Sadiq S.

Algorithm for Optimization

Practical No. 6

AIM: Apply Random Forest in surrogate Model.

Random forests is a supervised learning algorithm that randomly creates and merges multiple decision trees into one forest.

We are going to use a Random forests surrogate to optimize
f(x)=sin(x)+sin(10/3∗x).

First of all import Surrogates and Plots.

Code:

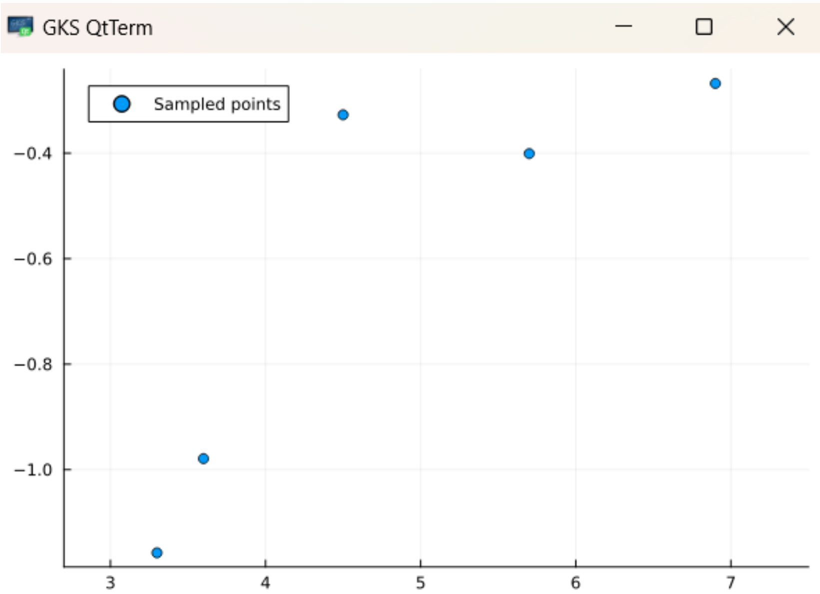
```
using Surrogates
using SurrogatesRandomForest
using Plots
default()
```

```
julia> using Surrogates

julia> using SurrogatesRandomForest

julia> using Plots

julia> default()
```

Sampling:
We choose to sample f in 4 points between 0 and 1 using the sample function. The sampling points are chosen using a Sobol sequence, this can be done by passing SobolSample() to the sample function.

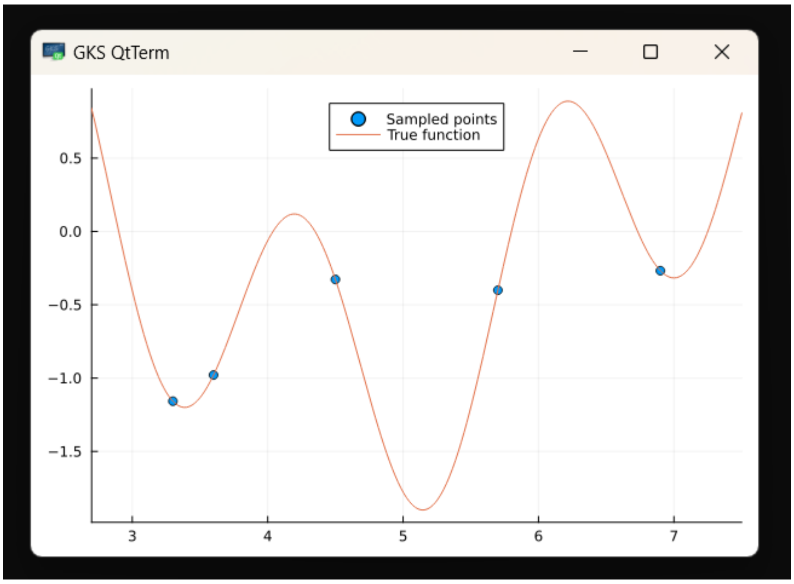
```
f(x) = sin(x) + sin(10/3 * x)
n_samples = 5
lower_bound = 2.7
upper_bound = 7.5
x = sample(n_samples, lower_bound, upper_bound, SobolSample())
y = f.(x)
scatter(x, y, label="Sampled points", xlims=(lower_bound, upper_bound))
```

```
julia> f(x) = sin(x) + sin(10/3 * x)
f (generic function with 1 method)

julia> n_samples = 5
5

julia> lower_bound = 2.7
2.7

julia> upper_bound = 7.5
7.5

julia> x = sample(n_samples, lower_bound, upper_bound, SobolSample())
5-element Vector{Float64}:
 4.5
 6.9
 5.7
 3.3000000000000003
 3.6

julia> y = f.(x)
5-element Vector{Float64}:
 -0.3272422775079802
 -0.2677806397869723
 -0.4008083329346852
 -1.157735900693952
 -0.9790933612952875

julia> scatter(x, y, label="Sampled points", xlims=(lower_bound, upper_bound))
```



