Mini Project; Zwei

TANA21 Beräkningsmatematik

Björn Modée bjomo323
Christopher Olli chrol799
Carl Lidman carli625

Linköping University
MAI
October 2019

1 Statement of the Problem

Solve the numerical solution of a two-point boundary value problem governed by a second-order, linear ordinary differential equation (ODE)

$$y'' = p(x)y'(x) + q(x)y(x) + r(x) \text{ where } x \in (a, b) \tag{1}$$

$$y(a) = \alpha \tag{2}$$

$$y(b) = \beta \tag{3}$$

by first creating a routine to solve a symmetric positive definite (SPD) linear system

$$Ax = b \tag{4}$$

1.1 Linear Solver for Symmetric Positive Definite Matrix

Use a variant of the Gauss-Seidel iterative method known as Successive Over-relaxation (SOR) to solve the linear system (4).

1.1.1 Specific Task

For over-relaxation the parameter $\omega \in (1, 2)$ where $= 1$ corresponds to standard Gauss-Seidel. Vary the value ω and observe its effect on the number of iterations required by SOR to achive a prescribed solution tolerance of $tol = 10^{-9}$.

1.2 Two Point Boundary Value Problem

Use central difference approximations of any derivatives to numerically approximate the solution of the ODE on the given interval with the prescribed boundary conditions:

$$y'' = \frac{-2}{x}y' + \frac{2}{x^2}y + \frac{\sin(\ln(x))}{x^2}, \quad 1 \le x \le 2 \quad \text{with} \quad y(1) = 1 \quad \text{and} \quad y(2) = 2 \tag{5}$$

The ODE has the exact solution

$$y = c_1 x + \frac{c_2}{x^2} - \frac{3}{10}\sin(\ln(x)) - \frac{1}{10}\cos(\ln(x)) \tag{6}$$

with

$$c_1 = \frac{11}{10} - c_2, \quad c_2 = \frac{1}{70}[8 - 12\sin(\ln(2)) - 4\cos(\ln(x))] \tag{7}$$

this exact solution should be used to verify the solution accuracy of the approximate solution.

1.2.1 Specific Task

Discuss the dicretization and assembly of the discrete matrix represented of the ODE using SOR. Verify the second order accuracy of the ODE discretization with increasing sets of nodes $n = 10, 20, 40$ and 80

2 Description of the Mathematics

2.1 Successive Over-relaxation

Using the variant of Gauss-Saidel method Successive Over-relaxation (SOR) results in faster convergence for solving a linear system of equations.
given a square system, of n linear equations with unknown x:

$$Ax = b$$

where:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

then the A matrix can be decomposed into the diagonal component D, the strictly lower triangular component L and the strictly upper triangular component U:

$$A = D + L + U$$

where:

$$D = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}, L = \begin{bmatrix} 0 & 0 & \dots & 0 \\ a_{21} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}, U = \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

The method of SOR is an iterativ technique that solves the left hand side of this expression for x, using the previous value for x on the right hand side. the x_{k+1} can be computed sequentially using forward substitution (8). where the relaxation parameter $\omega \in (0, 2)$ and is equivalent to Gauss-Seidel when $\omega = 1$.

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \tag{8}$$

2.2 Two-Point Boundary Value Problem

we rewrite the (5) function:

$$y'' = p(x_i)f'(x_i) + q(x_i)f(x_i) + r(x_i) \tag{9}$$

$$\frac{1}{h^2}(y_{i+1} - 2y_i + y_{i-1}) = \frac{p(x_i)}{2h}(y_{i+1} - y_{i-1}) + q(x_i)y_{xi} \tag{10}$$

$$\frac{1}{h^2}(y_{i+1} - 2y_i + y_{i-1} - \frac{p(x_i)}{2h}(y_{i+1} - y_{i-1})) - q(x_i)y_{xi} = y_i \qquad (11)$$

where:

$h = \frac{b-a}{N}, \quad N \in [10, 20, 40, 80]$

$x_i = a + ih$

grouping together we get that

$$(-1 - \frac{h}{x}) = \text{Upper}$$

$$(2 + \frac{2h^2}{x^2}) = \text{Diagonal}$$

$$(-1 + \frac{h}{x}) = \text{Lower}$$

which then construct the discrete matrix

$$\begin{pmatrix} (2 + \frac{2h^2}{x^2}) & (-1 - \frac{h}{x}) & \dots & 0 \\ (-1 + \frac{h}{x}) & (2 + \frac{2h^2}{x^2}) & \dots & \vdots \\ \vdots & \vdots & \ddots & (-1 - \frac{h}{x}) \\ 0 & \dots & (-1 + \frac{h}{x}) & (2 + \frac{2h^2}{x^2}) \end{pmatrix} \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N - 1) \end{pmatrix} = -h^2 \begin{pmatrix} t(x_1) \\ t(x_2) \\ \vdots \\ t(x_N - 1) \end{pmatrix} + \begin{pmatrix} (1 + \frac{h}{x})\alpha \\ 0 \\ \vdots \\ 0 \\ (1 - \frac{h}{x})\beta \end{pmatrix}$$

This can now be solved using the SOR method with a tolerance of $tol = 10^{-9}$
and initial guess 1

3 Description of the Algorithm

3.1 Successive Over-relaxation

We choose an initial guess ϕ to the solution

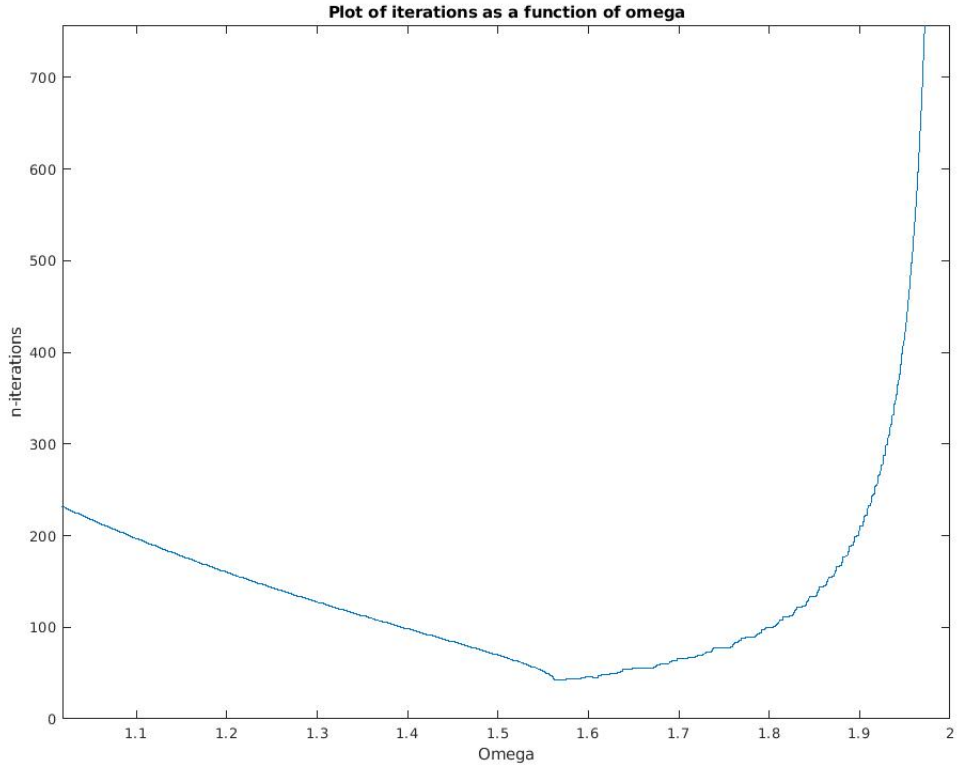
Algorithm:
inputs: Matrix A, solution b, initial guess ϕ, relaxation parameter ω and tolerance tol
outputs: ϕ
for i from 1 until the length n of the A matrix do:
    $\sigma \leftarrow 0$
    for j from 1 until n do:
        if j $\neq$ i then
            $\sigma \leftarrow \sigma + a_{ij}\sigma_j$
        end if
    end j-loop
    $\sigma_i \leftarrow (1 - \omega)\sigma_i + \frac{\omega}{a_{ii}}(b_i - \sigma)$

