Exercises V

Robin Greif (Exercise 3 Francisco Aros), Lia Hankla (Exercise 2 Victor Ksoll)

Due 2018/05/25

Exercise 2

1. Gaussian Elimination

Gaussian elimination means putting the matrix into row echelon form, i.e. getting rid of the coefficients a_i, where the set of equations reads $a_i x_{i-1} + b_i x_i + c_i x_{i+1}$ with $i = 1...N$ and $a_1 = c_N = 0$. In the following we do not bother making the diagonal components b_i equal to 1, the usual first step in Gaussian elimination. Doing so would simply result in dividing by a b_i.

Before reduction, an arbitrary two rows will have the form

$$\begin{aligned} a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= y_i \\ a_{i+1} x_i + b_{i+1} x_{i+1} + c_{i+1} x_{i+2} &= y_{i+1} \end{aligned}$$

After reducing the first row (row i) and hence removing a_i, the coefficients read

$$\tilde{b}_i x_i + \tilde{c}_i x_{i+1} = \tilde{y}_i \tag{1}$$

$$a_{i+1} x_i + b_{i+1} x_{i+1} + c_{i+1} x_{i+2} = y_{i+1} \tag{2}$$

We see that for the first row, $\tilde{b}_1 = b_1$, $\tilde{c}_i = c_i$ since this row is the starting point of the algorithm. For the other rows, we now eliminate a_{i+1} by multiplying row $i+1$ by $\tilde{b}_i$ and subtracting row i multiplied by a_{i+1}, leading to:

$$\begin{aligned} (\tilde{b}_i a_{i+1} - a_{i+1} \tilde{b}_i) x_i + (\tilde{b}_i b_{i+1} - a_{i+1} \tilde{c}_i) x_{i+1} + \tilde{b}_i c_{i+1} x_{i+2} &= \tilde{b}_i y_{i+1} - a_{i+1} \tilde{y}_i \\ \Rightarrow \tilde{a}_{i+1} x_{i+1} + \tilde{b}_{i+1} x_{i+1} + \tilde{c}_{i+1} x_{i+2} &= \tilde{y}_{i+1} \end{aligned}$$

where

$$\begin{aligned} \tilde{a}_{i+1} &= 0 & \tilde{b}_{i+1} &= \tilde{b}_i b_{i+1} - a_{i+1} \tilde{c}_i \\ \tilde{c}_{i+1} &= \tilde{b}_i c_{i+1} & \tilde{y}_{i+1} &= \tilde{b}_i y_{i+1} - a_{i+1} \tilde{y}_i \end{aligned}$$

Looping through the rows ($i = 2...N$) yields an echelon form matrix ready for backwards substitution. We call this subroutine `gaussian_elimination`:

```
def gaussian_elimination(a, b, c, y):
    """Given coefficients in an NxN tridiagonal matrix,
       reduce to echelon form.
       a, b, c, y: Nx1 arrays such that
       a_i x_{i-1} + b_i x_i + c_i x_{i+1} = y_i
       Note that a_1 and c_N should be 0.
       """
    N = a.size

    for i in range(1, N):
        y[i] = b[i - 1] * y[i] - a[i] * y[i - 1]
        b[i] = b[i - 1] * b[i] - a[i] * c[i - 1]
        c[i] = c[i] * b[i - 1]
        a[i] = 0
    return b, c, y
```

2. Backwards Elimination

Once the matrix is in echelon form, the last row has only one entry: $b_N x_N = y_N$. We can solve this immediately for $x_N = y_N/b_N$ and use it in the rest of the equations. For the second-to-last row, we have

$$\begin{aligned} b_{N-1}x_{N-1} + c_{N-1}x_N = y_{N-1} \\ \Rightarrow x_{N-1} = \frac{1}{b_{N-1}}(y_{N-1} - c_{N-1}x_N) \end{aligned}$$

where the tildes have been dropped. Generalizing to a row i and looping through the rows backwards ($i = N, N-1, ...2$),

$$x_i = \frac{1}{b_i}(y_i - c_i x_{i+1})$$

where again $c_N = 0$. We name this subroutine `backwards_substitution`:

```
def backwards_substitution(b, c, y):
    """Given coefficients in an NxN echelon form matrix,
    use backwards substitution to solve for x.
    All inputs are Nx1 arrays such that
    b_i x_i + c_i x_{i+1} = y_i.
    Note that c_N should be 0. """
```

```
    N = b.size
    x = np.zeros(N + 1)  # For last row
    for i in range(N - 1, -1, -1):
        x[i] = (y[i] - c[i] * x[i + 1]) / b[i]
    return x[:-1]
```

3. Gaussian Solve

With our two subroutines in hand, the solving itself goes fairly easily:

```
def gaussian_solve(a, b, c, y):
    """Given coefficients in an NxN tridiagonal matrix,
        use gaussian elimination and backwards substitution
        to solve for x.
        All inputs are Nx1 arrays such that
        a_i x_{i-1} + b_i x_i + c_i x_{i+1} = y_i.
        Note that a_0, c_N should be 0.
        """
    b, c, y = gaussian_elimination(a, b, c, y)
    x = backwards_substitution(b, c, y)
    return x
```

4. Example Solve

With $a_i = -1$, $b_i = 2$, $c_i = -1$, and $y_i = 0.1$, we find

$$\vec{x} = (0.5,\ 0.9,\ 1.2,\ 1.4,\ 1.5,\ 1.5,\ 1.4,\ 1.2,\ 0.9,\ 0.5)$$

5. Solver Check

Multiplying the tridiagonal matrix by the solution found above, we achieve relative errors on the order of 10^{-15} and 10^{-16}, which is pretty good.