Algorithm For Optimizations Practical

Aim: Apply Random Forest in Surrogate Model.

Random forest 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:

```
using Pkg
Pkg.add("Surrogates")
Pkg.add("SurrogatesRandomForest")
Pkg.add("Plots")
using Surrogates
using SurrogatesRandomForest
using Plots
default()
```

Output:

```
julia> using Surrogates

julia> using SurrogatesRandomForest

julia> using Plots

julia> default()

julia>
```

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.

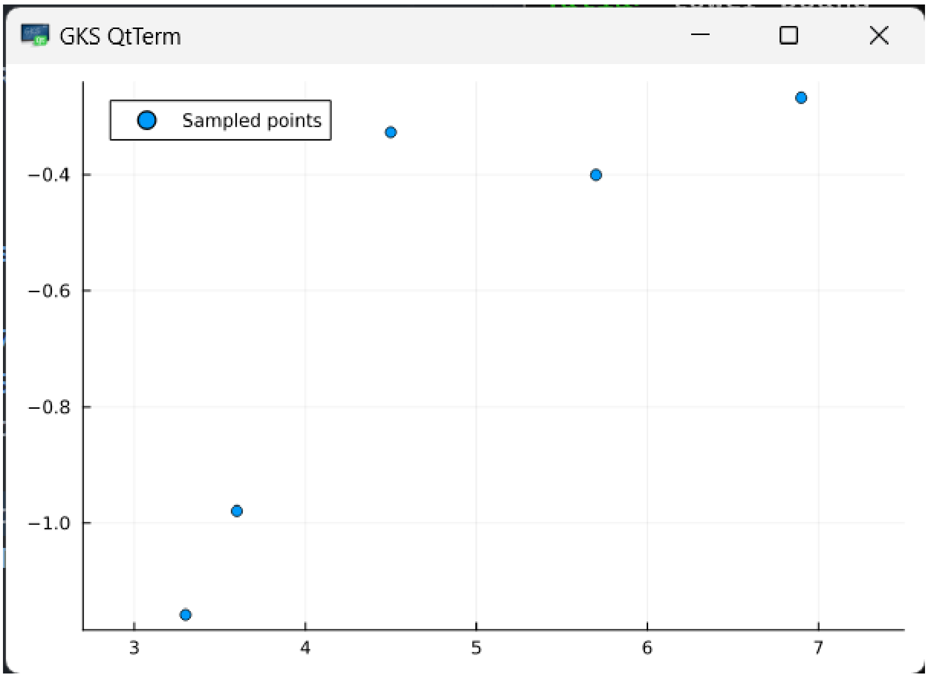
```
f(x) = sin(x) + sin(10 / 3 * x)
n_samples = 5
lower_bound = 2.7
upper_bound = 7.5
x = sample(n_samples, lower_bound, upper_bound, SobolSample())
y = f.(x)
```

Output:

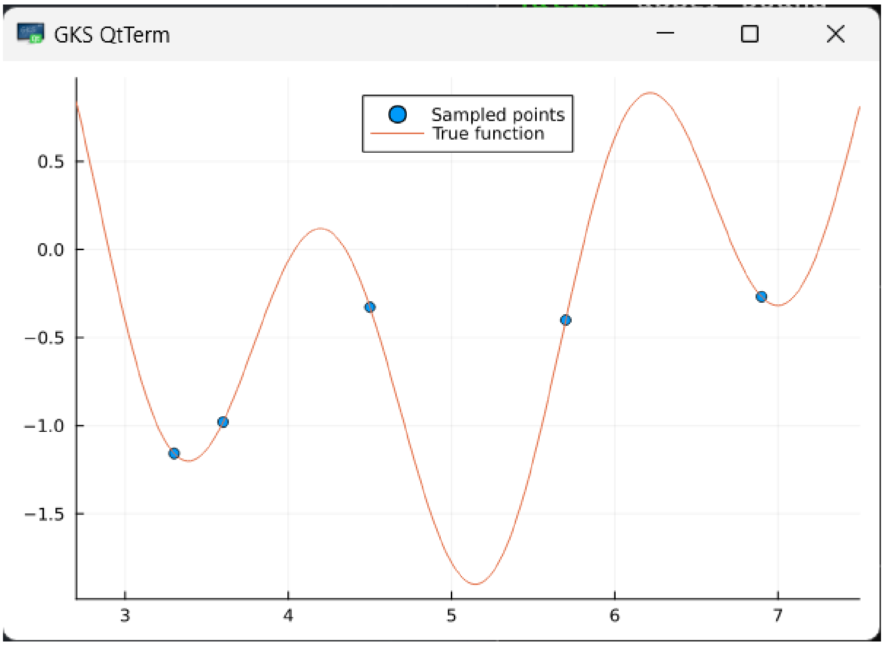
```
julia> f(x) = sin(x) + sin(10 / 3 * x)
f (generic function with 1 method)

julia> n_samples = 5
5

julia> lower_bound = 2.7
2.7

julia> upper_bound = 7.5
7.5

julia>
```

```
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))
```



