In [54]:
```
%run my_functions_library.ipynb                             # executing my library file

import math
import matplotlib.pyplot as plt                             # to be used for plotting
```

Question 1

In [55]:
```
def func_1(x):
    return math.sqrt(1+1/x)

eps, p, q =10**-6, 1, 4

MP=[]                                                                # creating list for mid-point
TR=[]                                                                # creating list for mid-point
SMP=[]                                                               # creating list for mid-point

##############################################################################################################################
# Below, I'm adding elements to the above created lists

n1=8
MP.append(int_mid_point(func_1, p, q, n1))
TR.append(int_trapezoidal(func_1, p, q, n1))
SMP.append(int_simpson(func_1, p, q, n1))

n2=16
MP.append(int_mid_point(func_1, p, q, n2))
TR.append(int_trapezoidal(func_1, p, q, n2))
SMP.append(int_simpson(func_1, p, q, n2))

n3=24
MP.append(int_mid_point(func_1, p, q, n3))
TR.append(int_trapezoidal(func_1, p, q, n3))
SMP.append(int_simpson(func_1, p, q, n3))

##############################################################################################################################
# Here I am calculating the actual value of integral


fn_mp=0.619                                   # for func_1   # feed here maximum of second derivative of function for Mid-point
fn_t=0.619                                    # for func_1   # feed here maximum of second derivative of function for trapezoidal
fn_s=6.016                                    # for func_1   # feed here maximum of fourth derivative of function for simpson

N_mp, N_t, N_s = calculate_N(fn_mp, fn_t, fn_s)

MP.append(int_mid_point(func_1, p, q, N_mp))
TR.append(int_trapezoidal(func_1, p, q, N_t))
SMP.append(int_simpson(func_1, p, q, N_s))
##############################################################################################################################


# printing the values

print ("{:<30} {:<30} {:<30} {:<30}".format('No. of iterations', 'Mid-point', 'Trapezoidal', 'Simpson'))
print()
print ("{:<25} {:<32} {:<30} {:<25}".format(n1, MP[0], TR[0], SMP[0]))
print ("{:<25} {:<32} {:<30} {:<25}".format(n2, MP[1], TR[1], SMP[1]))
print ("{:<25} {:<32} {:<30} {:<25}".format(n3, MP[2], TR[2], SMP[2]))
print()
print ("{:<25} {:<32} {:<30} {:<25}".format('True value', MP[3], TR[3], SMP[3]))
```

```
No. of iterations              Mid-point                      Trapezoidal                    Simpson

8                         3.6183138593298727               3.623956949398562              3.6203301434402904
16                        3.619709761707181                3.6211354043642174             3.6201948893527693
24                        3.619972785533525                3.620607687124767              3.620186449815972

Actual value              3.6201836884000755               3.6201848732718958             3.6201849759261933
```

Question 2

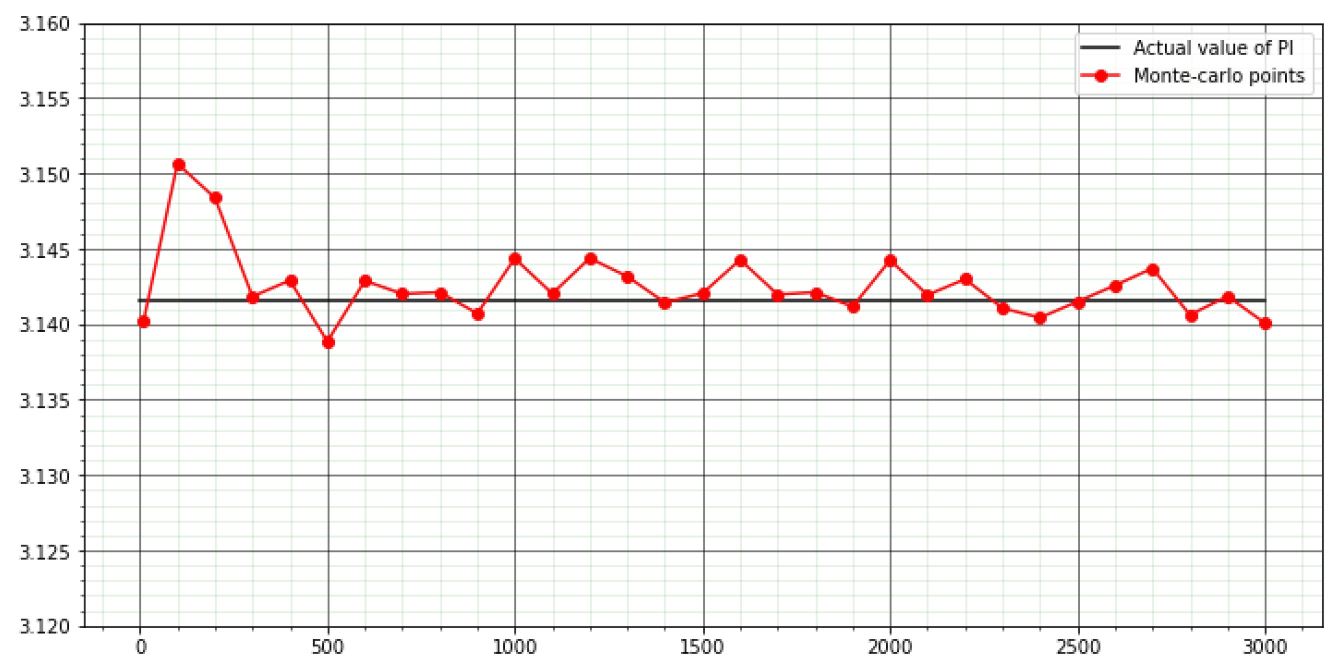
In [77]:
```
def func_2(x):
    return x*math.sqrt(1+x)

eps, p, q = 10**-4, 0, 1


fn_mp = 1                                    # for func_2   # feed here maximum of second derivative of function for Mid-point
fn_t = 1                                     # for func_2   # feed here maximum of second derivative of function for trapezoidal
fn_s = 1.5                                   # for func_2   # feed here maximum of fourth derivative of function for simpson

N_mp, N_t, N_s = calculate_N(fn_mp, fn_t, fn_s, eps)

MP=(int_mid_point(func_2, p, q, N_mp))
TR=(int_trapezoidal(func_2, p, q, N_t))
SMP=(int_simpson(func_2, p, q, N_s))

print("Simpson :" )
print(" N = " + str(N_s) , "  ;   integral = " + str(SMP) ,"\n")
print("Trapezoidal :" )
print(" N = " + str(N_t) , "  ;   integral = " + str(TR) , "\n")
print("Mid-point : ")
print("N = " + str(N_mp) , "  ;   integral = " + str(MP) , "n")
```

```
Simpson :
 N = 4   ;   integral = 0.6438016157592887

Trapezoidal :
 N = 28   ;   integral = 0.6438718899363646

Mid-point :
N = 20   ;   integral = 0.643710311759088 n
```

