NOTES_SistemPersLinear

March 3, 2019

1 TF2202 Teknik Komputasi - Sistem Persamaan Linear

Fadjar Fathurrahman

```
In [1]: import numpy as np
```

Pada catatan ini kita kan fokus pada metode untuk mencari solusi dari sistem persamaan linear yang dapat dituliskan dalam bentuk matriks sebagai berikut:

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

di mana A dan b masing-masing diberikan dan tugas kita adalah mencari $\mathbf{x}$.

Sebelum membahas mengenai metode numerik untuk menyelesaikan sistem persamaan linear, kita akan mulai dengan pembahasan mengenai operasi matriks dan vektor dengan dalam Numpy.

1.1 Matrix vs ndarray

`ndarray` adalah tipe array yang paling penting pada Numpy. Untuk merepresentasikan matriks kita dapat menggunakan `ndarray` dengan menggunakan `ndarray` dengan dua dimensi. Untuk merepresentasikan vektor (baris atau kolom) kita dapat menggunakan `ndarray` dengan satu dimensi.

Contoh untuk matriks:

$$A = \begin{bmatrix} 4 & 1 & 2 & 3 \\ 3 & 8 & 1 & 9 \\ 3 & 4 & 10 & 4 \end{bmatrix}$$

Dengan menggunakan `ndarray`, kita dapa mendefinisikan matriks A dengan:

```
In [2]: A = np.array([
            [4,1,2,3],
            [3,8,1,9],
            [3,4,10,4]
        ])
        A
```

```
Out[2]: array([[ 4,  1,  2,  3],
               [ 3,  8,  1,  9],
               [ 3,  4, 10,  4]])
```

```
In [3]: type(A)

Out[3]: numpy.ndarray
```

Properti `shape` dapat digunakan untuk mengetahu ukuran dari `ndarray`. Dalam hal ini kita akan mendapatkan tupel berisi dua integer, yang masing-masing integer merupakan jumlah baris dan kolom dari matriks A

```
In [4]: A.shape

Out[4]: (3, 4)
```

Anda dapat menuliskan sebagai berikut:

```
In [5]: Nrow = A.shape[0]
        Ncol = A.shape[1]
        print("Nrow = %d, Ncol = %d" % (Nrow, Ncol))

Nrow = 3, Ncol = 4


In [6]: Nrow, Ncol = A.shape
        print("Nrow = %d, Ncol = %d" % (Nrow, Ncol))

Nrow = 3, Ncol = 4
```

Untuk merepresentasikan vektor, kita dapat menggunakan `ndarray` 1d, misalnya:

```
In [7]: x = np.array([3,1,6,7])
        x

Out[7]: array([3, 1, 6, 7])

In [8]: type(x)

Out[8]: numpy.ndarray
```

Properti `shape` dapat digunakan seperti pada `ndarray` dua dimensi. Pada kasus ini akan dikembalikan tupel dengan satu bilangan integer.

```
In [9]: x.shape

Out[9]: (4,)
```

Fungsi `len` juga dapat digunakan dalam kasus `ndarray` 1d untuk mengetahui jumlah elemen pada suatu vektor:

```
In [10]: len(x)

Out[10]: 4
```

Fungsi `len` juga dapat diaplikasikan pada `ndarray` dua dimensi, namun fungsi ini akan mengembalikan banyak elemen pada dimensi pertama.

```
In [11]: len(A)

Out[11]: 3
```

Untuk mengetahui jumlah kolom, kita dapat mencari panjang dari `A[0]` (misalnya):

```
In [12]: len(A[0])

Out[12]: 4
```

1.1.1 Operasi perkalian matriks dan vektor

Untuk menghitung operasi perkalian, misalnya $\mathbf{b} = \mathbf{A}\mathbf{x}$. Untuk tipe `numpy.ndarray` kita dapat menggunakan fungsi `np.matmul`:

```
In [13]: b = np.matmul(A,x)
         b

Out[13]: array([ 46,  86, 101])
```

Operator `*` memiliki arti yang berbeda untuk operasi antara dua `ndarray`:

```
In [14]: A

Out[14]: array([[ 4,  1,  2,  3],
                [ 3,  8,  1,  9],
                [ 3,  4, 10,  4]])

In [15]: Ax = A*x
         Ax

Out[15]: array([[12,  1, 12, 21],
                [ 9,  8,  6, 63],
                [ 9,  4, 60, 28]])

In [16]: print(Ax[:,0]/A[:,0])
         print(Ax[:,1]/A[:,1])
         print(Ax[:,2]/A[:,2])

[3. 3. 3.]
[1. 1. 1.]
[6. 6. 6.]


In [17]: B = np.matrix([
             [1, 2],
             [3, 4],
             [5, 6],
             [7, 8]
         ])
         np.matmul(A,B)
```

```
Out[17]: matrix([[ 38,  48],
                 [ 95, 116],
                 [ 93, 114]])
```

