9. Numpy (Abhijit Kar Gupta)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

9.1 Introducing Arrays

In [2]:
```
l1 = [2,4,2,4,5,3,6,2,2,5,0,5]
l2 = [[2,5,1,3],[1,6,6,7]]
l3 = [[4,1,5],[5,2,6],[6,2,4+2j]]
l4 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'A','B','C','D','E','F']

ar1 = np.array(l1)
ar2 = np.array(l2)
ar3 = np.array(l3)
ar4 = np.array(l4)

display(ar1, ar2, ar3, ar4)
```

```
array([2, 4, 2, 4, 5, 3, 6, 2, 2, 5, 0, 5])

array([[2, 5, 1, 3],
       [1, 6, 6, 7]])

array([[4.+0.j, 1.+0.j, 5.+0.j],
       [5.+0.j, 2.+0.j, 6.+0.j],
       [6.+0.j, 2.+0.j, 4.+2.j]])

array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C',
       'D', 'E', 'F'], dtype='<U11')
```

In [3]:
```
print('shapes:', ar1.shape, ar2.shape)
print('ndim:', ar1.ndim, ar2.ndim)
print('reshape:', ar1.reshape(4,3), ar1.shape)
ar1.resize(6,2)
print('resize:', ar1, ar1.shape)

ar1c = np.array(l1, dtype='complex')
print('dtypes:', ar1.dtype, ar2.dtype, ar3.dtype, ar4.dtype, ar1c.dtype)
```

```
shapes: (12,) (2, 4)
ndim: 1 2
reshape: [[2 4 2]
 [4 5 3]
 [6 2 2]
 [5 0 5]] (12,)
resize: [[2 4]
 [2 4]
 [5 3]
 [6 2]
 [2 5]
 [0 5]] (6, 2)
dtypes: int32 int32 complex128 <U11 complex128
```

Special arrays

In [4]:
```python
print('zeros:', np.zeros(4), np.zeros((3,2)))
print('ones:', np.ones(4), np.ones((2,3)))
print('constant - full:', np.full((2,4),8))
print('identity matrix - eye:', np.eye(3))
print('random:', np.random.random((2,3)))
```

```
zeros: [0. 0. 0. 0.] [[0. 0.]
 [0. 0.]
 [0. 0.]]
ones: [1. 1. 1. 1.] [[1. 1. 1.]
 [1. 1. 1.]]
constant - full: [[8 8 8 8]
 [8 8 8 8]]
identity matrix - eye: [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
random: [[0.42145812 0.27985541 0.50836044]
 [0.49881903 0.99617629 0.43757142]]
```

arange: **`np.arange(start, end, step)`**

linspace: **`np.linspace(start, end, no. of elements)`**

In [5]:
```python
print('arange:', np.arange(5,25,2.5))
print('linspace:', np.linspace(1,7,10))
```

```
arange: [ 5.   7.5 10.  12.5 15.  17.5 20.  22.5]
linspace: [1.         1.66666667 2.33333333 3.         3.66666667 4.33333333
 5.         5.66666667 6.33333333 7.        ]
```

Algebra with arrays

In [6]:
```python
a = np.array([4,7,8,5,6,1])
b = np.array([1,4,2,9,8,3])
print('a =', a, '\nb =', b)

print('operations on elements:', 4*a**2-3*b/a)
print('sum =', sum(a), '\t product =', np.prod(a), '\t max =', max(a))
print('average =', np.average(a), '\t mean =', np.mean(a),
        '\t median =', np.median(a), '\t variance =', np.var(a))
print('difference between consecutive elements =', np.diff(a))

c = np.array([2, 4+1j, 7, 1, 7-2j])
print('conjugate:', np.conj(c))
```

```
a = [4 7 8 5 6 1]
b = [1 4 2 9 8 3]
operations on elements: [ 63.25       194.28571429 255.25        94.6        140.
  -5.        ]
sum = 31         product = 6720          max = 8
average = 5.166666666666667      mean = 5.166666666666667        median = 5.5     variance
= 5.138888888888889
difference between consecutive elements = [ 3  1 -3  1 -5]
conjugate: [2.-0.j 4.-1.j 7.-0.j 1.-0.j 7.+2.j]
```

In [7]:
```python
print('lower end integer:', np.floor(4.3656))
print('upper end integer:', np.ceil(np.exp(2)))
```

```
lower end integer: 4.0
upper end integer: 8.0
```

