THE UNIVERSITY OF MELBOURNE

Numerical Programming for Engineers

ENGR30003

Assignment 2

Jiawei Liao 756560

October 2017

2. Rootfinding

2.1 Analytical solution for $\theta = 0^o$

For $\theta = 0^o$, i.e. the object would be a flat plate, show that two possible solutions for β are

$$\beta_L = \arcsin\left(\frac{1}{M}\right), \quad \beta_U = 90^o \tag{3}$$

β_U and β_L are usually called the strong shock and weak shock solutions, respectively. Even though we can mathematically obtain two possible solutions, in reality, or physically, only the weak shock solution occurs. Note also that in order for the solution to be physically realizable, $\theta < \beta < 90^o$.

Sol: Consider equation (1):

$$\tan(\theta) = 2\cot(\beta)\frac{M^2\sin^2(\beta) - 1}{M^2(\gamma + \cos(2\beta)) + 2}, \quad \theta < \beta < 90^o$$

Substitute $\theta = 0^o$ into the equation yields:

$$0 = 2\cot(\beta)\frac{M^2\sin^2(\beta) - 1}{M^2(\gamma + \cos(2\beta)) + 2}, \qquad 0^o < \beta < 90^o$$

Which requires that:

$$2\cot(\beta)\,(M^2\sin^2(\beta) - 1) = 0$$

And

$$M^2(\gamma + \cos(2\beta)) + 2 \neq 0$$

We know that the denominator is not zero as $\gamma = 1.4$ for air, and $-1 \leq \cos(2\beta) \leq 1$, And M^2 is strictly positive, thus $M^2(\gamma + \cos(2\beta)) + 2 \neq 0$ holds.

For

$$2\cot(\beta)\,(M^2\sin^2(\beta) - 1) = 0$$

Which can be further factorized as

$$2\cot(\beta)\,(M\sin(\beta) + 1)(M\sin(\beta) - 1) = 0$$

$\Rightarrow$

$$2\cot(\beta) = 0 \;\; or \;\; (M\sin(\beta) + 1) = 0 \;\; or \;\; (M\sin(\beta) - 1) = 0$$

Case 1:

$$2\cot(\beta) = 0$$

$\Rightarrow$

$$\cot(\beta) = 0$$

$\Rightarrow$

$$\beta = 90^o \quad for\ 0^o < \beta < 90^o$$

This is the strong shock solution β_U.

Case 2:

$$(M\sin(\beta) + 1) = 0$$

$\Rightarrow$

$$\sin(\beta) = \frac{-1}{M}$$

$\Rightarrow$

$$no\ solution\ within\ 0^o < \beta < 90^o$$

This solution is rejected as β is negative, which does not satisfy $\theta < \beta < 90^o$.

Case 3:

$$(M\sin(\beta) - 1) = 0$$

$\Rightarrow$

$$\sin(\beta) = \frac{1}{M}$$

$\Rightarrow$

$$\beta = \arcsin\left(\frac{1}{M}\right)\ for\ 0^o < \beta < 90^o$$

This is the weak shock solution β_L

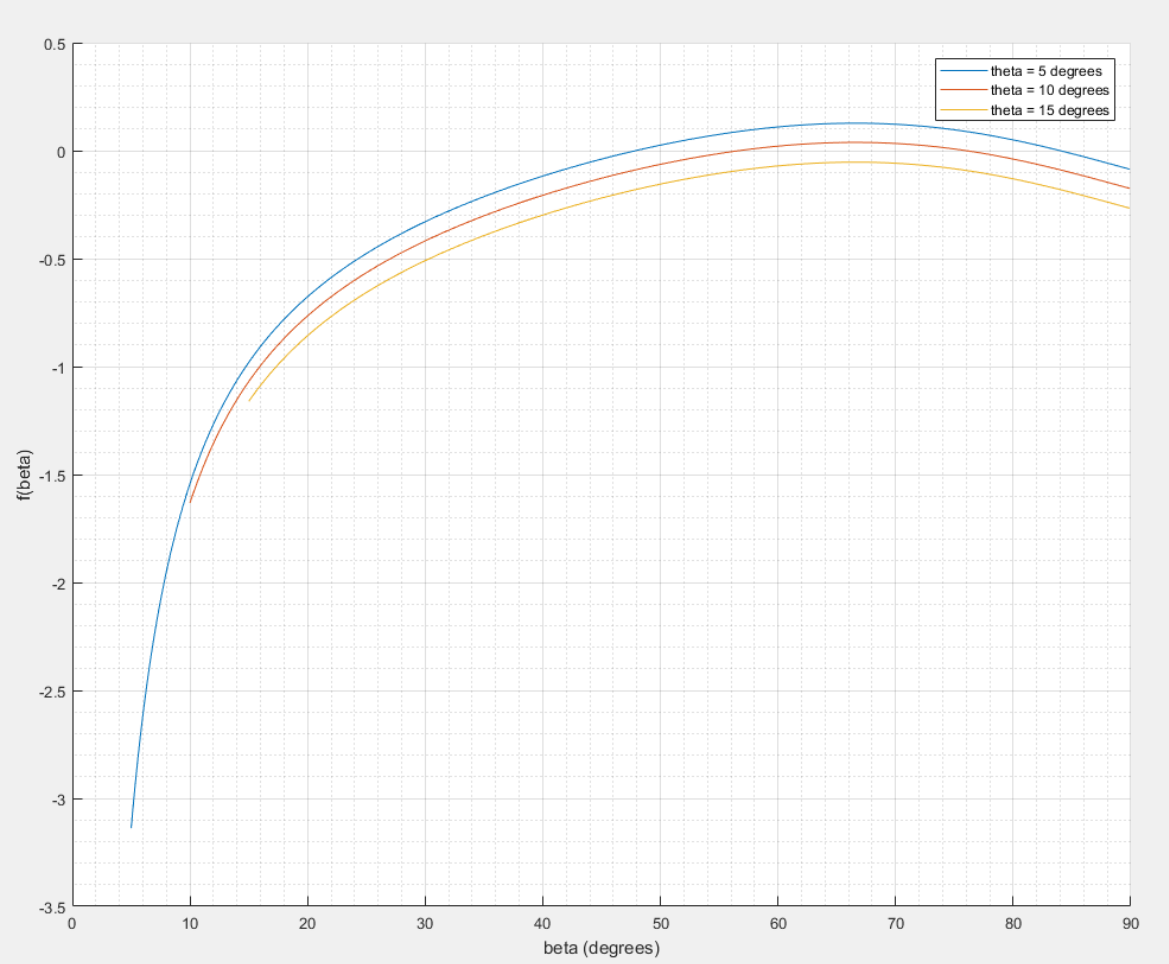
Thus:

$\beta_L = \arcsin\left(\frac{1}{M}\right)$, $\beta_U = 90^o$ are the two possible solutions for β for $\theta = 0^o$

2.2 Graphical Solution

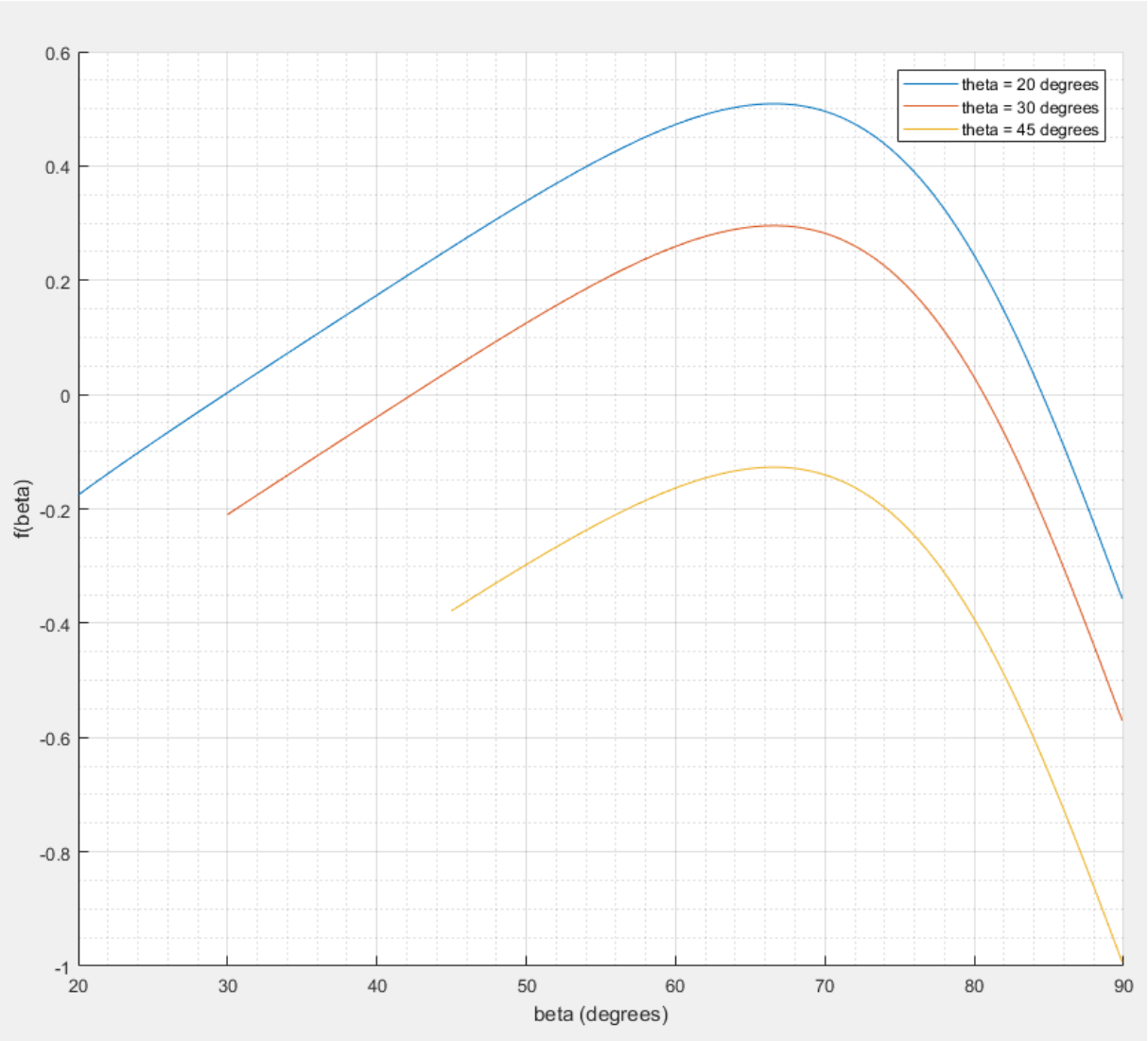
Plot $f(\beta)$ vs β for

a. $M = 1.5\ and\ \theta = 5^o, 10^o\ and\ 15^o$



It can be seen for this M value, as θ increases from 5^o to 10^o, β_U decreases and β_L increases. For $\theta = 15^o$, no solutions of β_L and β_U exist as the graph no longer intersects with the $f(\beta) = 0$ axis, indicating that the shock is detached. Thus $10^o < \theta_{max} < 15^o$, and can be estimated to be around 12.5^o

b. $M = 5.0\ and\ \theta = 20^o, 30^o\ and\ 45^o$



It can be seen for this M value, as θ increases from 20^o to 30^o , β_U decreases and β_L increases. For $\theta = 45^o$, no solutions of β_L and β_U exist as the graph no longer intersects with the $f(\beta) = 0$ axis, indicating that the shock is detached. Thus $30^o < \theta_{max} < 45^o$, and can be estimated to be around 35^o

Now, compare the two graphs of different M values, for M = 1.5, no solutions are present for $\theta = 15^o$, yet the M = 5.0 graph shows solutions for $\theta = 15^o$. This indicates that, for a fixed θ value, increasing the value of M will increase β_U and decrease β_L. Increasing M also has the effect of increasing θ_{max}.
For a fixed M value, increasing the value of θ will decrease β_U and increase β_L before θ_{max} is reached.

Matlab Code used for 2.2a

Editor - G:\2017_S2\ENGR30003\Ass2\Q2_2a.m

Q2_2a.m | Q2_2b.m

```
M = 1.5;
gamma = 1.4;
theta = [5, 10, 15];
beta = zeros(1,900);
fbeta = zeros(1,900);
betamin = 0.1;
for n = 1:3
    betamin = theta(n);
    beta = betamin:0.1:89.9;
    fbeta= 2.*cotd(beta).*(M^2*(sind(beta)).^2-1)./(M^2*(gamma+cosd(2.*beta))+2)-tand(betamin);
    hold on;
    grid on;
    grid minor;
    plot(beta,fbeta);
end
hold off;
xlabel('beta (degrees)');
ylabel('f(beta)');
legend('theta = 5 degrees','theta = 10 degrees','theta = 15 degrees');
```

Matlab Code used for 2.2b

Editor - G:\2017_S2\ENGR30003\Ass2\Q2_2b.m

Q2_2a.m | Q2_2b.m

```
M = 5.0;
gamma = 1.4;
theta = [20, 30, 45];
beta = zeros(1,900);
fbeta = zeros(1,900);
betamin = 0.1;
for n = 1:3
    betamin = theta(n);
    beta = betamin:0.1:89.9;
    fbeta= 2.*cotd(beta).*(M^2*(sind(beta)).^2-1)./(M^2*(gamma+cosd(2.*beta))+2)-tand(betamin);
    fdot = diff(fbeta);
    hold on;
    grid on;
    grid minor;
    plot(beta,fbeta);
    %plot(beta(1:699),fdot);
end
hold off;
xlabel('beta (degrees)');
ylabel('f(beta)');
legend('theta = 20 degrees','theta = 30 degrees','theta = 45 degrees');
```

2.3 C program to solve shock-wave equation

(a)

The newton Raphson method requires an analytical expression of the first order derivative of $f(\beta)$ with respect to β.

$$f(\beta) = 2\cot(\beta)\frac{M^2\sin^2(\beta) - 1}{M^2(\gamma + \cos(2\beta)) + 2} - \tan(\theta)$$

$$\frac{\delta(f(\beta))}{\delta\beta} = \frac{\delta}{\delta\beta}(2\cot(\beta)\frac{M^2\sin^2(\beta) - 1}{M^2(\gamma + \cos(2\beta)) + 2})$$

By using the quotient rule of differentiation

$$\text{Let } g(\beta) = (2\cot(\beta)\cdot(M^2\sin^2(\beta) - 1)$$
$$= M^2\sin(2\beta) - 2\cot(\beta)$$

$$g'(\beta) = 2M^2\cos(2\beta) + 2csc^2(\beta)$$

$$and\ h(\beta) = M^2(\gamma + \cos(2\beta)) + 2$$

$$h'(\beta) = -2M^2\sin(2\beta)$$

And then

$$\frac{\delta(f(\beta))}{\delta\beta} = \frac{g'(\beta)\cdot h(\beta) - h'(\beta)\cdot g(\beta)}{[h(\beta)]^2}$$

$$= \frac{(2M^2\cos(2\beta) + 2csc^2(\beta))(M^2(\gamma + \cos(2\beta)) + 2) + (2M^2\sin(2\beta))((2\cot(\beta)\cdot(M^2\sin^2(\beta) - 1))}{(M^2(\gamma + \cos(2\beta)) + 2)^2}$$

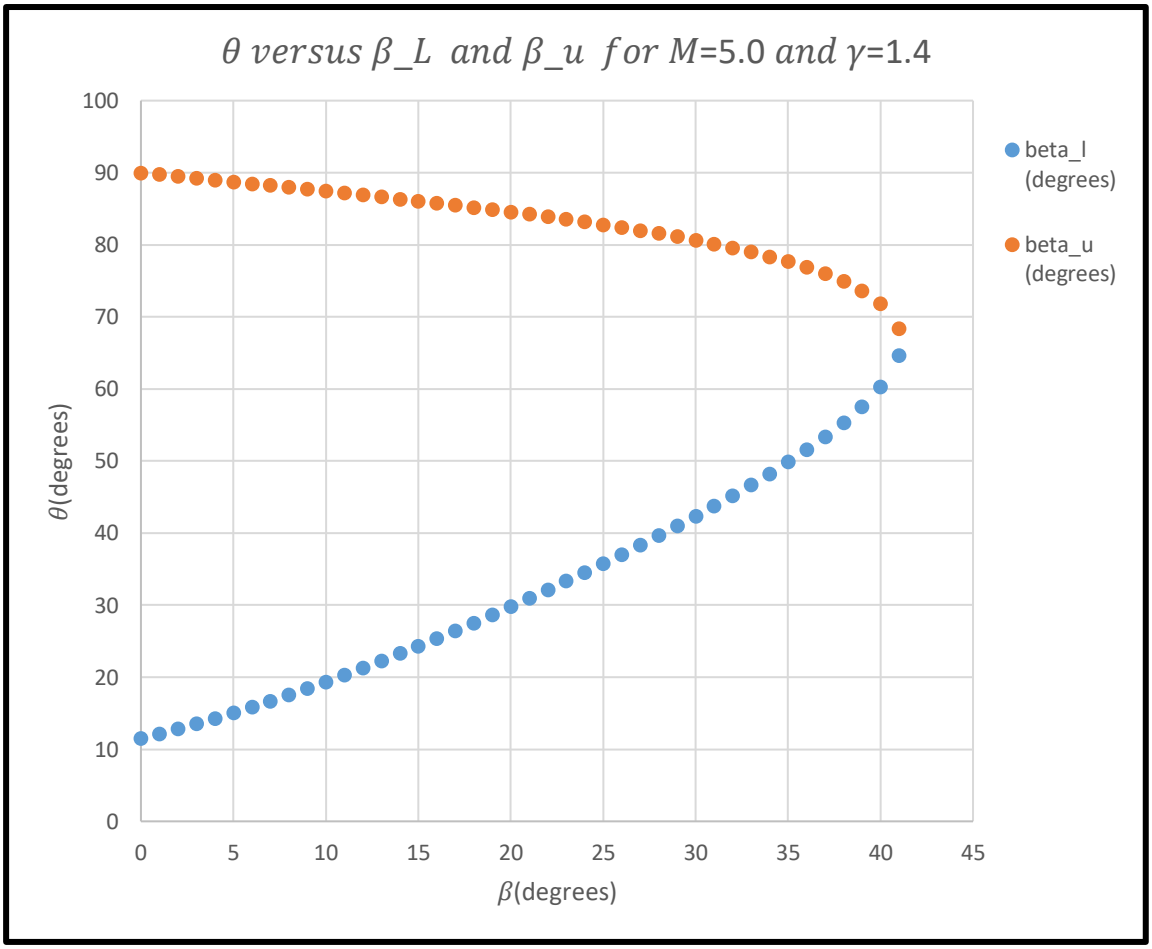
Now, for $M = 5.0\ and\ \theta = 20^o$, refer to the graph in 2.2b, it can be seen with the assist of the grid that the β_L value is around 30, and is definitely greater than 28, whereas the β_u value is around 85, and definitely below 86. So it is safe to use 28 and 86 as the initial guesses for β_L and β_u respectively.