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



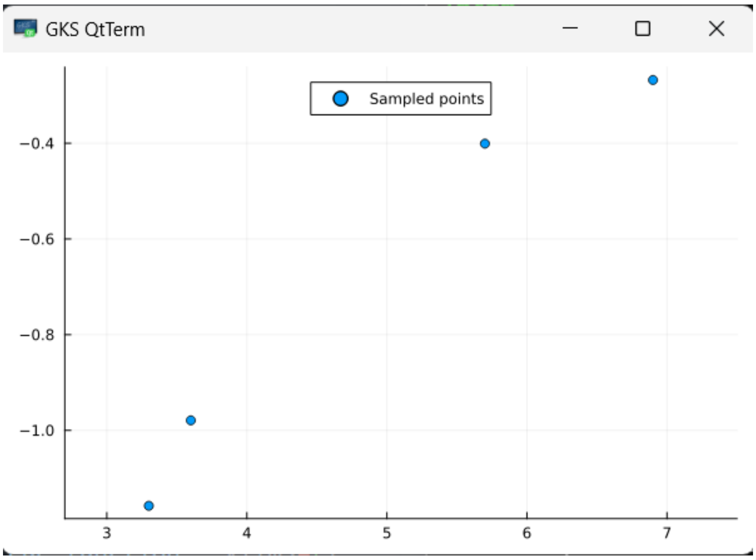
Building a surrogate:
With our sampled points we can build the Random forests surrogate using the RandomForestSurrogate function.
randomforest_surrogate behaves like an ordinary function which we can simply plot.
Additionally, you can specify the number of trees created using the parameter num_round

```julia
num_round = 2
randomforest_surrogate = RandomForestSurrogate(x, y, lower_bound, upper_bound,
num_round=2)
```

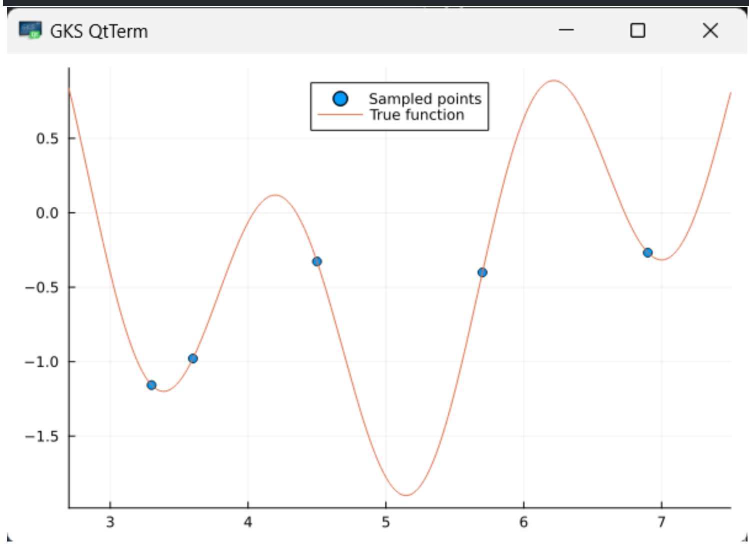
Output:

```
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>
```