In [8]:
```python
print('concatenation:', np.concatenate((a,b)))
```

```
concatenation: [4 7 8 5 6 1 1 4 2 9 8 3]
```

arrays as Matrix

In [9]:
```
amat, bmat = a.reshape(2,3), b.reshape(2,3)
bt = bmat.T
print('matrix a =', amat, '\nmatrix b =', bmat, '\nmatrix bt (transpose) =', bt)

print('addition and subtraction', np.add(amat, bmat), 4*amat-5*bmat/2)
print('matrix multiplication:', np.dot(amat, bt), amat @ bt)
print('trace:', np.trace(amat))
```

```
matrix a = [[4 7 8]
 [5 6 1]]
matrix b = [[1 4 2]
 [9 8 3]]
matrix bt (transpose) = [[1 9]
 [4 8]
 [2 3]]
addition and subtraction [[ 5 11 10]
 [14 14  4]] [[13.5 18.   27. ]
 [-2.5  4.  -3.5]]
matrix multiplication: [[ 48 116]
 [ 31  96]] [[ 48 116]
 [ 31  96]]
trace: 10
```

grids by arrays

In [10]:
```python
g1 = np.mgrid[0:4, 0:2]
g2 = np.mgrid[0:2, 0:3, 0:4]
g3 = np.mgrid[0:2:0.5, 0:3] # scale = 0.5
display(g1, g2, g3)
```

```
array([[[0, 0],
        [1, 1],
        [2, 2],
        [3, 3]],

       [[0, 1],
        [0, 1],
        [0, 1],
        [0, 1]]])

array([[[[0, 0, 0, 0],
         [0, 0, 0, 0],
         [0, 0, 0, 0]],

        [[1, 1, 1, 1],
         [1, 1, 1, 1],
         [1, 1, 1, 1]]],


       [[[0, 0, 0, 0],
         [1, 1, 1, 1],
         [2, 2, 2, 2]],

        [[0, 0, 0, 0],
         [1, 1, 1, 1],
         [2, 2, 2, 2]]],


       [[[0, 1, 2, 3],
         [0, 1, 2, 3],
         [0, 1, 2, 3]],

        [[0, 1, 2, 3],
         [0, 1, 2, 3],
         [0, 1, 2, 3]]]])

array([[[0. , 0. , 0. ],
        [0.5, 0.5, 0.5],
        [1. , 1. , 1. ],
        [1.5, 1.5, 1.5]],

       [[0. , 1. , 2. ],
        [0. , 1. , 2. ],
        [0. , 1. , 2. ],
        [0. , 1. , 2. ]]])
```

Functions of arrays

In [11]:
```python
def fa(x):
    return x**2 - np.sin(2*x)*np.exp(x/3)

print(fa(amat))
```

```
[[12.24670338 38.78460087 68.14348038]
 [27.88031436 39.96476739 -0.26902679]]
```

slicing of arrays

In [12]:
```
ar1d = np.arange(21)
ar2d = np.arange(20).reshape(4,5)
display('ar1d', ar1d, 'ar2d', ar2d)

display(ar1d[::4], ar1d[2:14:3])
display(ar2d[1:4,:5:2])
```

```
'ar1d'

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20])

'ar2d'

array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])

array([ 0,  4,  8, 12, 16, 20])

array([ 2,  5,  8, 11])

array([[ 5,  7,  9],
       [10, 12, 14],
       [15, 17, 19]])
```

arrays as vectors

In [13]:
```
v1 = np.array([4,7,2])
v2 = np.array([7,0,3])
m1, m2 = amat, bmat
display('v1', v1, 'v2', v2, 'm1', m1, 'm2', m2)

print('inner product:', np.inner(4,v1), np.inner(v1, v2), np.inner(np.eye(3),2.5), np.inner
print('vector product (dot):', np.vdot(v1, v2), np.vdot(m1, m2))
print('cross product:', np.cross(v1, v2))
```

```
'v1'

array([4, 7, 2])

'v2'

array([7, 0, 3])

'm1'

array([[4, 7, 8],
       [5, 6, 1]])

'm2'

array([[1, 4, 2],
       [9, 8, 3]])

inner product: [16 28  8] 34 [[2.5 0.  0. ]
 [0.  2.5 0. ]
 [0.  0.  2.5]] [[ 48 116]
 [ 31  96]]
vector product (dot): 34 144
cross product: [ 21   2 -49]
```

Example: Volume of a Parallelepiped:

Three sides are given bythree vectors: $\vec{A} = 2\hat{i} - 3\hat{j}$, $\vec{B} = \hat{i} + \hat{j} - \hat{k}$ and $\vec{C} = 3\hat{i} - \hat{k}$.

