MATH7003-00: Assignment #6

Sangman Jung
e-mail: sangmanjung@khu.ac.kr
Kyung Hee University – May 13, 2020

Problem. Consider the example in slide 17 and implement MATLAB code to generate Tables 6.20 and 6.21 using method of lines. Discuss about the results. [1].

Example. Solve the partial differential equation problem (6.9.19)-(6.9.21) with the function G, d_0, d_1, and f determined from known solution

$$U = e^{-0.1t}\sin(\pi x) \qquad 0 \le x \le 1 \qquad t \ge 0$$

Results for Euler's method (6.9.29) are given in Table 6.20, and results for the backward Euler method (6.9.33) are given in Table 6.21.

For Euler's method, we take $m = 4, 8, 16$, and to maintain stability, we take $h = \delta^2/2$, from (6.9.32). Note this leads to the respective time steps of $h \doteq 0.031, 0.0078, 0.0020$. From (6.9.26) and the error formula for Euler's method, we would expect the error to be proportional to δ^2, since $h = \delta^2/2$. This implies that the error should decrease by a factor of 4 when m is doubled, and the results in Table 6.20 agree. In the table, the column 'Error' denotes the maximum error at the node points (x_j, t), $0 \le j \le n$, for the given value of t.

Table 6.20 The method of lines: Euler's method

t	Error $m = 4$	Ratio	Error $m = 8$	Ratio	Error $m = 16$
1.0	3.89E − 2	4.09	9.52E − 3	4.02	2.37E − 3
2.0	3.19E − 2	4.09	7.79E − 3	4.02	1.94E − 3
3.0	2.61E − 2	4.09	6.38E − 3	4.01	1.59E − 3
4.0	2.14E − 2	4.10	5.22E − 3	4.02	1.30E − 3
5.0	1.75E − 2	4.09	4.28E − 3	4.04	1.06E − 3

Table 6.21 The method of lines: backward Euler's method

t	Error $m = 4$	Error $m = 8$	Error $m = 16$
1.0	4.45E − 2	1.10E − 2	2.86E − 3
2.0	3.65E − 2	9.01E − 3	2.34E − 3
3.0	2.99E − 2	7.37E − 3	1.92E − 3
4.0	2.45E − 2	6.04E − 3	1.57E − 3
5.0	2.00E − 2	4.94E − 3	1.29E − 3

For the solution of (6.9.28) by the backward Euler method, there need no longer be any connection between the space step δ and the time step h. By observing the error formula (6.9.26) for the method of lines and the truncation error formula (6.9.7) (use $p = 1$) for the backward Euler method, we see that the error in solving the problem (6.9.19)-(6.9.21) will be proportional to $h + \delta^2$. For the unknown function U of (6.9.34), there is a slow variation with t. Thus for the truncation error associated with

the time integration, we should be able to use a relatively large time step h as compared to the space step δ, in order to have the two sources of error be relatively equal in size. In Table 6.21, we use $h = 0.1$ and $m = 4, 8, 16$. Note that this time step is much larger than that used in Table 6.20 for Euler's method, and thus the backward Euler method is much more efficient for this particular example.

Discussion.

Introduction.

Method of lines is a technique for solving **partial differential equations** in which all but one dimension is discretized. In our textbook, the method fix the spatial variable as discretization using δ called **spatial step**. In other cases, discretization using the **time step** h is available.

In detail, the method of lines most often refers to the construction or analysis of numerical methods for partial differential equations that proceeds by first discretizing the spatial derivatives only and leaving the time variable continuous. This leads to a system of **ordinary differential equations** to which a numerical method for initial value ordinary equations can be applied.

Practical procedure of method of lines

Now, our solution of the partial differential equation is

$$U = e^{-0.1t}\sin(\pi x) \qquad 0 \le x \le 1 \qquad t \ge 0$$

Then, the function G is calculated by the partial derivatives U_t and U_{xx} in (6.9.19) as

$$G(x,t) = U_t - U_{xx} = (\pi^2 - 0.1)\,U(x,t).$$

The initial condition and boundary condition is

$$U(x,0) = \sin(\pi x) = f(x), \quad U(0,t) = \vec{0} = d_0(t) \quad \text{and} \quad U(1,t) = e^{-0.1t}\sin\pi = d_1(t).$$

For the spatial step $\delta = 1/m$, $m = 4, 8, 16$, compute $x_j = j\delta$ for $j = 0,1,\ldots,m$ and let $h = \delta^2/2$, then we obtain $0 \le t \le 5$ has the step size h. Finally reduced numerical ODE form of PDE is (6.9.23) that

$$u_j'(t) = \frac{1}{\delta^2}\left[u_{j+1}(t) - 2u_j(t) + u_{j-1}(t)\right] + G(x_j,t) \qquad j = 1,2,\ldots,m-1.$$