end i-loop
check if convergence is reached, repeat if not

4 Results

4.1 Successive Over-relaxation

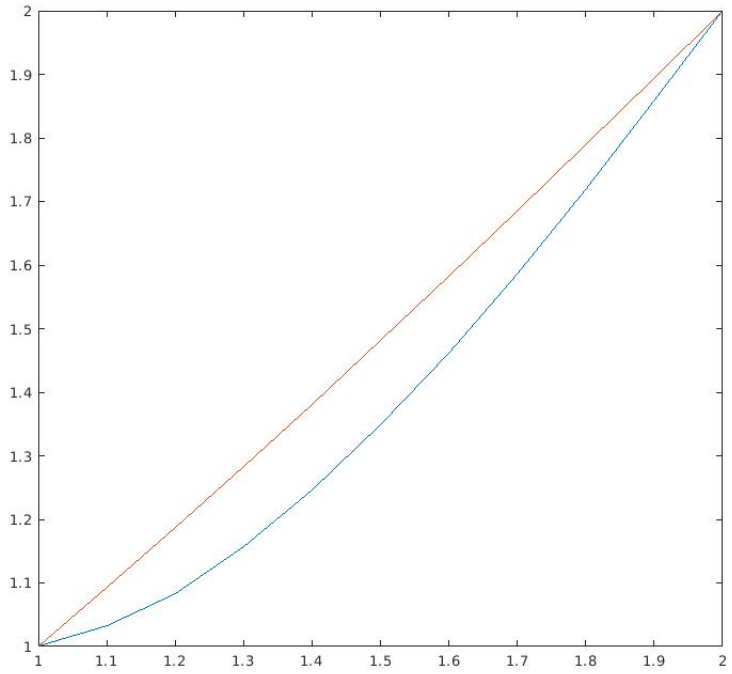
After creating the SOR by the lab instructions to solve a $Ax = b$ matrix we get the expected values for x. The SOR method converges faster depending on the optimal value of ω for the specific matrix. If the ω value is bigger than the optimal ω value the number of iterations needed grows exponentially.



graph 1.

4.2 Two-Point Boundary Value Problem

When solving for $Ax = b$ we get a vector that we compare with the actual values and get the following graph, where the red line is the expected and the red line is our solution.



graph 2.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
1.0000	1.0328	1.0839	1.1564	1.2454	1.3479	1.4618	1.5854	1.7177	1.8578	2.0000

5 Discussion

5.1 Successive Over-relaxation

From the result section we can see that the implementation of the Successive Over-relaxation corresponds to the actual values of the result matrix x. From this we are certain that our implementation is correct. However these matrices is solved using $\omega = 1$ which at this value is equal to the Gauss Seidel algorithm. Using the relaxation factor 1 is not very interesting to us due to the fact that we are looking for faster convergence rather that just convergence. From our result we can see that the least numbers of iterations is achieved when using $\omega \approx 1.56$ for the matrix (as can be seen in graph 2), while still maintaining the same tolerance of error. When evaluating different matrices we have observed that the optimal relaxation factor varies. From this we can't give a complete answer on how to chose ω for general $n \times n$ matrices. However we can say that it differs depending on the matrix and evaluations should be done for each unique matrix.

5.2 Two-Point Boundary Value Problem

We see that the approximation of our function differs with a maximum value of 0.1414. The most likely reason to why this is would be due to wrong assumptions when calculating the matrix A diagonals. More precise, when calculating the second and first derivatives of our functions Y, as well when handling the original function Y when we are to transform the ODE into discrete set of equations. If an error has occurred in this stage, the error will ripple through and increase our fault in contrast to the exact function, especially when the solution is built on approximations of the derivatives. At last there is another factor we always have to take into account when computing on computers, and that is round off errors. These round off errors might furthermore increase the fault. All these factors could result in why our graph does not exactly match the vales of the acquired function.