Python Coding Practical 1 solutions

1.1 practical1_matrix.py

1

```
import numpy as np

A=np.array([[2,9,0,0],[0,4,1,4],[7,5,5,1],[7,8,7,4]])
print(A)
```

```
[[2 9 0 0]
 [0 4 1 4]
 [7 5 5 1]
 [7 8 7 4]]
```

```
b=np.array([[-1],[6],[0],[9]])
print(b)
```

```
[[-1]
 [ 6]
 [ 0]
 [ 9]]
```

```
a=np.asmatrix(np.array([3,6,0,9]))   # create row vector
print(a)
```

```
[[3 6 0 9]]
```

```
c=np.dot(A,b)
print(c)
```

```
[[52]
 [60]
 [32]
 [77]]
```

```
c = A + 4
print(c)
```

```
[[ 6 13  4  4]
 [ 4  8  5  8]
 [11  9  9  5]
 [11 12 11  8]]
```

```
c=np.dot(b,a)
print(c)
```

```
[[-3 -6  0 -9]
 [18 36  0 54]
```

```
 [ 0  0  0  0]
 [27 54  0 81]]


c=np.dot(a,b.T)

ValueError: shapes (1,4) and (1,4) not aligned: 4 (dim 1) != 1
(dim 0)

c=np.dot(A,a.T)
print(c)

[[ 60]
 [ 60]
 [ 60]
 [105]]
```

2

```
# square each matrix entry
c=A*A
print(c)

[[ 4 81  0  0]
 [ 0 16  1 16]
 [49 25 25  1]
 [49 64 49 16]]

# square each matrix entry
c=A**2
print(c)

[[ 4 81  0  0]
 [ 0 16  1 16]
 [49 25 25  1]
 [49 64 49 16]]
```

Note: these multiply the corresponding elements of the matrices together.

```
# multiplies matrices
c=np.dot(A,A)
print(c)

[[  4  54   9  36]
 [ 35  53  37  33]
 [ 56 116  37  29]
 [ 91 162  71  55]]

# multiply matrices
c=np.matmul(A,A)
```

```
print(c)
```

```
[[  4  54   9  36]
 [ 35  53  37  33]
 [ 56 116  37  29]
 [ 91 162  71  55]]
```

3

```
# second column of A
Aslice=A[:,1:2]
print(Aslice)
```

```
[[9]
 [4]
 [5]
 [8]]
```

```
# second and third columns of A
Aslice=A[:,1:3]
print(Aslice)
```

```
[[9 0]
 [4 1]
 [5 5]
 [8 7]]
```

4

```
# solve Ax=b
x=np.linalg.solve(A,b)
print(x)
```

```
[[-2.97090909]
 [ 0.54909091]
 [ 3.6       ]
 [ 0.05090909]]
```

1.2 practical1_eigen.py

1

```
# randomly generated 2x2 matrix of positive integers in
range 1 to 50
Arand=np.zeros([2,2])
Arand[0][0]=np.random.randint(1,50)
Arand[0][1]=np.random.randint(1,50)
Arand[1][0]=np.random.randint(1,50)
Arand[1][1]=np.random.randint(1,50)
print(Arand)
```

```
[[33. 12.]
 [ 3. 47.]]
```

2

```
# max value in A
c=np.max(A)
print('max A={0:d}'.format(c))
```

```
max A=9
```

```
# min value in A
c=np.min(A)
print('min A={0:d}'.format(c))
```

```
min A=0
```

3

```
# sorting vector b
c=sorted(b)
print(c[0])
print(c[1])
print(c[2])
print(c[3])
```

```
[-1]
[0]
[6]
[9]
```

4

```
#inverse of A
B=np.linalg.inv(A)

# eigenvalues and eigenvectors of B
evals, evecs = np.linalg.eig(B)
print(evals)
```

```
[-1.43000237+0.j          0.02706353+0.18808977j
0.02706353-0.18808977j
  0.07042076+0.j        ]
```

