In [8]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                        # importing does not work with JupyterLab
# Chandan Kumar Sahu - 1911055
```

P346 Comp Phy Lab - End-sem exam

Date - 21 Nov 2021

Chandan Kumar Sahu - 1911055

Question 1

In [9]:

```
def f1(x):
    # Van der waal's equation of state # Chandan Kumar Sahu - 1911055
    return (P + a/x**2)*(x-b)-R*T

eps=10**-5

# Initialising given values
T=300
P=5.95
R=0.0821
a=6.254
b=0.05422

# calculating guess using ideas gas equation  =>  PV = RT   =>   V = RT/V
guess=R*T/P  # Chandan Kumar Sahu - 1911055
print("Initial guess using ideal gas law = "+str(guess))
print("\nNEWTON RAPHSON METHOD")

# calling Newton-Raphson function
root=newton_raphson(guess,f1)
print("Initial volume of real gas obtained from Newton raphson method is:\nV_o =  "+str(root))
# Chandan Kumar Sahu - 1911055
```

```
Initial guess using ideal gas law = 4.139495798319328

NEWTON RAPHSON METHOD
Initial volume of real gas obtained from Newton raphson method is:
V_o =  3.9299487677798326
```

Question 2

In [10]:

```
def f2(x):
    # Initialising given values
    L=4  # Chandan Kumar Sahu - 1911055
    return math.exp(-x**2/L**2)

def f3(x):
    # Initialising given values
    L=4
    l=1
    k=1
    d=1.5
    return k*f2(x)/math.sqrt(x**2+d**2)

eps=10**-4

# Initialising given values
L=4
l=1
# Chandan Kumar Sahu - 1911055

a=-1*l
b=L-l
N=12

# calling simpson function
SMP=int_simpson(f3, a, b, N)
print("For simpson method, with N = " + str(N) + ", we got the potential = " + str(SMP))
# Chandan Kumar Sahu - 1911055
```

```
For simpson method, with N = 12, we got the potential = 1.8728664575240428
```

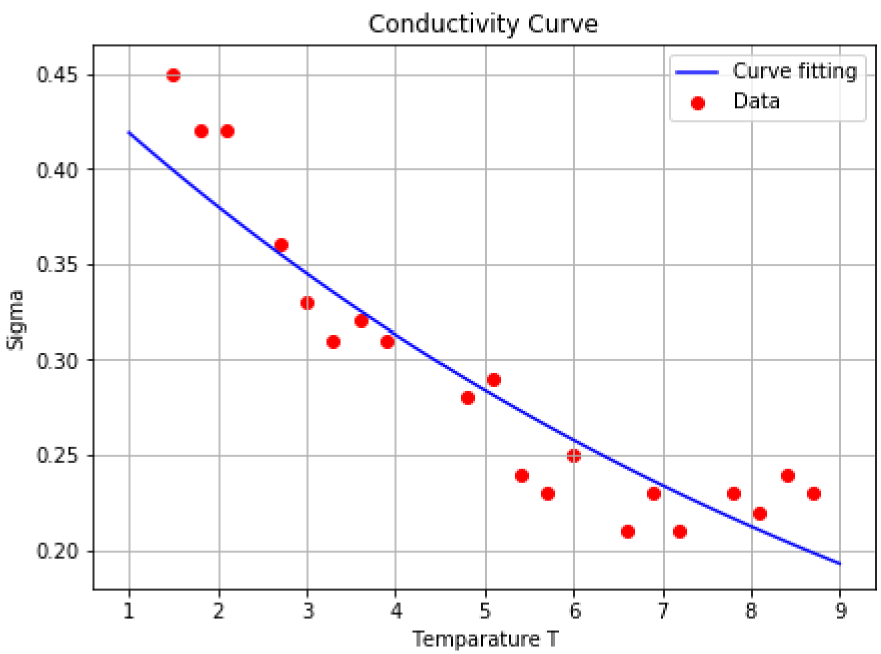
Question 3 (i) $\sigma = \sigma_{o} *e^{\alpha *T}$

In [11]:

```
plt.figure(figsize=(7,5))

