HOMEWORK

Jun Cheng

October 31, 2015

Problem 0: Homework checklist

✓I didn't talk with any one about this homework.
✓Source-code are included at the end of this document.

Problem 1: Direct Methods for Tridiagnonal Systems

1.

$$\boldsymbol{A} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

$$\boldsymbol{A}_{n-1} = \boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1}^T$$

Then matrix $\boldsymbol{L}_{n-1}$ must have the shape like

$$\boldsymbol{L}_{n-1} = \begin{bmatrix} X_{11} & 0 & & & \\ X_{12} & X_{22} & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & X_{n-2,n-2} & 0 \\ & & & X_{n-1,n-2} & X_{n-1,n-1} \end{bmatrix}$$

2.

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{L}_n\boldsymbol{L}_n^T \\ &= \begin{bmatrix} \boldsymbol{L}_{n-1}\boldsymbol{L}_{n-1} & \beta_{n-1} \\ \beta_{n-1} & \alpha_n \end{bmatrix} \\ &= \boldsymbol{L}_n\boldsymbol{L}_n^T \\ \boldsymbol{L}_n &= \begin{bmatrix} \boldsymbol{L}_{n-1} & 0 \\ b & l \end{bmatrix} \\ \boldsymbol{L}_n^T &= \begin{bmatrix} \boldsymbol{L}_{n-1}^T & b^T \\ 0 & l \end{bmatrix} \end{aligned}$$

Then we need to figure out b and l

$$b\boldsymbol{L}_{n-1}^T = \beta_{n-1} \tag{1}$$

$$\|b\|^2 + l^2 = \alpha_n^2 \tag{2}$$

Because b is a vector with only one non zero entry and $\boldsymbol{L}_{n-1}$ is a lower triangle matrix with one diagonal elements and one below diagonal. So both equation 1 and 2 can be solved in constant number of operations.

3. From part(1) and part(2), we can get,

$$\boldsymbol{L}_n = \begin{bmatrix} \sqrt{\delta_1} & 0 & & & \\ \frac{\beta_1}{\sqrt{\delta_1}} & \sqrt{\delta_2} & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{\beta_{n-2}}{\sqrt{\delta_{n-2}}} & \sqrt{\delta_{n-1}} & 0 \\ & & & \frac{\beta_{n-1}}{\sqrt{\delta_{n-1}}} & \sqrt{\delta_n} \end{bmatrix}$$

where

$$\delta_1 = \alpha_1$$

$$\delta_j = \alpha_j - \frac{\beta_{j-1}}{\delta_{j-1}}, j = 2, ...k,$$

Pseudocode:
1: $\delta_1 = \alpha_1$
2: for i=2:n
$\delta_j = \alpha_j - \frac{\beta_{j-1}}{\delta_{j-1}}$

There is only one loop with size n, so the complexity is $0(n)$.

Problem 1: Sparse Matrices in Matlab

1. Code:

```
N=4;
A = sparse((N-1)*(N-1),(N-1)*(N-1));

for i=1:(N-1)*(N-1)
        A(i,i) = -4;

        col = mod(i,N-1)
        if col==0
            col = N-1;
        end
        row = ceil(i/(N-1))
        if row-1≥1
            A(i, (row-2)*(N-1) + col) = 1; % UP
        end
        if row+1 ≤N-1
            A(i, (row)*(N-1) + col) = 1; % DOWN
        end
        if col-1≥1
            A(i, (row-1)*(N-1) + col-1) =1 ; % LEFT
        end
        if col+1≤N-1
            A(i, (row-1)*(N-1) + col+1) = 1; % RIGHT
        end
end
```

2. Code:

```
N=4;
nz = (N-1)^2 + 4*(N-1)*(N-2);
I = zeros(nz,1);
J = zeros(nz,1);
V = zeros(nz,1);
```

```

index = 1;
for i=1:(N-1)*(N-1)
         I(index) = i;
         J(index) = i;
         V(index) = -4;
         index = index+1;

        col = mod(i,N-1);
        if col==0
            col = N-1;
        end
        row = ceil(i/(N-1));
        if row-1≥1
            % UP
            I(index) = i;
            J(index) = (row-2)*(N-1) + col;
            V(index) = 1;
            index = index+1;
        end
        if row+1 ≤N-1
             % DOWN
            I(index) = i;
            J(index) = (row)*(N-1) + col;
            V(index) = 1;
             index = index+1;
        end
        if col-1≥1
             % LEFT
            I(index) = i;
            J(index) = (row-1)*(N-1) + col-1;
            V(index) = 1;
             index = index+1;
        end
        if col+1≤N-1
            % RIGHT
            I(index) = i;
            J(index) = (row-1)*(N-1) + col+1;
            V(index) = 1;
            index = index+1;
        end
end

A = sparse(I,J,V,(N-1)*(N-1),(N-1)*(N-1));
```