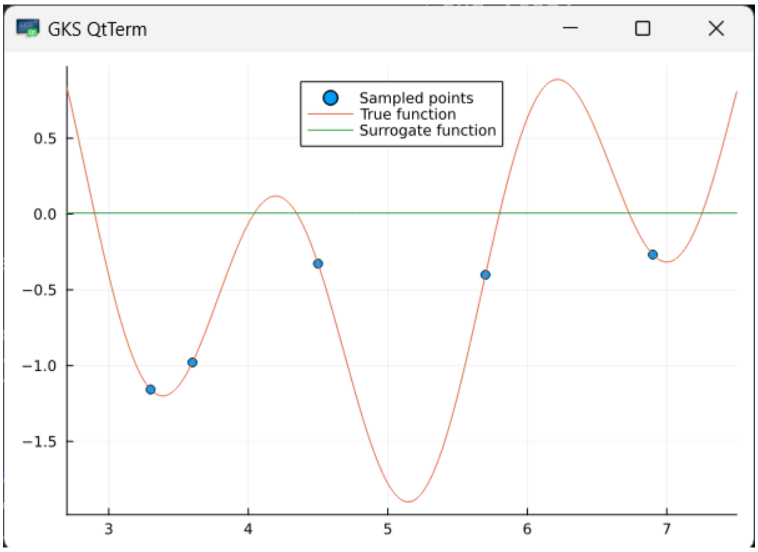
```julia
plot(x, y, seriestype=:scatter, label="Sampled points", xlims=(lower_bound,
upper_bound), legend=:top)
```




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




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



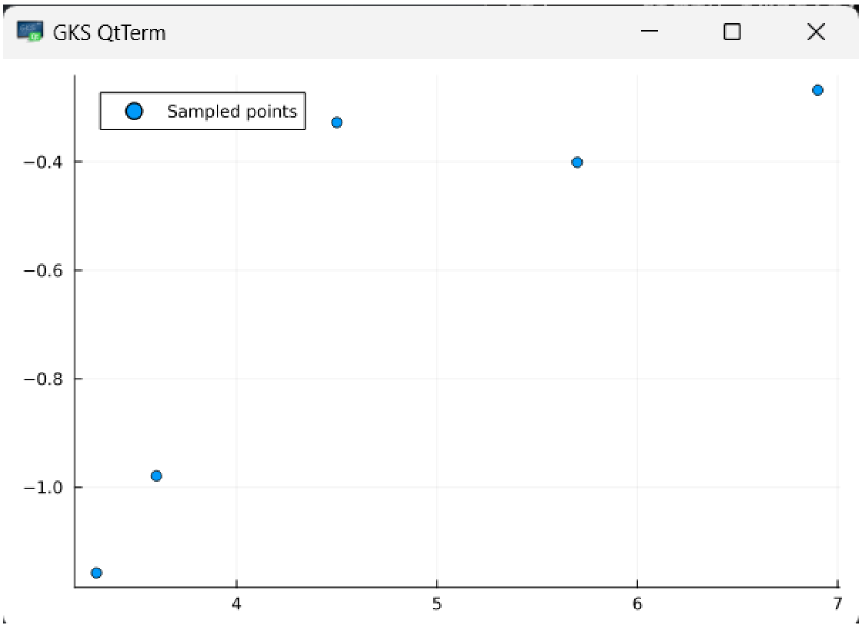
Optimizing:

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.

