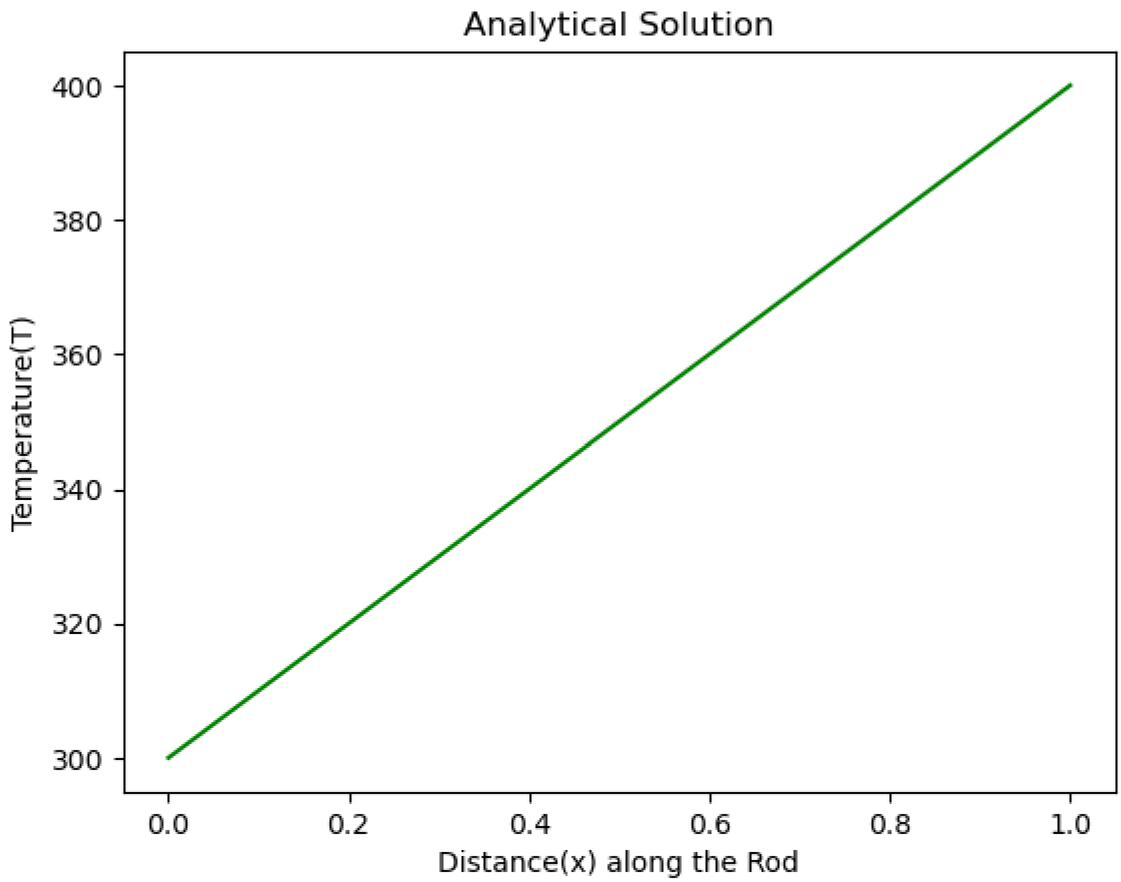
Assignment - 1

FEM and CFD Theory
ME3180

Name: Harshit Shambharkar
Roll No: ME21BTECH11019

Question - 1
Sol: The plot for Analytical solution is plotted below:




```
In [38]: T_analytical = 100*x + 300
         x_analytical = x
         plt.plot(x_analytical, T_analytical, 'g-')
         plt.title("Analytical Solution")
         plt.xlabel("Distance(x) along the Rod")
         plt.ylabel("Temperature(T)")
```

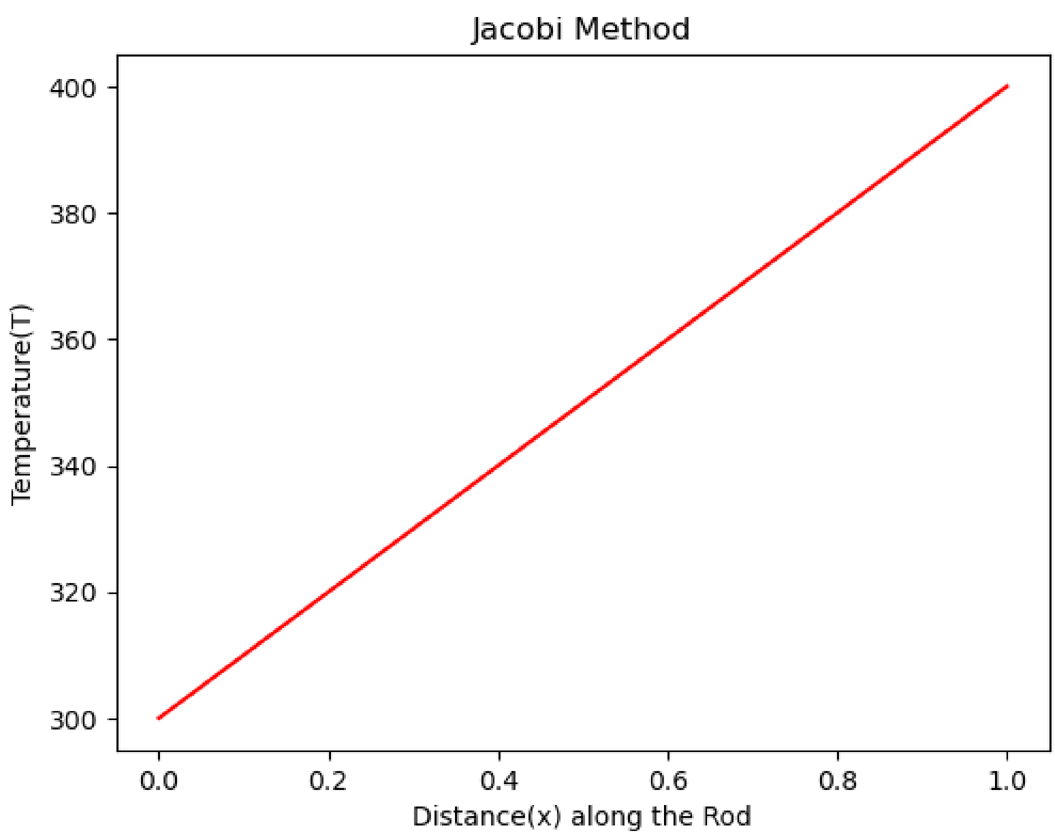
Plot for iterative methods like Jacobi, Gauss Siedel and direct method like TDMA is plotted below(compared with Analytical also).

