UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Aerospace Technology Engineering

EXERCISE 2

Linear Solvers in MATLAB

APMC++ to Thermal Engineering Problems

Abril Bertran Garrigos

May 2025

Index

1 Case of study **1**

2 Algorithm for solving 1D conduction cases **2**

2.1 Assembly A matrix 2

2.2 Assembly b matrix 3

2.3 Solvers and approaching methods 3

2.4 TDMA 5

3 Solutions and conclusions **7**

1 Case of study

For this exercise, the main objective is to analyze the heat behavior of the same case as exercise 1 using linear solvers. Therefore, the exercise will consist of finding the corresponding coefficients a_{ij} and b_{ij} as well as creating the matrix A and vector b. The nodes will have the following equations:

$$
\begin{array}{llr}
(1+3\gamma)\,T_i^{n+1} = \phantom{\gamma T_{i-1}^{n+1}+}\gamma T_{i+1}^{n+1} + T_i^n + 2\gamma T_l & & \text{left node} \\
(1+2\gamma)\,T_i^{n+1} = \gamma T_{i-1}^{n+1} + \gamma T_{i+1}^{n+1} + T_i^n & & \text{middle nodes} \\
(1+3\gamma)\,T_i^{n+1} = \gamma T_{i-1}^{n+1} \phantom{+\gamma T_{i+1}^{n+1}} + T_i^n + 2\gamma T_r & & \text{right node}
\end{array}
$$

Figure 1: Nodes equation

As for the numerical data, is required to use a time step $dt = 0.1s$ with a $t_{end} = 600s$ with a large discretization of $N = 500$ nodes number.

The expected results consist of a time comparison between the different solving methods that can be used in MATLAB to know which one is fastest. The cases that will be studied are:

1. Use the mldivide command (either using the function or with $T = A \setminus b$)
2. Use the linsolve knowing that the matrix will be symmetric and positive-definite
3. Invert the matrix every time iteration and perform $T = A^{-1}b$
4. Matrix A is constant, so you can invert it before the temporal iteration, store the result as a new matrix A_{inv} and use it to perform a matrix-vector multiplication every time step
5. Create a sparse matrix S and solve using mldivide ($T = S \setminus b$)
6. Using the self-programmed Gauss-Seidel approach. Time might be very dependent on ε, try $\varepsilon = 10^{-6}$ (sufficient precision) and $\varepsilon = 10^{-13}$ (High precision)
7. Solve the system using a TDMA algorithm

2 Algorithm for solving 1D conduction cases

For the algorithm it has been used the template provided by the teacher, therefore all the physical and numerical inputs are already done, as well as the structure of the time loop. The only parts to program were the assembly of the A matrix and b vector, as well as the linear solver and time calculation.

2.1 Assembly A matrix

To assembly the A matrix it is important to know that have a dimension of $[N \times N]$ as well as it has tridiagonal properties where interior elements follow a similar pattern. The values of the main diagonal have the same value of $1 + 2\gamma$ except for the extremes that have $1 + 3\gamma$, while the outer elements are $-\gamma$.

$$
\begin{pmatrix}
1+3\gamma & -\gamma & 0 & 0 & 0 & 0 & 0 & 0 \\
-\gamma & 1+2\gamma & -\gamma & 0 & 0 & 0 & 0 & 0 \\
0 & -\gamma & 1+2\gamma & -\gamma & 0 & 0 & 0 & 0 \\
0 & 0 & -\gamma & 1+2\gamma & -\gamma & 0 & 0 & 0 \\
0 & 0 & 0 & -\gamma & 1+2\gamma & -\gamma & 0 & 0 \\
0 & 0 & 0 & 0 & -\gamma & 1+2\gamma & -\gamma & 0 \\
0 & 0 & 0 & 0 & 0 & -\gamma & 1+2\gamma & -\gamma \\
0 & 0 & 0 & 0 & 0 & 0 & -\gamma & 1+3\gamma
\end{pmatrix}
\begin{pmatrix}
T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_5^{n+1} \\ T_6^{n+1} \\ T_7^{n+1} \\ T_8^{n+1}
\end{pmatrix}
=
\begin{pmatrix}
T_1^n + 2\gamma T_l \\ T_2^n \\ T_3^n \\ T_4^n \\ T_5^n \\ T_6^n \\ T_7^n \\ T_8^n + 2\gamma T_r
\end{pmatrix}
$$

Figure 2: A matrix and b vector

Therefore, to get the matrix a loop can be applied to get the inner values and once it finishes assign the values of the extremes.

```
for i=2:(N)
    for j=2:(N)
        if i==j
            A(i,j)=1+2*gamma;
            A(i-1,j)=-gamma;
            A(i,j-1)=-gamma;
        end
    end
end

A(1,1)=1+3*gamma;
A(end,end)=1+3*gamma;
```

2.2 Assembly b matrix

To get the b vector, a similar logic as the previous section is followed. The b vector has N elements and all the values follow the same pattern except for the first and last one. Since the b vector has a dependency with the time, for each dt needs to be recalculated so it is included inside the time loop.

```
    for i=1:length(b)
        if i==1
            b(i)=T(i)+2*gamma*Tl;
        elseif i==length(b)
            b(i)=T(i)+2*gamma*Tr;
        else
            b(i)=T(i);
        end
    end
```

2.3 Solvers and approaching methods

Once the A matrix and b vector are obtained, it can be proceeded to calculate the values of the temperature at $t = 600s$.

