Assignment 3

Ashmit Bathla, 210216

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

Question 1

In [5]:
```
def kinematics(i,  x , t):
    if i < 1 or i > np.size(x)- 2 :
        raise ValueError("Out of bound value for i indedx variable.")
    v = ( x[i+1] - x[i-1])/(t[i+1] - t[i-1])
    a = 2*( (x[i+1]-x[i])/(t[i+1]-t[i]) - (x[i]-x[i-1])/(t[i]-t[i-
1]))/( t[i+1] - t[i-1])
    return v , a


def check_kinematics( V ):
    t = np.array([0,.5,1.5,2.2])
    V = 2
    x = V*t
    for i in range( 1 , np.size(t) - 1):
        v , a = kinematics( i , x , t )
        print( 'i = ' , i , '; v[i] = ' , v , 'a[i] = ' , a )

check_kinematics( 2)
```

```
i =  1 ; v[i] =  2.0 a[i] =  0.0
i =  2 ; v[i] =  2.0 a[i] =  0.0
```

Question 2

In [8]:
```
def pair_count( mstring , sstring ):
    l = len( sstring )
    count = 0
    for i in range( 0 , len(mstring)):
        if mstring[i:i+l] == sstring :
            count = count + 1
    return count

pair_count( 'ACTGCTATCCATT' , 'AT')
```

Out[8]: 2

Question 3

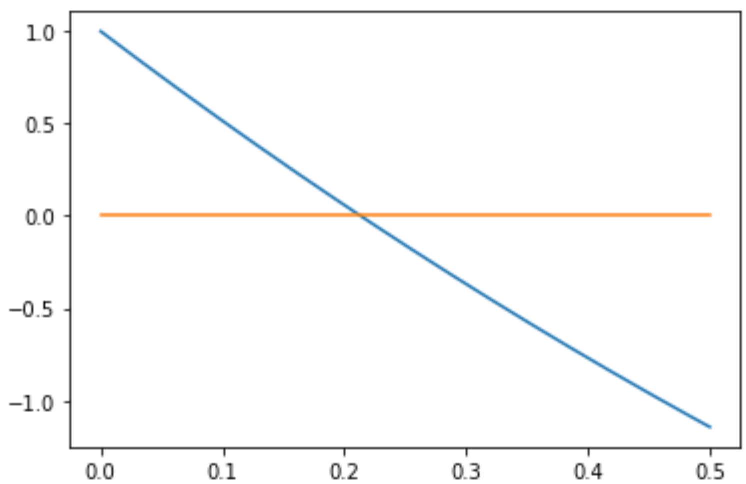
In [27]:
```
f = lambda x : x**2 - 4*x + math.e**(-x)

x = np.linspace( 0 , .5 , 1000 )
y = f(x)

plt.plot( x , y )
plt.plot( x , np.zeros( np.size(x)))
plt.show

print( 'a root lies between 0 and .5')
```

```
a root lies between 0 and .5
```



In [28]:
```
def bisection( f , a , b, tor = 1e-6 , max_itr = 100 )  :
    if  f(a)*f(b) >0 :
        raise ValueError('The function must have opposite signs at th
e two ends.')
    for i in range( 1 , max_itr + 1 ):
        c = ( a + b )/2
        k = f(c)
        if abs(k ) < tor :
            return c , i
        if f(a)*k < 0 :
            b = c
        else :
            a = c
    raise ValueError('Did not converge within the iteration limit.')

x , i = bisection( f , 0 , .5 )
print( 'at x = ', x , 'f(x) = ' , f(x))
print( 'number of iterations : ' , i )
```

```
at x =  0.2133479118347168 f(x) =  7.031824250658403e-07
number of iterations :  20
```