<u>Jacobi Method</u>

In [5]:

```python
T_j = np.zeros(n)
#Intialising Boundary Conditions
T_j[0] = Ta
T_j[n-1] = Tb

T_old_j = np.copy(T_j)

#This keep track of number of iterations perfomed by the algorithm
iterations = 0
Error = 1

while Error > Tolerance:
    for i in range(1,n-1):
        T_j[i] = 0.5*(T_old_j[i-1] + T_old_j[i+1])

    Error = max(abs(T_j - T_old_j))
    #print(T)
    #print(T_old)
    T_old_j = np.copy(T_j)
    iterations = iterations + 1

plt.plot(x, T_j, 'r-')

plt.title("Jacobi Method")
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
print("No. of Iterations in Jacobi Method: ", iterations)
```

```
No. of Iterations in Jacobi Method:  751
```

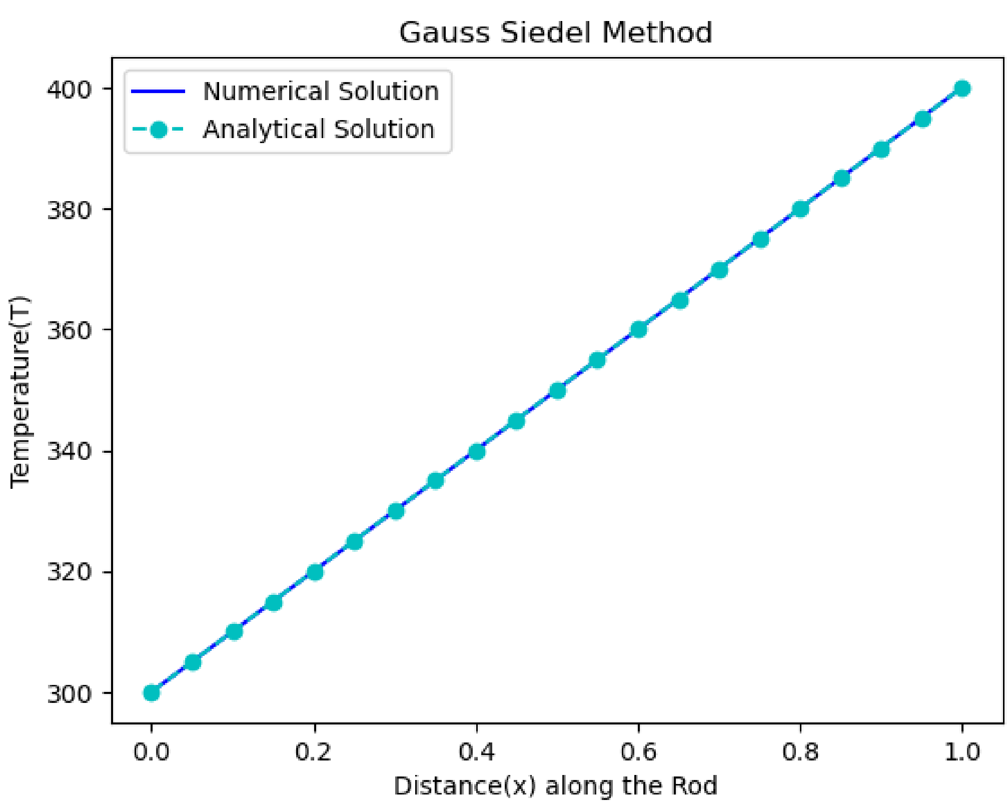


The number of iterations in Jacobi method are: 751

Gauss Siedel Method

In [7]:
```
T_gs = np.zeros(n)
T_gs[0] = Ta
T_gs[n-1] = Tb

T_old_gs = np.copy(T_gs)
iterations = 0
Error = 1

# An error array is made to store the error(deviation of numerical result from actual)
Errors_gs = []
iterate_gs = []

while Error > Tolerance:
    for i in range(1,n-1):
        T_gs[i] = 0.5*(T_gs[i-1] + T_old_gs[i+1])
    Error = max(abs(T_gs - T_old_gs))
    #print(T)
    #print(T_old)
    iterations = iterations + 1
    iterate_gs.append(iterations)
    Errors_gs.append((linalg.norm(T_gs - T_old_gs, 2)) / linalg.norm(T_old_gs, 2))
    T_old_gs = np.copy(T_gs)

plt.plot(x, T_gs, 'b-')
plt.title("Gauss Siedel Method")
plt.plot(x, T_analytical, 'c--o')
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
plt.legend(["Numerical Solution", "Analytical Solution"])

print("No. of Iterations in Gauss Siedel Method: ", iterations)
```

```
No. of Iterations in Gauss Siedel Method:  378
```

