Bab 1

Akar Persamaan

1.1 Metode bisection

Metode bisection sering juga disebut metode bagi-dua atau metode setengah-selang.

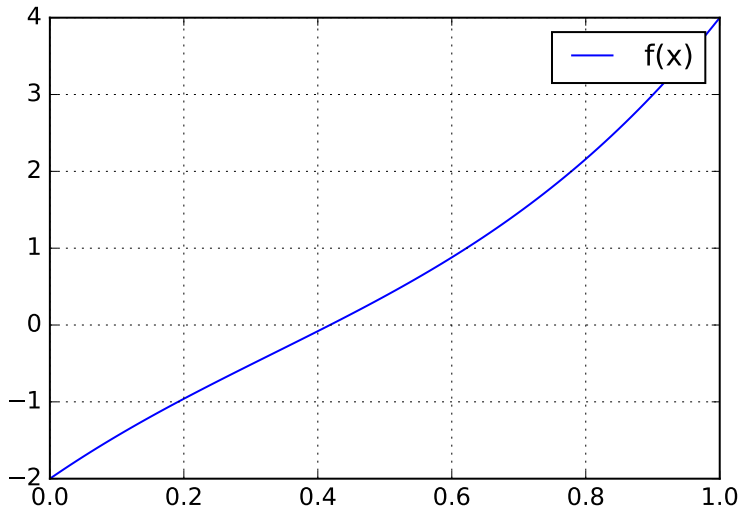
Ide dari metode bisection fakta bahwa jika tanda dari $f(x_1)$ dan $f(x_2)$ berbeda, maka setidaknya ada satu akar dari persamaan $f(x) = 0$ ada dalam selang (x_1, x_2).

Misalkan kita ingin mencari akar dari fungsi berikut:

$$f(x) = 5x^3 - 5x^2 + 6x - 2$$

Untuk memperoleh gambaran mengenai akar dari persamaan $f(x) = 0$, kita akan membuat plot dari dari $f(x)$ terlebih dahulu.

```
def func_01(x):
    return 5*x**3 - 5*x**2 + 6*x - 2

x = np.linspace(0,1,100)
y =  func_01(x)
plt.clf()
plt.plot(x, y, label="f(x)")
plt.legend()
plt.grid()
plt.savefig("figures/bisection_01_1.pdf")
```



Dari plot di atas dapat dilihat bahwa akar dari $f(x)$ terletak di sekitar $x = 0.5$. Sebagai ilustrasi untuk metode bisection kita akan menggunakan interval $x_1 = 0$ dan $x_2 = 1$.

Mari kita cek bahwa tanda $f(x_1)$ dan $f(x_2)$ memiliki tanda yang berbeda, atau $f(x_1)f(x_2) < 0$.

```
x1 = 0.0
x2 = 1.0
f1 = func_01(x1)
f2 = func_01(x2)
print("f1 = ", f1)
print("f2 = ", f2)
print("f1*f2 is negative = ", f1*f2 < 0)
```

```
f1 =  -2.0
f2 =  4.0
f1*f2 is negative =  True
```

Fungsi `np.sign()` juga bisa digunakan untuk mengecek tanda positif atau negatif dari suatu bilangan.

```
np.sign(-2.0), np.sign(2.1)
```

```
(-1.0, 1.0)
```

Sekarang, kita perlu menentukan tebakan akar dari selang x_1 dan x_2. Dengan metode bisection, tebakan akar dihitung tepat berada di tengah selang yaitu:

$$x_r = \frac{x_1 + x_2}{2}$$

```
xr = 0.5*(x1 + x2)
xr
```

```
0.5
```

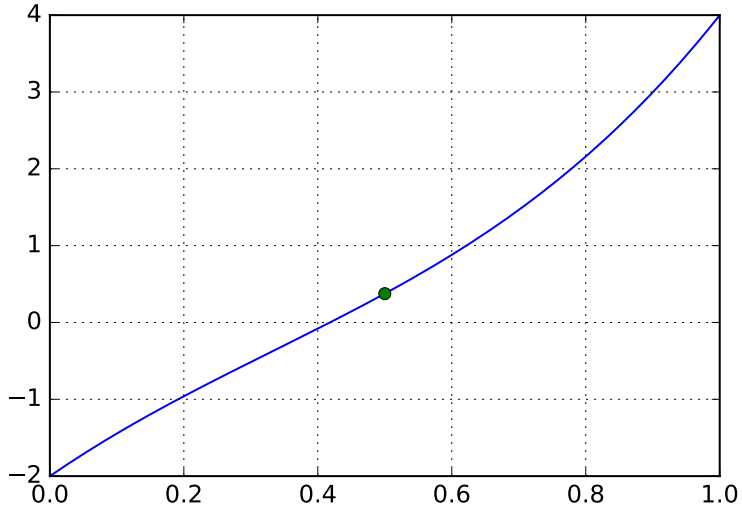
OK, sepertinya kita sudah berada dekat di akar sebenarnya. Mari kita cek nilai $f(x_r)$:

```
fxr = func_01(xr)
fxr
```

```
0.375
```

Ternyata nilai dari $f(x_r)$ tidak tepat pada 0. Mari buat plot $f(x)$ pada domain (interval) yang lebih sempit.

```
x = np.linspace(0.0, 1.0, 500)
y = func_01(x)
plt.clf()
plt.plot(x, y)
plt.plot(xr, fxr, marker="o") # Tandai nilai fx pada xr, yaitu (xr,fxr)
plt.grid()
plt.savefig("figures/bisection_01_2.pdf")
```



Kita dapat memperbaiki tebakan akar dengan memilih rentang baru di mana kita akan mengaplikasikan lagi metode bisection. Kita sekarang memiliki 3 titik yaitu x_1, x_2, dan x_r, dengan nilai fungsi pada titik-titik tersebut adalah:

```
f1, f2, fxr
```

```
(-2.0, 4.0, 0.375)
```

Dengan informasi tersebut, kita dapat menggunakan x_r sebagai pengganti dari x_2 karena selang ini lebih kecil dan diharapkan nilai tebakan akar dapat menjadi lebih dekat dengan akar sebenarnya.

```
x2 = xr
f2 = fxr
```

Cek apakah nilai fungsi pada interval baru ini berbeda tanda (hasil kali $f_(x_1)$ dan $f(x_2)$ adalah negatif.

```
print(f1*f2 < 0)
```

```
True
```

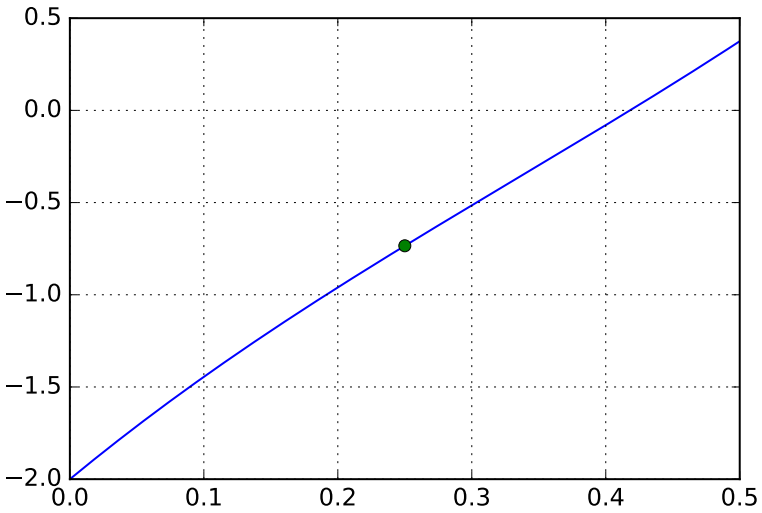
Kita hitung lagi tebakan akar x_r sebagai nilai tengah antara x_1 dan x_2:

```
xr = 0.5*(x1 + x2)
fxr = func_01(xr)
fxr
```

```
-0.734375
```

Sepertinya nilai akar yang kita dapatkan masih belum baik.

```
x = np.linspace(x1, x2, 500)
y = func_01(x)
plt.clf()
plt.plot(x, y)
plt.plot(xr, fxr, marker="o") # Tandai nilai fx pada xr, yaitu (xr,fxr)
plt.grid()
plt.savefig("figures/bisection_01_3.pdf")
```




```
f1, f2, fxr
```

```
(-2.0, 0.375, -0.734375)
```

Kita akan melakukan kembali prosedur bisection. Untuk interval baru kita akan ganti x_1 dengan x_r.

```
x1 = xr
f1 = fxr
print(f1*f2 < 0)
```

```
True
```

Hitung kembali tebakan akar pada selang x_1 dan x_2

```
xr = 0.5*(x1 + x2)
fxr = func_01(xr)
fxr
```

```
-0.189453125
```

Nilai ini sudah lebih dekat dari tebakan-tebakan kita sebelumnya. Untuk mendapatkan tebakan akar yang lebih baik kita akan lakukan sekali lagi metode bisection.

```
f1, f2, fxr
```

```
(-0.734375, 0.375, -0.189453125)
```

```
x1 = xr
f1 = fxr
print(f1*f2 < 0)
```

```
True
```

```
xr = 0.5*(x1 + x2)
fxr = func_01(xr)
fxr
```

```
0.086669921875
```

Tebakan ini lebih baik dari tebakan sebelumnya karena $f(x_r)$ yang diperoleh lebih dekat dengan 0. Kita dapat melakukan prosedur bisection sekali lagi.

```
f1, f2, fxr
```

```
(-0.189453125, 0.375, 0.086669921875)
```

```
x2 = xr
f2 = fxr
xr = 0.5*(x1 + x2)
fxr = func_01(xr)
fxr
```

```
-0.052459716796875
```

Setelah melakukan iterasi metode bisection secara manual, sekarang kita akan membuat prosedur bisection dalam suatu subrutin (fungsi). Fungsi ini menerima masukan `f` sebagai fungsi yang akan dicari akarnya, `x1` dan `x2` sebagai input selang di mana akar akan dicari. Fungsi ini juga menggunakan `TOL` dengan nilai default `1e-10` untuk menentukan akurasi hasil yang diperoleh dan juga `NiterMax` dengan nilai default `100` sebagai jumlah maksimum iterasi yang dilakukan.

```
def bisection(f, x1, x2, TOL=1e-10, NiterMax=100):

    f1 = f(x1)
    f2 = f(x2)

    if f1*f2 > 0:
        raise RuntimeError("f1 dan f2 memiliki tanda yang sama")

    for i in range(1,NiterMax+1):

        xr = 0.5*(x1 + x2)
        fxr = f(xr)

        if abs(fxr) <= TOL:
            print("Iterasi konvergen: akar ditemukan")
            return xr

        print("Iter = %5d, xr = %18.10f, abs(fxr) = %15.5e" % (i, xr, abs(fxr)))

        # f1 dan fxr berbeda tanda
        if f1*fxr < 0.0:
            x2 = xr
            f2 = fxr
        else:
            x1 = xr
            f1 = fxr

    print("WARNING: Konvergensi tidak diperleh setelah %d iterasi" % NiterMax)
    print("WARNING: Nilai tebakan akhir akan dikembalikan")
    return xr
```

Contoh penggunaan:

Cari akar pada selang $[0.0, 1.0]$

```
xr = bisection(func_01, 0.0, 1.0)
```

```
Iter =     1, xr =       0.5000000000, abs(fxr) =     3.75000e-01
Iter =     2, xr =       0.2500000000, abs(fxr) =     7.34375e-01
Iter =     3, xr =       0.3750000000, abs(fxr) =     1.89453e-01
Iter =     4, xr =       0.4375000000, abs(fxr) =     8.66699e-02
Iter =     5, xr =       0.4062500000, abs(fxr) =     5.24597e-02
Iter =     6, xr =       0.4218750000, abs(fxr) =     1.67809e-02
Iter =     7, xr =       0.4140625000, abs(fxr) =     1.79133e-02
Iter =     8, xr =       0.4179687500, abs(fxr) =     5.85616e-04
Iter =     9, xr =       0.4199218750, abs(fxr) =     8.09266e-03
Iter =    10, xr =       0.4189453125, abs(fxr) =     3.75230e-03
Iter =    11, xr =       0.4184570312, abs(fxr) =     1.58304e-03
Iter =    12, xr =       0.4182128906, abs(fxr) =     4.98635e-04
Iter =    13, xr =       0.4180908203, abs(fxr) =     4.35092e-05
Iter =    14, xr =       0.4181518555, abs(fxr) =     2.27558e-04
Iter =    15, xr =       0.4181213379, abs(fxr) =     9.20233e-05
Iter =    16, xr =       0.4181060791, abs(fxr) =     2.42567e-05
Iter =    17, xr =       0.4180984497, abs(fxr) =     9.62632e-06
Iter =    18, xr =       0.4181022644, abs(fxr) =     7.31519e-06
Iter =    19, xr =       0.4181003571, abs(fxr) =     1.15557e-06
Iter =    20, xr =       0.4181013107, abs(fxr) =     3.07981e-06
Iter =    21, xr =       0.4181008339, abs(fxr) =     9.62119e-07
Iter =    22, xr =       0.4181005955, abs(fxr) =     9.67255e-08
Iter =    23, xr =       0.4181007147, abs(fxr) =     4.32697e-07
Iter =    24, xr =       0.4181006551, abs(fxr) =     1.67986e-07
Iter =    25, xr =       0.4181006253, abs(fxr) =     3.56301e-08
Iter =    26, xr =       0.4181006104, abs(fxr) =     3.05477e-08
Iter =    27, xr =       0.4181006178, abs(fxr) =     2.54121e-09
Iter =    28, xr =       0.4181006141, abs(fxr) =     1.40032e-08
```

```
Iter =    29, xr =       0.4181006160, abs(fxr) =     5.73101e-09
Iter =    30, xr =       0.4181006169, abs(fxr) =     1.59490e-09
Iter =    31, xr =       0.4181006174, abs(fxr) =     4.73154e-10
Iter =    32, xr =       0.4181006171, abs(fxr) =     5.60874e-10
Iterasi konvergen: akar ditemukan
```

Cari akar pada selang $[0.0, 1.0]$, dengan toleransi 10^{-9} dan jumlah iterasi maksimum 10.

```
xr = bisection(func_01, 0.0, 0.5, TOL=1e-9, NiterMax=10)
```

```
Iter =     1, xr =       0.2500000000, abs(fxr) =     7.34375e-01
Iter =     2, xr =       0.3750000000, abs(fxr) =     1.89453e-01
Iter =     3, xr =       0.4375000000, abs(fxr) =     8.66699e-02
Iter =     4, xr =       0.4062500000, abs(fxr) =     5.24597e-02
Iter =     5, xr =       0.4218750000, abs(fxr) =     1.67809e-02
Iter =     6, xr =       0.4140625000, abs(fxr) =     1.79133e-02
Iter =     7, xr =       0.4179687500, abs(fxr) =     5.85616e-04
Iter =     8, xr =       0.4199218750, abs(fxr) =     8.09266e-03
Iter =     9, xr =       0.4189453125, abs(fxr) =     3.75230e-03
Iter =    10, xr =       0.4184570312, abs(fxr) =     1.58304e-03
WARNING: Konvergensi tidak diperleh setelah 10 iterasi
WARNING: Nilai tebakan akhir akan dikembalikan
```

Sama seperti sebelumnya namun menggunakan nilai iterasi maksimum default.

```
xr = bisection(func_01, 0.0, 0.5, TOL=1e-9)
```

```
Iter =     1, xr =       0.2500000000, abs(fxr) =     7.34375e-01
Iter =     2, xr =       0.3750000000, abs(fxr) =     1.89453e-01
Iter =     3, xr =       0.4375000000, abs(fxr) =     8.66699e-02
Iter =     4, xr =       0.4062500000, abs(fxr) =     5.24597e-02
Iter =     5, xr =       0.4218750000, abs(fxr) =     1.67809e-02
Iter =     6, xr =       0.4140625000, abs(fxr) =     1.79133e-02
Iter =     7, xr =       0.4179687500, abs(fxr) =     5.85616e-04
Iter =     8, xr =       0.4199218750, abs(fxr) =     8.09266e-03
Iter =     9, xr =       0.4189453125, abs(fxr) =     3.75230e-03
Iter =    10, xr =       0.4184570312, abs(fxr) =     1.58304e-03
Iter =    11, xr =       0.4182128906, abs(fxr) =     4.98635e-04
Iter =    12, xr =       0.4180908203, abs(fxr) =     4.35092e-05
Iter =    13, xr =       0.4181518555, abs(fxr) =     2.27558e-04
Iter =    14, xr =       0.4181213379, abs(fxr) =     9.20233e-05
Iter =    15, xr =       0.4181060791, abs(fxr) =     2.42567e-05
Iter =    16, xr =       0.4180984497, abs(fxr) =     9.62632e-06
Iter =    17, xr =       0.4181022644, abs(fxr) =     7.31519e-06
Iter =    18, xr =       0.4181003571, abs(fxr) =     1.15557e-06
Iter =    19, xr =       0.4181013107, abs(fxr) =     3.07981e-06
Iter =    20, xr =       0.4181008339, abs(fxr) =     9.62119e-07
Iter =    21, xr =       0.4181005955, abs(fxr) =     9.67255e-08
Iter =    22, xr =       0.4181007147, abs(fxr) =     4.32697e-07
Iter =    23, xr =       0.4181006551, abs(fxr) =     1.67986e-07
Iter =    24, xr =       0.4181006253, abs(fxr) =     3.56301e-08
Iter =    25, xr =       0.4181006104, abs(fxr) =     3.05477e-08
Iter =    26, xr =       0.4181006178, abs(fxr) =     2.54121e-09
Iter =    27, xr =       0.4181006141, abs(fxr) =     1.40032e-08
Iter =    28, xr =       0.4181006160, abs(fxr) =     5.73101e-09
Iter =    29, xr =       0.4181006169, abs(fxr) =     1.59490e-09
Iterasi konvergen: akar ditemukan
```

Cari akar pada selang $[0.3, 0.5]$,

```
xr = bisection(func_01, 0.3, 0.5, NiterMax=100)
```

```
Iter =     1, xr =       0.4000000000, abs(fxr) =     8.00000e-02
Iter =     2, xr =       0.4500000000, abs(fxr) =     1.43125e-01
Iter =     3, xr =       0.4250000000, abs(fxr) =     3.07031e-02
Iter =     4, xr =       0.4125000000, abs(fxr) =     2.48340e-02
Iter =     5, xr =       0.4187500000, abs(fxr) =     2.88452e-03
Iter =     6, xr =       0.4156250000, abs(fxr) =     1.09868e-02
Iter =     7, xr =       0.4171875000, abs(fxr) =     4.05420e-03
Iter =     8, xr =       0.4179687500, abs(fxr) =     5.85616e-04
Iter =     9, xr =       0.4183593750, abs(fxr) =     1.14926e-03
Iter =    10, xr =       0.4181640625, abs(fxr) =     2.81773e-04
Iter =    11, xr =       0.4180664063, abs(fxr) =     1.51934e-04
Iter =    12, xr =       0.4181152344, abs(fxr) =     6.49166e-05
Iter =    13, xr =       0.4180908203, abs(fxr) =     4.35092e-05
Iter =    14, xr =       0.4181030273, abs(fxr) =     1.07035e-05
Iter =    15, xr =       0.4180969238, abs(fxr) =     1.64029e-05
Iter =    16, xr =       0.4180999756, abs(fxr) =     2.84972e-06
Iter =    17, xr =       0.4181015015, abs(fxr) =     3.92688e-06
Iter =    18, xr =       0.4181007385, abs(fxr) =     5.38581e-07
Iter =    19, xr =       0.4181003571, abs(fxr) =     1.15557e-06
Iter =    20, xr =       0.4181005478, abs(fxr) =     3.08494e-07
Iter =    21, xr =       0.4181006432, abs(fxr) =     1.15043e-07
Iter =    22, xr =       0.4181005955, abs(fxr) =     9.67255e-08
Iter =    23, xr =       0.4181006193, abs(fxr) =     9.15899e-09
Iter =    24, xr =       0.4181006074, abs(fxr) =     4.37832e-08
Iter =    25, xr =       0.4181006134, abs(fxr) =     1.73121e-08
Iter =    26, xr =       0.4181006163, abs(fxr) =     4.07657e-09
Iter =    27, xr =       0.4181006178, abs(fxr) =     2.54121e-09
Iter =    28, xr =       0.4181006171, abs(fxr) =     7.67679e-10
Iter =    29, xr =       0.4181006175, abs(fxr) =     8.86765e-10
Iterasi konvergen: akar ditemukan
```

Contoh yang akan menghasilkan error:

```
xr = bisection(func_01, 0.3, 0.4)
```

```
---------------------------------------------------------------------------RuntimeError
Traceback (most recent call last)<ipython-input-1-2f68945ac062> in
<module>
----> 1 xr = bisection(func_01, 0.3, 0.4)
<ipython-input-1-3b4f53df89d5> in bisection(f, x1, x2, TOL, NiterMax)
      5
      6     if f1*f2 > 0:
----> 7         raise RuntimeError("f1 dan f2 memiliki tanda yang
sama")
      8
      9     for i in range(1,NiterMax+1):
RuntimeError: f1 dan f2 memiliki tanda yang sama
```

1.1.1 Contoh 2

Contoh dengan fungsi yang berbeda:

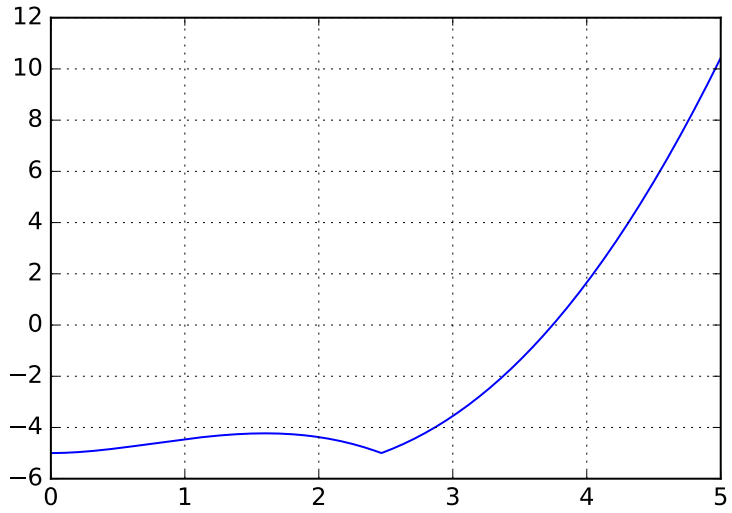
$$f = x^2 \left|\cos(\sqrt{(x)})\right| - 5$$

```
def func_02(x):
    return x**2 * np.abs(np.cos(np.sqrt(x))) - 5
```