And the terminal output confirms the answer from the original handout.

```
admin@DESKTOP-8JCNDTK /c/c
$ ass2_1 in_shock.csv in_linalsys.csv in_interp.csv 5 in_heateqn.csv
beta_l is 29.800916
beta_u is 84.556255
```

(b)

This part's data is directly drawn from the output file "out_shock.csv" in part c.
The graph is drawn using Microsoft Excel.
Some further information on the c code structure can be found in (c) and the code comments.

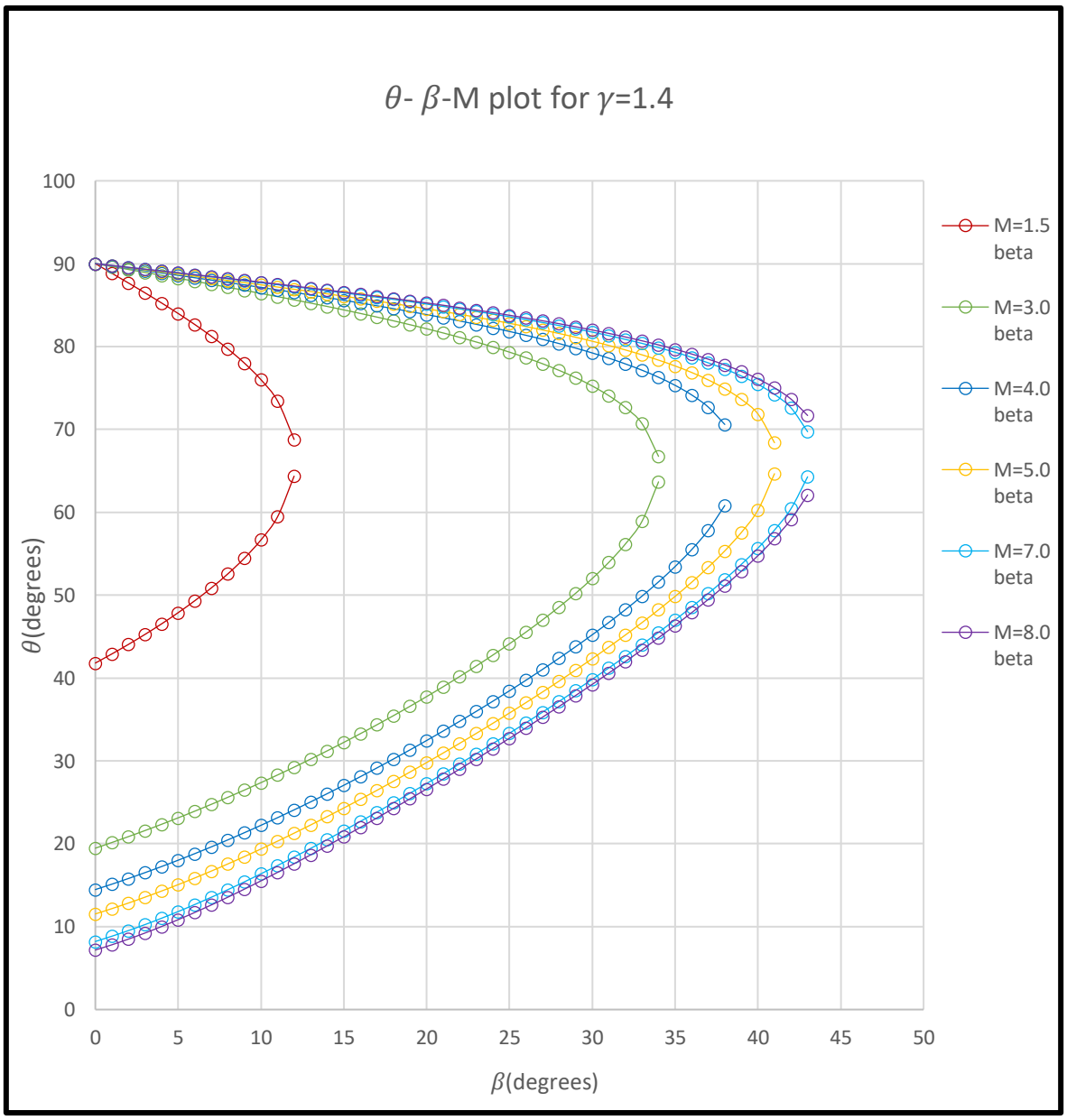


(c)

Newton Raphson method fails when it reaches a point of zero derivative and can sometimes give meaningless solutions. So, it is necessary to do sanity checks on the results.

For instance, we would like the solutions satisfy $0^o < \beta < 90^o$,

So this rule is used as a sanity check along with the boundaries to determine when θ_{max} is reached.



3. Linear Algebraic Systems

Consider the following tri-diagonal system:

$$\begin{bmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ c_2 & a_2 & b_2 & 0 & \cdots & 0 \\ 0 & c_3 & a_3 & b_3 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c_{N-1} & a_{N-1} & b_{N-1} \\ 0 & \cdots & 0 & 0 & c_N & a_N \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ \vdots \\ \vdots \\ Q_N \end{Bmatrix}$$

Now try to apply Gauss elimination on the above matrix to reduce it to row echelon form.

$$\begin{bmatrix} a_1^* & b_1^* & 0 & 0 & \cdots & 0 \\ 0 & a_2^* & b_2^* & 0 & \cdots & 0 \\ 0 & 0 & a_3^* & b_3^* & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & a_{N-1}^* & b_{N-1}^* \\ 0 & \cdots & 0 & 0 & 0 & a_N^* \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} Q_1^* \\ Q_2^* \\ Q_3^* \\ \vdots \\ \vdots \\ Q_N^* \end{Bmatrix}$$

Row 1 remains unchanged in Gauss elimination.

$$a_1^* = a_1\,, \qquad b_1^* = b_1$$

For Row 2: c2 can be eliminated by subtracting $\frac{c_2}{a_1^*}$ times Row 1 from Row 2.

$$a_2^* = a_2 - \frac{c_2}{a_1^*} b_1, \qquad b_2^* = b_2 - \frac{c_2}{a_1^*} \cdot 0 = b_2, \quad Q_2^* = Q_2 - \frac{c_2}{a_1^*} Q_1$$

For Row 3: c3 can be eliminated by subtracting $\frac{c_3}{a_2^*}$ times Row 2 from Row 3

$$a_3^* = a_3 - \frac{c_3}{a_2^*} b_2, \qquad b_3^* = b_3 - \frac{c_3}{a_2^*} \cdot 0 = b_3, \quad Q_3^* = Q_3 - \frac{c_3}{a_2^*} Q_2$$

An observable pattern emerges for a_i^*, b_i^*, Q_i^*, for $i > 1$

$$for\ all\ a_i^*\ where\ i > 1, \qquad a_i^* = a_i - \frac{c_i}{a_{i-1}^*} b_{i-1}$$

$$for\ all\ b_i^*\ where\ i > 1, \qquad b_i^* = b_i - \frac{c_i}{a_{i-1}^*} \cdot 0 = b_i$$

Note that $b_i^* = b_i$ for all values of i since $b_1^* = b_1$

$$for\ all\ Q_i^*\ where\ i > 1, \qquad Q_i^* = Q_i - \frac{c_i}{a_{i-1}^*} Q_{i-1}^*$$

So in conclusion:
The matrix will have form after the Gauss Elimination as

$$\begin{bmatrix} a_1^* & b_1 & 0 & 0 & \cdots & 0 \\ 0 & a_2^* & b_2 & 0 & \cdots & 0 \\ 0 & 0 & a_3^* & b_3 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & a_{N-1}^* & b_{N-1} \\ 0 & \cdots & 0 & 0 & 0 & a_N^* \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} Q_1^* \\ Q_2^* \\ Q_3^* \\ \vdots \\ \vdots \\ Q_N^* \end{Bmatrix}$$

Where

$$a_i^* = \begin{cases} a_i\,, & for \;\; i = 1 \\ a_i - \dfrac{c_i b_{i-1}}{a_{i-1}^*} & for\; i = 2,3,\dots,N \end{cases}$$

$$Q_i^* = \begin{cases} Q_i\,, & for \;\; i = 1 \\ Q_i - \dfrac{c_i Q_{i-1}^*}{a_{i-1}^*} & for\; i = 2,3,\dots,N \end{cases}$$

Now, the solution x_i can be found by using backward substitution. Starting from the last row

$$a_N^* \cdot x_N = Q_N^*$$
$$x_N = Q_N^*/a_N^*$$

And then the second last row

$$a_{N-1}^* \cdot x_{N-1} + b_{N-1} \cdot x_N = Q_{N-1}^*$$
$$x_{N-1} = (Q_{N-1}^* - b_{N-1} \cdot x_N)/\, a_{N-1}^*$$

And then the third last row

$$a_{N-2}^* \cdot x_{N-2} + b_{N-2} \cdot x_{N-1} = Q_{N-2}^*$$
$$x_{N-2} = (Q_{N-2}^* - b_{N-2} \cdot x_{N-1})/\, a_{N-2}^*$$

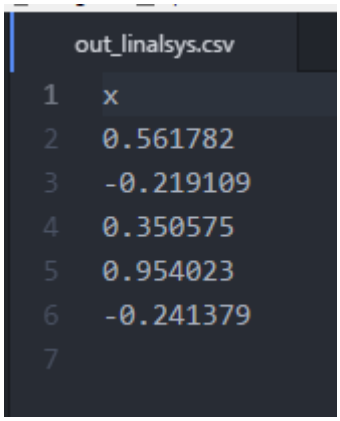
The pattern is obvious, and for each row, only x_i from the previous row is needed, since only two elements are non-zero in each row, and only one of them above a non-zero element from the previous row.

$$x_i = \begin{cases} Q_i^*/a_i^* & for \;\; i = N \\ \dfrac{Q_i^* - b_i\, x_{i+1}}{a_i^*} & for\; i = N-1, N-2, \dots, 1 \end{cases}$$

Using the Thomas Algorithm to apply on the tri-diagonal system

$$\begin{bmatrix} 1 & -2 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 \\ 0 & 8 & -9 & 2 & 0 \\ 0 & 0 & 6 & 3 & 4 \\ 0 & 0 & 0 & 6 & 3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ -3 \\ 4 \\ 5 \end{Bmatrix}$$

The results obtained from the C-code:



Matches the results from the handout.

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{Bmatrix} = \begin{Bmatrix} 0.561782 \\ -0.219109 \\ 0.350575 \\ 0.954023 \\ -0.241379 \end{Bmatrix}$$

4. Regression

From the lectures, this is the system of equations for linear regression $y = ax + b$.

$$\begin{bmatrix} \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & N \end{bmatrix} \begin{Bmatrix} a \\ b \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^{N} x_i y_i \\ \sum_{i=1}^{N} y_i \end{Bmatrix}$$

$$\begin{Bmatrix} a \\ b \end{Bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & N \end{bmatrix}^{-1} \begin{Bmatrix} \sum_{i=1}^{N} x_i y_i \\ \sum_{i=1}^{N} y_i \end{Bmatrix}$$

$$\begin{Bmatrix} a \\ b \end{Bmatrix} = \frac{1}{N\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} x_i} \begin{bmatrix} N & -\sum_{i=1}^{N} x_i \\ -\sum_{i=1}^{N} x_i & N\sum_{i=1}^{N} x_i^2 \end{bmatrix} \begin{Bmatrix} \sum_{i=1}^{N} x_i y_i \\ \sum_{i=1}^{N} y_i \end{Bmatrix}$$

$$thus\ \ a = \frac{1}{N\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} x_i} \left(N\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} y_i\right)$$

Now, need to refer to the basic definition of summation

$$\sum_{i=1}^{N} x_i = x_1 + x_2 + \cdots + x_N$$

1. And from this definition, we can prove that for a constant real coefficient a:

$$\begin{aligned} \sum_{i=1}^{N} ax_i &= ax_1 + ax_2 + \cdots + ax_N \\ &= a(x_1 + x_2 + \cdots + x_N) \\ &= a\sum_{i=1}^{N} x_i \end{aligned}$$

2. And we can also prove that

$$\begin{aligned} \sum_{i=1}^{N} (x_i + y_i) &= x_1 + y_1 + x_2 + y_2 + \cdots + x_N + y_N \\ &= (x_1 + x_2 + \cdots + x_N) + (y_1 + y_2 + \cdots + y_N) \\ &= \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i \end{aligned}$$

Now, we can continue with

$$a = \frac{N\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} y_i}{N\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} x_i}$$

$$= \frac{\sum_{i=1}^{N} x_i y_i - \frac{1}{N}\sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} y_i}{\sum_{i=1}^{N} x_i^2 - \frac{1}{N}\sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} x_i}$$

$$= \frac{\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} \frac{y_i}{N}}{\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} \frac{x_i}{N} \cdot \sum_{i=1}^{N} x_i} \qquad by\ \ 2.$$

$$= \frac{\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \cdot \bar{y}}{\sum_{i=1}^{N} x_i^2 - \bar{x} \cdot \sum_{i=1}^{N} x_i} \quad by\ def.\ mean\ \bar{x} = \sum_{i=1}^{N} x_i/N$$

$$= \frac{\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i\, \bar{y}}{\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} \bar{x} x_i} \qquad by\ \ 2.$$

$$= \frac{\sum_{i=1}^{N} (x_i y_i - x_i \bar{y})}{\sum_{i=1}^{N} (x_i^2 - \bar{x} x_i)} \qquad by\ \ 1.$$

$$= \frac{\sum_{i=1}^{N} x_i (y_i - \bar{y})}{\sum_{i=1}^{N} x_i (x_i - \bar{x})}$$

We know show that $\sum_{i=1}^{N}(x_i - \bar{x}) = 0$
We know that $\bar{x} \cdot N = \sum_{i=1}^{N} x_i$
By 1, $\sum_{i=1}^{N}(x_i - \bar{x}) = \sum_{i=1}^{N} x_i - \sum_{i=1}^{N} \bar{x} = \sum_{i=1}^{N} x_i - \bar{x} \cdot N = \sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i = 0$
And by 2, $a\sum_{i=1}^{N}(x_i - \bar{x}) = 0$ where a can be any real constant.