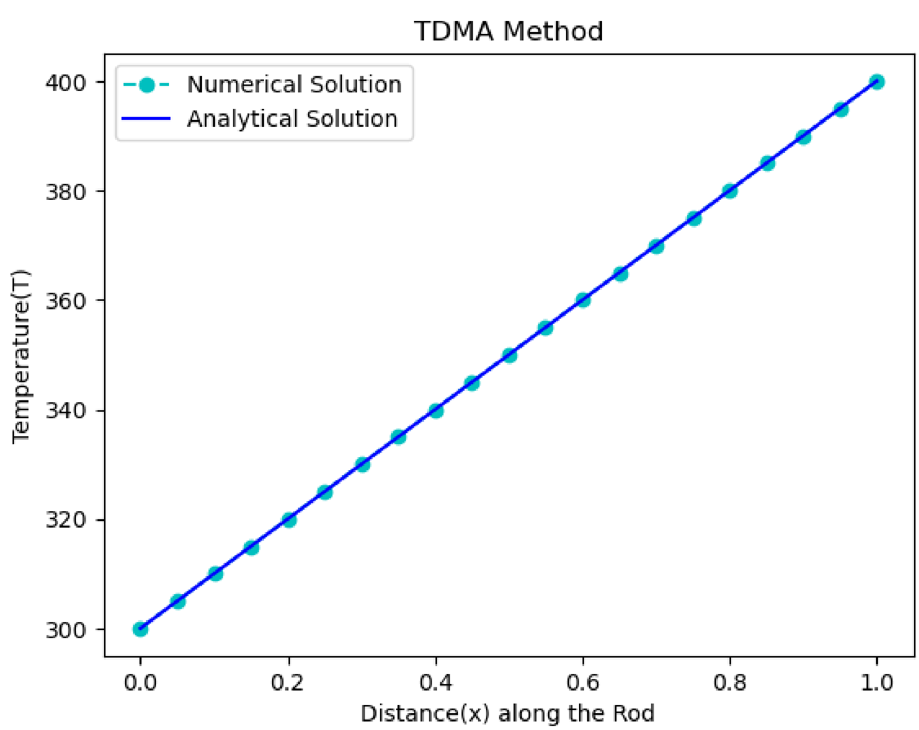


The number of iterations in Gauss Siedel method are: 378

Triadiagonal Matrix Algorithm(TDMA)

In [13]:
```
T_tdma = np.zeros(n)
T_tdma[0] = Ta
T_tdma[n-1] = Tb

P = np.zeros(n)
Q = np.zeros(n)

a, b, c, d = 2, 1, 1, 0

P[0] = 0
Q[0] = Ta

for i in range(1,n-1):
    P[i] = b / (a - c*P[i-1])
    Q[i] = (d + c*Q[i-1]) / (a - c*P[i-1])

Q[n-1] = T_tdma[n-1]

for i in range(n-2,-1, -1):
    T_tdma[i] = T_tdma[i+1]*P[i] + Q[i]

plt.plot(x, T_tdma, 'c--o')
plt.plot(x, T_gs, 'b-')
plt.title("TDMA Method")
plt.legend(["Numerical Solution", "Analytical Solution"])
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
```

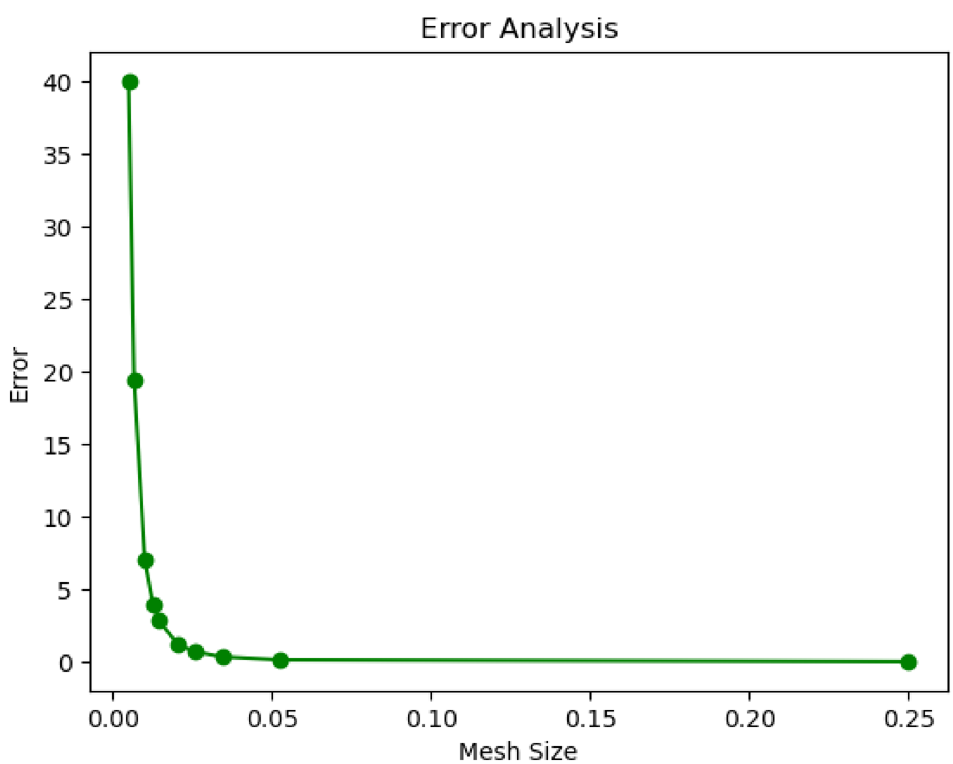
Out[13]:
```
Text(0.5, 0, 'Distance(x) along the Rod')
```