In order to get the time-consuming value of each method, it is used a matlab default chronometer that starts when it is written the command *tic* which position will vary depending on the method, and ends when it is written *toc* which is placed at the end of the time loop.

The complete code is the following one:

```
%% Physical propperties
T0 = 30;    % Initial temperature [C]
Tl = 100;   % Left temperature [C]
Tr = 20;    % Right temperature [C]
L = 1;      % Bar length [m]
tfin = 600; % End time [s]
alpha = 400/(8960*380); % thermal diffusivity [m2/s]

%% Numerical propperties
N = 500;    % Number of nodes
dt = 0.1;   % Timestep
dx = L/N;

```

```
%% Dimension set of matrix A ,vector b and vector T (started at T0)
A = zeros(N);
b = zeros(N,1);
T = ones(N,1)*T0;

%% Other calculations
x=dx/2:dx:L-dx/2;
gamma= alpha*dt/dx^2;

%% Construction of A matrix
for i=2:(N)
    for j=2:(N)
        if i==j
            A(i,j)=1+2*gamma;
            A(i-1,j)=-gamma;
            A(i,j-1)=-gamma;
        end
    end
end

A(1,1)=1+3*gamma;
A(end,end)=1+3*gamma;

%% Construction of S matrix
S=sparse(A);

%% Inverted A matrix
invA=inv(A);

tic
%% Temporal Iteration
t = 0;

while t <= tfin

    for i=1:length(b)
        if i==1
            b(i)=T(i)+2*gamma*Tl;
```

```
        elseif i==length(b)
            b(i)=T(i)+2*gamma*Tr;
        else
            b(i)=T(i);
        end
    end

    % A) MLDIVIDE
    %T_result=mldivide(A,b);

    % B) LINSOLVE
    %opts.POSDEF=true;
    %opts.SYM=true;

    %T_result= linsolve(A,b,opts);

    % C)INVERT MATRIX
    %T_result=inv(A)*b;

    % D) INVERT MATRIX BEFORE TIMELOOP
    %T_result=invA*b;

    % E) SPARSE MATRIX
    T_result=mldivide(S,b);

    T=T_result;
    t = t + dt;
end

temps=toc;
```

2.4 TDMA

Finally, it is asked to also solve the system using TDMA, therefore it is needed to initialize two auxiliary vectors of N size:

```
    P=zeros(N,1);
    R=zeros(N,1);
```

Then, it is used the formulas used in the class notes to recalculate those two vectors for each dt, as well as the temperature with a backward loop:

```
    while t <= tfin

    for i=1:length(b)
        if i==1
            b(i)=T(i)+2*gamma*Tl;
        elseif i==length(b)
            b(i)=T(i)+2*gamma*Tr;
        else
            b(i)=T(i);
        end
    end

    P(1)=A(1,2)/A(1,1);
    R(1)=b(1)/A(1,1);
    for i=2:(N-1)
        P(i)=A(i,i+1)/(A(i,i)-A(i,i-1)*P(i-1));
        R(i)=(b(i)-A(i,i-1)*R(i-1))/(A(i,i)-A(i,i-1)*P(i-1));
    end

    R(N)=(b(N)-A(N,N-1)*R(N-1))/(A(N,N)-A(N,N-1)*P(N-1));
    for i=N:-1:1
        if i==N
            T_result(i)=R(i);
        else
            T_result(i)=R(i)-P(i)*T_result(i+1);
        end
    end

    T=T_result;
    t = t + dt;
end
```

3 Solutions and conclusions

Once the code is done, it is presented the time results for each solving method in the next table:

Solving method	**Time [s]**
A) mldivide	7.81
B) linsolve	6.76
C) Invert matrix $T = A^{-1} \cdot b$	21.53
D) Invert matrix precalculated $T = A_{inv} \cdot b$	0.087
E) mldivide with sparse matrix S	0.068
F) Gauss-Seidel for $\varepsilon = 10^{-6}$	1.35
G) Gauss-Seidel for $\varepsilon = 10^{-13}$	4.15
H) TDMA	0.063

Table 1: Table Results

As expected, the (C) results are the worst ones since it is the most time-consuming method. This is because it requires to calculate the inverted A matrix for every, dt which it translates to a high computational cost. However, since the A matrix is constant, there is a big improvement of the time if its inverse is precalculated outside the time loop (D).

As for the mldivide (A) function, it has resulted in not the best method as well as the linsolve (B) which presents only a small improvement in time given that the properties of the matrix are entered into the program, saving some calculations.

Then, about the method (E) which is equal to the (A), but using just a sparse matrix it results in a most optimum solver, since it saves data memory in the matrix and just stores the elements directly defined.

Regarding the iterative method of Gauss-Seidel, it is seen that its computational cost varies as a function of the tolerance applied. Therefore, if there is not a lot of precision required, it can be convenient to use it, however it is better to use the linear solver (E) since it will be faster.

Finally, it is also implemented the TDMA which has resulted to be in one of the most optimum solving systems, since it has one of the lowest time-consuming values.

All in all, the best methods are concluded to be (D),(E) and (H) which present similar results and magnitude order.