Bonus Question

In [34]:
```
f2 = lambda x : 2*x - 4 - math.e**(-x)

def newton_rapson( f ,f2 , x0 , tor = 1e-6 ):
    x = x0
    i = 0
    while( abs( f(x)) > tor ) :
        x2= x - (f(x)/f2(x))
        i = i+ 1
        x = x2
    return x , i

x, i = newton_rapson( f , f2 , 0.0 )
print( 'at x = ' , x , 'f(x) =  ' , f(x))
print( 'number of iterations : ' , i )
```

```
at x =  0.2133480713032742 f(x) =   4.522134666729016e-09
number of iterations :  3
```

In [35]:
```
print( "thus NR method is much faster than  bisection method!")
```

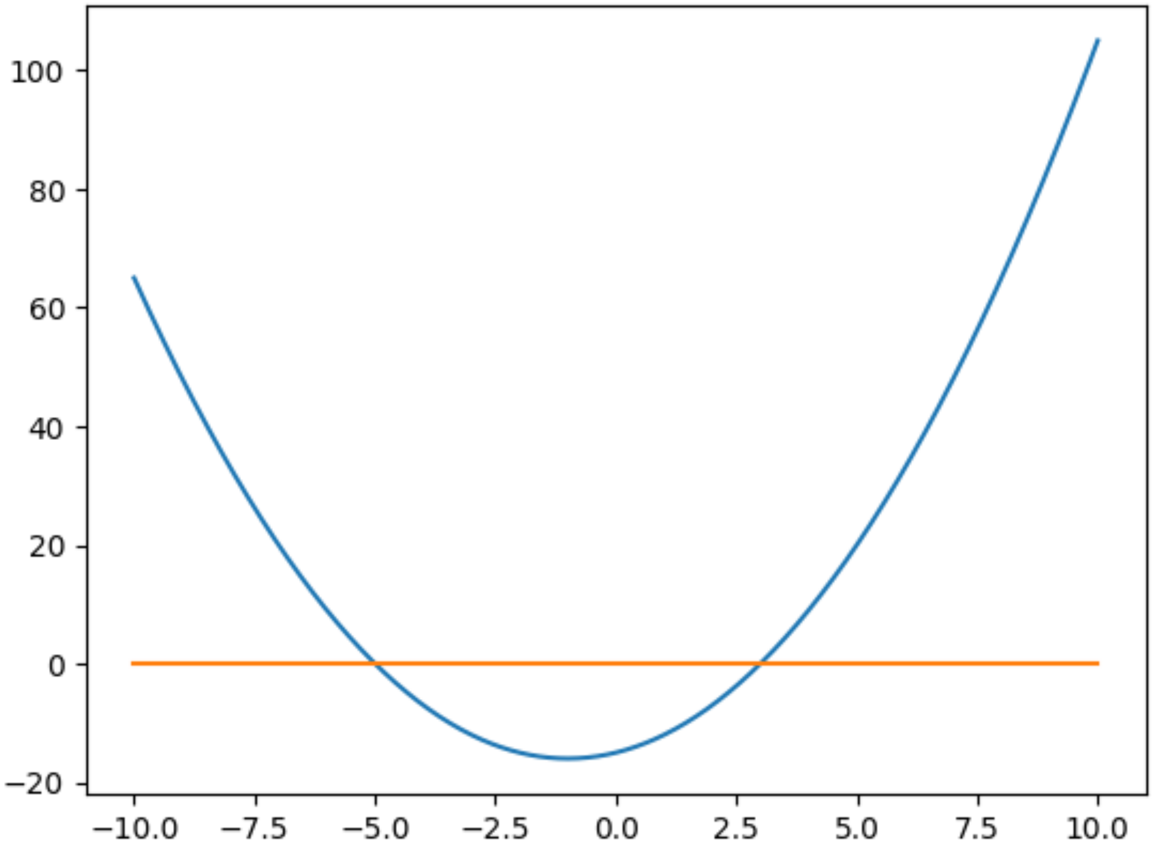
```
thus NR method is much faster than  bisection method!
```

In [28]:
```python
import math
import cmath
import numpy as np
import matplotlib.pyplot as plt
```