Kita perlu selang untuk tebakan awal akar. Untuk memperoleh informasi tersebut kita akan buat plot dari $f(x)$ terlebih dahulu.

```
x = np.linspace(0,5,500)
y = func_02(x)
plt.clf()
plt.plot(x, y)
```

```
plt.grid()
plt.savefig("figures/func_02_1.pdf")
```

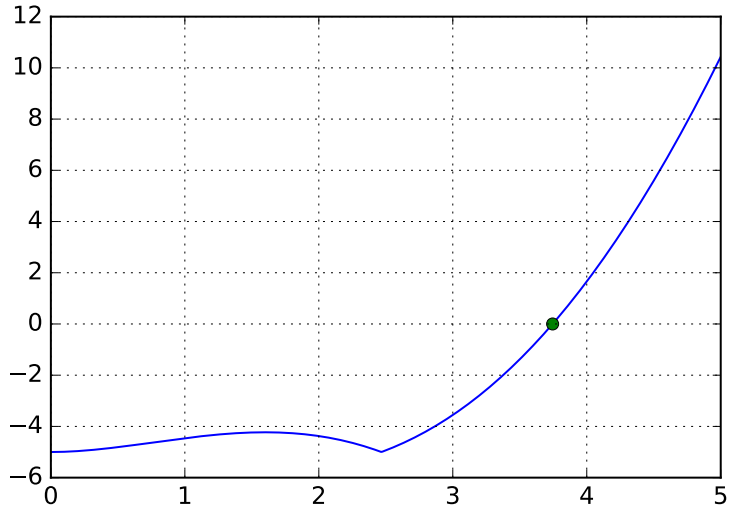


Akar terletak antara $x = 0$ dan $x = 5$.

```
xr = bisection(func_02, 0, 5)
```

```
Iter =      1, xr =       2.5000000000, abs(fxr) =     4.93536e+00
Iter =      2, xr =       3.7500000000, abs(fxr) =     2.87324e-02
Iter =      3, xr =       3.1250000000, abs(fxr) =     3.08887e+00
Iter =      4, xr =       3.4375000000, abs(fxr) =     1.69754e+00
Iter =      5, xr =       3.5937500000, abs(fxr) =     8.77063e-01
Iter =      6, xr =       3.6718750000, abs(fxr) =     4.34912e-01
Iter =      7, xr =       3.7109375000, abs(fxr) =     2.05785e-01
Iter =      8, xr =       3.7304687500, abs(fxr) =     8.92016e-02
Iter =      9, xr =       3.7402343750, abs(fxr) =     3.04035e-02
Iter =     10, xr =       3.7451171875, abs(fxr) =     8.77797e-04
Iter =     11, xr =       3.7475585938, abs(fxr) =     1.39167e-02
Iter =     12, xr =       3.7463378906, abs(fxr) =     6.51683e-03
Iter =     13, xr =       3.7457275391, abs(fxr) =     2.81886e-03
Iter =     14, xr =       3.7454223633, abs(fxr) =     9.70364e-04
Iter =     15, xr =       3.7452697754, abs(fxr) =     4.62426e-05
Iter =     16, xr =       3.7451934814, abs(fxr) =     4.15787e-04
Iter =     17, xr =       3.7452316284, abs(fxr) =     1.84775e-04
Iter =     18, xr =       3.7452507019, abs(fxr) =     6.92668e-05
Iter =     19, xr =       3.7452602386, abs(fxr) =     1.15123e-05
Iter =     20, xr =       3.7452650070, abs(fxr) =     1.73651e-05
Iter =     21, xr =       3.7452626228, abs(fxr) =     2.92641e-06
Iter =     22, xr =       3.7452614307, abs(fxr) =     4.29294e-06
Iter =     23, xr =       3.7452620268, abs(fxr) =     6.83264e-07
Iter =     24, xr =       3.7452623248, abs(fxr) =     1.12157e-06
Iter =     25, xr =       3.7452621758, abs(fxr) =     2.19155e-07
Iter =     26, xr =       3.7452621013, abs(fxr) =     2.32055e-07
Iter =     27, xr =       3.7452621385, abs(fxr) =     6.44991e-09
Iter =     28, xr =       3.7452621572, abs(fxr) =     1.06352e-07
Iter =     29, xr =       3.7452621479, abs(fxr) =     4.99512e-08
Iter =     30, xr =       3.7452621432, abs(fxr) =     2.17507e-08
Iter =     31, xr =       3.7452621409, abs(fxr) =     7.65038e-09
Iter =     32, xr =       3.7452621397, abs(fxr) =     6.00232e-10
Iter =     33, xr =       3.7452621391, abs(fxr) =     2.92484e-09
Iter =     34, xr =       3.7452621394, abs(fxr) =     1.16230e-09
Iter =     35, xr =       3.7452621396, abs(fxr) =     2.81036e-10
Iter =     36, xr =       3.7452621396, abs(fxr) =     1.59599e-10
Iterasi konvergen: akar ditemukan
```

```
x = np.linspace(0,5,500)
y = func_02(x)
plt.clf()
plt.plot(x, y)
plt.plot(xr, func_02(xr), marker="o")
plt.grid()
plt.savefig("figures/func_02_2.pdf")
```



Mari kita coba gunakan selang $x = 3$ dan $x = 4$.

```
xr = bisection(func_02, 3, 4)
```

```
Iter =     1, xr =       3.5000000000, abs(fxr) =     1.37950e+00
Iter =     2, xr =       3.7500000000, abs(fxr) =     2.87324e-02
Iter =     3, xr =       3.6250000000, abs(fxr) =     7.02772e-01
Iter =     4, xr =       3.6875000000, abs(fxr) =     3.43907e-01
Iter =     5, xr =       3.7187500000, abs(fxr) =     1.59314e-01
Iter =     6, xr =       3.7343750000, abs(fxr) =     6.57229e-02
Iter =     7, xr =       3.7421875000, abs(fxr) =     1.86034e-02
Iter =     8, xr =       3.7460937500, abs(fxr) =     5.03748e-03
Iter =     9, xr =       3.7441406250, abs(fxr) =     6.78970e-03
Iter =    10, xr =       3.7451171875, abs(fxr) =     8.77797e-04
Iter =    11, xr =       3.7456054688, abs(fxr) =     2.07942e-03
Iter =    12, xr =       3.7453613281, abs(fxr) =     6.00706e-04
Iter =    13, xr =       3.7452392578, abs(fxr) =     1.38572e-04
Iter =    14, xr =       3.7453002930, abs(fxr) =     2.31060e-04
Iter =    15, xr =       3.7452697754, abs(fxr) =     4.62426e-05
Iter =    16, xr =       3.7452545166, abs(fxr) =     4.61651e-05
Iter =    17, xr =       3.7452621460, abs(fxr) =     3.86710e-08
Iter =    18, xr =       3.7452583313, abs(fxr) =     2.30632e-05
Iter =    19, xr =       3.7452602386, abs(fxr) =     1.15123e-05
Iter =    20, xr =       3.7452611923, abs(fxr) =     5.73681e-06
Iter =    21, xr =       3.7452616692, abs(fxr) =     2.84907e-06
Iter =    22, xr =       3.7452619076, abs(fxr) =     1.40520e-06
Iter =    23, xr =       3.7452620268, abs(fxr) =     6.83264e-07
Iter =    24, xr =       3.7452620864, abs(fxr) =     3.22296e-07
Iter =    25, xr =       3.7452621162, abs(fxr) =     1.41813e-07
Iter =    26, xr =       3.7452621311, abs(fxr) =     5.15708e-08
Iter =    27, xr =       3.7452621385, abs(fxr) =     6.44991e-09
Iter =    28, xr =       3.7452621423, abs(fxr) =     1.61106e-08
Iter =    29, xr =       3.7452621404, abs(fxr) =     4.83032e-09
Iter =    30, xr =       3.7452621395, abs(fxr) =     8.09795e-10
Iter =    31, xr =       3.7452621399, abs(fxr) =     2.01026e-09
Iter =    32, xr =       3.7452621397, abs(fxr) =     6.00232e-10
```

```
Iter =    33, xr =       3.7452621396, abs(fxr) =     1.04781e-10
Iter =    34, xr =       3.7452621397, abs(fxr) =     2.47725e-10
Iterasi konvergen: akar ditemukan
```

1.2 Metode Regula-Falsi

Metode regula-falsi mirip dengan metode bisection, namun dengan persamaan yang berbeda untuk menentukan aproksimasi akar. Pada metode bisection, tebakan akar diberikan sebagai nilai tengah dari x_1 dan x_2 sedangkan pada metode regula-falsi digunakan interpolasi linear antara $f(x_1)$ dan $f(x_2)$. Tebakan akar adalah perpotongan antara garis interpolasi linear ini dengan sumbu x. Hasil akhirnya adalah:

$$x_r = x_2 - \frac{f(x_2)}{f(x_1) - f(x_2)}(x_1 - x_2)$$

```
def regula_falsi(f, x1, x2, TOL=1e-10, NiterMax=100):

    f1 = f(x1)
    f2 = f(x2)

    if f1*f2 > 0:
        raise RuntimeError("f1 dan f2 memiliki tanda yang sama")

    for i in range(1,NiterMax+1):

        xr = x2 - f2*(x1 - x2)/(f1 - f2)
        fxr = f(xr)

        if abs(fxr) <= TOL:
            print("Iterasi konvergen: akar ditemukan")
            return xr

        print("Iter = %5d, xr = %18.10f, abs(fxr) = %15.5e" % (i, xr, abs(fxr)))

        # f1 dan fxr berbeda tanda
        if f1*fxr < 0.0:
            x2 = xr
            f2 = fxr
        else:
            x1 = xr
            f1 = fxr

    print("WARNING: Konvergensi tidak diperleh setelah %d iterasi" % NiterMax)
    print("WARNING: Nilai tebakan akhir akan dikembalikan")
    return xr
```

Contoh penggunaan:

```
xr = regula_falsi(func_01, 0.0, 1.0)
```

```
Iter =     1, xr =       0.3333333333, abs(fxr) =     3.70370e-01
Iter =     2, xr =       0.3898305085, abs(fxr) =     1.24648e-01
Iter =     3, xr =       0.4082699418, abs(fxr) =     4.35410e-02
Iter =     4, xr =       0.4146417183, abs(fxr) =     1.53464e-02
Iter =     5, xr =       0.4168789156, abs(fxr) =     5.42383e-03
Iter =     6, xr =       0.4176685323, abs(fxr) =     1.91870e-03
Iter =     7, xr =       0.4179477285, abs(fxr) =     6.78967e-04
Iter =     8, xr =       0.4180465103, abs(fxr) =     2.40291e-04
Iter =     9, xr =       0.4180814678, abs(fxr) =     8.50444e-05
Iter =    10, xr =       0.4180938398, abs(fxr) =     3.00995e-05
Iter =    11, xr =       0.4180982185, abs(fxr) =     1.06531e-05
Iter =    12, xr =       0.4180997683, abs(fxr) =     3.77043e-06
Iter =    13, xr =       0.4181003168, abs(fxr) =     1.33446e-06
Iter =    14, xr =       0.4181005109, abs(fxr) =     4.72306e-07
```

```
Iter =    15, xr =       0.4181005796, abs(fxr) =     1.67163e-07
Iter =    16, xr =       0.4181006039, abs(fxr) =     5.91639e-08
Iter =    17, xr =       0.4181006125, abs(fxr) =     2.09398e-08
Iter =    18, xr =       0.4181006156, abs(fxr) =     7.41122e-09
Iter =    19, xr =       0.4181006167, abs(fxr) =     2.62305e-09
Iter =    20, xr =       0.4181006170, abs(fxr) =     9.28374e-10
Iter =    21, xr =       0.4181006172, abs(fxr) =     3.28579e-10
Iter =    22, xr =       0.4181006172, abs(fxr) =     1.16294e-10
Iterasi konvergen: akar ditemukan
```

Contoh lain:

```
xr = regula_falsi(func_02, 3, 4)
```

```
Iter =     1, xr =       3.6819027315, abs(fxr) =     3.76606e-01
Iter =     2, xr =       3.7407725061, abs(fxr) =     2.71536e-02
Iter =     3, xr =       3.7449486892, abs(fxr) =     1.89809e-03
Iter =     4, xr =       3.7452402785, abs(fxr) =     1.32391e-04
Iter =     5, xr =       3.7452606151, abs(fxr) =     9.23275e-06
Iter =     6, xr =       3.7452620333, abs(fxr) =     6.43873e-07
Iter =     7, xr =       3.7452621322, abs(fxr) =     4.49024e-08
Iter =     8, xr =       3.7452621391, abs(fxr) =     3.13140e-09
Iter =     9, xr =       3.7452621396, abs(fxr) =     2.18379e-10
Iterasi konvergen: akar ditemukan
```

1.3 Metode fixed-point

Pada metode ini, persamaan $f(x)$ yang ingin kita cari akarnya diubah menjadi $x = g(x)$. Contoh: untuk mencari akar dari persamaan $f(x) = e^{-x} - x = 0$ kita mengubah persamaan tersebut menjadi $x = g(x) = e^{-x}$.

Iterasi dimulai dengan suatu tebakan awal x_0. Nilai tebakan akar berikutnya dihitung dengan persamaan

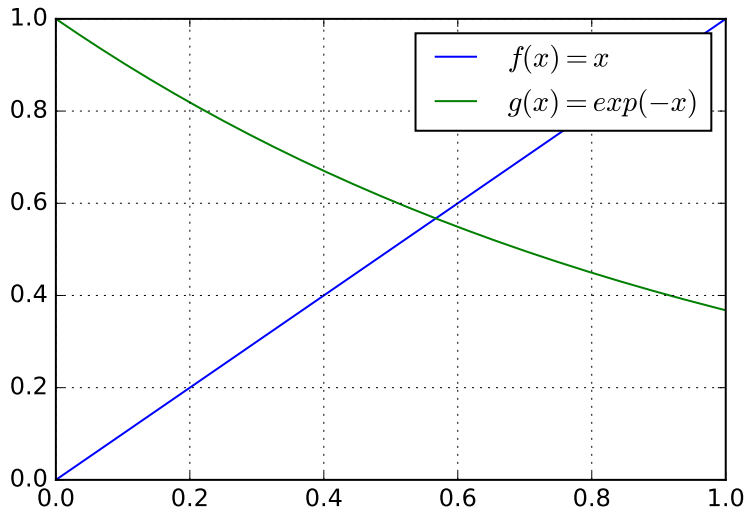
$$x_{i+1} = g(x_i)$$

Jika iterasi ini konvergen, maka x_{i+1} adalah akar dari persamaan $f(x) = 0$.

Perhatikan bahwa metode ini tidak selalu konvergen.

```
# definisi fungsi g(x) = exp(-x)
def func_03(x):
    return np.exp(-x)
```

```
plt.clf()
x = np.linspace(0,1,500)
plt.plot(x, x, label="$f(x) = x$")
plt.plot(x, func_03(x), label="$g(x) = exp(-x)$")
plt.legend()
plt.grid()
plt.savefig("figures/func_03_1.pdf")
```




```
def fixed_point(g, x, TOL=1e-10, NiterMax=100):

    for i in range(1,NiterMax+1):
        gx = g(x)
        print("Iter = %5d, x = %18.10f, g(x) = %18.10f abs(x-g(x)) %15.5e" % (i, x, g(x)
        if abs(x - gx) <= TOL:
            print("Iterasi konvergen: akar ditemukan")
            return x
        x = gx

    print("WARNING: Konvergensi tidak diperleh setelah %d iterasi" % NiterMax)
    print("WARNING: Nilai tebakan akhir akan dikembalikan")
    return x
```

