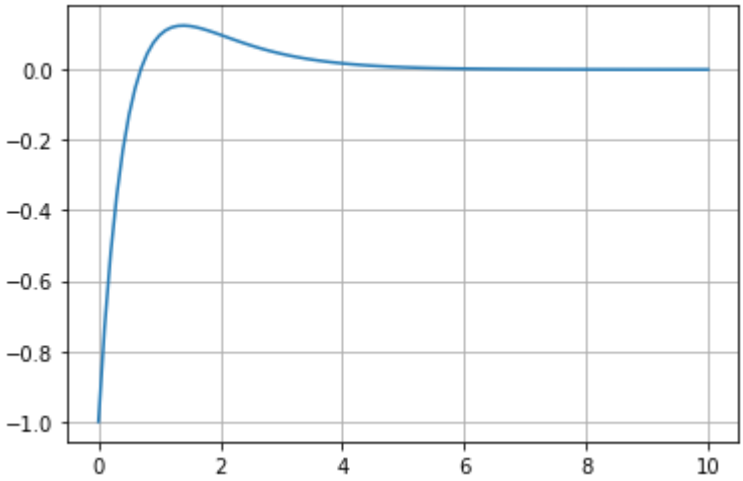
数値計算試験問題

2025/07/09 実施

2分法, Newton-Raphson法の収束性：25点

以下の関数$f(x)$の解を区間$x = 0..1$において 2分法とNewton-Raphson法で求めよ. また, それぞれの収束挙動をlogで同時プロットし比較せよ.

$$f(x) = \mathrm{e}^{-x} - 2\,\mathrm{e}^{-2x}$$



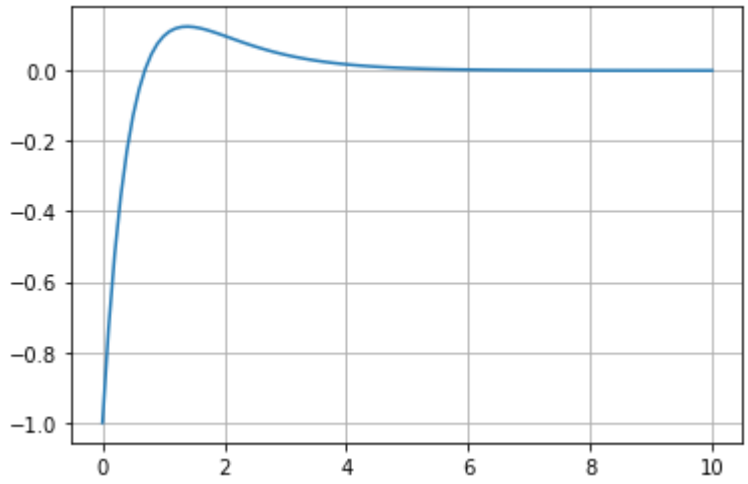
In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})

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

def df(x):
    return -np.exp(-x)+4*np.exp(-2*x)

x1=0.0
x2=10.0
x = np.linspace(x1,x2,101)
y = func(x)
plt.plot(x,y)
plt.grid()
plt.show()
```



In [2]:
```
from scipy.optimize import fsolve
x0 = fsolve(func, 0.0)[0]
x0
```

Out[2]:
```
0.6931471805599453
```

In [3]:
```
x1=0.0
x2=1.0
f1,f2=func(x1),func(x2)
list_bisec=[[0],[abs(x1-x0)]]
for i in range(1,20):
    x=(x1+x2)/2
    f=func(x)
    list_bisec[0].append(i)
    list_bisec[1].append(abs(x-x0))
    if (f*f1>=0.0):
        x1,f1=x,f
    else:
        x2,f2=x,f
    print(x1,x2,f1,f2)