The below plot shows how the error goes down as the order of discretisation scheme

In [14]:
```
grid_points = [5, 20, 30, 40, 50, 70, 80, 100, 150, 200]
# Error(deviation from analytical solution) from one of the iterative method, gauss siedel is plotted below
Error_t = []
mesh_size = []

for grid_point in grid_points:

    x_t = np.linspace(0,L,grid_point,endpoint=True) #linspace(0, L, n)
    mesh_size.append(x_t[1] - x_t[0])
    # Gauss Siedel
    T_t = np.zeros(grid_point)
    T_t[0] = Ta
    T_t[grid_point-1] = Tb

    T_old_t = np.copy(T_t)
    iterations = 0
    Error = 1

    while Error > Tolerance:
        for i in range(1,grid_point-1):
            T_t[i] = 0.5*(T_t[i-1] + T_old_t[i+1])
        Error = max(abs(T_t - T_old_t))
        #print(T)
        #print(T_old)
        iterations = iterations + 1
        T_old_t = np.copy(T_t)

    T_analytic = 100*x_t + 300
    # Norm 2  is used
    Error_t.append((linalg.norm(T_t - T_analytic, 2)))

plt.plot(mesh_size, Error_t, 'g-o')
plt.title("Error Analysis")
plt.xlabel("Mesh Size")
plt.ylabel("Error")
```

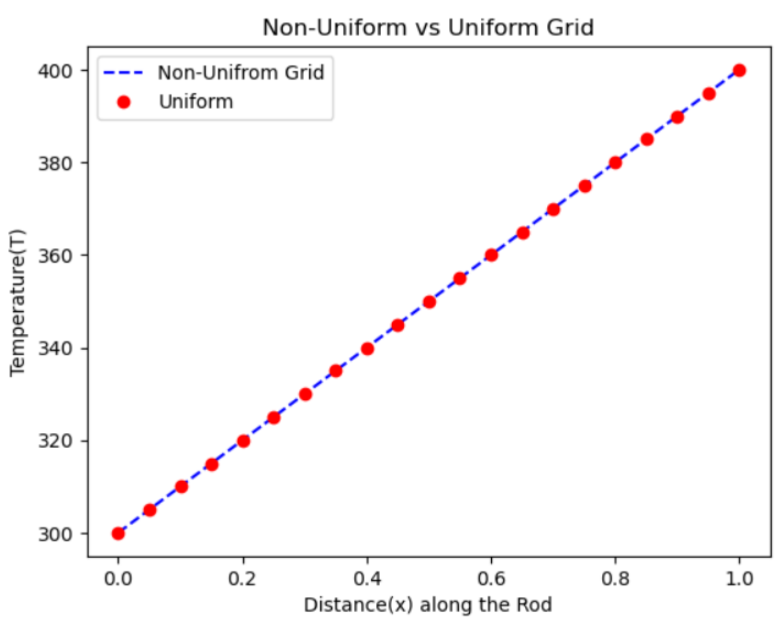
Out[14]:
```
Text(0, 0.5, 'Error')
```



Non - Uniform Grid

In [11]:
```
#Discritise domain with non uniform grid
# Function used for generating non - uniform grid as taught in class
# I have used Gauss Siedel Method to solve for non-uniform grid
p = 0.3
x_nu = np.zeros(n)

for i in range(1,n):
    x_nu[i] = L*((i + 1 - 1) / (n - 1))**p
# print(x)

T_nu = np.zeros(n)
T_nu[0] = Ta
T_nu[n-1] = Tb

T_old_nu = np.copy(T_nu)

iterations = 0
Error = 1
Errors_nu = []
iterate_nu = []

while Error > Tolerance:
    for i in range(1,n-1):
        R = (x_nu[i+1] - x_nu[i]) / (x_nu[i] - x_nu[i-1])
        T_nu[i] = (R*T_nu[i-1] + T_old_nu[i+1]) / (1 + R)
    Error = max(abs(T_nu - T_old_nu))
    #print(T)
    #print(T_old)
    iterations = iterations + 1
    iterate_nu.append(iterations)
    # Storing error for non uniform grid, l2 norm is used
    Errors_nu.append((linalg.norm(T_nu - T_old_nu, 2)) / linalg.norm(T_old_nu, 2))
    T_old_nu = np.copy(T_nu)


plt.plot(x_nu, T_nu, 'b--')
plt.plot(x, T_gs, 'ro')

plt.title("Non-Uniform vs Uniform Grid")
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
plt.legend(["Non-Unifrom Grid", "Uniform"])

print("No. of Iterations in Gauss Siedel Method with Non Uniform Grid: ", iterations)
```