Jika ingin menggunakan operator * kita dapat menggunakan konstruktor `np.matrix`.

```
In [18]: AA = np.matrix(A)
         BB = np.matrix(B)

In [19]: AA

Out[19]: matrix([[ 4,  1,  2,  3],
                 [ 3,  8,  1,  9],
                 [ 3,  4, 10,  4]])

In [20]: BB

Out[20]: matrix([[1, 2],
                 [3, 4],
                 [5, 6],
                 [7, 8]])

In [21]: type(AA)

Out[21]: numpy.matrixlib.defmatrix.matrix

In [22]: AA.shape, BB.shape

Out[22]: ((3, 4), (4, 2))

In [23]: AA*BB

Out[23]: matrix([[ 38,  48],
                 [ 95, 116],
                 [ 93, 114]])
```

1.1.2 Perkalian dot (skalar)

Untuk operasi skalar antara dua vektor kita dapat menggunakan fungsi `np.dot`

```
In [24]: y = np.array([2,1,3,6])
         y

Out[24]: array([2, 1, 3, 6])

In [25]: np.dot(y,y)

Out[25]: 50

In [26]: np.dot(x,y)
```

```
Out[26]: 67

In [27]: np.dot(x,x)

Out[27]: 95
```

Metode `dot` juga dapat digunakan untuk operasi dot product:

```
In [28]: x.dot(x)

Out[28]: 95
```

Operasi `dot` juga dapat digunakan untuk melakukan perkalian antara matriks dengan vektor:

```
In [29]: A.dot(x)

Out[29]: array([ 46,  86, 101])

In [30]: np.matmul(A,x)

Out[30]: array([ 46,  86, 101])
```

Untuk tipe `matrix`:

```
In [31]: xx = np.matrix(x).transpose()
         xx

Out[31]: matrix([[3],
                 [1],
                 [6],
                 [7]])

In [32]: xx.transpose().dot(xx)[0,0]

Out[32]: 95

In [33]: AA.dot(xx)

Out[33]: matrix([[ 46],
                 [ 86],
                 [101]])

In [34]: AA*xx

Out[34]: matrix([[ 46],
                 [ 86],
                 [101]])

In [35]: AA.dot(BB)
```

```
Out[35]: matrix([[ 38,  48],
                 [ 95, 116],
                 [ 93, 114]])

In [36]: AA*BB

Out[36]: matrix([[ 38,  48],
                 [ 95, 116],
                 [ 93, 114]])
```

Pada catatan ini, penulis akan menggunakan `matrix`. Jika Anda lebih suka menggunakan `ndarray` langsung, Anda dapat menggunakannya dengan dengan sedikit modifikasi.

Operasi vektorisasi sebisa mungkin akan dihindari.

1.2 Metode Eliminasi Gauss

Dalam metode eliminasi Gauss, matriks **A** dan vektor kolom **b** akan ditransformasi sedemikian rupa sehingga diperoleh matriks dalam bentuk segitiga atas atau segitiga bawah.

Transformasi yang dilakukan adalah sebagai berikut.

$$\begin{aligned} A_{ij} &\leftarrow A_{ij} - \alpha A_{kj} \\ b_i &\leftarrow b_i - \alpha b_k \end{aligned}$$

Setelah matriks **A** direduksi menjadi bentuk segitiga atas, solusi persamaan linear yang dihasilkan dapat dicari dengan menggunakan substitusi mundur (*backward substitution*).

$$x_k = \left(b_k - \sum_{j=k+1}^{N} A_{kj} x_j \right) \frac{1}{A_{kk}} \quad , k = N-1, N-2, \ldots, 1$$

1.2.1 Contoh penggunaan metode eliminasi Gauss

Perhatikan sistem persamaan linear berikut ini:

$$\begin{aligned} x_1 + x_2 + x_3 &= 4 \\ 2x_1 + 3x_2 + x_3 &= 9 \\ x_1 - x_2 - x_3 &= -2 \end{aligned}$$

