GOKHALE EDUCATION SOCIETY'S

Above Self... Above Politics

N. B. MEHTA (V) SCIENCE COLLEGE

ACHARYA BHISE VIDYANAGAR, BORDI

TAL - DAHANU, DIST. PALGHAR, 401701, Tel. 02528 - 254357

DEPARTMENT OF INFORMATION TECHNOLOGY AND COMPUTER SCIENCE

Certificate

Class ____________ Year ___________

This is to certify that the work entered in this journal is the work of Shri/Kumari ____

Of _________________ division___________________ Roll No. ______________________

Uni. Exam No. ___________________ has satisfactorily completed the required number of practical and worked for the 1st term / 2nd term/ both the terms of the Year _______________ in the college laboratory as laid down by the university.

Head of the Department	External Examiner	Internal Examiner Subject teacher

Date : / / 2022 Department of IT-CS

INDEX

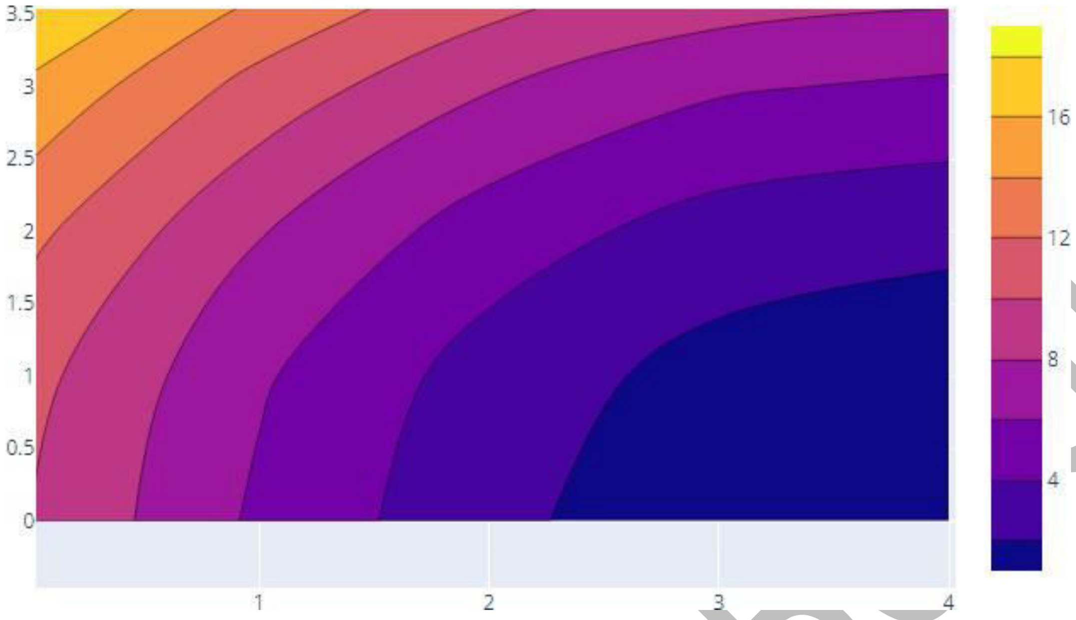
Practical 1

Aim :- Implement Contour Plots

Code :-

```
using PlotlyJS
plot(contour(
z=[
10   10.625   12.5   15.625   20
5.625   6.25   8.125   11.25   15.625
2.5   3.125   5.   8.125    12.5
0.62   1.25   3.125   6.25   10.625
 0   0.625   2.5   5.625   10
    ]'
))
```

OutPut:-



Practical 2

Aim :- Implement Fibonacci and Golden section search.