print(list_bisec)
```

```
0.5 1.0 -0.12922822263025124 0.09720887469821693
0.5 0.75 -0.12922822263025124 0.026106232444155053
0.625 0.75 -0.037748165201389905 0.026106232444155053
0.6875 0.75 -0.0028476136385520157 0.026106232444155053
0.6875 0.71875 -0.0028476136385520157 0.012319438522702841
0.6875 0.703125 -0.0028476136385520157 0.004914818435146795
0.6875 0.6953125 -0.0028476136385520157 0.0010791491791509178
0.69140625 0.6953125 -0.0008727414902032216 0.0010791491791509178
0.69140625 0.693359375 -0.0008727414902032216 0.00010606345573976883
0.6923828125 0.693359375 -0.00038262248448772684 0.00010606345573976883
0.69287109375 0.693359375 -0.0001381005851953665 0.00010606345573976883
0.693115234375 0.693359375 -1.5973857910744904e-05 0.00010606345573976883
0.693115234375 0.6932373046875 -1.5973857910744904e-05 4.505597243553705e-05
0.693115234375 0.69317626953125 -1.5973857910744904e-05 1.4543851040493827e-05
0.693145751953125 0.69317626953125 -7.143049408631086e-07 1.4543851040493827e-05
0.693145751953125 0.6931610107421875 -7.143049408631086e-07 6.914947667135962e-06
0.693145751953125 0.6931533813476562 -7.143049408631086e-07 3.1003650182714892e-06
0.693145751953125 0.6931495666503906 -7.143049408631086e-07 1.1930409525851005e-06
0.693145751953125 0.6931476593017578 -7.143049408631086e-07 2.3937073434510125e-07
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19], [0.6931471805599453,
0.1931471805599453, 0.056852819440054714, 0.06814718055994529, 0.005647180559945286, 0.025602
819440054714, 0.009977819440054714, 0.0021653194400547138, 0.0017409305599452862, 0.000212194
44005471377, 0.0007643680599452862, 0.0002760868099452862, 3.194618494528623e-05, 9.012412755
471377e-05, 2.9088971304713773e-05, 1.4286068202862268e-06, 1.3830182242213773e-05, 6.2007877
10963773e-06, 2.3860904453387732e-06, 4.787418125262732e-07]]
```

In [4]:
```
x1=1.0
f1=func(x1)
```

```
list_newton=[[0],[x1]]
for i in range(1,20):
    x1 = x1-f1/df(x1)
    f1 =func(x1)
    print(x1,f1)
    list_newton[0].append(i)
    list_newton[1].append(abs(x1-x0))

list_newton
```

```
0.43959456578860423 -0.18594117646945774
0.6225751274995484 -0.03923434945945836
0.6863677428631368 -0.0034243718517862343
0.6930789079672248 -3.413979240607379e-05
0.6931471735689144 -3.495515543683325e-09
0.6931471805599454 5.551115123125783e-17
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
0.6931471805599453 0.0
```

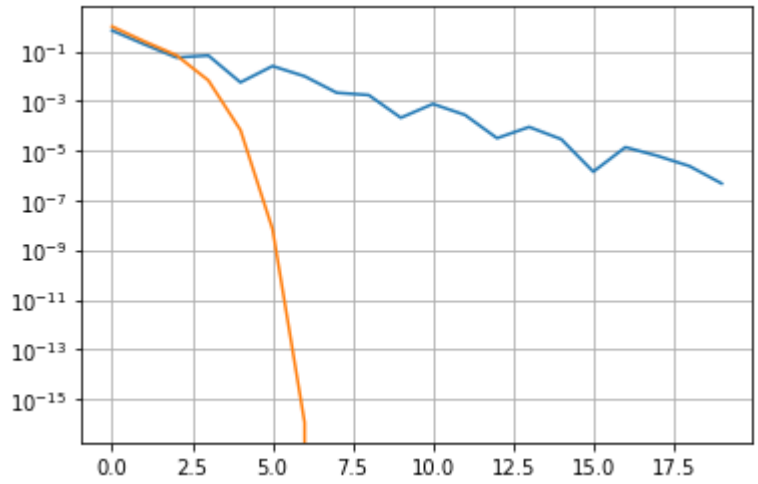
Out[4]:
```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [1.0,
  0.25355261477134106,
  0.07057205306039693,
  0.0067794376968084435,
  6.827259272046415e-05,
  6.9910308653220454e-09,
  1.1102230246251565e-16,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0]]
```

In [5]:
```
import matplotlib.pyplot as plot