Since $u_0(t) = d_0(t)$ and $u_m(t) = d_1(t)$, we just have to calculate $j = 1,2,\ldots,m-1$. Now, let the matrix form of system (6.9.23) is

$$\boldsymbol{u}'(t) = A\,\boldsymbol{u}(t) + \boldsymbol{g}(t) \qquad \boldsymbol{u}(0) = \boldsymbol{u}_0$$

$$\boldsymbol{u}(t)=\begin{bmatrix} u_1(t) \\ \vdots \\ u_{m-1}(t) \end{bmatrix},\ \boldsymbol{u_0}=\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{m-1}) \end{bmatrix},\ \boldsymbol{g}(t)=\begin{bmatrix} \frac{1}{\delta^2}d_0(t)+G(x_1,t) \\ G(x_2,t) \\ \vdots \\ G(x_{m-2},t) \\ \frac{1}{\delta^2}d_1(t)+G(x_{m-1},t) \end{bmatrix},$$

$$\Lambda=\frac{1}{\delta^2}\begin{bmatrix} -2 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & -2 & 1 & 0 & & \vdots \\ 0 & 1 & -2 & 1 & & \vdots \\ \vdots & & & \ddots & & 0 \\ & & & & 1 & -2 & 1 \\ 0 & \cdots & 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \in \mathrm{R}^{\mathrm{m-1\times m-1}}.$$

If $m=4$, then we obtain three column vectors of size 3×1 and Λ of size 3×3. Then, for the boundary condition d_0 and d_1, the size of the matrix as entire solution of the equation is $5\times N(h)$. If **Euler's method** is applied, we have the numerical method

$$V_{n+1}=V_n+h\left[\Lambda V_n+\boldsymbol{g}(t_n)\right] \qquad V_0=\boldsymbol{u_0}$$

with $t_n=nh$ and $V_n \doteq \boldsymbol{u}(t_n)$. The **backward Euler method** is that

$$V_{n+1}=V_n+h\left[\Lambda V_{n+1}+\boldsymbol{g}(t_{n+1})\right] \Rightarrow V_{n+1}=(I-h\Lambda)^{-1}\left[V_n+h\,\boldsymbol{g}(t_{n+1})\right] \qquad V_0=\boldsymbol{u_0}$$

$I-h\Lambda$ is a **tridiagonal**, it is **invertible,** so that we can compute $(I-h\lambda)^{-1}$. (see the paper **[4]**)

Results: Table 6.20 & 6.21

명령 창

Table 6.20 The method of lines: Euler's method

t	Error m = 4	Ratio	Error m = 8	Ratio	Error m = 16
1.0	4.85e-02	4.10	1.18e-02	4.02	2.94e-03
2.0	4.39e-02	4.10	1.07e-02	4.02	2.66e-03
3.0	3.97e-02	4.10	9.69e-03	4.02	2.41e-03
4.0	3.59e-02	4.10	8.77e-03	4.02	2.18e-03
5.0	3.25e-02	4.10	7.93e-03	4.02	1.97e-03

Table 6.21 The method of lines: backward Euler method

t	Error m = 4	Error m = 8	Error m = 16
1.0	4.85e-02	1.19e-02	2.99e-03
2.0	4.39e-02	1.08e-02	2.70e-03
3.0	3.98e-02	9.73e-03	2.45e-03
4.0	3.60e-02	8.81e-03	2.21e-03
5.0	3.25e-02	7.97e-03	2.00e-03

fx >>

Figure 1. The results of Euler's method and backward Euler using MATLAB

** **Figure 1**의 제 결과는 주교재인 **[1]**의 결과와 값이 어느 정도 다르게 나오지만 같은 저자(Kendall Atkinson)의 교재 중 **[2]**의 결과와 같으므로 **[2]**의 Table을 함께 제시합니다. 두 교재 모두 주어진 예제의 내용과 방법 모두 완전히 같게 서술되어 있고, Table 결과만 다르다고 판단됩니다. 이 점 확인하시어 부족한 부분이 있으면 지도 부탁드립니다.