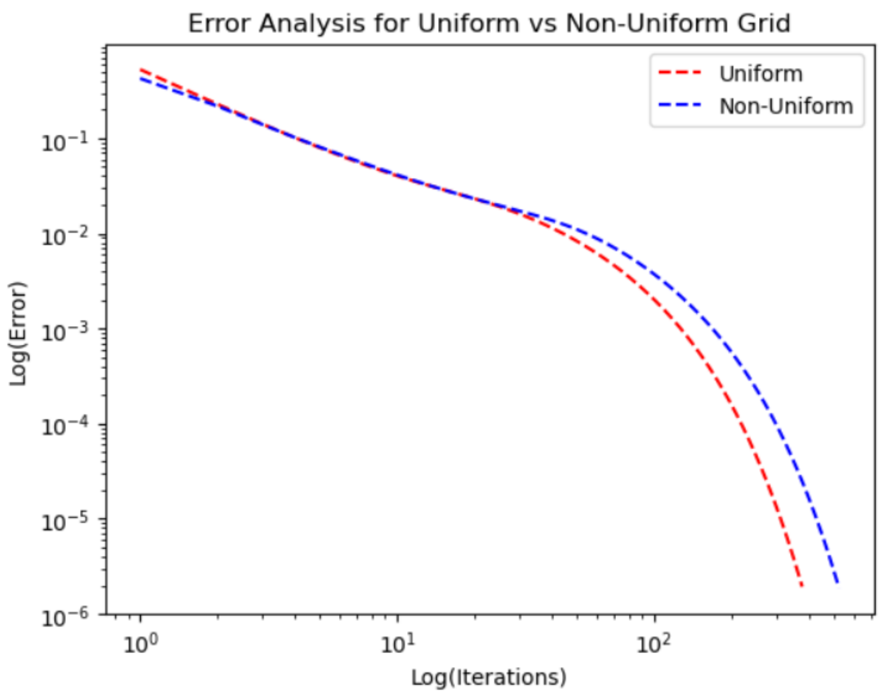
```
No. of Iterations in Gauss Siedel Method with Non Uniform Grid:  526
```



In [12]:
```
# Plotting of error(in log scale)
plt.xscale('log')
plt.yscale('log')
plt.plot((iterate_gs), (Errors_gs), 'r--')
plt.plot((iterate_nu), (Errors_nu), 'b--')
plt.title("Error Analysis for Uniform vs Non-Uniform Grid")
plt.legend(["Uniform", "Non-Uniform"])
plt.xlabel("Log(Iterations)")
plt.ylabel("Log(Error)")
#plt.ylim(0,3)
```

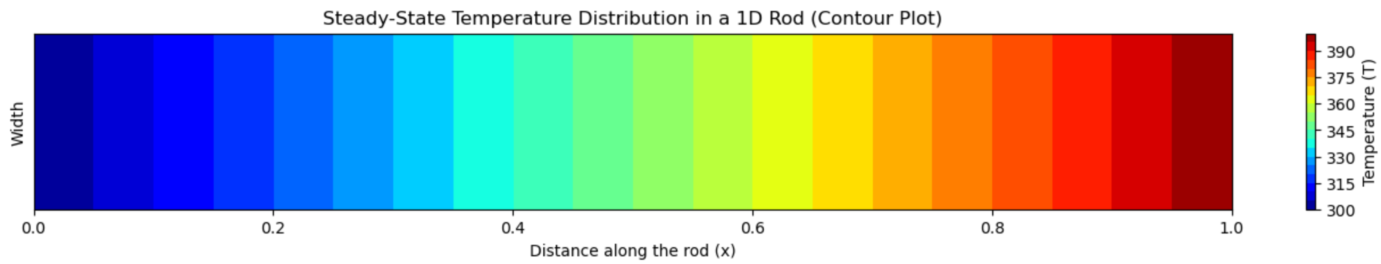
Out[12]: Text(0, 0.5, 'Log(Error)')

Out[12]: Text(0, 0.5, 'Log(Error)')



From above plot we can infer that uniform grid converges faster than non uniform grid, also as derived non-uniform is of O(dx) while uniform grid has discritisation of order O(dx**2).

Contour plot for Temperature distribution along the rod



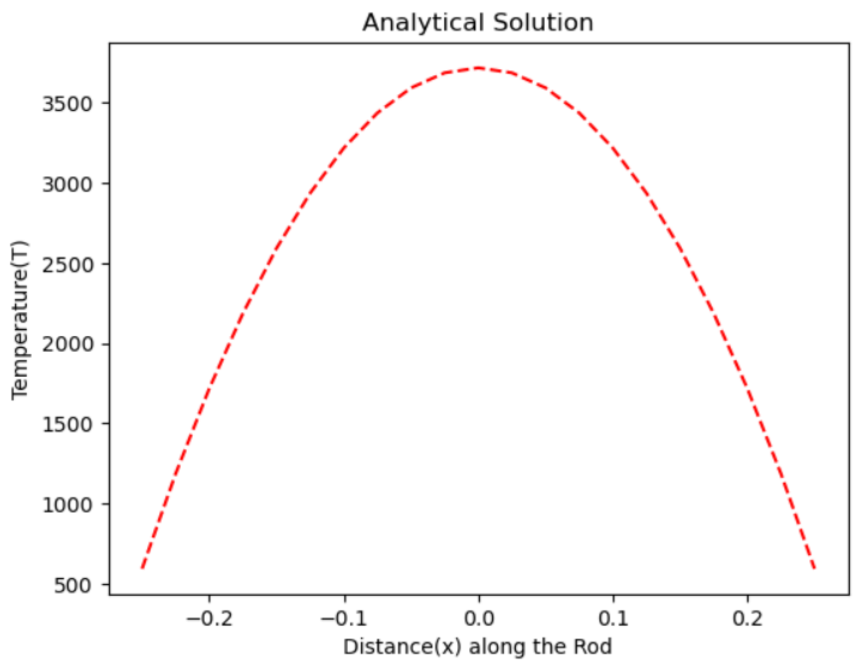
****

End of 1st Question

Question - 2

The plot for various method along with comparison of analytical solution is plotted below:

Analytical Solution

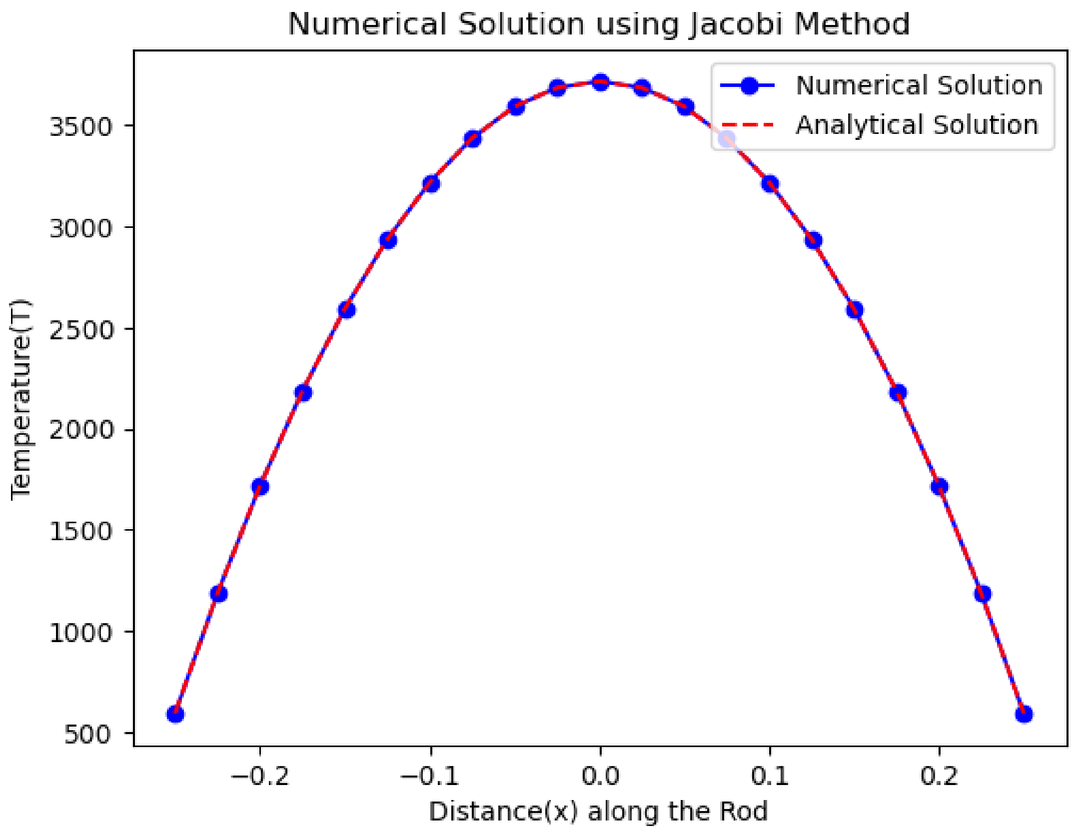


In [3]:
```
#Analytical Solution
T_surf = T_ambient + q_dot*L/h_conv #using Boundary Condition calculated T at surface
print(T_surf)
T_analytical = 0.5*q_dot*(L**2)*(1 - x**2/L**2)/k + T_surf

plt.plot(x, T_analytical, 'r--')
plt.title("Analytical Solution")
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
```

```
593.1818181818181
```