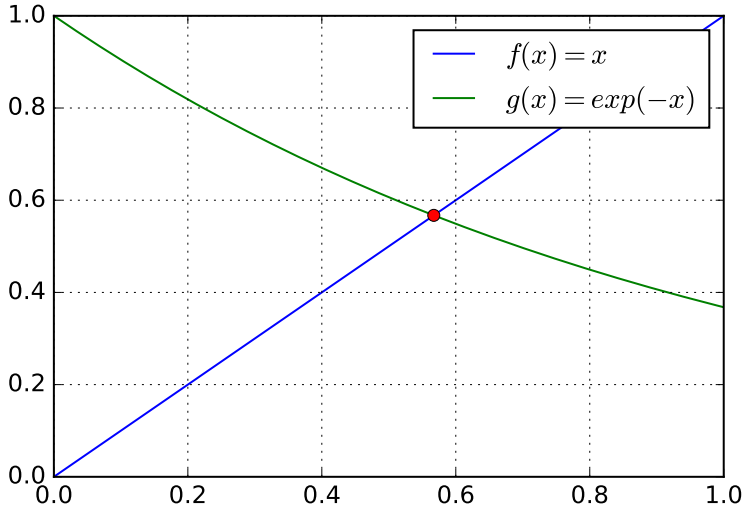
```
x0 = 0.0
xr = fixed_point(func_03, x0)
```

```
Iter =     1, x =       0.0000000000, g(x) =       1.0000000000
abs(x-g(x))     1.00000e+00
Iter =     2, x =       1.0000000000, g(x) =       0.3678794412
abs(x-g(x))     6.32121e-01
Iter =     3, x =       0.3678794412, g(x) =       0.6922006276
abs(x-g(x))     3.24321e-01
Iter =     4, x =       0.6922006276, g(x) =       0.5004735006
abs(x-g(x))     1.91727e-01
Iter =     5, x =       0.5004735006, g(x) =       0.6062435351
abs(x-g(x))     1.05770e-01
Iter =     6, x =       0.6062435351, g(x) =       0.5453957860
abs(x-g(x))     6.08477e-02
Iter =     7, x =       0.5453957860, g(x) =       0.5796123355
abs(x-g(x))     3.42165e-02
Iter =     8, x =       0.5796123355, g(x) =       0.5601154614
abs(x-g(x))     1.94969e-02
Iter =     9, x =       0.5601154614, g(x) =       0.5711431151
abs(x-g(x))     1.10277e-02
Iter =    10, x =       0.5711431151, g(x) =       0.5648793474
abs(x-g(x))     6.26377e-03
Iter =    11, x =       0.5648793474, g(x) =       0.5684287250
abs(x-g(x))     3.54938e-03
Iter =    12, x =       0.5684287250, g(x) =       0.5664147331
abs(x-g(x))     2.01399e-03
Iter =    13, x =       0.5664147331, g(x) =       0.5675566373
abs(x-g(x))     1.14190e-03
Iter =    14, x =       0.5675566373, g(x) =       0.5669089119
```

```
abs(x-g(x))     6.47725e-04
Iter =    15, x =       0.5669089119, g(x) =       0.5672762322
abs(x-g(x))     3.67320e-04
Iter =    16, x =       0.5672762322, g(x) =       0.5670678984
abs(x-g(x))     2.08334e-04
Iter =    17, x =       0.5670678984, g(x) =       0.5671860501
abs(x-g(x))     1.18152e-04
Iter =    18, x =       0.5671860501, g(x) =       0.5671190401
abs(x-g(x))     6.70100e-05
Iter =    19, x =       0.5671190401, g(x) =       0.5671570440
abs(x-g(x))     3.80039e-05
Iter =    20, x =       0.5671570440, g(x) =       0.5671354902
abs(x-g(x))     2.15538e-05
Iter =    21, x =       0.5671354902, g(x) =       0.5671477143
abs(x-g(x))     1.22241e-05
Iter =    22, x =       0.5671477143, g(x) =       0.5671407815
abs(x-g(x))     6.93280e-06
Iter =    23, x =       0.5671407815, g(x) =       0.5671447133
abs(x-g(x))     3.93189e-06
Iter =    24, x =       0.5671447133, g(x) =       0.5671424834
abs(x-g(x))     2.22995e-06
Iter =    25, x =       0.5671424834, g(x) =       0.5671437481
abs(x-g(x))     1.26470e-06
Iter =    26, x =       0.5671437481, g(x) =       0.5671430308
abs(x-g(x))     7.17265e-07
Iter =    27, x =       0.5671430308, g(x) =       0.5671434376
abs(x-g(x))     4.06792e-07
Iter =    28, x =       0.5671434376, g(x) =       0.5671432069
abs(x-g(x))     2.30709e-07
Iter =    29, x =       0.5671432069, g(x) =       0.5671433378
abs(x-g(x))     1.30845e-07
Iter =    30, x =       0.5671433378, g(x) =       0.5671432636
abs(x-g(x))     7.42080e-08
Iter =    31, x =       0.5671432636, g(x) =       0.5671433056
abs(x-g(x))     4.20866e-08
Iter =    32, x =       0.5671433056, g(x) =       0.5671432818
abs(x-g(x))     2.38691e-08
Iter =    33, x =       0.5671432818, g(x) =       0.5671432953
abs(x-g(x))     1.35372e-08
Iter =    34, x =       0.5671432953, g(x) =       0.5671432876
abs(x-g(x))     7.67754e-09
Iter =    35, x =       0.5671432876, g(x) =       0.5671432920
abs(x-g(x))     4.35427e-09
Iter =    36, x =       0.5671432920, g(x) =       0.5671432895
abs(x-g(x))     2.46949e-09
Iter =    37, x =       0.5671432895, g(x) =       0.5671432909
abs(x-g(x))     1.40056e-09
Iter =    38, x =       0.5671432909, g(x) =       0.5671432901
abs(x-g(x))     7.94316e-10
Iter =    39, x =       0.5671432901, g(x) =       0.5671432906
abs(x-g(x))     4.50491e-10
Iter =    40, x =       0.5671432906, g(x) =       0.5671432903
abs(x-g(x))     2.55493e-10
Iter =    41, x =       0.5671432903, g(x) =       0.5671432905
abs(x-g(x))     1.44901e-10
Iter =    42, x =       0.5671432905, g(x) =       0.5671432904
abs(x-g(x))     8.21796e-11
Iterasi konvergen: akar ditemukan
```

```
plt.clf()
x = np.linspace(0,1,500)
plt.plot(x, x, label="$f(x) = x$")
plt.plot(x, func_03(x), label="$g(x) = exp(-x)$")
plt.plot(xr, func_03(xr), marker="o")
```

```
plt.legend()
plt.grid()
plt.savefig("figures/func_03_2.pdf")
```




```
x0 = 1.0
xr = fixed_point(func_03, x0)
```

```
Iter =     1, x =       1.0000000000, g(x) =       0.3678794412
abs(x-g(x))     6.32121e-01
Iter =     2, x =       0.3678794412, g(x) =       0.6922006276
abs(x-g(x))     3.24321e-01
Iter =     3, x =       0.6922006276, g(x) =       0.5004735006
abs(x-g(x))     1.91727e-01
Iter =     4, x =       0.5004735006, g(x) =       0.6062435351
abs(x-g(x))     1.05770e-01
Iter =     5, x =       0.6062435351, g(x) =       0.5453957860
abs(x-g(x))     6.08477e-02
Iter =     6, x =       0.5453957860, g(x) =       0.5796123355
abs(x-g(x))     3.42165e-02
Iter =     7, x =       0.5796123355, g(x) =       0.5601154614
abs(x-g(x))     1.94969e-02
Iter =     8, x =       0.5601154614, g(x) =       0.5711431151
abs(x-g(x))     1.10277e-02
Iter =     9, x =       0.5711431151, g(x) =       0.5648793474
abs(x-g(x))     6.26377e-03
Iter =    10, x =       0.5648793474, g(x) =       0.5684287250
abs(x-g(x))     3.54938e-03
Iter =    11, x =       0.5684287250, g(x) =       0.5664147331
abs(x-g(x))     2.01399e-03
Iter =    12, x =       0.5664147331, g(x) =       0.5675566373
abs(x-g(x))     1.14190e-03
Iter =    13, x =       0.5675566373, g(x) =       0.5669089119
abs(x-g(x))     6.47725e-04
Iter =    14, x =       0.5669089119, g(x) =       0.5672762322
abs(x-g(x))     3.67320e-04
Iter =    15, x =       0.5672762322, g(x) =       0.5670678984
abs(x-g(x))     2.08334e-04
Iter =    16, x =       0.5670678984, g(x) =       0.5671860501
abs(x-g(x))     1.18152e-04
Iter =    17, x =       0.5671860501, g(x) =       0.5671190401
abs(x-g(x))     6.70100e-05
Iter =    18, x =       0.5671190401, g(x) =       0.5671570440
abs(x-g(x))     3.80039e-05
Iter =    19, x =       0.5671570440, g(x) =       0.5671354902
```

```
abs(x-g(x))     2.15538e-05
Iter =    20, x =       0.5671354902, g(x) =       0.5671477143
abs(x-g(x))     1.22241e-05
Iter =    21, x =       0.5671477143, g(x) =       0.5671407815
abs(x-g(x))     6.93280e-06
Iter =    22, x =       0.5671407815, g(x) =       0.5671447133
abs(x-g(x))     3.93189e-06
Iter =    23, x =       0.5671447133, g(x) =       0.5671424834
abs(x-g(x))     2.22995e-06
Iter =    24, x =       0.5671424834, g(x) =       0.5671437481
abs(x-g(x))     1.26470e-06
Iter =    25, x =       0.5671437481, g(x) =       0.5671430308
abs(x-g(x))     7.17265e-07
Iter =    26, x =       0.5671430308, g(x) =       0.5671434376
abs(x-g(x))     4.06792e-07
Iter =    27, x =       0.5671434376, g(x) =       0.5671432069
abs(x-g(x))     2.30709e-07
Iter =    28, x =       0.5671432069, g(x) =       0.5671433378
abs(x-g(x))     1.30845e-07
Iter =    29, x =       0.5671433378, g(x) =       0.5671432636
abs(x-g(x))     7.42080e-08
Iter =    30, x =       0.5671432636, g(x) =       0.5671433056
abs(x-g(x))     4.20866e-08
Iter =    31, x =       0.5671433056, g(x) =       0.5671432818
abs(x-g(x))     2.38691e-08
Iter =    32, x =       0.5671432818, g(x) =       0.5671432953
abs(x-g(x))     1.35372e-08
Iter =    33, x =       0.5671432953, g(x) =       0.5671432876
abs(x-g(x))     7.67754e-09
Iter =    34, x =       0.5671432876, g(x) =       0.5671432920
abs(x-g(x))     4.35427e-09
Iter =    35, x =       0.5671432920, g(x) =       0.5671432895
abs(x-g(x))     2.46949e-09
Iter =    36, x =       0.5671432895, g(x) =       0.5671432909
abs(x-g(x))     1.40056e-09
Iter =    37, x =       0.5671432909, g(x) =       0.5671432901
abs(x-g(x))     7.94316e-10
Iter =    38, x =       0.5671432901, g(x) =       0.5671432906
abs(x-g(x))     4.50491e-10
Iter =    39, x =       0.5671432906, g(x) =       0.5671432903
abs(x-g(x))     2.55493e-10
Iter =    40, x =       0.5671432903, g(x) =       0.5671432905
abs(x-g(x))     1.44901e-10
Iter =    41, x =       0.5671432905, g(x) =       0.5671432904
abs(x-g(x))     8.21796e-11
Iterasi konvergen: akar ditemukan
```

```
x0 = -10.0
xr = fixed_point(func_03, x0)
```

```
Iter =     1, x =     -10.0000000000, g(x) =   22026.4657948067
abs(x-g(x))     2.20365e+04
Iter =     2, x =   22026.4657948067, g(x) =       0.0000000000
abs(x-g(x))     2.20265e+04
Iter =     3, x =       0.0000000000, g(x) =       1.0000000000
abs(x-g(x))     1.00000e+00
Iter =     4, x =       1.0000000000, g(x) =       0.3678794412
abs(x-g(x))     6.32121e-01
Iter =     5, x =       0.3678794412, g(x) =       0.6922006276
abs(x-g(x))     3.24321e-01
Iter =     6, x =       0.6922006276, g(x) =       0.5004735006
abs(x-g(x))     1.91727e-01
Iter =     7, x =       0.5004735006, g(x) =       0.6062435351
abs(x-g(x))     1.05770e-01
```

```
Iter =     8, x =       0.6062435351, g(x) =       0.5453957860
abs(x-g(x))     6.08477e-02
Iter =     9, x =       0.5453957860, g(x) =       0.5796123355
abs(x-g(x))     3.42165e-02
Iter =    10, x =       0.5796123355, g(x) =       0.5601154614
abs(x-g(x))     1.94969e-02
Iter =    11, x =       0.5601154614, g(x) =       0.5711431151
abs(x-g(x))     1.10277e-02
Iter =    12, x =       0.5711431151, g(x) =       0.5648793474
abs(x-g(x))     6.26377e-03
Iter =    13, x =       0.5648793474, g(x) =       0.5684287250
abs(x-g(x))     3.54938e-03
Iter =    14, x =       0.5684287250, g(x) =       0.5664147331
abs(x-g(x))     2.01399e-03
Iter =    15, x =       0.5664147331, g(x) =       0.5675566373
abs(x-g(x))     1.14190e-03
Iter =    16, x =       0.5675566373, g(x) =       0.5669089119
abs(x-g(x))     6.47725e-04
Iter =    17, x =       0.5669089119, g(x) =       0.5672762322
abs(x-g(x))     3.67320e-04
Iter =    18, x =       0.5672762322, g(x) =       0.5670678984
abs(x-g(x))     2.08334e-04
Iter =    19, x =       0.5670678984, g(x) =       0.5671860501
abs(x-g(x))     1.18152e-04
Iter =    20, x =       0.5671860501, g(x) =       0.5671190401
abs(x-g(x))     6.70100e-05
Iter =    21, x =       0.5671190401, g(x) =       0.5671570440
abs(x-g(x))     3.80039e-05
Iter =    22, x =       0.5671570440, g(x) =       0.5671354902
abs(x-g(x))     2.15538e-05
Iter =    23, x =       0.5671354902, g(x) =       0.5671477143
abs(x-g(x))     1.22241e-05
Iter =    24, x =       0.5671477143, g(x) =       0.5671407815
abs(x-g(x))     6.93280e-06
Iter =    25, x =       0.5671407815, g(x) =       0.5671447133
abs(x-g(x))     3.93189e-06
Iter =    26, x =       0.5671447133, g(x) =       0.5671424834
abs(x-g(x))     2.22995e-06
Iter =    27, x =       0.5671424834, g(x) =       0.5671437481
abs(x-g(x))     1.26470e-06
Iter =    28, x =       0.5671437481, g(x) =       0.5671430308
abs(x-g(x))     7.17265e-07
Iter =    29, x =       0.5671430308, g(x) =       0.5671434376
abs(x-g(x))     4.06792e-07
Iter =    30, x =       0.5671434376, g(x) =       0.5671432069
abs(x-g(x))     2.30709e-07
Iter =    31, x =       0.5671432069, g(x) =       0.5671433378
abs(x-g(x))     1.30845e-07
Iter =    32, x =       0.5671433378, g(x) =       0.5671432636
abs(x-g(x))     7.42080e-08
Iter =    33, x =       0.5671432636, g(x) =       0.5671433056
abs(x-g(x))     4.20866e-08
Iter =    34, x =       0.5671433056, g(x) =       0.5671432818
abs(x-g(x))     2.38691e-08
Iter =    35, x =       0.5671432818, g(x) =       0.5671432953
abs(x-g(x))     1.35372e-08
Iter =    36, x =       0.5671432953, g(x) =       0.5671432876
abs(x-g(x))     7.67754e-09
Iter =    37, x =       0.5671432876, g(x) =       0.5671432920
abs(x-g(x))     4.35427e-09
Iter =    38, x =       0.5671432920, g(x) =       0.5671432895
abs(x-g(x))     2.46949e-09
Iter =    39, x =       0.5671432895, g(x) =       0.5671432909
abs(x-g(x))     1.40056e-09
Iter =    40, x =       0.5671432909, g(x) =       0.5671432901
```

```
abs(x-g(x))     7.94316e-10
Iter =    41, x =       0.5671432901, g(x) =       0.5671432906
abs(x-g(x))     4.50491e-10
Iter =    42, x =       0.5671432906, g(x) =       0.5671432903
abs(x-g(x))     2.55493e-10
Iter =    43, x =       0.5671432903, g(x) =       0.5671432905
abs(x-g(x))     1.44901e-10
Iter =    44, x =       0.5671432905, g(x) =       0.5671432904
abs(x-g(x))     8.21796e-11
Iterasi konvergen: akar ditemukan
```

1.4 Metode Newton-Raphson

Metode Newton-Raphson adalah salah satu metode yang paling sering digunakan untuk mencari akar persamaan nonlinear. Metode ini memerlukan informasi tebakan awal akar dan turunan pertama dari fungsi yang akan dicari akarnya. Metode Newton-Rapshon dapat diturunkan dari deret Taylor untuk $f(x)$ disekitar x:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \mathcal{O}(x_{i+1} - x_i)^2$$

Jika x_{i+1} adalah akar dari $f(x) = 0$ maka diperoleh:

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \mathcal{O}(x_{i+1} - x_i)^2$$

Dengan mengasumsikan x_i dekat dengan x_{i+1}, suku $\mathcal{O}(x_{i+1} - x_i)$ dapat dianggap nol sehingga diperoleh:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Sebagai contoh, kita akan menghitung akar dari persamaan $f(x) = x^3 - 35$

```
def func_04(x):
    return x**3 - 35.0

def dfunc_04(x):
    return 3*x**2
```

```
plt.clf()
x = np.linspace(2,4,500)
plt.plot(x, func_04(x))
plt.grid(); plt.savefig("figures/func_04.pdf")
```

```
def newton_raphson(f, df, x, TOL=1e-10, NiterMax=100):

    SMALL = np.finfo(float).eps

    for i in range(1,NiterMax+1):
        fx = f(x)
        dfx = df(x)

        if abs(dfx) <= SMALL:
            raise RuntimeError("Turunan f(x) sangat kecil")

        xr = x - fx/dfx

        print("Iter = %5d, x = %18.10f, abs(f(x)) %15.5e" % (i, x, abs(fx)))
        if abs(fx) <= TOL:
            print("Iterasi konvergen: akar ditemukan")
            return x

        x = xr

    print("WARNING: Konvergensi tidak diperleh setelah %d iterasi" % NiterMax)
```

```
    print("WARNING: Nilai tebakan akhir akan dikembalikan")
    return x
```

Kita akan coba mencari akar persamaan `func_04` dengan beberapa tebakan awal.

```
x0 = 3.0
xr = newton_raphson(func_04, dfunc_04, x0)
```

```
Iter =     1, x =       3.0000000000, abs(f(x))     8.00000e+00
Iter =     2, x =       3.2962962963, abs(f(x))     8.16136e-01
Iter =     3, x =       3.2712589288, abs(f(x))     6.18335e-03
Iter =     4, x =       3.2710663215, abs(f(x))     3.64060e-07
Iter =     5, x =       3.2710663102, abs(f(x))     0.00000e+00
Iterasi konvergen: akar ditemukan
```

```
x0 = 4.0
xr = newton_raphson(func_04, dfunc_04, x0)
```

```
Iter =     1, x =       4.0000000000, abs(f(x))     2.90000e+01
Iter =     2, x =       3.3958333333, abs(f(x))     4.15968e+00
Iter =     3, x =       3.2755942598, abs(f(x))     1.45547e-01
Iter =     4, x =       3.2710725664, abs(f(x))     2.00823e-04
Iter =     5, x =       3.2710663102, abs(f(x))     3.84091e-10
Iter =     6, x =       3.2710663102, abs(f(x))     0.00000e+00
Iterasi konvergen: akar ditemukan
```

```
x0 = 10.0
xr = newton_raphson(func_04, dfunc_04, x0)
```

```
Iter =     1, x =      10.0000000000, abs(f(x))     9.65000e+02
Iter =     2, x =       6.7833333333, abs(f(x))     2.77126e+02
Iter =     3, x =       4.7757703873, abs(f(x))     7.39257e+01
Iter =     4, x =       3.6953637378, abs(f(x))     1.54628e+01
Iter =     5, x =       3.3179190978, abs(f(x))     1.52560e+00
Iter =     6, x =       3.2717248105, abs(f(x))     2.11419e-02
Iter =     7, x =       3.2710664427, abs(f(x))     4.25408e-06
Iter =     8, x =       3.2710663102, abs(f(x))     1.70530e-13
Iterasi konvergen: akar ditemukan
```

Kita coba mencari akar dari `func_01`. Kita perlu mendefinisikan turunan dari `func_01`.

```
def func_01(x):
    return 5*x**3 - 5*x**2 + 6*x - 2

def dfunc_01(x):
    return 15*x**2 - 10*x + 6
```

```
xr = newton_raphson(func_01, dfunc_01, 0.0)
```

```
Iter =     1, x =       0.0000000000, abs(f(x))     2.00000e+00
Iter =     2, x =       0.3333333333, abs(f(x))     3.70370e-01
Iter =     3, x =       0.4188034188, abs(f(x))     3.12185e-03
Iter =     4, x =       0.4181007594, abs(f(x))     6.31253e-07
Iter =     5, x =       0.4181006173, abs(f(x))     2.57572e-14
Iterasi konvergen: akar ditemukan
```

```
xr = newton_raphson(func_01, dfunc_01, 1.0)
```

```
Iter =     1, x =       1.0000000000, abs(f(x))     4.00000e+00
Iter =     2, x =       0.6363636364, abs(f(x))     1.08189e+00
Iter =     3, x =       0.4469148796, abs(f(x))     1.29143e-01
Iter =     4, x =       0.4183866706, abs(f(x))     1.27050e-03
Iter =     5, x =       0.4181006407, abs(f(x))     1.04260e-07
Iter =     6, x =       0.4181006173, abs(f(x))     8.88178e-16
Iterasi konvergen: akar ditemukan
```

```
xr = newton_raphson(func_01, dfunc_01, 10.0)
```

```
Iter =     1, x =      10.0000000000, abs(f(x))     4.55800e+03
Iter =     2, x =       6.7581792319, abs(f(x))     1.35352e+03
Iter =     3, x =       4.5873900492, abs(f(x))     4.02992e+02
Iter =     4, x =       3.1261527539, abs(f(x))     1.20650e+02
Iter =     5, x =       2.1317684971, abs(f(x))     3.65069e+01
Iter =     6, x =       1.4409899718, abs(f(x))     1.12244e+01
Iter =     7, x =       0.9473245383, abs(f(x))     3.44759e+00
Iter =     8, x =       0.6021555613, abs(f(x))     8.91659e-01
Iter =     9, x =       0.4375613295, abs(f(x))     8.69457e-02
Iter =    10, x =       0.4182241073, abs(f(x))     5.48453e-04
Iter =    11, x =       0.4181006216, abs(f(x))     1.94077e-08
Iter =    12, x =       0.4181006173, abs(f(x))     0.00000e+00
Iterasi konvergen: akar ditemukan
```

Sebagai perbandingan dengan metode fixed-point, kita akan menghitung akar dari persamaan $f(x) = e^{-x} - x$. Turunan pertama dari fungsi ini adalah $f'(x) = -e^{-x} - 1$

```
def func_05(x):
    return np.exp(-x) - x

def dfunc_05(x):
    return -np.exp(-x) - 1
```

```
x0 = 0.0
xr = newton_raphson(func_05, dfunc_05, x0)
```

```
Iter =     1, x =       0.0000000000, abs(f(x))     1.00000e+00
Iter =     2, x =       0.5000000000, abs(f(x))     1.06531e-01
Iter =     3, x =       0.5663110032, abs(f(x))     1.30451e-03
Iter =     4, x =       0.5671431650, abs(f(x))     1.96480e-07
Iter =     5, x =       0.5671432904, abs(f(x))     4.44089e-15
Iterasi konvergen: akar ditemukan
```

Dapat diamati bahwa metode Newton-Raphson konvergen dengan cepat dibandingkan dengan metode fixed-point.

1.5 Metode secant

Metode secant menggunakan ide yang sama dengan metode Newton-Raphson. Perbedaannya adalah metode secant menggunakan aproksimasi terhadap turunan pertama dari $f(x)$.

$$f'(x) \approx \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

```
def secant(f, x, TOL=1e-10, NiterMax=100, DELTA=0.001):
    SMALL = np.finfo(float).eps
    # Untuk aproksimasi turunan pertama
    x_old = x + DELTA
    for i in range(1,NiterMax+1):
        fx = f(x)
        fx_old = f(x_old)
        dfx = (fx_old - fx)/(x_old - x)
        if abs(dfx) <= SMALL:
            raise RuntimeError("Turunan f(x) sangat kecil")
        xr = x - fx/dfx
        print("Iter = %5d, x = %18.10f, abs(f(x)) %15.5e" % (i, x, abs(fx)))
        if abs(fx) <= TOL:
            print("Iterasi konvergen: akar ditemukan")
            return x
        x_old = x
        x = xr
    print("WARNING: Konvergensi tidak diperoleh setelah %d iterasi" % NiterMax)
    print("WARNING: Nilai tebakan akhir akan dikembalikan")
    return x
```

```
x0 = 0.0
xr = secant(func_05, x0)
```

```
Iter =     1, x =       0.0000000000, abs(f(x))     1.00000e+00
Iter =     2, x =       0.5001249896, abs(f(x))     1.06330e-01
Iter =     3, x =       0.5596304161, abs(f(x))     1.17898e-02
Iter =     4, x =       0.5670511519, abs(f(x))     1.44397e-04
Iter =     5, x =       0.5671431650, abs(f(x))     1.96523e-07
Iter =     6, x =       0.5671432904, abs(f(x))     3.27660e-12
Iterasi konvergen: akar ditemukan
```

```
x0 = 0.0
xr = secant(func_01, x0)
```

```
Iter =     1, x =       0.0000000000, abs(f(x))     2.00000e+00
Iter =     2, x =       0.3336110645, abs(f(x))     3.69167e-01
Iter =     3, x =       0.4091296137, abs(f(x))     3.97425e-02
Iter =     4, x =       0.4182403537, abs(f(x))     6.20610e-04
Iter =     5, x =       0.4181002700, abs(f(x))     1.54234e-06
Iter =     6, x =       0.4181006172, abs(f(x))     6.17357e-11
Iterasi konvergen: akar ditemukan
```

```
x0 = 1.0
xr = secant(func_04, x0)
```

```
Iter =     1, x =       1.0000000000, abs(f(x))     3.40000e+01
Iter =     2, x =      12.3220075518, abs(f(x))     1.83587e+03
Iter =     3, x =       1.2058686147, abs(f(x))     3.32465e+01
Iter =     4, x =       1.4035942709, abs(f(x))     3.22348e+01
Iter =     5, x =       7.7034593939, abs(f(x))     4.22149e+02
Iter =     6, x =       1.8505185189, abs(f(x))     2.86630e+01
Iter =     7, x =       2.2226542773, abs(f(x))     2.40197e+01
Iter =     8, x =       4.1476654265, abs(f(x))     3.63528e+01
Iter =     9, x =       2.9885349159, abs(f(x))     8.30838e+00
Iter =    10, x =       3.2041693472, abs(f(x))     2.10375e+00
Iter =    11, x =       3.2772827235, abs(f(x))     1.99924e-01
Iter =    12, x =       3.2709375923, abs(f(x))     4.13163e-03
Iter =    13, x =       3.2710660659, abs(f(x))     7.84243e-06
Iter =    14, x =       3.2710663102, abs(f(x))     3.08610e-10
Iter =    15, x =       3.2710663102, abs(f(x))     0.00000e+00
```

```
Iterasi konvergen: akar ditemukan
```

Bab 2

Sistem Persamaan Linear

2.1 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.

$$A_{ij} \leftarrow A_{ij} - \alpha A_{kj}$$
$$b_i \leftarrow b_i - \alpha b_k$$

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

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

Contoh penggunaan metode eliminasi Gauss

Perhatikan sistem persamaan linear berikut ini:

$$x_1 + x_2 + x_3 = 4$$
$$2x_1 + 3x_2 + x_3 = 9$$
$$x_1 - x_2 - x_3 = -2$$

Persamaan di atas dapat diubah dalam bentuk matriks sebagai

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

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

```
matrix([[ 1,  1,  1],
        [ 2,  3,  1],
        [ 1, -1, -1]])
```

```
b = np.matrix([4, 9, -2]).transpose()
b
```

```
matrix([[ 4],
        [ 9],
        [-2]])
```

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

```
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.