Question 1

In [29]:
```python
class quad:
    def __init__(self , a , b ,c ):
        if( a == 0 ):
            raise ValueError("a should not be zero.")
        self.a = a
        self.b = b
        self.c = c
    def value( self , x ):
        return self.a*(x**2) + self.b*(x) + self.c
    def table( self , l , r,  n ):
        if( r < l ):
            raise ValueError("l should be smaller than r.")
        x = np.linspace( l , r , n )
        return x , self.value( x )
    def roots( self ):
        const = self.b**2 - 4*self.a*self.c
        if( const >=0 ):
            const = math.sqrt( const )
        else:
            const = cmath.sqrt( const )
        return ( -self.b + const )/(2*self.a) , ( -self.b - const )/(2*se
```

In [30]:
```python
q1 = quad( 1 , 2 , -15 )
x , y = q1.table( -10 , 10 , 1000 )
plt.plot( x, y )
plt.plot( x, np.zeros(x.size))
plt.show()
print( 'roots ' , q1.roots())
```




```
roots  (3.0, -5.0)
```

In [31]:
```
q2 = quad( 3, -4 , 10 )
x , y = q2.table( -1 , 1 , 10 )
print( f'x = {x} \n and \n y = {y}')
print( 'roots ' , q2.roots())
```

```
x = [-1.         -0.77777778 -0.55555556 -0.33333333 -0.11111111  0.11111
111
  0.33333333  0.55555556  0.77777778  1.        ]
 and
 y = [17.         14.92592593 13.14814815 11.66666667 10.48148148  9.5925
9259
  9.          8.7037037   8.7037037   9.        ]
roots  ((0.6666666666666666+1.699673171197548j), (0.6666666666666666-1.6
99673171197548j))
```

Question 3

In [55]:
```python
class poly:
    def __init__(self, coeff ):
        if not isinstance(coeff, np.ndarray):
            raise ValueError("coeff must be a NumPy array")
        if not issubclass(coeff.dtype.type, np.floating):
            raise ValueError("coeff elements must be of type float")
        self.coeff = coeff
        self.power = self.coeff.size - 1
    def set_coeff( self , new_coeff ):
        if not isinstance(new_coeff, np.ndarray):
            raise ValueError("coeff must be a NumPy array")
        if not issubclass(new_coeff.dtype.type, np.floating):
            raise ValueError("coeff elements must be of type float")
        self.coeff = new_coeff.astype(float)
        self.power = self.coeff.size - 1
    def __add__( self , other ):
        if not isinstance( other , poly ):
            raise ValueError("Cannot add two different object types.")
        result = np.zeros( max( self.coeff.size , other.coeff.size ))
        result[:self.coeff.size] += self.coeff
        result[:other.coeff.size] += other.coeff
        return poly(result)
    def __call__( self , x ):
        z = np.array([ x**i for i in range( 0 , self.power + 1 )])
        return np.sum( self.coeff * z )
    def __mul__( self , other ):
        if not isinstance( other , poly ):
            raise ValueError("Cannot add two different object types.")
        result = np.zeros( self.power + other.power + 1  )
        for i in range( 0 , self.power + 1 ):
            for j in range( 0 , other.power + 1 ):
                result[ i + j ] += self.coeff[i]*other.coeff[j]
        return poly( result )
```

In [56]:
```python
p1 = poly( np.array([1.0,-1.0]))
print( f'p1 : {p1.coeff}' )
print( [ p1(i) for i in range( 0 ,5 )])
p2 = poly( np.array([0,1,0,0,-6,-1] , dtype = float ) )
print( f'p2 : {p2.coeff}')
print( [ p2(i) for i in range( 0 ,5 )])
p3 = p1 + p2
print( f'p3 : {p3.coeff}')
print( [ p3(i) for i in range( 0 ,5 )])
p4 = p1*p2
print( f'p3 : {p4.coeff}')
print( [ p4(i) for i in range( 0 ,5 )])
```

```
p1 : [ 1. -1.]
[1.0, 0.0, -1.0, -2.0, -3.0]
p2 : [ 0.  1.  0.  0. -6. -1.]
[0.0, -6.0, -126.0, -726.0, -2556.0]
p3 : [ 1.  0.  0.  0. -6. -1.]
[1.0, -6.0, -127.0, -728.0, -2559.0]
p3 : [ 0.  1. -1.  0. -6.  5.  1.]
[0.0, 0.0, 126.0, 1452.0, 7668.0]
```