# Importing data from csv file
file='CPL_data1.csv'  # Chandan Kumar Sahu - 1911055
x,y=get_from_csv(file)

# We take log here for fitting
for i in range(len(y)):
    y[i] = math.log(y[i])

# Curve fitting # Chandan Kumar Sahu - 1911055
c,m=Line_fit(x,y)

# Pearson coeff calculation
print("\nPearson coefficient = "+str(Pearson_coeff(x,y)))
print()

# Calculation of sigma_0 and alpha
sigma_0 = math.exp(c)
alpha = m
print("sigma_o  =  "+str(sigma_0))
print("alpha    =  "+str(alpha))
print()

# Reverting back to original set of values by taking exponent
for i in range(len(y)):
    y[i] = math.exp(y[i])
# Chandan Kumar Sahu - 1911055

# Plotting the best fit
t = np.linspace(1, 9, 100)
sigma = sigma_0 * np.exp(alpha* t)

plt.plot(t, sigma, 'b-', label='Curve fitting')
plt.scatter(x, y, color='r', label='Data')

plt.grid()
plt.xlabel("Temparature T")
plt.ylabel("Sigma")  # Chandan Kumar Sahu - 1911055
plt.title("Conductivity Curve")
```

```
plt.legend()
plt.show()
```

```
Pearson coefficient = 0.9231412967039353

sigma_o  =  0.462048311788727774
alpha    =  -0.09716764829979381
```



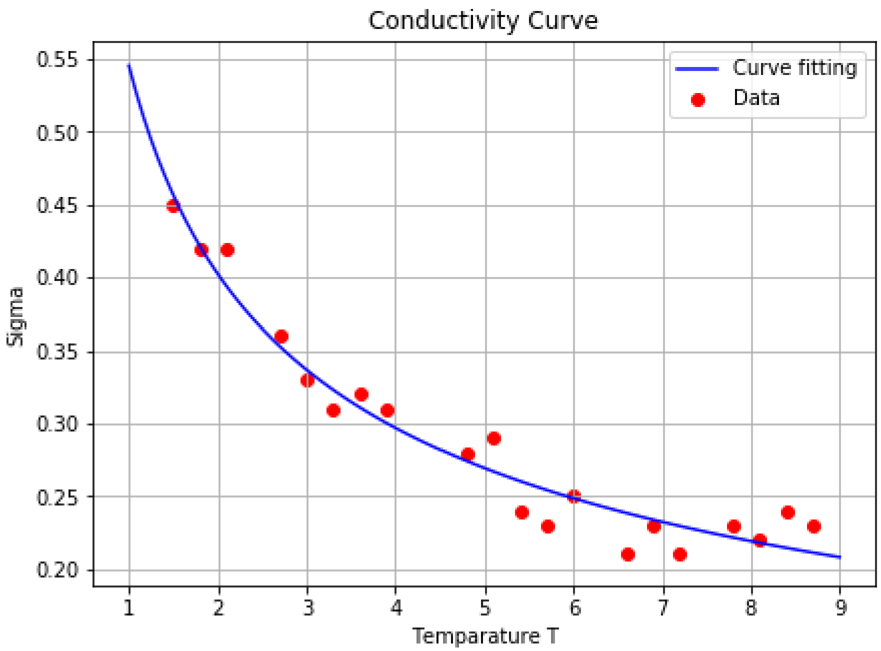
Question 3 (ii) $\sigma = \sigma_{o} *T^{\alpha}$

In [12]:

```
# Chandan Kumar Sahu - 1911055
plt.figure(figsize=(7,5))

# Importing data from csv file
file='CPL_data1.csv'
x,y=get_from_csv(file)

# We take log here for fitting
for i in range(len(y)):
    x[i] = math.log(x[i])
    y[i] = math.log(y[i])

# Curve fitting
c,m=Line_fit(x,y)

# Pearson coeff calculation  # Chandan Kumar Sahu - 1911055
print("\nPearson coefficient = "+str(Pearson_coeff(x,y)))
print()

# Calculation of sigma_0 and alpha
sigma_0 = math.exp(c)
alpha = m
print("sigma_o  =  "+str(sigma_0))
print("alpha    =  "+str(alpha))
print()

