Department of Physics
Umeå University

November 3, 2018

Matrix computations and applications
Assignment 5

Gustav Nystedt - oi14gnt

Department of computer science
Supervisors: Carl Christian Kjelgaard Mikkelsen

Contents

1 Introduction

In this assignment we investigate the behaviour of different types of methods such as Direct solvers, Jacobi's method and preconditioned conjugate convergence method, using Cholesky factorization for defining preconditioners. We look at the convergence properties of the residuals as well as the time dependence with respect to different accuracy conditions. Lastly we use what we have discovered in early exercises to construct an efficient solver for the heat equation in a two-dimensional plate.

2 Problems

2.1 Direct solvers

Table 1: Table showing variables for increasing N

N	M^2	nnz(A)	nnz(L)	time	timeSolve
10	81	369	737	1.06e-02	4.32e-04
20	361	1729	6877	3.49e-04	4.31e-05
30	841	4089	24417	1.42e-03	6.44e-05
40	1521	7449	59357	3.92e-03	2.18e-04
50	2401	11809	117697	3.28e-02	4.75e-04
60	3481	17169	205437	1.16e-02	3.89e-04
70	4761	23529	328577	1.95e-02	9.77e-04
80	6241	30889	493117	6.36e-02	2.83e-03
90	7921	39249	705057	4.20e-02	1.50e-03
100	9801	48609	970397	5.99e-02	2.07e-03
110	11881	58969	1295137	8.99e-02	6.08e-03
120	14161	70329	1685277	1.23e-01	3.86e-03
130	16641	82689	2146817	1.28e-01	4.82e-03
140	19321	96049	2685757	1.45e-01	5.99e-03
150	22201	110409	3308097	1.91e-01	7.76e-03
160	25281	125769	4019837	2.44e-01	9.83e-03
170	28561	142129	4826977	3.07e-01	1.16e-02
180	32041	159489	5735517	3.40e-01	1.75e-02
190	35721	177849	6751457	4.30e-01	2.27e-02
200	39601	197209	7880797	4.85e-01	2.19e-02
210	43681	217569	9129537	5.76e-01	3.09e-02
220	47961	238929	10503677	6.54e-01	2.74e-02
230	52441	261289	12009217	6.80e-01	3.01e-02
240	57121	284649	13652157	7.78e-01	3.69e-02
250	62001	309009	15438497	8.82e-01	4.63e-02
260	67081	334369	17374237	1.03e+00	3.93e-02
270	72361	360729	19465377	1.19e+00	4.39e-02
280	77841	388089	21717917	1.20e+00	4.21e-02
290	83521	416449	24137857	1.69e+00	5.81e-02
300	89401	445809	26731197	1.99e+00	5.75e-02
310	95481	476169	29503937	1.79e+00	6.31e-02
320	101761	507529	32462077	1.92e+00	7.23e-02
330	108241	539889	35611617	2.20e+00	1.73e-01
340	114921	573249	38958557	2.39e+00	8.13e-02
350	121801	607609	42508897	2.70e+00	1.91e-01
360	128881	642969	46268637	2.97e+00	2.89e-01
370	136161	679329	50243777	3.32e+00	2.94e-01
380	143641	716689	54440317	3.57e+00	3.10e-01
390	151321	755049	58864257	3.93e+00	3.51e-01
400	159201	794409	63521597	4.52e+00	3.39e-01

The N at which the nnz(L) exceeds nnz(A) by a factor of 10 is N = 60, and the N at which the factorization time exceeds 1 second is N = 260. It is clearly visible in Table 1 that the time for computing the Cholesky factorization is increasing dramatically for increasing N.

2.2 Jacobi's method

In the first task of this part, we are asked to plot the residual history for a solver that uses Jacobi's method. This is shown in Figure 1 below. It is clear that the method converges.

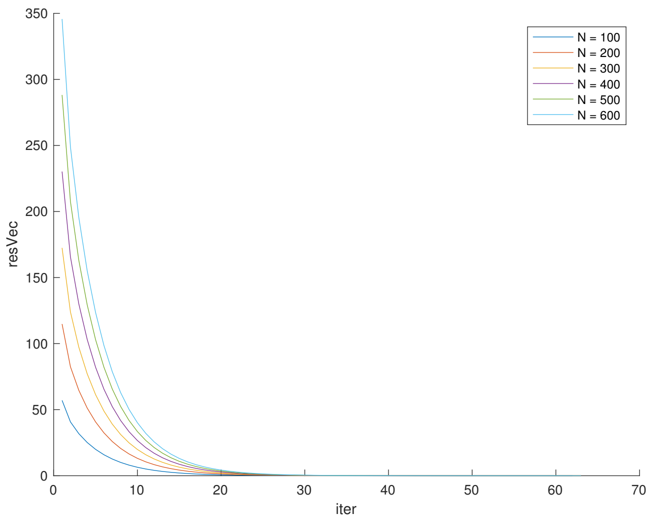

Figure 1: Residual history for Jacobi's method.

To convince ourselves that the method converges linearly however, we plot

$$\frac{\|res_{i+1}\|}{\|res_i\|},$$

shown in Figure 2, and can then conclude that this converges towards some finite value. This implies that the method converges linearly.

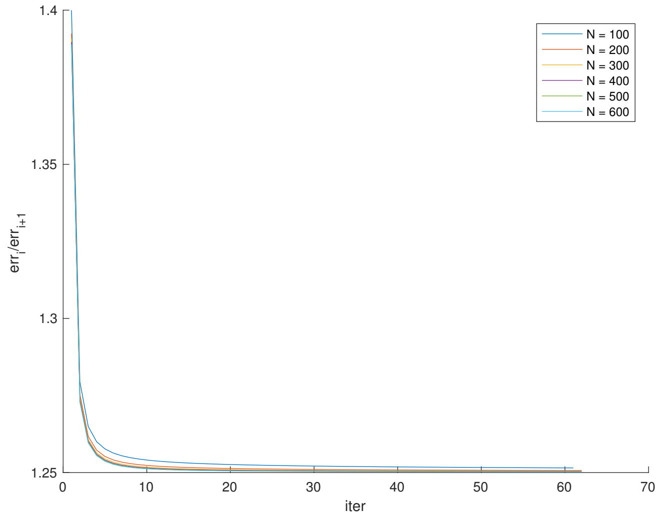

Figure 2: Shows that the method converges linearly, since the quote plotted converges towards a finite value.

Further, to show that the method is independent of N, we plot the residual history divided by the size of the system. This is shown in Figure 3 below.

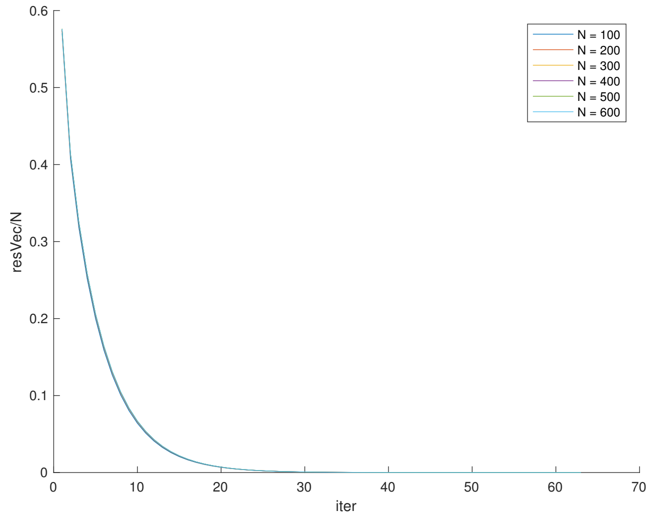

Figure 3: Shows residuals independence of N.

2.3 CG-method with no preconditioner

When testing the conjugate gradient method without preconditioner, we can conclude that it needs more than one second to converge when the size of the system corresponds to $\mathrm{N} = 1440$. The code for this test is given in the appendix and is named `testCG.m`.

3 CG-algorithm with preconditioning

In this exercise we test how CG algorithms depends on their preconditioners, and start of by varying N, λ,τ and tol. The tables given in Section 3.1 below was generated.