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

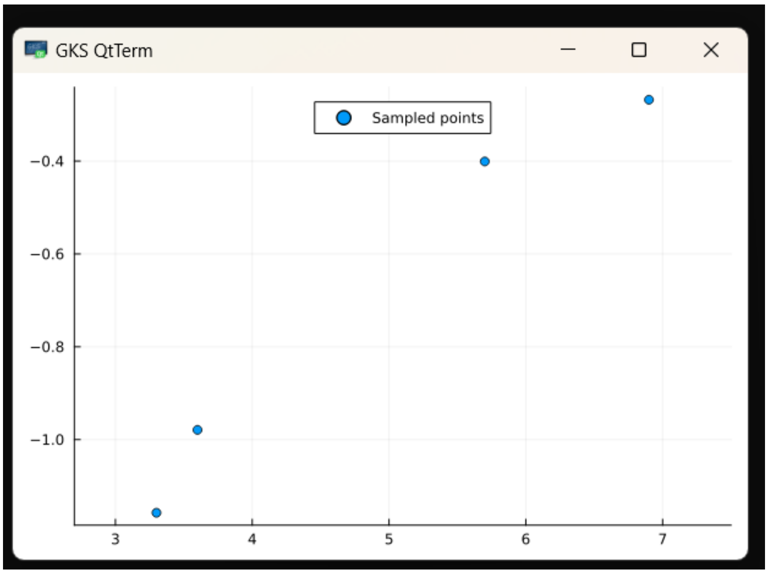
```
julia> plot!(f, label="True function", xlims=(lower_bound, upper_bound), legend=:top)
```



Building a surrogate:
With our sampled points we can build the Random forests surrogate using the RandomForestSurrogate function.
randomforest_surrogate behaves like an ordinary function which we can simply plot. Additionally, you can specify the number of trees created using the parameter num_round