Solution: Formula for Volume of a Parallelepiped:,

$$V = \vec{A}.(\vec{B} \times \vec{C})$$

In [14]:
```
av = np.array([2,-3,0])
bv = np.array([1,1,-1])
cv = np.array([3,0,-1])

vol = np.vdot(av, np.cross(bv, cv))
print('Volume of the Parallelepiped:', vol)
```

```
Volume of the Parallelepiped: 4
```

9.2 Polynomial by Numpy

In [15]:
```
from numpy import poly1d
```

In [16]:
```
p1 = np.poly1d([1,4])
p2 = np.poly1d([2,5,3])
p3 = np.poly1d([1,4,2,5])

print('polynomials:', p1)
print(p2)
print(p3)
print('using as a function:', p1(3), p2(1j), p3(-5))
display('coefficients:', p1.c, p2.c, p3.c)
print('orders:', p1.order, p2.order, p3.order)
```

```
polynomials:
1 x + 4
   2
2 x + 5 x + 3
   3     2
1 x + 4 x + 2 x + 5
using as a function: 7 (1+5j) -30
```

```
'coefficients:'
```

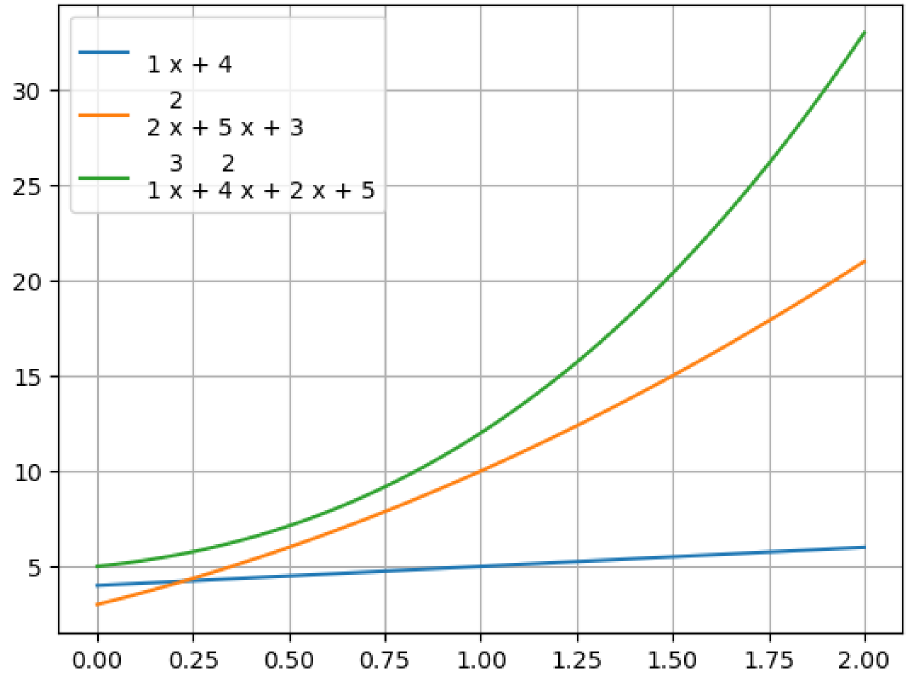
```
array([1, 4])
```

```
array([2, 5, 3])
```

```
array([1, 4, 2, 5])
```

```
orders: 1 2 3
```

In [17]:
```
x = np.linspace(0,2,50)
plt.plot(x, p1(x), label=p1)
plt.plot(x, p2(x), label=p2)
plt.plot(x, p3(x), label=p3)
plt.legend()
plt.grid()
plt.show()
```