Persamaan di atas dapat diubah dalam bentuk matrix sebagai

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \\ -2 \end{bmatrix}$$

```
In [37]: A = np.matrix([
             [1, 1, 1],
             [2, 3, 1],
             [1, -1, -1]
         ])
         A
```

```
Out[37]: matrix([[ 1,  1,  1],
                 [ 2,  3,  1],
                 [ 1, -1, -1]])

In [38]: b = np.matrix([4, 9, -2]).transpose()
         b

Out[38]: matrix([[ 4],
                 [ 9],
                 [-2]])
```

Karena kita akan memodifikasi matrix `A` dan `b`, maka kita harus membuat backup (copy) dari nilai asli mereka.

```
In [39]: A_orig = np.matrix.copy(A)
         b_orig = np.matrix.copy(b)
```

Dimulai dengan menggunakan elemen matriks A_{11} (atau `A[0,0]` dalam notasi Numpy) sebagai pivot, kita akan melakukan reduksi pada baris kedua dan ketiga.

Kita akan mulai dengan reduksi baris kedua.

```
In [40]: alpha = A[1,0]/A[0,0]
         A[1,:] = A[1,:] - alpha*A[0,:]
         b[1] = b[1] - alpha*b[0]

         print("A = \n", A)
         print("b = \n", b)

A =
 [[ 1  1  1]
 [ 0  1 -1]
 [ 1 -1 -1]]
b =
 [[ 4]
 [ 1]
 [-2]]
```

Masih menggunakan `A[0,0]` sebagai pivot, kita akan reduksi baris ketiga:

```
In [41]: alpha = A[2,0]/A[0,0]
         A[2,:] = A[2,:] - alpha*A[0,:]
         b[2] = b[2] - alpha*b[0]

         print("A = \n", A)
         print("b = \n", b)

A =
 [[ 1  1  1]
```

```
 [ 0  1 -1]
 [ 0 -2 -2]]
b =
 [[ 4]
 [ 1]
 [-6]]
```

Setelah menjadikan `A[0,0]` sebagai pivot dan mereduksi baris kedua dan ketiga, kita akan menggunakan `A[1,1]` sebagai pivot dan mereduksi baris ketiga:

```
In [42]: alpha = A[2,1]/A[1,1]
         A[2,:] = A[2,:] - alpha*A[1,:]
         b[2] = b[2] - alpha*b[1]

         print("A = \n", A)
         print("b = \n", b)
```

```
A =
 [[ 1  1  1]
 [ 0  1 -1]
 [ 0  0 -4]]
b =
 [[ 4]
 [ 1]
 [-4]]
```

Sekarang `A` telah tereduksi menjadi bentuk matriks segitiga atas. Persamaan yang kita miliki sekarang adalah:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -3 \\ 0 & 0 & -8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -4 \end{bmatrix}$$

Sistem persamaan linear ini dapat dengan mudah diselesaikan dengan menggunakan substitusi balik: mulai dari mencari x_3, kemudian x_2, dan akhirnya x_1.

```
In [43]: N = len(b)
         x = np.matrix(np.zeros((N,1)))

         x[N-1] = b[N-1]/A[N-1,N-1]
         for k in range(N-2,-1,-1):
             ss = 0.0
             for j in range(k+1,N):
                 ss = ss + A[k,j]*x[j]
             x[k] = (b[k] - ss)/A[k,k]

         for i in range(N):
             print(x[i])
```

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

Sekarang kita dapat mengecek apakah solusi yang kita dapatkan sudah benar.

```
In [44]: # Gunakan matrix dan vektor original
         # Hasil seharunya berupa vektor kolom dengan elemen-element 0
         A_orig*x - b_orig
```

```
Out[44]: matrix([[0.],
                 [0.],
                 [0.]])
```

1.2.2 Kode Python untuk eliminasi Gauss

Berikut ini adalah implementasi metode eliminasi Gauss pada Python.

