数値計算試験問題

2022/12/23 実施

1 簡単な行列計算(25点)

関数

$$f(x) = \frac{4}{1+x^2}$$

を多項式

$$F(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

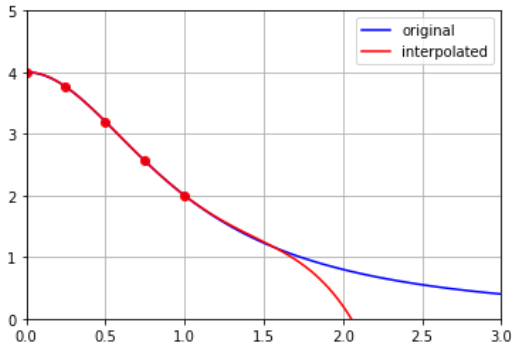
で補間することを試みる． 与関数のx=[0, 0.25, 0.5, 0.75, 1.0] での値から直接逆行列から多項式補間する手法を試す． 連立方程式の係数行列$\mathbf{A}$(ヴァンデルモンド行列と呼ばれる)およびデータベクトル$\mathbf{y}$は

```
A = [[1.         0.         0.         0.         0.        ]
 [1.         0.25       0.0625     0.015625   0.00390625]
 [1.         0.5        0.25       0.125      0.0625    ]
 [1.         0.75       0.5625     0.421875   0.31640625]
 [1.         1.         1.         1.         1.        ]]
y = [4.         3.76470588 3.2        2.56       2.        ]
```

となる．ヴァンデルモンド行列$\mathbf{A}$の逆行列をデータベクトル$\mathbf{y}$に掛けることで， 係数の値を求めよ．

以下は$\mathbf{A}$，$\mathbf{y}$を求めるコードと，与関数，補間関数のプロットである．

```
import scipy.linalg as linalg   # SciPy Linear Algebra Library
import numpy as np

nn = 5 # x_i number

def func(x):
    return 4.0/(1+x**2)
xx = []
yy = []
for x in np.linspace(0,1,nn,endpoint=True):
    xx.append(x)
    yy.append(func(x))

print(xx)
print(yy)

a_matrix = []
y_vector = []
for i in range(nn):
    for j in range(nn):
        a_matrix.append(xx[i]**j)
    y_vector.append(yy[i])

A = np.array(a_matrix).reshape(nn,nn)
y = np.array(y_vector)
print(A)
print(y)

inv_A = linalg.inv(A)
print(np.dot(inv_A,y))
```

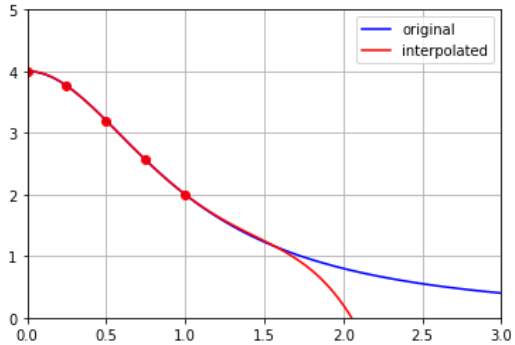


In [9]:

```
import scipy.linalg as linalg   # SciPy Linear Algebra Library
import numpy as np

nn = 5 # x_i number

def func(x):
   return 4.0/(1+x**2)
xx = []
yy = []
for x in np.linspace(0,1,nn,endpoint=True):
   xx.append(x)
   yy.append(func(x))

print(xx)
print(yy)

a_matrix = []
y_vector = []
for i in range(nn):
   for j in range(nn):
      a_matrix.append(xx[i]**j)
   y_vector.append(yy[i])

A = np.array(a_matrix).reshape(nn,nn)
y = np.array(y_vector)
print(A)
print(y)

inv_A = linalg.inv(A)
print(np.dot(inv_A,y))
```

```
[0.0, 0.25, 0.5, 0.75, 1.0]
[4.0, 3.764705882352941, 3.2, 2.56, 2.0]
[[1.       0.       0.       0.       0.      ]
 [1.       0.25     0.0625   0.015625 0.00390625]
 [1.       0.5      0.25     0.125    0.0625  ]
 [1.       0.75     0.5625   0.421875 0.31640625]
 [1.       1.       1.       1.       1.      ]]
[4.       3.76470588 3.2      2.56     2.      ]
[ 4.        0.15529412 -5.39294118  4.29176471 -1.05411765]
```