So we can subtract the term which equal to zero to top and bottom of the fraction
Not that the both extra term equal to zero

$$a = \frac{\sum_{i=1}^{N} x_i (y_i - \bar{y}) - \sum_{i=1}^{N} \bar{x}(y_i - \bar{y})}{\sum_{i=1}^{N} x_i (x_i - \bar{x}) - \sum_{i=1}^{N} \bar{x}(x_i - \bar{x})}$$

$$= \frac{\sum_{i=1}^{N} (x_i (y_i - \bar{y}) - \bar{x}(y_i - \bar{y}))}{\sum_{i=1}^{N} (x_i (x_i - \bar{x}) - \bar{x}(x_i - \bar{x}))} \qquad by\ \ 1.$$

$$= \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N} (x_i - \bar{x})(x_i - \bar{x})}$$

$$= \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N} (x_i - \bar{x})^2} \qquad as\ required$$

Now for b, it's no longer needed to use the inverse expression, we can simply look at the original matrix system.

$$\sum_{i=1}^{N} x_i \cdot a + N \cdot b = \sum_{i=1}^{N} y_i$$

$$N \cdot b = \sum_{i=1}^{N} y_i - \sum_{i=1}^{N} x_i \cdot a$$

$$b = \frac{1}{N}\sum_{i=1}^{N} y_i - \frac{1}{N}\sum_{i=1}^{N} x_i \cdot a$$

$$= \frac{1}{N}\sum_{i=1}^{N} y_i - \frac{1}{N}\sum_{i=1}^{N} x_i \cdot a$$

$$= \sum_{i=1}^{N} \frac{y_i}{N} - a \cdot \sum_{i=1}^{N} \frac{x_i}{N} \quad by\ 2.$$

$$= \bar{y} - a \cdot \bar{x} \qquad by\ def.\ mean\ \bar{x} = \sum_{i=1}^{N} x_i/N$$

So it can be shown that $b = \bar{y} - a \cdot \bar{x}$ and $a = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$ as required

By considering Eq.9, the linear regression will fail to find a solution if $\sum_{i=1}^{N}(x_i - \bar{x})^2$ happen to be zero. The computation done may not be exact, and it will possibly produce a very large number for a and b and make the results erroneous

5. Interpolation

Given the following data

$$x_0 = 0 \quad f(x_0) = 0$$
$$x_1 = 1 \quad f(x_1) = 11$$
$$x_2 = 3 \quad f(x_2) = 17$$
$$x_3 = 8 \quad f(x_3) = 17$$

Use both the second-order Lagrange interpolating polynomials and cubic splines to estimate $f(x)$. Estimate the value of $f(x = 5)$ using both methods.

Second-order Lagrange:

I made my C-program general and it can take as many data points (at least 3) as the RAM can handle but the second-order Lagrange interpolating are restricted to only three points. Then, the safest way is to find the three points that are closest to xo and use them as for the 2nd order Lagrange interpolation. In this case, the last three points.

Consider input x_i is in increasing order, and $x_1 \le xo \le x_{N-2}$,
Then we can find index $j, 1 \le j \le N-2$, that $x_j \le xo \le x_{j+1}$,
So x_j, x_{j+1} are the two points to use, and the last one would either be $x_{j-1}\ or\ x_{j+2}$, whichever is closer to xo.
If $xo < x_1$, then obviously the first three points are to be chosen.
If $xo > x_{N-2}$, then the last three points are to be chosen.
This allows us to interpolate three points for xo for number of points more or equal to three. For this set of data, the last three points are chosen using this algorithm.

An nth-order Lagrange interpolation will require (n+1) data points, and uses them to compute n Lagrange multipliers $L_i(x)$.
The general formula is (from (21) in interpolation slides):

$$f_n(x) = \sum_{i=0}^{n} L_i(x)\, f(x_i)$$

The value of n is the order of Lagrange interpolation, where $f(x_i)$ is the $f(x)$ values at the known data points (from (22) in interpolation slides):

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_i}{x_i - x_j}$$

For a second-order Lagrange interpolating polynomial, the $L_i(x)\ is$ defined as:

$$L_0(x) = \begin{cases} 1 & if\ x = x_0 \\ 0 & if\ x = x_1 \\ 0 & if\ x = x_2 \end{cases}$$

$$L_1(x) = \begin{cases} 0 & if\ x = x_0 \\ 1 & if\ x = x_1 \\ 0 & if\ x = x_2 \end{cases}$$

$$L_2(x) = \begin{cases} 0 & if\ x = x_0 \\ 0 & if\ x = x_1 \\ 1 & if\ x = x_2 \end{cases}$$

And thus:

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

$$L_1(x) = \frac{(x - x_2)(x - x_0)}{(x_1 - x_2)(x_1 - x_0)}$$

$$L_2(x) = \frac{(x - x_1)(x - x_0)}{(x_2 - x_1)(x_2 - x_0)}$$

So

$$f_n(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x)$$

Cubic spline:

Cubic spline aims to connect each two adjacent data points with a cubic function, and each cubic function is only defined for its interval, which includes the space between the two adjacent points and the two end points.

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

$$S(x) = S_i(x) \quad for \quad x_i < x < x_{i+1}$$

The total function is continuous, and also its first order and second order derivatives.

1. At $x = x_i$,

$$S_i(x) = a_i = f(x_i) \qquad (34)$$

Where $i = 0 \ldots n-1, n = number\ of\ data\ points$

2. The function values at adjacent polynomials must be equal at the interior knots.

$$S_i(x_{i+1}) = S_{i+1}(x_{i+1})$$
$$a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 = a_{i+1}$$

$Let\ h_i = x_{i+1} - x_i,$

$$a_i + b_i h_i + c_i {h_i}^2 + d_i {h_i}^3 = a_{i+1} \quad (35)$$

3. Derivative of $S(x)$ is continuous.

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \quad (36)$$

Which gives

$$b_i + 2c_i h_i + 3d_i {h_i}^2 = b_{i+1} \quad (37)$$

4. Double derivative of $S(x)$ is continuous.

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \quad (38)$$

Which gives

$$c_i + 3d_i h_i = c_{i+1} \quad (39)$$

Now, there are two types of cubic splines to choose from, the former being the clamped spline and the latter being the natural spline. Since we do not know additional information about the function, I will use the natural spline method, which assumes the double derivative at the two end points to be zero.
Which leads to:

$$c_0 = 0 \quad (40)$$
$$c_n = 0 \quad (41)$$

These conditions give 4n equations, such that all a_i, b_i, c_i, d_i can be solved.

Rearrange (39):

$$d_i = \frac{c_{i+1} - c_i}{3h_i} \quad (42)$$

Put (42) into (37):

$$b_{i+1} = b_i + (c_{i+1} - c_i)\, h_i \quad (43)$$

Put (42) into (35):

$$b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(2c_i + c_{i+1}) \qquad (44)$$

Put (44) into (43):

$$h_{j-1}c_{j-1} + 2(h_j + h_{j-1})c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) + \frac{3}{h_{j-1}}(a_{j-1} - a_j) \qquad (45)$$

Equation (45) can be represented by a tri-diagonal matrix system as in Task 3.
Since the C-programming language uses index 0 as the starting point, I will modify equations from task 2 to suit the mind better.
And for getting rid of confusion, all the terms for tridiagonal representation are now written in capital.

For N number of points

$$A_i = \begin{cases} 1, & for \;\; i = 0, N-1 \\ 2(h_{i-1} + h_i), & for\, i = 2,3, \dots, N-1 \end{cases}$$

$$B_i = \begin{cases} 0, & for \;\; i = 0, N-1 \\ h_i, & for\, i = 2,3, \dots, N-1 \end{cases}$$

$$C_i = \begin{cases} 0, & for \;\; i = 0, N-1 \\ h_{i-1}, & for\, i = 2,3, \dots, N-1 \end{cases}$$

$$Q_i = \begin{cases} 0, & for \;\; i = 0, N-1 \\ \frac{3}{h_i}(a_{i+1} - a_i) + \frac{3}{h_{i-1}}(a_{i-1} - a_i), & for\, i = 2,3, \dots, N-1 \end{cases}$$

X_i *contains the values of* c_i *, and we know* $c_0, c_{N-1} = 0$
This can be solved to obtain the values of c_i.
It must be noted that a_i and c_i have N elements, where the other variables have one less element.

Once c_i are solved, use equation (42) and (44) to obtain d_i and b_i.

Cubic natural splines require at least three points to compute a result.

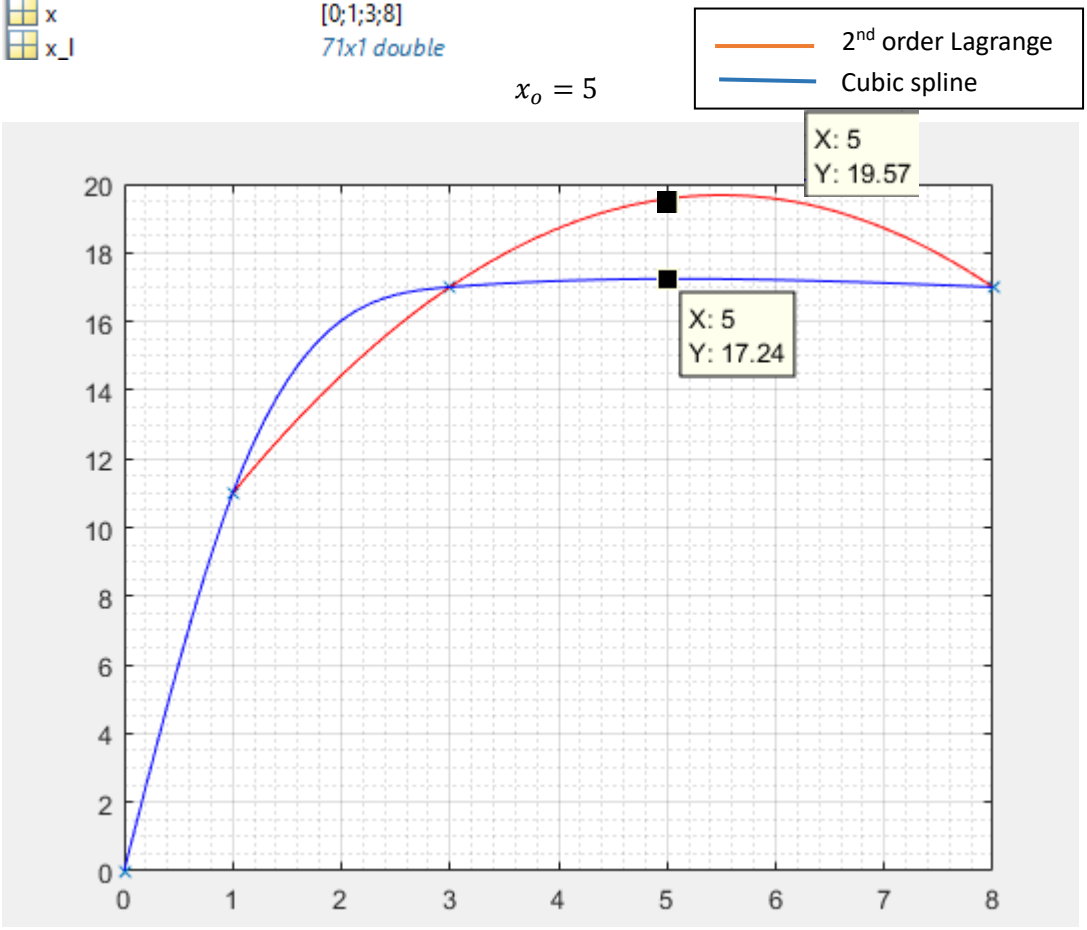
Results and outputs:

For the interpolated results, a step size of 0.1 is chosen for the x-values. The interpolated f(x) values are directly computed in c and outputted to a file where areas the coefficients of the cubic functions are computed in c for cubic spline and outputted to another file. Technically my code supports more than 4 input points and interpolate those using cubic splines. Yet, only the last three points will be chosen to do LaGrange as 2nd order LaGrange can only use three.

Workspace

Name	Value
a	[0;11;17]
b	[12.3250;8.3500;0.2500
c	[0;-3.9750;-0.0750]
d	[-1.3250;0.6500;0.0050
fx	[0;11;17;17]
fx_l	*71x1 double*
x	[0;1;3;8]
x_l	*71x1 double*

$x_o = 5$



The 4 decimal place value is $f_2(5) = 19.5714,\ S_2(5) = 17.2400$

It can be easily seen that the spline interpolation looks much smoother than the second-order

Lagrange interpolation polynomial. And since the second-order Lagrange can only use three points, it looks a bit off as it cannot take the (0,0) point into account. So for this data set cubic spline is definitely more convincing than the 2nd order Lagrange method.

6. Differentiation, differential equations

6.1:

The heat equation:

$$\frac{\partial f}{\partial t} = \mu \frac{\partial^2 f}{\partial x^2}$$

Can also be written as:

$$\frac{\partial f}{\partial t} = RHS(f)$$

Using the spatial interval $0 \leq x \leq 1$ and the time interval $0 \leq t \leq 2$ where x, t are discretized using $Nx\ and\ Nt$, the differential equation can be numerically solved using two Euler schemes and two boundary conditions.

$$f_i^{n+1} = f_i^n + \Delta t \cdot RHS(f_i^n) \qquad (explicit)$$
$$f_i^{n+1} = f_i^n + \Delta t \cdot RHS\left(f_i^{n+1}\right) \qquad (implicit)$$

Where n denotes the time step index and i denotes the x step index.
And the value of RHS can be approximated by

$$\mu \frac{\partial^2 f}{\partial x^2} \approx \mu \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} \quad for\ i = 1, \dots, Nx - 1$$

And the two boundary conditions to be used are respectively:

Fixed Ends:
For $i = 0\ and\ i = Nx$, the value of $\mathrm{RHS}(f_i)$ is 0.0.

Variable Ends:
For $i = 0$, use the boundary stencil

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_i - 2f_{i+1} + f_{i+2}}{\Delta x^2}$$

For $i = 0$, use the boundary stencil

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_i - 2f_{i-1} + f_{i-2}}{\Delta x^2}$$

Initial conditions:
The initial condition for f is

$$\begin{aligned} 0 \leq x < 0.125 &\rightarrow f(x, t=0) = 0 \\ 0.125 \leq x \leq 0.375 &\rightarrow f(x, t=0) = 0.5[1 - \cos\{8\pi(x - 0.125)\}] \\ 0.375 < x \leq 1 &\rightarrow f(x, t=0) = 0 \end{aligned}$$

The results of explicit methods can be computed directly from the results of the previous time step iteration. However, it requires a more complex approach to obtain the results of the next time step iteration using the implicit method.

For explicit method:

$$f_i^{n+1} = \Delta t \cdot RHS\left(f_i^{n+1}\right)$$

And thus

$$f_i^{n+1} \approx \begin{cases} f_i^n & if\ i = 0\ \ (x = 0) \\ f_i^n + \Delta t \cdot \mu \cdot \dfrac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{\Delta x^2}, & for\ i = 1, \dots, Nx - 1 \\ f_i^n & if\ i = Nx\ (x = 1) \end{cases}$$

From the initial conditions we know that

$$f(x = 0, t = 0) = 0$$
$$f(x = 1, t = 0) = 0$$

And since $f_i^{n+1} = f_i^n\ \ for\ the\ x\ values\ at\ the\ two\ ends\ points\ 0, 1$

$$f(x = 0) = 0\ \ for\ all\ time\ iterations$$
$$f(x = 1) = 0\ \ for\ all\ time\ iterations$$

Now for the x values in between

$$f_i^{n+1} \approx f_i^n + \Delta t \cdot \mu \cdot \frac{f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}}{\Delta x^2}$$

$$f_i^{n+1} \approx f_i^n + \frac{\Delta t \cdot \mu}{\Delta x^2} \cdot (f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1})$$

$$-f_i^n \approx \frac{\Delta t \cdot \mu}{\Delta x^2} \cdot \left(f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}\right) - f_i^{n+1}$$

$$-f_i^n \approx \frac{\Delta t \cdot \mu}{\Delta x^2} \cdot \left(f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}\right) - f_i^{n+1}$$

$$-f_i^n \approx \frac{\Delta t \cdot \mu}{\Delta x^2} \cdot \left(f_{i+1}^{n+1} - 2f_i^{n+1} + f_{i-1}^{n+1}\right) - f_i^{n+1} \frac{\Delta x^2}{\Delta t \cdot \mu} \cdot \frac{\Delta t \cdot \mu}{\Delta x^2}$$

$$-f_i^n \approx \frac{\Delta t \cdot \mu}{\Delta x^2} \cdot \left(f_{i+1}^{n+1} - 2f_i^{n+1} - f_i^{n+1} \frac{\Delta x^2}{\Delta t \cdot \mu} + f_{i-1}^{n+1}\right)$$

$$-f_i^n \approx \frac{\Delta t \cdot \mu}{\Delta x^2} \cdot \left(f_{i+1}^{n+1} - (2 + \frac{\Delta x^2}{\Delta t \cdot \mu}) f_i^{n+1} + f_{i-1}^{n+1}\right)$$

$$f_i^n \approx -\frac{\Delta t \cdot \mu}{\Delta x^2} \cdot \left(f_{i+1}^{n+1} - (2 + \frac{\Delta x^2}{\Delta t \cdot \mu}) f_i^{n+1} + f_{i-1}^{n+1}\right)$$

Simplify by using

$$C_1 = -\frac{\Delta t \cdot \mu}{\Delta x^2}$$

$$C_2 = -(2 + \frac{\Delta x^2}{\Delta t \cdot \mu})$$

Then for $i = 1, \dots, Xn$:

$$f_i^n \approx C_1 \cdot \left(f_{i+1}^{n+1} + C_2 \cdot f_i^{n+1} + f_{i-1}^{n+1}\right)$$

$$f_i^n \approx C_1 \cdot f_{i+1}^{n+1} + C_1 \cdot C_2 \cdot f_i^{n+1} + C_1 \cdot f_{i-1}^{n+1}$$

And we can easily see that each f_i^n is a linear combination of three values of the next time step $f_{i+1}^{n+1}, f_i^{n+1}, f_{i-1}^{n+1}$.