Question 3

In [70]:
```
def func_3(x):
    return 4/(1+x**2)

print("Please wait\n Processing will take around 10 seconds")
plt.figure(figsize=(12,6))

p, q, N = 0, 1, 10

NN=[]
MC=[]
NN.append(N)
MC.append(int_monte_carlo(func_3,pdf,p,q,N))


x=[0,3000]
y=[math.pi,math.pi]
plt.plot(x,y,'k-', label='Actual value of PI')

for N in range(100, 3001, 100):
    NN.append(N)
    MC.append(int_monte_carlo(func_3,pdf,p,q,N))
plt.plot(NN,MC,'r-o', label='Monte-carlo points')
print("The value of the integral in the last iteration is = " + str(MC[-1]))

##############################################################################################################################
'''
One could also use N=10 and proceed with a step size of 10 as instructed in question 3 but
that will not give any significant improvement.
Hence, I made the plot with step size=100 upto 5000 taking a total of 50 points.
If one still wishes to use step-size = 10 and work for 100 points, then the following code should be used :

x=[0,500]
y=[math.pi,math.pi]
plt.plot(x,y,'r-', label='Actual value of PI')

for N in range(10, 501, 10):
    NN.append(N)
    MC.append(int_monte_carlo(f3,pdf,a,b,N))
plt.plot(NN,MC,'b-o', label='Monte-carlo points')
print("The value of the integral in the last iteration is = " + str(MC[-1]))

'''
##############################################################################################################################

plt.grid(b=True, which='major', color='k', alpha=1, ls='-', lw=0.5)
plt.minorticks_on()
plt.grid(b=True, which='minor', color='g', alpha=0.2, ls='-', lw=0.5)
plt.ylim(3.12,3.16)
plt.legend()
plt.show()
```

```
Please wait
 Processing will take around 10 seconds
The value of the integral in the last iteration is = 3.140063418670694
```



Question 4

In [78]:
```
def func_4(x):
    return x**3

def func_5(x):
    return x**2

eps, p, q = 10**-6, 0, 2

print("Linear mass density = T = x^2")
print("Centre of mass = integral(xT) dx / integral(T) dx")

##############################################################################################################################

f4_mp = 12                                   # for func_4   # feed here maximum of second derivative of function for Mid-point
f4_t = 12                                    # for func_4   # feed here maximum of second derivative of function for trapezoidal
f4_s = 0                                     # for func_4   # feed here maximum of fourth derivative of function for simpson


N_mp, N_t, N_s = calculate_N(f4_mp, f4_t, f4_s, eps)

MP1=(int_mid_point(func_4, p, q, N_mp))
TR1=(int_trapezoidal(func_4, p, q, N_t))
SMP1=(int_simpson(func_4, p, q, N_s))

##############################################################################################################################

fn_mp = 2                                   # for func_5   # feed here maximum of second derivative of function for Mid-point
fn_t = 2                                    # for func_5   # feed here maximum of second derivative of function for trapezoidal
fn_s = 0                                    # for func_5   # feed here maximum of fourth derivative of function for simpson


N_mp, N_t, N_s = calculate_N(f5_mp, f5_t, f5_s, eps)

MP2=(int_mid_point(func_5, p, q, N_mp))
TR2=(int_trapezoidal(func_5, p, q, N_t))
SMP2=(int_simpson(func_5, p, q, N_s))

##############################################################################################################################

print("\n Using Mid-point, the centre of mass = " + str(MP1/MP2))
print("\n Using Trapezoidal, the centre of mass = " + str(TR1/TR2))
print("\n Using Simpson, the centre of mass = " + str(SMP1/SMP2))
```

```
linear mass density = T = x^2
Centre of mass = integral(xT) dx / integral(T) dx

 Using Mid-point, the centre of mass = 1.5000030206786539

 Using Trapezoidal, the centre of mass = 1.4999969945303646

 Using Simpson, the centre of mass = 1.5
```

In [ ]: