FYS4150 Computational Physics, project 1

James Claxton, Amund Fredriksen

September 7, 2020

Abstract

This report aims to solve the one-dimensional Poisson equation by discretising the function and calculating the second derivative at each discrete point. Discretising the continuous function leads to a set of linear equations, which in matrix form results in a diagonal matrix with elements along the diagonal and the off-diagonal. As a result, the solution can be optimised to reduce the number of floating point operations and the memory required. The solution from algorithms in this report have been compared to the analytic solution and the solution from libraries for the LU decomposition.

1 Introduction

The aim of this report is to solve the one-dimensional Poisson equation:

$$-u''(x) = f(x), \qquad x \in (0,1), \qquad u(0) = u(1) = 0.$$

where the discrete approximation to the analytic function $u(x)$ is named v_i. Euler's method will be used to numerically solve this differential equation. Hence, v_i has points $x_i = ih$ in the interval of $x_0 = 0$ to $x_{n+1} = 1$, and where h is the step length, $h = 1/(n+1)$. The second derivative of $u(x)$ takes the numerical form:

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \quad \text{for } i = 1, \ldots, n, \tag{1}$$

where $f_i = f(x_i)$.

This can be written as a linear set of equations:

$$\mathbf{A}\mathbf{v} = \mathbf{g}, \tag{2}$$

where $g_i = h^2 f_i$ and the matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \ldots & \ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots & \ldots \\ 0 & -1 & 2 & -1 & 0 & \ldots \\ & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & & -1 & 2 & -1 \\ 0 & \ldots & & 0 & -1 & 2 \end{bmatrix}.$$

This can be shown by performing the matrix multiplication in the left-hand side of equation (2) and comparing with equation (1):

$$\mathbf{Av} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$
$$= \begin{bmatrix} 2v_1 - v_2 \\ -v_1 + 2v_2 - v_3 \\ \vdots \\ -v_{n-2} + 2v_{n-1} - v_n \\ -v_{n-1} + 2v_n \end{bmatrix}$$
$$= - \begin{bmatrix} v_2 + v_0 - 2v_1 \\ v_3 + v_1 - 2v_2 \\ \vdots \\ v_{n+1} + v_{n-1} - 2v_n \end{bmatrix} \overset{(1)}{=} h^2 \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \equiv \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{bmatrix} \equiv \mathbf{g}$$

since $v_0 = v_{n+1} = 0$.

In this project $f(x)$ will be assumed to be $f(x) = 100e^{-10x}$, which has the solution $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$. This can be shown by inserting the solution $u(x)$ into the Poisson equation. The first derivative of $u(x)$ with respect to x is therefore:

$$\frac{du(x)}{dx} = -(1 - e^{-10}) + 10e^{-10x}$$

and the second derivative is:

$$\frac{d^2u(x)}{dx^2} = -100e^{-10x}$$

and hence $-u''(x) = 100e^{-10x} = f(x)$.

Gaussian elimination

A way to solve a linear system of equations is to write the system as an augmented matrix and then perform the algorithm named 'Gaussian elimination'.

The algorithm will be used in the case of a general tridiagonal matrix. A general tridiagonal matrix has the form

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & \dots & 0 & 0 \\ a_1 & b_2 & c_2 & 0 & \vdots & 0 \\ 0 & a_2 & b_3 & c_3 & 0 & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & & & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_n \end{bmatrix}. \tag{3}$$

A linear equation set as in eq. (2) can be written as an augmented matrix, informally written as

$$[\mathbf{A}|\mathbf{g}],$$

where the elements of $\mathbf{v}$ have been omitted since the matrix multiplication $\mathbf{Av}$ is implied. Now column number i of $\mathbf{A}$ corresponds to the variable v_i. Written out, the augmented matrix will have the following form:

$$\left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ a_1 & b_2 & c_2 & 0 & \vdots & 0 & g_2 \\ 0 & a_2 & b_3 & c_3 & 0 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & & & a_{n-2} & b_{n-1} & c_{n-1} & g_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_n & g_n \end{array}\right]. \tag{4}$$

The goal is to get the matrix on the form

$$\left[\begin{array}{cccccc|c} 1 & 0 & \dots & & \dots & 0 & \gamma_1 \\ 0 & 1 & 0 & \dots & & \vdots & \gamma_2 \\ \vdots & 0 & 1 & 0 & \dots & & \vdots \\ & \vdots & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & & \vdots & 0 & 1 & 0 & \gamma_{n-1} \\ 0 & \dots & & \dots & 0 & 1 & \gamma_n \end{array}\right]. \tag{5}$$

Then the system of equation is solved, and the solution is $v_i = \gamma_i$. Performing Gaussian elimination will get the augmented matrix on this form. Being a tridiagonal matrix makes Gaussian elimination simpler in the sense that it requires fewer mathematical operations. The algorithm will now be described in detail.

Each row in the auqmented matrix corresponds to one equation. So we can multiply and divide individual rows, and we can add and subtract rows from each other. The first part of the algorithm is to eliminate the elements in the lower diagonal. We see from eq. (4) that we can eliminate the leading element in the second row, a_1, by subtracting the second row by the first row multiplied by a_1/b_1. This process is called forward substitution. Then the augmented matrix becomes

$$\left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & b_2 - c_1 \cdot \frac{a_1}{b_1} & c_2 - 0 & 0 & \vdots & 0 & g_2 - g_1 \cdot \frac{a_1}{b_1} \\ 0 & a_2 & b_3 & c_3 & 0 & \vdots & g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & & & a_{n-2} & b_{n-1} & c_{n-1} & g_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_n & g_n \end{array}\right]$$

$$\equiv \left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & B_2 & c_2 & 0 & \vdots & 0 & G_2 \\ 0 & a_2 & b_3 & c_3 & 0 & \vdots & g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & & & a_{n-2} & b_{n-1} & c_{n-1} & g_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_n & g_n \end{array}\right],$$

where we have defined $B_2 \equiv b_2 - c_1 \cdot \frac{a_1}{b_1}$ and $G_2 \equiv g_2 - g_1 \cdot \frac{a_1}{b_1}$. Next, the leading element in row three, a_2, can be eliminated by subtracting the third row by the second row multiplied by a_2/B_2.

Then the augmented matrix becomes

$$\left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & B_2 & c_2 & 0 & \vdots & 0 & G_2 \\ 0 & 0 & b_3 - c_2 \cdot \frac{a_2}{B_2} & c_3 - 0 & 0 & \vdots & g_3 - G_2 \cdot \frac{a_2}{B_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & & & a_{n-2} & b_{n-1} & c_{n-1} & g_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_n & g_n \end{array}\right]$$

$$\equiv \left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & B_2 & c_2 & 0 & \vdots & 0 & G_2 \\ 0 & 0 & B_3 & c_3 & 0 & \vdots & G_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & & & a_{n-2} & b_{n-1} & c_{n-1} & g_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_n & g_n \end{array}\right],$$

where we have defined $B_3 \equiv b_3 - c_2 \cdot \frac{a_2}{B_2}$ and $G_3 \equiv \tilde{b}_3 - G_2 \cdot \frac{a_2}{B_2}$. By continuing this pattern, subtracting the current row by the previous row divided by the previous row's leading element and multiplied by the current row's leading element, until all the elements in the lower diagonal have been eliminated, the augmented matrix will have the following form:

$$\left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & B_2 & c_2 & 0 & \vdots & 0 & G_2 \\ 0 & 0 & B_3 & c_3 & 0 & \vdots & G_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & & 0 & B_{n-1} & c_{n-1} & G_{n-1} \\ 0 & 0 & \dots & \dots & 0 & B_n & G_n \end{array}\right]$$

where B_n and G_n are defined in a similar manner as for the previous rows. The next part of the algorithm is to eliminate the elements in the upper diagonal. To do this, we start by eliminating c_{n-1} from row $n-1$. This is done by subtracting row $n-1$ by row n multiplied by c_{n-1}/B_n. This process is called backward substitution. Then the augmented matrix becomes

$$\left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & B_2 & c_2 & 0 & \vdots & 0 & G_2 \\ 0 & 0 & B_3 & c_3 & 0 & \vdots & G_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & & 0 & B_{n-1} & 0 & G_{n-1} - G_n \cdot \frac{c_{n-1}}{B_n} \\ 0 & 0 & \dots & \dots & 0 & B_n & G_n \end{array}\right]$$

$$\equiv \left[\begin{array}{cccccc|c} b_1 & c_1 & 0 & \dots & 0 & 0 & g_1 \\ 0 & B_2 & c_2 & 0 & \vdots & 0 & G_2 \\ 0 & 0 & B_3 & c_3 & 0 & \vdots & G_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & & 0 & B_{n-1} & 0 & G^*_{n-1} \\ 0 & 0 & \dots & \dots & 0 & B_n & G_n \end{array}\right],$$