3.1 Tables

Table 2: $\tau = 0.1000$, $\lambda = 1$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	2.92e+04	1.85e-03	0	1.61e-02	1.79e-02
200	3.96e+04	1.97e+05	1.18e+05	3.07e-03	0	1.88e-02	2.19e-02
300	8.94e+04	4.46e+05	2.68e+05	7.09e-03	0	4.37e-02	5.08e-02
400	1.59e+05	7.94e+05	4.77e+05	1.37e-02	0	7.59e-02	8.96e-02
500	2.49e+05	1.24e+06	7.46e+05	3.30e-02	0	1.33e-01	1.66e-01
600	3.59e+05	1.79e+06	1.08e+06	3.41e-02	0	1.90e-01	2.24e-01

Table 3: $\tau = 0.1000$, $\lambda = 10$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	2.92e+04	2.30e-03	0	1.64e-02	1.87e-02
200	3.96e+04	1.97e+05	1.18e+05	2.73e-03	0	4.71e-02	4.99e-02
300	8.94e+04	4.46e+05	2.68e+05	7.36e-03	0	1.12e-01	1.19e-01
400	1.59e+05	7.94e+05	4.77e+05	1.31e-02	0	1.81e-01	1.95e-01
500	2.49e+05	1.24e+06	7.46e+05	2.15e-02	0	3.16e-01	3.38e-01
600	3.59e+05	1.79e+06	1.08e+06	3.11e-02	0	4.12e-01	4.43e-01

Table 4: $\tau = 0.1000$, $\lambda = 100$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	2.92e+04	2.26e-03	0	5.34e-02	5.57e-02
200	3.96e+04	1.97e+05	1.18e+05	2.81e-03	0	1.36e-01	1.39e-01
300	8.94e+04	4.46e+05	2.68e+05	7.69e-03	0	3.88e-01	3.96e-01
400	1.59e+05	7.94e+05	4.77e+05	1.39e-02	0	5.84e-01	5.97e-01
500	2.49e+05	1.24e+06	7.46e+05	2.39e-02	0	9.79e-01	1.00e+00
600	3.59e+05	1.79e+06	1.08e+06	3.53e-02	0	1.45e+00	1.48e+00

Table 5: $\tau = 0.0100$, $\lambda = 1$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	3.88e+04	4.19e-03	0	2.13e-02	2.55e-02
200	3.96e+04	1.97e+05	1.58e+05	3.11e-03	0	1.76e-02	2.07e-02
300	8.94e+04	4.46e+05	3.56e+05	9.24e-03	0	3.69e-02	4.62e-02
400	1.59e+05	7.94e+05	6.35e+05	1.80e-02	0	6.47e-02	8.28e-02
500	2.49e+05	1.24e+06	9.94e+05	2.72e-02	0	9.96e-02	1.27e-01
600	3.59e+05	1.79e+06	1.43e+06	4.10e-02	0	1.43e-01	1.84e-01

Table 6: $\tau = 0.0100$, $\lambda = 10$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	4.83e+04	3.87e-03	0	3.71e-02	4.09e-02
200	3.96e+04	1.97e+05	1.97e+05	3.95e-03	0	3.08e-02	3.48e-02
300	8.94e+04	4.46e+05	4.45e+05	1.15e-02	0	7.04e-02	8.19e-02
400	1.59e+05	7.94e+05	7.93e+05	2.09e-02	0	1.21e-01	1.42e-01
500	2.49e+05	1.24e+06	1.24e+06	3.21e-02	0	1.90e-01	2.22e-01
600	3.59e+05	1.79e+06	1.79e+06	4.73e-02	0	2.75e-01	3.22e-01

Table 7: $\tau = 0.0100$, $\lambda = 100$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	4.83e+04	2.23e-03	0	4.70e-02	4.93e-02
200	3.96e+04	1.97e+05	1.97e+05	3.85e-03	0	7.98e-02	8.37e-02
300	8.94e+04	4.46e+05	4.45e+05	1.08e-02	0	1.85e-01	1.96e-01
400	1.59e+05	7.94e+05	7.93e+05	2.12e-02	0	3.12e-01	3.34e-01
500	2.49e+05	1.24e+06	1.24e+06	3.20e-02	0	5.03e-01	5.35e-01
600	3.59e+05	1.79e+06	1.79e+06	4.70e-02	0	7.21e-01	7.68e-01

Table 8: $\tau = 0.0010$, $\lambda = 1$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	6.72e+04	7.76e-03	0	1.19e-02	1.97e-02
200	3.96e+04	1.97e+05	2.74e+05	1.26e-02	0	1.57e-02	2.83e-02
300	8.94e+04	4.46e+05	6.22e+05	2.10e-02	0	3.24e-02	5.34e-02
400	1.59e+05	7.94e+05	1.11e+06	3.84e-02	0	5.97e-02	9.81e-02
500	2.49e+05	1.24e+06	1.74e+06	6.16e-02	0	8.77e-02	1.49e-01
600	3.59e+05	1.79e+06	2.50e+06	9.36e-02	0	1.30e-01	2.24e-01