Table 8.2 The method of lines: Euler's method ($h = \frac{1}{2}\delta^2$)

t	Error $m = 4$	Ratio	Error $m = 8$	Ratio	Error $m = 16$
1.0	$4.85e-2$	4.096	$1.18e-2$	4.024	$2.94e-3$
2.0	$4.39e-2$	4.096	$1.07e-2$	4.024	$2.66e-3$
3.0	$3.97e-2$	4.096	$9.69e-3$	4.024	$2.41e-3$
4.0	$3.59e-2$	4.096	$8.77e-3$	4.024	$2.18e-3$
5.0	$3.25e-2$	4.096	$7.93e-3$	4.024	$1.97e-3$

Table 8.3 The method of lines: Backward Euler method ($h = 0.1$)

t	Error $m = 4$	Error $m = 8$	Error $m = 16$
1.0	$4.85e-2$	$1.19e-2$	$2.99e-3$
2.0	$4.39e-2$	$1.08e-2$	$2.70e-3$
3.0	$3.98e-2$	$9.73e-3$	$2.45e-3$
4.0	$3.60e-2$	$8.81e-3$	$2.21e-3$
5.0	$3.25e-2$	$7.97e-3$	$2.00e-3$

Figure 2. The results of Euler's method and backward Euler in **[2]**

Comparison of the numerical stability between two methods

The result of the analysis of stability in Euler's method is

$$\left|1+h\lambda_j\right|<1, \qquad \lambda_j=-\frac{4}{\delta^2}\sin^2\left(\frac{j\pi}{2m}\right), \qquad j=1,\cdots,m-1.$$

Then we obtain the condition of h as $h \le \delta^2/2$. This implies that the error is proportional to δ^2, and this implies that the error should decrease by a factor of 4 when m is doubled. (refer to Table 6.20)

In contrast, the backward Euler method is **A-stable**, so there is no restriction for h. In our problem, we let the step size $h_E = \delta^2/2 \doteq 0.031, 0.0078, 0.0020$, $h_{BE} = 0.1$, each are the Euler, and backward Euler, respectively. Then $h_E < h_{BE}$ but backward Euler method is much more efficient if h is larger.

Table_6_20_HW6.m

```
%% MATH7003-00: Assignment #6-1, 2019310290 Sangman Jung.
%%% Method of Lines: Euler's method %%%
clear,clc
% exact solution U and the function G
U = @(x,t) exp(-0.1*t).*sin(pi*x);
G = @(x,t) U(x,t)*(pi^2-0.1);
% set the parameters
m = [4 8 16]; % the length of spatial variable x
delta = 1./m; % space step delta
% method of lines loop
for m_iter = 1:length(m) % m = 4, 8, 16
    x = (0:m(m_iter))*delta(m_iter); % calculate the variable x
    h = (delta(m_iter)^2)/2; % step size depend on delta
    t = 0:h:5; % time variable t
    A = zeros(m(m_iter)-1,m(m_iter)-1); % allocate the matrix 'LAMBDA'
    A = (1/delta(m_iter)^2)*(diag(diag(-2*ones(size(A)))) +...
        diag(ones(size(A,1)-1,1),-1) +...
        diag(ones(size(A,1)-1,1),1)); % the matrix LAMBDA
    d0 = U(x(1),t); d1 = U(x(end),t); % boundary value of U
    u0 = U(x(2:end-1),0)'; % initial value of U
    g = zeros(length(x)-2,length(t)); % allocate the function g
    g(1,:) = (d0/delta(m_iter)^2)+G(x(2),t);
    g(end,:) = (d1/delta(m_iter)^2)+G(x(end-1),t);
    g(2:end-1,:) = G(x(3:end-2)',t); % the function g
    % Euler's method
    V = zeros(length(x)-2,1); % allocate the value of V
    V(:,1) = u0; % initial condition for the method
    for n = 1:length(t)-1
        V(:,n+1) = V(:,n)+h*(A*V(:,n)+g(:,n)); % Euler's method
    end
    u = zeros(length(x),length(t)); % allocate the value of U
    u(1,:) = d0; u(end,:) = d1; % boundary condition
    u(2:end-1,:) = V; % save the value of V
    % recode the values
    xval(m_iter) = {x'};
    tval(m_iter) = {t};
    Uval(m_iter) = {u};
    Vval(m_iter) = {V};
end
% setting the values for table 6.20
x4 = cell2mat(xval(1));
t4 = cell2mat(tval(1));
u4 = cell2mat(Uval(1));
x8 = cell2mat(xval(2));
t8 = cell2mat(tval(2));
u8 = cell2mat(Uval(2));
x16 = cell2mat(xval(3));
t16 = cell2mat(tval(3));
u16 = cell2mat(Uval(3));
Table = 1:5;
for i = 1:5
    ind_t4(i) = find(t4 == i);
    ind_t8(i) = find(t8 == i);
    ind_t16(i) = find(t16 == i);
end
Error4 = max(abs(U(x4,t4(ind_t4))-u4(:,ind_t4)));
Error8 = max(abs(U(x8,t8(ind_t8))-u8(:,ind_t8)));
Error16 = max(abs(U(x16,t16(ind_t16))-u16(:,ind_t16)));
% Table 6.20
fprintf("Table 6.20 The method of lines: Euler's method\n")
fprintf('-------------------------------------------------------------\n')
fprintf('         Error             Error             Error \n')
fprintf('  t      m = 4     Ratio     m = 8     Ratio     m = 16\n')
fprintf('-------------------------------------------------------------\n')
for i = 1:5
    fprintf('% 1.1f   % 1.2e   % 1.2f   % 1.2e    % 1.2f    % 1.2e\n',...
        [Table(i) Error4(i) Error4(i)/Error8(i) Error8(i) Error8(i)/Error16(i) Error16(i)]);
end
fprintf('-------------------------------------------------------------\n\n\n')
```