 # Fibonacci section search

Code:-

```
function fibonacci_section_search(f, a, b, n, g=0.01)
    s=(1-0.5)/(1+0.5)
    p=1/(1.618*(1-s^(n+1))/(1-s^n))
    d=p*b+(1-p)*a
    yd=f(d)
    for i in 1:n-1
        print(a)
```

```
         print("/n")
        print(b)
        print("/n")
        if i==n-1
              c=g*a+(1-g)*d
       else
            c=p*a+(1-p)*b
       end
              yc=f(c)
        if yc<yd
              b,d,yd=d,c,yc
         else
               a,b=b,c
      end
          p=1(1.618*(1-s^(n-i+1))/(1-s^(n-i)))
   end
   return a < b ? (a, b) : (b, a)end
```

 # fibonacci_search (generic function with 3 methods)

```
function f(x)
   returnx*x-x+1
   end1
   end
```

 # f (generic function with 1 method)

```
Result=fibonacci_section_search(f,-1,1,10)
```

OutPut:-

(-0.011235955056179792 , 0.011235955056179796)

Golden section search

Code:-

```
function golden_section_search(f, a, b, n)
       p=1.618-1
       d=p*b+(1-p)*a
       yd=f(d)
       for i = 1 : n - 1
            c=p*a+(1-p)*b
            yc=f(c)
             if yc<yd
                    b, d, yd=d, c, yc
               else
                     a, b=b, c
              end
       end
          return a < b ? (a, b) : (b, a)
       end
```

 # golden_section_search (generic function with 1 method)

```
function f(x)
       return x*x
       end
```

 # f (generic function with 1 method)

```
result=golden_section_search(f,2,8,5)
```

Output :-

(2, 2.8755439999999997)

Practical 3

Aim:- Implement Quadratic Fit Search

Code:-

```
function quadratic_fit_search(f,a,b,c,n)
            ya,yb,yc=f(a),f(b),f(c)
      for i in 1:n-3
            print(a,"/n",b,"/n",c,"/n")
                  x=0.5*(ya*(b^2-c^2)+yb*(c^2-a^2)+yc*(a^2 -
b^2))/(ya*(b-c)+yb*(c-a)+yc*(a-b))
          yx=f(x)
          if x > b
               if yx > yb
                     c, yc = x, yx
               else
                    a, ya, b, yb = b, yb, x, yx
               end
           elseif x < b
                if yx > yb
                     a, ya = x, yx
                else
```

```
            c, yc, b, yb = b, yb, x, yx
          end
        end
      end
        return (a, b, c)
    end
```
 # quadratic_fit_search (generic function with 1 method)

```
function f(n)
        return n*n+2*n-1
        end
```
 # f (generic function with 1 method)

```
result=quadratic_fit_search(f,1,6,10,5)
```

Output :-

```
    1/n6/n10/n1/n-1.0/n6/n(1, -1.0, 6)
```

Practical 4

Aim :- Implement Gradient descent

Code:-

```
function gradient_descent(p, q, x1; a=0.1,
maxiter=1000, g=1e-5)
    x-copy(x1)
```

```
f=x -> 0 * x + q
x2= -f(x)
iter = 0
while norm(x2) > g || iter <= maxiter
  iter +=1
  x += a * x2
  x2 = -f(x)
end
return x
end
```

 # gradient_descent (generic function with 1 method)

```
p=[10.0 -1.0;
    -1.0 1.0 ];
 q=[0; -10.0];
x2=zeros(2);
```

Output :-

1.1111111111111103
11.111111111111104

Practical 5

Aim :- Implement quasi-Newton methods to find the local maxima

Code:-

```
function  newtonsMethodForUnivariate (x_guess,
max_iter)
      f_1=2 * x_guess
      f_2=2
      converged = false
      iter = 0
      while converged == false
        x_optimum = x_guess - (f_1/f_2)
        x_guess = x_optimum
      println("Iteration : $iter, Current Guess:
$x_guess")
      if x_guess - 1< 0.01
         converged = true
      end
      if iter>max_iter
```

```
    converged = true
   end
   iter=iter+1
   end
   end
```

 # newtonsMethodForUnivariate (generic function with 1 method)

newtonsMethodForUnivariate (3,100)

Output :-

Iteration : 0, Current Guess: 0.0

Practical 6

Aim :- Implement the Adagrad method with application, RMSprop and Adadelta.