Kode ini menerima masukan matriks `A_` dan vektor kolom `b_`. dan mengembalikan solusi `x` dari sistem persamaan linear `A_*x = b_`. Tanda `_` digunakan untuk menunjukkan input asli. Pada kode berikut kita bekerja dengan matriks `A` dan vektor `x` yang merupakan kopi dari input-input awal. Kode ini terbatas pada vektor input `b_` yang hanya terdiri dari satu kolom. Kode dapat dikembangkan untuk kasus kolom lebih dari satu.

```
In [45]: def gauss_elim(A_, b_):

             N, Nrhs = b_.shape

             assert Nrhs == 1

             A = np.matrix.copy(A_)
             b = np.matrix.copy(b_)

             # Eliminasi maju
             for k in range(0,N-1):
                 for i in range(k+1,N):
                     if A[i,k] != 0.0:
                         alpha = A[i,k]/A[k,k]
                         A[i,:] = A[i,:] - alpha*A[k,:]
                         b[i] = b[i] - alpha*b[k]

             # Alokasi mememori untuk solusi
             x = np.matrix(np.zeros((N,1)))

             # Substitusi balik
             x[N-1] = b[N-1]/A[N-1,N-1]
             for k in range(N-2,-1,-1):
                 ss = 0.0
                 for j in range(k+1,N):
```

```
                ss = ss + A[k,j]*x[j]
            x[k] = (b[k] - ss)/A[k,k]

        return x
```

```
In [46]: gauss_elim(A_orig, b_orig)
```

```
Out[46]: matrix([[1.],
                 [2.],
                 [1.]])
```

1.3 Dekomposisi LU

Pada dekomposisi LU, matriks persegi **A** dapat dinyatakan sebagai hasil kali dari matriks segitiga bawah **L** dan matriks segitiga atas **U**:

$$\mathbf{A} = \mathbf{LU}$$

Proses untuk mendapatkan matriks **L** dan **U** dari matriks **A** disebut dengan dekomposisi LU atau faktorisasi LU. Dekomposisi LU tidak unik, artinya bisa terdapat banyak kombinasi **L** dan **U** yang mungkin untuk suatu matriks **A** yang diberikan. Untuk mendapatkan pasangan **L** dan **U** yang unik, kita perlu memberikan batasan atau konstrain terhadap proses dekomposisi LU. Terdapat beberapa varian dekomposisi LU:

Nama	Konstrain
Dekomposisi Doolittle	$L_{ii} = 1$
Dekomposisi Crout	$U_{ii} = 1$
Dekomposisi Cholesky	$\mathbf{L} = \mathbf{U}^T$ (untuk matriks **A** simetrik dan definit positif

Dengan dekomposisi LU kita dapat menuliskan sistem persamaan linear

$$\mathbf{Ax} = \mathbf{b}$$

menjadi:

$$\mathbf{LUx} = \mathbf{b}$$

Dengan menggunakan $\mathbf{y} = \mathbf{Ux}$ kita dapat menuliskan:

$$\mathbf{Ly} = \mathbf{b}$$

Persamaan ini dapat diselesaikan dengan menggunakan substitusi maju. Solusi **x** dapat dicari dengan menggunakan substitusi balik (mundur).

Varian Doolittle untuk dekomposisi LU memiliki bentuk berikut untuk matriks **L** dan **U**, misalnya untuk ukuran 3×3

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Dengan melakukan perkalian $\mathbf{A} = \mathbf{LU}$

$$\mathbf{A} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ U_{11}L_{21} & U_{12}L_{21} + U_{22} & U_{13}L_{21} + U_{23} \\ U_{11}L_{31} & U_{12}L_{31} + U_{22}L_{32} & U_{13}L_{31} + U_{23}L_{32} + U_{33} \end{bmatrix}$$

Dapat ditunjukkan bahwa matriks **U** adalah matriks segitiga atas hasil dari eliminasi Gauss pada matriks **A**. Elemen *off-diagonal* dari matriks **L** adalah pengali α, atau L_{ij} adalah pengali yang mengelimisasi A_{ij}.

1.3.1 Kode Python untuk dekomposisi LU (varian Doolittle)

```
In [47]: def LU_decomp(A_):

             Nrow, Ncol = A_.shape

             assert Nrow == Ncol

             A = np.matrix.copy(A_)

             # Eliminasi Gauss (maju)
             for k in range(0,N-1):
                 for i in range(k+1,N):
                     if A[i,k] != 0.0:
                         alpha = A[i,k]/A[k,k]
                         A[i,k+1:N] = A[i,k+1:N] - alpha*A[k,k+1:N]
                         A[i,k] = alpha

             L = np.matrix( np.tril(A,-1) )
             for i in range(N):
                 L[i,i] = 1.0 # konstrain Doolittle
             U = np.matrix( np.triu(A) )

             return L, U # kembalikan matriks L dan U
```

Definisikan lagi matriks dan vektor yang ada pada contoh sebelumnya.

```
In [48]: A = np.matrix([
             [1, 1, 1],
             [2, 3, 1],
             [1, -1, -1]
         ])
         b = np.matrix([4, 9, -2]).transpose()
```

