Tutorial 3
SI 507 - Numerical Analysis

Direct and Iterative methods

October 3, 2019

Syllabus

(D) Direct methods

(D.a) Thomas algorithm

(I) Iterative methods

(I.a) Jacobi Iteration

(I.b) Gauss-Seidel Iteration

(R) Root-finding algorithms

(R.a) Bisection method

(R.b) Regula-Falsi and Secant

(R.c) Newton-Raphson

(R.d) Fixed point method

Tentative schedule

In tutorial-1 *(4/10/2019)*, we will instruct how to install GNU-Octave from scratch, what are the functionalities, how to implement and run a code in GUN-Octave et c. Specifically, we implement the methods in (D) and (I) with simple generic problems as prototypes. Once the students are comfortable with running the codes, we instruct them to implement questions (6) and (9) in tutorial sheet 4. Depending on the availability of time, we show how these algorithms are actually used in solving a second order ordinary differential equation (minimal amount of theory of difference equations needs to provided for this).

In tutorial-2 *(9/10/2019)*, we implementfour root finding algorithms in (R). We instruct the coding part for simple generic problems. Questions (1) and (6) in tutorial sheet 6, and questions (1), (4) and (5) in tutorial sheet 7 are solved as examples.

1 General Introduction

Numerical implementation of various schemes is an essential and complementing part of numerical analysis, where we test algorithms, improve and optimise codes and programs, and transfer them to industrial applications. There are many high-level programming languages (C, C++, Fortran, Python), packages (Fenics, Comsol, FreeFem, Pyclaw), and consolidated softwares (Matlab, GNU-Octave, Scilab, Mathematica, OptimJ, Maple) available in the market today, and often we need to choose a language or software that best suits with our purpose, gives fast results, and versatile to a wide range of problems. In the forthcoming prose, we actively use *GNU-Octave*, which is a free-software, as our implementation platform.

2 Problems

Problem 2.1 Thomas algorithm for tri-diagonal systems

Use Thomas method to solve the tri-diagonal system of equations

$$2x_1 + 3x_2 = 1, \quad x_1 + 2x_2 + 3x_3 = 4, \tag{1}$$
$$x_2 + 2x_3 + 3x_4 = 5, \quad x_3 + 2x_4 = 2. \tag{2}$$

Solution

We consider a general system $A\boldsymbol{x} = \boldsymbol{b}$, where A is an $N \times n$ real and tri-diagonal matrix, $\boldsymbol{x}$ and $\boldsymbol{b}$ are $N \times 1$ vectors. While solving the system, we need to bear in mind, that in general Thomas algorithm in not stable. However, most of the systems we encounter have a symmetric positive definite or diagonally dominant coefficient matrix, A, in which case, Thomas algorithm is stable and very efficient (in fact, the operations count for Thomas algorithm is $\mathcal{O}(n)$). Firstly, assume that system $A\boldsymbol{x} = \boldsymbol{b}$ has the following structure:

$$\begin{pmatrix} b_1 & c_1 & & & & 0 \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & b_{N-1} & c_{N-1} \\ 0 & & & & a_N & b_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{pmatrix}. \tag{3}$$

In this case, the Thomas algorithm is as follows:

Thomas algorithm

Step 1. Define $p_i := a_i/b_{i-1}$, $b_i := b_i - p_i c_{i-1}$, $d_i := d_i - p_i d_{i-1}$ for $i = 2, \ldots, n$. (Why c_i and a_i is unchanged?)

Step 2. Back substitution: get $x_n = d_n/b_n$, $x_i = (d_i - c_i x_{i+1})/b_i$ for $i = n - 1, \ldots, 1$.

