project3_demo2

December 3, 2024

1 Linear Systems

In `project3_demo1.ipynb`, we have learned how to use `scipy.linalg.solve` to solve the linear equation $Ax = b$.

In this notebook, we will learn how to solve linear systems step by step. We consider two special matrices first: lower triangular mastrix and upper triagnular matrix.

References: * https://books.google.com.tw/books?id=f6Z8DwAAQBAJ&hl=zh-TW * https://docs.scipy.org/doc/scipy/reference/linalg.html

[1]:
```
import numpy as np
from scipy import linalg # just for checking
```

1.1 Lower Triagnular Matrix

Implment the solver to solve the solution with a lower trianuglar matrix.

[7]:
```
def solveLowerTriangular(L,b):
    """
    Solve a linear system with a lower triangular matrix L.

    Arguments:
    L -- a lower triangular matrix
    b -- a vector

    Returns:
    x -- the solution to the linear system
    """
    n  = len(b)
    x  = np.zeros(n)

    # TODO: implement the algorithm
    for i in range(n):
        if L[i, i] == 0:
            raise ValueError("Matrix is singular")
        x[i] = (b[i] - np.dot(L[i, :i], x[:i])) / L[i, i]
    return x
```

1.2 Exercise 1: Lower Triangular Matrix

Write a python progem to solve this lower triangular matrix. Do NOT use `scipy.linalg`. Your lower triangular matrix solver should be general for $N \times N$ matrix.

$$\begin{bmatrix} -1 & 0 & 0 \\ -6 & -4 & 0 \\ 1 & 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -6 \\ 3 \end{bmatrix}$$

[ ]:
```
L = np.array([[-1,0,0],[-6,-4,0],[1,2,2]])
b = np.array([1,-6,3])
print(L,b)
```

```
[[-1  0  0]
 [-6 -4  0]
 [ 1  2  2]] [ 1 -6  3]
```

[9]:
```
tmp = b
x = solveLowerTriangular(L,tmp)
print(x)
print(b)
```

```
[-1.  3. -1.]
[ 1 -6  3]
```

Check if

$$L \cdot x = b$$

[10]:
```
# check L x = b
print(np.dot(L,x))
print(b)
```

```
[ 1. -6.  3.]
[ 1 -6  3]
```

[11]:
```
# check L^-1 b = x
print(linalg.inv(L).dot(b))
print(x)
```

```
[-1.  3. -1.]
[-1.  3. -1.]
```

[12]:
```
# compare with solution from scipy
print(linalg.solve(L,b))
```

```
[-1.  3. -1.]
```

1.3 Upper Triagnular Matrix

1.4 Exercise: Solving upper trianular matrix

Write a python progem to solve the upper triangular matrix. Do NOT use `scipy.linalg`. Your upper triangular matrix solver should be general for $N \times N$ matrix.

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix}$$

[13]:
```python
def solveUpperTriangular(U,b):
    """
    Solve a linear system with an upper triangular matrix U.

    Arguments:
    U -- an upper triangular matrix
    b -- a vector

    Returns:
    x -- the solution to the linear system

    """
    n  = len(b)
    x  = np.zeros(n)

    # TODO: implement the algorithm
    for i in range(n - 1, -1, -1):  # Start from the last row and move upward
        if U[i, i] == 0:
            raise ValueError("Matrix is singular")

        # Calculate x[i] using known values
        x[i] = (b[i] - np.dot(U[i, i + 1:], x[i + 1:])) / U[i, i]

    return x
```

[14]:
```python
U = np.array([[1,2,2],[0,-4,-6],[0,0,-1]])
b = np.array([3,-6,1])
print(U,b)
```

```
[[ 1  2  2]
 [ 0 -4 -6]
 [ 0  0 -1]] [ 3 -6  1]
```

[15]:
```python
x = solveUpperTriangular(U,b)
print(x)
```

```
[-1.  3. -1.]
```

[16]:
```
# check U x = b
print(np.dot(U,x))
print(b)
```

```
[ 3. -6.  1.]
[ 3 -6  1]
```

[17]:
```
# check U^-1 b = x
print(linalg.inv(U).dot(b))
print(x)
```

```
[-1.  3. -1.]
[-1.  3. -1.]
```

[18]:
```
# compare with solution from scipy
print(linalg.solve(U,b))
```

```
[-1.  3. -1.]
```