And we know $f_i^{n+1} = f_i^n \ for\ the\ x\ values\ at\ the\ two\ ends\ points\ 0, 1$. So at the two end points, f_i^n is just a linear combination of one value of the next time step iteration (f_i^{n+1}).

This relationship allows us to relate an entire row of f_i^n and f_i^{n+1} where $i = 0,1, \dots, Nx$ as a tri-diagonal matrix system as discussed in Task 3.

$$\begin{bmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ c_2 & a_2 & b_2 & 0 & \cdots & 0 \\ 0 & c_3 & a_3 & b_3 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c_{N-1} & a_{N-1} & b_{N-1} \\ 0 & \cdots & 0 & 0 & c_N & a_N \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_N \end{Bmatrix} = \begin{Bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ \vdots \\ \vdots \\ Q_N \end{Bmatrix}$$

For convenience of working with C-programming, the index of array of matrix is adjusted so that it starts from 0

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ C_1 & C_1 \cdot C_2 & C_1 & 0 & \cdots & 0 \\ 0 & C_1 & C_1 \cdot C_2 & C_1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & C_1 & C_1 \cdot C_2 & C_1 \\ 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} f_0^{n+1} \\ f_1^{n+1} \\ f_2^{n+1} \\ \vdots \\ f_{Nx-1}^{n+1} \\ f_{Nx}^{n+1} \end{Bmatrix} = \begin{Bmatrix} f_0^n \\ f_1^n \\ f_2^n \\ \vdots \\ f_{Nx-1}^n \\ f_{Nx}^n \end{Bmatrix}$$

So

$$\tilde{a} = [a_0 \ \ a_1 \ \ a_2 \ \ \dots \ \ a_{Nx-1} \ \ a_{Nx}] = [1 \ \ C_1 \cdot C_2 \ \ C_1 \cdot C_2 \ \ \dots \ \ C_1 \cdot C_2 \ \ 1]$$

$$\tilde{b} = [b_0 \ \ b_1 \ \ b_2 \ \ \dots \ \ b_{Nx-1} \ \ b_{Nx}] = [0 \ \ C_1 \ \ C_1 \ \ \dots \ \ C_1 \ \ 0]$$

$$\tilde{c} = [c_0 \ \ c_1 \ \ c_2 \ \ \dots \ \ c_{Nx-1} \ \ c_{Nx}] = [0 \ \ C_1 \ \ C_1 \ \ \dots \ \ C_1 \ \ 0]$$

$$\tilde{Q} = [Q_0 \ \ Q_1 \ \ Q_2 \ \ \dots \ \ Q_{Nx-1} \ \ Q_{Nx}] = [f_0^n \ \ f_1^n \ \ f_2^n \ \ \dots \ f_{Nx-1}^n \ \ f_{Nx}^n]$$

$$\tilde{X} = [x_0 \ \ x_1 \ \ x_2 \ \ \dots \ \ x_{Nx-1} \ \ x_{Nx}] = [f_0^n \ \ f_1^n \ \ f_2^n \ \ \dots \ f_{Nx-1}^n \ \ f_{Nx}^n]$$

6.2: Tasks and Analysis

Run the code until a time of $t = 2.0$ for different resolutions Δx. The time steps should satisfy the Diffusion number $\frac{\mu \Delta t}{(\Delta x)^2} \leq 1$.

I will investigate the effect of

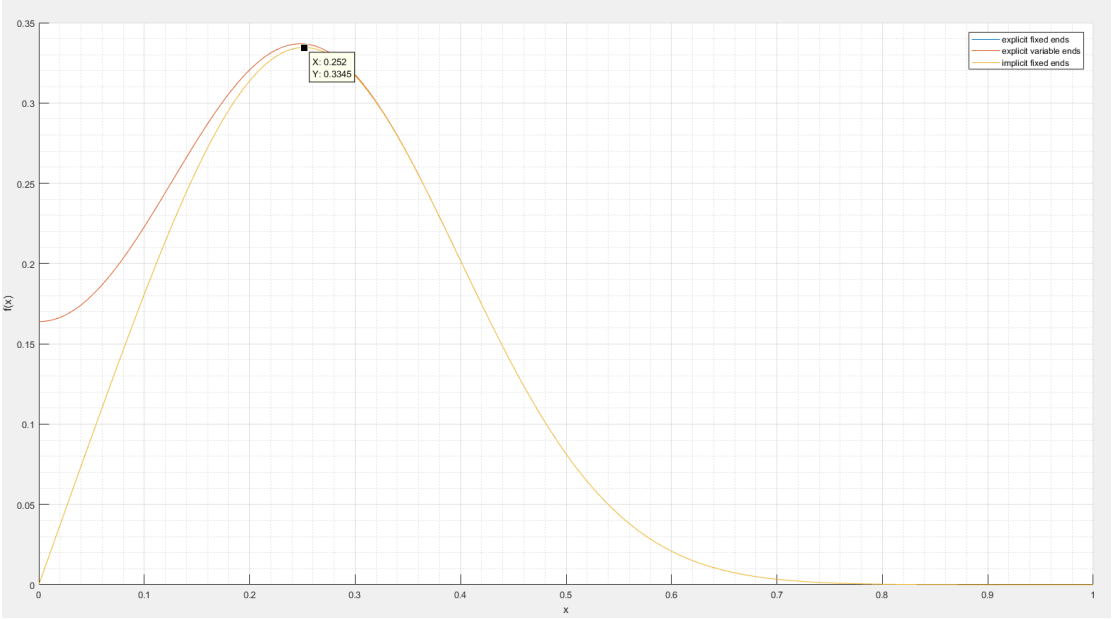
- Spatial resolution, i.e. Δx, determined by the number of points Nx used
- Temporal resolution Δt (or, for a given Δx, variations of the Diffusion number $\frac{\mu \Delta t}{(\Delta x)^2}$
- Effect of time integration type on the maximum time step permitted

On the accuracy of the solution.

$$\frac{\mu \Delta t}{(\Delta x)^2} \leq 1$$
$$\Delta t \leq \frac{(\Delta x)^2}{\mu}$$
$$\frac{2}{Nt} \leq \frac{1}{\mu Nx^2}$$
$$Nt \geq \frac{2Nx^2}{\mu}$$

The grid-independent reference solution at $Nx = 1000\ and\ diffusion\ number$ 0.1, and this gives $Nt = 100000$. The solution can be seen for the three methods used.

$$Nx = 1000, \qquad Nt = 100000$$



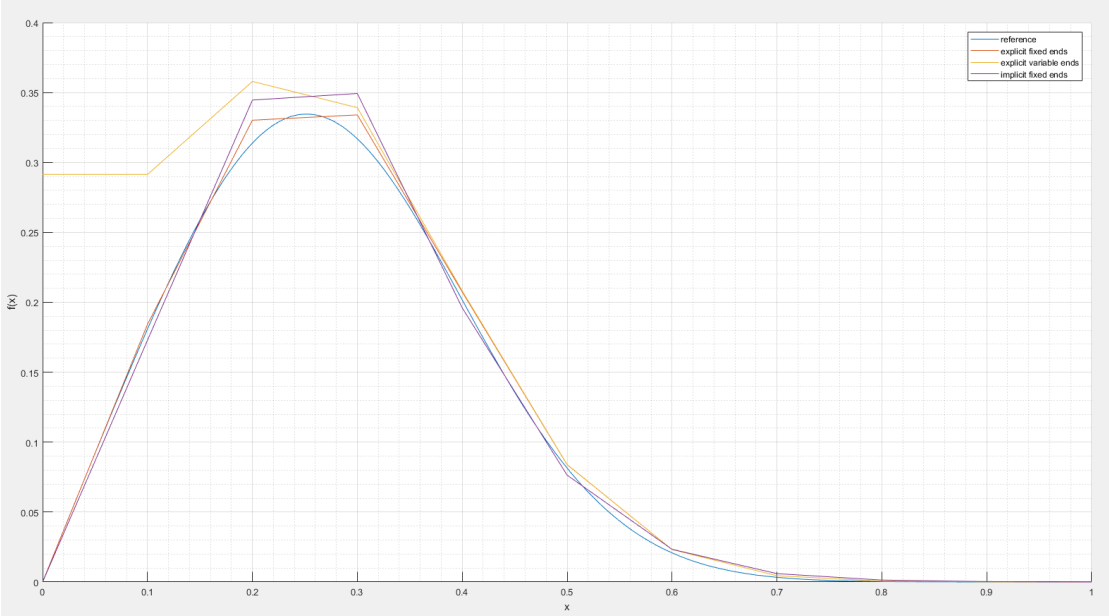
Note that explicit fixed ends and implicit fixed ends agree with each other on the plot, the explicit variable ends does not agree with them on the left half, and is due to the boundary conditions used, and is not accurate for this context. The peak is at $x = 0.252, f(x) = 0.3345$

For investigating the effect of spatial resolution. I tried to fix $Nt = 100$ and decrease Nx from 100 gradually down to 10 and to see its effect on the results for the three different algorithm. However, this will inevitably change the diffusion number, and different algorithms perform very differently under different diffusion numbers.

So instead, I will fix the diffusion number at 0.1.

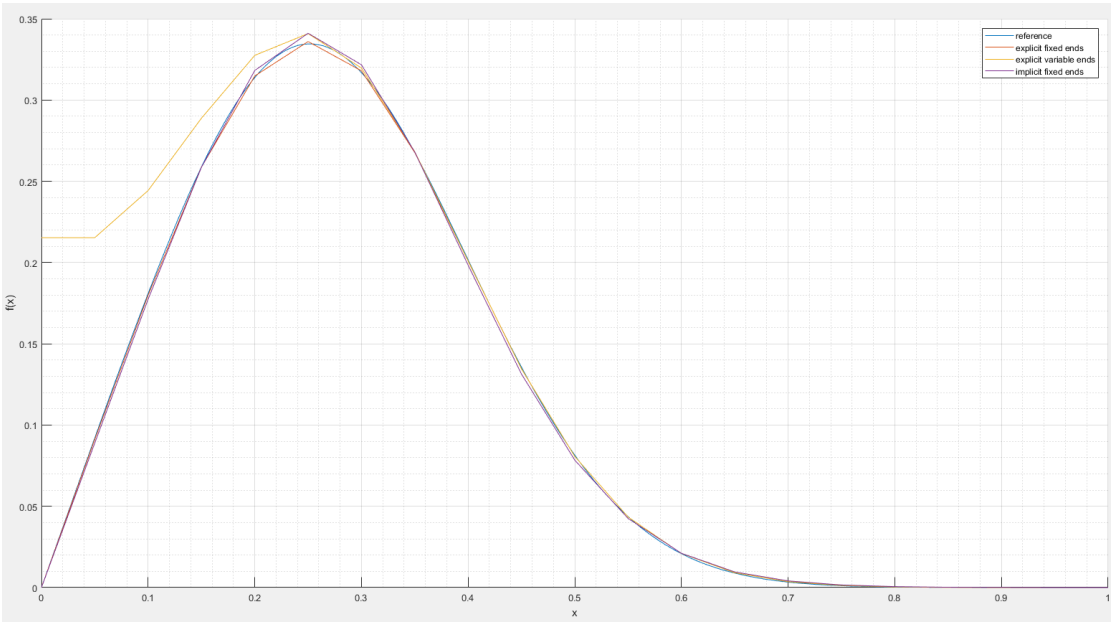
$$\frac{0.005\Delta t}{(\Delta x)^2} = 0.1$$
$$\frac{0.005\Delta t}{(\Delta x)^2} = 0.1$$
$$\Delta t = 20(\Delta x)^2$$
$$\frac{2}{Nt} = \frac{20}{(Nx)^2}$$
$$So \quad Nt = \frac{(Nx)^2}{10}$$

$$Nx = 10, \qquad Nt = 10, \qquad diffusion\, number = 0.1$$



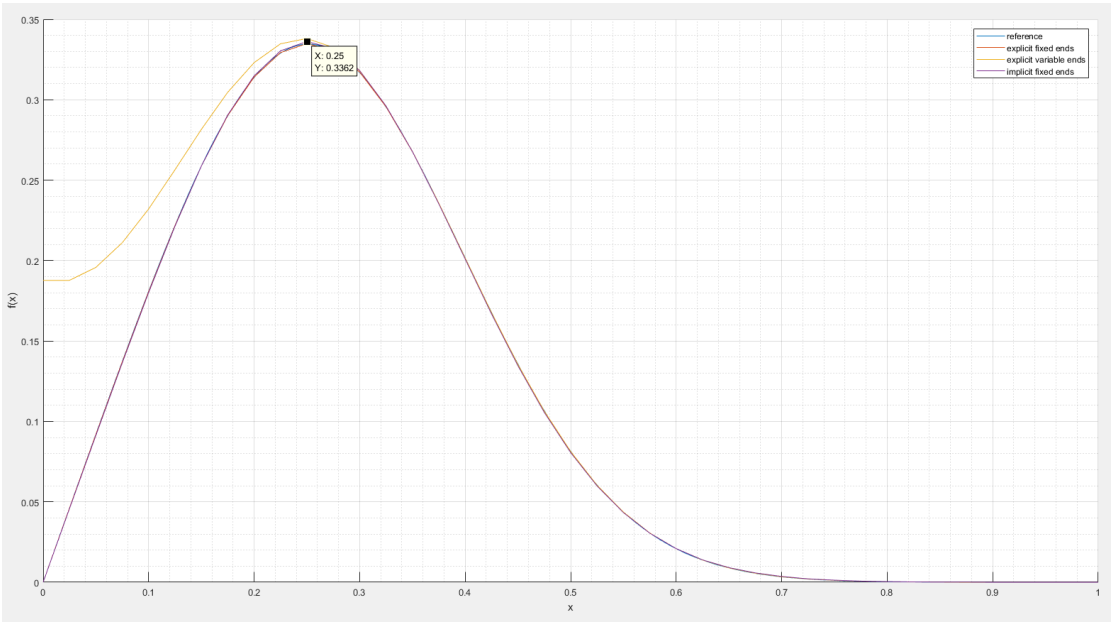
Explicit fixed end produces a result closest to the reference at this setting