where $G^*_{n-1} \equiv G_{n-1} - G_n \cdot \frac{c_{n-1}}{B_n}$. By repeating this process for all the other rows in ascending order, the augmented matrix will have the following form in the end:

$$\left[\begin{array}{cccccc|c} b_1 & 0 & 0 & \dots & 0 & 0 & G^*_1 \\ 0 & B_2 & 0 & 0 & \vdots & 0 & G^*_2 \\ 0 & 0 & B_3 & 0 & 0 & \vdots & G^*_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & & 0 & B_{n-1} & 0 & G^*_{n-1} \\ 0 & 0 & \dots & \dots & 0 & B_n & G_n \end{array}\right].$$

Then, finally, we get the solution of the set of equations by dividing each row by its leading element:

$$\left[\begin{array}{cccccc|c} 1 & 0 & \dots & & \dots & 0 & G^*_1/b_1 \\ 0 & 1 & 0 & \dots & & \vdots & G^*_2/B_2 \\ \vdots & 0 & 1 & 0 & \dots & & G^*_3/B_3 \\ & \vdots & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & & \vdots & 0 & 1 & 0 & G^*_{n-1}/B_{n-1} \\ 0 & \dots & & \dots & 0 & 1 & G_n/B_n \end{array}\right] \equiv \left[\begin{array}{cccccc|c} 1 & 0 & \dots & & \dots & 0 & \gamma_1 \\ 0 & 1 & 0 & \dots & & \vdots & \gamma_2 \\ \vdots & 0 & 1 & 0 & \dots & & \gamma_3 \\ & \vdots & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & & \vdots & 0 & 1 & 0 & \gamma_{n-1} \\ 0 & \dots & & \dots & 0 & 1 & \gamma_n \end{array}\right].$$

The augmented matrix is now reduced to the same form as in eq. (5), and the set of equation is solved. The solutions are $v_i = \gamma_i$, for $i = 1, 2, \dots, n$.

1.1 Number of Floating Point Operations

We will now count the number of floating point operations (FLOPs) for this algorithm. First for a general tridiagonal matrix.

Forward substitution: The number of operations done on each row can be seen from for example the one done on the second row, $b_2 - c_1 \cdot \frac{a_1}{b_1}$. The entire row is multiplied by the factor $F = a_1/b_1$. This requires 1 division for each row (this factor is different for each row), and this division does not need to be repeated for each element in the row. The expression becomes $b_2 - c_1 \cdot F$, which amounts to 2 FLOPs, one multiplication and one subtraction. These operations are performed on 3 elements along each row in the augmented matrix. This is done to all rows from row 2 up to and including row n, a total of $n-1$ rows. So the forward substitution requires $(1 + 2 \cdot 3) \cdot (n-1) = 7(n-1)$ FLOPs.

Backward substitution: Again, each row requires one division to get the factor F, and each element operated on will require one multiplication and one subtraction. However, now only 2 elements will be operated on on each row. These operations are done to all rows between row $n-1$ down to and including row 1, again a total of $n-1$ rows. So the backward substitution requires $(1 + 2 \cdot 2) \cdot (n-1) = 5(n-1)$ FLOPs.

Normalization: For the normalization of the diagonal elements, each element along the diagonal is simply set to 1 (requires no FLOPs), and each element along the final column is divided by the element at the diagonal for each row. This is done for all n rows, resulting in a total of n FLOPs for the normalization. The total number of FLOPs required by the algorithm is the sum of the FLOPs from forward substitution, backward substitution and normalization, which is $7(n-1)+5(n-1)+n = 13n-12$ FLOPs.

The number of FLOPs required for the special algorithm is $6n - 4$. However if $B_i = (i+1)/i$ is pre-calculated, this leads to $4n - 4$ FLOPs.

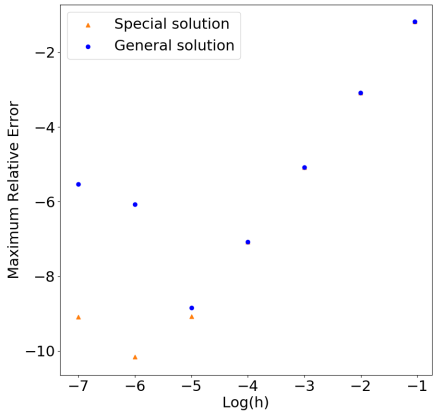


Figure 1: The relative error plotted against log(h) for both the special and general algorithms. As log(h) decreases below -5 the general algorithm results in greater relative error compared to the special algorithm.