In [81]:
```
def mod( a ):
    if a > 0 :
        return '+ '
    else:
        return '- '
```

Question 4

In [82]:
```
class polynomial:
    def __init__(self, coeff ):
        self.coeff = coeff
    def __add__( self , other ):
        result = {}
        for key, value in self.coeff.items():
            if key in result:
                result[key] += value
            else:
                result[key] = value
        for key, value in other.coeff.items():
            if key in result:
                result[key] += value
            else:
                result[key] = value
        return polynomial( result )
    def __sub__( self , other ):
        result = {}
        for key, value in self.coeff.items():
            if key in result:
                result[key] += value
            else:
                result[key] = value
        for key, value in other.coeff.items():
            if key in result:
                result[key] -= value
            else:
                result[key] = -value
        return polynomial( result )
    def __str__(self) :
        result = ''
        for key , value in self.coeff.items():
            result = result + mod(value) + str(abs(value)) + "x^" + str(k
        return result
```

In [86]:
```
p1_dict = { 4 : 1 , 2 : -3 , 0 : 3 }
p2_dict = { 9 : 11 , 7 : 5 , 3 : 4 , 1 : -2 }
p1 = polynomial( p1_dict)
p2 = polynomial( p2_dict )
print( p1 )
print( p2 )
print( p1 + p2 )
print( p1 - p2 )
```

```
+ 1x^4 - 3x^2 + 3x^0
+ 11x^9 + 5x^7 + 4x^3 - 2x^1
+ 1x^4 - 3x^2 + 3x^0 + 11x^9 + 5x^7 + 4x^3 - 2x^1
+ 1x^4 - 3x^2 + 3x^0 - 11x^9 - 5x^7 - 4x^3 + 2x^1
```

Linear Interpolation

between two points

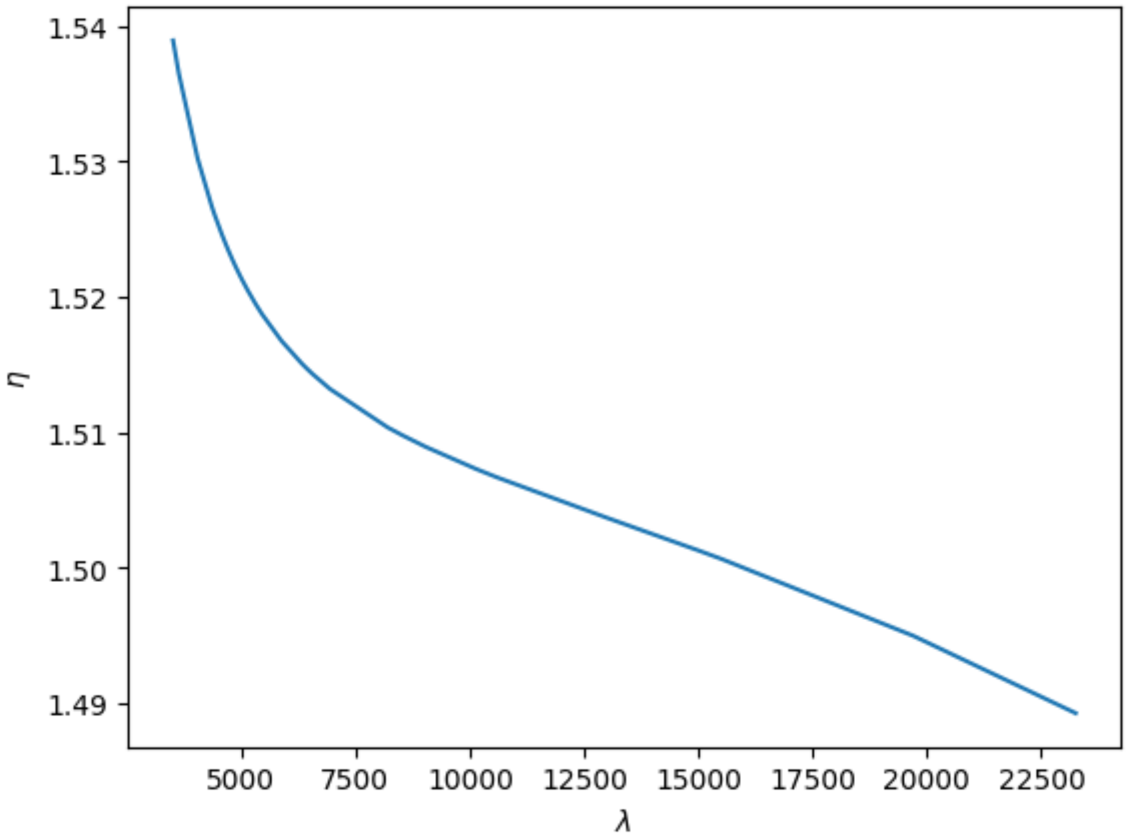
$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