Out[3]:
```
Text(0.5, 0, 'Distance(x) along the Rod')
```

Jacobi Method

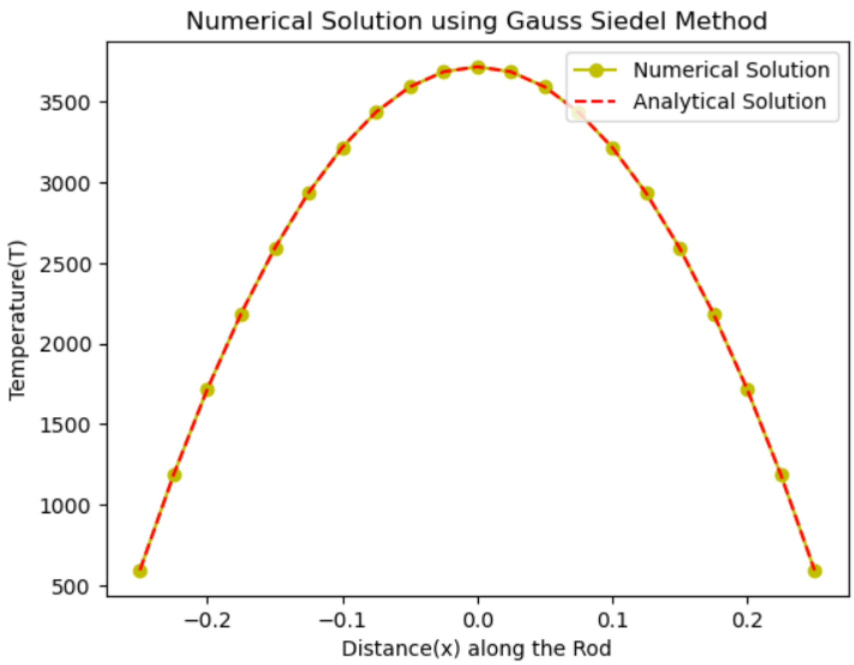


The number of iterations in Jacobi Method: 887

In [4]:
```
T_j = np.zeros(n)

C = (q_dot)*(h**2)/k

T_j[0] = T_surf
T_j[n-1] = T_surf

T_old_j = np.copy(T_j)
iterations = 0
Error = 1

while Error > Tolerance:

    for i in range(1,n-1):
        T_j[i] = 0.5*(C + T_old_j[i-1] + T_old_j[i+1])

    Error = max(abs(T_j - T_old_j))
    T_old_j = np.copy(T_j)
    iterations = iterations + 1

plt.plot(x, T_j, 'b-o')
plt.plot(x, T_analytical, 'r--')
plt.title("Numerical Solution using Jacobi Method")
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
plt.legend(["Numerical Solution", "Analytical Solution"], loc='upper right')

print("No. of Iterations in Jacobi Method: ", iterations)
```

```
No. of Iterations in Jacobi Method:  887
```

Gauss Siedel

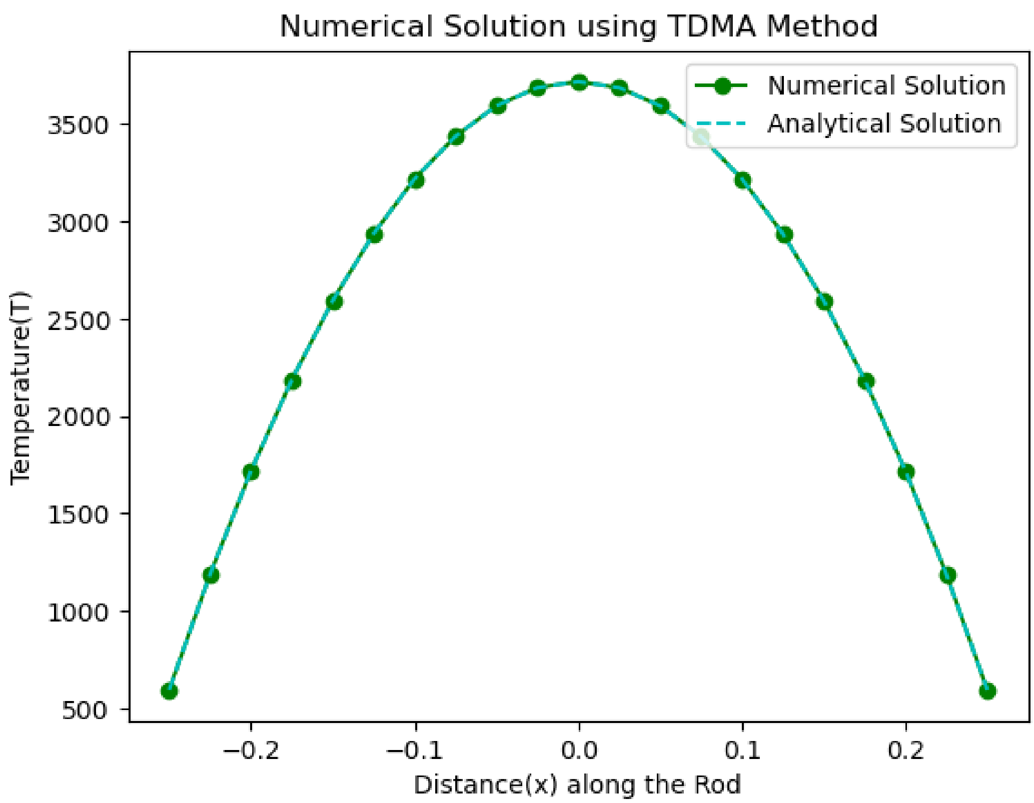


In [5]:
```
T_gs = np.zeros(n)
B = (q_dot/k)*np.ones(n)

T_gs[0] = T_surf
T_gs[n-1] = T_surf

T_old_gs = np.copy(T_gs)
iterations = 0
Error = 1

while Error > Tolerance:

    for i in range(1,n-1):
        T_gs[i] = 0.5*(C + T_gs[i-1] + T_old_gs[i+1])

    Error = max(abs(T_gs - T_old_gs))
    T_old_gs = np.copy(T_gs)
    iterations = iterations + 1


plt.plot(x, T_gs, 'y-o')
plt.plot(x, T_analytical, 'r--')
plt.title("Numerical Solution using Gauss Siedel Method")
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
plt.legend(["Numerical Solution", "Analytical Solution"], loc='upper right')

print("No. of Iterations in Gauss Siedel Method: ", iterations)
```

```
No. of Iterations in Gauss Siedel Method:  465
```