X=list_bisec[0]
Y=list_bisec[1]
plt.plot(X,Y)

X=list_newton[0]
Y=list_newton[1]
plt.plot(X,Y)

plt.yscale("log")
plt.grid()
plt.show()
```



グラム・シュミットの直交化法とQR分解：25点

グラム・シュミットの直交化法

与えられたベクトルから正規直交基底を作るアルゴリズムとして グラム・シュミットの直交化法がある.

その手順は，ベクトル$v_1, \ldots, v_n$が与えられたとすると

1. $u_1 = v_1$
2. $u_2 = v_2 - \frac{u_1^{\mathrm{T}} v_2}{u_1^{\mathrm{T}} u_1} u_1$
3. $u_n = v_n - \frac{u_1^{\mathrm{T}} v_n}{u_1^{\mathrm{T}} u_1} u_1 - \frac{u_2^{\mathrm{T}} v_n}{u_2^{\mathrm{T}} u_2} u_2 - \cdots - \frac{u_{n-1}^{\mathrm{T}} v_n}{u_{n-1}^{\mathrm{T}} u_{n-1}} u_{n-1} = v_n - \sum_{i=1}^{n-1} \frac{u_i^{\mathrm{T}} v_n}{u_i^{\mathrm{T}} u_i} u_i$

で互いに直交した基底を求め，これらを

$$e_i = \frac{u_i}{(u_i^{\mathrm{T}} u_i)^{1/2}}$$

によって正規化すれば，$\{e_1, e_2, \ldots, e_n\}$ が正規直交基底となる.

次の$\mathbb{R}^4$ベクトルから正規直交基底を作れ.

$$v_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \ v_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \ v_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

In [6]:
```
import numpy as np
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
v1=np.array([1,1,0,0])
v2=np.array([0,1,1,0])
v3=np.array([0,0,1,1])
u1=v1
u2=v2-np.dot(u1,v2)/np.dot(u1,u1)*u1
u3=v3-np.dot(u1,v3)/np.dot(u1,u1)*u1--np.dot(u2,v3)/np.dot(u2,u2)*u2
print(u1)
```

```
print(u2)
print(u3)
e1=u1/np.linalg.norm(u1)
e2=u2/np.linalg.norm(u2)
e3=u3/np.linalg.norm(u3)
print(e1,e2,e3)
```

```
[1 1 0 0]
[-0.500  0.500  1.000  0.000]
[-0.333  0.333  1.667  1.000]
[ 0.707  0.707  0.000  0.000] [-0.408  0.408  0.816  0.000] [-0.167  0.167  0.833  0.500]
```

In [7]:
```
def gram_schmidt(vectors):
    basis = []
    for v in vectors:
        w = v - sum(np.dot(v, b)*b for b in basis)
        if (w > 1e-10).any():
            basis.append(w / np.linalg.norm(w))
    return np.array(basis)

gram_schmidt([v1,v2,v3])

# https://aikostudio.hatenablog.com/entry/2023/08/28/074308#google_vignette
# PythonによるGram-Schmidt正規直交化プロセス
```

Out[7]:
```
array([[ 0.707,  0.707,  0.000,  0.000],
       [-0.408,  0.408,  0.816,  0.000],
       [ 0.289, -0.289,  0.289,  0.866]])
```

QR分解

次の行列AのQR分解を求めよ.

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Gram-Schmidtで求めた正規直交基底と比べよ.

In [8]:
```
import numpy as np
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
A=np.transpose(np.array([[1,1,0,0],[0,1,1,0],[0,0,1,1]]))
print(A)
Q,R = np.linalg.qr(A)
print(Q)
print(R)
```

