving linear systems of equations

with block Cholesky method

Project 1/2019

**1. Task:**

The goal of the project was creating an algorithm to solve linear systems of equations where is positive definite square matrix of *n* dimensions and :

=

where , are square matrices of *p* dimensions and *n* = 3*p.*

The system had to be solved with Cholesky method applied to block matrices which were part of matrix *A.* Proper program and examples of its use had to be prepared using MATLAB.

**2.Theory:**

Cholesky method is a method that turns the positive definite matrix *A* from the linear system of equations into multiple of another matrix and its transpose .

Matrix is the lower triangular matrix and matrix is upper triangular matrix. This allows us to turn linear system into two separate systems, which are easier to solve:

Now we introduce another variable. The results are two linear equations:

As both and are triangular, solving those equations is very simple.

Block method works exactly like regular method, but instead of operating on values inside matrix it operates on smaller matrices ,which are parts of the main matrix. These smaller matrices are called blocks. In our case:

\* = =

After multiplication it turns out that is always 0, therefore:

\* = =

= =

Having those equations we can easily obtain all elements of matrix .

**3.Code:**

Main function BlockChol.m:

function [r] = BlockChol(A,b)

%Block Cholesky function for specific matrix

% Detailed explanation goes here

if ~ismatrix(A)

error("A should be a matrix")

end

if ~isvector(b)

error("b should be a vector")

end

if rem(length(A),3) ~= 0

error("Matrix should have proper number of rows")

end

if size(A,1) ~= size(b,1)

error("Matrix and vector size is not equal")

end

%since Matlab cannot make matrices of matrices

% A(1,1) = B, A(2,2) = C, A(3,3) = D

% L(1,1) = E, L(2,1) = F, L(2,2) = G

% L(3,2) = H, L(3,3) = J

% I - identity matrix p\*p

n = size(A,1);

p = n/3;

B=A(1:p,1:p);

C=A(p+1:2\*p,p+1:2\*p);

D=A((2\*p)+1:3\*p,(2\*p)+1:3\*p);

I = eye(p);

E = CholDec(B);

F = inv(transpose(E)) \* I;

S = C - (F \* transpose(F));

G = CholDec(S);

H = inv(transpose(G)) \* I;

S = D - (F \* transpose(F));

J = CholDec(S);

%at this point we have our block Cholesky matrix

%[E 0 0] [E' F' 0 ]

%[F G 0] = L [0 G' H'] = L'

%[0 H J] [0 0 J']

c = b(1:p);

d = b(p+1:2\*p);

e = b((2\*p)+1:3\*p);

% [c] [x] [f]

% b = [d] x = [y] m = [g]

% [e] [z] [h]

% A \* x = b

% A = L \* L'

% L \* L' \* x = b

% L' \* x = m

% L \* m = b

f = E\c;

g = inv(G) \* (d-(F\*f));

h = inv(J) \* (e - (H\*g));

z = transpose(J)\h;

y = inv(transpose(G)) \* (g -(transpose(H)\*z));

x = inv(transpose(E)) \* (f - (transpose(F)\*y));

r=[x;y;z];

disp(r);

o = linsolve(A,b);

disp(o);

%function displays results of BlockChol as well as

%linsolve for simple comparison

end

Supporting function CholDec.m:

function [L] = CholDec(A)

%Cholesky lower decomposition

% Returns lower triangular matrix from Cholesky decomposition

if ~ismatrix(A)

error("Input must be a square matrix")

end

if size(A,1) ~= size(A,2)

error("Input must be a square matrix")

end

n = length(A);

e = eig(A);

for i=1:n

if e(i)< 0

error("Input must be a positive definite matrix")

end

end

L = zeros(n);

for i=1:n

L(i, i) = sqrt(A(i, i) - L(i, :)\*transpose(L(i, :)));

for j=(i + 1):n

L(j, i) = (A(j, i) - L(i,:)\*transpose(L(j ,:)))/L(i, i);

end

end

end

**4.Example:**

Randomly generated matrix:

A =

89.9217 -2.9227 1.0000 0 0 0

-2.9227 83.0783 0 1.0000 0 0

1.0000 0 46.9211 -12.3637 1.0000 0

0 1.0000 -12.3637 36.0789 0 1.0000

0 0 1.0000 0 71.8663 23.4915

0 0 0 1.0000 23.4915 73.1337

>> b = [5;7;22;3;4;1]

b =

5

7

22

3

4

1

>> BlockChol(A,b);

0.0523

0.0829

0.5365 resu

0.2649

0.0502

-0.0061

0.0523

0.0829

0.5365

0.2649

0.0502

-0.0061

Generated matrix with diagonal blocks being Pascal matrices:

A =

6 6 1 0 0 0

6 12 0 1 0 0

1 0 6 6 1 0

0 1 6 12 0 1

0 0 1 0 6 6

0 0 0 1 6 12

>> b = [1;2;3;4;5;6]

b =

1

2

3

4

5

6

>> BlockChol(A,b);

-0.0294

0.1640

0.1926

0.2088

0.6206

0.1742

-0.0243

0.1608

0.1809

0.2161

0.6424

0.1608

Matrices like Hilbert matrix are not accepted, because they do not fulfill requirements of the task.