Table 1: Table shows the timing of the special and general algorithm for each n. Timing calculated in python script, `main.py`.

n	h	timeSpec	timeGen
10	9.090909e-02	0.000000	0.000000
100	9.900990e-03	0.000000	0.001001
1000	9.990010e-04	0.001998	0.001999
10000	9.999000e-05	0.005059	0.032960
100000	9.999900e-06	0.230106	0.257134
1000000	9.999990e-07	2.040263	2.424836
10000000	9.999999e-08	20.298138	23.899186

2 Results and Discussion

Fig. 1 shows that by decreasing h (increasing n) the numerical solution results in better agreement with the analytic solution. The relative error decreases until log(10) approximately equals -5 and -6 for the general and special algorithms, respectively. Around these points the relative error begins to increase. The relative error initially decreases with decreasing h since the algorithm calculates the derivative at more points between $x = 0$ and $x = 1$. The gradient shows that by decreasing h by a power of 1, leads to the relative error decreasing by a power of 2. However, at h equal to 10^{-5} and 10^{-6} loss of numerical precision begins. Each element of the matrix/array is stored as a double (c++) or float64 (python), hence information is stored in 8 bytes (64 bits). When floating point operations are made, the most significant parts can be lost in order to store the least significant parts of the mantissa. This leads to loss of numerical precision.

Fig. 2 show the comparison between the exact and the numerical solutions. In (a) it is clear the discrepancy between the exact and numerical solutions, however, in other sub-figures it is not easy to distinguish.