```
[[1 0 0]
 [1 1 0]
 [0 1 1]
 [0 0 1]]
[[-0.707  0.408 -0.289]
 [-0.707 -0.408  0.289]
 [-0.000 -0.816 -0.289]
 [-0.000 -0.000 -0.866]]
[[-1.414 -0.707  0.000]
 [ 0.000 -1.225 -0.816]
 [ 0.000  0.000 -1.155]]
```

常微分方程式：25点

フォーミュラEと呼ばれるEV車(電気自動車)のレースでは，停止時から毎時100キロメートルまでの加速時間は1.86秒となる．これは，普通のF1よりも3割速いらしい．日経(2025/6/4)

1. 空気抵抗などの規格化した抵抗が速度の0.5として，この加速性能を達成するためには何G程度の推力が必要となるか小数点以下二桁程度で求めよ．
2. F1で使われる車両とフォーミュラEで使われる車両の重量はほぼ同じである．抵抗も同じとして，クラッチロスなどがないと仮定して，F1車の推力を求めよ．
3. 結果を検討せよ．

注: 例えば「2Gの推力」とは機体(普通はロケット)の自重の2倍の力で推進していることを表し，加速度は重力加速度(9.8m/sec^2)の2倍になるとします．

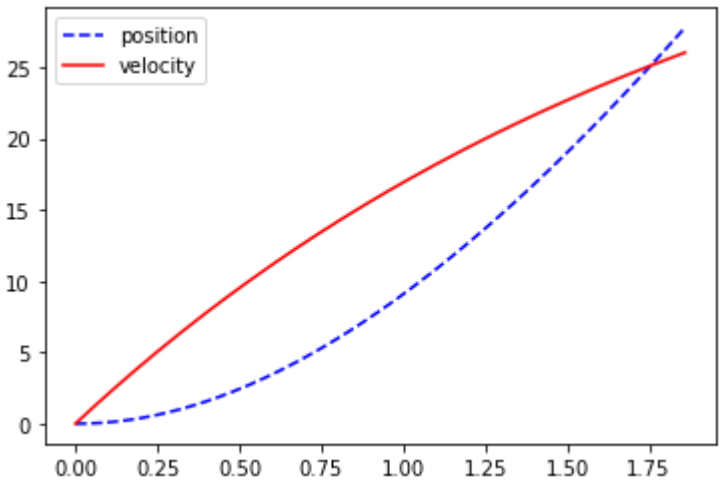
In [9]:
```
import matplotlib.pyplot as plt

def my_plot(xx,vv,tt):
    plt.plot(tt,xx,color='b',linestyle='--',label='position')
    plt.plot(tt,vv,color='r',label='velocity')
    plt.legend()
    plt.show()

def euler2(x0,v0):
    v1=v0+(-cc*v0+g)*dt
    x1=x0+v0*dt
    return [x1,v1]
```

In [10]:
```
g, dt, cc=2.19*9.8,0.01,0.5
tt,xx,vv=[0.0],[0.0],[0.0]
t=0.0
for i in range(0,186):
    t += dt
    x,v=euler2(xx[-1],vv[-1])
    tt.append(t)
    xx.append(x)
    vv.append(v)
print(xx[-1])
my_plot(xx,vv,tt)
```