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

In [28]:
```
data = np.loadtxt('data.txt', skiprows = 1 )
l = data[: , 0] # wavelength
r = data[: , 1] # refractive index
print( f' l = {l} \n and \n r = {r}')
```

```
 l = [ 3511.  3638.  4047.  4358.  4416.  4579.  4658.  4727.  4765.  4800.
  4861.  4880.  4965.  5017.  5145.  5320.  5461.  5876.  5893.  6328.
  6438.  6563.  6943.  8210.  8300.  8521.  9040. 10140. 10600. 13000.
 15000. 15500. 19701. 23254.]
 and
 r = [1.53894 1.53648 1.53024 1.52669 1.52611 1.52462 1.52395 1.52339 1.5231
 1.52283 1.52238 1.52224 1.52165 1.5213  1.52049 1.51947 1.51872 1.5168
 1.51673 1.51509 1.51472 1.51432 1.51322 1.51037 1.51021 1.50981 1.50894
 1.50731 1.50669 1.50371 1.5013  1.50068 1.495   1.48929]
```

In [29]:
```
# plt.figure( figsize = ( 17   , 8 ))
plt.plot( l , r )
plt.xlabel(r'$\lambda$')
plt.ylabel(r'$\eta$')
plt.show()
```

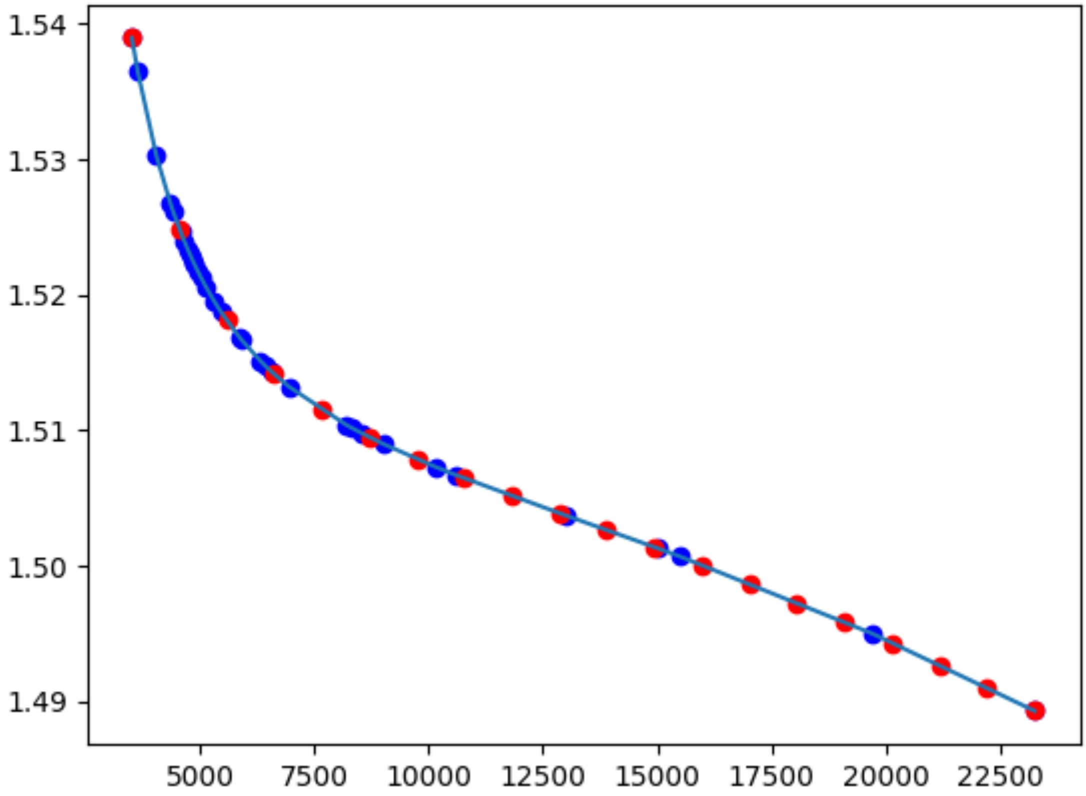


In [30]:
```
print( 'Shortes wavelength in the data = ' , np.min( l ))
print( 'Longest wavelength in the data = ' , np.max( l ))
```

```
Shortes wavelength in the data =  3511.0
Longest wavelength in the data =  23254.0
```

In [54]:
```
def linear_interpolate_vector( l , r , n ): # n = number of points
    l_li = np.linspace( l[0] , l[-1] , n )
    r_li = np.zeros( l_li.size )
    for i in range( l_li.size ):
        index = np.sum( np.array( l < l_li[i] , dtype = int )) - 1