Lakukan dekomposisi LU:

```
In [49]: L, U = LU_decomp(A)
         print("L = \n", L)
         print("U = \n", U)
```

```
L =
 [[ 1  0  0]
 [ 2  1  0]
 [ 1 -2  1]]
U =
```

```
[[ 1  1  1]
 [ 0  1 -1]
 [ 0  0 -4]]
```

Periksa bahwa $\mathbf{A} = \mathbf{LU}$:

```
In [50]: L*U - A
```

```
Out[50]: matrix([[0, 0, 0],
                 [0, 0, 0],
                 [0, 0, 0]])
```

1.3.2 Implementasi solusi dengan matrix L dan U yang sudah dihitung

Substitusi maju

$$y_k = b_k - \sum_{j=1}^{k-1} L_{kj} y_j, \quad k = 2, 3, \ldots, N$$

```
In [51]: def LU_solve(L, U, b):

             N = L.shape[0]

             x = np.matrix(np.zeros((N,))).transpose()
             y = np.matrix(np.zeros((N,))).transpose()

             # Ly = b
             # Substitusi maju
             y[0] = b[0]/L[0,0]
             for k in range(1,N):
                 ss = 0.0
                 for j in range(k):
                     ss = ss + L[k,j]*y[j]
                 y[k] = (b[k] - ss)/L[k,k]

             # Ux = y
             # Substitusi balik
             x[N-1] = y[N-1]/U[N-1,N-1]
             for k in range(N-2,-1,-1):
                 ss = 0.0
                 for j in range(k+1,N):
                     ss = ss + U[k,j]*x[j]
                 x[k] = (y[k] - ss)/U[k,k]

             return x
```

```
In [52]: x = LU_solve(L, U, b)
```

Cek hasil:

```
In [53]: A*x - b

Out[53]: matrix([[0.],
                 [0.],
                 [0.]])
```

1.4 Latihan 1

Selasaikan persamaan:

$$\begin{bmatrix} 6 & -4 & 1 \\ 4 & 6 & -4 \\ 1 & -4 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -14 \\ 36 \\ 6 \end{bmatrix}$$

1.4.1 Latihan 1 (Solusi)

Menggunakan eliminasi Gauss:

```
In [54]: A = np.matrix([
             [6, -4, 1],
             [-4, 6, -4],
             [1, -4, 6]], dtype=np.float64)
         b = np.matrix([-14, 36, 6], dtype=np.float64).transpose()
         x = gauss_elim(A, b)
         print("Solusi x=\n", x)
         print("Cek solusi: Ax - b\n", A*x - b)

Solusi x=
 [[10.]
 [22.]
 [14.]]
Cek solusi: Ax - b
 [[1.77635684e-15]
 [2.13162821e-14]
 [0.00000000e+00]]
```

Menggunakan dekomposisi LU:

```
In [55]: A = np.matrix([
             [6, -4, 1],
             [-4, 6, -4],
             [1, -4, 6]], dtype=np.float64)
         b = np.matrix([-14, 36, 6], dtype=np.float64).transpose()
         L, U = LU_decomp(A)
         x = LU_solve(L, U, b)
         print("Solusi x=\n", x)
         print("Cek solusi: Ax - b\n", A*x - b)
```

```
Solusi x=
 [[10.]
 [22.]
 [14.]]
Cek solusi: Ax - b
 [[1.77635684e-15]
 [2.13162821e-14]
 [0.00000000e+00]]
```

1.5 Latihan 2

Selasaikan persamaan:

$$\begin{bmatrix} 3 & 1 & -1 \\ 5 & 8 & 2 \\ 3 & 1 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ 5 \end{bmatrix}$$

Menggunakan eliminasi Gauss:

```
In [56]: A = np.matrix( [
             [3, 1, -1],
             [5, 8, 2],
             [3, 1, 10]
         ], dtype=np.float64)
         b = np.matrix([1, -4, 5], np.float64).transpose()
         x = gauss_elim(A, b)
         print("Solusi x=\n", x)
         print("Cek solusi: Ax - b\n", A*x - b)
```

```
Solusi x=
 [[ 0.82296651]
 [-1.10526316]
 [ 0.36363636]]
Cek solusi: Ax - b
 [[-1.11022302e-16]
 [-8.88178420e-16]
 [ 0.00000000e+00]]
```