Adagrad method with application

Code:-

```
function ada_grad(x_guess, max_iter, alpha)
   fd=2 * x__guess - 2
```

```
    Converged = false
    Iter = 0
    prev_sgs = 0
    while converged == false
      delta = alpha * fd
      sgs = prev_sgs + (fd)^2
      x_optimum = x_guess  -  delta/sqrt(sgs)
      x_guess  =  x_optimum
      prev_sgs = sgs
      println("Iteration : $iter, current Guess: $x_guess")
      if x_guess - 1 < 0.01
        converged  =  true
      end
      if iter > max_iter
        converged = true
      end
      iter = iter+1
      end
   end
```

 # ada_grad (generic function with 1 method)

```
ada_grad(3,20000,0.01)
```

Output :-

Iteration : 10043, current Guess : 1.0101584845215472

Iteration : 10044, current Guess : 1.0100587087650088

Iteration : 10045, current Guess : 1.0099589379745384

#RMSprop

Code :-

```
function rms_prop(x_guess, max_iter, alpha, beta)

    fd = 2 * x_guess - 2
    converged = fasle
    iter = 0
    prev_sgs = 0
    while converged == false
      delta = alpha * fd
      sgs = (prev_sgs * beta ) + ((fd)^2) * (1-beta)
      x_optimum = x_guess - delta/sqrt(sgs)
      x_guess = x_optimum

      prev_sgs = sgs
      println("Iteration : $iter, Current_Guess : $x_guess")
      if x_guess - 1 < 0.01
        converged = true
      end
      if iter>max_iter
        converged = true
      end
     iter=iter+1
     end
    end
```

 # rms_prop (generic function with 1 method)

rms_prop(3, 4000, 0.01, 0.99)

Output:-

Iteration:95, Current_Guess: 1.0244024028520233
Iteration:96, Current_Guess: 1.0117305980069258
Iteration:97, Current_Guess: 0.9990969991751104

Adadelta

Code:-

```
function ada_delta(x_guess, max_iter, gamma)
      fd= 2 * x_guess - 2
      converged = false
      iter = 0
      prev_sgs = 0
      Ex = 0
      ep = 1e - 5
      while converged == false
        sgs = (gamma * prev_sgs) +((1-gamma) * (fd^2))
        rms_g = sqrt(sgs + ep)
        rms_x = sqrt(Ex + ep)
        x = (rms_x/rms_g) * fd
        Ex = (gamma * Ex) + ((1-gamma) * (x^2))

        prev_sgs = sgs
        x_optimum = x_guess - x
```

```
 x_guess = x_optimum
println("Iteration : $iter, Current_Guess : $x_guess")

 if x_guess - 1 < 0.00001
    converged = true
end
if iter>max_iter
converged = true
end
 iter =iter+1
end
end
```

 # ada_delta (generic function with 1 method)

ada_delta(3, 400, 0.9)

Output :-

Iteration :143 ,   Current_Guess :1.03482265980986482
Iteration :144,   Current_Guess :1.0186375926957296
Iteration :145,   Current_Guess :1.002417732057021
Iteration :146,   Current_Guess :1.0861670747700315