Operations

In [18]:
```
p1a = poly1d([1,2])
p1b = poly1d([2,5])
print(p1a, p1b)
print('addition and subtraction:', 2*p1a + p1b, 5*p1a-3*p1b)
print('multiplication:')
print(p1b*p1a)
print('division:', p1b/p1a,
      '\nresult:', (p1b/p1a)[0], '\nremainder:', (p1b/p1a)[1])

print('functions:')
print(p1a**2 + 8*np.cos(p1a)*np.exp(p1b/2) - 7)

display('roots:', p2.r, p3.r)
```

```
1 x + 2
2 x + 5
addition and subtraction:
4 x + 9
-1 x - 5
multiplication:
   2
2 x + 9 x + 10
division: (poly1d([2.]), poly1d([1.]))
result:
2
remainder:
1
functions:
   2
1 x + 15.75 x - 43.56
```

```
'roots:'
```

```
array([-1.5, -1. ])
```

```
array([-3.81912114+0.j        , -0.09043943+1.14062371j,
       -0.09043943-1.14062371j])
```

Differentiation and Integration

In [19]:
```
print('\t derivatives:')
print(p2.deriv(), '; first derivative.')
print(p2.deriv(2), '; second derivative.')

print('\n\t indefinite integration (without constants):')
print(p1.integ())
print(p1.integ(2), '; double integral.')
```

```
	 derivatives:

4 x + 5 ; first derivative.

4 ; second derivative.

	 indefinite integration (without constants):
     2
0.5 x + 4 x
        3     2
0.1667 x + 2 x ; double integral.
```

9.3 Curve Fitting by numpy

In [20]:
```
import numpy.polynomial.polynomial as poly
```

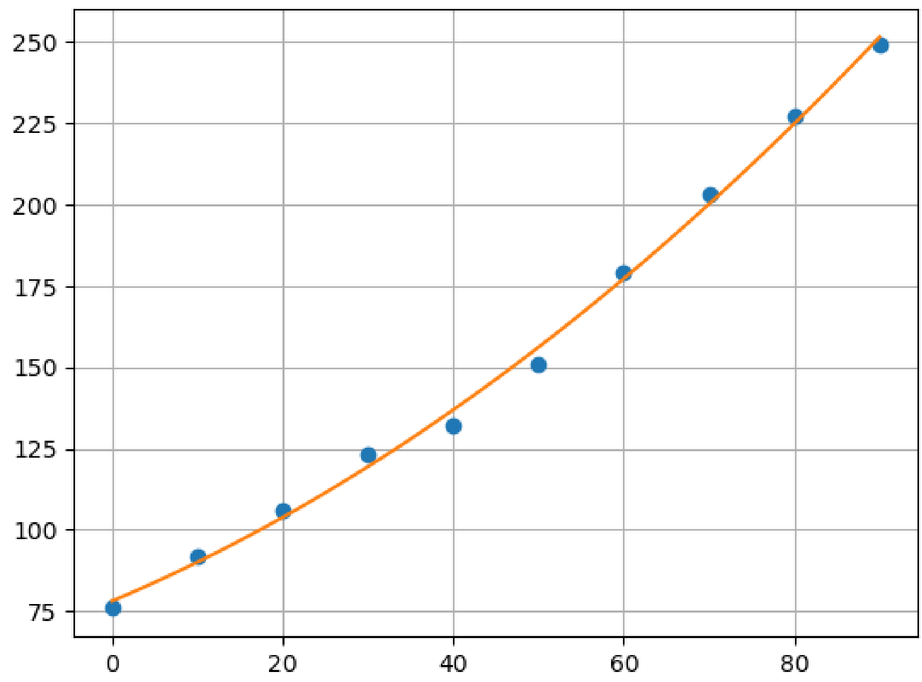
In [21]:
```
xdata = np.array([0,10,20,30,40,50,60,70,80,90])
ydata = np.array([76, 92,106, 123, 132, 151, 179,203,227,249])

coeffs = poly.polyfit(xdata, ydata, 2)  # order = 2
display(coeffs)

yfit = poly.polyval(xdata, coeffs)
x1 = np.linspace(0,90,100)
yfit1 = poly.polyval(x1, coeffs)
```

```
array([7.81909091e+01, 1.10204545e+00, 9.12878788e-03])
```

In [22]:
```
# for full information:
coeffs1, res = poly.polyfit(xdata, ydata, 2, full = True)
display(res)
```

```
[array([97.48333333]),
 3,
 array([1.64219501, 0.53895301, 0.11280603]),
 2.220446049250313e-15]
```

In [23]:
```
plt.plot(xdata, ydata, 'o')
plt.plot(x1, yfit1)
plt.grid()
plt.show()
```



All the polynomial modules and functions being used here, are available with scipy too.