Table 1: My caption

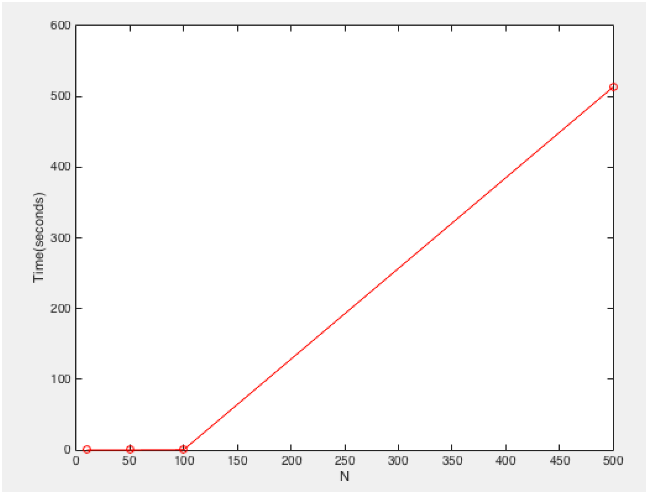
N	Method_1	Method2
10	0.0086	0.0004
50	0.0438	0.0022
100	0.3845	0.0060
500	513.0644	0.1364

3. The method in part 2 is faster. Figures are shown below.

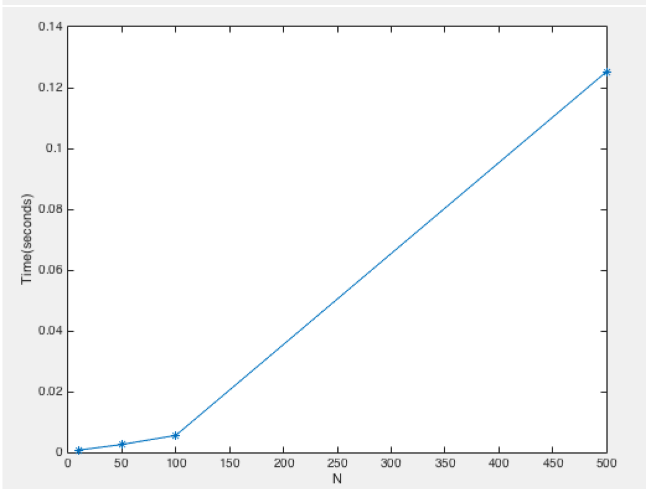
4. When using the backslash in Matlab, sparse matrix representation would be faster. The results are shown.

5. Code:

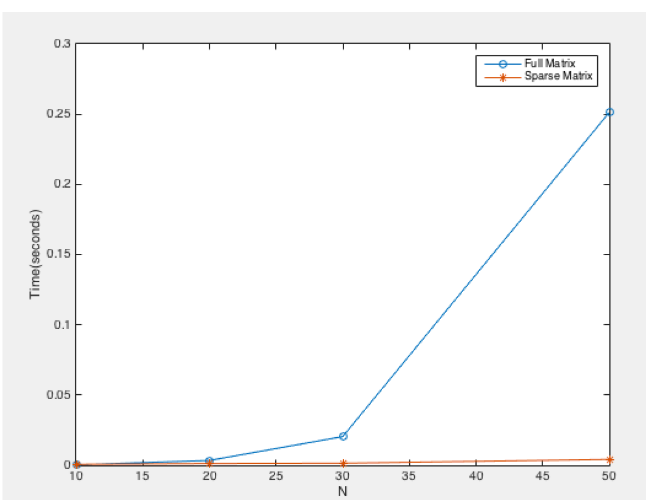
```
function [U, iter] = jacobian(A, b)
% Assume A is a sparse matrix
[N,t] = size(b);
U = zeros(N,1) ;
```



600
500
400
300
200
100
0
Time(seconds)
0
50
100
150
200
250
300
350
400
450
500
N



0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
Time(seconds)
0
50
100
150
200
250
300
350
400
450
500
N



Full Matrix
Sparse Matrix
0.3
0.25
0.2
0.15
0.1
0.05
0
Time(seconds)
10
15
20
25
30
35
40
45
50
N

```
%diff = zeros(20, 1) ;
[J,I,V] = find(A);
res = 100;
iter = 0 ;
nb = norm(b);
[len,t]= size(I);
while res > 1.0e-4
    temp = U;
    ss = 4*temp;
    for index=1:len
        i = I(index);
        j = J(index);
        ss(i) = ss(i) + A(i,j)* temp(j);
    end
    for i=1:N
        U(i) = 1.0/A(i,i)*(b(i)-ss(i));
    end
    res  =  norm(b-A*U)/nb
    iter = iter + 1;
end
end
```

Table 2: Problem2.5

N	Iteration
10	182
50	4567
100	18256
500	**

6. The results are shown in the following tables for different right side vectors. $**$ means it took significant long time without results.

Table 3: $f(x, y) = 1$

N	Iteration
10	182
50	4567
100	18256
500	**

Table 4: $f(x, y) = -1$

N	Iteration
10	182
50	4567
100	18256
500	**

Table 5: $f(x, y) = -(x - 0.5)^2 - (y - 0.5)^2$

N	Iteration
10	177
50	4427
100	**
500	**

Table 6: $f(x, y) = sin(100x)cos(100y)$

N	Iteration
10	182
50	4567
100	18256
500	**