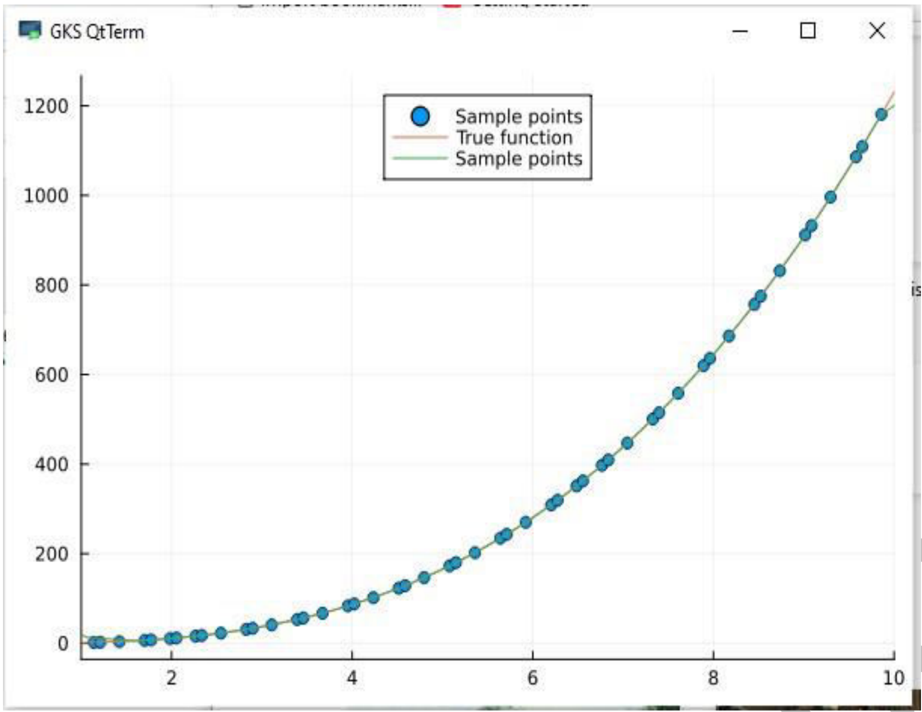
Practical 7

Aim :- Implement radial basis functions using surrogate modelling.

Code:-

```
using Surrogates
using Plots
f=x -> log(x) * x^2 + x^3
ib=1.0
ub=10.0
x = sample(50, ib, ub, SobolSample())
y = f.(x)
my_radial_basis = RadialBasis(x,y,ib,ub)
approx = my_radial_basis(5.4)
using Plots
plot(x, y, seriestype=:scatter, label="Sample points",
xlims=(ib, ub), legend=:top)
plot!(f, label="True function", xlims=(ib,ub), legend=:top)
plot!(my_radial_basis, label="Sample points", xlims=(ib, ub),
legend=:top)
```

Output:-



Practical 8

Aim:- Apply Random Forest in surrogate Model.

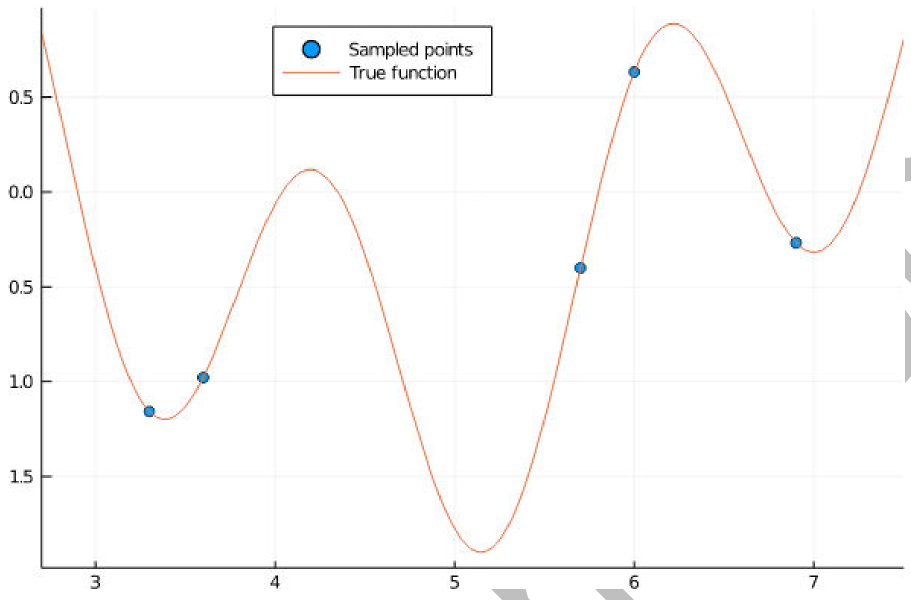
Code:-

```
using Surrogates
using Plots

f(x) = sin(x) + sin(10/3 * x)
n_samples = 5
lower_bound = 2.7
upper_bound = 7.5
x = sample(n_samples, lower_bound,
upper_bound, SobolSample())
y = f.(x)
scatter(x, y, label="Sampled points",
xlims=(lower_bound, upper_bound))
```