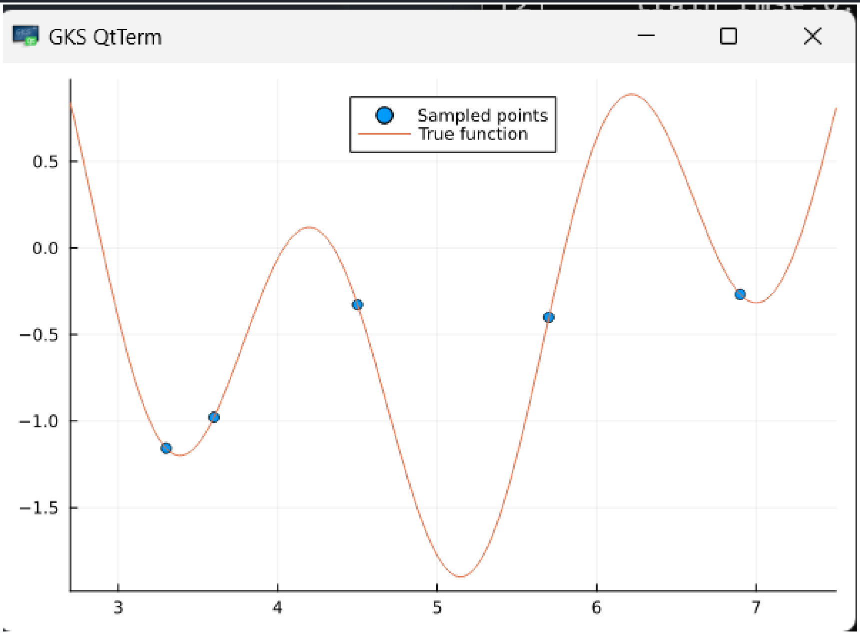
```
@show surrogate_optimize(f, SRBF(), lower_bound, upper_bound,
randomforest_surrogate, SobolSample())
```

```
julia> @show surrogate_optimize(f, SRBF(), lower_bound, upper_bound, randomforest_surrogate, SobolSample())
[1]     train-rmse:0.95658490418620512
[2]     train-rmse:0.74256970870539318
[1]     train-rmse:0.95244535183376022
[2]     train-rmse:0.72868553891778742
[1]     train-rmse:0.91247892143124687
[2]     train-rmse:0.69662504529969971
[1]     train-rmse:0.95206655470840285
[2]     train-rmse:0.72240936082817953
[1]     train-rmse:0.97901675703409219
[2]     train-rmse:0.73392341202244027
[1]     train-rmse:1.00227437200570568
[2]     train-rmse:0.74644344728113488
[1]     train-rmse:1.01902347284667094
[2]     train-rmse:0.75467692565881217
[1]     train-rmse:1.02419476876355042
[2]     train-rmse:0.75492892151547719
[1]     train-rmse:1.03354009541140091
[2]     train-rmse:0.75873872842853440
[1]     train-rmse:1.02156228132489679
[2]     train-rmse:0.74965539671374803
[1]     train-rmse:1.03344301835234287
[2]     train-rmse:0.75594491006437159
[1]     train-rmse:1.04212854701235913
[2]     train-rmse:0.76009740987454033
[1]     train-rmse:1.04654834879749381
[2]     train-rmse:0.76137023951960858
[1]     train-rmse:1.05500310522109197
[2]     train-rmse:0.76586941148042631
[1]     train-rmse:1.06260916555808338
[2]     train-rmse:0.76990793226169518
[1]     train-rmse:1.06930384346182383
[2]     train-rmse:0.77341119071026598
[1]     train-rmse:1.07511544625894140
[2]     train-rmse:0.77638220317228412
[1]     train-rmse:1.07975589982219877
[2]     train-rmse:0.77859138703162212
[1]     train-rmse:1.08482350984962772
[2]     train-rmse:0.78122996701941028
[1]     train-rmse:1.08940807589963384
[2]     train-rmse:0.78359595628391676
[1]     train-rmse:1.09359437810022464
[2]     train-rmse:0.78574355503922366
[1]     train-rmse:1.09714144380539103
[2]     train-rmse:0.78731498385216836
[1]     train-rmse:1.10059135223886417
[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>
```

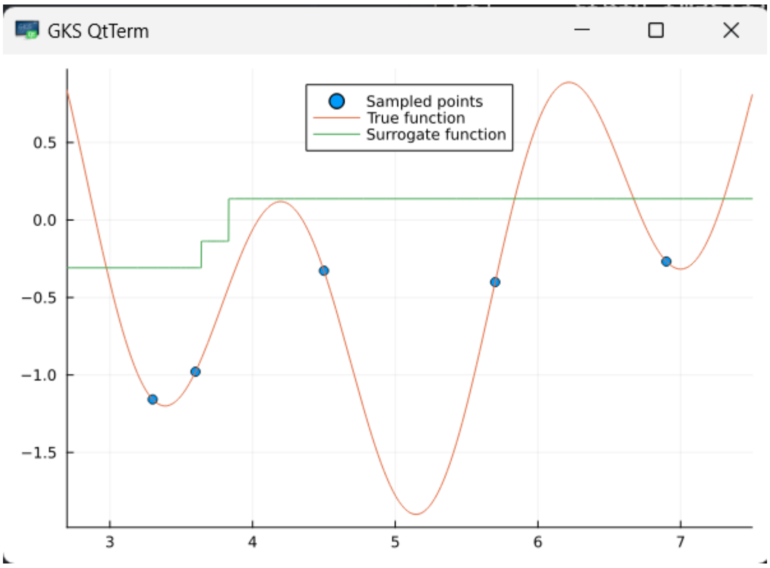
```
scatter(x, y, label="Sampled points")
```




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




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


GKS QtTerm
Sampled points
True function
Surrogate function
0.5
0.0
−0.5
−1.0
−1.5
3
4
5
6
7