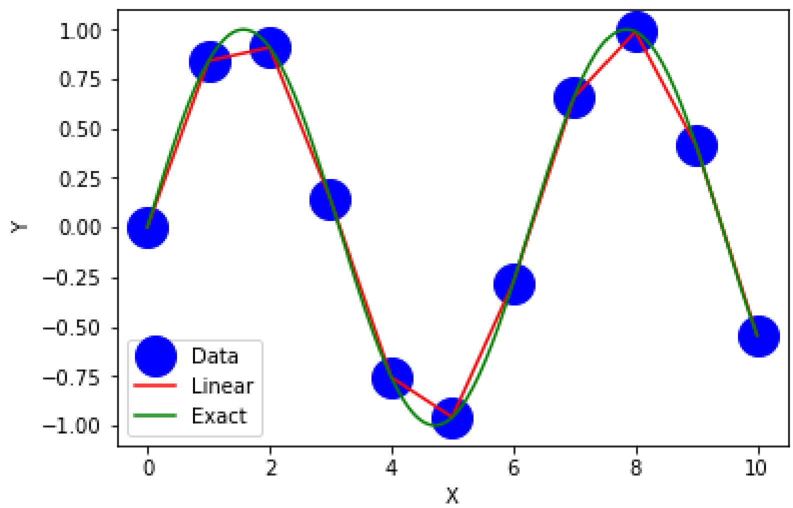
In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d

# make our tabular values
x_table = np.arange(11)
y_table = np.sin(x_table)

# linearly interpolate
x = np.linspace(0.,10.,201)

# here we create linear interpolation function
linear = interp1d(x_table,y_table,'linear')

# apply and create new array
y_linear = linear(x)

# plot results to illustrate
plt.ion()
plt.plot(x_table,y_table,'bo',markersize=20)
plt.plot(x,y_linear,'r')
plt.plot(x,np.sin(x),'g')
plt.legend(['Data','Linear','Exact'],loc='best')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



In [5]:

```
import numpy as np

def lagrange(x,i,xm):
    """
    Evaluates the i-th Lagrange polynomial
    at xbased on grid data xm
    """
    n=len(xm)-1
    y=1
    for j in range(n+1):
        if i!=j:
            y*=(x-xm[j])/(xm[i]-xm[j])
    return y

def interpolation(x,xm ,ym):
    n=len(xm)-1
    lagrpoly=np.array([lagrange(x,i,xm) for i in range(n+1)])
    y = np.dot(ym ,lagrpoly)
    return y

# Example
xm = np.array([1,2,3,4,5,6])
ym = np.array([-3,0,-1,2,1,4])
xplot = np.linspace(0.9,6.1,100)
yplot = interpolation(xplot ,xm,ym)
```

In [6]:

```
xplot
```

Out[6]:

```
array([ 0.9       ,  0.95252525,  1.00505051,  1.05757576,  1.11010101,
        1.16262626,  1.21515152,  1.26767677,  1.32020202,  1.37272727,
        1.42525253,  1.47777778,  1.53030303,  1.58282828,  1.63535354,
        1.68787879,  1.74040404,  1.79292929,  1.84545455,  1.8979798 ,
        1.95050505,  2.0030303 ,  2.05555556,  2.10808081,  2.16060606,
        2.21313131,  2.26565657,  2.31818182,  2.37070707,  2.42323232,
        2.47575758,  2.52828283,  2.58080808,  2.63333333,  2.68585859,
        2.73838384,  2.79090909,  2.84343434,  2.8959596 ,  2.94848485,
        3.0010101 ,  3.05353535,  3.10606061,  3.15858586,  3.21111111,
        3.26363636,  3.31616162,  3.36868687,  3.42121212,  3.47373737,
        3.52626263,  3.57878788,  3.63131313,  3.68383838,  3.73636364,
        3.78888889,  3.84141414,  3.89393939,  3.94646465,  3.9989899 ,
        4.05151515,  4.1040404 ,  4.15656566,  4.20909091,  4.26161616,
        4.31414141,  4.36666667,  4.41919192,  4.47171717,  4.52424242,
        4.57676768,  4.62929293,  4.68181818,  4.73434343,  4.78686869,
        4.83939394,  4.89191919,  4.94444444,  4.9969697 ,  5.04949495,
        5.1020202 ,  5.15454545,  5.20707071,  5.25959596,  5.31212121,
        5.36464646,  5.41717172,  5.46969697,  5.52222222,  5.57474747,
        5.62727273,  5.67979798,  5.73232323,  5.78484848,  5.83737374,
        5.88989899,  5.94242424,  5.99494949,  6.04747475,  6.1       ])
```