# Reverting back to original set of values by taking exponent
for i in range(len(y)):
    x[i] = math.exp(x[i])
    y[i] = math.exp(y[i])
# Chandan Kumar Sahu - 1911055

# Plotting the best fit
T = np.linspace(1, 9, 100)
sigma = sigma_0 * T**alpha

plt.plot(t, sigma, 'b-', label='Curve fitting')
plt.scatter(x, y, color='r', label='Data')

plt.grid()  # Chandan Kumar Sahu - 1911055
plt.xlabel("Temparature T")
```

```
plt.ylabel("Sigma")
plt.title("Conductivity Curve")
plt.legend()
plt.show()
```

```
Pearson coefficient = 0.964482406736276

sigma_o  =  0.5455160765174955
alpha    =  -0.43849688288816585
```

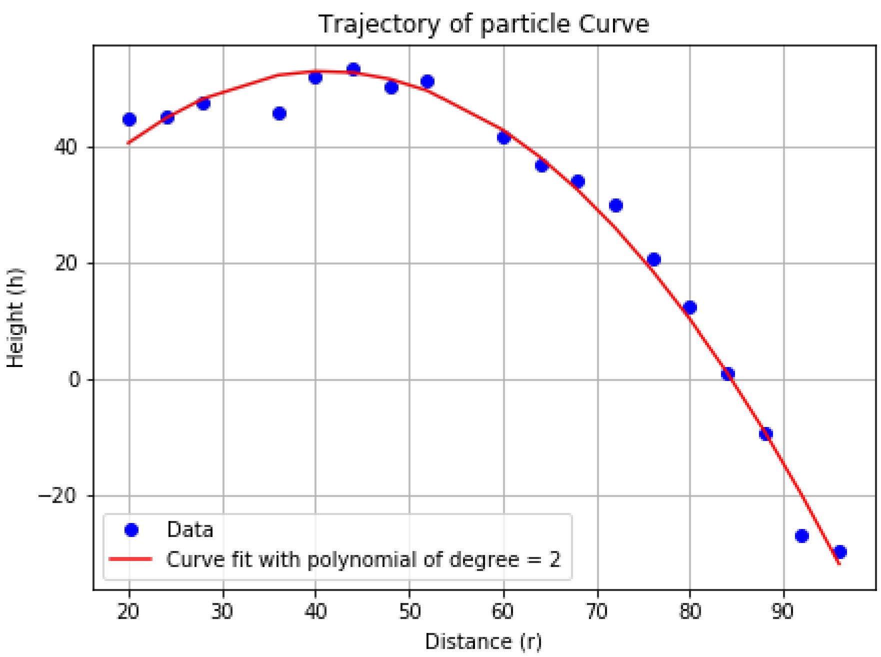


Question 4

In [13]:

```
plt.figure(figsize=(7,5))

# Importing data from csv file  # Chandan Kumar Sahu - 1911055
file='CPL_data2.csv'
x,y=get_from_csv(file)

# Degree of polynomial to fit
degree=2

# Curve fitting - finding solution coefficients
solution=polynomial_fitting(x,y,degree)

# plot points in array X
x,y=get_from_csv(file)  # Chandan Kumar Sahu - 1911055
plt.plot(x, y,'bo', label="Data")

# Curve fitting - plotting
plot_graph_poly(x,y, solution, degree+1)

print("Coefficients of the fitted polynomial are\n")
print(solution)
print()  # Chandan Kumar Sahu - 1911055

plt.grid()
plt.xlabel('Distance (r)')
plt.ylabel('Height (h)')
plt.title("Trajectory of particle Curve")
plt.legend()
plt.show()
# Chandan Kumar Sahu - 1911055
```

```
Determinant is = -584656400056320.0
Determinant is not zero. Inverse exists.