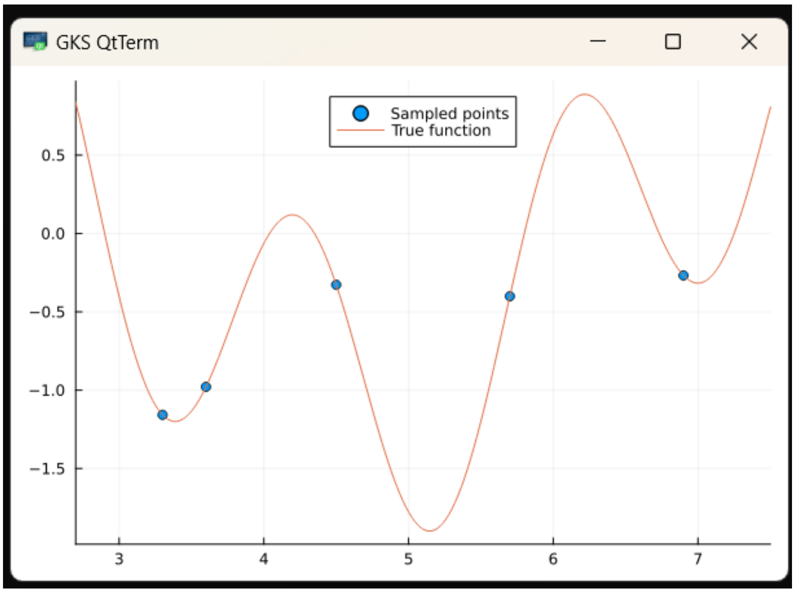
num_round = 2
randomforest_surrogate = RandomForestSurrogate(x ,y ,lower_bound, upper_bound, num_round = 2)
plot(x, y, seriestype=:scatter, label="Sampled points", xlims=(lower_bound, upper_bound), legend=:top)

```
julia> num_round = 2
2

julia> randomforest_surrogate = RandomForestSurrogate(x ,y ,lower_bound, upper_bound, num_round = 2)
[1]     train-rmse:0.92140953630052491
[2]     train-rmse:0.73258731448583370
(::RandomForestSurrogate{Vector{Float64}, Vector{Float64}, XGBoost.Booster, Float64, Float64, Int64}) (generic function with 2 methods)

julia> plot(x, y, seriestype=:scatter, label="Sampled points", xlims=(lower_bound, upper_bound), legend=:top)
```



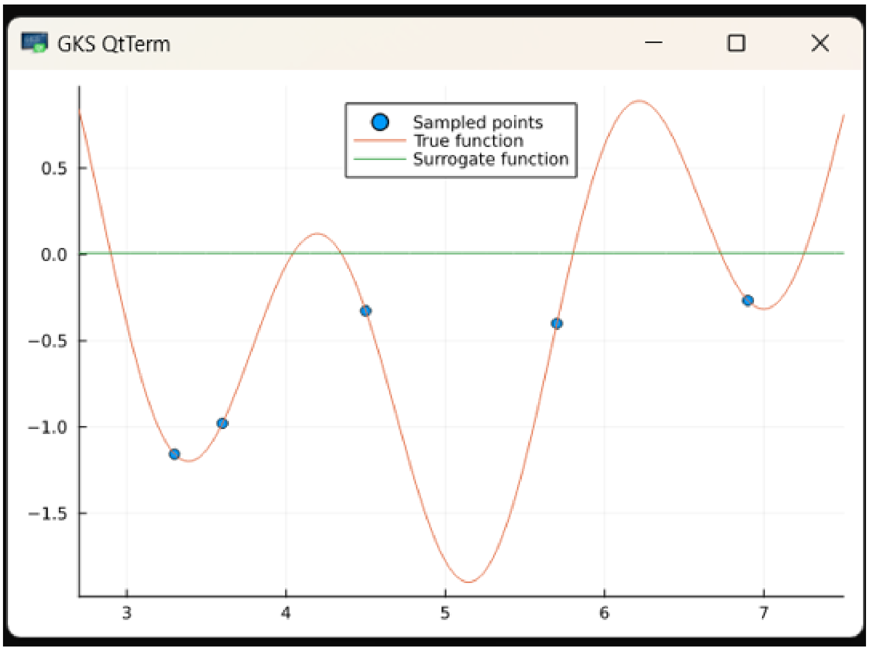
plot!(f, label="True function", xlims=(lower_bound, upper_bound), legend=:top)

```
julia> 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)