Table 9: $\tau = 0.0010$, $\lambda = 10$

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	1.04e+05	1.22e-02	0	2.06e-02	3.28e-02
200	3.96e+04	1.97e+05	4.27e+05	1.89e-02	0	2.64e-02	4.53e-02
300	8.94e+04	4.46e+05	9.71e+05	4.61e-02	0	5.91e-02	1.05e-01
400	1.59e+05	7.94e+05	1.73e+06	7.64e-02	0	9.95e-02	1.76e-01
500	2.49e+05	1.24e+06	2.72e+06	1.16e-01	0	1.56e-01	2.72e-01
600	3.59e+05	1.79e+06	3.92e+06	1.73e-01	0	2.22e-01	3.95e-01

Table 10: τ = 0.0010, λ = 100

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	1.20e+05	1.42e-02	0	2.68e-02	4.10e-02
200	3.96e+04	1.97e+05	5.00e+05	2.06e-02	0	5.10e-02	7.16e-02
300	8.94e+04	4.46e+05	1.14e+06	4.93e-02	0	1.21e-01	1.70e-01
400	1.59e+05	7.94e+05	2.04e+06	8.46e-02	0	2.24e-01	3.09e-01
500	2.49e+05	1.24e+06	3.20e+06	1.34e-01	0	3.51e-01	4.85e-01
600	3.59e+05	1.79e+06	4.62e+06	1.98e-01	0	4.94e-01	6.93e-01

Table 11: τ = 0.0001, λ = 1

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	1.04e+05	9.44e-03	0	2.34e-02	3.29e-02
200	3.96e+04	1.97e+05	4.28e+05	1.97e-02	0	1.55e-02	3.52e-02
300	8.94e+04	4.46e+05	9.73e+05	4.31e-02	0	3.47e-02	7.78e-02
400	1.59e+05	7.94e+05	1.74e+06	7.48e-02	0	5.66e-02	1.31e-01
500	2.49e+05	1.24e+06	2.72e+06	1.20e-01	0	8.88e-02	2.09e-01
600	3.59e+05	1.79e+06	3.93e+06	1.73e-01	0	1.35e-01	3.08e-01

Table 12: τ = 0.0001, λ = 10

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	1.74e+05	1.77e-02	0	1.89e-02	3.66e-02
200	3.96e+04	1.97e+05	7.28e+05	2.95e-02	0	2.55e-02	5.50e-02
300	8.94e+04	4.46e+05	1.66e+06	7.02e-02	0	5.75e-02	1.28e-01
400	1.59e+05	7.94e+05	2.98e+06	1.34e-01	0	1.02e-01	2.36e-01
500	2.49e+05	1.24e+06	4.67e+06	1.99e-01	0	1.65e-01	3.64e-01
600	3.59e+05	1.79e+06	6.74e+06	2.90e-01	0	2.35e-01	5.25e-01

Table 13: τ = 0.0001, λ = 100

N	M^2	nnz(A)	nnz(L)	time	flag	timeSolve	timeSum
100	9.80e+03	4.86e+04	2.31e+05	2.87e-02	0	1.24e-02	4.11e-02
200	3.96e+04	1.97e+05	9.82e+05	5.55e-02	0	4.95e-02	1.05e-01
300	8.94e+04	4.46e+05	2.25e+06	1.24e-01	0	1.07e-01	2.31e-01
400	1.59e+05	7.94e+05	4.04e+06	2.28e-01	0	1.94e-01	4.22e-01
500	2.49e+05	1.24e+06	6.35e+06	3.58e-01	0	3.03e-01	6.61e-01
600	3.59e+05	1.79e+06	9.18e+06	5.15e-01	0	4.25e-01	9.40e-01

3.2 Time plots

Further, we wanted to see how the time consumption of the CG process depends on the drop tolerance τ, and so plots shown in Figure 4 to 9 shows the factorisation time, solve time and the sum of these two separately versus the drop tolerance.