        r_li[i] = r[index] + ( r[index + 1] - r[index] )*( l_li[i] - l[index
    return l_li , r_li  # wavelenght and refractive index
```

In [61]:
```
lnew , rnew = linear_interpolate_vector( l , r, 20  )
plt.plot( l , r )
plt.scatter( l , r , color = 'b')
plt.scatter( lnew , rnew , color = 'r')
plt.show()
```



Question 2

$$p_L(x) = \sum_{k=0}^{n} y_k L_k(x)$$

$$L_k(x) = \prod_{i \neq k}^{n} \frac{x - x_i}{x_k - x_i}$$

In [115…
```
def L( x , k , xp ):
    xk = xp[k]
    temp = np.delete( xp , k)
    return np.prod(np.divide( x - temp , xk - temp ))
# L = np.vectorize(L)
```

```python
def P( x , xp , yp ):
    ans = 0.0
    for k in range( xp.size ):
        ans += yp[k]*L( x , k , xp )
    return ans
def interpol( x , xp , yp ):
    y = np.zeros( x.size )
    for i in range( x.size ):
        y[i] = P( x[i] , xp , yp )
    return y
```

In [108…

```python
# def L2( x , k , xp ):
#     xk = xp[k]
#     temp = np.delete( xp , k)
#     return np.prod(np.divide( x - temp , xk - temp ))
# L2 = np.vectorize(L2 , excluded=['x' , 'xp'])
# def P2( x , xp , yp ):
#     ans = 0.0
#     k = np.arange( xp.size )
#     l = L( x , k , xp )
#     return np.sum( yp*l )
# P2 = np.vectorize( P2 , excluded=['xp' , 'yp'])
```

In [110…

```python
def testP( xp , yp ):
    result = np.zeros( xp.size )
    for i in range( xp.size ):
        result[i] = P( xp[i] , xp , yp ) - yp[i]
    return result
def testP2( xp , yp ):
    return P( xp , xp , yp ) - yp
```

