# project3_demo3

December 3, 2024

## 1 Linear systems

[25]:
```
import numpy as np
from mylinalg import solveLowerTriangular, solveUpperTriangular, lu, lu_solve
```

## 2 Exercise: LU decomposition

Write a python program to solve it. Do not use any linear algebra packackes. Use your own linear algebra solvers in `mylinalg.py`.

$$Ax = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = b$$

[26]:
```
A = np.array([[2,4,-2],[4,9,-3],[-2,-3,7]])
```

[27]:
```
l, u = lu(A)
```

[28]:
```
print(l,u)
```

```
[[ 1.  0.  0.]
 [ 2.  1.  0.]
 [-1.  1.  1.]] [[ 2.  4. -2.]
 [ 0.  1.  1.]
 [ 0.  0.  4.]]
```

Check $LU = A$

[29]:
```
print(np.dot(l,u))
print(A)
```

```
[[ 2.  4. -2.]
 [ 4.  9. -3.]
 [-2. -3.  7.]]
[[ 2  4 -2]
 [ 4  9 -3]
 [-2 -3  7]]
```

```
[30]: b = np.array([2,8,10])
      x = lu_solve(A,b)
      print(x)
```

```
[-1.  2.  2.]
```

compare your solution with scipy

```
[31]: from scipy.linalg import lu as scipy_lu
      from scipy.linalg import lu_factor as scipy_lu_factor
      from scipy.linalg import lu_solve as scipy_lu_solve
      from scipy.linalg import solve as scipy_solve
```

```
[32]: P, L, U = scipy_lu(A)

      # A = PLU
      print(np.dot(P,np.dot(L,U)))
      print(A)
```

```
[[ 2.  4. -2.]
 [ 4.  9. -3.]
 [-2. -3.  7.]]
[[ 2  4 -2]
 [ 4  9 -3]
 [-2 -3  7]]
```

```
[33]: b = np.array([2,8,10])
      lu, piv = scipy_lu_factor(A)
      x = scipy_lu_solve((lu, piv), b)
      print(x)
```

```
[-1.  2.  2.]
```

```
[34]: x = scipy_solve(A,b)
      print(x)
```

```
[-1.  2.  2.]
```

## 3 Apply to the Laplace's equation

Copy your previous codes in `project3_demo1.ipynb` but use your own linear algebra solver.

```
[35]: import numpy as np
      from scipy.sparse import dia_array  # if dia_array is not able, use dia_matrix
      from scipy.sparse import dia_matrix
      from numba import jit, njit, prange
      import matplotlib.pyplot as plt
      from scipy import linalg
      from scipy.linalg import solve
```

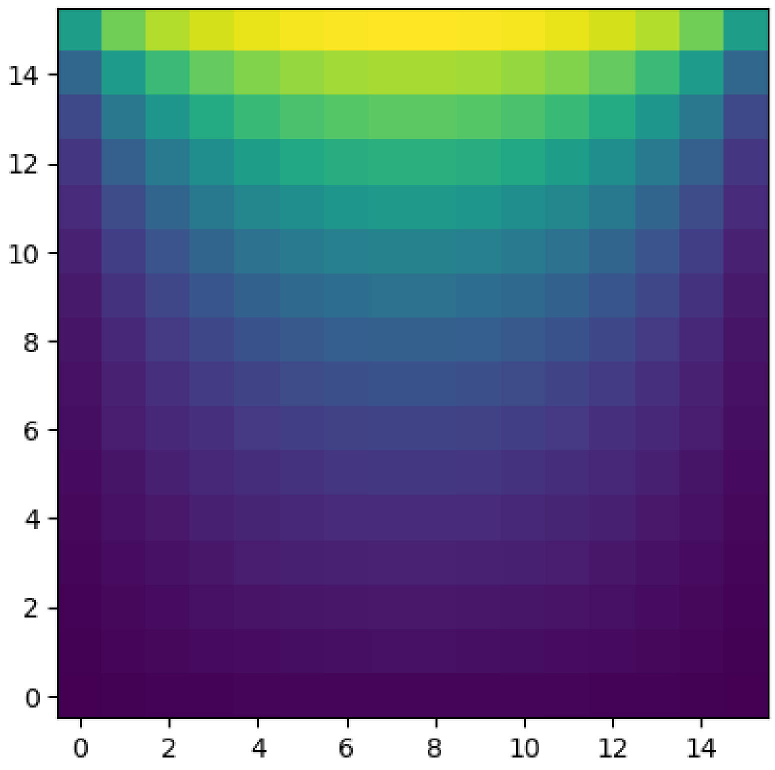
[36]:
```
# Copy your function from the previous notebook here
def generate_the_laplace_matrix_with_size(N=4):
    """
    assume sqrt(N) is an integer.

    """
    nsq = N*N
    A   = np.zeros((nsq,nsq))

    # TODO
    for i in range(N):
        for j in range(N):
            index = i * N + j
            A[index, index] = 4
            if j > 0:
                A[index, index - 1] = -1
            if j < N - 1:
                A[index, index + 1] = -1
            if i > 0:
                A[index, index - N] = -1
            if i < N - 1:
                A[index, index + N] = -1

    return A

def generate_the_rhs_vector_with_size(N=4):
    b = np.zeros(N*N)
    #TODO
    b[-N:] = 1   # Setting the boundary conditions for the first few points
    return b

def convert_solution(x):
    usize = np.sqrt(len(x))
    u = x.reshape(int(usize),int(usize)).transpose()
    return u
```

[37]:
```
def solve_laplace(N=16):
    u = np.zeros((N,N)) # A place holder for the solution

    # TODO Copy your solver here
    A = generate_the_laplace_matrix_with_size(N=N)
    b = generate_the_rhs_vector_with_size(N=N)
    x = lu_solve(A,b)
    u = convert_solution(x)

    return u
```

[38]:
```
u = solve_laplace(N=16)
```

[39]:
```
plt.imshow(u.T,origin="lower")
```

[39]:
```
<matplotlib.image.AxesImage at 0x23c3bc15b90>
```



You could see that our solver is much slower than `scipy.linalg`. Could you speed it up?