```
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:

```
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:

```
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.

```
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.

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

```
matrix([[0.],
        [0.],
        [0.]])
```

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.

```
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
```

Contoh penggunaan:

```
gauss_elim(A_orig, b_orig)
```

```
matrix([[1.],
        [2.],
        [1.]])
```

Selesaikan persamaan:

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

```
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]]
```

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:

```
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]]
```

2.2 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}.

Kode Python untuk dekomposisi LU (varian Doolittle)

```
def LU_decomp(A_):
    Nrow, Ncol = A_.shape
    assert Nrow == Ncol
    N = Nrow
    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.

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

Lakukan dekomposisi LU:

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

```
L*U - A
```

```
matrix([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

Implementasi solusi dengan matrix L dan U yang sudah dihitung

Substitusi maju

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

```
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
```

```
x = LU_solve(L, U, b)
```

Cek hasil:

```
A*x - b
```

```
matrix([[0.],
        [0.],
        [0.]])
```

Contoh:

Menggunakan dekomposisi LU:

```
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]]
```

Menggunakan dekomposisi LU:

```
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]]
```

2.3 Pivoting

Metode eliminasi Gauss dan LU memiliki beberapa kekurangan. Salah satu yang sering 4 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/bin/pweave:11: RuntimeWarning: divide by zero
encountered in double_scalars
/home/efefer/miniconda3/bin/pweave:12: RuntimeWarning: invalid value
encountered in multiply
/home/efefer/miniconda3/bin/pweave:11: RuntimeWarning: invalid value
encountered in double_scalars
```

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:

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

```
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 ilai-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:

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

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.

Kode Python untuk eliminasi Gauss dengan pivoting

```
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],:]

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:

```
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]]
```

Kode Python untuk dekomposisi LU dengan pivoting

```
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
```

```
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:

```
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:

```
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.]]
```

2.4 Pustaka terkait sistem persamaan linear

```
import numpy as np
```

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:

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

```
x = np.linalg.solve(A,B)
x
```

```
matrix([[3.],
        [1.],
        [0.]])
```

```
A*x - B
```

```
matrix([[0.],
        [0.],
        [0.]])
```

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

```
np.linalg.det(A)
```

```
-8.000000000000002
```

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

```
np.linalg.inv(A)
```

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

Bab 3

Interpolasi fungsi

3.1 Interpolasi dengan polinomial Lagrange

Teori polinomial Lagrange

Polinomial Lagrange didefinisikan sebagai:

$$P_n(x) = \sum_{i=0}^{n} y_i L_i(x)$$

Polinomial ini adalah interpolant yang memiliki derajat n dan melewati $(n+1)$ titik data atau pasangan (x_i, y_i) dan $L_i(x)$ adalah fungsi polinomial dengan bentuk:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Implementasi interpolasi Lagrange

```
def lagrange_interp(x, y, xx):

    assert len(x) == len(y)

    # Jumlah data adalah N + 1 dan derajat polynomial adalah N
    # atau:
    # Jumlah data adalah N dan derajat polynomial adalah N - 1
    N = len(x) - 1

    yy = 0.0
    for i in range(N+1):
        # Evaluasi fungsi kardinal
        Li = 1.0 # inisialisasi ke ke 1.0
        for j in range(N+1):
            if i != j:
                Li = Li * (xx - x[j])/(x[i] - x[j])
        yy = yy + y[i]*Li
    return yy
```

Contoh

Sebagai contoh, diberikan data sebagai berikut:

x_i	y_i
1	0
4	1.386294
6	1.791760

```
x = np.array([1.0, 4.0, 6.0])
y = np.array([0, 1.386294, 1.791760])
```

Sebagai contoh, kita ingin mengetahui berapa nilai y jika $x = 2$:

```
lagrange_interp(x, y, 2.0)
```

```
0.5658439999999999
```

```
NptsPlot = 1000
xx = np.linspace(x[0], x[-1], NptsPlot)
yy = np.zeros(NptsPlot)
for i in range(NptsPlot):
    yy[i] = lagrange_interp(x, y, xx[i])
plt.clf()
plt.plot(x, y, marker="o", label="data")
plt.plot(xx, yy, label="Interpolasi Lagrange (orde-2)")
plt.grid()
plt.legend()
plt.savefig("figures/interp_lagrange_1.pdf")
```

3.2 Interpolasi dengan polinomial Newton

3.2.1 Teori polinomial Newton

Polinomial Newton memiliki bentuk sebagai berikut:

$$P_n = a_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + \cdots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})a_n$$

Koefisien a_n dapat dihitung dengan:

$$\begin{aligned} a_0 &= f(x_0) \\ a_1 &= f[x_1, x_0] \\ a_2 &= f[x_2, x_1, x_0] \\ &\vdots \\ a_n &= f[x_n, x_{n-1}, \ldots, x_1, x_0] \end{aligned}$$

di mana fungsi dengan tanda kurung siku merupakan beda terbagi hingga (*finite divided differences*).
Beda terbagi hingga pertama didefinisikan sebagai:

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

Beda terbagi hingga kedua didefinisikan sebagai:

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, f_k]}{x_i - x_k}$$

Secara umum, untuk beda terbagi hingga ke-n adalah:

$$f[x_n, x_{n-1}, \ldots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \ldots, x_1] - f[x_{n-1}, x_{n-2}, \ldots, x_0]}{x_n - x_0}$$

Implementasi interpolasi Newton

```
def create_newton_polynom(x, y):
    Ndata = len(x) # jumlah data
    coefs = np.copy(y)
    for k in range(1,Ndata):
        coefs[k:Ndata] = (coefs[k:Ndata] - coefs[k-1])/(x[k:Ndata] - x[k-1])
    return coefs
```

```
def eval_newton_polynom(coefs, x, xo):
    N = len(x) - 1 # derajat polinom
    p = coefs[N]
    for k in range(1,N+1):
        p = coefs[N-k] + (xo - x[N-k])*p
    return p
```

Contoh penggunaan interpolasi Newton

```
x = np.array([1.0, 4.0, 6.0])
y = np.array([0, 1.386294, 1.791760])
```

```
coefs = create_newton_polynom(x, y)
coefs
```

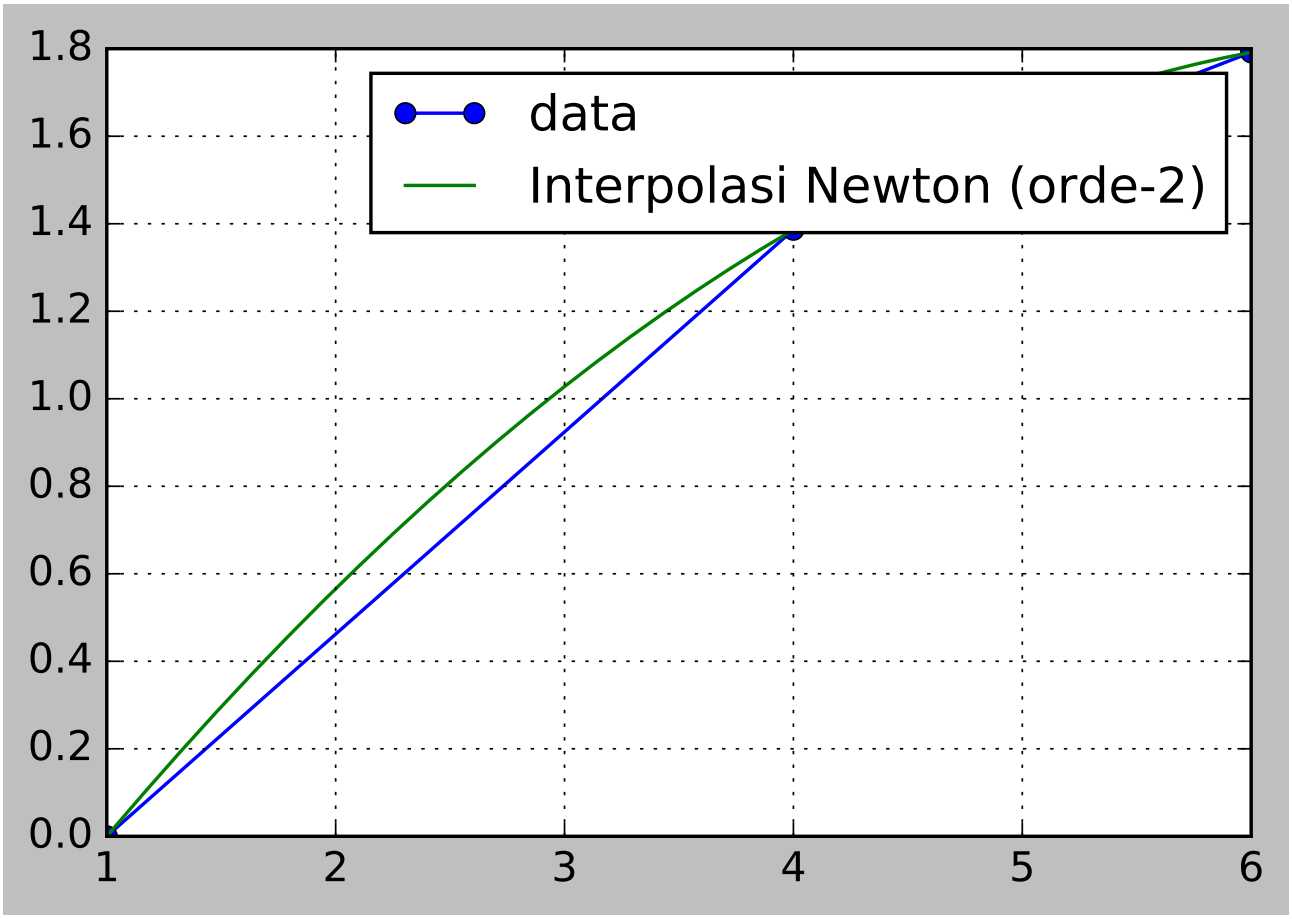
```
array([ 0.      ,  0.462098, -0.051873])
```

```
eval_newton_polynom(coefs, x, 2.0)
```

```
0.5658439999999999
```

```
NptsPlot = 1000
xx = np.linspace(x[0], x[-1], NptsPlot)
yy = np.zeros(NptsPlot)
coefs = create_newton_polynom(x, y)
for i in range(NptsPlot):
    yy[i] = eval_newton_polynom(coefs, x, xx[i])

# Plot
plt.clf()
plt.plot(x, y, marker="o", label="data")
plt.plot(xx, yy, label="Interpolasi Newton (orde-2)")
plt.grid()
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f0fbde88630>
```



3.2.2 Aplikasi interpolasi Newton pada fungsi cos

```
def func_01(x):
    return np.cos(2*x)
```

```
N = 5
A = 0.0
B = 2*np.pi
x_sample = np.linspace(A, B, N)
y_sample = func_01(x_sample)
```

```
x_sample
```

```
array([0.        , 1.57079633, 3.14159265, 4.71238898, 6.28318531])
```

```
y_sample
```

```
array([ 1., -1.,  1., -1.,  1.])
```

```
plt.clf()
plt.plot(x_sample, y_sample);
```

```
[<matplotlib.lines.Line2D at 0x7f0fbde18400>]
```

```
NptsPlot = 500
x_dense = np.linspace(A,B,NptsPlot)
y_dense = func_01(x_dense)

Ninterp = 10
x_interp = np.linspace(A,B,Ninterp)
y_interp = func_01(x_interp)
coefs = create_newton_polynom(x_interp, y_interp)

x_interp_plt = np.linspace(A,B,NptsPlot)
y_interp_plt = np.zeros(NptsPlot)
for i in range(NptsPlot):
    y_interp_plt[i] = eval_newton_polynom(coefs, x_interp, x_interp_plt[i])
```

```
plt.clf()
plt.plot(x_sample, y_sample, marker="o", label="sampled")
plt.plot(x_dense, y_dense, label="exact")
plt.plot(x_interp_plt, y_interp_plt, label="interp")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f0fbdd8d6a0>
```

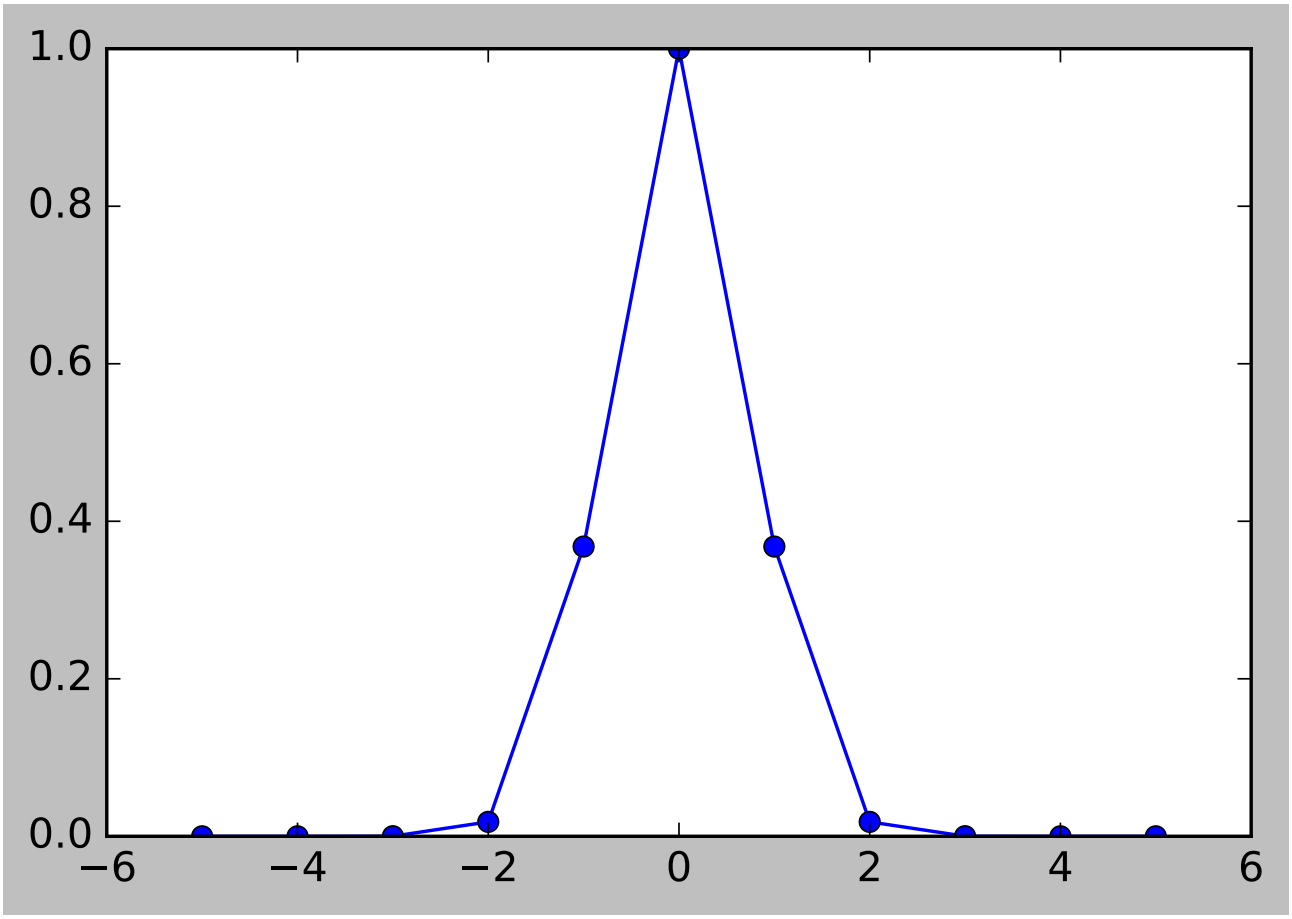
Latihan dengan fungsi yang lain

```
def func_02(x):
    return np.exp(-x**2)
```

```
N = 11
A = -5
B = 5
x_sample = np.linspace(A, B, N)
y_sample = func_02(x_sample)
```

```
plt.clf()
plt.plot(x_sample, y_sample, marker="o");
```

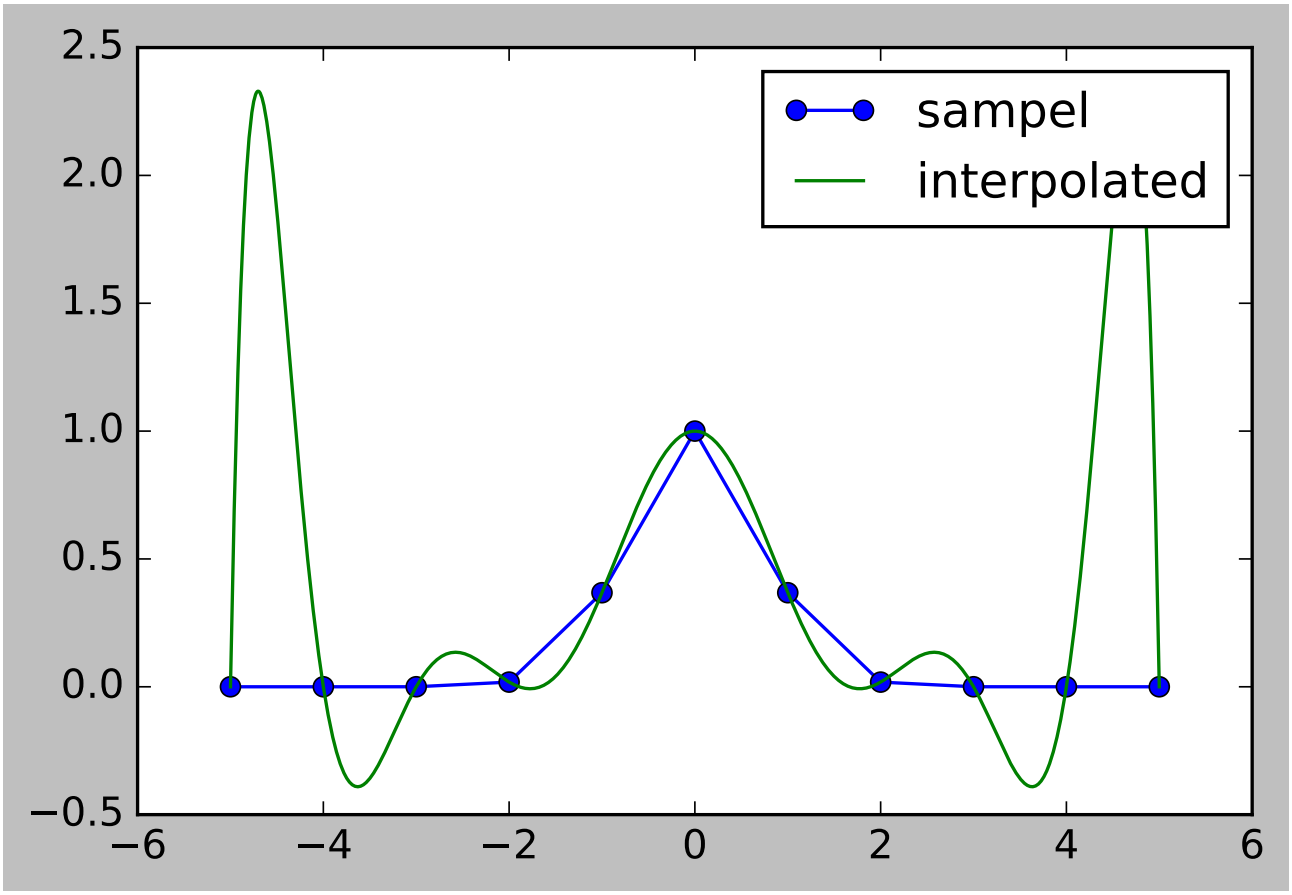
```
[<matplotlib.lines.Line2D at 0x7f0fbdea3da0>]
```




```
NptsPlot = 1000
xx = np.linspace(A, B, NptsPlot)
yy = np.zeros(NptsPlot)
for i in range(NptsPlot):
    yy[i] = lagrange_interp(x_sample, y_sample, xx[i])
```

```
plt.clf()
plt.plot(x_sample, y_sample, marker="o", label="sampel")
plt.plot(xx, yy, label="interpolated");
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f0fbdea33c8>
```

