# Home work 3&4 python

1- Calculating the inverse of a 2x2 matrix without using numpy

Sol:

```
import numpy as np
def det(arr):
    return arr[0,0]*arr[1,1]-arr[0,1]*arr[1,0]
def interchange(arr):
    c=np.array([(arr[1,1],-arr[0,1]),(-arr[1,0],arr[0,0])])
    return c
def inv_matrix(arr):
    if det(arr)==0:return "Error the matrix is singular"
    return (1/det(arr))*interchange(arr)
a=[]
for i in range(2):
    b=list(map(int,input().split()))
    a.append(b)
a=np.array(a)
inv_matrix(a)
```

Home_work1.ipynb | Determinants and inverses.ip

Code

[10]:
```
import numpy as np
```

[50]:
```
def det(arr):
    return arr[0,0]*arr[1,1]-arr[0,1]*arr[1,0]
def interchange(arr):
    c=np.array([(arr[1,1],-arr[0,1]),(-arr[1,0],arr[0,0])])
    return c
def inv_matrix(arr):
    if det(arr)==0:return "Error the matrix is singular"
    return (1/det(arr))*interchange(arr)
a=[]
for i in range(2):
    b=list(map(int,input().split()))
    a.append(b)
a=np.array(a)
inv_matrix(a)
```

```
1 2
3 4
```

[50]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Home_work1.ipynb | Determinants and inverses.ip

Code

[10]:
```
import numpy as np
```

[51]:
```
def det(arr):
    return arr[0,0]*arr[1,1]-arr[0,1]*arr[1,0]
def interchange(arr):
    c=np.array([(arr[1,1],-arr[0,1]),(-arr[1,0],arr[0,0])])
    return c
def inv_matrix(arr):
    if det(arr)==0:return "Error the matrix is singular"
    return (1/det(arr))*interchange(arr)
a=[]
for i in range(2):
    b=list(map(int,input().split()))
    a.append(b)
a=np.array(a)
inv_matrix(a)
```

```
1 2
2 4
```

[51]:
```
'Error the matrix is singular'
```

2- Coding a Python code to inverse a 3x3 matrix:

Sol:

```
import numpy as np
def det(arr):
    n=len(arr)
    if n==1:return arr[0,0]
    if n==2:return arr[0,0]*arr[1,1]-arr[1,0]*arr[0,1]
    sum=0
    for i in range(0,n):
        m=minor(arr,0,i)
        sum=sum+((-1)**i)*arr[0,i]*det(m)
    return sum

def minor(arr,i,j):
    c=np.delete(arr,i,0)
    c=np.delete(c,j,1)
    return c

def cofactor(arr):
    n=len(arr)
    c=np.empty([3,3])
    l=0
    for i in range(0,n):
        for j in range(0,n):
```

```
            c[i,j]=((-1)**l)*det(minor(arr,i,j))
            l+=1
    return c

def transpose(arr):
    b=np.array([])
    for i in range(0,3):
        b=np.concatenate((b,arr[:,i]))

    return b.reshape([3,3])



def inv_matrix(arr):
    if det(arr)==0:return "Error the matrix is singular"
    d= det(arr)
    m=cofactor(arr)
    adj=transpose(m)
    return (1/d)*adj

a=[]
for i in range(3):
    b=list(map(int,input().split()))
    a.append(b)
```

```
a=np.array(a)
c=inv_matrix(a)
print(c)
```

Home_work1.ipynb | IntoToNumpy-checkpoint.ipy | Determi

Code

[89]:
```
def det(arr):
    n=len(arr)
    if n==1:return arr[0,0]
    if n==2:return arr[0,0]*arr[1,1]-arr[1,0]*arr[0,1]
    sum=0
    for i in range(0,n):
        m=minor(arr,0,i)
        sum=sum+((-1)**i)*arr[0,i]*det(m)
    return sum

def minor(arr,i,j):
    c=np.delete(arr,i,0)
    c=np.delete(c,j,1)
    return c

def cofactor(arr):
    n=len(arr)
    c=np.empty([3,3])
    l=0
    for i in range(0,n):
        for j in range(0,n):
            c[i,j]=((-1)**l)*det(minor(arr,i,j))
            l+=1
    return c

def transpose(arr):
    b=np.array([])
    for i in range(0,3):
        b=np.concatenate((b,arr[:,i]))

    return b.reshape([3,3])



def inv_matrix(arr):
    if det(arr)==0:return "Error the matrix is singular"
    d= det(arr)
    m=cofactor(arr)
    adj=transpose(m)
    return (1/d)*adj

a=[]
for i in range(3):
    b=list(map(int,input().split()))
    a.append(b)
a=np.array(a)
c=inv_matrix(a)
print(c)
```

```
 3 1 1
 1 2 3
 2 3 8
[[ 0.38888889 -0.27777778  0.05555556]
 [-0.11111111  1.22222222 -0.44444444]
 [-0.05555556 -0.38888889  0.27777778]]
```

Home_work1.ipynb | IntoToNumpy-checkpoint.ipy | Determinants a

Code

[90]:

```
def det(arr):
    n=len(arr)
    if n==1:return arr[0,0]
    if n==2:return arr[0,0]*arr[1,1]-arr[1,0]*arr[0,1]
    sum=0
    for i in range(0,n):
        m=minor(arr,0,i)
        sum=sum+((-1)**i)*arr[0,i]*det(m)
    return sum

def minor(arr,i,j):
    c=np.delete(arr,i,0)
    c=np.delete(c,j,1)
    return c

def cofactor(arr):
    n=len(arr)
    c=np.empty([3,3])
    l=0
    for i in range(0,n):
        for j in range(0,n):
            c[i,j]=((-1)**l)*det(minor(arr,i,j))
            l+=1
    return c

def transpose(arr):
    b=np.array([])
    for i in range(0,3):
        b=np.concatenate((b,arr[:,i]))

    return b.reshape([3,3])



def inv_matrix(arr):
    if det(arr)==0:return "Error the matrix is singular"
    d= det(arr)
    m=cofactor(arr)
    adj=transpose(m)
    return (1/d)*adj

a=[]
for i in range(3):
    b=list(map(int,input().split()))
    a.append(b)
a=np.array(a)
c=inv_matrix(a)
print(c)
```

```
 1 2 3
 2 4 6
 4 8 12
Error the matrix is singular
```