Menggunakan dekomposisi LU:

```
In [57]: A = np.matrix( [
             [3, 1, -1],
             [5, 8, 2],
             [3, 1, 10]
         ], dtype=np.float64)
         b = np.matrix([1, -4, 5], np.float64).transpose()
         L, U = LU_decomp(A)
         x = LU_solve(L, U, b)
```

```
        print("Solusi x=\n", x)
        print("Cek solusi: Ax - b\n", A*x - b)
```

```
Solusi x=
 [[ 0.82296651]
 [-1.10526316]
 [ 0.36363636]]
Cek solusi: Ax - b
 [[-1.11022302e-16]
 [-8.88178420e-16]
 [ 0.00000000e+00]]
```

1.6 Pivoting

Metode eliminasi Gauss dan LU memiliki beberapa kekurangan. Salah satu yang sering ditemui adalah ketika elemen pivot yang ditemukan adalah 0. Misalkan pada persamaan berikut ini:

$$\begin{bmatrix} 0 & -3 & 7 \\ 1 & 2 & -1 \\ 5 & -2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

Jika kita langsung menggunakan fungsi `gauss_elim` kita akan mendapatkan pesan peringatan sebagai berikut:

```
A = np.matrix( [
    [0, -3, 7],
    [1, 2, -1],
    [5, -2, 0]
], dtype=np.float64)
b = np.matrix([2, 3, 4], np.float64).transpose()
x = gauss_elim(A, b)
```

```
/home/efefer/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:14: RuntimeWarning: d

/home/efefer/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:15: RuntimeWarning: i
  from ipykernel import kernelapp as app
/home/efefer/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:14: RuntimeWarning: i
```

Solusi yang dapat digunakan adalah dengan cara menukar baris atau pivoting sedemikian rupa sehingga elemen pivot yang diperoleh tidak menjadi nol. Dalam kasus ini, kita dapat menukar baris pertama dengan baris ketiga:

$$\begin{bmatrix} 5 & -2 & 0 \\ 1 & 2 & -1 \\ 0 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_3 \\ x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}$$

```
In [58]: A = np.matrix( [
             [5, -2, 0],
             [1, 2, -1],
```

```
        [0, -3, 7]
    ], dtype=np.float64)
    b = np.matrix([4, 3, 2], np.float64).transpose()
    x = gauss_elim(A, b)
    print("Solusi x=\n", x)
    print("Cek solusi: Ax - b\n", A*x - b)
```

```
Solusi x=
 [[1.30434783]
 [1.26086957]
 [0.82608696]]
Cek solusi: Ax - b
 [[ 0.0000000e+00]
 [ 4.4408921e-16]
 [-8.8817842e-16]]
```

Dapat dilihat bahwa pivoting pada dasarnya bertujuan untuk memperbaik sifat dominan diagonal dari matriks. Suatu matriks dikatakan dominan diagonal apabila nilai absolut dari elemen diagoanal matriks memiliki nilai yang terbesar bila dibandingkan dengan nilai-nilai elemen lainnya dalam satu baris.

Ada beberapa strategi yang dapat digunakan untuk pivoting, salah satu yang paling sederhana adalah dengan menggunakan pivoting terskala. Dengan metode ini, pertama kali kita mencari array **s** dengan elemen sebagai berikut:

$$s_i = \max \left| A_{ij} \right|, \quad i = 1, 2, \ldots, N$$

s_i akan disebut sebagai faktor skala dari baris ke-i yang merupakan elemen dengan nilai absolut terbesar dari baris ke-i. Ukuran relatif dari elemen A_{ij} relatif terhadap elemen dengan nilai terbesar adalah:

$$r_{ij} = \frac{|A_{ij}|}{s_i}$$

Elemen pivot dari matriks **A** akan ditentukan berdasarkan r_{ij}. Elemen A_{kk} tidak secara otomatis menjadi elemen pivot, namun kita akan mencari elemen lain di bawah A_{kk} pada kolom ke-k untuk elemen pivot yang terbaik. Misalkan elemen terbaik ini ada pada baris ke-p, yaitu A_{pk} yang memiliki ukuran relatif terbesar, yakni:

$$r_{pk} = \max_j r_{jk}, \quad j \geq k$$

Jika elemen tersebut ditemukan maka kita melakukan pertukaran baris antara baris ke-k dan ke-p.