A GNU-Octave (matlab) code to implement this algorithm is given below.

```
function x = solve_thomas(A,d)
  na = size(A,1); % number of elements in A

  for i = 2:na
    p = A(i,i-1)/A(i-1,i-1);            % p = a_i/b_{i-1}
    A(i,i) = A(i,i) - p*A(i-1,i);      % b_i = b_i - p*c_{i-1}
    d(i,1) = d(i,1) - p*d(i-1,1);       % d_i = d_i - p*d_{i-1}
  endfor

  x = zeros(na,1);
  x(na,1) = d(na,1)/A(na,na);
  for i = na-1:-1:1
    x(i,1) = (d(i,1) - A(i,i+1)*x(i+1,1))/A(i,i);
    endfor

endfunction
```

Now, we will illustrate how to use this function to solve the system in Problem 2. In Octave command window *(Ctrl + 0)*, initialize the coefficient matrix A, and the load vector $\boldsymbol{b}$.

```
>> A = [2 3 0 0;1 2 3 0;0 1 2 3;0 0 1 2]; b = [1;4;5;2];
>>
```

Calling the function solve_thomas(A,b) gives the following output.

```
>> solve_thomas(A,b)
ans =
-1.81818
1.54545
0.90909
0.54545
>>
```

To verify the results compute $A\boldsymbol{x}$, and check if it is same as $\boldsymbol{b}$.

```
>> A*ans
ans =
1
4
5
2
>>
```

We use the same procedure to verify the reliability of the encoded schemes in the next problems as well.

Problem 2.2 Jacobi iteration method

Implement Jacobi iteration method in GNU-Octave, and obtain the solution of the following linear system:

$$\begin{pmatrix} 3 & 2 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}. \tag{4}$$

Your code must include a precaution to avoid infinite loops.

Solution

The Jacobi algorithm iteratively finds an approximate solution to the linear system $A\boldsymbol{x} = \boldsymbol{b}$ in the following way. For an iterative $\boldsymbol{x}^{(k)}$, obtain the successive iterate $\boldsymbol{x}^{(k+1)}$ as:

$$x_m^{(k+1)} = -\sum_{n \neq m}^{N} \frac{A_{mn}}{A_{mm}} x_n^{(k)} + \frac{b_m}{A_{mm}}. \tag{5}$$

Jacobi algorithm is very efficient in the sense it could be vectorially implemented in GNU-Octave. This means, we completely avoid *for* loops, and obtain modifications for vectors as such. For instance, consider a situation where $[1, 2, 3]$ is changed to $[2, 3, 4] = [1, 2, 3] + 1$. We can implement this in two ways: the inefficient and slow *for* loop method,

```
b = [1 2 3]
for i=1:3
b(i) = b(i) + 1;
endfor
```

or the efficient vectorial method.

```
b = [1 2 3]
b = b + 1;
```

The vectorial algorithm for Jacobi iteration method is given below.

Jacobi iteration method

Step 1. Start with an initial guess $\boldsymbol{x}^{(0)}$.

Step 2. Set the matrix $\widetilde{A}$, and the vector $\widetilde{\boldsymbol{b}}$ as $\widetilde{A}_{ij} = (\delta_{ij} - 1)A_{ij}/A_{ii}$ and $\widetilde{b}_i = b_i/a_{ii}$ for $i, j = 1, \cdots, N$. Here, δ_{ij} is the Kronecker delta.

Step 3. For $k \geq 0$, $\boldsymbol{x}^{(k+1)} = \widetilde{A}\boldsymbol{x}^k + \widetilde{b}$.

Step 4. Stopping criteria: Compute the relative error $\epsilon_k = ||\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}||_2/||\boldsymbol{x}^{(k)}||_2$. Fix a tolerance value ϵ_{tol}, and a maximum iteration number, $K_{\max}$. Exit the loop if either of $\epsilon_k < \epsilon_{\text{tol}}$ or $k > K_{\max}$.

