Introduction to Computational Physics – Exercise 5

Simon Groß-Bölting, Lorenz Vogel, Sebastian Willenberg

May 29, 2020

Numerical linear algebra methods: Tridiagonal matrices

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

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

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':

$$M\vec{x} = \vec{y} \qquad \xrightarrow{\text{Gauß elimination}} \qquad M'\vec{x} = \vec{y}\,' \tag{2}$$

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**:

$$x_N = \frac{y'_N}{m'_{N,N}} \qquad x_{N-1} = \frac{1}{m'_{N-1,N-1}} \left(y'_{N-1} - m'_{N-1,N-1} x_N \right) \qquad \cdots \tag{3}$$

Or in general:

$$x_i = \frac{1}{m'_{ii}} \left(y'_i - \sum_{j>i} m'_{ij} x_j \right) \tag{4}$$

The **Thomas algorithm** is a simplified form of Gauß elimination that can be used to solve tridiagonal systems of equation by creating a upper triangular form M' of the matrix $M = (m_{ij})$:

$$\left.\begin{aligned} y_{i+1} \quad &\longrightarrow \quad y_{i+1} - y_i \frac{m_{i+1,i}}{m_{ii}} \\ m_{i+1,i+1} \quad &\longrightarrow \quad m_{i+1,i+1} - m_{i,i+1} \frac{m_{i+1,i}}{m_{ii}} \\ m_{i+1,i} \quad &\longrightarrow \quad m_{i+1,i} - m_{ii} \frac{m_{i+1,i}}{m_{ii}} \end{aligned}\right\} \quad \text{for } i = 1, \ldots, N \tag{5}$$

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

$$\vec{x} = \begin{pmatrix} 0.12359551 \\ 0.17078652 \\ 0.18876404 \\ 0.19550562 \\ 0.19775281 \\ 0.19775281 \\ 0.19550562 \\ 0.18876404 \\ 0.17078652 \\ 0.12359551 \end{pmatrix} \tag{6}$$

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

$$\left| \frac{M \cdot \vec{x} - \vec{y}}{\vec{y}} \right| \tag{7}$$

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

$$\Delta \vec{y} =) (\tag{8}$$

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

```python
# -*- coding: utf-8 -*-
"""
Introduction to Computational Physics
- Exercise 05:  Numerical Linear Algebra Methods
                Tridiagonal Matrices and Gaussian Elimination
- Group: Simon Groß-Bölting, Lorenz Vogel, Sebastian Willenberg
"""

import numpy as np; import matplotlib.pyplot as plt
from scipy.sparse import diags; from copy import deepcopy


def Gaussian_elimination(A,y):
    ''' Numerical subroutine for the iterative expression for
        Gaussian elimination without pivoting '''
    a, b = deepcopy(A), deepcopy(y)
    N = np.shape(a)[0]
    for i in range(N):
        for k in range(i+1,N):
            factor = a[k,i]/a[i,i]
            b[k] -= b[i]*factor
            for j in range(i,N):
                a[k,j] -= a[i,j]*factor
    return (a,b)

def Thomas_algorithm(A,y):
    ''' Numerical subroutine for the Thomas algorithm (a simplified form
        of Gaussian elimination that can be used to solve tridiagonal
        systems of equations) '''
    a, b = deepcopy(A), deepcopy(y)
    N = np.shape(a)[0]
    for i in range(N-1):
        print(i)
        factor = a[i+1,i]/a[i,i]
        a[i+1,i] -= factor*a[i,i]
        a[i+1,i+1] -= factor*a[i,i+1]
        b[i+1] -= factor*b[i]
    return (a,b)

def backward_substitution(A,y):
    ''' Numerical subroutine for the iterative expression for
        backward substitution '''
    N = np.shape(A)[0]
    x = np.zeros(N)

    x[N-1] = y[N-1]/A[N-1,N-1]
    for i in range(N-2,-1,-1):
        x[i] = (y[i]-A[i,i+1]*x[i+1])/A[i,i]
    return x

def solve_tridiagonal_system(a,b,c,y,method):
    ''' Numerical subroutine that finds the solution vector x for a
        tridiagnonal equation system Ax=y '''
    tridiag = diags([a,b,c], [-1,0,1]).toarray() # create tridiagonal matrix
    if (method=='Gauss'):
        out = Gaussian_elimination(tridiag,y)
    elif (method=='Thomas'):
        out = Thomas_algorithm(tridiag,y)
    return (tridiag, backward_substitution(out[0],out[1]))

def relative_error(A,x,y):
    ''' Function that puts the numerical solution x back into the original
        matrix equation Ax=y and finds how much the result deviates from the
        original right-hand-side y '''
    return abs(np.dot(A,x)-y)/abs(y)


N = 10                    # size of the tridiagonal matrix
a = -1.*np.ones(N-1)      # values for the diagonal entries a
b = 3.*np.ones(N)         # values for the diagonal entries b
c = -1.*np.ones(N-1)      # values for the diagonal entries c
y = 0.2*np.ones(N)        # values for the right-hand-side vector y
```

```python

tridiag, solution = solve_tridiagonal_system(a,b,c,y,method='Gauss')
rel_error = relative_error(tridiag,solution,y)
print('\nGaussian Elimination:\n')
print('Solution vector:\n{}'.format(solution))
print('Relative Error:\n{}'.format(rel_error))

tridiag, solution = solve_tridiagonal_system(a,b,c,y,method='Thomas')
rel_error = relative_error(tridiag,solution,y)
print('\nThomas Algorithm:\n')
print('Solution vector:\n{}'.format(solution))
print('Relative Error:\n{}'.format(rel_error))
```