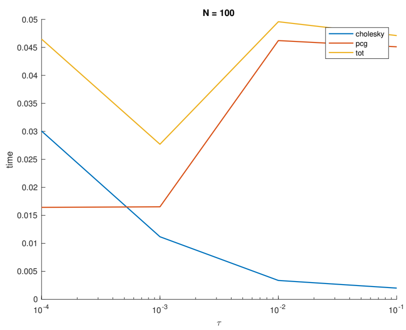

Figure 4: N=100

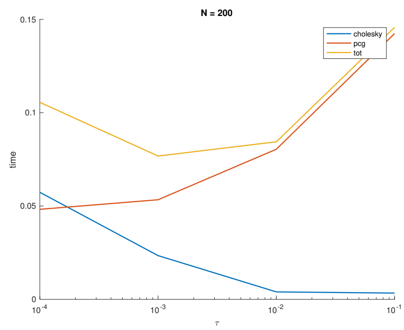

Figure 5: N=200

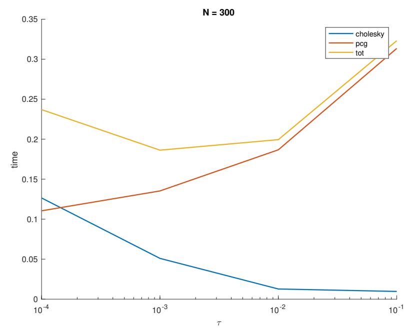

Figure 6: N=300

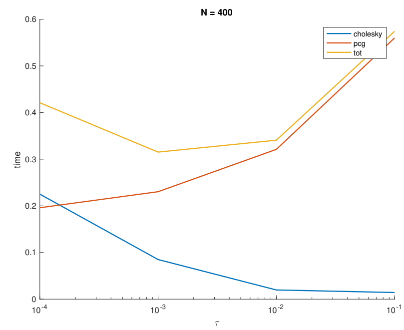

Figure 7: N=400

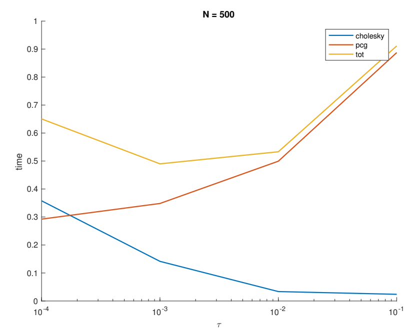

Figure 8: N=500

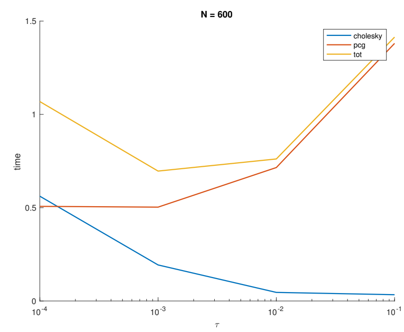

Figure 9: N=600

From the plots, it is obvious that the choice of drop tolerance has an impact on the time consumption, since it has a clear minimum at $\tau = 10^{-3}$ for all N.

4 Solution of the heat equation for a two dimensional plate

In this part we want to modify heatSim.m in order to make it faster and finish with a solution in less than 12 seconds. This is done by implementing preconditioned conjugate gradient method using Matlabs built-in pcg(). We use an incomplete Cholesky factorization to construct a preconditioner for pcg, and in this a drop tolerance of $\tau = 10^{-3}$ since this proved to be efficient in previous exercise. This resulted in a computation time of 10.6 seconds on my Macbook pro, which should mean that it should be able to run even faster on a cs-computer. The plot given in Figure 10 shows a representation of the solution at the final time step.

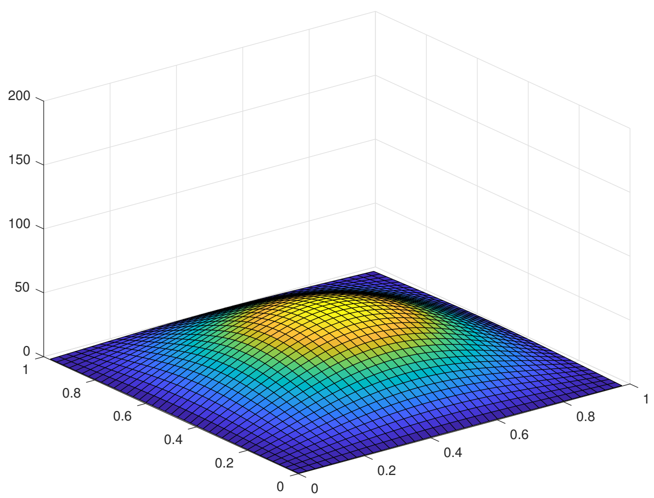

Figure 10: Solution of heat equation plotted after the simulation is done.

