Liner Equation and LU decomposition

Shivam Kumaran , SC17B122 09/10/2020

Assignment

Given Matrix equation

$$\mathbf{MX} = \mathbf{b}$$

Get Augmented matrix $[\mathbf{M}|\mathbf{b}]$
convert it into Echelon Form

0.1 Echelon Form

Algorithm for converting any $m \times n$ matrix

- for i the iteration -
  - consider a submatrix considering (ith row to all row)*(ith col to all col)

Listing 1: Python Code for the Plot

```
def calc_ecl(m):
    temp_mat = np.copy(m)
    for i in range(len(temp_mat)):
        temp_mat[i:,i:] = update_sub_mat(temp_mat[i:,i:])
    return(temp_mat)
```

  - Using 1st row of this submatrix, for all the subsequent rows of this :

```
def update_sub_mat(mat):
    m_temp = []
    m_temp.append(mat[0])
    for i in range(1,np.shape(mat)[0]):
        m_temp.append(update_row(mat[i],mat[0]))
    return (np.asarray(m_temp) )
```

    * Make 1st element 0

```
def update_row(r2,r1):
    coeff = r2[0]/r1[0]
    r_temp = r2 - coeff*r1
    return(r_temp)
```

Listing 2: Python Code for the Plot

- do this starting from lrgest submatrix , the given matrix itself.

0.2 Linear Equation

- Convert given equation into augmented matrix form
- Convert this augmented matrix to Echelon Form
- use back substitution to solve the given matrix

0.2.1 Back-substitution

Algorithm:

- for i:[n-1,0]
- - Consider i^{th} row of the augmented matrix and starting from ith element of this row
  - this will be $[a_{ii}, \ldots, a_{in}|b_n]$
  - assume at ith iteration we have soltion till $[x_{i+1}, \ldots x_n]$, and we need to calculate x_i we are starting from calculating x_n and moving towards x_o
  - pass this to $calc\_row\_sol()$
    * extract coefficient vector from (i+1)th element of this row upto 2nd last term $coeff\_vec = [a_{i,(i+1)}, \ldots, a_{i,n}]$
    * $const = b_i$
    * now, x_i will be simply $x_i = (b_i - coeff\_vec \cdot sol\_vec)/a_{ii}$
    * Augment this x_i to the starting of sol_vec

implementation:

```
def calc_row_sol(mat_vec,sol_vec):
        sol = (mat_vec[-1] - np.dot(mat_vec[1:-1],sol_vec))/mat_vec[0]
        return(sol)

def solve_mat_eqn(m,b_vec):
    if(np.shape(m)[0]!=len(b_vec)):
        raise ValueError('Matrix and vector dim mismatch')
    else:
        mat_given = np.column_stack((m,b_vec))
        mat = calc_ecl(mat_given)
        sol_vec = []
        for i in reversed(range(np.shape(mat)[0])):
```

```
            sol_vec.insert(0,calc_row_sol(mat[i,i:],sol_vec))
        return(sol_vec)
```

0.3 LU decomposition

Factorizing a given matrix in Lower and Upper triangle matrices :

0.3.1 Using Equation solution

Here I will use the linear equation solution developed in previous section

$$\mathbf{M} = \mathbf{L} \times \mathbf{U}$$

Algorithm

- Convert $\mathbf{M}$ in echleon form , and assign it to $\mathbf{U}$
- transpose $\mathbf{U}$ say , $\mathbf{U}^T$
- Now ith row of $\mathbf{L}$ will be simply the solution of the matrix equation

  $$\mathbf{U}^T \times \mathbf{x} = m$$

  where m is the i^{th} row of $\mathbf{M}$

implementation

```
from ecln import calc_ecl
from linear_solver_v2 import solve_mat_eqn

def lu_decomp(m):
    u = calc_ecl(m)
    u_t = (np.copy(u)).transpose()
    l = [solve_mat_eqn(u_t,m_vec) for m_vec in m]
    return(np.asarray(l),u)
```

0.3.2 Doolittle's Algorithm

:

- Initialize l_{nxn} and u_{nxn} as Zero matrices
- make diagonal elements of l = 1
- For $k = 1 : n$

- for $m = k, k+1, \ldots, n$

$$u_{k,m} = a_{k,m} - \sum_{j=1}^{k-1} l_{kj} u_{jm}$$

- for $i = k+1, \ldots, m$

$$l_{ik} = \frac{a_{ik} - \sum_{j=1}^{k-1} l_{ij}, u_{jk}}{u_{kk}}$$

```python
import numpy as np

'''
lower and upper triangle decomposition
using Doolittle's Algorithm
'''

def lu_decomp_doolittle(m):
    m = np.asarray(m)
    n = np.shape(m)[0]
    l = np.zeros((3,3))
    u = np.zeros((3,3))

    for k in range(n):
        for m in range(k,n):
            s = 0
            for j in range(k):
                s += l[k][j]*u[j][m]
            u[k][m] = a[k][m] - s

        for i in range(k,n):
            if(i==k):l[i][k] = 0
            else:
                s = 0
                for j in range(k):
                    s = l[i][j]*u[j][k]
                l[i][k] = (a[i][k]-s)/u[k][k]

    return(np.asarray(l),np.asarray(u))


a = [[2.0,3.0,-4.0],[1.0,5.0,-1.0],[3.0,7.0,-3.0]]
l , u = lu_decomp_doolittle(a)

print('lower:\n' , l)
print('upper:\n', u)
```