$$Nx = 20, \quad Nt = 40, \quad diffusion\ number = 0.1$$



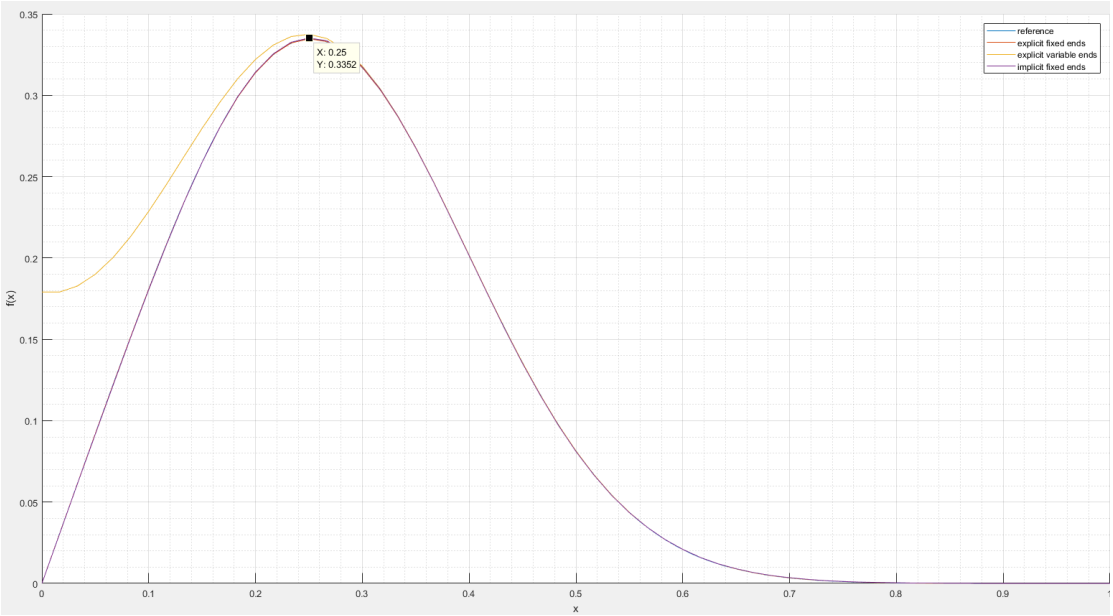
Explicit fixed end produces a result closest to the reference at this setting

$$Nx = 40, \quad Nt = 160, \quad diffusion\ number = 0.1$$



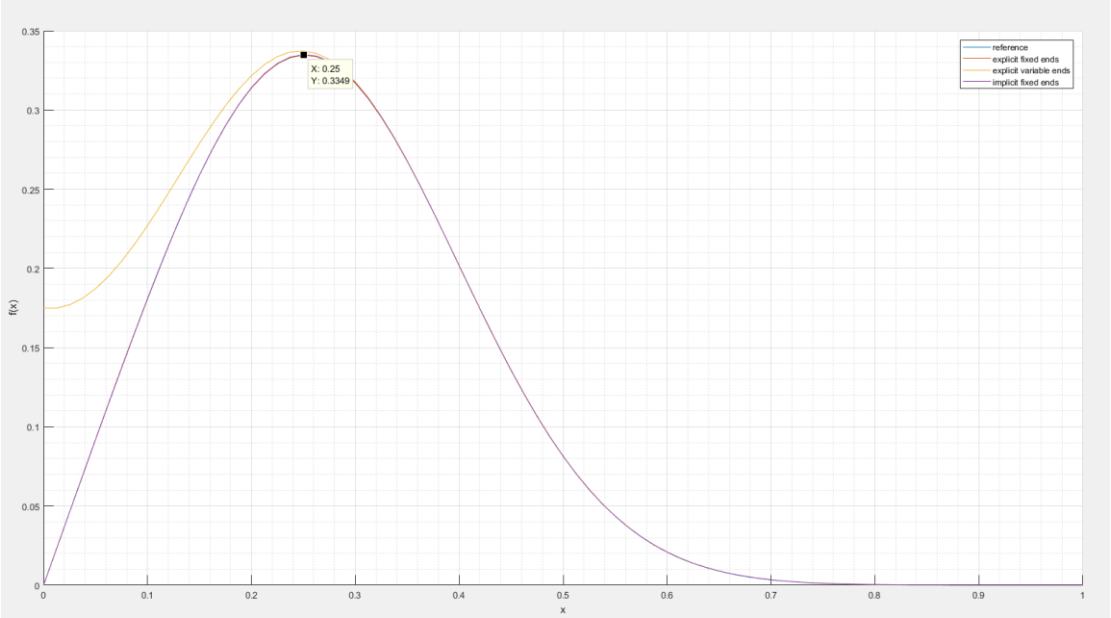
Both explicit fixed ends and implicit fixed ends converge to the reference well, with error within 0.002.

$$Nx = 60, \quad Nt = 360, \quad diffusion\ number = 0.1$$



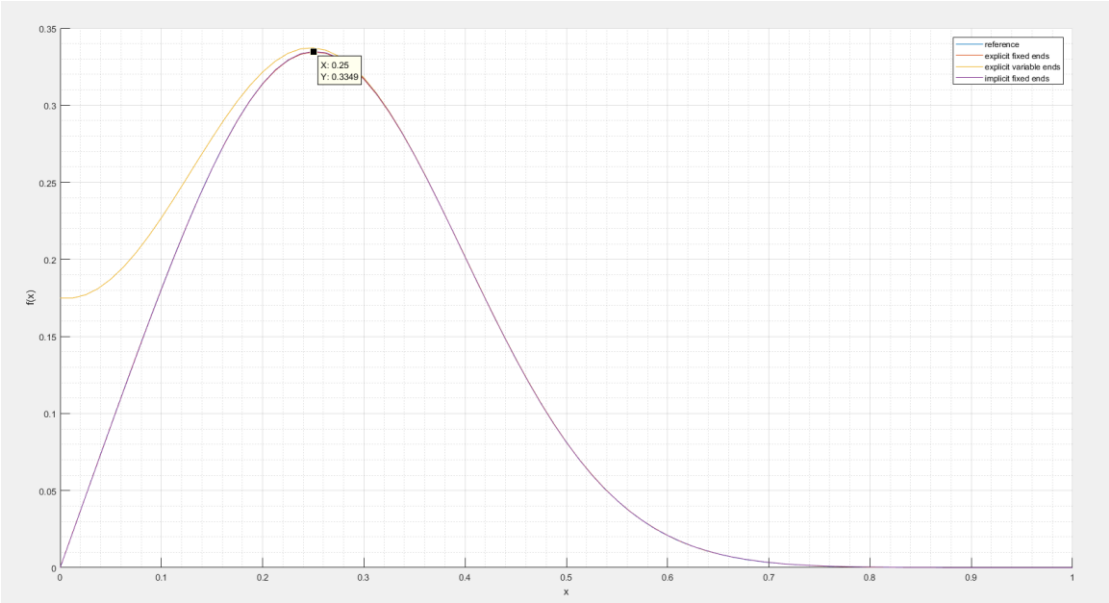
Both explicit fixed ends and implicit fixed ends converge to the reference well, with error within 0.001

$$Nx = 60, \quad Nt = 360, \quad diffusion\ number = 0.1$$



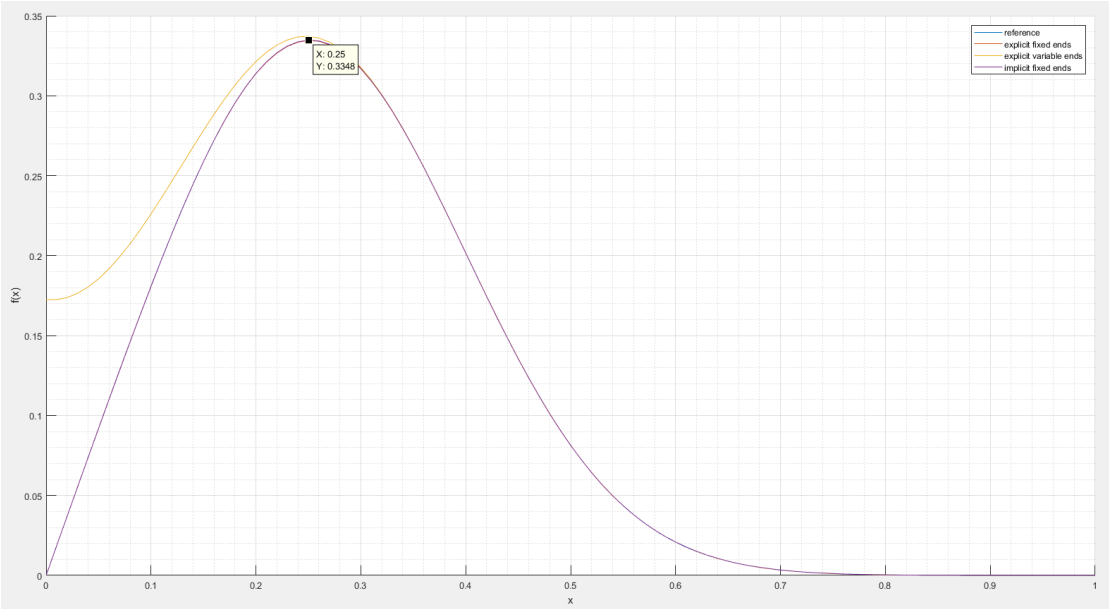
Both explicit fixed ends and implicit fixed ends converge to the reference well, with error within 0.0005

$$Nx = 80 \;\; Nt = 64 \quad\quad diffusion\, number = 0.1$$



Both explicit fixed ends and implicit fixed ends converge to the reference well, with error within 0.0005

$$Nx = 100 \; Nt = 1000 \quad\quad diffusion\, number = 0.1$$



Both explicit fixed ends and implicit fixed ends converge to the reference well, with error within 0.0003.

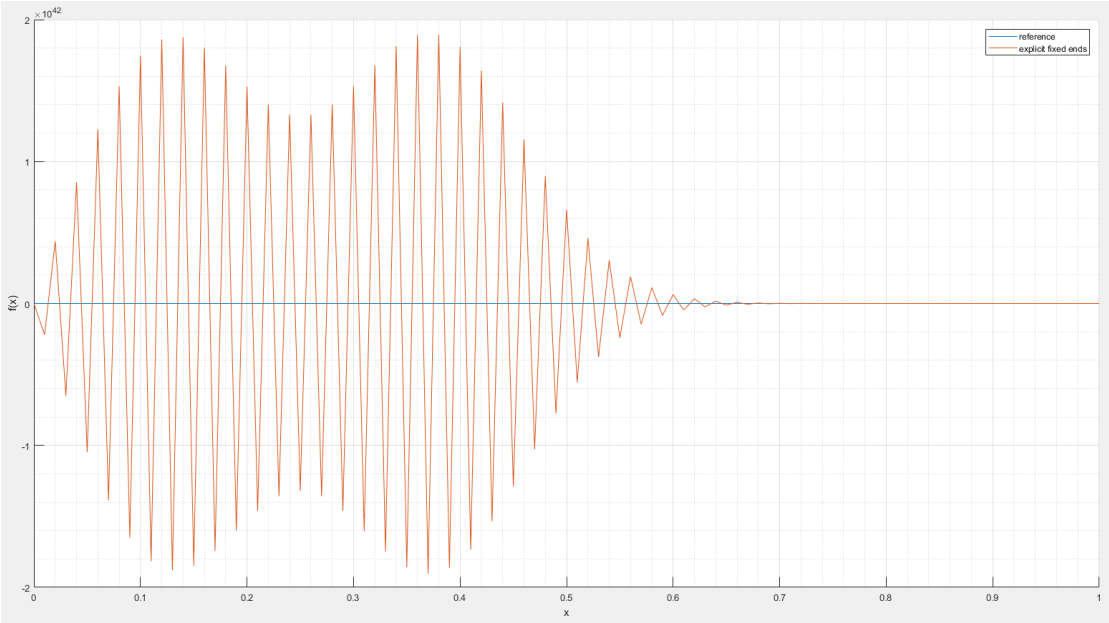
In all above cases, the results produced by explicit variable ends are unsatisfactory due to the boundary conditions setting, only the right side of the peak converges to the reference solution

So, overall, increase the spatial resolution will also increase the accuracy of the solution, given that the choice of the temporal resolution is suitable such that all three algorithms can converge using the temporal resolution settings. For explicit methods, the convergence is tricky beyond a limit for the diffusion, I will show this in details later.

For investigating the effect of temporal resolution, I will keep $Nx = 100$ and increase Nt from 100 and onwards and see its effect on the results for the three different algorithm. This is equivalent of investigating the effect of decreasing the diffusion number for a given Δx.

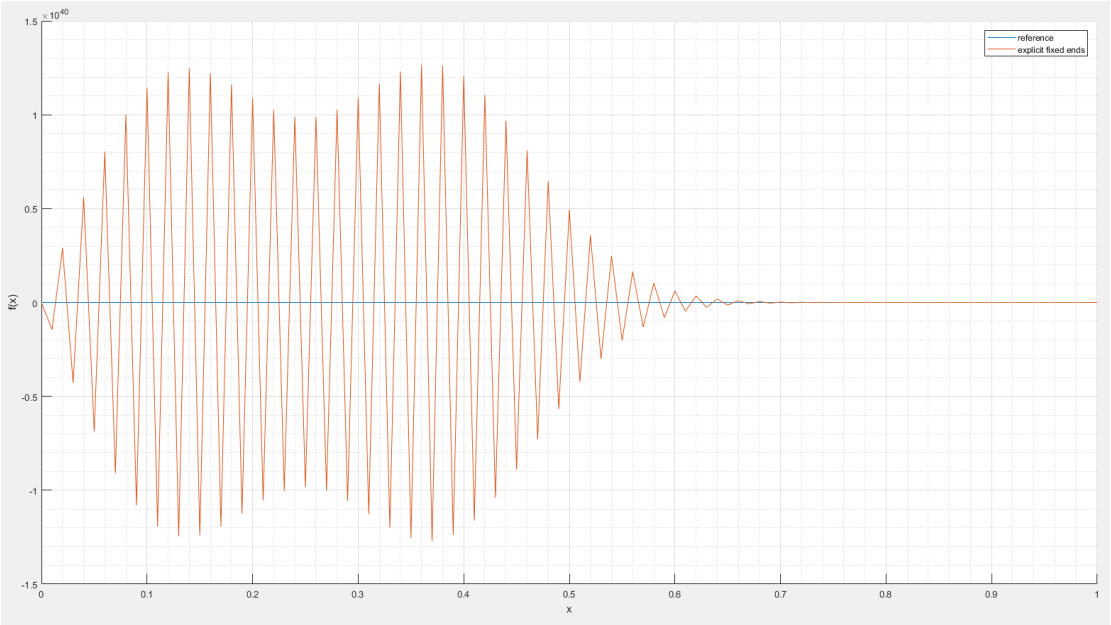
For explicit fixed ends:

$$Nx = 100\ Nt = 100 \qquad diffusion\ number = 1$$



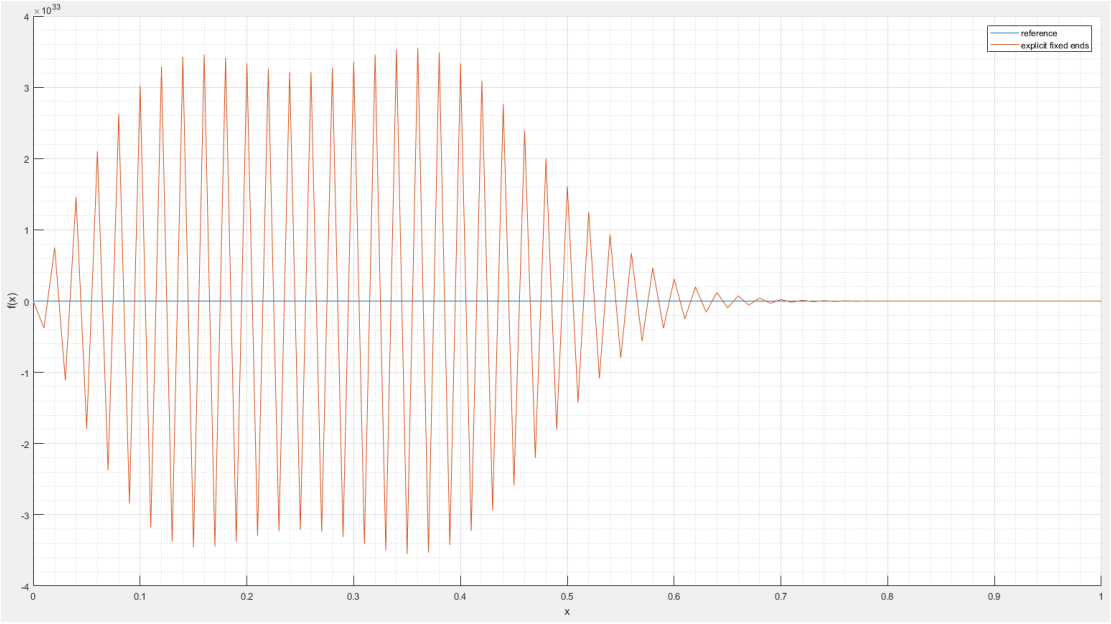
The solution diverges.

$$Nx = 100\ Nt = 120 \quad diffusion\ number = 0.833$$



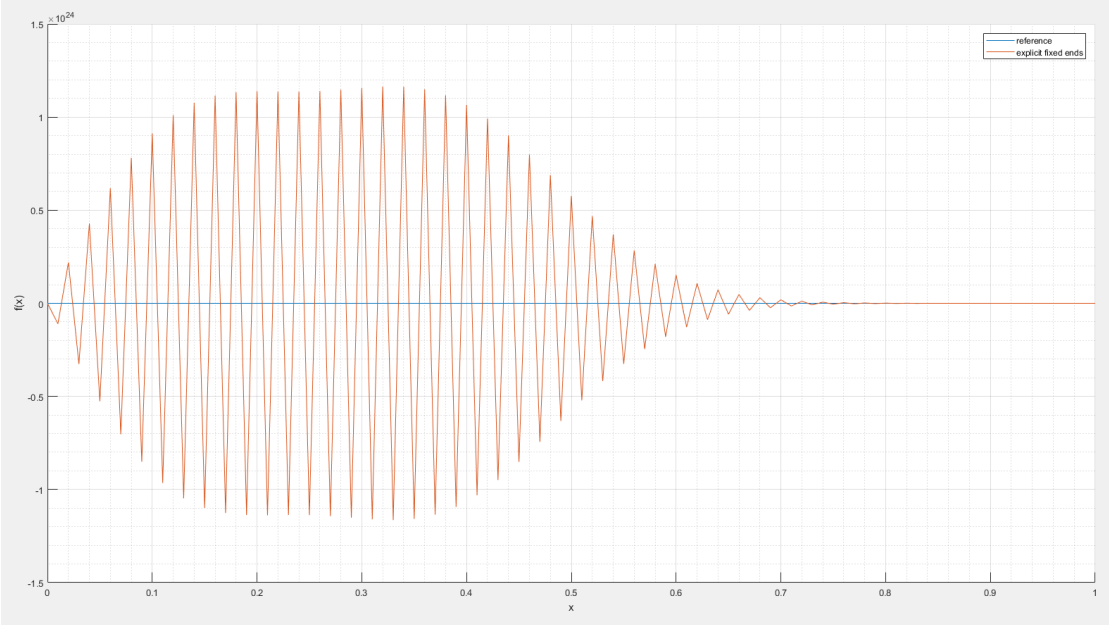
The solution diverges.

$$Nx = 100\ Nt = 140 \quad diffusion\ number = 0.714$$



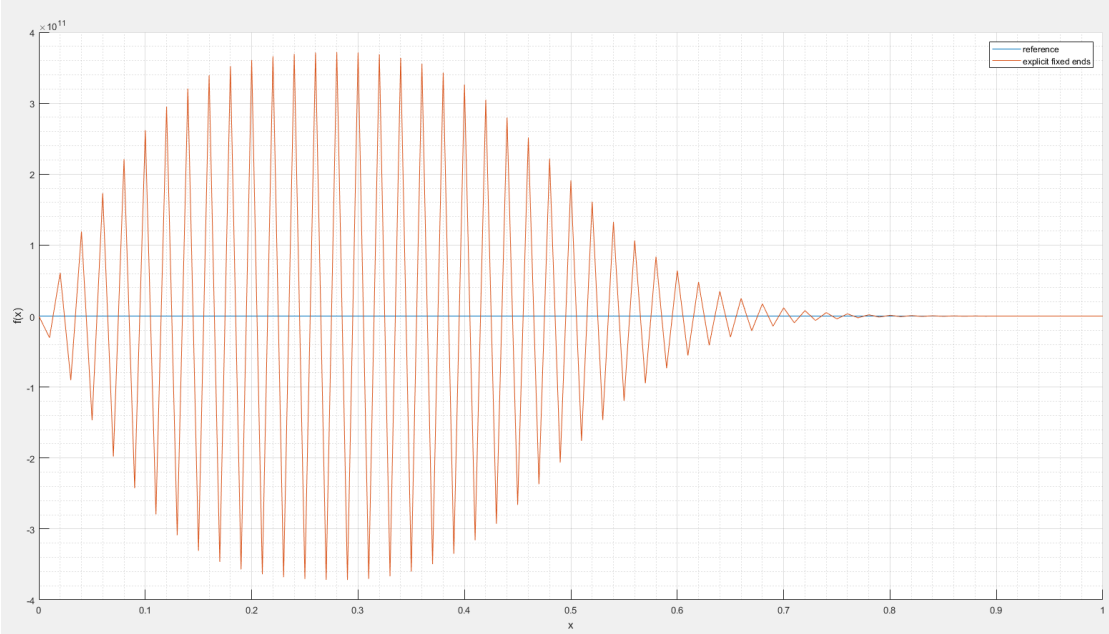
The solution diverges

$$Nx = 100\ Nt = 160 \quad diffusion\ number = 0.625$$



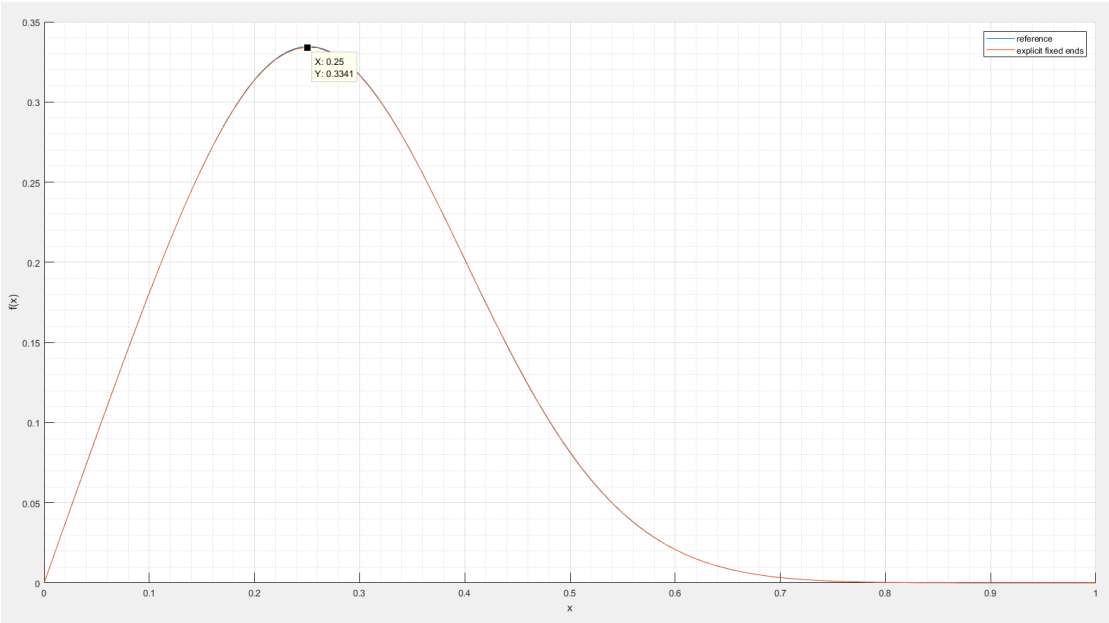
The solution diverges

$$Nx = 100\ Nt = 180 \quad diffusion\ number = 0.556$$



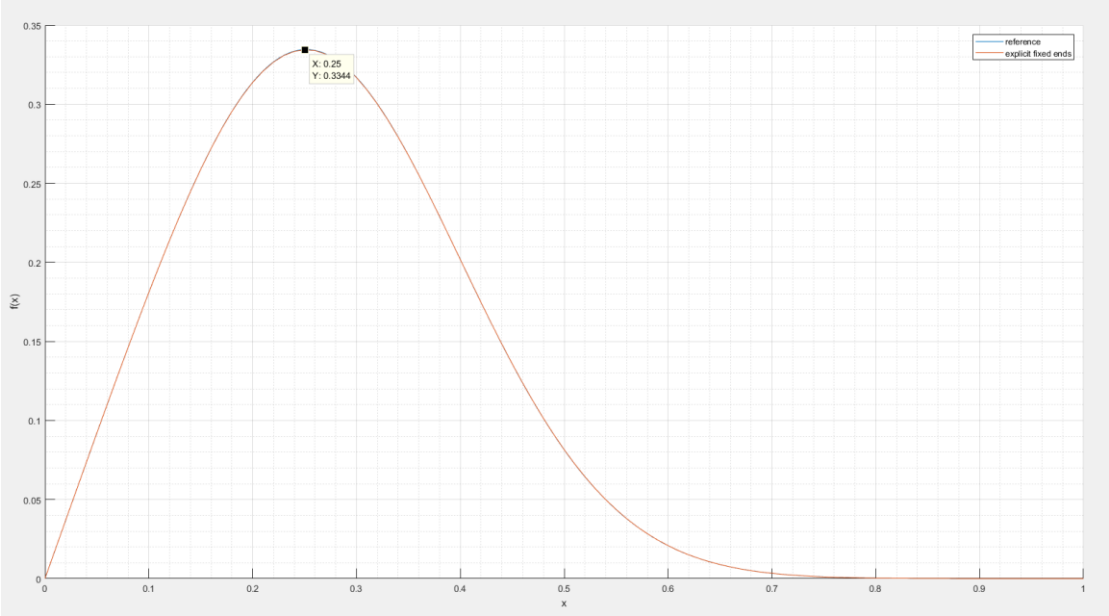
The solution diverges

$$Nx = 100\ Nt = 200 \quad diffusion\ number = 0.5$$



The solution converges, with error within 0.0005

$$Nx = 100\ Nt = 400 \quad diffusion\ number = 0.25$$

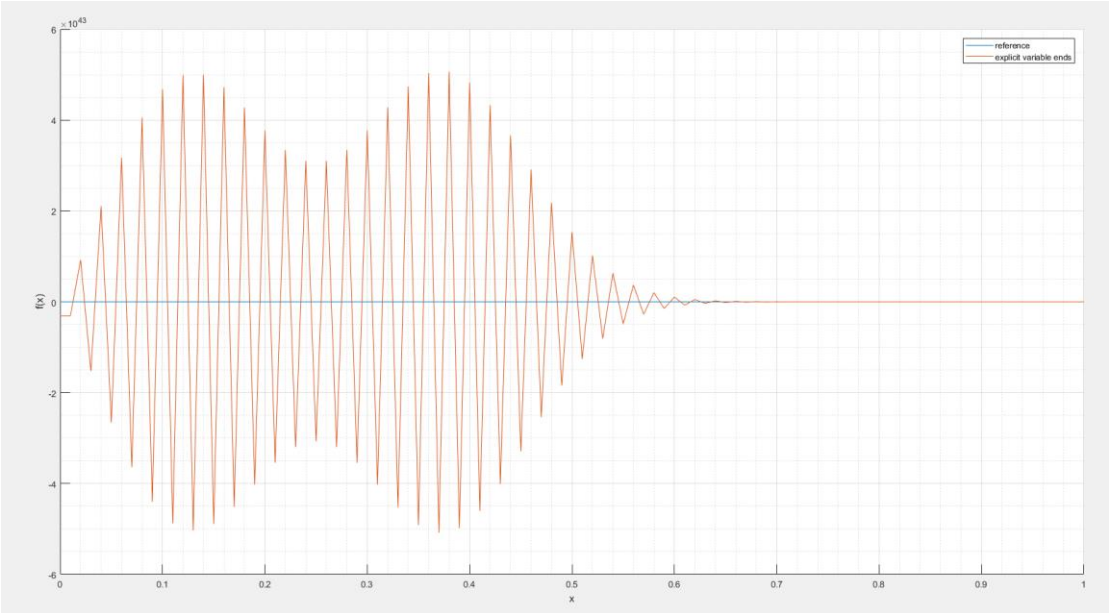


The solution converges, with error within 0.0001.

I will stop here, as 4 decimal precision's limit will not tell the further differences.
So for explicit fixed end methods, the solution only converges when the diffusion number is less than or equal to 0.5. Which means for $\mu = 0.005$, the value of Nt has to be at least 2 times of the value of Nx, or in other words, the temporal step must be smaller or equal to half of the spatial resolution for this method to converge. But once it converges, it is very accurate and decreasing the temporal resolution will make it more accurate, the convergence of error is also fast.

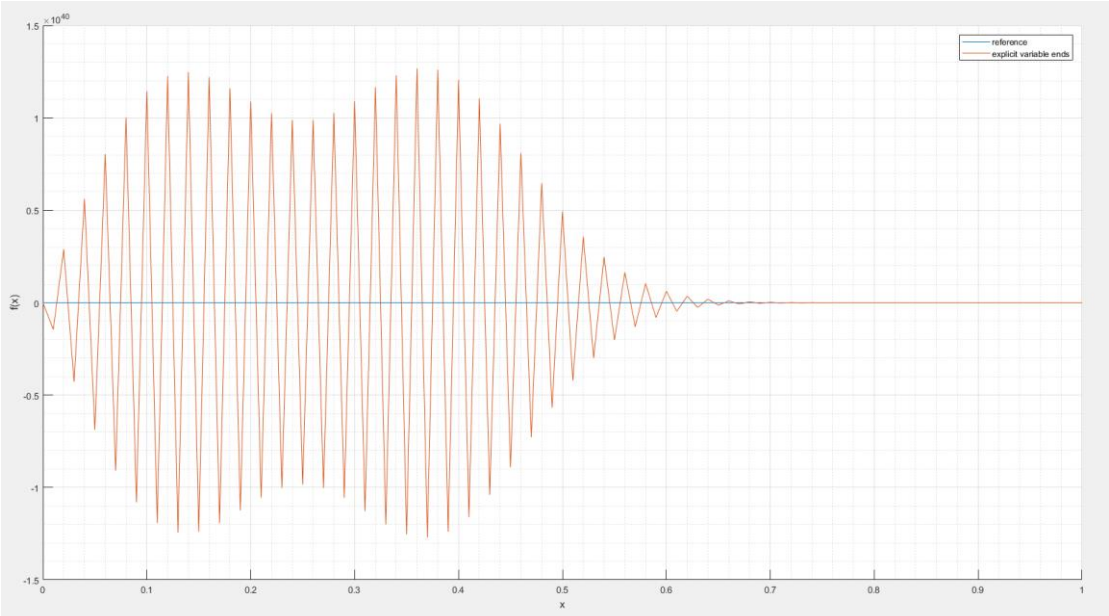
For explicit variable ends:

$$Nx = 100\ Nt = 100 \quad diffusion\ number = 1$$



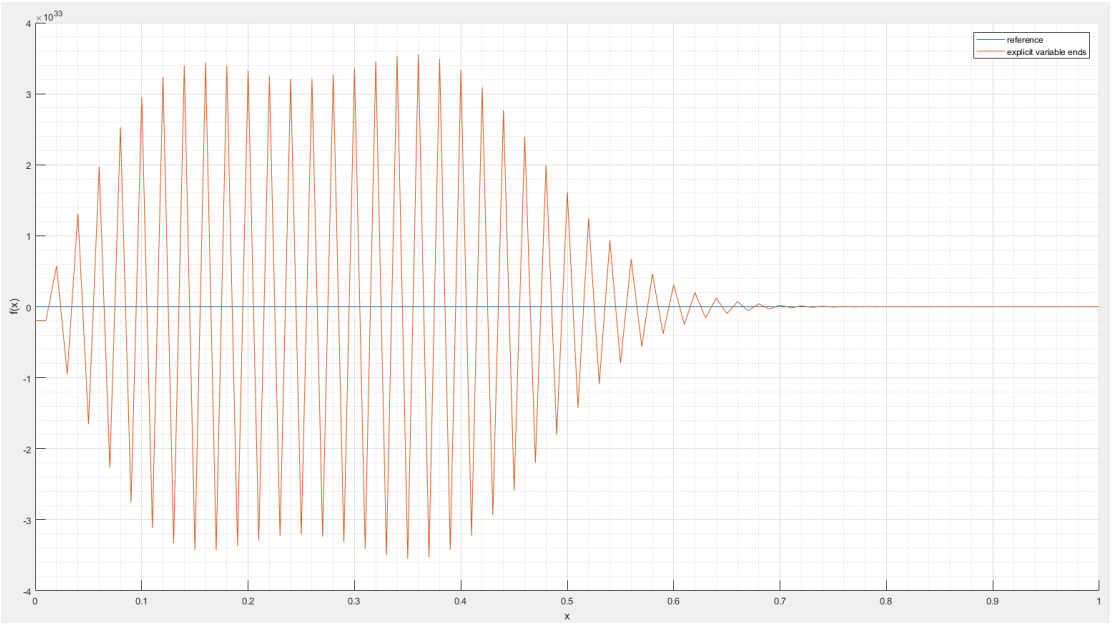
The solution diverges

$$Nx = 100\ Nt = 120 \quad diffusion\ number = 0.833$$



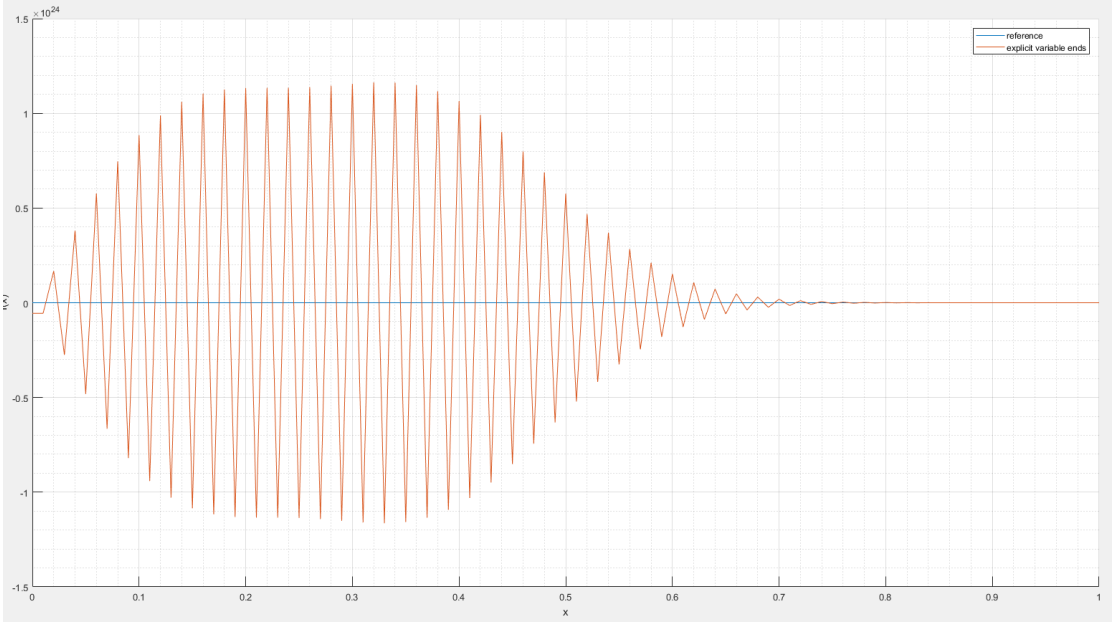
The solution diverges

$$Nx = 100\ Nt = 140 \quad diffusion\ number = 0.714$$



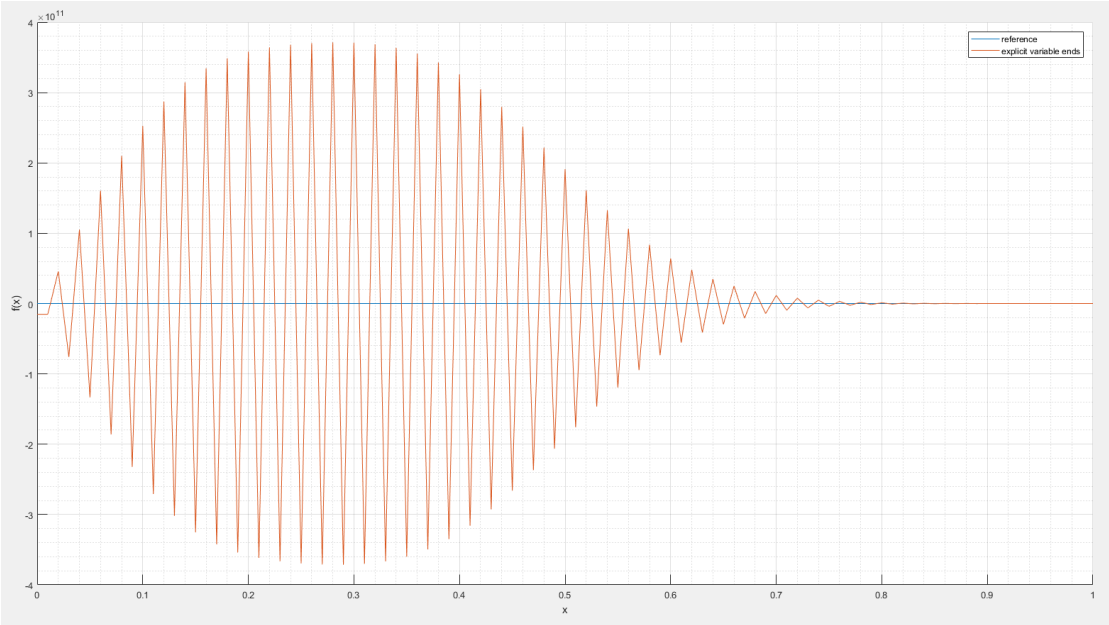
The solution diverges

$$Nx = 100\ Nt = 160 \quad diffusion\ number = 0.625$$



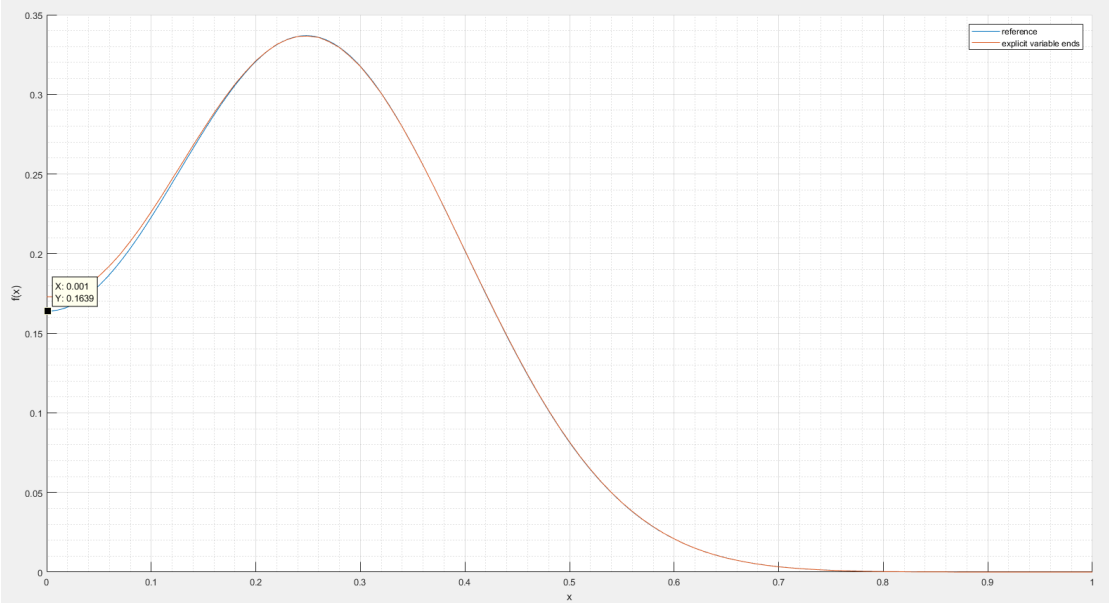
The solution diverges

$$Nx = 100\ Nt = 180 \quad diffusion\ number = 0.556$$



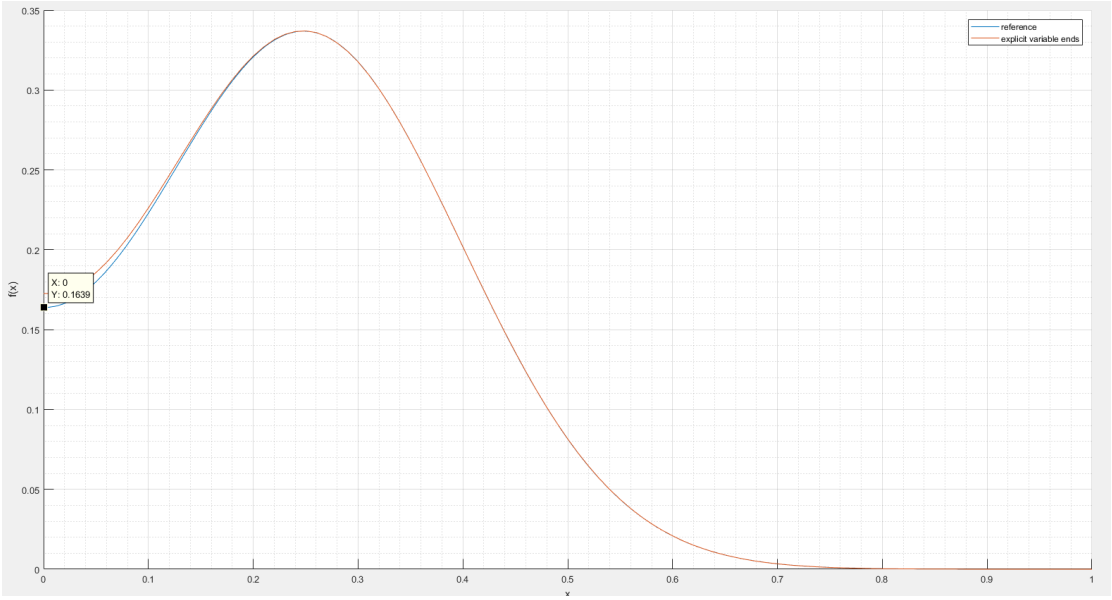
The solution diverges.

$$Nx = 100\ Nt = 200 \quad diffusion\ number = 0.5$$



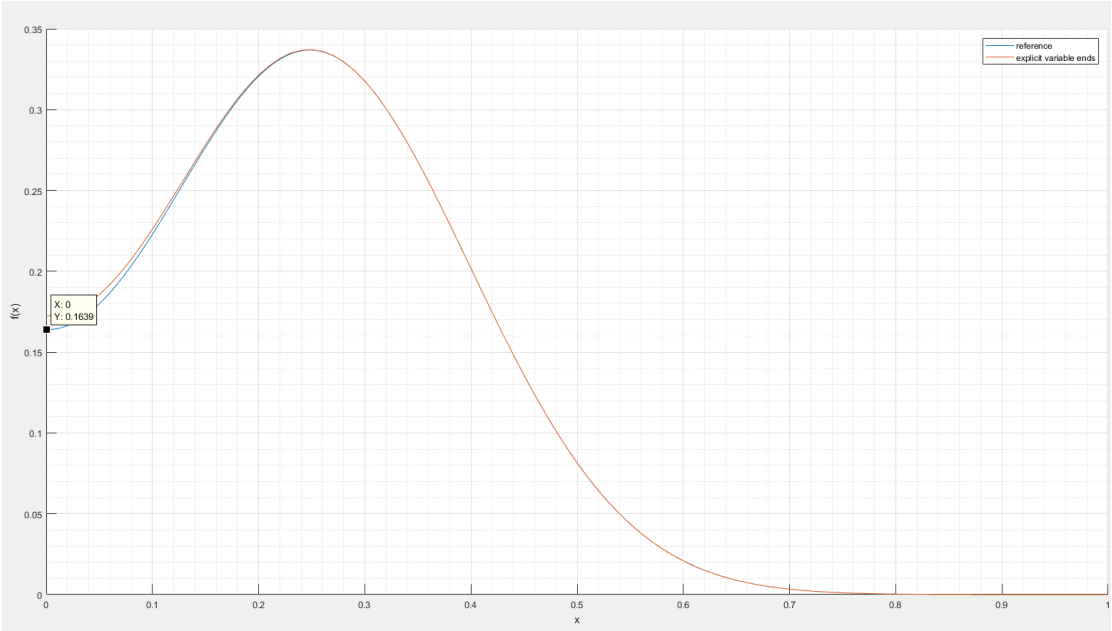
Solution converges, has an error within 0.01.

$$Nx = 100\ Nt = 400 \quad diffusion\ number = 0.25$$



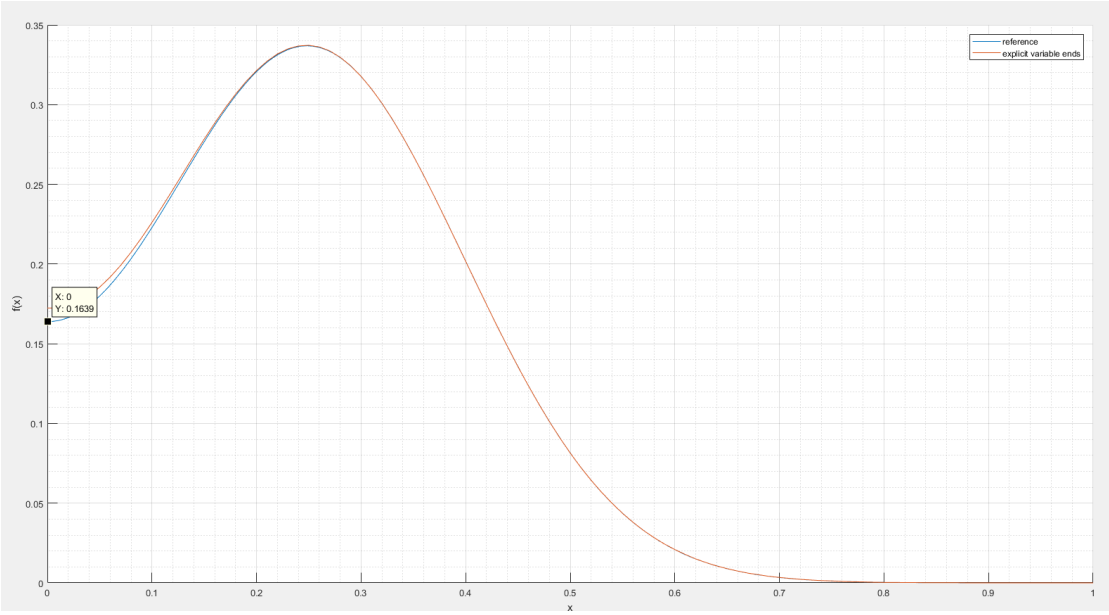
Solution converges, has an error within 0.01.

$$Nx = 100\ Nt = 800 \quad diffusion\ number = 0.125$$



Solution converges, has an error within 0.01.

$$Nx = 100\ Nt = 2000 \quad diffusion\ number = 0.005$$

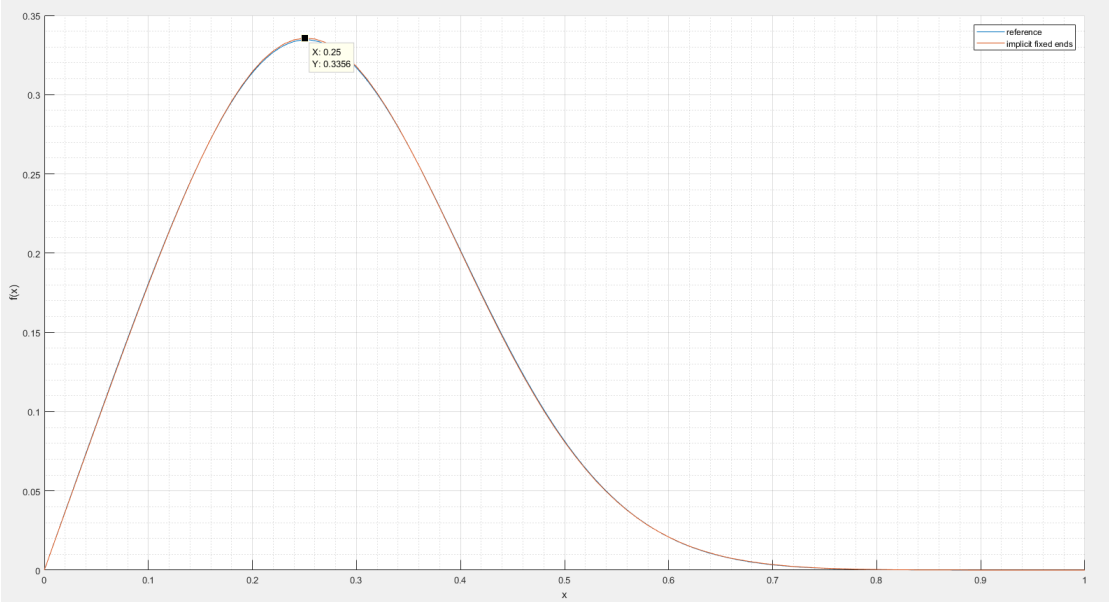


Solution converges, has an error within 0.01.

So for explicit variable end methods, the solution only converges when the diffusion number is less than or equal to 0.5. Which means for $\mu = 0.005$, the value of Nt has to be at least 2 times of the value of Nx, or in other words, the temporal step must be smaller or equal to half of the spatial resolution for this method to converge.
But the converged solution does not agree with the reference.
The left side error does not diminish as we increase the value of Nt. In fact, the solution of explicit variable method does not converge to the same value for the left side using different x- values.
Even the reference solution does not agree with the solution produced by explicit fixed end and implicit fixed end methods.