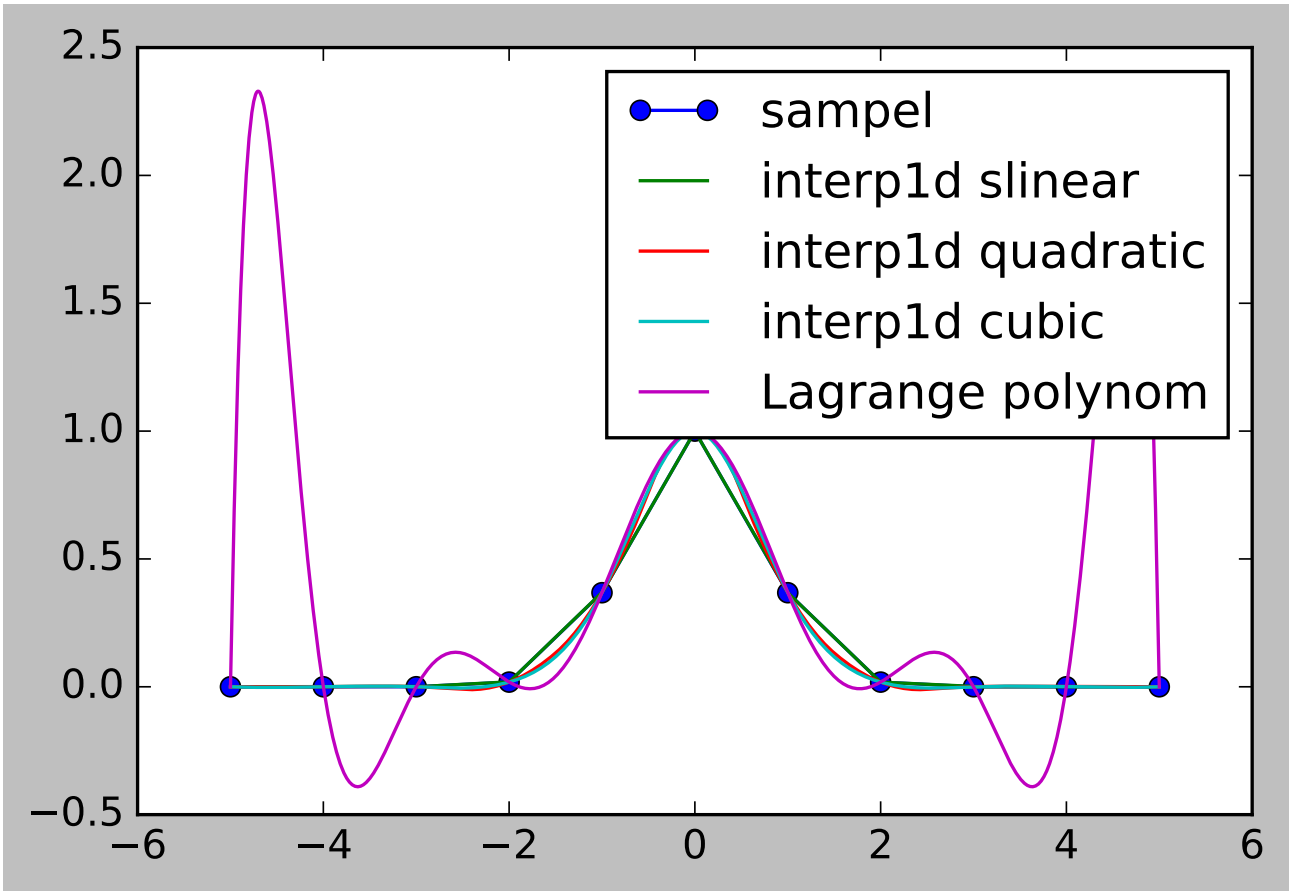


Fungsi Gaussian dengan interp1d

```
NptsPlot = 1000
xx = np.linspace(A, B, NptsPlot)

f_slinear = scipy.interpolate.interp1d(x_sample, y_sample, kind="slinear")
f_quadratic = scipy.interpolate.interp1d(x_sample, y_sample, kind="quadratic")
f_cubic = scipy.interpolate.interp1d(x_sample, y_sample, kind="cubic")

yy_slinear = f_slinear(xx)
yy_quadratic = f_quadratic(xx)
yy_cubic = f_cubic(xx)
```

```
plt.clf()
plt.plot(x_sample, y_sample, marker="o", label="sampel")
plt.plot(xx, yy_slinear, label="interp1d slinear")
plt.plot(xx, yy_quadratic, label="interp1d quadratic")
plt.plot(xx, yy_cubic, label="interp1d cubic")
plt.plot(xx, yy, label="Lagrange polynom");
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f0fbdef84a8>
```

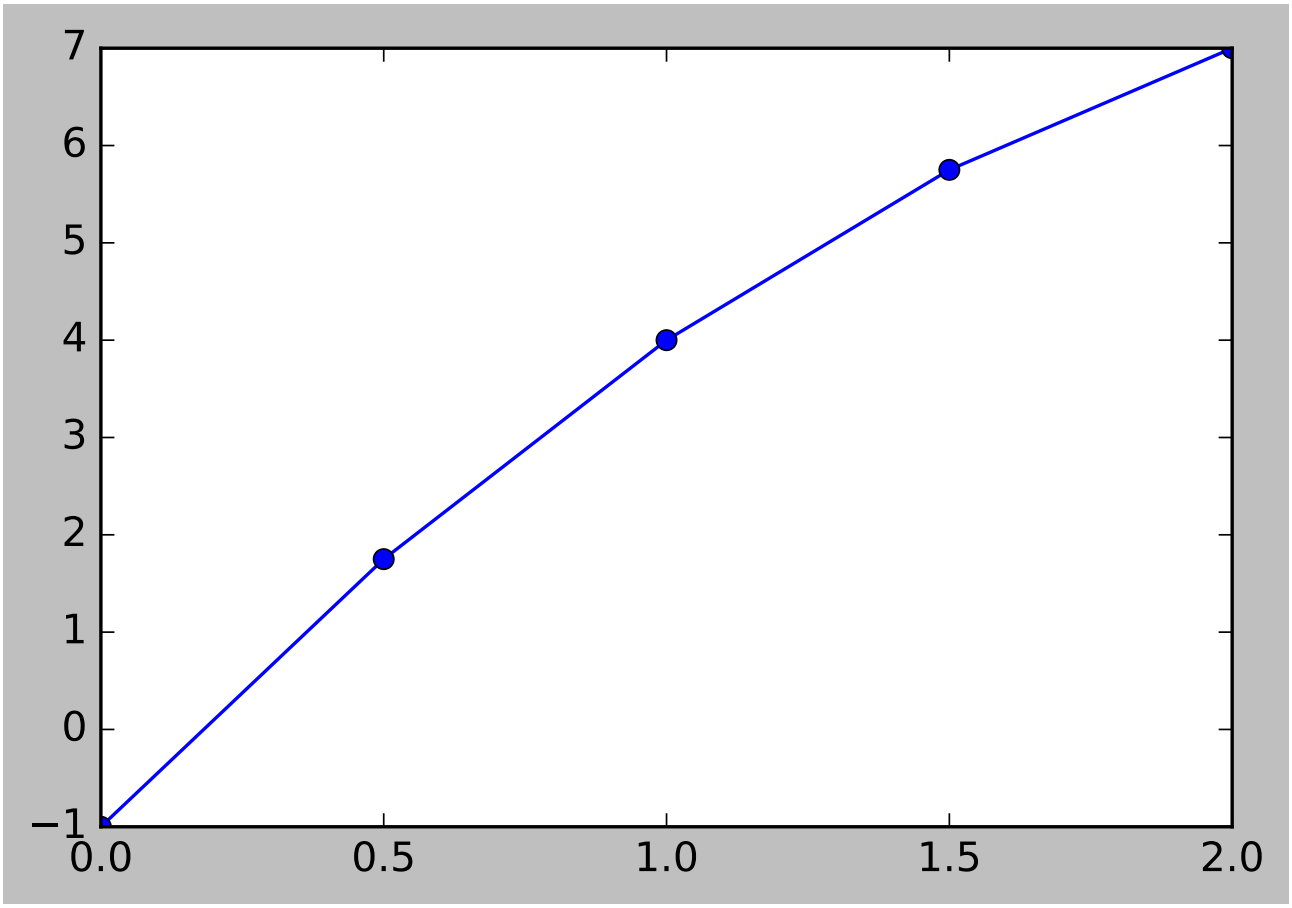


Kiusalaas 3.1 no. 4

```
x = np.array([0, 0.5, 1, 1.5, 2])
y = np.array([-1.00, 1.75, 4.00, 5.75, 7.00])
```

```
plt.clf()
plt.plot(x, y, marker="o")
```

```
[<matplotlib.lines.Line2D at 0x7f0fbd908898>]
```



Nilai fungsi pada $x = \pi/4$ adalah

```
lagrange_interp(x, y, np.pi/4)
```

```
3.0955387053166046
```

```
np.pi/4
```

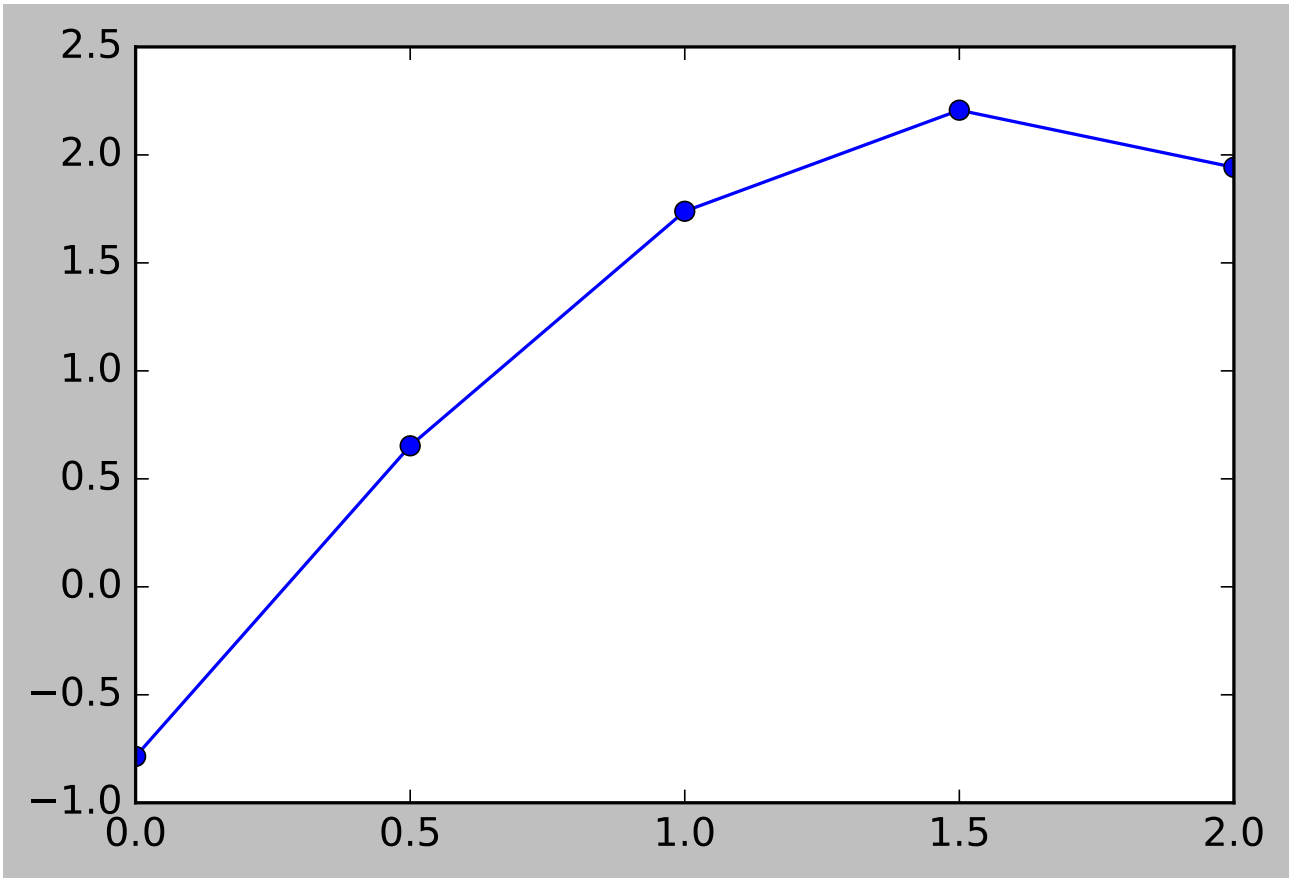
```
0.7853981633974483
```

Kiusalaas 3.1 no 5

```
x = np.array([0, 0.5, 1, 1.5, 2])
y = np.array([-0.7854, 0.6529, 1.7390, 2.2071, 1.9425])
```

```
plt.clf()
plt.plot(x, y, marker="o")
```

```
[<matplotlib.lines.Line2D at 0x7f0fbc052160>]
```



Nilai fungsi pada $x = \pi/4$ adalah

```
lagrange_interp(x, y, np.pi/4)
```

```
1.3364947265058855
```

Nilai fungsi pada $x = \pi/2$

```
lagrange_interp(x, y, np.pi/2)
```

```
2.2149439701239673
```

```
neville_interp(x, y, np.pi/2)
```

```
---------------------------------------------------------------------------NameError
Traceback (most recent call last)<ipython-input-1-261f5d87dec5> in
<module>
----> 1 neville_interp(x, y, np.pi/2)
NameError: name 'neville_interp' is not defined
```

Dengan menggunakan `interp1d`:

```
f_cubic = scipy.interpolate.interp1d(x, y, kind="cubic")
```

```
f_cubic([np.pi/2])[0]
```

```
2.2159412331093766
```

```
type(f_cubic(0.1))
```

```
numpy.ndarray
```

```
res = f_cubic(0.1)
```

```
res.item()
```

```
-0.4841356
```

3.3 Metode Neville

Metode ini pada dasarnya merupakan bentuk alternatif dari polinomial Newton.
Implementasi metode Neville

```
def neville_interp(x, y_, xx):
    m = len(x)
    y = np.copy(y_)
    for k in range(1,m):
        y[0:m-k] = ((xx- x[k:m])*y[0:m-k] + (x[0:m-k] - xx)*y[1:m-k+1])/(x[0:m-k] - x[k:m])
    return y[0]
```

Contoh penggunaan metode Neville

```
x = np.array([1.0, 4.0, 6.0])
y = np.array([0, 1.386294, 1.791760])
neville_interp(x, y, 2.0)
```

```
0.5658439999999999
```

3.4 Interpolasi dengan fungsi rasional

Aproksimasi ini baik digunakan untuk fungsi yang memiliki pole, akan tetapi pada beberapa jenis data atau fungsi interpolasi ini tidak stabil.
Implementasi dalam Python

```
def rational_interp(x, y, xx):
    m = len(x)
    r = np.copy(y)
    rOld = np.zeros(m)
    #SMALL = np.finfo("float64").eps
    SMALL = 1e-8
    for k in range(m-1):
        for i in range(m-k-1):
            if (abs(xx - x[i+k+1]) < SMALL):
                return y[i+k+1]
            else:
                c1 = r[i+1] - r[i]
                c2 = r[i+1] - rOld[i+1]
                c3 = (xx - x[i])/(xx - x[i+k+1])
                r[i] = r[i+1] + c1/(c3*(1.0 - c1/c2) - 1.0)
                rOld[i+1] = r[i+1]
    return r[0]
```

Contoh penggunaan interpolasi fungsi rasional

```
x = np.array([0.0, 0.6, 0.8, 0.95])
y = np.array([0.0, 1.3764, 3.0777, 12.7062])
```

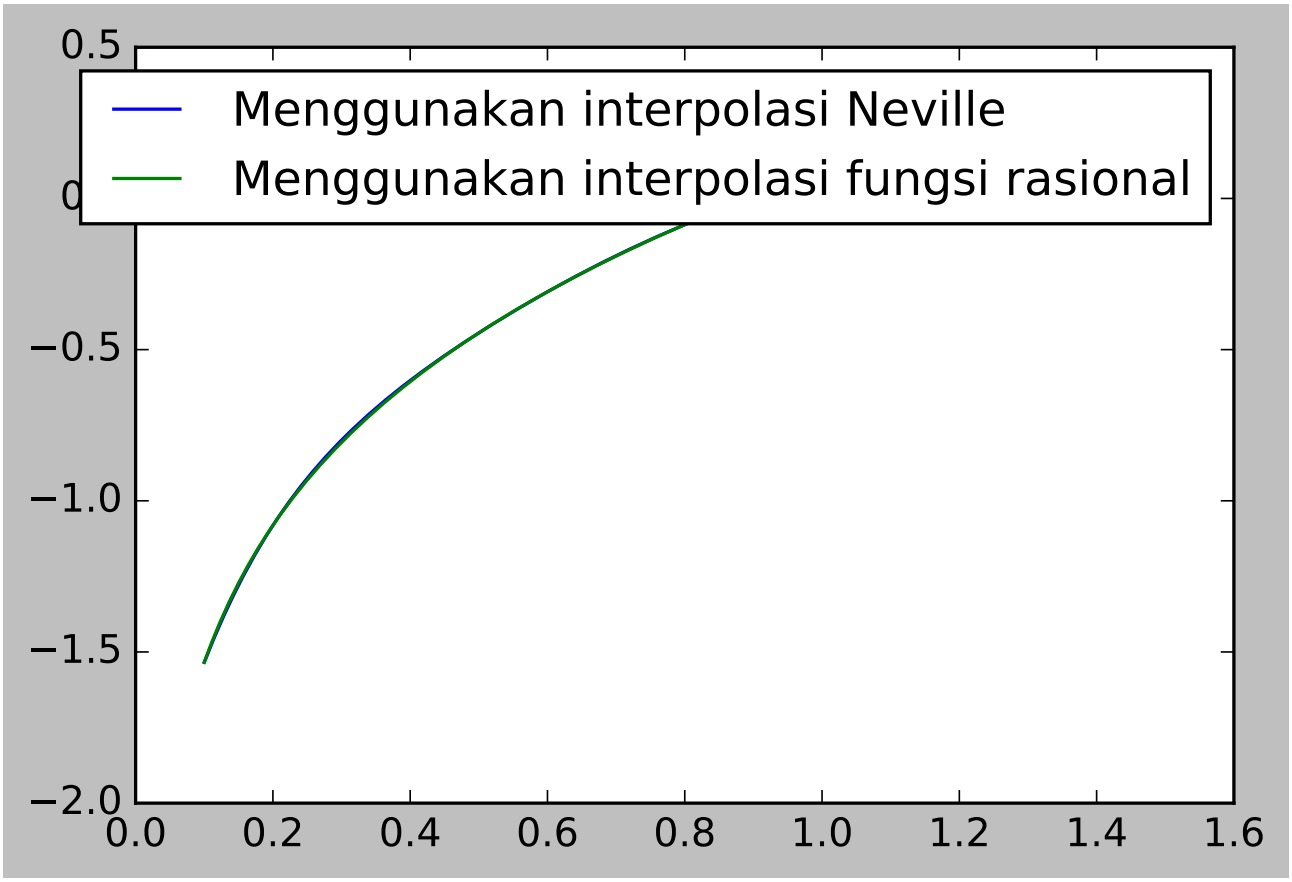
```
rational_interp(x, y, 0.5)
```

```
1.0131205116558464
```

```
x = np.array([0.1, 0.2, 0.5, 0.6, 0.8, 1.2, 1.5])
y = np.array([-1.5342, -1.0811, -0.4445, -0.3085, -0.0868, 0.2281, 0.3824])
```

```
NptsPlot = 500
A = 0.1
B = 1.5
x_plot = np.linspace(0.1, 1.5, NptsPlot)
y_neville = np.zeros(NptsPlot)
y_rational = np.zeros(NptsPlot)
for i in range(NptsPlot):
    y_neville[i] = neville_interp(x, y, x_plot[i])
    y_rational[i] = rational_interp(x, y, x_plot[i])
plt.clf()
plt.plot(x_plot, y_neville, label="Menggunakan interpolasi Neville")
plt.plot(x_plot, y_rational, label="Menggunakan interpolasi fungsi rasional")
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f0fac702e10>
```



Tes fungsi x^2

```
def func_02(x):
    return x**2 - 3.0
```

```
A = -1.5
B = 1.5

Nsample = 5
x_sample = np.linspace(A, B, Nsample)
y_sample = func_02(x_sample)

NptsPlot = 100
x_plot = np.linspace(A, B, NptsPlot)
y_exact = func_02(x_plot)
y_neville = np.zeros(NptsPlot)
y_rational = np.zeros(NptsPlot)
for i in range(NptsPlot):
    y_neville[i] = neville_interp(x_sample, y_sample, x_plot[i])
    y_rational[i] = rational_interp(x_sample, y_sample, x_plot[i])
plt.clf()
plt.plot(x_plot, y_exact, label="eksak")
plt.plot(x_plot, y_neville, label="Interpolasi Neville")
plt.plot(x_plot, y_rational, label="Interpolasi fungsi rasional")
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f0fac6fce10>
```

Tes fungsi x^3

```
def func_03(x):
    return x**3 - 3.0
```

```
A = -1.5
B = 1.5

Nsample = 10
x_sample = np.linspace(A, B, Nsample)
y_sample = func_03(x_sample)

NptsPlot = 100
x_plot = np.linspace(A, B, NptsPlot)
y_exact = func_03(x_plot)
y_neville = np.zeros(NptsPlot)
y_rational = np.zeros(NptsPlot)
for i in range(NptsPlot):
    y_neville[i] = neville_interp(x_sample, y_sample, x_plot[i])
    y_rational[i] = rational_interp(x_sample, y_sample, x_plot[i])
plt.clf()
plt.plot(x_plot, y_exact, label="eksak")
plt.plot(x_plot, y_neville, label="Interpolasi Neville")
plt.plot(x_plot, y_rational, label="Interpolasi rational")
plt.legend();
```

```
/home/efefer/miniconda3/bin/pweave:15: RuntimeWarning: divide by zero
encountered in double_scalars
/home/efefer/miniconda3/bin/pweave:15: RuntimeWarning: invalid value
encountered in double_scalars
```

```
<matplotlib.legend.Legend at 0x7f0fac673e48>
```

3.5 Interpolasi dengan spline

TODO.

Merupakan metode yang dianggap paling robus dan banyak digunakan dalam aplikasi. Implementasinya cukup rumit untuk dilakukan.

3.6 Menggunakan pustaka Python

Modul `scipy.interpolate` menyediakan banyak fungsi untuk melakukan interpolasi. Informasi lebih lengkap dapat dilihat pada dokumentasinya [1].