In [10]:
```
%matplotlib inline

import matplotlib.pyplot as plt
from scipy import interpolate

for i in range(0,5):
   plt.plot(xx[i],yy[i],'o',color='r')

# Lagrange補間
f = interpolate.lagrange(xx,yy)
# print(f)
x = np.linspace(0,3, 100)
y0=func(x)
y1 = f(x)
plt.plot(x, y0, color = 'b', label="original")
plt.plot(x, y1, color = 'r', label="interpolated")
plt.xlim(0,3)
plt.ylim(0,5)
plt.legend()
plt.grid()
plt.show()
```



2 誤差・精度(25点)

最初に紛れ込んだ丸め誤差が次第に拡大されて，最後には真の値と全く違う値となる 「不安定な」例として「Numeriacl Recipes in C」で紹介されている漸化式の誤差を検討する．

次のような，いわゆる黄金比

$$\phi = \frac{\sqrt{5}-1}{2} = 0.61803399$$

の累乗計算を考える．素直に累乗(power)で求めた場合と，

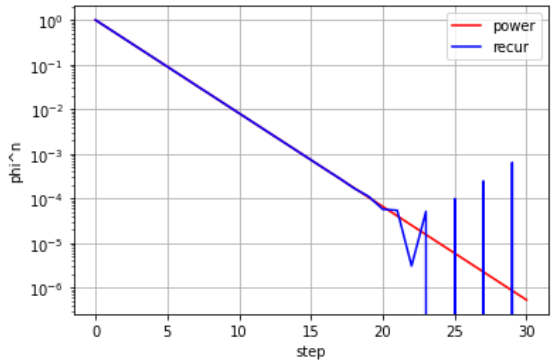
$$\phi^{n+1} = \phi^{n-1} - \phi^n \tag{1}$$
$$\phi^0 = 1.0 \tag{2}$$
$$\phi^1 = 0.61803398 \tag{3}$$

という漸化式(recurrence formula)で求めた場合とで，数値を%20.15fで出力して， それらの誤差をn=30程度までで議論せよ．

以下は，それぞれのリスト(phi_power, phi_recur)を片対数でプロットした結果である．

「Numeriacl Recipes in C, C言語による数値計算のレシピ」, W.H.Press他(技術評論社, 1993), p.44.

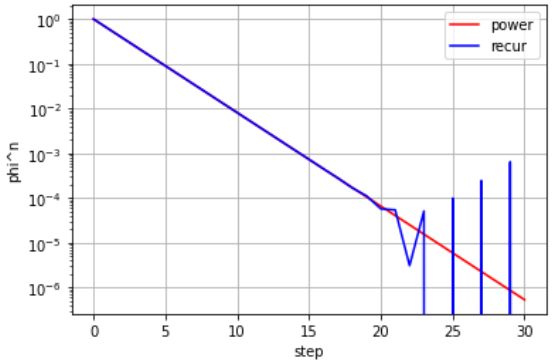
```
import matplotlib.pyplot as plt

phi1 = 0.61803399
phi_recur = [1]
phi_recur.append(phi1)
phi_power = [1]
phi_power.append(phi1)
step = [0, 1]

for i in range(2,31):
    # ...
    #ここを考える
    # ...

plt.plot(step, phi_power, color = 'r', label="power")
plt.plot(step, phi_recur, color = 'b', label="recur")
plt.legend()
plt.xlabel('step')
plt.ylabel('phi^n')
plt.yscale('log')
plt.grid()
plt.show()
```