The maximum iteration number ensures that the loop does not become infinite. However, a message must be given to the user in this case, that the maximum iteration number is reached, and the relative error is not below the desired tolerance value within $K_{\max}$ number of iterations. A working code and the output are as follows.

```
1  function x = solve_jacobi(A,b,x0,tol,kmax)
2
3    N = size(A,1);                % get the dimension of the system
4    x = x0;
5    Atemp = zeros(N,N);
```

```
6    Btemp = zeros(N,1);
7    b = b./diag(A);
8    A = -bsxfun(@rdivide,A,diag(A)); % matrix-vector pairwise multiplication
9    A = A - diag(diag(A));           % eliminates diagonal entries of A
10
11   for k =1:kmax
12     x = A*x + b;                        % Jacobi iteration
13     if norm(x - x0)/norm(x0 + eps) < tol % relative error (why eps?)
14       break
15     endif
16       x0 = x;
17   endfor
18
19   fprintf('\n Relative eroor is %e \n',norm(x - x0)/norm(x0 + eps));
20
21   if k == kmax % error message on maximum iteration number reaching
22     disp('Maximum iteration reached. Solution may be erraneous. ');
23   endif
24
25
26 endfunction
```

```
 >> A = [3 2;1 2]; b = [-1;1]; x0 = [0;0]; tol = 1e-3; kmax = 100;
>> solve_jacobi(A,b,x0,tol,kmax)
Relative eroor is 5.836790e-04
ans =
-9.990855052583447e-01
9.993141289437585e-01
>>
```

Problem 2.3 Gauss-Seidel method

Suppose the temperature of a metal rod of length 10 m has been measure to be 0^oC and 10^oC at each end, respectively. Find the temperatures x_1, x_2, x_3 and x_4 at the four points equally spaced with the interval of 2 m, assuming that the temperature at each point is the average of the temperatures of both neighbouring points. Use Gauss-Seidel method to solve the resulting linear system.

The linear system corresponding to Problem 2 is given by

$$x_2 - 2x_1 = 0, \quad x_1 - 2x_2 + x_3 = 0,$$
$$x_2 - 2x_3 + x_4 = 0, \quad x_3 - 2x_4 = -10.$$

Gauss - Seidel method is a time-improvement over the Jacobi iteration method, where we use the recent updated values $(x_i^{(k+1)})_{\{1\leq i\leq m\}}$ to obtain $x_{m+1}^{(k+1)}$. The formal method is as follows:

$$x_{m+1}^{(k+1)} = -\sum_{n=1}^{m} \frac{A_{mn}}{A_{mm}} x_n^{(k+1)} - \sum_{n=m+2}^{N} \frac{A_{mn}}{A_{mm}} x_n^{(k)} + \frac{b_m}{A_{mm}}. \tag{6}$$

The algorithm is as follows:

Gauss-Seidel algorithm

Step 1. Start with an initial guess $\boldsymbol{x}^{(0)}$.

Step 2. Set the matrix $\widetilde{A}$, and the vector $\widetilde{\boldsymbol{b}}$ as $\widetilde{A}_{ij} = (\delta_{ij} - 1)A_{ij}/A_{ii}$ and $\widetilde{b}_i = b_i/a_{ii}$ for $i, j = 1, \cdots, N$. Here, δ_{ij} is the Kronecker delta.

Step 3. For $k \geq 0$, set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)}$. For $i = 1, \ldots, N$, $x^{(k+1)}(i) = \widetilde{A}\boldsymbol{x}^{(k+1)} + \widetilde{b}$.

Step 4. Stopping criteria: Compute the relative error $\epsilon_k = ||\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}||_2/||\boldsymbol{x}^{(k)}||_2$. Fix a tolerance value ϵ_{tol}, and a maximum iteration number, $K_{\max}$. Exit the loop if either of $\epsilon_k < \epsilon_{\text{tol}}$ or $k > K_{\max}$.