Beberapa fungsi yang sering dipakai adalah:

- fungsi ‘interp1d‘
- ‘splrep‘ dan ‘splev‘ (menggunakan metode B-spline)

Buat data dengan menggunakan fungsi yang sudah kita ketahui bentuknya.

```
A = 0.0
B = np.pi
Nsample = 7
x_sample = np.linspace(A,B,Nsample)
y_sample = func_01(x_sample)
```

Buat interpolant, dengan menggunakan spline linear, kuadratic, dan kubik.

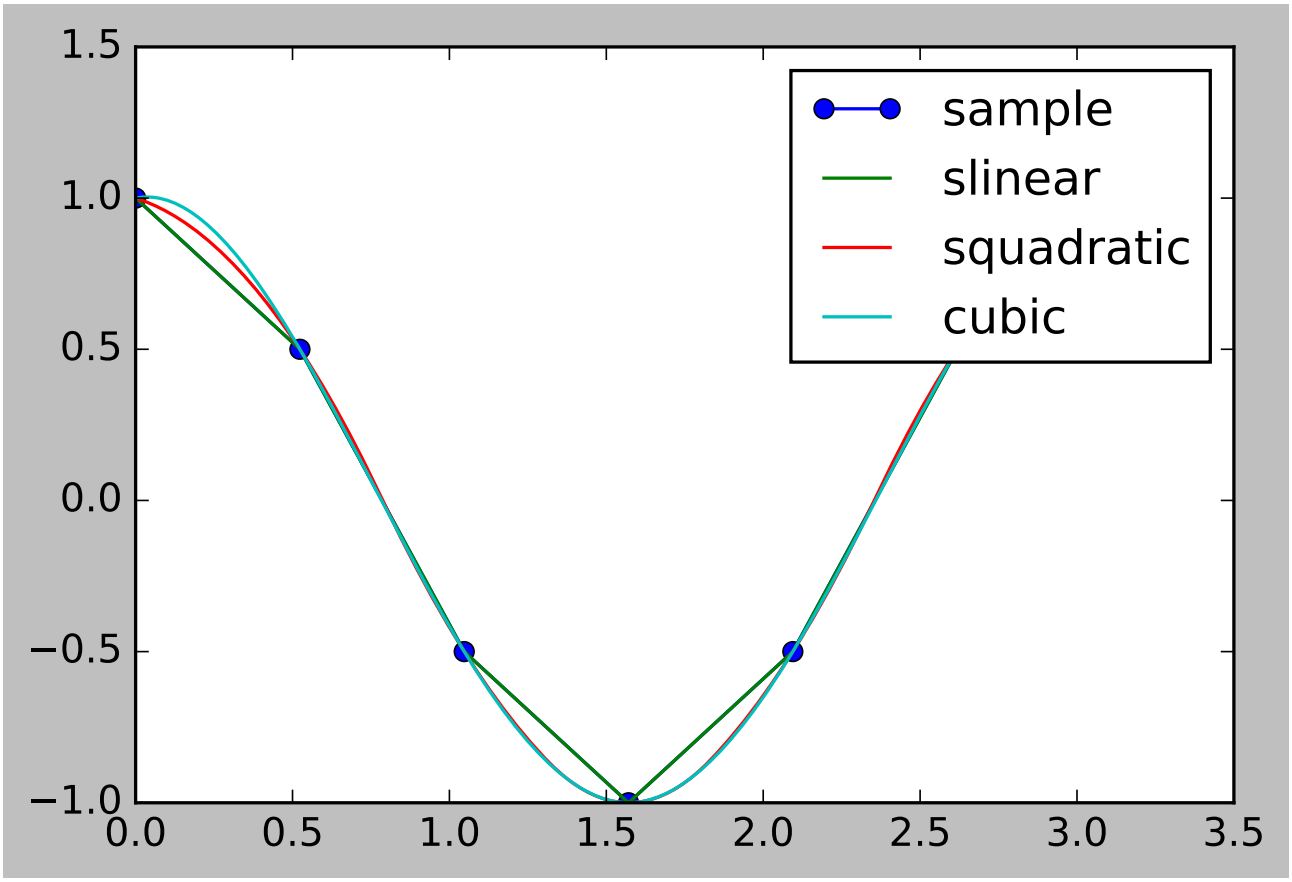
```
f_slinear = scipy.interpolate.interp1d(x_sample, y_sample, kind="slinear")
f_quadratic = scipy.interpolate.interp1d(x_sample, y_sample, kind="quadratic")
f_cubic = scipy.interpolate.interp1d(x_sample, y_sample, kind="cubic")
```

```
NptsPlot = 500
x_plot = np.linspace(A,B,NptsPlot)
y_slinear = f_slinear(x_plot)
y_quadratic = f_quadratic(x_plot)
y_cubic = f_cubic(x_plot)
```

```
plt.clf()
plt.plot(x_sample, y_sample, marker="o", label="sample")
plt.plot(x_plot, y_slinear, label="slinear")
plt.plot(x_plot, y_quadratic, label="squadratic")
plt.plot(x_plot, y_cubic, label="cubic")
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f0fac5f9eb8>
```

[1] https://docs.scipy.org/doc/scipy/reference/interpolate.html


sample
slinear
squadratic
cubic
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

Bab 4

Pencocokan Kurva

4.1 Kuadrat terkecil (least square)

```
"""
c = polyFit(xData,yData,m).
Returns coefficients of the polynomial
p(x) = c[0] + c[1]x + c[2]x2 +...+ c[m]xm
that fits the specified data in the least squares sense.
"""
def polyN_fit(xData, yData, m):
    a = np.zeros((m+1,m+1))
    b = np.zeros(m+1)
    s = np.zeros(2*m+1)
    for i in range(len(xData)):
        temp = yData[i]
        for j in range(m+1):
            b[j] = b[j] + temp
            temp = temp*xData[i]
        temp = 1.0
        for j in range(2*m+1):
            s[j] = s[j] + temp
            temp = temp*xData[i]
    for i in range(m+1):
        for j in range(m+1):
            a[i,j] = s[i+j]

    c = np.linalg.solve(a,b)
    return c
```

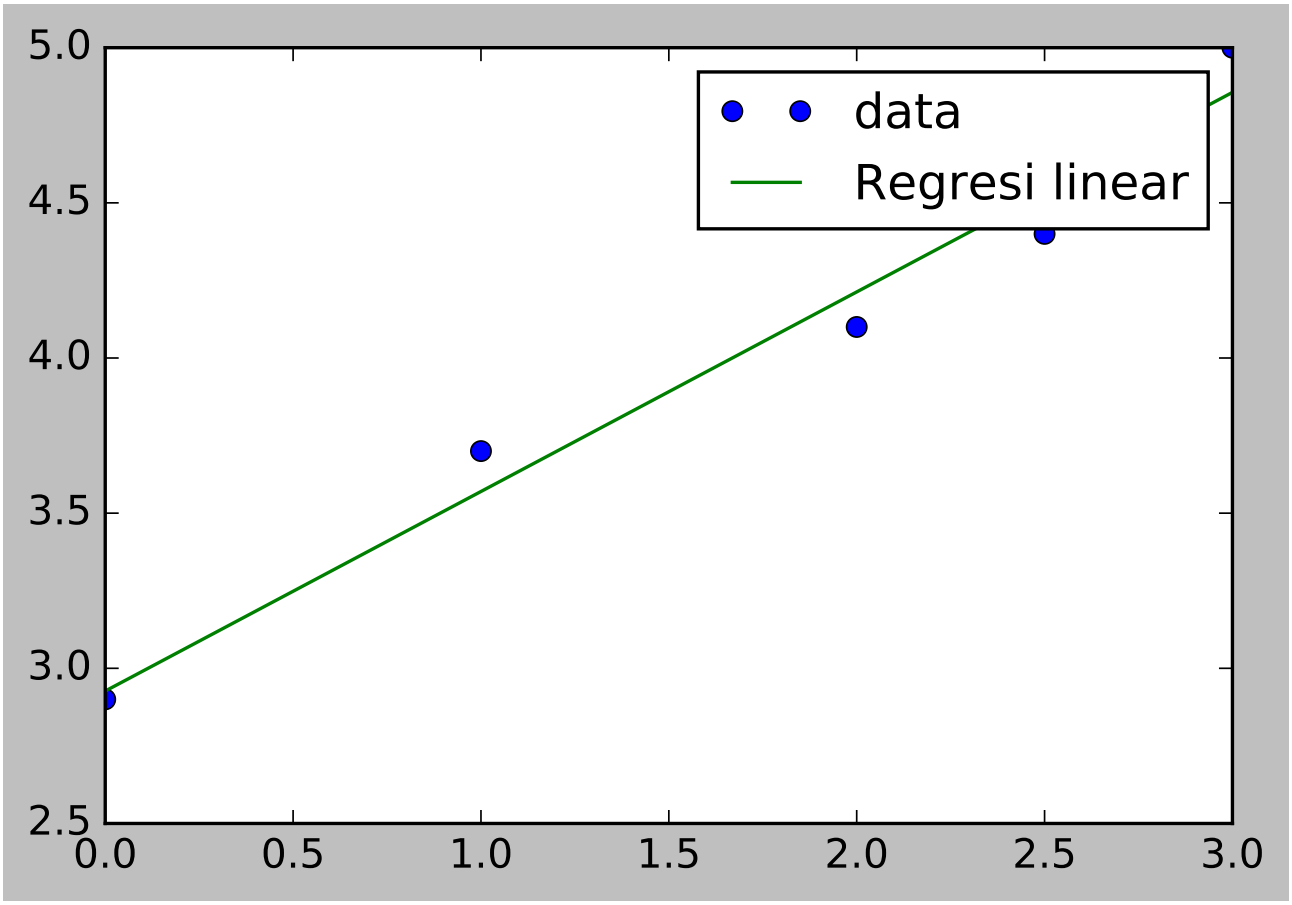
```
x_data = np.array([0.0, 1.0, 2.0, 2.5, 3.0])
y_data = np.array([2.9, 3.7, 4.1, 4.4, 5.0])
```

```
linear_reg = polyN_fit(x_data, y_data, 1)
linear_reg
```

```
array([2.92672414, 0.64310345])
```

```
plt.clf()
plt.plot(x_data, y_data, marker="o", linewidth=0, label="data")
x_plt = np.linspace(x_data[0], x_data[-1], 200)
plt.plot(x_plt, x_plt*linear_reg[1] + linear_reg[0], label="Regresi linear")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f10fc4dd748>
```




```
n = len(x_data) - 1
m = len(c) - 1
sigma = 0.0
for i in range(n+1):
    p = evalPoly(c, x_data[i])
    sigma = sigma + (y_data[i] - p)**2
    sigma = math.sqrt(sigma/(n - m))
    return sigma
```

```
---------------------------------------------------------------------------NameError
Traceback (most recent call last)<ipython-input-1-d2c53ce21a17> in
<module>
      1 n = len(x_data) - 1
----> 2 m = len(c) - 1
      3 sigma = 0.0
      4 for i in range(n+1):
      5     p = evalPoly(c, x_data[i])
NameError: name 'c' is not defined
```

Bab 5

Integrasi Numerik

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.style.use("classic")
```

5.1 Metode segiempat

Asumsi: data diberikan dalam bentuk tabular dengan selang Δx yang sama.

```
def integ_rectangular(f, x):
    N = len(f)
    assert N == len(x)
    a = x[0]
    b = x[-1]
    s = 0.0
    for i in range(N):
        s = s + f[i]
    return s*(b-a)/N
```

Contoh 1

Hitung integral berikut secara numerik dengan metode segi empat

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x$$

```
def my_func(x):
    return np.sin(x)

a = 0.0
b = np.pi
Npts = 500
x = np.linspace(a,b,Npts)
f_array = np.zeros(Npts)
for i in range(Npts):
    f_array[i] = my_func(x[i])

integ_result = integ_rectangular(f_array, x)
print("integ_result = ", integ_result)
```

```
integ_result =  1.995993407073535
```

Contoh 2

Hitung integral berikut secara numerik dengan metode segi empat

$$\int_{-\pi/2}^{\pi/2} \cos(x)\,\mathrm{d}x$$

```
def my_func(x):
    return np.cos(x)

a = -np.pi/2
b = np.pi/2
Npts = 500
x = np.linspace(a,b,Npts)
f_array = np.zeros(Npts)
for i in range(Npts):
    f_array[i] = my_func(x[i])

integ_result = integ_rectangular(f_array, x)
print("integ_result = ", integ_result)
```

```
integ_result =  1.9959934070735346
```

5.2 Metode Trapesium

Asumsi: data diberikan dalam bentuk tabular dengan selang Δx yang sama.

```
def integ_trapezoid(f, x):
    N = len(f)
    assert N == len(x)
    a = x[0]
    b = x[-1]
    s = f[0] + f[-1]
    for i in range(1,N-1):
        s = s + 2*f[i]
    return 0.5*s*(b-a)/(N-1)
```

Contoh 1

Hitung integral berikut secara numerik dengan metode trapesium.

$$\int_0^{\pi} \sin(x)\,\mathrm{d}x$$

```
def my_func(x):
    return np.sin(x)

a = 0.0
b = np.pi
Npts = 500
x = np.linspace(a,b,Npts)
f_array = np.zeros(Npts)
for i in range(Npts):
    f_array[i] = my_func(x[i])

integ_result = integ_trapezoid(f_array, x)
print("integ_result = ", integ_result)
```

```
integ_result =  1.9999933938612575
```

Contoh 2

Hitung integral berikut secara numerik dengan metode trapesium.

$$\int_{-pi/2}^{\pi/2} \cos(x)\,\mathrm{d}x$$

```
def my_func(x):
    return np.cos(x)

a = -np.pi/2
b = np.pi/2
Npts = 500
x = np.linspace(a,b,Npts)
f_array = np.zeros(Npts)
for i in range(Npts):
    f_array[i] = my_func(x[i])

integ_result = integ_trapezoid(f_array, x)
print("integ_result = ", integ_result)
```

```
integ_result =  1.999993393861257
```

5.3 Metode Simpson

TODO

Bab 6

Persamaan Diferensial Biasa

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.style.use("default")
import scipy.integrate
```

Dalam bab ini akan diberikan implementasi sederhana dari beberapa metode yang dapat digunakan untuk menyelesaikan persamaan diferensial biasa.

Bentuk umum dari persamaan diferensial biasa orde 1 adalah:

$$y' = \frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y)$$

Solusi dari persamaan ini memiliki satu konstanta sembarang. Konstanta ini dapat ditentukan nilainya jika diberikan nilai y pada suatu titik, misalnya $y(a) = \alpha$.

Persamaan diferensial biasa orde n dapat dituliskan sebagai:

$$y^{(n)} = \frac{\mathrm{d}^n y}{\mathrm{d}x^n} = f\left(x, y, y', \ldots, y^{(n-1)}\right)$$

Dengan menggunakan notasi $y_0 = y$, $y_1 = y'$, $y_2 = y''$, ..., $y_{n-1} = y^{(n-1)}$, persamaan diferensial biasa orde n dapat diubah menjadi persamaan diferensial oder satu sebagai berikut:

$$\begin{aligned}
y_0' &= y_1 \\
y_1' &= y_2 \\
y_2' &= y_3 \\
\cdots &= \cdots \\
y_n' &= f\left(x, y_0, y_1, \ldots, y_{n-1}\right)
\end{aligned}$$

6.1 Metode Euler

```
def ode_euler(f, xi, xf, y0, N, verbose=False):

    h = (xf - xi)/N

    if verbose:
        print("ode_euler h = %18.10f\n" % (h))

    xsol = xi + np.arange(0,N+1)*h

    # orde dari ODE ditentukan dari jumlah syarat awal yang diberikan
    Ndim = len(y0)

    # array untuk solusi
    ysol = np.zeros((N+1,Ndim))
```

```
    # aplikasi syarat awal
    ysol[0,:] = y0[:]

    for i in range(N):
        ysol[i+1,:] = ysol[i,:] + h*f(xsol[i], ysol[i,:])

    return xsol, ysol
```

Contoh 1

Cari solusi dari persamaan diferensial:

$$y' + 4y = x^2$$

pada rentang x $[0, 1]$ dengan syarat awal $y(0) = 1$. Bandingkan hasil yang diperoleh dengan solusi analitik:

$$y = \frac{31}{32}e^{-4x} + \frac{1}{4}x^2 - \frac{1}{8}x + \frac{1}{32}$$

Ubah persamaan menjadi bentuk standar $y' = f(x, y)$

$$y' = x^2 - 4y$$

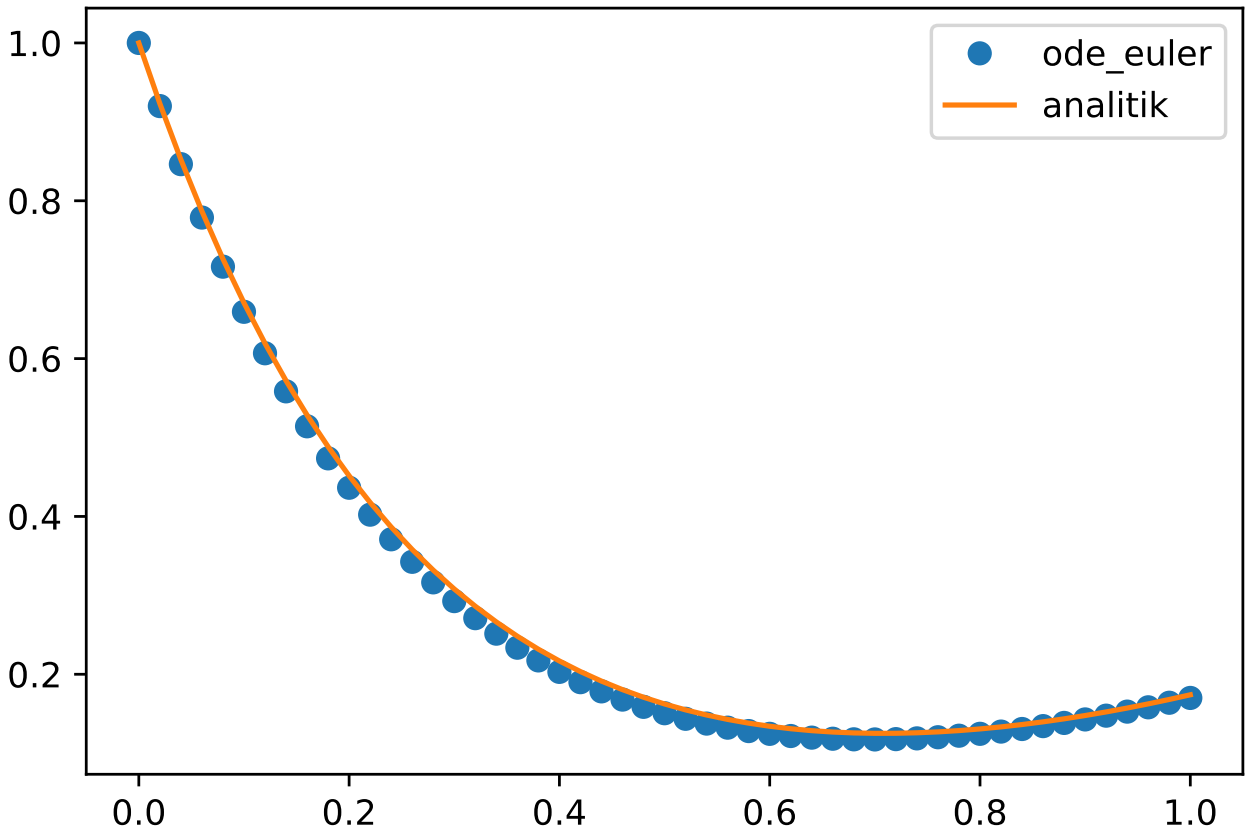
sehingga $f(x, y) = x^2 - 4y$

```
def my_ode_problem_01(x,y):
    return x**2 - 4*y

xi = 0.0
xf = 1.0
y0 = [1.0]  # syarat batas dalam bentuk list
Ninterval = 50
xsol, ysol = ode_euler(my_ode_problem_01, xi, xf, y0, Ninterval, verbose=True)

def analytic_sol_01(x):
    return (31.0/32.0)*np.exp(-4*x) + x**2/4 - x/8 + 1.0/32.0
```

```
ode_euler h =       0.0200000000
```

```
plt.clf()
plt.plot(xsol, ysol[:,0], label="ode_euler", marker="o", linewidth=0)
plt.plot(xsol, analytic_sol_01(xsol), label="analitik")
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f759b1b90f0>
```



Contoh 2

Selesaikan persamaan diferensial berikut:

$$y'' = -0.1y' - x$$

pada $x = 0$ sampai dengan 2 dengan syarat awal $y(0) = 0$ dan $y'(0) = 1$ dengan menggunakan ukuran langkah $h = 0.05$.