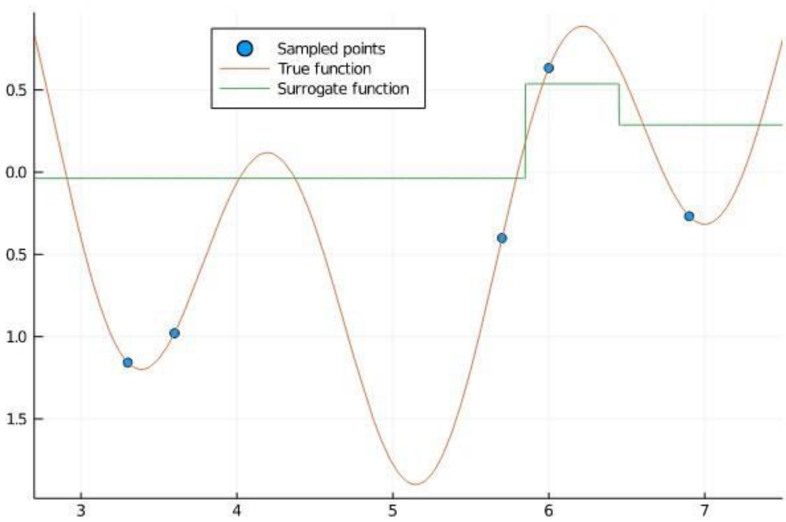
```
plot!(f, label="True function",
xlims=(lower_bound, upper_bound),
legend=:top)
```

Output:-




```
num_round = 2
randomforest_surrogate = RandomForestSurrogate(x ,y ,lower_bound,
upper_bound, num_round = 2)
plot(x, y, seriestype=:scatter, label="Sampled points",
xlims=(lower_bound, upper_bound), legend=:top)
plot!(f, label="True function", xlims=(lower_bound, upper_bound),
legend=:top)
plot!(randomforest_surrogate, label="Surrogate function",
xlims=(lower_bound, upper_bound), legend=:top)
```

Output:



Practical 9

Aim: - Implement Gaussian Process and its application.

Code:-

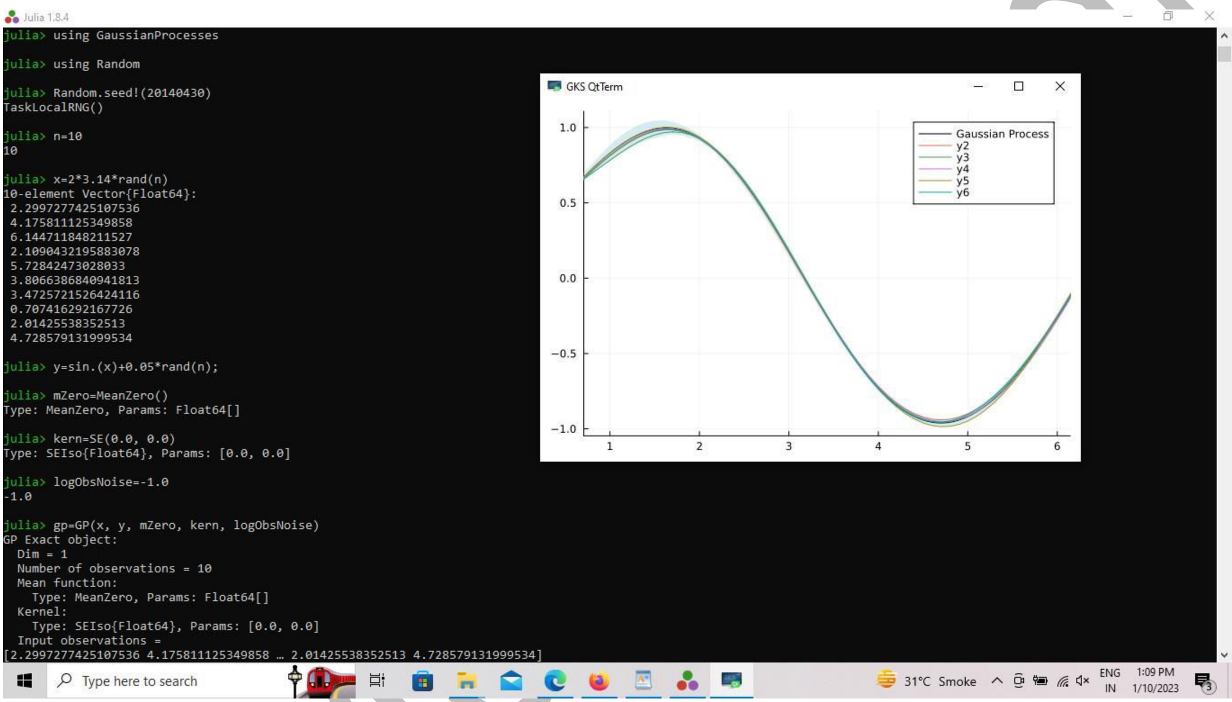
```
using GaussianProcesses
using Random
Random.seed!(20140430)
n=10
x=2*3.14*rand(n)
y=sin.(x)+0.05*rand(n);
mZero=MeanZero()
kern=SE(0.0, 0.0)
logObsNoise=-1.0
gp=GP(x, y, mZero, kern, logObsNoise)
x=0:0.1:2*3.14
```

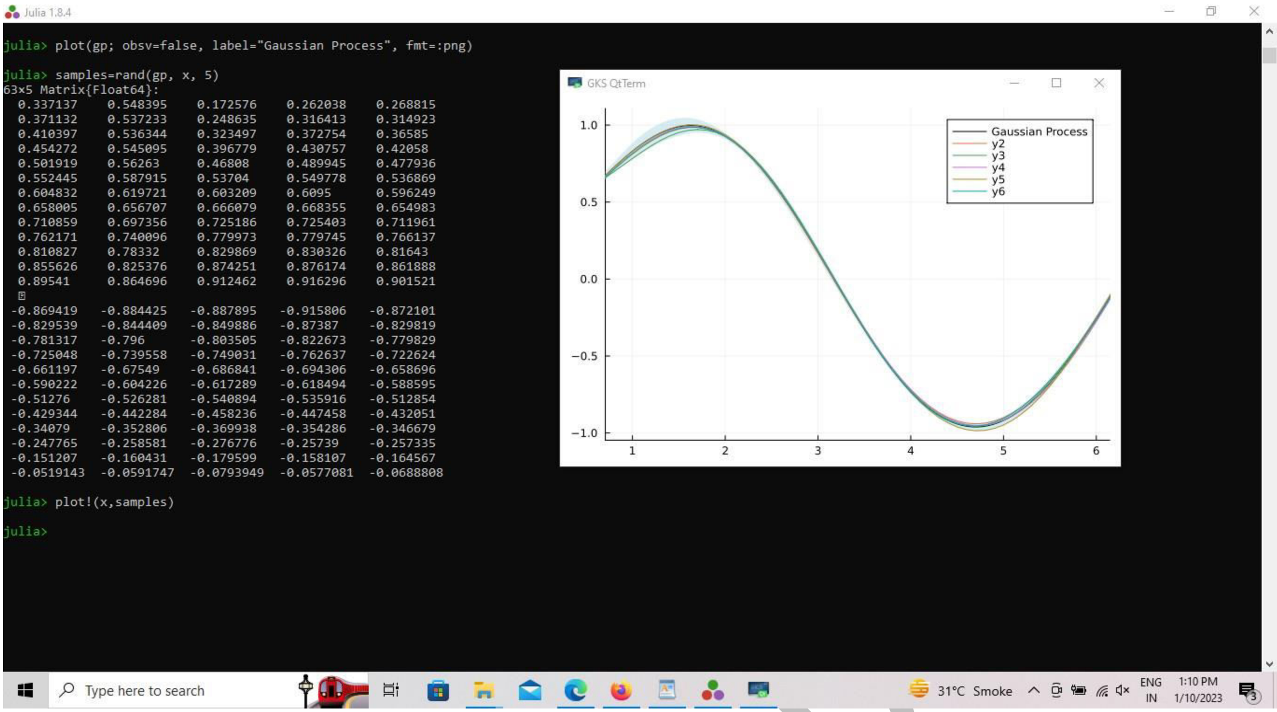
Method1

```
using Plots
plot(gp; obsv=false)
```

```
optimize!(gp)
plot(gp; obsv=false, label="Gaussian Process", fmt=:png)
samples=rand(gp, x, 5)
plot!(x,samples)
```

Output:-

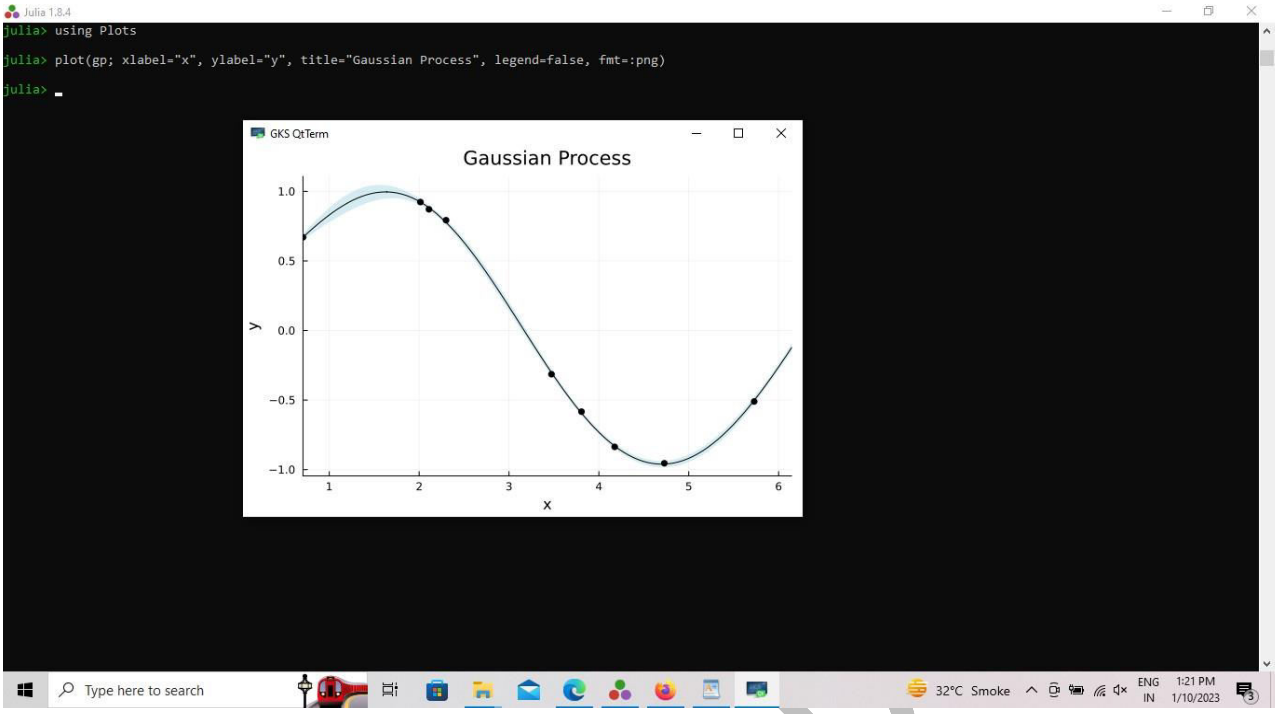




Method2

Code:-

```
using Plots
plot(gp; xlabel="x", ylabel="y", title="Gaussian Process", legend=false, fmt=:png)
```



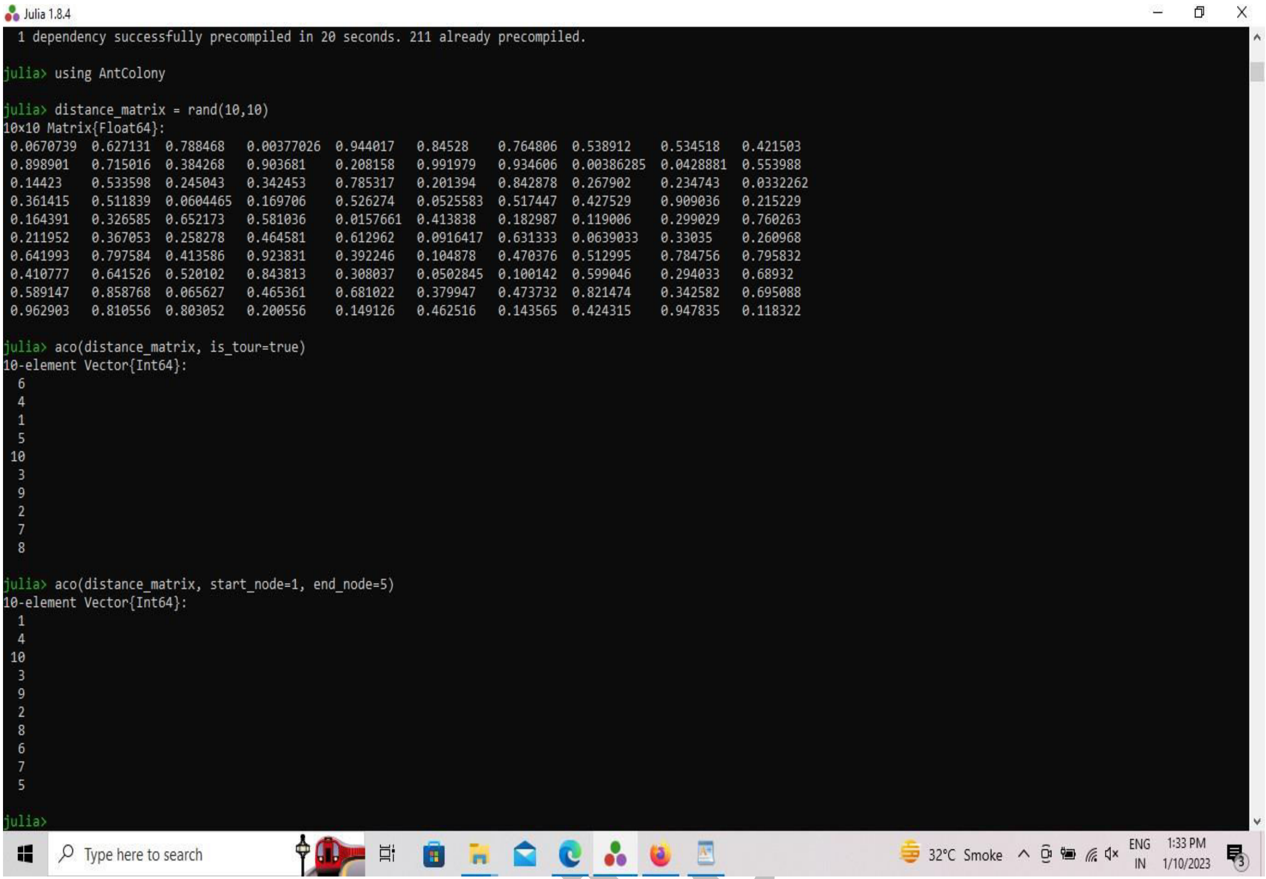
Practical 10

Aim :- Path finding using Ant Colony Optimization with an application

Code:-

```
using AntColony
distance_matrix = rand(10,10)
aco(distance_matrix, is_tour=true)
aco(distance_matrix, start_node=1, end_node=5)
```

Output:-


Julia 1.8.4
1 dependency successfully precompiled in 20 seconds. 211 already precompiled.
julia> using AntColony
julia> distance_matrix = rand(10,10)
10×10 Matrix{Float64}:
0.0670739 0.627131 0.788468 0.00377026 0.944017 0.84528 0.764806 0.538912 0.534518 0.421503
0.898901 0.715016 0.384268 0.903681 0.208158 0.991979 0.934606 0.00386285 0.0428881 0.553988
0.14423 0.533598 0.245043 0.342453 0.785317 0.201394 0.842878 0.267902 0.234743 0.0332262
0.361415 0.511839 0.0604465 0.169706 0.526274 0.0525583 0.517447 0.427529 0.909036 0.215229
0.164391 0.326585 0.652173 0.581036 0.0157661 0.413838 0.182987 0.119006 0.299029 0.760263
0.211952 0.367053 0.258278 0.464581 0.612962 0.0916417 0.631333 0.0639033 0.33035 0.260968
0.641993 0.797584 0.413586 0.923831 0.392246 0.104878 0.470376 0.512995 0.784756 0.795832
0.410777 0.641526 0.520102 0.843813 0.308037 0.0502845 0.100142 0.599046 0.294033 0.68932
0.589147 0.858768 0.065627 0.465361 0.681022 0.379947 0.473732 0.821474 0.342582 0.695088
0.962903 0.810556 0.803052 0.200556 0.149126 0.462516 0.143565 0.424315 0.947835 0.118322
julia> aco(distance_matrix, is_tour=true)
10-element Vector{Int64}:
6
4
1
5
10
3
9
2
7
8
julia> aco(distance_matrix, start_node=1, end_node=5)
10-element Vector{Int64}:
1
4
10
3
9
2
8
6
7
5
julia>
Type here to search
32°C Smoke
ENG
IN
1:33 PM
1/10/2023