```
27.783363945996378
```



In [11]:
```
100*10**3/60/60
```

Out[11]:
```
27.77777777777778
```

In [12]:
```
g, dt, cc=1.41*9.8,0.01,0.5
tt,xx,vv=[0.0],[0.0],[0.0]
t=0.0
for i in range(0,int(186*1.3)):
    t += dt
    x,v=euler2(xx[-1],vv[-1])
    tt.append(t)
    xx.append(x)
    vv.append(v)
print(xx[-1])
#my_plot(xx,vv,tt)
```

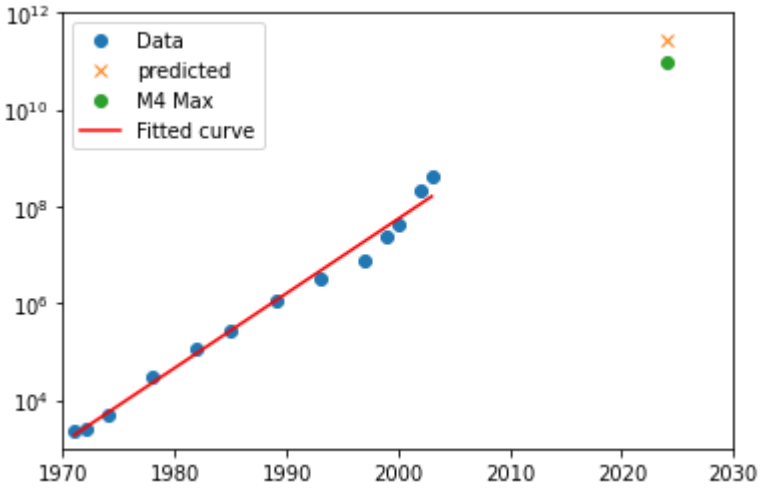
```
27.84534403977935
```

In [13]:
```
2.19/1.41
```

Out[13]:
```
1.5531914893617023
```

ムーアの法則：25点

次の図の青点は13種類のマイクロプロセッサの発売年と，それぞれに含まれるトランジスタ数Nを対数スケールで表示している．



それぞれのデータは次の配列で与えられる．

In [14]:
```
import numpy as np

list_year=[1971,1972,1974,1978,1982,1985,1989,1993,1997,1999,2000,2002,2003]
list_cpu_num=[2250,2500,5000,29000,120000,275000,1180000,3100000,7500000,
         24000000,42000000,220000000,410000000]
```

このデータに対して，以下のモデルを用いて最小二乗法による直線当てはめを求めよ．

$$\log_{10} N = \theta_1 + \theta_2 (t - 1970)$$

ここで，tは年，Nはトランジスタ数である． なお，θ_1 はモデルが予測する1970年におけるトランジスタ数の対数，10^{θ_2}はモデルが予測する毎年のトランジスタ数の増加率である．

1. そのモデルを用いて2024年のマイクロプロセッサ中のトランジスタ数を予測せよ．
2. Apple M4 Max(2024)のトランジスタ数(約100億，100×10^9)と比較せよ．

この予測は「集積回路のトランジスタ数は1年半から2年でおよそ2倍になる」というムーアの法則を満たしている.

In [15]:
```
array1=np.ones(13)
array2=np.array(list_year)-1970
AA=np.column_stack((array1, array2))
```

In [16]:
```
def logarithmic(x):
    return np.log10(x)

bb=np.vectorize(logarithmic)(list_cpu_num)
```

逆行列の求め方のヒント

最小二乗法のデザイン行列(AA)の逆行列の求め方にはいろいろあるがどれもこの問題では一致する.　どれか好きなのを使ってください.

正規方程式(Normal Equations)による解

テキストで紹介している方法.

$$AA^{-1} = (AA^{\mathrm{T}}.AA)^{-1}$$

ただし, $Ax = b$の解xは,

$$x = AA^{-1}(AA^{\mathrm{T}}b)$$

In [17]:
```
AA_inv=np.linalg.inv(np.dot(np.transpose(AA),AA))
```

In [18]:
```
np.dot(AA_inv, np.dot(np.transpose(AA),bb))
```

Out[18]:
```
array([ 3.126,  0.154])
```

QR分解による解

先ほどのQR分解を用いると以下の通り求められる.

$$AA = Q.R$$
$$AA^{-1} = R^{-1}Q^{\mathrm{T}}$$
$$x = AA^{-1}.b$$

In [19]:
```
Q,R = np.linalg.qr(AA)
print(Q)
print(R)
```