Table_6_21_HW6.m

```
%% MATH7003-00: Assignment #6-2, 2019310290 Sangman Jung.
%%% Method of Lines: Backward Euler method %%%
clear
% exact solution U and the function G
U = @(x,t) exp(-0.1*t).*sin(pi*x);
G = @(x,t) U(x,t)*(pi^2-0.1);
% set the parameters
m = [4 8 16]; % the length of spatial variable x
delta = 1./m; % space step delta
% method of lines loop
for m_iter = 1:length(m) % m = 4, 8, 16
    x = (0:m(m_iter))*delta(m_iter); % calculate the variable x
    h = 0.1; % step size not depend on delta
    t = 0:h:5; % time variable t
    A = zeros(m(m_iter)-1,m(m_iter)-1); % allocate the matrix 'LAMBDA'
    A = (1/delta(m_iter)^2)*(diag(diag(-2*ones(size(A)))) +...
        diag(ones(size(A,1)-1,1),-1) +...
        diag(ones(size(A,1)-1,1),1)); % the matrix LAMBDA
    d0 = U(x(1),t); d1 = U(x(end),t); % boundary value of U
    u0 = U(x(2:end-1),0)'; % initial value of U
    g = zeros(length(x)-2,length(t)); % allocate the function g
    g(1,:) = (d0/delta(m_iter)^2)+G(x(2),t);
    g(end,:) = (d1/delta(m_iter)^2)+G(x(end-1),t);
    g(2:end-1,:) = G(x(3:end-2)',t); % the function g
    % backward Euler method
    I = diag(diag(ones(size(A)))); % identity matrix
    V = zeros(length(x)-2,1); % allocate the value of V
    V(:,1) = u0; % initial condition for the method
    for n = 1:length(t)-1
        V(:,n+1) = inv(I-h*A)*(V(:,n)+h*g(:,n+1)); % backward Euler method
    end
    u = zeros(length(x),length(t)); % allocate the value of U
    u(1,:) = d0; u(end,:) = d1; % boundary condition
    u(2:end-1,:) = V; % save the value of V
    % recode the values
    xval(m_iter) = {x'};
    tval(m_iter) = {t};
    Uval(m_iter) = {u};
    Vval(m_iter) = {V};
end
% setting the values for table 6.20
x4 = cell2mat(xval(1));
t4 = cell2mat(tval(1));
u4 = cell2mat(Uval(1));
x8 = cell2mat(xval(2));
t8 = cell2mat(tval(2));
u8 = cell2mat(Uval(2));
x16 = cell2mat(xval(3));
t16 = cell2mat(tval(3));
u16 = cell2mat(Uval(3));
Table = 1:5;
for i = 1:5
    ind_t4(i) = find(t4 == i);
    ind_t8(i) = find(t8 == i);
    ind_t16(i) = find(t16 == i);
end
Error4 = max(abs(U(x4,t4(ind_t4))-u4(:,ind_t4)));
Error8 = max(abs(U(x8,t8(ind_t8))-u8(:,ind_t8)));
Error16 = max(abs(U(x16,t16(ind_t16))-u16(:,ind_t16)));
% Table 6.21
fprintf("Table 6.21 The method of lines: backward Euler's method\n")
fprintf('-----------------------------------------------------\n')
fprintf('           Error          Error         Error \n')
fprintf('    t      m = 4          m = 8         m = 16\n')
fprintf('-----------------------------------------------------\n')
for i = 1:5
    fprintf('  % 1.1f    % 1.2e       % 1.2e       % 1.2e\n',...
        [Table(i) Error4(i) Error8(i) Error16(i)]);
end
fprintf('-----------------------------------------------------\n')
```

References.

[1] Atkinson, K. E. (2008). An introduction to numerical analysis. John wiley & sons.

[2] Atkinson, K., Han, W., & Stewart, D. E. (2011). Numerical solution of ordinary differential equations (Vol. 108). John Wiley & Sons.

[3] Atkinson, K. E., & Han, W. (1985). Elementary numerical analysis (p. 17). New York et al.: Wiley.

[4] Brugnano, L., & Trigiante, D. (1992). Tridiagonal matrices: invertibility and conditioning. Linear algebra and its applications, 166, 131-150.