Jacobi Method

April 23, 2022

1 Jacobi Method

In numerical linear algebra, the Jacobi method is **an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations.** Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges.

Given a linear system of equation and initial values for unknowns, assume zeros if there are no initial values.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3
\end{aligned}
$$

solitions are

$$
\begin{aligned}
x_1 &= \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \\
x_2 &= \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \\
x_3 &= \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}}
\end{aligned}
$$

Substitute with the initial values and repaet `n` times

1.1 Implementation

We define the `jacobi_method` function. The function accepts the matrices A and b. Where A is the coefficient matrix and b is the column vector of constant terms. Solves the equation $Ax = b$ and return an array of the solutions.

[1]:
```
# import needed packages

import numpy as np
```

```
def jacobi_method(A, b, number_of_iterations = 10, guess = None):

    # User enters the initial guess. If none, the default initial value is zeros
    if guess is None:
        guess = np.zeros(A.ncols());


    diagonal_elements = np.diagonal(A)

    matrix_without_diagonal_elements = A - np.diagflat(diagonal_elements)


    for i in range(number_of_iterations):
        guess = (b - np.dot(matrix_without_diagonal_elements, guess)) /␣
↪diagonal_elements

    return guess
```

1.2 Test with an example

Use the Jacobi method to solve

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

with number of iterations = 3

[2]:
```
A = matrix([[3,-0.1,-0.2], [0.1,7,-0.3], [0.3,-0.2,10]])

b = np.array([7.85, -19.3, 71.4])

jacobi_method(A, b, 3)
```

[2]:
```
array([ 3.00080635, -2.49973844,  7.00020667])
```

The exact solution is

[3]:
```
A.solve_right(b)
```

[3]:
```
(3.00000000000000, -2.50000000000000, 7.00000000000000)
```