A working code and the output are as follows.

```
function x = solve_gseidel(A,b,x0,tol,kmax)

  N = size(A,1);               % get the dimension of the system
  x = x0;
  Atemp = zeros(N,N);
  Btemp = zeros(N,1);
  b = b./diag(A);
  A = -bsxfun(@rdivide,A,diag(A)); % matrix-vector pairwise multiplication
  A = A - diag(diag(A));           % eliminates diagonal entries of A

  for k =1:kmax

    for i = 1:N
      x(i) = A(i,:)*x + b(i);          % Gauss-Seidel loop
    end

    if norm(x - x0)/norm(x0 + eps) < tol % relative error (why eps?)
      break
    endif
      x0 = x;
  endfor

  fprintf('\n Relative eroor is %e \n',norm(x - x0)/norm(x0 + eps));

  if k == kmax % error message on maximum iteration number reaching
    disp('Maximum iteration reached. Solution may be erraneous. ');
  endif

endfunction
```

```
>> A = [-2 1 0 0;1 -2 1 0;0 1 -2 1; 0 0 1 -2]; b = [0;0;0;-10];
>> x0 = [0;0;0;0;]; tol = 1e-3; kmax = 100;
>> solve_gseidel(A,b,x0,tol,kmax)
Relative eroor is 8.391027e-04
ans =
1.991432942450047e+00
3.988785576075315e+00
5.990927340462804e+00
7.995463670231402e+00
>>
```

3 Additional problems

(A.1) Find the $n \times n$ matrix B, and the n-dimensional vector $\boldsymbol{c}$ such that the Gauss-Seidel method can be written in the form

$$\boldsymbol{x}^{(k+1)} = B\boldsymbol{x}^{(k)} + \boldsymbol{c}. \tag{7}$$

Use (7) to eliminate the for loop

```
for i = 1:N
x(i) = A(i,:)*x + b(i);          % Gauss-Seidel loop
endfor
```

in the code for Gauss-Seidel method.

(A.2) Let $\boldsymbol{x}^{(7)}$ be the 7th term of the Gauss-Seidel iterative sequence for the system

$$\begin{aligned} 3x_1 + 2x_2 &= 1 \\ 4x_1 + 12x_2 + 3x_3 &= -2 \\ x_1 + 3x_2 - 5x_3 &= 3 \end{aligned}$$

with $\boldsymbol{x}^{(0)} = (0,\, 0,\, 0)^T$. If $\boldsymbol{x}$ denotes the exact solution of the given system, then show that

$$||\boldsymbol{e}^{(7)}||_\infty \leq 0.058527664||\boldsymbol{x}||_\infty. \tag{8}$$

(A.3) Consider the boundary value problem $y''(x)+y'(x)+y(x) = \cos(x)$ for $0 < x < \pi$, $y(0) = 0 = y(\pi)$. Partition the interval $(0, \pi)$ into N uniform subintervals (x_i, x_{i+1}) for $0 \leq i \leq N-1$ with $x_{i+1} - x_i = h$. Use the difference formulae

$$\begin{aligned} y'(x) &= \frac{y_{i+1} - y_{i-1}}{2h}, \\ y''(x) &= \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}, \end{aligned}$$

to obtain the scheme, for $2 \leq i \leq N-2$

$$y_{i+1}(1 + h/2) + y_i(h^2 - 2) + y_{i-1}(1 - h/2) = h^2 \cos(x_i), \tag{9}$$

for $i = 1$ and $i = N-1$

$$y_2(1 + h/2) + y_1(h^2 - 2) = h^2 \cos(x_1), \tag{10}$$

$$y_{N-1}(h^2 - 2) + y_{N-2}(1 - h/2) = h^2 \cos(x_N). \tag{11}$$

Construct an $N-1 \times N-1$ system $A\boldsymbol{y} = \boldsymbol{b}$ from (9), (10) and (11), where

$$A = \begin{pmatrix} h^2-2 & 1+h/2 & 0 & \cdots & 0 & 0 \\ 1-h/2 & h^2-2 & 1+h/2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & 1-h/2 & h^2-2 \end{pmatrix} \tag{12}$$

$$\boldsymbol{y} = (y_1 \cdots y_{N-1})^T, \text{ and } \boldsymbol{b} = h^2(\cos(x_1) \cdots \cos(x_{N-1}))^T. \tag{13}$$

Use Thomas algorithm, Jacobi iteration method and Gauss-Seidel method to obtain $\boldsymbol{y}$. Use *tic-toc* command in GNU-Octave to find which method is faster.