In [116…

```python
x = np.linspace( 0 , np.pi , 5 )
print( 'size of x = '  , x.size )
y = np.sin( x )
print( testP( x , y ))
```

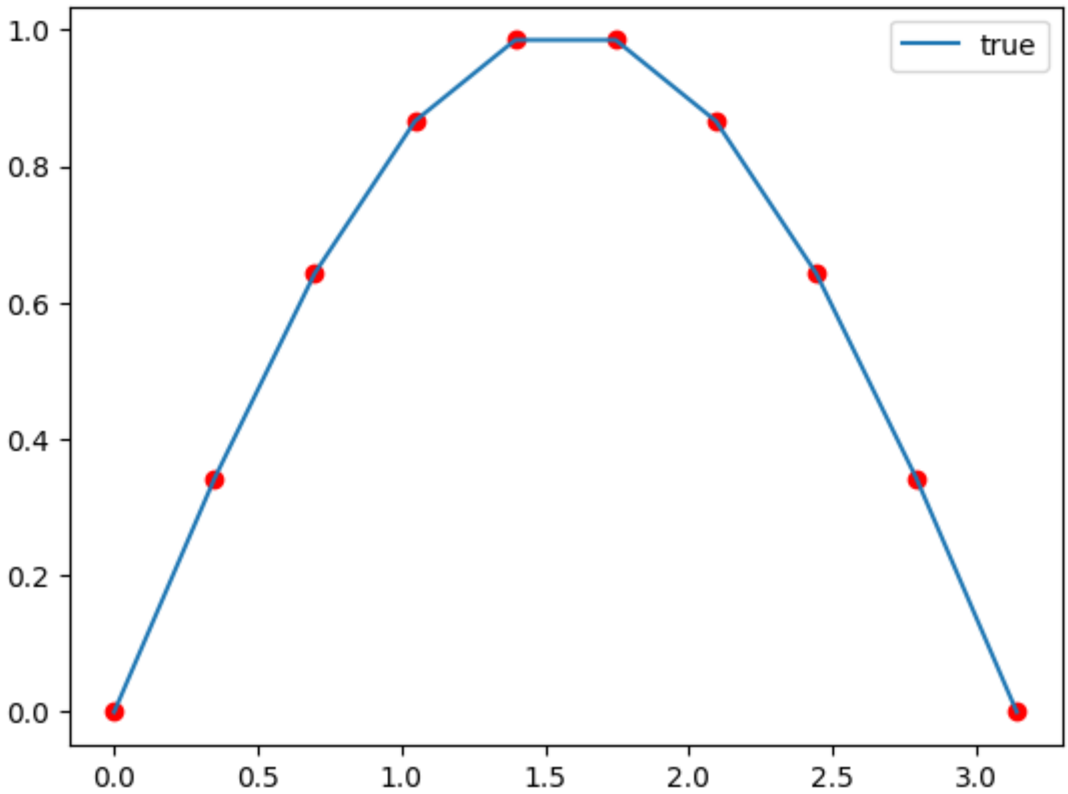
```
size of x =  5
[0. 0. 0. 0. 0.]
```

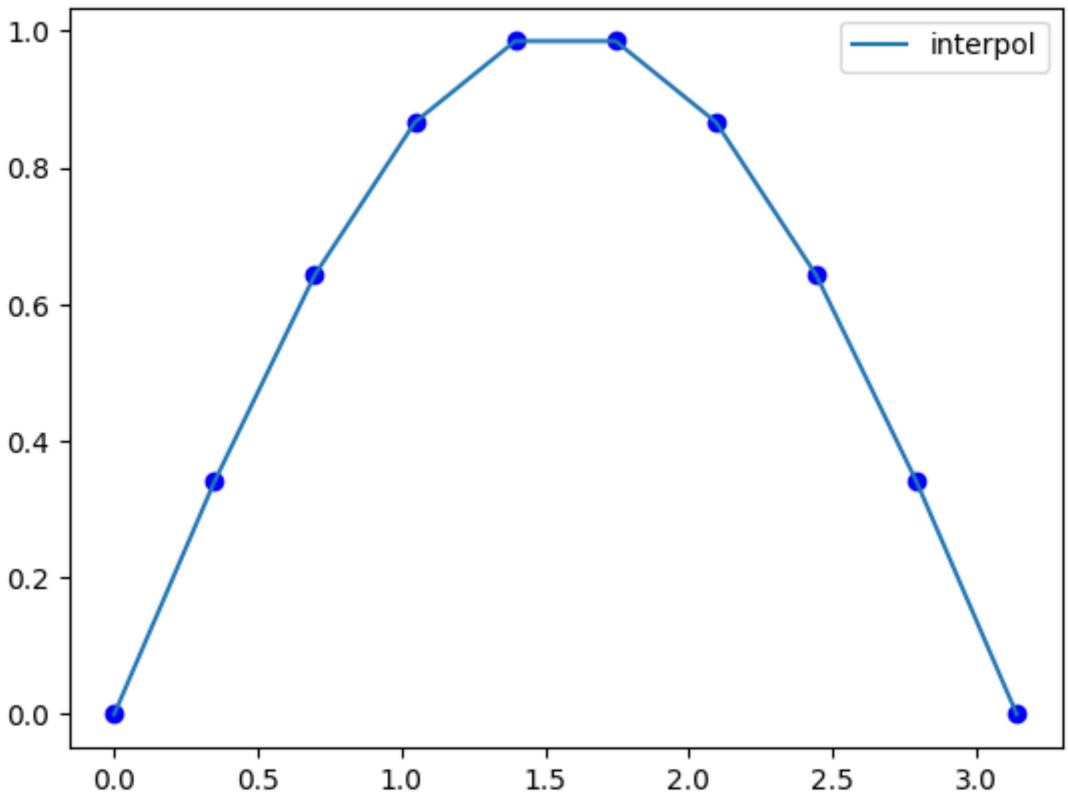
In [123…

```python
test_size = 10
test_x = np.linspace( 0 , np.pi , test_size )
true_y = np.sin( test_x )
inter_y = interpol( test_x , x , y )

plt.plot( test_x , true_y , label = r'true')
plt.scatter( test_x , true_y , color = 'red')
plt.legend()
plt.show()

plt.plot( test_x , inter_y , label = 'interpol')
plt.scatter( test_x , inter_y , color = 'b')
plt.legend()
plt.show()
```





In [124…

```
print( inter_y - true_y )
```

```
[ 0.00000000e+00 -1.63021667e-03 -2.43693689e-04  2.69526223e-04
  5.71903358e-05  5.71903358e-05  2.69526223e-04 -2.43693689e-04
 -1.63021667e-03  0.00000000e+00]
```