χ^2 (chi-square) test to see the goodness of fit:

In [24]:
```
from scipy.stats import chisquare
chisquare(ydata, yfit)
```

Out[24]:
```
Power_divergenceResult(statistic=0.6945991858131235, pvalue=0.9998765425805308)
```

by **`np.polyfit()`** :

In [25]:
```python
coeffs2 = np.polyfit(xdata, ydata, 2) # opposite order
display(coeffs, coeffs2)

coeffs2p, cov = np.polyfit(xdata, ydata, 2, cov=True)
display(cov) # covariance matrix; diagonal elements are the variances
stddev = np.sqrt(np.diag(cov)) # standard deviation
display(stddev)

fn = poly1d(coeffs2)
print('function:')
print(fn)
```

```
array([7.81909091e+01, 1.10204545e+00, 9.12878788e-03])

array([9.12878788e-03, 1.10204545e+00, 7.81909091e+01])

array([[ 2.63753608e-06, -2.37378247e-04,  3.16504329e-03],
       [-2.37378247e-04,  2.30520653e-02, -3.60814935e-01],
       [ 3.16504329e-03, -3.60814935e-01,  8.60891775e+00]])

array([1.62404928e-03, 1.51829066e-01, 2.93409573e+00])

function:
          2
0.009129 x + 1.102 x + 78.19
```

User defined functions (**curve_fit**)

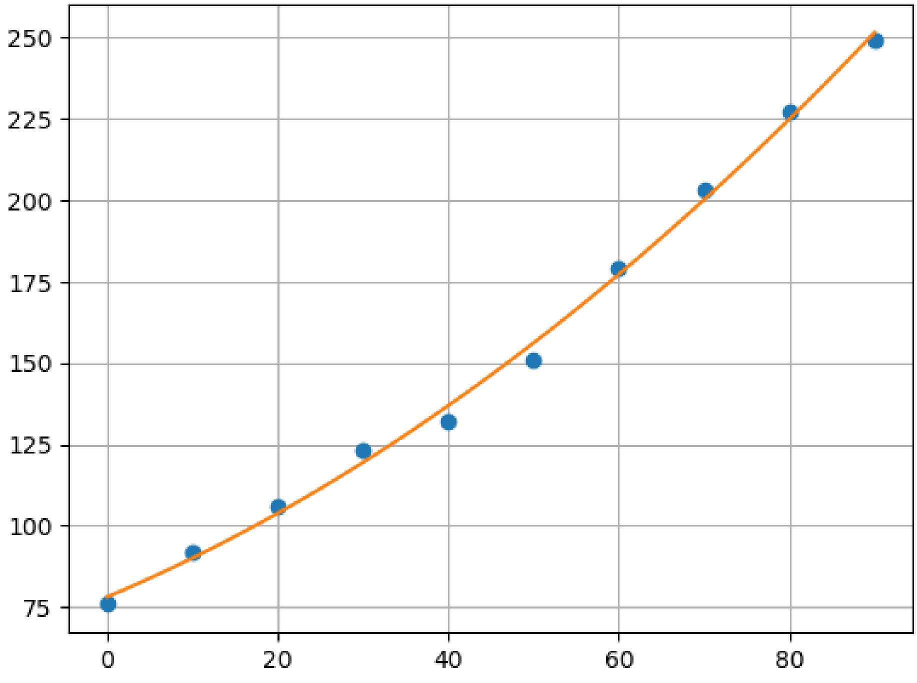
In [26]:
```python
from scipy.optimize import curve_fit
```

In [27]:
```python
def f1(x,a,b,c):
    return a*x**2 + b*x + c

par, var = curve_fit(f1, xdata, ydata)
display('parameters', par, 'variances', var)
```

```
'parameters'

array([9.12878788e-03, 1.10204545e+00, 7.81909091e+01])

'variances'

array([[ 2.63753558e-06, -2.37378200e-04,  3.16504276e-03],
       [-2.37378200e-04,  2.30520607e-02, -3.60814882e-01],
       [ 3.16504276e-03, -3.60814882e-01,  8.60891713e+00]])
```

In [28]:

```
yfit2 = f1(x1, par[0], par[1], par[2])
plt.plot(xdata, ydata, 'o')
plt.plot(x1, yfit2)
plt.grid()
plt.show()
```



9.4 System of Linear Equations

In [ ]:

```
```