A Matlab Code

A.1 direct.m

direct.m

```
%DIRECT - Script that runs tests on direct solvers and prints the
    result
%         from questions in 4.1 of the assignment
%
%   MINIMAL WORKING EXAMPLE:
%
%   >> direct;

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-11-01: Initial version .
%
% Function code starts here...
N = 10:10:400; %define range of N
M = N-1; %from N, define M

lambda=1; %set the critical parameter lambda
time = zeros(length(N),1); %pre-define zero vector for time
nonZeroA = zeros(length(N),1); %pre-define zero vetor for nonZeroA
nonZeroL = zeros(length(N),1); %pre-define zero vetor for nonZeroL
aux = zeros(length(N),6); %pre-define zero vetor for aux
p = 0; %set criterion variable

%loop over all N
for i = 1:length(N)
    A=Heat(N(i),lambda); %construct heat equation matrix
    nonZeroA = length(nonzeros(A)); %store non-zero elements in A
    tic %start timing
    L = chol(A); %perform Cholesky factorization on A
    time = toc; %store time spent on chol(A)
    nonZeroL = length(nonzeros(L)); %store non-zero elements in L
    b = ones(length(L),1); %set b to vector of all ones
    tic %start timing
    x = L\b; %solve system with cholesky-L
    timeSolve = toc; %store time for solving system

    %set variables for printing
    aux(i,:) = [N(i), M(i)^2, nonZeroA, nonZeroL, time, timeSolve];

    %check if ratio is larger than 10 (for the first time (--> p))
    if nonZeroL/nonZeroA > 10 && p == 0
        exceed = N(i); %N for which ratio exceeds 10
        p = 1; %set criterion variable
    end

end

%print table
fprintf('      N      M^2    nnz(A)      nnz(L)      time
    timeSolve \n');
for i = 1:length(N)
    fprintf('%8d %8d %8d %10d %10.2e %10.2e \n',aux(i,:))
end

timeExc = find(aux(:,5) > 1,1); %time for which factorization time
    exceed 1
NtimeExc = N(timeExc); %N for which factorization time exceed 1
```

A.2 Basic.m

Basic.m

```
%BASIC - Script that runs tests on Jacobi's method and prints the result
%        from questions in 4.2 of the assignment
%
%   MINIMAL WORKING EXAMPLE:
%
%   >> Basic;

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-11-01: Initial version .
%
% Function code starts here...
N = 100:100:600; %define N
M = N-1; %define M
lambda=1; %set critical parameter lambda
tol = 1e-6; %set tolerance
maxit = 100; %set maximum iterations before definite termination
resvec = cell(length(N),1); %pre-define empty cell array for residuals

%loop over all N
for i = 1:length(N)
    A=Heat(N(i),lambda); %construct heat equation matrix A
    seed = 2019; %(omg future seed)
    rng(seed);
    b=rand(M(i)^2,1); %construct randomized b vector for system
    x0 = zeros(M(i)^2,1); %set initial condition

    %solve system using Jacobi's method
    [x, flag, relres, it, resvec{i}] = Jacobi(A, b, tol, maxit, x0);
end

%% Plot residual history
figure
hold on
for i = 1:length(N)
    plot(resvec{i})
    xlabel('iter')
    ylabel('resVec')
    legendInfo{i} = ['N = ' num2str(N(i))];
end
legend(legendInfo);

%% Linear convergence
figure
hold on
for i = 1:length(N)
    plot(resvec{i}(1:end-1)./resvec{i}(2:end))
    xlabel('iter')
    ylabel('err_i/err_{i+1}')
    legendInfo{i} = ['N = ' num2str(N(i))];
end
legend(legendInfo);

%% N independence of Jacobi method
figure
hold on
```

```
for i = 1:length(N)
    plot(resvec{i}/N(i))
    xlabel('iter')
    ylabel('resVec/N')
    legendInfo{i} = ['N = ' num2str(N(i))];
end
legend(legendInfo);
```

A.3 testCG.m

testCG.m

```
%TESTCG - Script that determines the smallest value of N, that will make
%          the non-preconditioned CG algorithm require more than one second
%          to solve a linear system of size A.
%
%   MINIMAL WORKING EXAMPLE:
%
%   >> testCG;

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-11-01: Initial version .
%
% Function code starts here...
N = 1400:10:1600; %define range for N
M = N-1; %define M
lambda=1; %define chritical parameter
tol = 1e-6; %set tolerance
maxit = 100; %set maximum iterations before termination
resvec = cell(length(N),1); %pre-define empty cell array for residuals
time = zeros(length(N),1); %pre-define zero vector for storing time

%loop over all N
for i = 1:length(N)
    A=Heat(N(i),lambda); %construct heat equation matrix
    seed = 2019; %(omg future seed)
    rng(seed);
    b=rand(M(i)^2,1); %construct randomized b vector
    tic %start timing
    [x, flag, relres, iter, resvec]=pcg(A,b,tol,maxit); %call pcg for systm
    time(i) = toc; %collect time spent
end

N(find(time>1,1)) %print N-limit asked for. 1440 for example (one occasion)
```

A.4 myPCG.m

myPCG.m

```
function [aux] = myPCG(N,lambda,tau,tol)
%MYPCG - Constructs a linear system based on given parameters, and then
%        solves it using pcg. Prints result for all N given
%
%   MINIMAL WORKING EXAMPLE: Construct and solve systems with size
%                            N = 10:10:100, with critical parameter
%                            lambda=1, drop tolerance tau = 1e-3 and
%                            tolerance tol = 1e-6:
%
%   >> aux = myPCG([10:10:100],1,1e-3,1e-6);

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-11-01: Initial version .
%
% Function code starts here...

M = N-1; %define M

aux = zeros(length(N),8); %pre-define zero array for aux
p = 0; %pre-define termination variable for onZeroL/nonZeroA condition
maxit = 100; %set maximum iterations

% Loop over all N
for i = 1:length(N)

    A=Heat(N(i),lambda); %Construct A using Heat.m
    nonZeroA = length(nonzeros(A)); %Find non-zero elements in A

    %set options for ichol
    opts=struct('type','ict','droptol',tau,'michol','off');
    tic %start timing for ichol
    L = ichol(A,opts); %construct L by using ichol
    time = toc; %stop timing, collect time spent
    nonZeroL = length(nonzeros(L)); %find non-zero elements in L

    seed = 2019; %(omg future seed)
    rng(seed); %set seed
    b=rand(M(i)^2,1); %get randomized b vector

    tic %start timing for pcg
    %solve system using pcg
    [x, flag, relres, iter, resvec]=pcg(A,b,tol,maxit,L,L');
    timeSolve = toc; %stop timing for pcg, collect time spent

    timeSum = time+timeSolve; %sum the total time spent

    %set values in aux for printing
    aux(i,:) = [N(i), M(i)^2, nonZeroA, nonZeroL,...
                time, flag, timeSolve, timeSum];

    %check if ratio has exceeded 10 and if it is the first time
    if nonZeroL/nonZeroA > 10 && p == 0
        exceed = N(i); %N for which ratio exceeds 10
        p = 1; %set first time check-variable
    end
```

```
end

%define header string
text1 = ['       N     M^2   nnz(A)    nnz(L)     time   '...
         '\t flag  timeSolve   timeSum\n'];
fprintf(text1); %print header string

%below is code for latex insertion
% text2 = ['N & M^2 & nnz(A) & nnz(L) & time & flag & timeSolve'...
%          '& timeSum \\\\ \\hline \n'];
% fprintf(text2);

for i = 1:length(N)
    %set row string
    text3 = ['%8d %8d %8d %10d %10.2e   %d %10.2e %10.2e\n'];
    fprintf(text3,aux(i,:)) %print row string

%below is code for latex insertion
%     text4 = ['%8d&%10.2e&%10.2e&%10.2e&%10.2e&%d&%10.2e&%10.2e'
    ...
%         '\\\\ \\hline \n'];
%     fprintf(text4,aux(i,:))

end
```

A.5 testMyPCG.m

testMyPCG.m

```
%TESTMYPCG - Script that solves the questions 4.4.2-4.4.4 in
    Assignment 5.
%
%   MINIMAL WORKING EXAMPLE:
%
%   >> testMyPCG;

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-11-01: Initial version .
%
% Function code starts here...

%% Performs tests in question 4.4.2
N = 100:100:600; %define N
lambda = [1,10,100]; %define lambda
tau = [1e-1 1e-2 1e-3 1e-4]; %define drop tolerance
tol = 1e-6; %define tolerance

%loop over all tau and lambda => 12 tables
for i = 1:length(tau)
    for j = 1:length(lambda)
        %print header
        fprintf('tau = %4.4f, lambda = %d \n',tau(i),lambda(j))

        %call myPCG for relevant lambda and tau
        myPCG(N,lambda(j),tau(i),tol);
        fprintf('\n \n') %print new line
    end
end

%% Time plots, i.e. tests in question 4.4.3
aux = cell(length(tau),1); %pre-define zero array for aux

%Loop over all tau to perform pcg using myPCG
for i = 1:length(tau)
   fprintf('tau = %f, lambda = %d \n',tau(i),lambda(j))
   aux{i} = myPCG(N,lambda(3),tau(i),tol);
   fprintf('\n \n')
end

%plot the factorization time as a function of the drop tolerance
for i = 1:length(N)
    figure
    hold on
    for j = 1:length(tau)
        plotVar1(j) = aux{j}(i,5);
        plotVar2(j) = aux{j}(i,7);
        plotVar3(j) = aux{j}(i,8);
    end
    plot(tau,plotVar1,'linewidth',1.4)
    plot(tau,plotVar2,'linewidth',1.4)
    plot(tau,plotVar3,'linewidth',1.4)
    title(sprintf('N = %d', N(i)))
    xlabel('\tau')
    ylabel('time')
    legend('cholesky','pcg','tot')
    set(gca,'xscale','log')

```

```
    %!!!!! NOTE TO SELF: tau = 1e-3 gives lowest total time
    !!!!!!!!
end
```

A.6 myHeatSim.m

myHeatSim.m

```
%MYHEATSIM - Script that solves heat equation in 2d-plate, using
    pcg that
%            uses ichol for preconditioning.
%
%   MINIMAL WORKING EXAMPLE:
%
%   >> myHeatSim;

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-11-01: Initial version .
%
% Function code starts here...

% HeatSim  Similation of heat in 2D plate
tic
% Set number of intervals in spatial direction
N=400; M=N-1;

% Compute spatial stepsize
h=1/N;

% Define grid
t=linspace(1,M,M)*h; [x, y]=meshgrid(t,t);

% Define initial temperatur distribution
u=g(x,y);

% Suppress details from plot to make it readable
nb=floor(N/40); tau=t(1:nb:end,1:nb:end);

% Plot the initial distribution of heat
u1=u(1:nb:end,1:nb:end); surf(tau,tau,u1);

% Maximum temperature display
maxtemp=200;

% Enforce axis
axis([0 1 0 1 0 maxtemp]);

% Hold graphics
hold;

% Set the simulation time
T=0.03;

% Set the number of timesteps
count=120;

% Compute time step
k=T/count;

% Compute critical parameter lambda
lambda=k/h^2; %multiply by 0.025 to get 1?

% Generate the matrix
A=Heat(N,lambda);

% Generate right hand side
```

```
b=reshape(u,M^2,1);

% Set initial guess
x0=zeros(M^2,1);

% Set tolerance
tol=1e-3;

% Set maximum nummber of Jacobi iterations per time step
maxit=10000;

%--------------------my added code
    ----------------------------------------
dropTol = 1e-3;
opts=struct('type','ict','droptol',dropTol,'michol','off');
L = ichol(A,opts);
%
    -------------------------------------------------------------------------

% Loop over the timesteps
for n=1:count
    % Pause to give graphics routine time to catch up
    % pause(0.01);
    % Solve linear system using pcg to advance one time step
    [x, flag, relres, it, resvec]=pcg(A,b,tol,maxit,L,L',x0);

    % Update right hand side
    b=x;

    % Display solution
    clf; u=reshape(x,M,M); u1=u(1:nb:end,1:nb:end);  surf(tau,tau,
    u1); ...
         view([0,90]); axis([0 1 0 1 0 maxtemp]);
end
toc
```