1.6.1 Kode Python untuk eliminasi Gauss dengan pivoting

```
In [59]: def tukar_baris(v, i, j):
             if len(v.shape) == 1: # array satu dimensi atau vektor kolom
                 v[i], v[j] = v[j], v[i]
             else:
                 v[[i,j],:] = v[[j,i],:]
```

```
In [60]: def gauss_elim_pivot(A_, b_):
             N, Nrhs = b_.shape

             assert Nrhs == 1

             A = np.matrix.copy(A_)
             b = np.matrix.copy(b_)

             # Faktor skala
             s = np.matrix(np.zeros((N,1)))
             for i in range(N):
                 s[i] = np.max(np.abs(A[i,:]))

             SMALL = np.finfo(np.float64).eps

             # Eliminasi maju
             for k in range(0,N-1):

                 r = np.abs(A[k:N,k])/s[k:N]
                 p = np.argmax(r) + k
                 if abs(A[p,k]) < SMALL:
                     raise RuntimeError("Matriks A singular")
                 # Tukar baris jika diperlukan
                 if p != k:
                     print("INFO: tukar baris %d dengan %d" % (p,k))
                     tukar_baris(b, k, p)
                     tukar_baris(s, k, p)
                     tukar_baris(A, k, p)

                 for i in range(k+1,N):
                     if A[i,k] != 0.0:
                         alpha = A[i,k]/A[k,k]
                         A[i,:] = A[i,:] - alpha*A[k,:]
                         b[i] = b[i] - alpha*b[k]

             # Alokasi mememori untuk solusi
             x = np.matrix(np.zeros((N,1)))

             # Substitusi balik
             x[N-1] = b[N-1]/A[N-1,N-1]
             for k in range(N-2,-1,-1):
                 ss = 0.0
                 for j in range(k+1,N):
                     ss = ss + A[k,j]*x[j]
                 x[k] = (b[k] - ss)/A[k,k]

             return x
```

Contoh penggunaan:

```
In [61]: A = np.matrix( [
             [0, -3, 7],
             [1, 2, -1],
             [5, -2, 0]
         ], dtype=np.float64)
         b = np.matrix([2, 3, 4], np.float64).transpose()

         x = gauss_elim_pivot(A, b)
         print("Solusi x=\n", x)
         print("Cek solusi: Ax - b\n", A*x - b)

INFO: tukar baris 2 dengan 0
Solusi x=
 [[1.30434783]
 [1.26086957]
 [0.82608696]]
Cek solusi: Ax - b
 [[-8.8817842e-16]
 [ 4.4408921e-16]
 [ 0.0000000e+00]]
```

1.6.2 Kode Python untuk dekomposisi LU dengan pivoting

```
In [62]: def LU_decomp_pivot(A_):

             Nrow, Ncol = A_.shape

             assert Nrow == Ncol

             N = Nrow

             A = np.matrix.copy(A_)

             # Faktor skala
             s = np.matrix(np.zeros((N,1)))
             for i in range(N):
                 s[i] = np.max(np.abs(A[i,:]))

             iperm = np.arange(N)

             SMALL = np.finfo(np.float64).eps

             # Eliminasi Gauss (maju)
             for k in range(0,N-1):
```

```
            r = np.abs(A[k:N,k])/s[k:N]
            p = np.argmax(r) + k
            if abs(A[p,k]) < SMALL:
                raise RuntimeError("Matriks A singular")
            # Tukar baris jika diperlukan
            if p != k:
                print("INFO: tukar baris %d dengan %d" % (p,k))
                tukar_baris(A, k, p)
                tukar_baris(s, k, p)
                tukar_baris(iperm, k, p)

            for i in range(k+1,N):
                if A[i,k] != 0.0:
                    alpha = A[i,k]/A[k,k]
                    A[i,k+1:N] = A[i,k+1:N] - alpha*A[k,k+1:N]
                    A[i,k] = alpha

        L = np.matrix( np.tril(A,-1) )
        for i in range(N):
            L[i,i] = 1.0 # konstrain Doolittle
        U = np.matrix( np.triu(A) )

        return L, U, iperm # kembalikan matriks L dan U serta vektor permutasi
```

In [63]:
```
def LU_solve_pivot(L, U, iperm, b_):

    N = L.shape[0]

    x = np.matrix(np.zeros((N,))).transpose()
    y = np.matrix(np.zeros((N,))).transpose()

    b = np.matrix.copy(b_)
    for i in range(N):
        b[i] = b_[iperm[i]]

    # Ly = b
    # Substitusi maju
    y[0] = b[0]/L[0,0]
    for k in range(1,N):
        ss = 0.0
        for j in range(k):
            ss = ss + L[k,j]*y[j]
        y[k] = (b[k] - ss)/L[k,k]

    # Ux = y
    # Substitusi balik
    x[N-1] = y[N-1]/U[N-1,N-1]
    for k in range(N-2,-1,-1):
```

```
                ss = 0.0
                for j in range(k+1,N):
                    ss = ss + U[k,j]*x[j]
                x[k] = (y[k] - ss)/U[k,k]

            return x
```