In [7]:

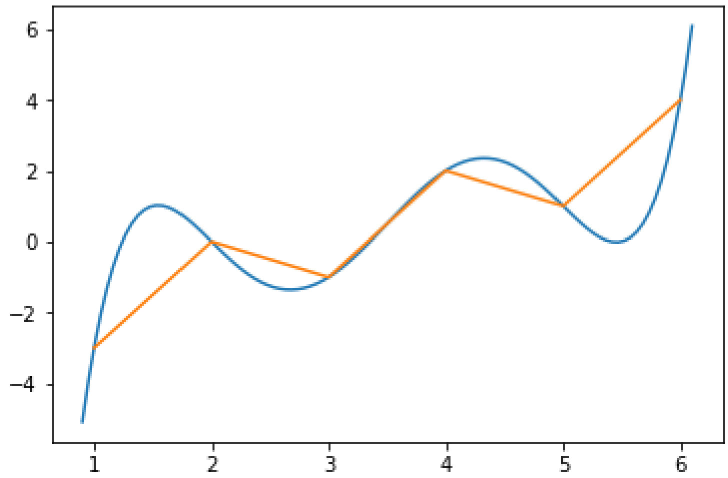
```
yplot
```

Out[7]:

```
array([-5.088336  , -3.91939949, -2.90943252, -2.04543428, -1.31501164,
       -0.70636638, -0.20828239,  0.18988713,  0.49823246,  0.72630016,
        0.88310586,  0.97714704,  1.01641584,  1.00841186,  0.96015492,
        0.8781979 ,  0.76863949,  0.63713702,  0.48891922,  0.32879903,
        0.16118642, -0.00989888, -0.18081452, -0.34828279, -0.50937777,
       -0.66151261, -0.80242667, -0.93017274, -1.04310429, -1.13986262,
       -1.21936411, -1.28078738, -1.32356056, -1.34734843, -1.35203968,
       -1.33773406, -1.30472967, -1.25351007, -1.18473155, -1.09921033,
       -0.99790974, -0.88192745, -0.75248267, -0.61090334, -0.45861338,
       -0.29711985, -0.12800016,  0.04711068,  0.2265329 ,  0.40855474,
        0.59144526,  0.7734671 ,  0.95288932,  1.12800016,  1.29711985,
        1.45861338,  1.61090334,  1.75248267,  1.88192745,  1.99790974,
        2.09921033,  2.18473155,  2.25351007,  2.30472967,  2.33773406,
        2.35203968,  2.34734843,  2.32356056,  2.28078738,  2.21936411,
        2.13986262,  2.04310429,  1.93017274,  1.80242667,  1.66151261,
        1.50937777,  1.34828279,  1.18081452,  1.00989888,  0.83881358,
        0.67120097,  0.51108078,  0.36286298,  0.23136051,  0.1218021 ,
        0.03984508, -0.00841186, -0.01641584,  0.02285296,  0.11689414,
        0.27369984,  0.50176754,  0.81011287,  1.20828239,  1.70636638,
        2.31501164,  3.04543428,  3.90943252,  4.91939949,  6.088336  ])
```

In [13]:

```
import matplotlib.pyplot as plt
plt.plot(xplot,yplot)
plt.plot(xm,ym)
plt.show()
```



In [12]:

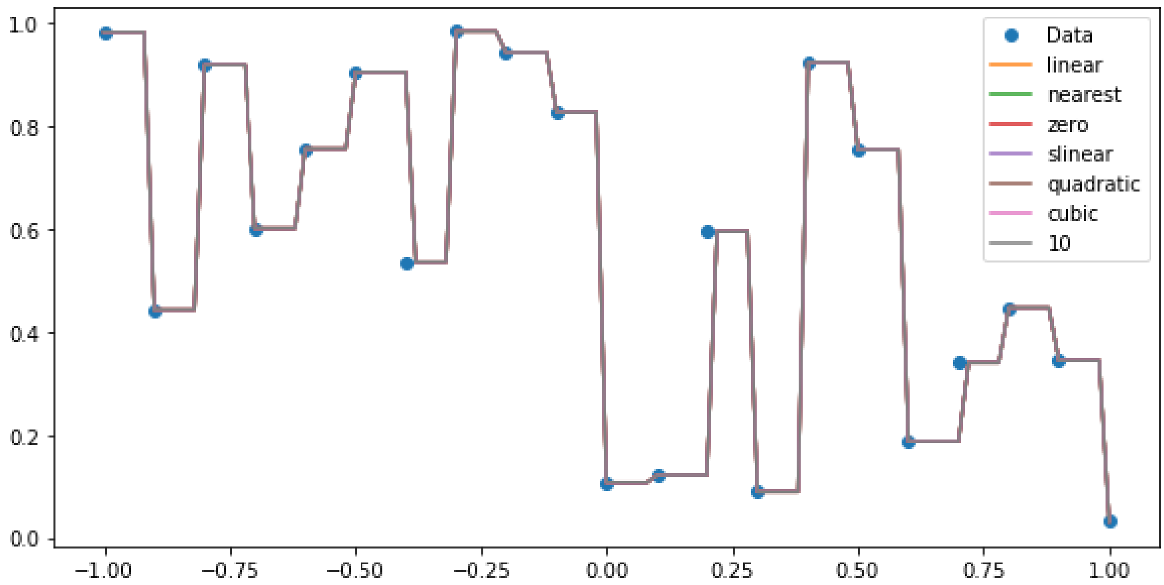
```
# Program to interpolate using
# newton forward interpolation
# calculating u mentioned in the formula
def u_cal(u, n):
    temp = u;
    for i in range(1, n):
        temp = temp * (u - i);
    return temp;

# calculating factorial of given number n
def fact(n):
    f = 1;
    for i in range(2, n + 1):
        f *= i;
    return f;

# Driver Code
# Number of values given
n = 4;
x = [ 45, 50, 55, 60 ];
# y[][] is used for difference table
# with y[][0] used for input
y = [[0 for i in range(n)]
        for j in range(n)];
y[0][0] = 0.7071;
y[1][0] = 0.7660;
y[2][0] = 0.8192;
y[3][0] = 0.8660;
# Calculating the forward difference table

for i in range(1, n):
    for j in range(n - i):
        y[j][i] = y[j + 1][i - 1] - y[j][i - 1];
    # Displaying the forward difference table

for i in range(n):
    print(x[i], end = "\t");
    for j in range(n - i):
        print(y[i][j], end = "\t");
    print("");

# Value to interpolate at
value = 52;
# initializing u and sum
sum = y[0][0];
u = (value - x[0]) / (x[1] - x[0]);

for i in range(1,n):
    sum = sum + (u_cal(u, i) * y[0][i]) / fact(i);

print("\nValue at", value,"is", round(sum, 6));
# This code is contributed by mits
```

```
45      0.7071  0.05890000000000006     -0.005700000000000038   -0.0007000
000000000339
50      0.766   0.05320000000000025     -0.006400000000000072
55      0.8192  0.04679999999999995
60      0.866
```

```
Value at 52 is 0.788003
```

In [41]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# Original "data set" --- 21 random numbers between 0 and 1.
x0 = np.linspace(-1,1,21)
y0 = np.random.random(21)
plt.figure(frameon=False,figsize=(10,5))
plt.plot(x0, y0, 'o', label='Data')

# Array with points in between those of the data set for interpolation.
x = np.linspace(-1,1,101)

# Available options for interp1d
options = ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 10)

for o in options:
    f = interp1d(x0, y0,kind=o)     # interpolation function
    plt.plot(x, f(x), label=o)       # plot of interpolated data

plt.legend(loc='best')

plt.show()
```



Consider the vapour - liquid equilibrium mole fractiom data below for the binary system of methanol and water at 1 atm.

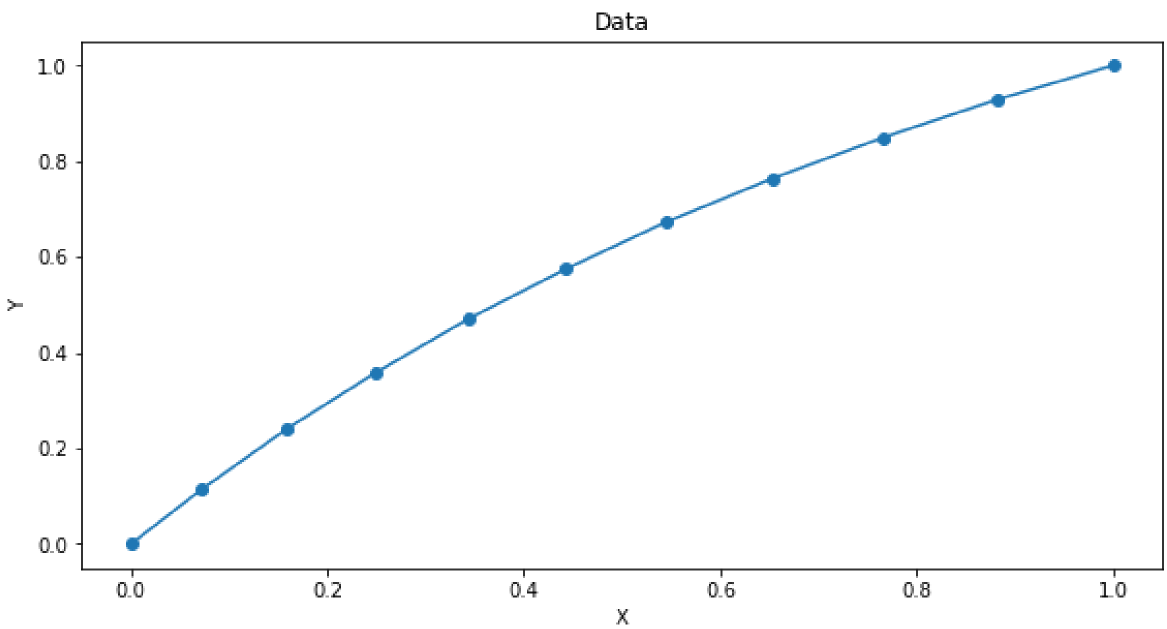
$X = [1, 0.882, 0.765, 0.653, 0.545, 0.443, 0.344, 0.25, 0.159, 0.072, 0]$
$Y = [1, 0.929, 0.849, 0.764, 0.673, 0.575, 0.471, 0.359, 0.241, 0.114, 0]$

Determine the vapour mole fraction of methanol (y) corresponding to the liquid mole fraction of methanol of $x = 0.15$ by linear interpolation and a quadratic Lagrange interpolating

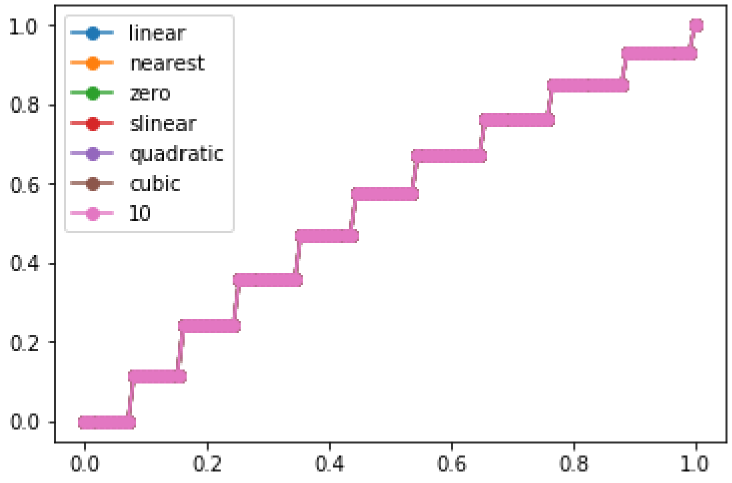
polynomial. Compare your results to the experimental value of $y = 0.517$

In [21]:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d

X = [1,0.882,0.765,0.653,0.545,0.443,0.344,0.25,0.159,0.072,0]
Y = [1,0.929,0.849,0.764,0.673,0.575,0.471,0.359,0.241,0.114,0]

plt.figure(frameon=False,figsize=(10,5))
plt.plot(X, Y,marker = "o")
plt.title("Data")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()

xi = interp1d(Y,X,kind = 2)
yi = interp1d(X,Y,kind = 2)

print("Vapour mole fraction of methanol (y) at x = 0.15 is: {0}".format(yi(0.15)))
print("Experimental value at y = 0.517 is: {0}".format(xi(0.517)))

# Array with points in between those of the data set for interpolation.
x = np.linspace(0,1,100)

# Available options for interp1d
options = ('linear', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 10)

for o in options:
    f = interp1d(X, Y, kind=0)    # interpolation function
    plt.plot(x, f(x), label=o,marker = "o")      # plot of interpolated data

plt.legend(loc='best')
plt.show()
```




```
Vapour mole fraction of methanol (y) at x = 0.15 is: 0.2285253925303365
Experimental value at y = 0.517 is: 0.38649191954545176
```



The interpolating polynomial has resulted in this graph