Coefficients of the fitted polynomial are

[5.7241062860246394, 2.31001829870581, -0.028162602079190243]
```



In [14]:

```
# So the value of h we got is 5.7241062860246394 + 2.310018298705810 R - 0.028162602079190243 R^2
# To calculate the h max, we take derivative          # Chandan Kumar Sahu - 1911055
# The analytical derivative is h' = 2.310018298705810 - 0.056325204158380486 R   which is equated to zero
# Eqution =   2.310018298705810 - 0.056325204158380486 R = 0

# Calculation of R and h_max
R=-solution[1]/(2*solution[2])
print("R at h_max is = "+str(R))
print()  # Chandan Kumar Sahu - 1911055
h_max=solution[0] + solution[1]*R + solution[2]*R**2
print("H_max is = "+str(h_max))
```

```
R at h_max is = 41.012160243756675

H_max is = 53.09352660229106
```

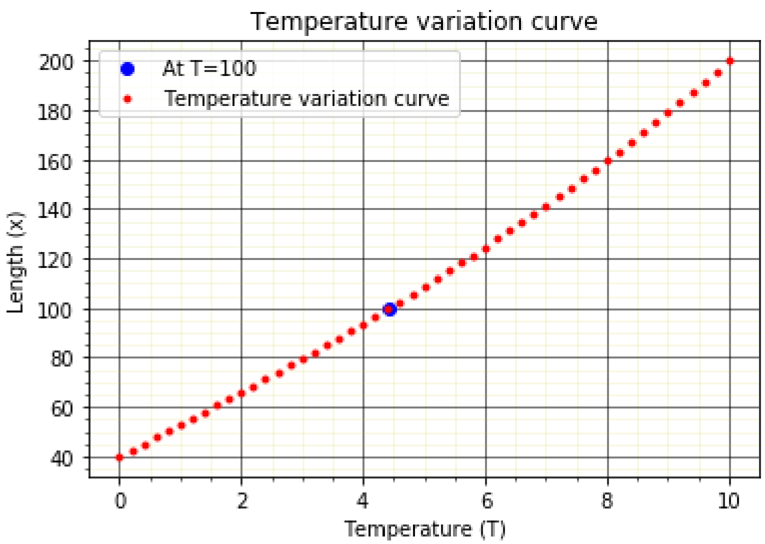
Question 5

In [15]:

```
# def shooting_method(d2ydx2, dydx, x0, y0, xf, yf, z_guess1, z_guess2, step_size, tol=1e-6):

# second order function
def d2ydt2(x, y, z):
    alpha = 10**-2
    Ta=20    # Chandan Kumar Sahu - 1911055
    return alpha * (y-Ta)

# first order function z = dy/dt
def dydt(x, y, z):
    return z

# Chandan Kumar Sahu - 1911055
# Define boundary values
x_initial = 0
x_final = 10
T_initial = 40
T_final = 200

# Calling shooting method
x, y, z = shooting_method(d2ydt2, dydt, x_initial, T_initial, x_final, T_final, 10, 300, step_size=0.2)


# Next 7 lines contain finding value of x at T=100 and fincding exact value
# by linear interpolation since this approximation is valid for such small interval
for i in range(len(y)):     # Chandan Kumar Sahu - 1911055
    if y[i]<=100 and y[i+1]>=100:
        p=i

q=x[p] + (100-y[p])/(y[p+1]-y[p])*(x[p+1]-x[p])
print("Value of x at T = 100 degrees is = "+str(q))
print("\nThis has been obtained from the solution array \nand the answer has been interpolated linearly to get proper val
ue\n")

# Plotting stuff    # Chandan Kumar Sahu - 1911055
plt.plot(q,100,'bo',label="At T=100")

plt.plot(x,y,'r.', label='Temperature variation curve')
plt.title('Temperature variation curve')
plt.grid(b=True, which='major', color='k', alpha=1, ls='-', lw=0.5)
plt.minorticks_on()
```

```
plt.grid(b=True, which='minor', color='y', alpha=0.2, ls='-', lw=0.5)
plt.xlabel('Temperature (T)')
plt.ylabel('Length (x)')
plt.legend()
plt.show()    # Chandan Kumar Sahu - 1911055
```

```
Value of x at T = 100 degrees is = 4.425142531789387

This has been obtained from the solution array
and the answer has been interpolated linearly to get proper value
```



In [ ]:

In [ ]:

```

```

In [ ]:

```

```