Contoh penggunaan:

```
In [64]: A = np.matrix([
             [2, -2, 6],
             [-2, 4, 3],
             [-1, 8, 4]
         ], dtype=np.float64)
         b = np.matrix([16, 0, -1]).transpose()
         L, U, iperm = LU_decomp_pivot(A)
         print("L = \n", L)
         print("U = \n", U)
         x = LU_solve_pivot(L, U, iperm, b)
         print("Solusi x = \n", x)
         print("Cek solusi A*x - b =\n", A*x - b)

INFO: tukar baris 1 dengan 0
INFO: tukar baris 2 dengan 1
L =
 [[ 1.          0.          0.        ]
 [ 0.5         1.          0.        ]
 [-1.          0.33333333  1.        ]]
U =
 [[-2.          4.          3.        ]
 [ 0.          6.          2.5       ]
 [ 0.          0.          8.16666667]]
Solusi x =
 [[ 1.]
 [-1.]
 [ 2.]]
Cek solusi A*x - b =
 [[0.]
 [0.]
 [0.]]
```

Contoh lain:

```
In [65]: A = np.matrix([
             [0, 2, 5, -1],
             [2, 1, 3, 0],
             [-2, -1, 3, 1],
             [3, 3, -1, 2]
```

```
        ], dtype=np.float64)
        b = np.matrix([-3, 3, -2, 5]).transpose()
        L, U, iperm = LU_decomp_pivot(A)
        print("L = \n", L)
        print("U = \n", U)
        x = LU_solve_pivot(L, U, iperm, b)
        print("Solusi x = \n", x)
        print("Cek solusi A*x - b =\n", A*x - b)
```

```
INFO: tukar baris 3 dengan 0
INFO: tukar baris 3 dengan 1
INFO: tukar baris 3 dengan 2
L =
 [[ 1.          0.          0.          0.        ]
 [ 0.          1.          0.          0.        ]
 [ 0.66666667 -0.5         1.          0.        ]
 [-0.66666667  0.5        -0.02702703  1.        ]]
U =
 [[ 3.          3.         -1.          2.        ]
 [ 0.          2.          5.         -1.        ]
 [ 0.          0.          6.16666667 -1.83333333]
 [ 0.          0.          0.          2.78378378]]
Solusi x =
 [[ 2.00000000e+00]
 [-1.00000000e+00]
 [ 3.60072332e-17]
 [ 1.00000000e+00]]
Cek solusi A*x - b =
 [[0.]
 [0.]
 [0.]
 [0.]]
```

1.7 Menggunakan pustaka pada Python

Untuk berbagai aplikasi pada sains dan teknik, kita biasanya menyelesaikan sistem persamaan linear yang ditemui dengan menggunakan berbagai macam pustaka yang sudah tersedia.

Python sudah memiliki beberapa fungsi yang terkait dengan sistem persamaan linear dan operasi terkait seperti menghitung determinan dan invers matriks.

Fungsi `np.linalg.solve` dapat digunakan untuk menyelesaikan sistem persamaan linear:

```
In [66]: A = np.matrix([
             [1, 1, 1],
             [2, 3, -1],
             [1, -1, -1]
         ])
         B = np.matrix([4, 9, 2]).transpose()
```

```
In [67]: x = np.linalg.solve(A,B)
         x

Out[67]: matrix([[3.00000000e+00],
                 [1.00000000e+00],
                 [1.73472348e-17]])

In [68]: A*x - B

Out[68]: matrix([[0.],
                 [0.],
                 [0.]])
```

Fungsi `np.linalg.det` dapat digunakan untuk menghitung determinan dari suatu matriks

```
In [69]: np.linalg.det(A)

Out[69]: -8.000000000000002
```

Fungsi `np.linalg.inv` dapat digunakan untuk menghitung invers dari suatu matriks

```
In [70]: np.linalg.inv(A)

Out[70]: matrix([[ 0.5  ,  0.   ,  0.5  ],
                 [-0.125,  0.25 , -0.375],
                 [ 0.625, -0.25 , -0.125]])
```