```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#
#Created on Thu Nov 30 11:23:24 2017
#
#@author: christianehelzel
#
```

1 9. Vorlesung

1.1 Wiederholung:

Bestimme zu einer gegebenen Matrix eine Zerlegung der Form PA = LR, mit P Permutationsmatrix, L untere Dreiecksmatrix mit 1en auf der Diagonale und R obere Dreicksmatrix (U upper matrix)

```
import numpy as np
import scipy.linalg

A = np.array([ [7., 3, -1, 2], [3, 8, 1, -4], \
               [-1, 1, 4, -1], [2, -4, -1, 6] ])
P, L, U = scipy.linalg.lu(A)
```

ACHTUNG: SciPy erstellt die Matrix P der LR-Zerlegung so, dass A=P*L*U gilt.

Unsere Funktionen werden P so berechnen, dass P*A=L*U gilt.

.

Wir wollen nun selbst einen Algorithmus zur Berechnung dieser Zerlegung implementieren.

Wir betrachten zunächst eine Zerlegung der Form A = LR

```
def LR2(A):
    import numpy as np

    n = A.shape[0]
    L = np.eye(n)
    # Damit A nicht überschreiben wird und es keine Fehler mit Integern gibt
    R = A.astype('float')

    for k in range(n - 1):
        for j in range(k+1,n):
            L[j,k] = R[j,k]/R[k,k]
            R[j,k:] = R[j,k:]-L[j,k]*R[k,k:]

    return (L,R)
```

Lösung linearer Gleichungssysteme Ax=b unter Verwendung der LR Zerlegung:

1. löse L y = b
2. löse R x = y

```python
# Löse R x = b
def SolveR(R,b):
    import numpy as np

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

    x[-1] = b[-1]/R[-1,-1]
    for k in range(n-2,-1,-1):
        x[k] = (b[k]-np.sum(R[k,k+1:]*x[k+1:]))/R[k,k]
    return x



# Löse Lx = b
def SolveL(L,b):
    import numpy as np

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

    x[0] = b[0]/L[0,0]
    for k in range(1,n):
        x[k] = (b[k]-np.sum(L[k,:k]*x[:k]))/L[k,k]
    return x
```

1.2 LR Zerlegung mit Zeilentausch

```python
def PLR(A):
    import numpy as np

    n = A.shape[0]
    L = np.eye(n)
    P = np.eye(n)
    R = A.astype('float')

    for k in range(n - 1):
        p = np.argmax(abs(R[k:n,k])) # finde Pivotelement
        # Tausche Zeilen von R, L, P
        R[[k,p+k]] = R[[p+k, k]]
        P[[k,p+k]] = P[[p+k,k]]
        L[[k,p+k]] = L[[p+k,k]]
        # Tausche Spalten von L
        L.T[[k,p+k]] = L.T[[p+k,k]]

        for j in range(k+1,n):
            L[j,k] = R[j,k]/R[k,k]
            R[j,k:] = R[j,k:]-L[j,k]*R[k,k:]

    return (P,L,R)
```

Quelle: Trefethen und Bau, Lecture 21 (siehe "Literaturempfehlung" auf der Vorlesugnsseite)
Beispiel

```
A = np.array([ [ -1, 1, 4, -1], [3, 8, 1, -4],\
               [7., 3, -1, 2], [2, -4, -1, 6] ])

P1, L1, R1 = PLR(A)

A = np.random.uniform(0, 1, (10, 10))
P,L,R = PLR(A)
## Probe
print(np.max(P@A-L@R))
```

```
2.22044604925e-16
```

Jede nichtsinguläre Matrix A hat eine Zerlegung der Form PA=LR.

Nicht jede nichtsinguläre Matrix hat eine Zerlegung der Form A=LR

```
import numpy.linalg as npl

A = np.array([[0,1],[1,1]])
npl.det(A)

L,R = LR2(A)
print(L)
print(R)

P,L,R = PLR(A)
print(P)
print(L)
print(R)
```

```
[[  1.   0.]
 [ inf   1.]]
[[  0.   1.]
 [ nan -inf]]
[[ 0.  1.]
 [ 1.  0.]]
[[ 1.  0.]
 [ 0.  1.]]
[[ 1.  1.]
 [ 0.  1.]]
/home/troll/miniconda3/bin/pweave:11: RuntimeWarning: divide by zero
encountered in double_scalars
/home/troll/miniconda3/bin/pweave:12: RuntimeWarning: invalid value
encountered in multiply
```

Die PA=LR Zerlegung ist weniger anfällig gegenüber Rundungsfehler als die A =LR Zerlegung

Betrachte dazu das folgende Beispiel

```
A = np.array([[1.E-20,1.],[1.,1.]])
L1,R1=LR2(A)
# Überprüfe, ob L1@R1 = A gilt
print(L1@R1-A)
```

```
[[ 0.  0.]
 [ 0. -1.]]
```

Und nun mit Zeilentausch

```
P,L2,R2=PLR(A)
# Überprüfe, ob L2@R2 = PA gilt
print(L2@R2 - P@A)
```

```
[[ 0.  0.]
 [ 0.  0.]]
```

Noch ein Beispiel

```
A = np.array([[1.,0,0,0,1],[-1,1,0,0,1],[-1.,-1,1,0,1],\
              [-1,-1,-1,1,1],[-1,-1,-1,-1,1]])

P,L,R = PLR(A)
print(P)
print(L)
print(R)
```

```
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
[[ 1.  0.  0.  0.  0.]
 [-1.  1.  0.  0.  0.]
 [-1. -1.  1.  0.  0.]
 [-1. -1. -1.  1.  0.]
 [-1. -1. -1. -1.  1.]]
[[  1.   0.   0.   0.   1.]
 [  0.   1.   0.   0.   2.]
 [  0.   0.   1.   0.   4.]
 [  0.   0.   0.   1.   8.]
 [  0.   0.   0.   0.  16.]]
```

Die Werte in der letzten Spalte von "R" werden immer größer

1.3 Löse lineares Gleichungssystem A x = b

Beispiel:

10 x - 7 y = 7

-3x + 2y + 6z = 4

5x - y + 5z = 6

```
import numpy as np

A = np.array([[10., -7, 0], [-3, 2, 6], [5,-1,5]])
b = np.array([7., 4, 6])

P,L,R = PLR(A);

# Löse L y = Pb
```

```
y = SolveL(L,P@b)
# Löse R x = y
x = SolveR(R,y)
```

Probe:

```
print(A@x-b)
```

```
[ 0.  0.  0.]
```

2 Inverse Iteration:

Berechne betragskleinsten Eigenwert einer symmetrischen Matrix und den zugehörigen Eigenvektor

2.1 1. Naive Implementation (ohne Verwendung der LR-Zerlegung)

```
import numpy as np
import numpy.linalg as npl

A = np.random.uniform(0, 1, (100, 100))
A = A+A.T  # A ist eine symmetrische Matrix

v = np.random.uniform(0,1,(100,1)) # Zufälliger Anfangsvektor
v = v/npl.norm(v);

for k in range(100):
    v = npl.solve(A,v)
    v = v/npl.norm(v)

# Ohne "float" wäre eig ein 1x1-Array
eig = float(v.T @ A @ v)

print(A@v-eig*v)
```

```
[[  1.40598193e-10]
 [  8.51995032e-10]
 [ -1.24651556e-10]
 [ -1.27745719e-10]
 [ -1.78482955e-10]
 [ -7.19578451e-11]
 [  5.40334069e-10]
 [  3.92557507e-10]
 [  6.13463544e-10]
 [ -1.59456823e-12]
 [ -9.46076995e-10]
 [  2.78914884e-10]
 [ -1.08660542e-09]
 [ -3.85217778e-11]
 [  7.98923467e-10]
 [ -9.89010452e-10]
 [ -6.68818306e-10]
```

```
[  6.86593048e-11]
[ -1.36837142e-10]
[ -1.82237182e-10]
[ -9.37096716e-10]
[  1.35658230e-10]
[  4.08263950e-10]
[ -1.77825869e-10]
[  3.61782556e-10]
[  5.94052006e-10]
[  4.18998575e-10]
[  6.17393893e-10]
[ -2.09409892e-10]
[ -7.08560263e-10]
[ -3.85643358e-11]
[  6.35842108e-11]
[ -7.90067269e-10]
[  4.09774971e-10]
[  2.91562735e-10]
[  3.60216051e-10]
[ -3.16869593e-10]
[  6.43634129e-10]
[ -6.68973748e-11]
[ -4.27782813e-10]
[ -7.77358872e-10]
[ -1.47397766e-10]
[ -1.80859823e-10]
[  2.52371281e-10]
[  4.90087722e-10]
[ -2.52475424e-10]
[  5.32629985e-10]
[  7.39912871e-11]
[  2.83641250e-10]
[  4.26325763e-10]
[  2.85047853e-11]
[ -3.36686371e-10]
[ -4.43900482e-10]
[ -3.44391828e-10]
[  2.60241096e-10]
[ -3.29315553e-10]
[  1.41380980e-10]
[ -4.75055328e-10]
[ -3.56356217e-10]
[ -2.14555719e-10]
[ -1.32243663e-09]
[  3.92315819e-10]
[  1.03985182e-09]
[ -2.76914002e-10]
[ -5.60276516e-10]
[  7.54461387e-10]
[ -1.06693825e-09]
[  7.44434154e-11]
[ -5.31816898e-10]
[  3.02223177e-12]
[ -2.21415387e-10]
[ -9.45177443e-10]
[ -6.64775027e-10]
[  1.25109578e-10]
[  5.41835981e-10]
[ -4.18722475e-10]
```

```
 [ -1.98359183e-10]
 [ -9.19295317e-11]
 [  3.61415964e-10]
 [  2.17952646e-10]
 [  5.45062142e-11]
 [  3.23860661e-11]
 [ -2.61069695e-10]
 [  3.53090720e-10]
 [  5.94360129e-10]
 [  8.79130532e-10]
 [  3.39306222e-10]
 [  4.02004090e-10]
 [  2.18246704e-10]
 [  5.01842948e-10]
 [ -1.35736289e-10]
 [ -4.90062150e-10]
 [ -1.64785330e-10]
 [  7.99050134e-10]
 [  2.20054096e-10]
 [  6.41146492e-10]
 [ -6.51949602e-10]
 [ -6.58428735e-10]
 [  9.04690174e-10]
 [  5.42637843e-10]]
```

2.2 2. Verwende LR Zerlegung der Matrix

```python
import numpy as np
import numpy.linalg as npl

A = np.random.uniform(0, 1, (100, 100))
A = A+A.T

P, L, R = PLR(A)
v = np.random.uniform(0,1,(100,1))
v = v/npl.norm(v);

for k in range(100):
    y = SolveL(L,P@v)
    v = SolveR(R,y)
    v = v/npl.norm(v)

eig = float(v.T @ A @ v)

print(A@v-eig*v)
```

```
[  9.77489574e-16  -7.21943122e-16  -4.32786402e-16  -5.03069808e-17
   5.85252333e-16   2.54787511e-16   2.86771475e-16   1.70545002e-16
  -1.93381010e-16  -3.19622800e-16  -2.70942123e-16  -4.80084722e-16
   2.38216158e-16  -5.96311195e-17  -2.55749740e-16   2.04588950e-16
  -1.55799852e-16   5.49311031e-16  -2.00197931e-16   3.05324884e-16
  -6.70036943e-17  -4.68483759e-16   7.45931095e-17   8.17488438e-17
  -1.86157513e-16  -3.54263060e-17  -5.00901404e-17  -1.49728320e-16
  -3.64752716e-16  -5.75851961e-16  -4.57750157e-16  -1.42626796e-16
  -2.00794242e-16  -2.53703308e-16   8.67361738e-18   5.78435412e-16
  -3.44559450e-16  -7.12320827e-17   2.85921054e-16   2.19794885e-16
  -2.27248775e-16  -2.51262159e-16  -2.39608680e-17  -1.02131845e-16
```

```
 3.90312782e-16  -2.27140355e-17   1.40641351e-16  -6.21979681e-16
-3.84078620e-17  -4.50594423e-16  -1.41163123e-16  -9.02489888e-16
-9.31302561e-16   1.53739868e-16  -7.32378568e-17   4.74141939e-16
-1.12702816e-16  -2.89075404e-17   3.16803875e-16  -5.52401007e-17
-2.57714856e-16   3.35885833e-16   2.74845251e-16   2.36491599e-16
-1.23013536e-16  -9.23902881e-16  -3.69726493e-16  -1.65232411e-16
-6.54153381e-16   2.99890321e-16  -3.91180144e-16  -3.76489204e-16
-1.79626889e-16   1.24574830e-16  -2.41777084e-16   2.77867464e-16
-1.26174028e-16   1.24683250e-16  -4.96673015e-16  -1.16605944e-16
-5.88775990e-16   8.37004077e-17  -5.99997482e-16  -3.06693690e-16
 5.42189178e-16  -7.86561571e-16  -5.80427633e-16   5.01226664e-16
 3.61039323e-17   2.97152710e-16  -2.78206277e-16   6.19946802e-16
 1.55122226e-16  -3.50468352e-16   1.90928003e-16  -4.67291136e-16
-5.83354979e-16  -9.06393016e-17  -4.06575815e-16  -3.49248625e-17]
```

Bemerkung zu den unterschiedlichen Formen der beiden Ausgaben: Bei der “naiven” Variante wird ein (100,1)-Array zurückgegeben, bei der LR-Variante ein (100,)-Array