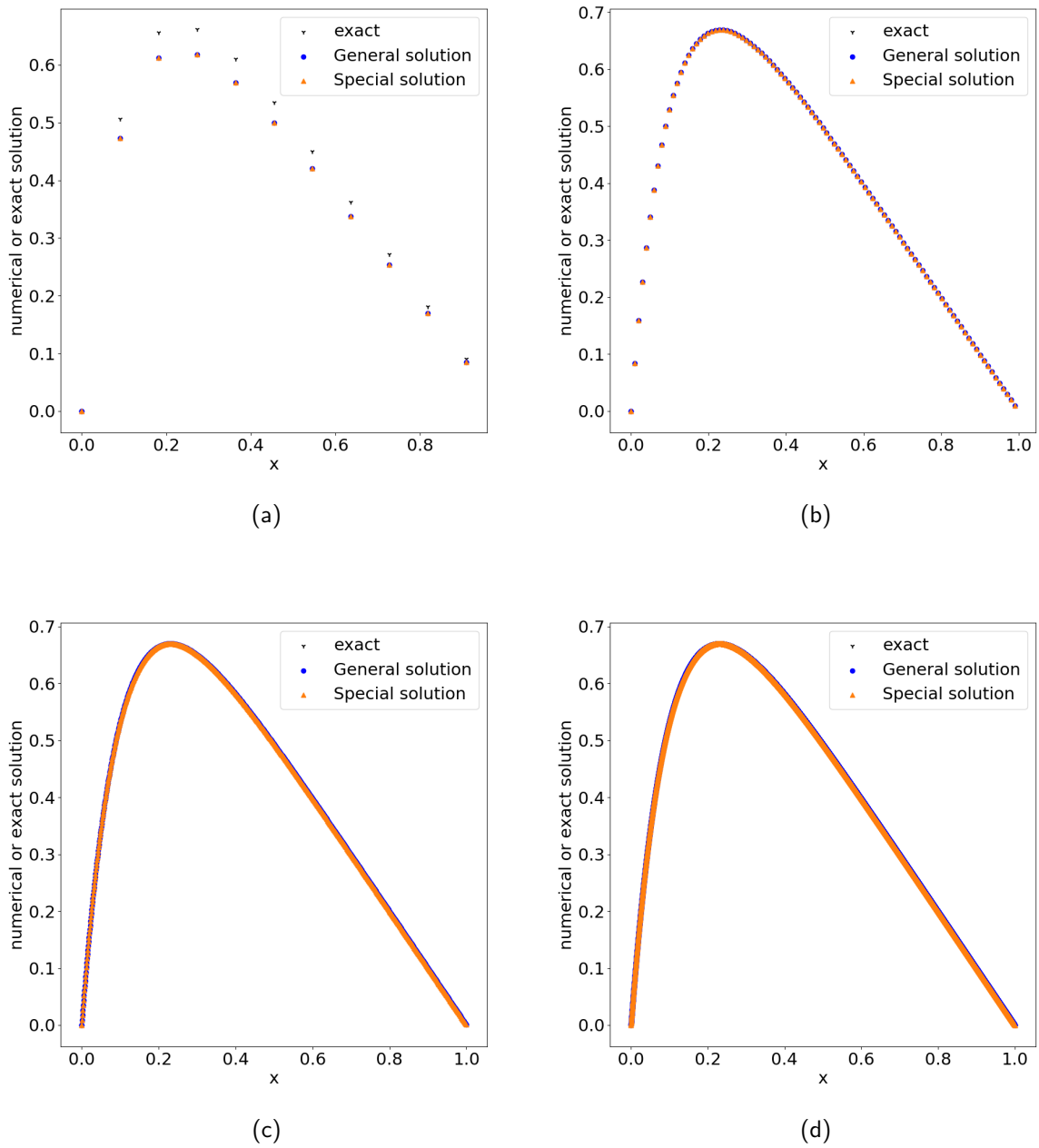
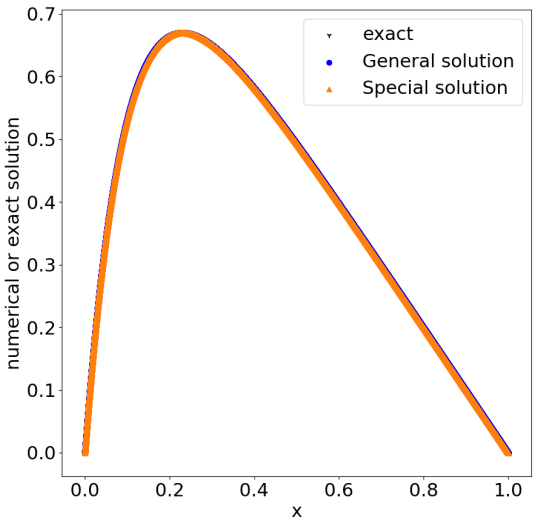
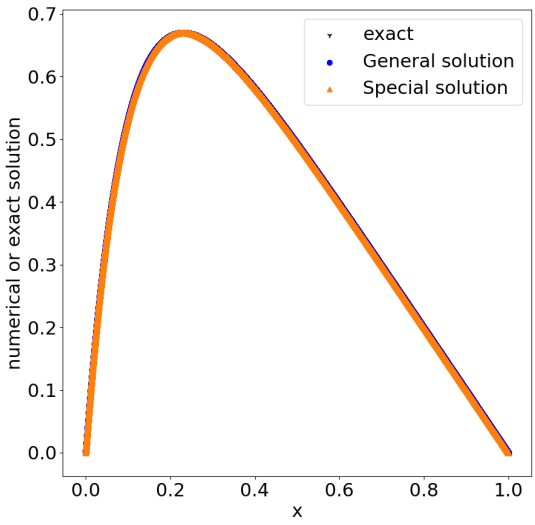


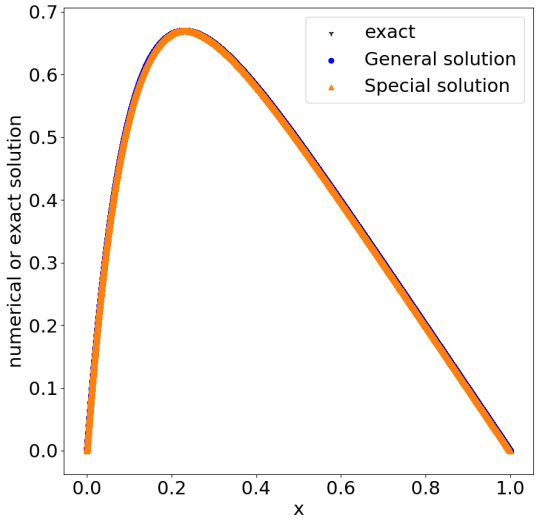
Figure 2: The numerical and exact solutions plotted against x. (a), (b), (c), (d), (e), (f), (g) show the numerical solutions compared to the exact solution at n equal to 10, 100, 1000, 10000, 1000000, 10000000, respectively.



(e)



(f)



(g)

Figure 2: The numerical and exact solutions plotted against x. (a), (b), (c), (d), (e), (f), (g) show the numerical solutions compared to the exact solution at n equal to 10, 100, 1000, 10000, 1000000, 10000000, respectively.