```
julia> plot!(randomforest_surrogate, label="Surrogate function", xlims=(lower_bound, upper_bound), legend=:top)
```



Optimizing:
Having built a surrogate, we can now use it to search for minima in our original function f. To optimize using our surrogate we call surrogate_optimize method. We choose to use Stochastic RBF as optimization technique and again Sobol sampling as sampling technique.

@show surrogate_optimize(f, SRBF(), lower_bound, upper_bound, randomforest_surrogate, SobolSample())
scatter(x, y, label="Sampled points")

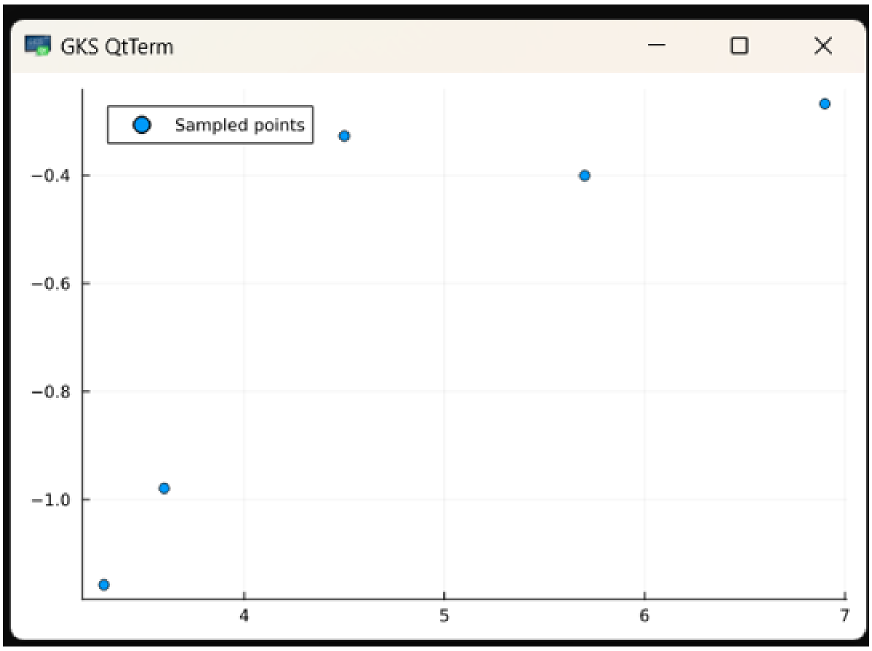
```
julia> @show surrogate_optimize(f, SRBF(), lower_bound, upper_bound, randomforest_surrogate, SobolSample())
[1]     train-rmse:0.95658490418620512
```

.

.

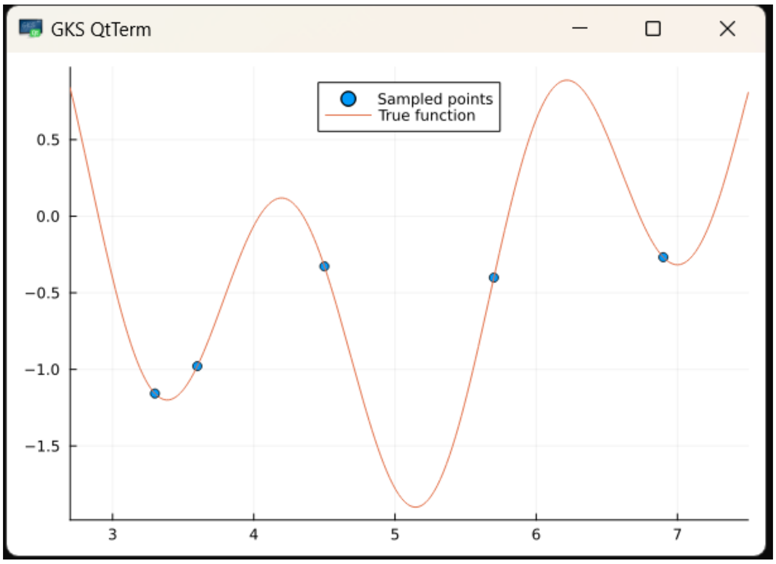
.

```
[2]     train-rmse:0.78901737183904808
Out of sampling points
surrogate_optimize(f, SRBF(), lower_bound, upper_bound, randomforest_surrogate, SobolSample()) = (3.3878250000000003, -1.199918997447965)
(3.3878250000000003, -1.199918997447965)

julia> scatter(x, y, label="Sampled points")
```



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

```
julia> 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)