In [11]:
```
import matplotlib.pyplot as plt

phi1 = 0.61803399
phi_recur = [1]
phi_recur.append(phi1)
phi_power = [1]
phi_power.append(phi1)
```

```
step = [0, 1]

for i in range(2,31):
    recur = phi_recur[-2]-phi_recur[-1]
    power = phi1**(i)
    phi_recur.append(recur)
    phi_power.append(power)
    step.append(i)
    print("%2d: %15.10f, %15.10f" % (i, power, recur))

plt.plot(step, phi_power, color = 'r', label="power")
plt.plot(step, phi_recur, color = 'b', label="recur")
plt.legend()
plt.xlabel('step')
plt.ylabel('phi^n')
plt.yscale('log')
plt.grid()
plt.show()
```

```
 2:   0.3819660128,   0.3819660100
 3:   0.2360679789,   0.2360679800
 4:   0.1458980349,   0.1458980300
 5:   0.0901699447,   0.0901699500
 6:   0.0557280907,   0.0557280800
 7:   0.0344418542,   0.0344418700
 8:   0.0212862366,   0.0212862100
 9:   0.0131556177,   0.0131556600
10:   0.0081306189,   0.0081305500
11:   0.0050249989,   0.0050251100
12:   0.0031056201,   0.0031054400
13:   0.0019193788,   0.0019196700
14:   0.0011862413,   0.0011857700
15:   0.0007331375,   0.0007339000
16:   0.0004531039,   0.0004518700
17:   0.0002800336,   0.0002820300
18:   0.0001730703,   0.0001698400
19:   0.0001069633,   0.0001121900
20:   0.0000661070,   0.0000576500
21:   0.0000408564,   0.0000545400
22:   0.0000252506,   0.0000031100
23:   0.0000156057,   0.0000514300
24:   0.0000096449,  -0.0000483200
25:   0.0000059609,   0.0000997500
26:   0.0000036840,  -0.0001480700
27:   0.0000022768,   0.0002478200
28:   0.0000014072,  -0.0003958900
29:   0.0000008697,   0.0006437100
30:   0.0000005375,  -0.0010396000
```



3 数値積分，解，収束性(25点)

数値積分に際して，関数値の計算にコストがかかる場合，限られた点での値から高精度で計算するNewton-Cotesの公式を使う． 4区間5点の閉区間(closed)公式(Boole's formula)は次のとおりである．

$$h = \frac{b-a}{4} \tag{4}$$

$$\int_a^b f(x)dx = \frac{2}{45}h\{7f(a) + 32f(a+h) + 12f(a+2h) \tag{5}$$

$$+32f(a+3h) + 7f(a+4h)\} \tag{6}$$

これに従って求めた

$$\int_0^1 \frac{4}{1+x^2}dx\,(=\pi) \tag{7}$$

の値は以下の通りである．

数値積分の中点則を用いて求めた場合，同じ程度の精度を達成するには何点が必要となるか．

```
import numpy as np
def func(x):
    return 4.0/(1+x**2)

n = 4
h = 1.0/n

boole = 2/45*h*
(7*func(0)+32*func(h)+12*func(2*h)+32*func(3*h)+7*func(4*h))
print("%26s : %15.10f" % ("from Boole's formula",boole) )
print("%26s : %15.10f" % ("actual Error", boole-np.pi))
print("%26s : %15.10f" % ("Estimated Error f^(6)(0.5)",
h**7*8/945*1311.8)) # 1311.8=$f^6(0.5)$
```

```
        from Boole's formula :    3.1421176471
                actual Error :    0.0005249935
  Estimated Error f^(6)(0.5) :    0.0006778067
```

In [12]:
```
import numpy as np
def func(x):
  return 4.0/(1+x**2)

n = 4
h = 1.0/n

boole = 2/45*h*(7*func(0)+32*func(h)+12*func(2*h)+32*func(3*h)+7*func(4*h))
boole = 2/45*h*(7*func(0)+32*func(h)+12*func(2*h)+32*func(3*h)+7*func(4*h))
print("%26s : %15.10f" % ("from Boole's formula",boole) )
print("%26s : %15.10f" % ("actual Error", boole-np.pi))
print("%26s : %15.10f" % ("Estimated Error f^(6)(0.5)", h**7*8/945*1311.8)) # 1311.8=$f^
```

```
     from Boole's formula :    3.1421176471
           actual Error :    0.0005249935
Estimated Error f^(6)(0.5) :    0.0006778067
```