The number of iterations in Gauss Siedel are: 465

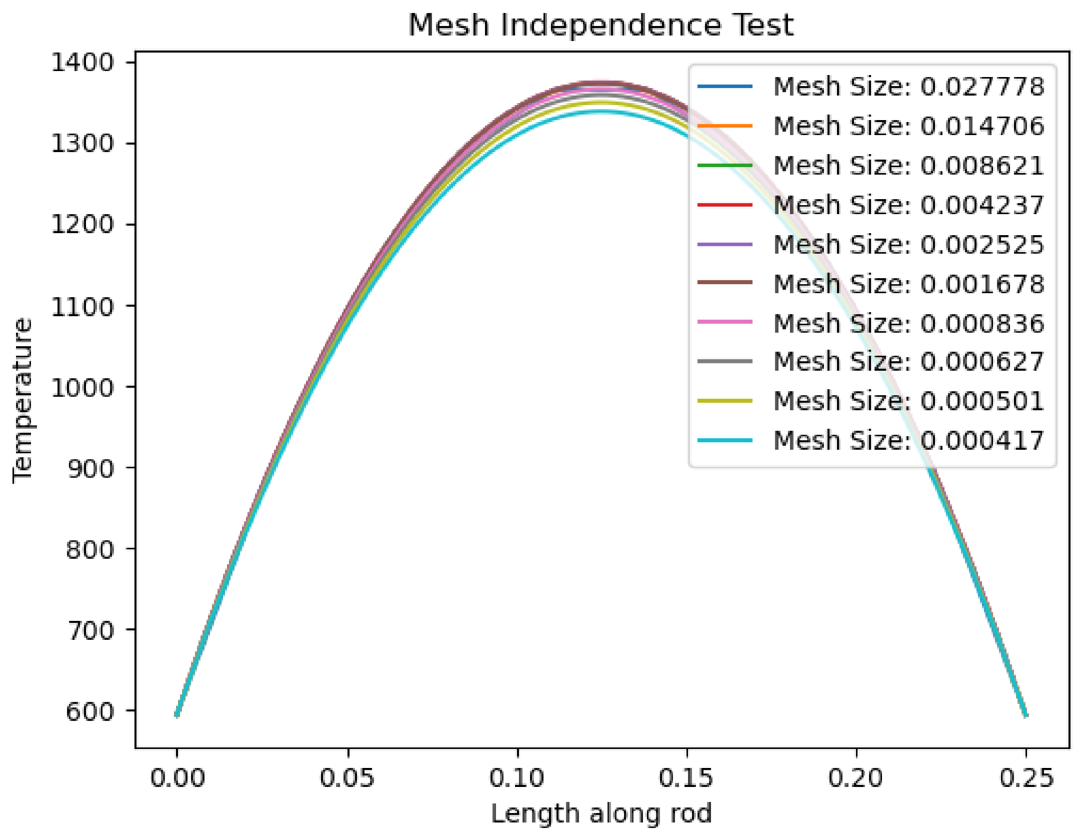
Triadiagonal Matrix Algorithm(TDMA)



In [6]:
```
T_tdma = np.zeros(n)
T_tdma[0] = T_surf
T_tdma[n-1] = T_surf

P = np.zeros(n)
Q = np.zeros(n)

a, b, c, d = 2/h**2, 1/h**2, 1/h**2, q_dot/k

P[0] = 0
Q[0] = T_surf

for i in range(1,n):
    P[i] = b / (a - c*P[i-1])
    Q[i] = (d + c*Q[i-1]) / (a - c*P[i-1])

Q[n-1] = T_tdma[n-1]

for i in range(n-2,-1, -1):
    T_tdma[i] = T_tdma[i+1]*P[i] + Q[i]


plt.plot(x, T_tdma, 'g-o')
plt.plot(x, T_analytical, 'c--')
plt.title("Numerical Solution using TDMA Method")
plt.ylabel("Temperature(T)")
plt.xlabel("Distance(x) along the Rod")
plt.legend(["Numerical Solution", "Analytical Solution"], loc='upper right')
```

Out[6]: <matplotlib.legend.Legend at 0x7fcae0855410>

Grid Independence Test



In [7]:

```
# plotting results using different grid points
grid_points = [10, 18, 30 , 60 ,100, 150, 300, 400, 500, 600]
plt.figure()

for m in grid_points:
    x_test = np.linspace(0,L,m)
    h = x_test[1] - x_test[0]
    T = np.zeros(m)
    C = (q_dot)*(h**2)/k

    T[0] = T_surf
    T[m-1] = T_surf

    T_old = np.copy(T)
    iterations = 0
    Error = 1
    while Error > Tolerance:

        for i in range(1,m-1):
            T[i] = 0.5*(C + T[i-1] + T_old[i+1])

        Error = max(abs(T - T_old))
        T_old = np.copy(T)
        iterations = iterations + 1

    plt.plot(x_test, T, label = f"Mesh Size: {h:.6f}")
    plt.title("Mesh Independence Test")
    plt.ylabel("Temperature")
    plt.xlabel("Length along rod")
    #plt.legend("ramankumar", loc='upper right')
    plt.legend(loc='upper right')
```

The above plots shows that for different grid points result are same, ensuring grid independence.