test

In [96]:

```
x = np.arange( 4 , 10 )
y = np.array( x < 6.5  , dtype = int  )
print( x )
```

```
print( y )
print( 'index = ' , np.sum( y ) - 1 )
for i in range( x.size ):
    print( x[i])
    print( x )
```

```
[4 5 6 7 8 9]
[1 1 1 0 0 0]
index =  2
4
[4 5 6 7 8 9]
5
[4 5 6 7 8 9]
6
[4 5 6 7 8 9]
7
[4 5 6 7 8 9]
8
[4 5 6 7 8 9]
9
[4 5 6 7 8 9]
```

In [97]:

```
print( np.prod( x ))
```

```
60480
```

In [94]:

```
print( x )
y = np.delete( x , 1 )
print(  y )
print( x )
```

```
[0.         0.78539816 1.57079633 2.35619449 3.14159265]
[0.         1.57079633 2.35619449 3.14159265]
[0.         0.78539816 1.57079633 2.35619449 3.14159265]
```

In [93]:

```
xp = [ 0 , 1 , 2 , 3 ]
xp = np.array( xp )

k = np.arange( 0 , xp.size  )

print( L( 2 ,  3 , xp ))
print( x )
```

```
0.0
[0.         0.78539816 1.57079633 2.35619449 3.14159265]
```