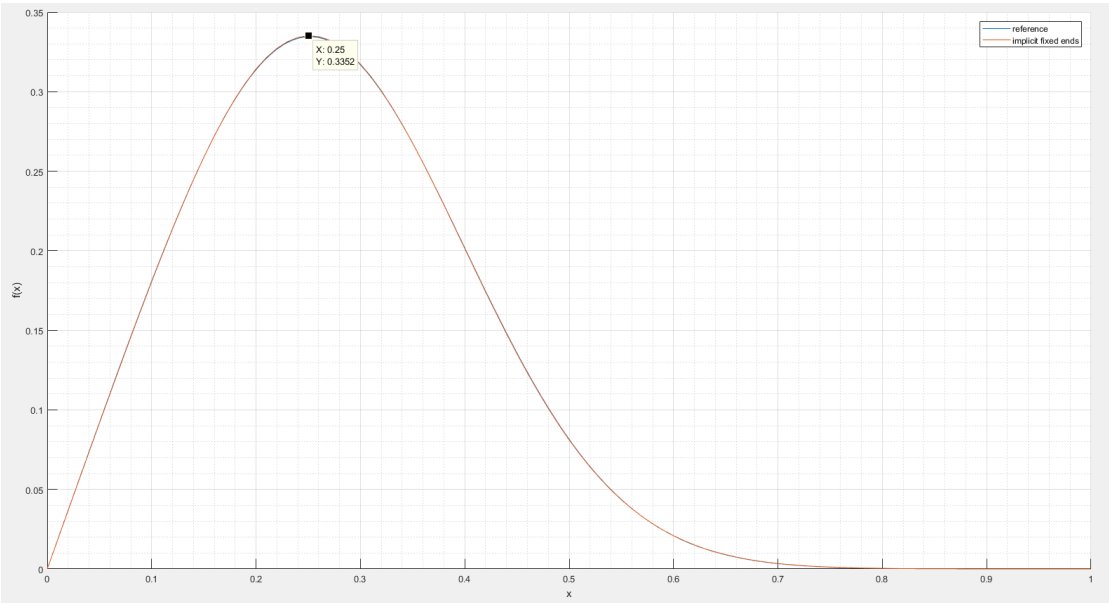
For implicit fixed ends:

$$Nx = 100\ Nt = 100 \quad diffusion\ number = 1$$



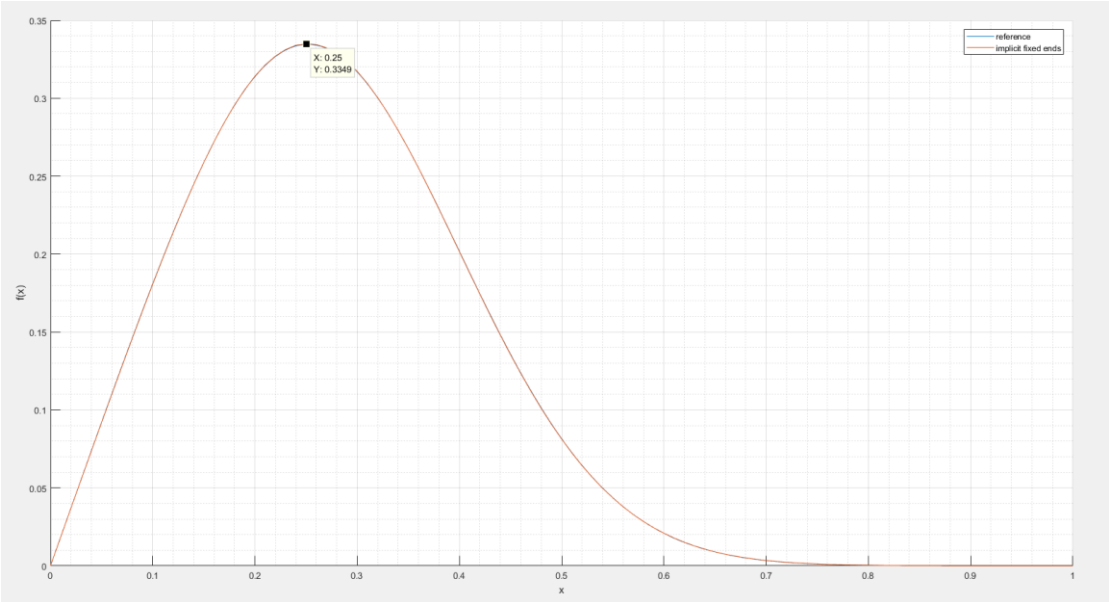
Converges, the error is within 0.0012

$$Nx = 100\ Nt = 200 \quad diffusion\ number = 0.5$$



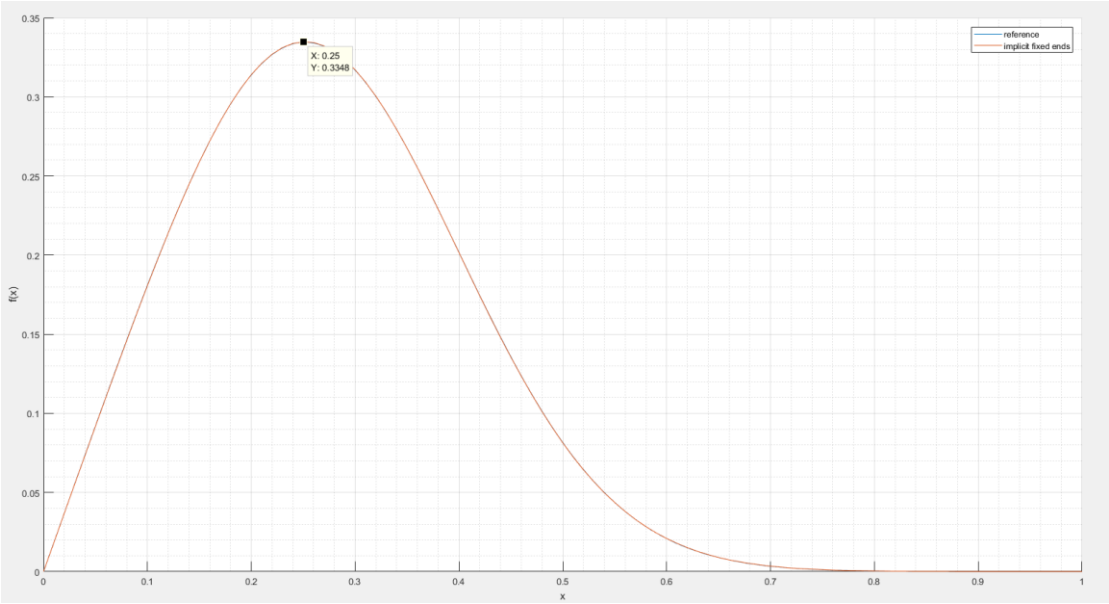
Converges, the error is within 0.0008

$$Nx = 100\ Nt = 400 \quad diffusion\ number = 0.25$$



Converges, the error is within 0.0005

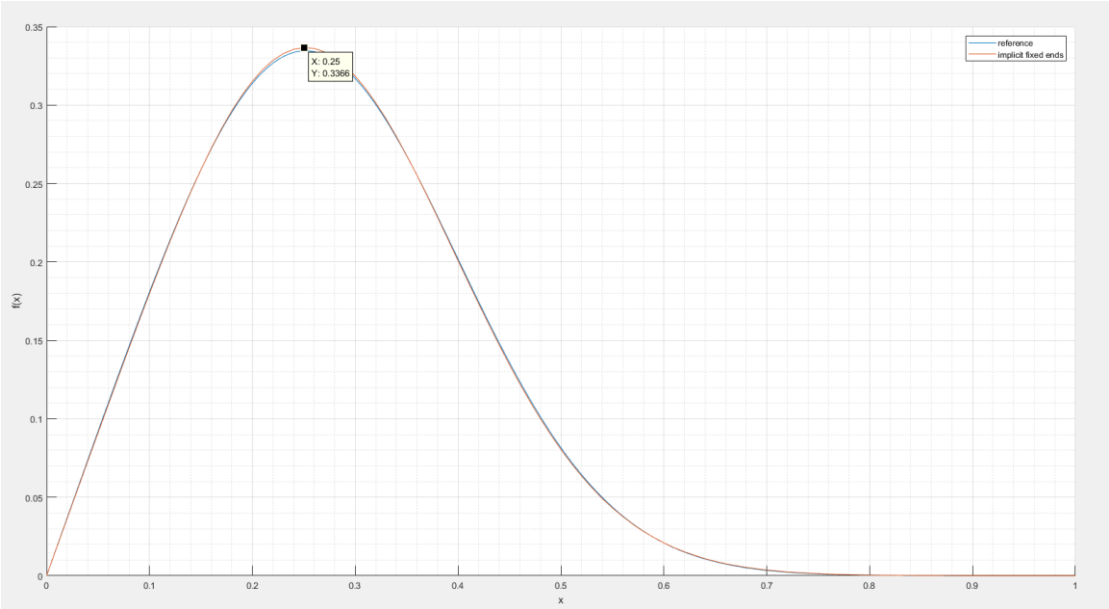
$$Nx = 100\ Nt = 800 \quad diffusion\ number = 0.125$$



Converges, the error is within 0.0004

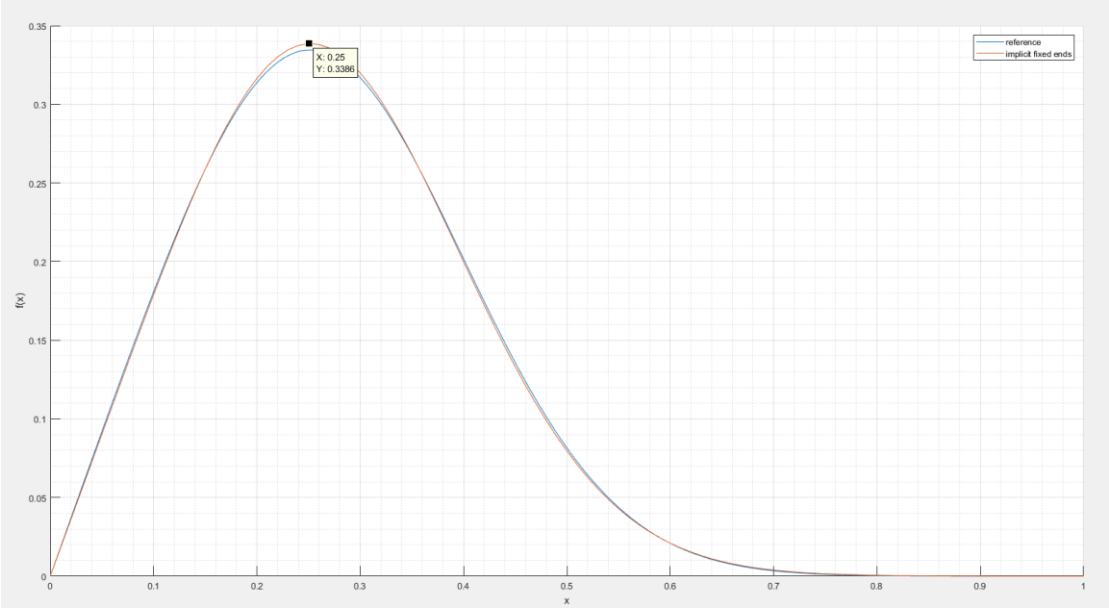
Try to use diffusion number more than 1.

$$Nx = 100\ Nt = 50 \quad diffusion\ number = 2$$



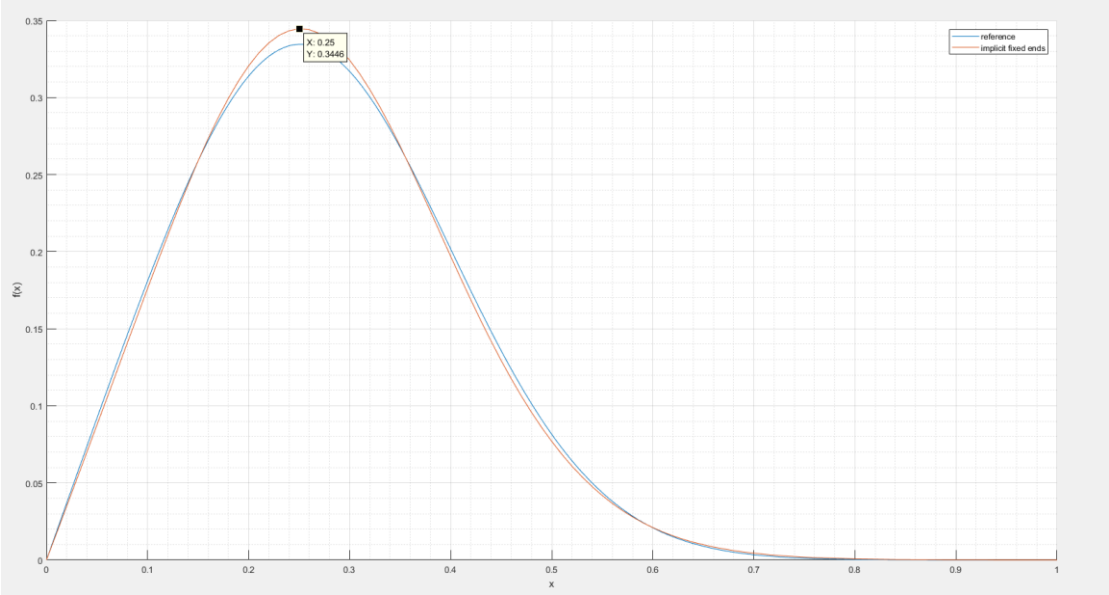
Converges, the error is within 0.0025

$$Nx = 100\ Nt = 25 \quad diffusion\ number = 4$$



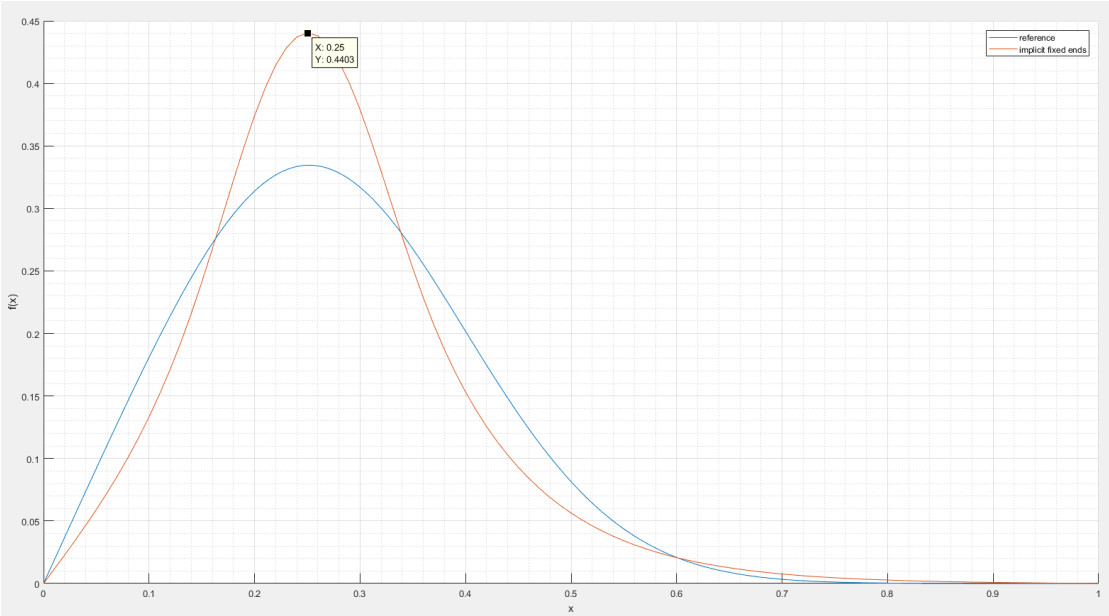
Converges, the error is within 0.005

$$Nx = 100\ Nt = 10 \qquad diffusion\ number = 10$$



Converges, the error is within 0.01

$$Nx = 100\ Nt = 1 \qquad diffusion\ number = 100$$



Converges, the error is within 0.1

So for implicit fixed end method, there is no limit of the temporal step size for it to converge. It converges for any Nt values. However, compared to explicit fixed-end method, the error converges much slower.

Overall, if the Nt values allow any of the method to converge, then increasing the value of Nt will definitely increase the accuracy of the solution. Where the error in explicit method converges faster with respect to increase in Nt, the implicit fixed end method can work for any Nt value, and is good for quick computation where the accuracy requirement is not that high.

Explicit variable end method is unsatisfying as its convergence results depends on the value of Nx, it produces a reference result different to the other two methods.