Bandingkan dengan solusi analitik:

$$y = 100x - 5x^2 + 990\left(e^{-0.1x} - 1\right)$$

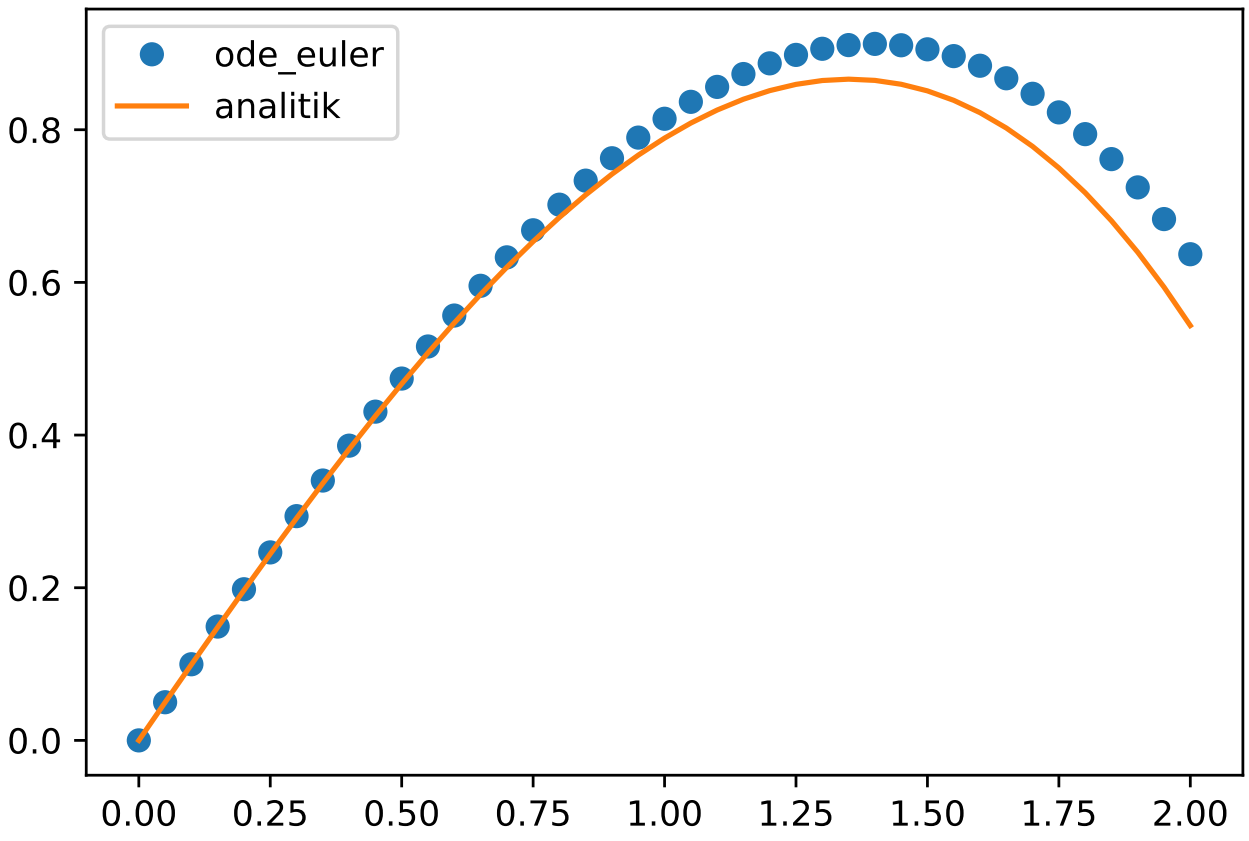
```
def my_ode_problem_02(x, y):
    f = np.zeros(2)
    f[0] = y[1]
    f[1] = -0.1*y[1] - x
    return f

def analytic_sol_02(x):
    return 100*x - 5*x**2 + 990*(np.exp(-0.1*x) -1)
```

```
xi = 0.0
xf = 2.0
y0 = [0.0, 1.0]
h = 0.05
Ninterval = int( (xf - xi)/h )
xsol, ysol = ode_euler(my_ode_problem_02, xi, xf, y0, Ninterval, verbose=True)
```

```
ode_euler h =       0.0500000000
```

```
plt.clf()
plt.plot(xsol, ysol[:,0], label="ode_euler", marker="o", linewidth=0)
plt.plot(xsol, analytic_sol_02(xsol), label="analitik")
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f759b185128>
```



6.2 Metode Runge-Kutta orde 4

```
def ode_RK4(f, xi, xf, y0, N, verbose=False):

    h = (xf - xi)/N

    if verbose:
        print("ode_RK4 h = %18.10f\n" % (h))

    xsol = xi + np.arange(0,N+1)*h

    # orde dari ODE ditentukan dari jumlah syarat awal yang diberikan
    Ndim = len(y0)

    # array untuk solusi
    ysol = np.zeros((N+1,Ndim))

    # aplikasi syarat awal
    ysol[0,:] = y0[:]

    for i in range(N):
        f1 = h*f( xsol[i], ysol[i,:] )
        f2 = h*f( xsol[i] + h/2, ysol[i,:] + f1/2 )
        f3 = h*f( xsol[i] + h/2, ysol[i,:] + f2/2 )
        f4 = h*f( xsol[i] + h, ysol[i,:] + f3 )
        ysol[i+1,:] = ysol[i,:] + (f1 + 2*(f2 + f3) + f4)/6

    return xsol, ysol
```

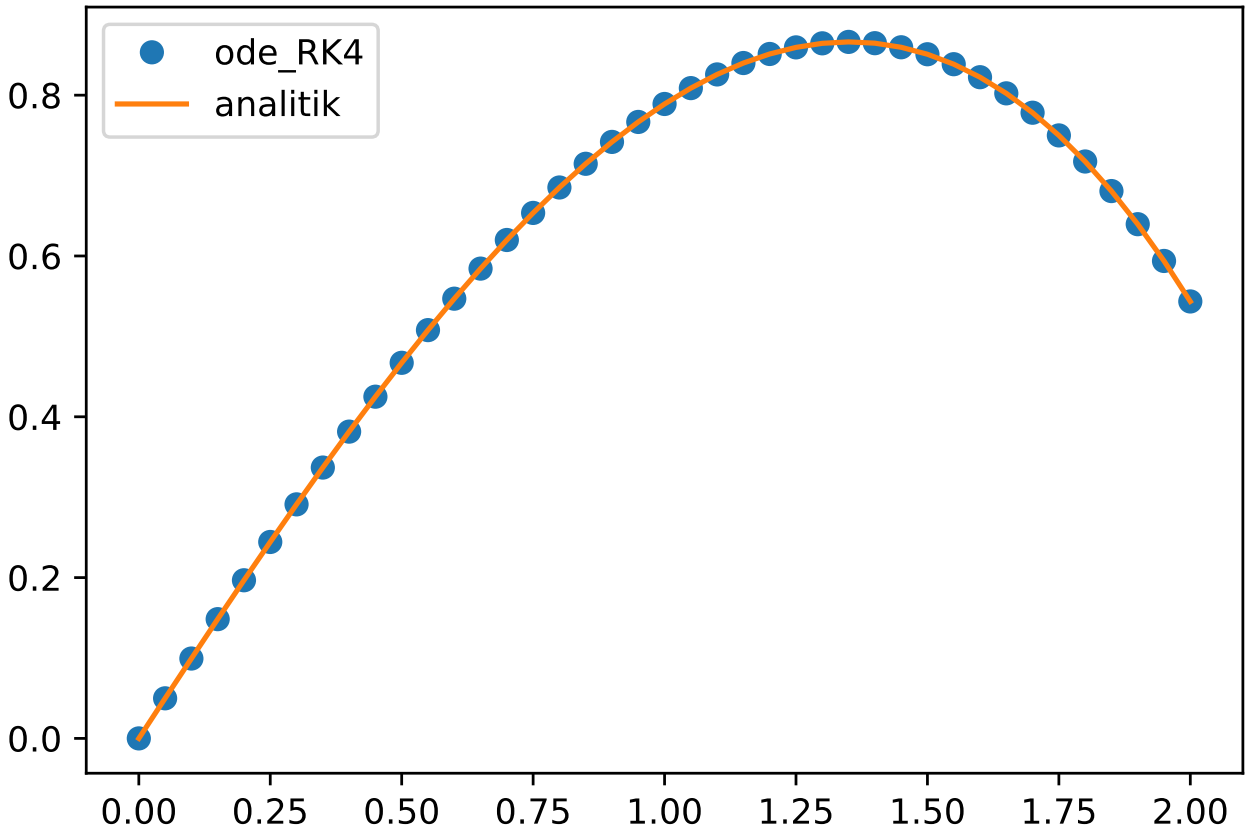
Contoh

```
xi = 0.0
xf = 2.0
y0 = [0.0, 1.0]
h = 0.05
Ninterval = int( (xf - xi)/h )
xsol, ysol = ode_RK4(my_ode_problem_02, xi, xf, y0, Ninterval, verbose=True)
```

```
ode_RK4 h =       0.0500000000
```

```
plt.clf()
plt.plot(xsol, ysol[:,0], label="ode_RK4", marker="o", linewidth=0)
plt.plot(xsol, analytic_sol_02(xsol), label="analitik")
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7f759b0d3ba8>
```



6.3 Latihan soal di slide

```
def soal_slide_no15(x, y):
    return -2*x**3 + 12*x**2 - 20*x + 8.5

def soal_slide_no15_analitik(x):
    return -0.5*x**4 + 4*x**3 - 10*x**2 + 8.5*x + 1
```

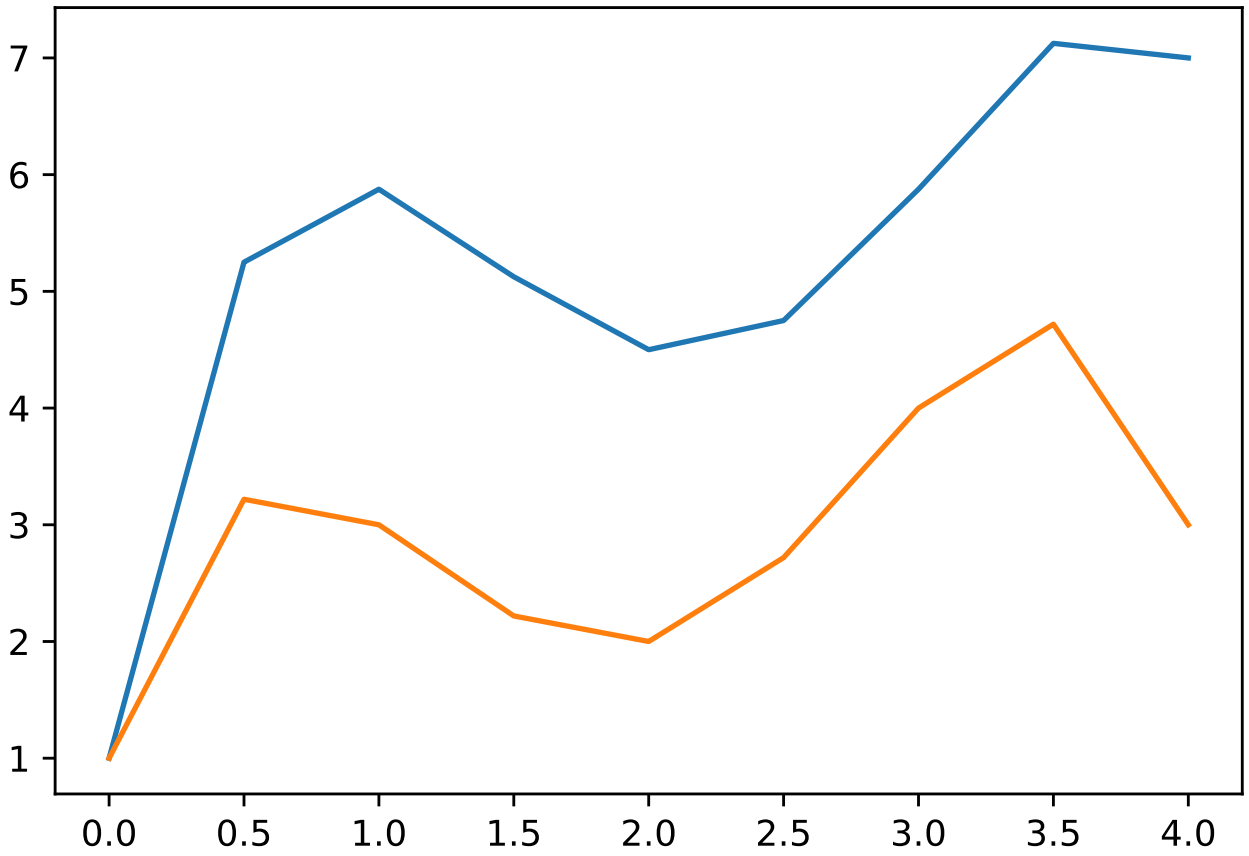
```
xi = 0.0
xf = 4.0
y0 = [1.0] # syarat awal
h = 0.5
```

```
Ninterval = int( (xf - xi)/h )
xsol, ysol = ode_euler(soal_slide_no15, xi, xf, y0, Ninterval, verbose=True)
```

```
ode_euler h =       0.5000000000
```

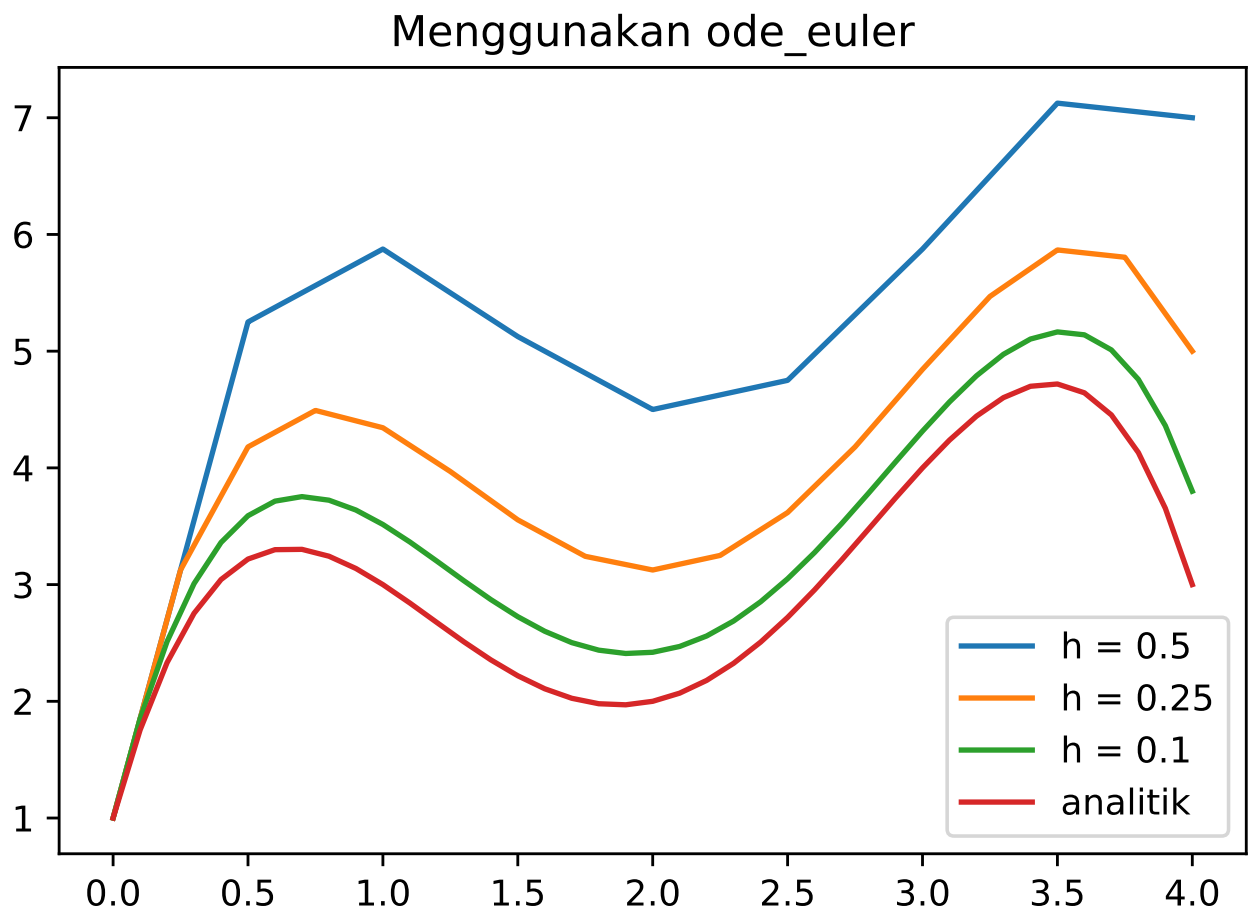
```
plt.clf()
plt.plot(xsol, ysol)
plt.plot(xsol, soal_slide_no15_analitik(xsol))
```

```
[<matplotlib.lines.Line2D at 0x7f759b05df60>]
```



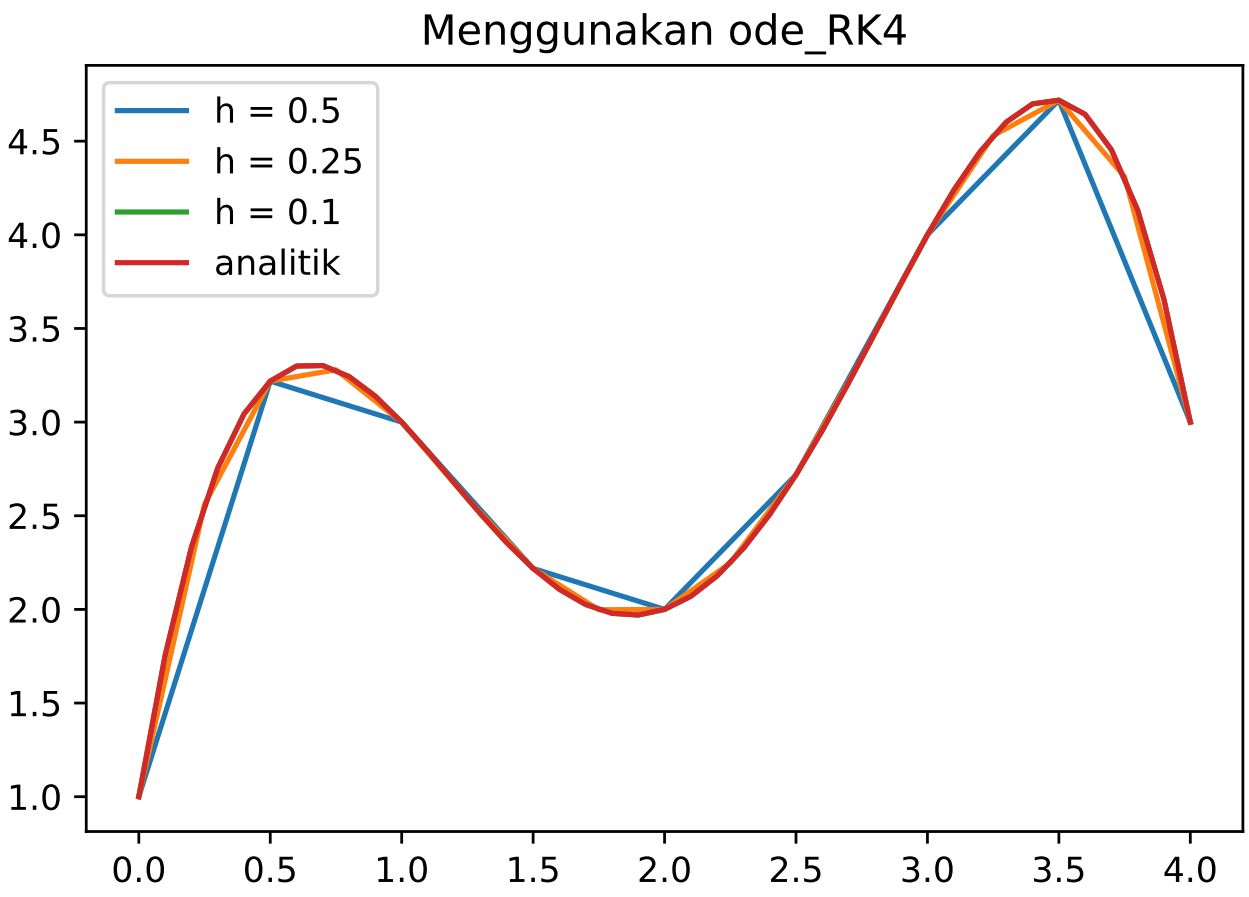

```
xi = 0.0
xf = 4.0
y0 = [1.0] # syarat awal
hh = [0.5, 0.25, 0.1]

plt.clf()
for h in hh:
    Ninterval = int( (xf - xi)/h )
    xsol, ysol = ode_euler(soal_slide_no15, xi, xf, y0, Ninterval)
    plt.plot(xsol, ysol, label="h = "+str(h))
plt.plot(xsol, soal_slide_no15_analitik(xsol), label="analitik")
plt.legend()
plt.title("Menggunakan ode_euler")
```

```
Text(0.5, 1.0, 'Menggunakan ode_euler')
```




```
xi = 0.0
xf = 4.0
y0 = [1.0] # syarat awal
hh = [0.5, 0.25, 0.1]

plt.clf()
for h in hh:
    Ninterval = int( (xf - xi)/h )
    xsol, ysol = ode_RK4(soal_slide_no15, xi, xf, y0, Ninterval)
    plt.plot(xsol, ysol, label="h = "+str(h))
plt.plot(xsol, soal_slide_no15_analitik(xsol), label="analitik")
plt.legend()
plt.title("Menggunakan ode_RK4")
```

```
Text(0.5, 1.0, 'Menggunakan ode_RK4')
```



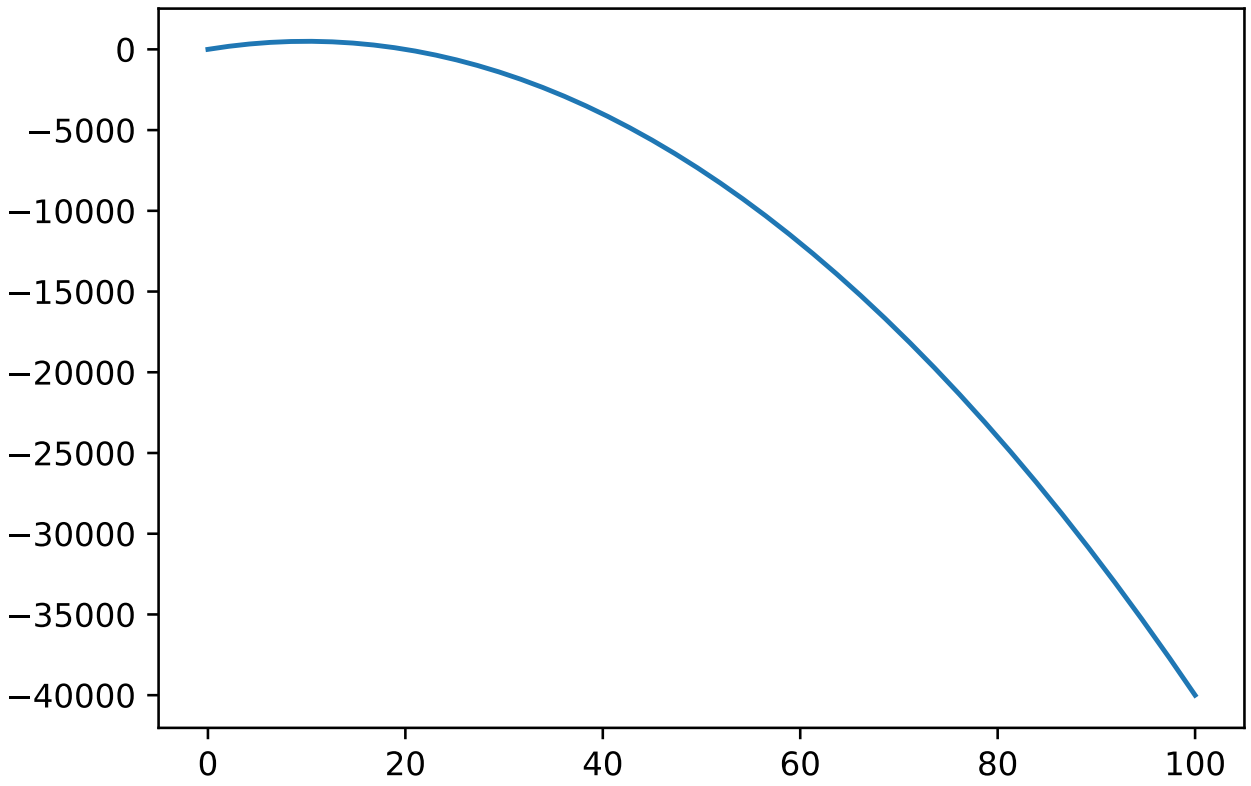

```
def soal_slide_no34(x, y):
    f = np.zeros(2)
    f[0] = y[1]
    f[1] = -10.0
    return f
```

```
h = 0.1
xi = 0.0
xf = 100.0
y0 = [0.0, 100.0]
Ninterval = int( (xf - xi)/h )
xsol, ysol = ode_RK4(soal_slide_no34, xi, xf, y0, Ninterval, verbose=True)
```

```
ode_RK4 h =       0.1000000000
```

```
plt.clf()
plt.plot(xsol, ysol[:,0])
```

```
[<matplotlib.lines.Line2D at 0x7f759aeb9160>]
```


0
−5000
−10000
−15000
−20000
−25000
−30000
−35000
−40000
0
20
40
60
80
100

Bab 7

Persamaan Diferensial Parsial

7.1 Persamaan Kalor 1d

```
import numpy as np
import matplotlib.pyplot as plt
```

Dalam satu dimensi spasial (misalkan x):

$$\alpha \frac{\partial^2}{\partial x^2} u(x,t) = \frac{\partial}{\partial t} u(x,t)$$

Domain: - spasial: $0 \leq x \leq x_f$ - temporal: $0 \leq t \leq t_f$.
Syarat batas: - $u(0,t) = b_0(t)$ - $u(x_f,t) = b_{x_f}(t)$
Syarat awal: - $u(x,0) = u_0(x)$
Catatan: persamaan yang sama juga digunakan untuk menjelaskan fenomena difusi.