```
[[-0.277 -0.417]
 [-0.277 -0.392]
 [-0.277 -0.344]
 [-0.277 -0.246]
 [-0.277 -0.148]
 [-0.277 -0.075]
 [-0.277  0.023]
 [-0.277  0.120]
 [-0.277  0.218]
 [-0.277  0.267]
 [-0.277  0.291]
 [-0.277  0.340]
 [-0.277  0.364]]
```

```
[[-3.606 -65.177]
 [ 0.000  40.975]]
```

In [20]:
```
AA_inv=np.dot(np.linalg.inv(R), np.transpose(Q))
```

In [21]:
```
np.dot(AA_inv, bb)
```

Out[21]:
```
array([ 3.126,  0.154])
```

pseudo inverseによる解

linalgには優決定方程式(縦長行列)の (Moore-Penrose) 擬似逆行列(pseudo-inverse of a matrix)を計算する関数としてpinvが用意されている．内部ではSVD(singular value decomposition)を使っている．

```
pseudo_inv = np.linalg.pinv(AA)
```

で求めたpseudo_invをbに左から掛けると求められる．ん？そうか．

In [22]:
```
pseudo_inv = np.linalg.pinv(AA)
theta = pseudo_inv.dot(bb)
print(theta)
```

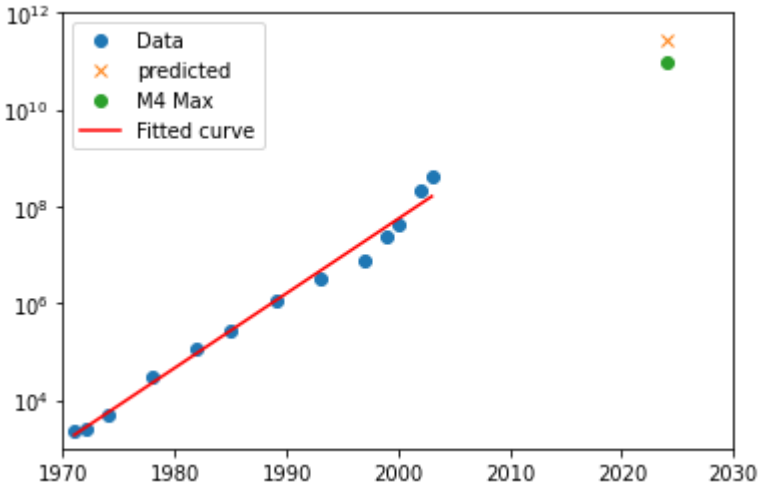
```
[ 3.126  0.154]
```

In [23]:
```
def fit_func(x, theta1, theta2):
    return 10**(theta1+theta2*x)

y_fit = np.vectorize(fit_func)(array2, *theta)

import matplotlib.pyplot as plt
fig, ax=plt.subplots()
ax.plot(list_year, list_cpu_num, 'o', label='Data')
ax.plot([2024], [277060328872.69775], 'x', label='predicted')
ax.plot([2024], [100*10**9], 'o', label='M4 Max')
ax.plot(list_year,y_fit,  'r-', label='Fitted curve')
ax.set_yscale("log")
ax.axis([1970,2030,10**3,10**12])
ax.legend(loc='upper left')
```

Out[23]:
```
<matplotlib.legend.Legend at 0x7fd8b87a5ee0>
```



In [24]:
```
fit_func(2024-1970, *theta)
```

Out[24]:
```
277060328872.69775
```

In [25]:
```
(100*10**9)/fit_func(2024-1970, *theta)
```

Out[25]:
```
0.3609322215377413
```

$10^{\theta_2 x} = 2$を解く.

In [26]:
```
np.log10(2)/0.154
```

Out[26]:
```
1.9547402315842934
```

Apple M4 Max(最新のTSMC 3nmで作られたプロセッサー)においても，外挿値の0.4倍程度であり，2025年時点でもトランジスタ数は予測通りに増加している.

In [ ]:
```

```