

# Introduction to Computational Physics – Exercise 5

Simon Groß-Böltung, Lorenz Vogel, Sebastian Willenberg

May 29, 2020

## Numerical linear algebra methods: Tridiagonal matrices

We consider the following tridiagonal  $N \times N$  matrix equation:

$$\underbrace{\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & a_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{N-1} \\ 0 & \cdots & 0 & a_N & b_N \end{pmatrix}}_{=: M = (m_{ij})} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix}}_{=: \vec{x} = (x_j)} = \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-1} \\ y_N \end{pmatrix}}_{=: \vec{y} = (y_i)} \quad (1)$$

**Gauß elimination** is the method of choice when one is interested in the solution  $\vec{x}$  of the linear set of equations  $M\vec{x} = \vec{y}$ , but does not need the information about the inverse matrix  $M^{-1}$  (which we would get using the Gauß-Jordan method). In this case, it is sufficient to convert the matrix  $M$  into an upper (or lower) triangular matrix  $M'$ :

$$M\vec{x} = \vec{y} \xrightarrow{\text{Gauß elimination}} M'\vec{x} = \vec{y}' \quad (2)$$

If we have obtained such a system  $M'\vec{x} = \vec{y}'$  with  $M'$  being in upper triangular form, i.e.  $m'_{ij} = 0$  for  $i > j$ , then we can compute the solution vector  $\vec{x}$  by **back substitution**:

$$x_N = \frac{y'_N}{m'_{N,N}} \quad x_{N-1} = \frac{1}{m'_{N-1,N-1}} (y'_{N-1} - m'_{N-1,N-1}x_N) \quad \cdots \quad x_i = \frac{1}{m'_{ii}} \left( y'_i - \sum_{j=i+1}^N m'_{ij}x_j \right) \quad (3)$$

If we choose the parameters  $a_i = -1$ ,  $b_i = 3$ ,  $c_i = -1$  and  $y_i = 0.2 \forall i$  the solution vector becomes the following:

$$\vec{x} = \begin{pmatrix} 0.12359551 \\ 0.17078652 \\ 0.18876404 \\ 0.19550562 \\ 0.19775281 \\ 0.19775281 \\ 0.19550562 \\ 0.18876404 \\ 0.17078652 \\ 0.12359551 \end{pmatrix} \quad (4)$$

To verify the solution we have implemented the Thomas Algoithm. The solution is the same. The relative difference is calculated in the following way:

$$\left| \frac{M \cdot \vec{x} - \vec{y}}{\vec{y}} \right| \quad (5)$$

We get the following relative difference from our solution to  $\vec{y}$ :

$$\Delta \vec{y} \Rightarrow ( \quad (6)$$

Python-Code 1: Numerical solution of a tridiagonal system of equations

```

1 # -*- coding: utf-8 -*-
2 """
3 Introduction to Computational Physics
4 - Exercise 05: Numerical Linear Algebra Methods
5             Tridiagonal Matrices and Gaussian Elimination
6 - Group: Simon Groß-Böltung, Lorenz Vogel, Sebastian Willenberg
7 """
8
9 import numpy as np; import matplotlib.pyplot as plt
10 from scipy.sparse import diags; from copy import deepcopy
11
12
13 def Gaussian_elimination(A,y):
14     ''' Numerical subroutine for the iterative expression for
15         Gaussian elimination without pivoting '''
16     a, b = deepcopy(A), deepcopy(y)
17     N = np.shape(a)[0]
18     for i in range(N):
19         for k in range(i+1,N):
20             factor = a[k,i]/a[i,i]
21             b[k] -= b[i]*factor
22             for j in range(i,N):
23                 a[k,j] -= a[i,j]*factor
24     return (a,b)
25
26 def Thomas_algorithm(A,y):
27     ''' Numerical subroutine for the Thomas algorithm (a simplified form
28         of Gaussian elimination that can be used to solve tridiagonal
29         systems of equations) '''
30     a, b = deepcopy(A), deepcopy(y)
31     N = np.shape(a)[0]
32     for i in range(N-1):
33         factor = a[i+1,i]/a[i,i]
34         a[i+1,i] -= factor*a[i,i]
35         a[i+1,i+1] -= factor*a[i,i+1]
36         b[i+1] -= factor*b[i]
37     return (a,b)
38
39 def backward_substitution(A,y):
40     ''' Numerical subroutine for the iterative expression for
41         backward substitution '''
42     N = np.shape(A)[0]
43     x = np.zeros(N)
44
45     x[N-1] = y[N-1]/A[N-1,N-1]
46     for i in range(N-2,-1,-1):
47         x[i] = (y[i]-A[i,i+1]*x[i+1])/A[i,i]
48     return x
49
50 def solve_tridiagonal_system(a,b,c,y,method):
51     ''' Numerical subroutine that finds the solution vector x for a
52         tridiagonal equation system Ax=y '''
53     tridiag = diags([a,b,c], [-1,0,1]).toarray() # create tridiagonal matrix
54     if (method=='Gauss'):
55         out = Gaussian_elimination(tridiag,y)
56     elif (method=='Thomas'):
57         out = Thomas_algorithm(tridiag,y)
58     return (tridiag, backward_substitution(out[0],out[1]))
59
60 def relative_error(A,x,y):
61     ''' Function that puts the numerical solution x back into the original
62         matrix equation Ax=y and finds how much the result deviates from the
63         original right-hand-side y '''
64     return abs(np.dot(A,x)-y)/abs(y)
65
66
67 N = 10          # size of the tridiagonal matrix
68 a = -1.*np.ones(N-1)    # values for the diagonal entries a
69 b = 3.*np.ones(N)      # values for the diagonal entries b
70 c = -1.*np.ones(N-1)    # values for the diagonal entries c
71 y = 0.2*np.ones(N)      # values for the right-hand-side vector y
72

```

```
73 tridiag , solution = solve_tridiagonal_system(a,b,c,y,method='Gauss')
74 rel_error = relative_error(tridiag,solution,y)
75 print ('\nGaussian Elimination:\n')
76 print ('Solution vector:\n{}' .format(solution))
77 print ('Relative Error:\n{}' .format(rel_error))
78
79 tridiag , solution = solve_tridiagonal_system(a,b,c,y,method='Thomas')
80 rel_error = relative_error(tridiag,solution,y)
81 print ('\nThomas Algorithm:\n')
82 print ('Solution vector:\n{}' .format(solution))
83 print ('Relative Error:\n{}' .format(rel_error))
```