7.1.1 Metode Euler Eksplisit

Domain spasial dibagi menjadi N_x segmen dengan $\Delta x = x_f/N_x$.
Domain temporal dibagi menjadi N_t segmen dengan $\Delta t = t_f/N_t$.
Turunan parsial kedua terhadap x diaproksimasi dengan menggunakan central difference.
Turunan parsial pertama terhadap t diaproksimasi dengan forward difference.
Dengan menggunakan notasi berikut:

- $u(x,t) = u_i^k$
- $u(x+\Delta x,t) = u_{i+1}^k$
- $u(x-\Delta x,t) = u_{i-1}^k$
- $u(x,t+\Delta t) = u_i^{k+1}$
- $u(x,t-\Delta t) = u_i^{k-1}$

dapat dituliskan:

$$\alpha \frac{u_{i+1}^k - 2u_i^k + u_{i-1}^k}{(\Delta x)^2} = \frac{u_i^{k+1} - u_i^k}{\Delta t}$$

Dengan menggunakan defisi:

$$r = \alpha \frac{\Delta t}{(\Delta x)^2}$$

Persamaan ini dapat dipecahkan untuk mendapatkan u_i^{k+1}

$$u_i^{k+1} = r\left(u_{i+1}^k + u_{i-1}^k\right) + (1-2r)u_i^k$$

untuk $i = 1, 2, \ldots, N_x - 1$.

Dapat ditunjukkan bahwa skema ini akan stabil jika:

$$r = \alpha \frac{\Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

Implementasi

```
def heat_1d_euler_exp( alpha, xf, tf, u0x, bx0, bxf, Nx, Nt ):
    dx = xf/Nx
    x = np.linspace(0.0, xf, Nx+1)
    dt = tf/Nt
    t = np.linspace(0.0, tf, Nt+1)
    u = np.zeros( (Nx+1, Nt+1) )
    # Aplikasi syarat awal
    for i in range(Nx+1):
        u[i,0] = u0x( x[i] )
    # Syarat batas
    for k in range(Nt+1):
        u[0,k] = bx0( t[k] )
        u[Nx,k] = bxf( t[k] )
    r = alpha*dt/dx**2
    if r > 0.5:
        print("heat_1d_euler_exp:")
        print("WARNING: r lebih besar dari 0.5: %f\n" % r)
        print("WARNING: solusi tidak stabil !!")
    else:
        print("heat_1d_euler_exp:")
        print("r = %f >= 0.5\n" % r)
        print("Solusi seharusnya stabil")
    for k in range(0,Nt):
        for i in range(1,Nx):
            u[i,k+1] = r*( u[i+1,k] + u[i-1,k] ) + (1 - 2*r)*u[i,k]

    return u, x, t
```

Contoh

Cari solusi numerik persamaan kalor:

$$\frac{\partial^2}{\partial x^2} u(x,t) = \frac{\partial}{\partial t} u(x,t)$$

pada domain: - spasial: $0 \leq x \leq 1$ - temporal: $0 \leq t \leq 0.1$.

Syarat batas: - $u(0,t) = 0$ - $u(1,t) = 0$

Syarat awal: - $u(x,0) = \sin(\pi x)$

Bandingkan dengan solusi analitik:

$$u(x,t) = \sin(\pi x) \exp(-\pi^2 t)$$

```
# Syarat awal
def initial_temp( x ):
    return np.sin(np.pi*x)
# Syarat batas kiri
def bx0( t ):
    return 0.0
# Syarat batas kanan
def bxf( t ):
    return 0.0
```

```
def sol_01_analitik(x,t):
    return np.sin(np.pi*x)*np.exp(-np.pi**2 * t)
```

Kasus solusi stabil

```
# Dari soal atau masalah yang diberikan
alpha = 1.0; xf = 1.0; tf = 0.1
# ditentukan pengguna
Nx = 25; Nt = 200
#
u_exp, x_exp, t_exp = \
heat_1d_euler_exp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

```
heat_1d_euler_exp:
r = 0.312500 >= 0.5

Solusi seharusnya stabil
```

Plot hasilnya untuk beberapa nilai t.

```
plt.clf()
plt.plot(x_exp, u_exp[:,0], label="t="+str(t_exp[0]))
plt.plot(x_exp, u_exp[:,4], label="t="+str(t_exp[4]))
plt.plot(x_exp, u_exp[:,-1], label="t="+str(t_exp[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c04a19b0>
```

Perbandingan dengan solusi analitik.

```
plt.clf()
plt.plot(x_exp, u_exp[:,-1], label="numerik t="+str(t_exp[-1]), marker="o")
plt.plot(x_exp, sol_01_analitik(x_exp, t_exp[-1]), label="analitik t="+str(t_exp[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c03b9fd0>
```

Animasi

```
anim = create_anim_2d(u_exp, x_exp, t_exp, 0.0, 1.1)
IPython.display.HTML( anim.to_html5_video() )
```

Alternatif, tulis ke file:

```
for k in range(Nt+1):
    plt.clf()
    plt.plot( x_exp, u_exp[:,k], label="t="+format("%.5f" % t_exp[k]))
    plt.legend();
    plt.ylim(0.0, 1.05)
    plt.savefig(format("TEMP_%04d.png" % (k)))
```

Kasus solusi tidak stabil

```
# Dari soal atau masalah yang diberikan
alpha = 1.0
xf = 1.0
tf = 0.1
# ditentukan pengguna
Nx = 50
Nt = 200

u_exp, x_exp, t_exp = \
heat_1d_euler_exp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

```
heat_1d_euler_exp:
WARNING: r lebih besar dari 0.5: 1.250000

WARNING: solusi tidak stabil !!
```

Plot untuk beberapa nilai t (paling awal dan paling akhir)

```
plt.clf()
plt.plot( x_exp, u_exp[:,0], label="t="+str(t_exp[0]))
plt.plot( x_exp, u_exp[:,-1], label="t="+str(t_exp[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c0346c50>
```

Perhatikan bahwa pada kasus ini solusi numerik yang diperoleh tidak stabil (kesalahan semakin membesar).

7.1.2 Metode Euler implisit

Domain spasial dibagi menjadi N_x segmen dengan $\Delta x = x_f/N_x$.
Domain temporal dibagi menjadi N_t segmen dengan $\Delta t = t_f/N_t$.
Turunan parsial kedua terhadap x diaproksimasi dengan menggunakan central difference.
Turunan parsial pertama terhadap t diaproksimasi dengan backward difference.

$$\alpha \frac{u_{i+1}^k - 2u_i^k + u_{i-1}^k}{(\Delta x)^2} = \frac{u_i^k - u_i^{k-1}}{\Delta t}$$

Dengan menggunakan notasi:

$$r = \alpha \frac{\Delta t}{(\Delta x)^2}$$

Diperoleh persamaan implisit:

$$-ru_{i-1}^k + (1+2r)u_i^k - ru_{i+1}^k = u_i^{k-1}$$

untuk $i = 1, 2, \ldots, N_x - 1$.
Dalam bentuk matriks:

$$\begin{bmatrix} 1+2r & -r & 0 & \cdot & 0 & 0 \\ -r & 1+2r & -r & \cdot & 0 & 0 \\ 0 & -r & 1+2r & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 1+2r & -r \\ 0 & 0 & 0 & \cdot & -r & 1+2r \end{bmatrix} \begin{bmatrix} u_1^k \\ u_2^k \\ u_3^k \\ \cdots \\ u_{N_x-2}^k \\ u_{N_x-1}^k \end{bmatrix} = \begin{bmatrix} u_1^{k-1} + ru_0^k \\ u_2^{k-1} \\ u_3^{k-1} \\ \cdots \\ u_{N_x-2}^{k-1} \\ u_{N_x-1}^{k-1} + ru_{N_x}^k \end{bmatrix}$$

Implementasi
Dalam kode di bawah ini akan didefinisikan matriks **A** sebagai matriks koefisien pada ruas kiri dan vektor **b** pada ruas kanan.

```
def heat_1d_euler_imp( alpha, xf, tf, u0x, bx0, bxf, Nx, Nt ):

    dx = xf/Nx
    x = np.linspace(0.0, xf, Nx+1)

    dt = tf/Nt
    t = np.linspace(0.0, tf, Nt+1)

    u = np.zeros( (Nx+1, Nt+1) )

    # Aplikasi syarat awal
    for i in range(Nx+1):
        u[i,0] = u0x( x[i] )

    # Syarat batas
    for k in range(Nt+1):
```

```
            u[0,k] = bx0( t[k] )
            u[Nx,k] = bxf( t[k] )

        r = alpha*dt/dx**2

        # Bangun matriks A
        A = np.zeros( (Nx-1,Nx-1) )
        for i in range(Nx-1):
            A[i,i] = 1 + 2*r
            if i > 0:
                A[i-1,i] = -r
                A[i,i-1] = -r

        # Bangun vektor b
        b = np.zeros(Nx-1)
        for k in range(1,Nt+1):
            b = np.copy(u[1:Nx,k-1])
            b[0] = b[0] + r*u[0,k]
            b[Nx-2] = b[Nx-2] + r*u[Nx,k]
            # Selesaikan sistem persamaan linear
            u[1:Nx,k] = np.linalg.solve(A, b)

        return u, x, t
```

Contoh

Untuk parameter-parameter berikut, metode eksplisit stabil.

```
alpha = 1.0
xf = 1.0; tf = 0.1
Nx = 25; Nt = 200
u_imp, x_imp, t_imp = \
heat_1d_euler_imp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

Plot hasilnya untuk beberapa nilai t.

```
plt.clf()
plt.plot(x_imp, u_imp[:,0], label="t="+str(t_imp[0]))
plt.plot(x_imp, u_imp[:,4], label="t="+str(t_imp[4]))
plt.plot(x_imp, u_imp[:,-1], label="t="+str(t_imp[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c02d68d0>
```

Untuk parameter-parameter berikut ini, metode eksplisit tidak stabil.

Bagaimana untuk metode implisit?

```
alpha = 1.0
xf = 1.0; tf = 0.1
Nx = 50; Nt = 200
u_imp, x_imp, t_imp = \
heat_1d_euler_imp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

Plot hasilnya untuk beberapa nilai t.

```
plt.clf()
plt.plot(x_imp, u_imp[:,0], label="t="+str(t_imp[0]))
plt.plot(x_imp, u_imp[:,4], label="t="+str(t_exp[4]))
plt.plot(x_imp, u_imp[:,-1], label="t="+str(t_imp[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c0250cf8>
```

Dapat dilihat bahwa solusi numerik yang dihasikan oleh metode implisit bersifat stabil. Dapat ditunjukkan dengan menggunakan analisis numerik bahwa metode implisit bersifat stabil tanpa syarat.

Sekarang akan kita coba mencari solusi untuk nilai t_f yang lebih besar.

```
alpha = 1.0

xf = 1.0
tf = 0.2

Nx = 50
Nt = 200

u_imp, x_imp, t_imp = \
heat_1d_euler_imp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

Plot hasilnya untuk beberapa nilai t.

```
plt.clf()
plt.plot(x_imp, u_imp[:,0], label="t="+str(t_imp[0]))
plt.plot(x_imp, u_imp[:,100], label="t="+str(t_imp[100]))
plt.plot(x_imp, u_imp[:,-1], label="t="+str(t_imp[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c036c588>
```

```
anim = create_anim_2d(u_imp, x_imp, t_imp, 0.0, 1.1)
IPython.display.HTML(anim.to_html5_video())
```

Bagaimana jika kita menggunakan metode eksplisit ?

```
alpha = 1.0
xf = 1.0; tf = 0.2
Nx = 50; Nt = 200
u_exp, x_exp, t_exp = \
heat_1d_euler_exp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

```
heat_1d_euler_exp:
WARNING: r lebih besar dari 0.5: 2.500000

WARNING: solusi tidak stabil !!
```

```
plt.clf()
plt.plot(x_exp, u_exp[:,0], label="t="+str(t_exp[0]))
plt.plot(x_exp, u_exp[:,10], label="t="+str(t_exp[10]))
plt.plot(x_exp, u_exp[:,15], label="t="+str(t_exp[15]))
plt.plot(x_exp, u_exp[:,16], label="t="+str(t_exp[16]))
plt.ylim(0.0, 1.0)
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c24cfd68>
```

Dapat dilihat bahwa solusi ini tidak stabil.

```
anim = create_anim_2d(u_exp, x_exp, t_exp, 0.0, 1.1)
IPython.display.HTML(anim.to_html5_video())
```

7.1.3 Metode Crank-Nicholson

Metode Crank-Nicholson diperoleh dengan menggunakan rata-rata aproksimasi central difference antara titik waktu $k+1$ dan k sehingga diperoleh:

$$\frac{\alpha}{2}\left(\frac{u_{i+1}^{k+1}-2u_i^{k+1}+u_{i-1}^{k+1}}{(\Delta x)^2}+\frac{u_{i+1}^{k}-2u_i^{k}+u_{i-1}^{k}}{(\Delta x)^2}\right)=\frac{u_i^{k+1}-u_i^{k}}{\Delta t}$$

atau:

$$ru_{i+1}^{k+1} - 2ru_i^{k+1} + ru_{i-1}^{k+1} + ru_{i+1}^k - 2ru_i^k + ru_{i-1}^k = 2u_i^{k+1} - 2u_i^k$$

$$-ru_{i+1}^{k+1} + 2(1+r)u_i^{k+1} - ru_{i-1}^{k+1} = ru_{i+1}^k + 2(1-r)u_i^k + ru_{i-1}^k$$

Dalam bentuk matriks:

$$\mathbf{A}\mathbf{u}^{k+1} = \mathbf{B}\mathbf{u}^k$$

dengan matriks sebagai berikut.

$$\mathbf{A} = \begin{bmatrix} 2(1+r) & -r & 0 & \cdots & 0 & 0 \\ -r & 2(1+r) & -r & \cdots & 0 & 0 \\ 0 & -r & 2(1+r) & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 2(1+r) & -r \\ 0 & 0 & 0 & \cdots & -r & 2(1+r) \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 2(1-r) & r & 0 & \cdots & 0 & 0 \\ r & 2(1-r) & r & \cdots & 0 & 0 \\ 0 & r & 2(1-r) & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdot & 2(1-r) & r \\ 0 & 0 & 0 & \cdot & r & 2(1-r) \end{bmatrix}$$

$$\mathbf{u}^k = \begin{bmatrix} u_1^k \\ u_2^k \\ u_3^k \\ \cdot \\ u_{M-1}^k \\ u_M^k \end{bmatrix}$$

Implementasi

```
def heat_1d_CN( alpha, xf, tf, u0x, bx0, bxf, Nx, Nt ):

    dx = xf/Nx
    x = np.linspace(0.0, xf, Nx+1)

    dt = tf/Nt
    t = np.linspace(0.0, tf, Nt+1)

    u = np.zeros( (Nx+1, Nt+1) )

    # Aplikasi syarat awal
    for i in range(Nx+1):
        u[i,0] = u0x( x[i] )

    # Syarat batas
    for k in range(Nt+1):
        u[0,k] = bx0( t[k] )
        u[Nx,k] = bxf( t[k] )

    r = alpha*dt/dx**2

    A = np.zeros( (Nx-1,Nx-1) )
    for i in range(Nx-1):
        A[i,i] = 2*(1 + r)
        if i > 0:
            A[i-1,i] = -r
            A[i,i-1] = -r

    B = np.zeros( (Nx-1,Nx-1) )
    for i in range(Nx-1):
        B[i,i] = 2*(1 - r)
        if i > 0:
```

```
            B[i-1,i] = r
            B[i,i-1] = r

    for k in range(1,Nt+1):
        b = np.matmul(B, u[1:Nx,k-1] )
        u[1:Nx,k] = np.linalg.solve(A, b)

    return u, x, t
```

Contoh

```
alpha = 1.0
xf = 1.0; tf = 0.2
Nx = 50; Nt = 200
u_CN, x_CN, t_CN = \
heat_1d_CN( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

```
plt.clf()
plt.plot(x_CN, u_CN[:,0], label="t="+str(t_CN[0]))
plt.plot(x_CN, u_CN[:,4], label="t="+str(t_CN[4]))
plt.plot(x_CN, u_CN[:,10], label="t="+str(t_CN[10]))
plt.plot(x_CN, u_CN[:,50], label="t="+str(t_CN[50]))
plt.plot(x_CN, u_CN[:,-1], label="t="+str(t_CN[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c01d7cf8>
```

Perbandingan antara metode implisit dan Crank-Nicholson pada waktu terakhir simulasi t_f.

```
plt.clf()
plt.plot(x_imp, u_imp[:,-1], label="imp t="+str(t_imp[-1]))
plt.plot(x_CN, u_CN[:,-1], label="CN t="+str(t_CN[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c015d710>
```

Perbandingan antara solusi numerik dengan Crank-Nicholson dengan solusi analitik.

```
plt.clf()
plt.plot(x_CN, u_CN[:,-1], label="numerik t="+str(t_CN[-1]), marker="o")
plt.plot(x_CN, sol_01_analitik(x_CN, t_CN[-1]), label="analitik t="+str(t_CN[-1]))
plt.legend();
```

```
<matplotlib.legend.Legend at 0x7fb9c00d8ac8>
```

Latihan 1

Mirip dengan soal sebelumnya, hanya saja dengan syarat awal:

$$u(x,0) = e^{-50(x-0.5)^2}$$

Kita akan gunakan metode implisit untuk mencari solusi numeriknya.

```
# Syarat awal
def initial_temp( x ):
    return np.exp( -50*(x-0.5)**2 )

# Syarat batas kiri
def bx0( t ):
    return 0.0

# Syarat batas kanan
def bxf( t ):
    return 0.0
```

```
alpha = 1.0

xf = 1.0
Nx = 50

tf = 0.1
Nt = 200

u_imp, x_imp, t_imp = \
heat_1d_euler_imp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )
```

```
plt.clf()
plt.plot( x_imp, u_imp[:,0], label="t="+str(t_imp[0]) )
plt.plot( x_imp, u_imp[:,-1], label="t="+str(t_imp[-1]) )
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb9bffb5f98>
```

Latihan 2

```
# Syarat awal
def initial_temp( x ):
    if x > 0.4 and x < 0.6:
        return 1.0
    else:
        return 0.0

# Syarat batas kiri
def bx0( t ):
    return 0.0

# Syarat batas kanan
def bxf( t ):
    return 0.0

alpha = 1.0
xf = 1.0; Nx = 50
tf = 0.1; Nt = 200
u_imp, x_imp, t_imp = \
heat_1d_euler_imp( alpha, xf, tf, initial_temp, bx0, bxf, Nx, Nt )

plt.clf()
plt.plot( x_imp, u_imp[:,0], label="t="+str(t_imp[0]) )
plt.plot( x_imp, u_imp[:,-1], label="t="+str(t_imp[-1]) )
plt.legend()
plt.savefig("figures/heat_imp_lat2.pdf")
```