In [13]:
```
import numpy as np

def func(x):
  return 4.0/(1.0+x**2)

def mid(N):
  x0, xn = 0.0, 1.0

  h = (xn-x0)/N
  S = 0.0
  for i in range(0, N):
    xi = x0 + (i+0.5)*h
    dS = h * func(xi)
    S = S + dS
  return S

x, y = [], []
for i in range(1,20):
  x.append(i)
  error = abs(mid(i)-np.pi)
  y.append(error)
  print("%3d:%15.10f %15.10f" % (i, mid(i), error))
```

```
 1:  3.2000000000   0.0584073464
 2:  3.1623529412   0.0207602876
 3:  3.1508492099   0.0092565563
 4:  3.1468005184   0.0052078648
 5:  3.1449258640   0.0033332104
 6:  3.1439074272   0.0023147736
 7:  3.1432933175   0.0017006639
 8:  3.1428947296   0.0013020760
 9:  3.1426214566   0.0010288030
10:  3.1424259850   0.0008333314
11:  3.1422813577   0.0006887041
12:  3.1421713566   0.0005787031
13:  3.1420857498   0.0004930962
14:  3.1420178234   0.0004251698
15:  3.1419630238   0.0003703702
16:  3.1419181743   0.0003255207
17:  3.1418810041   0.0002883506
18:  3.1418498552   0.0002572016
19:  3.1418234938   0.0002308402
```

ちなみに問１で求めた補間多項式を積分した値は，3.142183333となりBoole公式で求めた値と大体一致する．

4 微分方程式 (25点)

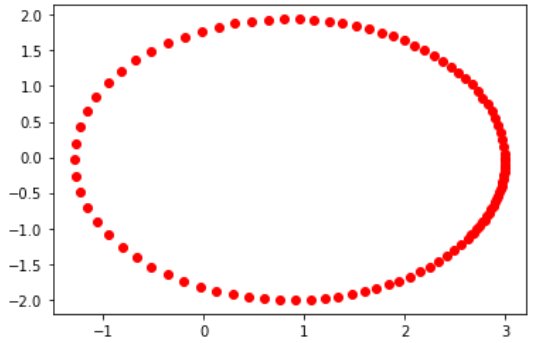
Verlet法による小惑星軌道のシミュレーションを次のような条件で行った．

```
def force(pos):
    x=pos[0]
    y=pos[1]
    L=(x*x+y*y)**(3/2)
    return [-x/L,-y/L]

def Verlet(r0,rh):
    f=force(r0)
    x=2*r0[0]-rh[0]+h**2/m*f[0]
    y=2*r0[1]-rh[1]+h**2/m*f[1]
    return [x,y]

h=0.1
dx, dy=0.0, h
m=0.2
xx=[3.0, 3.0-dx]
yy =[0.0,-dy]
```

1. この小惑星軌道をplotせよ．
2. この小惑星の公転周期が規格化した単位でいかほどになるか答えよ．
3. 異なる初期条件を使って軌跡を表示し，その飛翔体の振る舞いを解説せよ．

In [14]:
```
%matplotlib inline

import matplotlib.pyplot as plt
def force(pos):
  x=pos[0]
  y=pos[1]
  L=(x*x+y*y)**(3/2)
  return [-x/L,-y/L]

def Verlet(r0,rh):
  f=force(r0)
  x=2*r0[0]-rh[0]+h**2/m*f[0]
  y=2*r0[1]-rh[1]+h**2/m*f[1]
  return [x,y]

h=0.1
dx, dy=0.0, h
m=0.2
xx=[3.0, 3.0-dx]
yy =[0.0,-dy]

max_i = 100
for i in range(1,max_i):
  x1, y1 = Verlet([xx[-1],yy[-1]],[xx[-2],yy[-2]])
  xx.append(x1)
  yy.append(y1)

plt.plot(xx,yy,'o',color='r')
plt.show
```

Out[14]: <function matplotlib.pyplot.show(close=None, block=None)>



In [15]:
```
d_period = -12
width = 4
for i in range(d_period-width, d_period+width+1):
    print(i, xx[i])
```

```
-16 2.958663685489252
-15 2.977664855859823
-14 2.9910895191270233
-13 2.99894878154761
-12 3.00125172107917
-11 2.9980053601418253
-10 2.9892146658513203
-9 2.9748825770008547
-8 2.955010058153888
```

In [16]:
```
(max_i+d_period)*h
```

Out[16]: 8.8

In [ ]: