Exercise sheet 5

by Robin Heinemann (group 4), Paul Rosendahl (group 4)

February 1, 2019

1 Tridiagonal matrices

Consider the following linear system

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \vdots \\ y_{n-2} \\ y_{n-1} \\ y_n \end{pmatrix}$$

where $n \in \mathbb{N}$. This can be solved using Gaussian elimination. To transform the matrix to a upper triangular matrix the following algorithm can be used

$$\begin{aligned} \tilde{b}_1 &= b_1 \\ \tilde{y}_1 &= y_1 \\ \tilde{b}_{i+1} &= b_{i+1} - \frac{a_{i+1}}{\tilde{b}_i} c_i \\ \tilde{y}_{i+1} &= y_{i+1} - \frac{a_{i+1}}{\tilde{b}_i} \tilde{y}_i \end{aligned}$$

This results in the following linear system with the same solution as the original system

$$\begin{pmatrix} \tilde{b}_1 & c_1 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & c_{n-1} \\ 0 & 0 & 0 & \tilde{b}_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \tilde{y}_1 \\ \vdots \\ \vdots \\ \tilde{y}_n \end{pmatrix}$$

This can be solved using a simple backward substitution

$$\begin{aligned} x_n &= \frac{\tilde{y}_n}{\tilde{b}_n} \\ x_{i-1} &= \frac{1}{\tilde{b}_{i-1}} (\tilde{y}_{i-1} - c_{n-1} x_i) \end{aligned}$$

This algorithm is used to solve the linear system where $a_i = -1, b_i = 2, c_i = -1, y_i = 0.1 \forall i \in \{1, \dots, n\}$. The solution is the used to calculate the right-hand-side, which should be 0.1 for every entry. The calculated values comes quite close to the expected 0.1, with the biggest relative error being $3.1 \times 10^{-15} \approx 30 \cdot \text{eps}$. This means the error is still quite close to the minimal possible error. For most applications a error this small will probably be irrelevant.

```rust
use std::ops::{Div, Mul, SubAssign, AddAssign, Sub, DivAssign};

type Vector<T> = Vec<T>;

fn tridiag<T>(a: &Vector<T>, b: &mut Vector<T>, c: &mut Vector<T>, y: &mut Vector<T>) -> Vector<T>
    where T: Div<Output = T> + SubAssign + Mul<Output = T> + Sub<Output = T> + Copy
{
    let n = y.len();

    for i in 0..(n - 1) {
    let f = a[i] / b[i];

    b[i + 1] -= f * c[i];
    y[i + 1] -= f * y[i];
    }

    let mut x = y.clone();

    x[n - 1] = y[n - 1] / b[n - 1];

    for i in 1..n {
    let ii = n - i - 1;

    x[ii] = (y[ii] - c[ii] * x[ii + 1]) / b[ii];
    }

    x
}

fn main() {
    let n = 10;
    let mut a = vec![-1.0; n - 1];
    let mut b = vec![2.0; n];
    let mut c = vec![-1.0; n - 1];
    let mut y = vec![0.1; n];

    let x = tridiag(&a, &mut b, &mut c, &mut y);

    let a = vec![-1.0; n - 1];
    let b = vec![2.0; n];
    let c = vec![-1.0; n - 1];
    let mut yy = vec![0.0; n];

```

```
    yy[0] = b[0] * x[0] + c[0] * x[1];
    yy[n - 1] = a[n - 2] * x[n - 2] + b[n - 1] * x[n - 1];
    for i in 1..(n - 1) {
    yy[i] = a[i - 1] * x[i - 1] + b[i] * x[i] + c[i] * x[i + 1];
    }

    println!("|$x_n$|$y'_n$");
    println!("|-");

    for i in 0..n {
    println!("| {} | {}", x[i], yy[i]);
    }
}
```

Listing 1: rust implementation of gauss elimination for a tridiagonal matrix

Table 1: solution of the linear system and right-hand-side calculated from the solution

x_n	y'_n
0.5	0.09999999999999998
0.9	0.09999999999999987
1.2000000000000002	0.09999999999999987
1.4000000000000006	0.10000000000000031
1.5000000000000007	0.10000000000000009
1.5000000000000007	0.10000000000000031
1.4000000000000004	0.09999999999999987
1.2000000000000002	0.10000000000000009
0.8999999999999999	0.09999999999999976
0.4999999999999999	0.09999999999999987

2 Additional notes

All programs written are written using the programming language *rust*. Extra dependencies (*rust crates*) will be listed in a comment in the first line. To get the source files of each program just unzip this *pdf* file. You will find directories for every program in this file. To execute one of the programs run `cargo run` in it's directory. All plots are made with *gnuplot*. This document was written in *org-mode* and converted to *pdf*. The corresponding *org-mode* sources can also be found by unzipping this *pdf* file.