5

```
# determinant of B - lamda_j I
C = B
for i in range(4):
    C[i][i] = C[i][i] - evals[i]

determ = np.linalg.det(C)
print('determinant={0:8.4f}'.format(determ))
```

```
determinant= -0.0095
```

1.3 practical1_loops.py

1

```
import numpy as np
from scipy.linalg import hilbert

H = np.zeros([5,5])
print(H)
```

```
[[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.]]
```

2

```
# set up the Hilbert matrix
for i in range(5):
    for j in range(5):
        H[i][j] = 1.0/(i+j+1)

print(H)
```

```
[[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125
0.11111111]]
```

```
# using hilbert function in scipy
H2 = hilbert(5)
print(H2)
```

```
[[1.         0.5        0.33333333 0.25       0.2       ]
 [0.5        0.33333333 0.25       0.2        0.16666667]
 [0.33333333 0.25       0.2        0.16666667 0.14285714]
 [0.25       0.2        0.16666667 0.14285714 0.125     ]
 [0.2        0.16666667 0.14285714 0.125
0.11111111]]
```

1.4 practical1_plotting.py

1

```
import numpy as np

x=np.linspace(0,5,101)
```

2

```
y = x**2
```

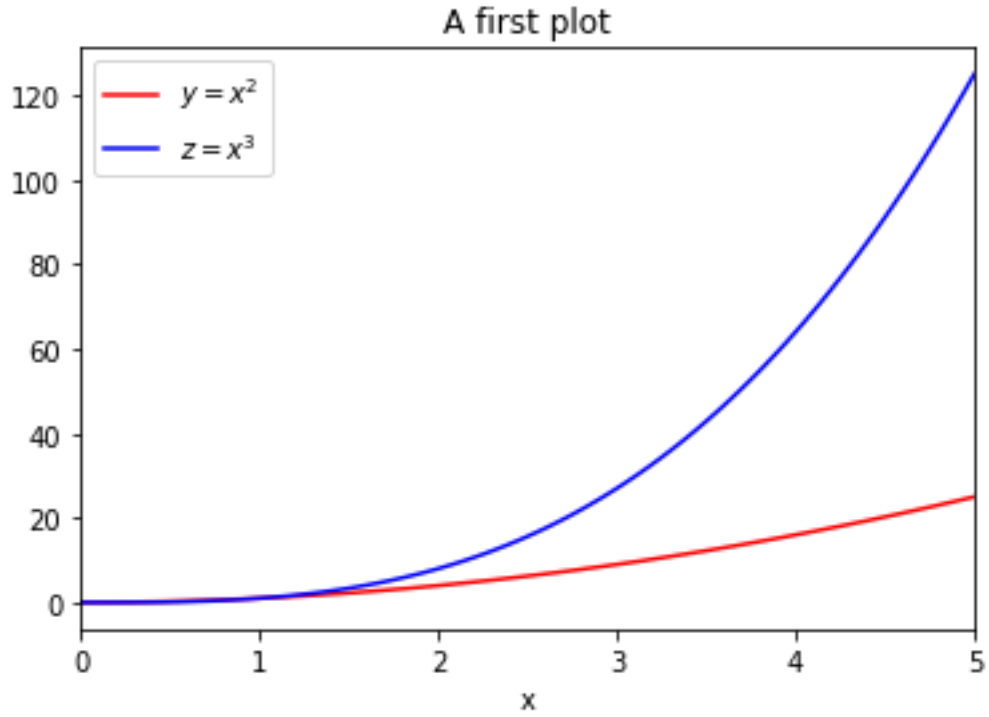
3

```
z = x**3
```

4&5

```
import matplotlib.pyplot as plt
plot1 = plt.figure(1)
plt.plot(x,y,'r')
plt.plot(x,z,'b')
plt.xlim([0,5])
plt.xlabel('x')
plt.legend([r'$y=x^2$','$z=x^3$'])
plt.title('A first plot')
```

Creates the plot:



1.5 practical1_functions.py

1

```
# function definition
def myfunction(a):
    b=a+1
    return b
```

2

```
a=1
b=myfunction(a)
```

```
print(b)
```

```
2
```

3

```
b=0
for i in range(10):
    b=myfunction(b)

print('b={0:d}'.format(b))
```

```
b=10
```

1.6 practical1_carbon.py

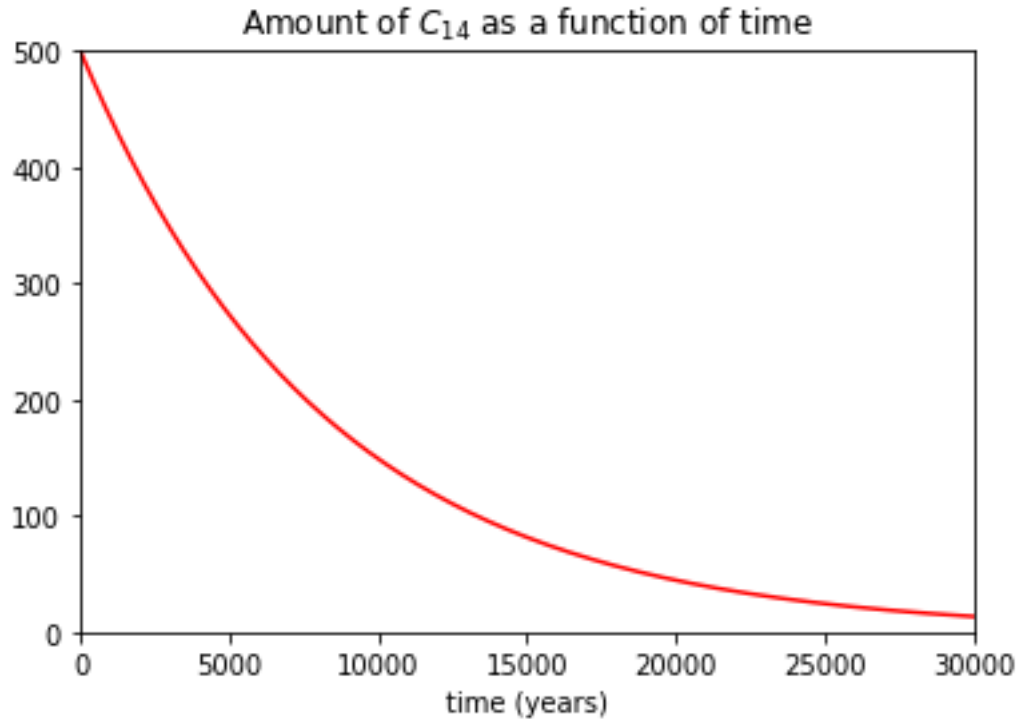
1

```
import numpy as np

# c14 as a function of time and initial value
def c14(initval,t):
    lam = 1.21e-4
    c = initval*np.exp(-lam*t)
    return c
```

2

```
# plot out C14 as a function of t
initval = 500
t = np.linspace(0,30000,3001)

import matplotlib.pyplot as plt
plot1 = plt.figure(1)
plt.plot(t,c14(initval,t),'r')
plt.xlim([0,30000])
plt.ylim([0,500])
plt.xlabel('time (years)')
plt.title(r'Amount of $C_{14}$ as a function of time')
```

create the output:



3

```
# age as a function of ratio of current and initial
values
def age(r_t,r_init):
    lam = 1.21e-4
    t = -np.log(r_t/r_init)/lam
    return t
```

4

```
# calculate ages from dataset of ratios - take initial
ratio = 1
r = [1.4e-12, 1.1e-12,0.8e-12,0.4e-12,0.2e-12,0.1e-
12,0.01e-12]
for ratio in r:
    print('age={0:10.3f} years'.format(age(ratio,1)))
```

```
age=225574.784 years
age=227567.859 years
age=230199.708 years
age=235928.197 years
age=241656.686 years
age=247385.175 years
age=266414.804 years
```