1 A linear introduction to BOMoD.jl

```
using BOMoD: go2lab, ground_truth, BO_Lin_wrapper, Linear_TS_sampling, model_linear,
model_random, update_prior_model_linear, make_plot, design_maxtrix, make_predictions
using BOMoD
using Random
using Distributions
using DataFrames
using Plots
using StatsPlots
```

1.1 Part 0: Introduction

To introduce the concept of the package, we start with an easy problem made up out of toy data.In this example, every module will receive a specific activity.The activity of a particular construct is then simply the sum of all the activities of the corresponding modules. As a result, we assume there no interactions between the modules. In many real-world application, this assumption isn't valid, but it is a good starting point because:

1. Show the concept of the BOMoD package
2. Explain the steps of the BOMoD package
3. Shows how easy it is to create combinatorial design spaces with BOMoD
4. Avoided complicated math, what isn't the focus of this package
5. The transition to real-world data is intuitive and mathematically solid.

This example is build up in different sections **Part 1**: Setup
 Explains how to set up a BOMoD.jl project **Part 2**: Procedure Explain all steps of the Optimisation process in great detail. **Part 3**: Example Example of a full BOMoD.jl project **Part 4**: Evaluation Evaluated the performance of the used setup. **Part 5**: Conclusion

1.2 Part 1 : Setup

1.2.1 1.1 Inputs design

Step 1 is the set up of the design space. The user has to give some inputs to the package:

1. The different modules that can be used
2. The length of the construct that is used currently under investigation.
3. Set the model to use unordered or ordered constructs
4. Optional: constraints, but these aren't used in this example.

```
length_constructs = 3;
pre_mod = ["a","b","c","d","e","f","g","h","i"];
order = false ;
```

1.2.2 Inputs model

The inputs of the model are users defined settings. The n*first determines how many data points are sampled in the first model step (2.1). The n*sample parameters sets the number of data point that the model proposes for the next round(2.4). For this easy model, these are all the parameters that the user needs to set.

```
n_first = 3;
n_samples = 3;
```

1.2.3 Construct of Design and Space

Now the design can be constructed using the parameters set in 1.1. In this example, we use unordered combinations, which means that the order of the construct will not influence the activity of constructs. Because there are no constraints, the BOMoD package allows efficient generation of the design space. All constructs aren't explicitly generated.

```
mod = group_mod(pre_mod);
design = construct_design(mod,3,order = order);
space = getspace(design);
```

1.2.4 Hidden Step

The hidden step sets up the toy data. In this toy data example, we assume that every construct has one " true" activity value. This value is a sample from a uniform distribution. In a later step, noise is attributed.

```
rng  = MersenneTwister();
activities = rand(rng,Uniform(0,5),length(mod));
Toy_data = Dict(mod .=> activities);
```

1.3 PART 2 Procedure

1.3.1 Introduction on the model

In julia, many excellent packages are available to do the math operation required to construct the model of choice. The advantage of using these packages is that they are efficient and tested on the correctness of their implementation. Unfortunately, most models request some user-defined starting point and have many overwhelming features. Additionally, this starting point should fit the ecosystem of the chosen package. The BOMoD package tries to solve most of these complex models setting in an elegant wrapper. Still, the wrapper should allow advanced users to alternate the parameters of the underlying model.

1.3.2 Linear model

This example is a simple linear model is proposed. $y = x_i^T \beta + \epsilon_i$

In most case, minimising some loss function will obtain the beta values. Only to allow the model to propose new samples uncertainties on the beta parameters are required. A simple way to get this uncertainty is to transform the linear model to a Bayesian linear model. The

model now predicts a mean value and standard deviation for every beta parameter. The mathematics behind the model isn't necessary to understand, see for more infromation (C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006, ISBN 026218253X. c 2006 Massachusetts Institute of Technology) The same equation above holds only $\beta \sim \mathcal{N}(\mu, \sigma)$ And the model needs a pior to start with in this case we use $\beta \sim \mathcal{N}(0, 1)$

As stated above, don't lose yourself in the model setup,they are only used to support the optimisation process. The package imports most of the mathematical operation form other good tested packages.

1.3.3 First data points

The first step is to option our first data point that needs to be tested in the lab. To keep things as simple as possible, we just random sample our first data points with the number of data point as a users input.

Unfortunately, the model can't start without a prior that is first set by the user. This requires some basic knowledge on how to use these packages of i For this specific problem, we use the prior $\beta \sim \mathcal{N}(0, 1)$

```
constructs = sample(rng,space,n_first)
```

```
3-element Array{Unordered_Construct{Mod{String}},1}:
 Any[Mod{String}("h") Mod{String}("e") Mod{String}("a")]
 Any[Mod{String}("i") Mod{String}("f") Mod{String}("c")]
 Any[Mod{String}("i") Mod{String}("c") Mod{String}("b")]
```

1.3.4 Evaluation first constructs

The new constructs need to be evaluated in the lab. To mimic this on our toy data set a small function is used that gives a mean and standard deviation for every construct.

Three values were sampled from a distribution $\mathcal{N}(\mu, \sigma)$ to obtain the mean and stander deviation. Where: μ is the sum of the individual activities and σ is set to 1 and can be changed.

Afterwards, everything is put in a Dataframe to keep all data well organised.

```
lab_μ,lab_σ = go2lab(rng,constructs,Toy_data);
#lab_μ,lab_σ = go2lab(rng,constructs,Toy_datan,repeat = 3 , sigma = 1);
df = DataFrame(constructs = constructs,μ = lab_μ, σ = lab_σ)
```

	constructs		
	Unordere…	Float64	Float64
1	Any[Mod{String}(”h”) Mod{String}(”e”) Mod{String}(”a”)]	8.28173	1.26088
2	Any[Mod{String}(”i”) Mod{String}(”f”) Mod{String}(”c”)]	6.32879	2.28096
3	Any[Mod{String}(”i”) Mod{String}(”c”) Mod{String}(”b”)]	5.81037	0.521044

1.3.5 Start Frist Cycle

In bayesian optimisation, the same steps are repeated in cycles. The first two cycles are explicitly given as an example

Fit Model: Cycle 1 The Bayesian linear regression is implemented in BayesianLinearRegressors.jl package. A simple wrapper made to use this model in our ecosystem.

```
Model_output = BO_Lin_wrapper(df.constructs,df.μ,df.σ,mod);
```

Sampling step: Cycle 1 The sampling step needs some additional tweak to deal with larger design space. For now, the entire space is first generated. The know constructs are removed and afterwards used in a Thomson sampler.

```
Unseen_space = filter(x-> !(x in constructs) , getspace(design,full = true).space);
new_constructs = Linear_TS_sampling(Model_output,mod,Unseen_space,n_samples);
```

1.3.6 Evaluation: Cycle 1

Evaluated the new proposed construct in the lab and join to the existing data frame.

```
lab_μ,lab_σ = go2lab(rng,new_constructs,Toy_data);
new_df = DataFrame(constructs = new_constructs,μ = lab_μ, σ = lab_σ);
append!(df,new_df);
```

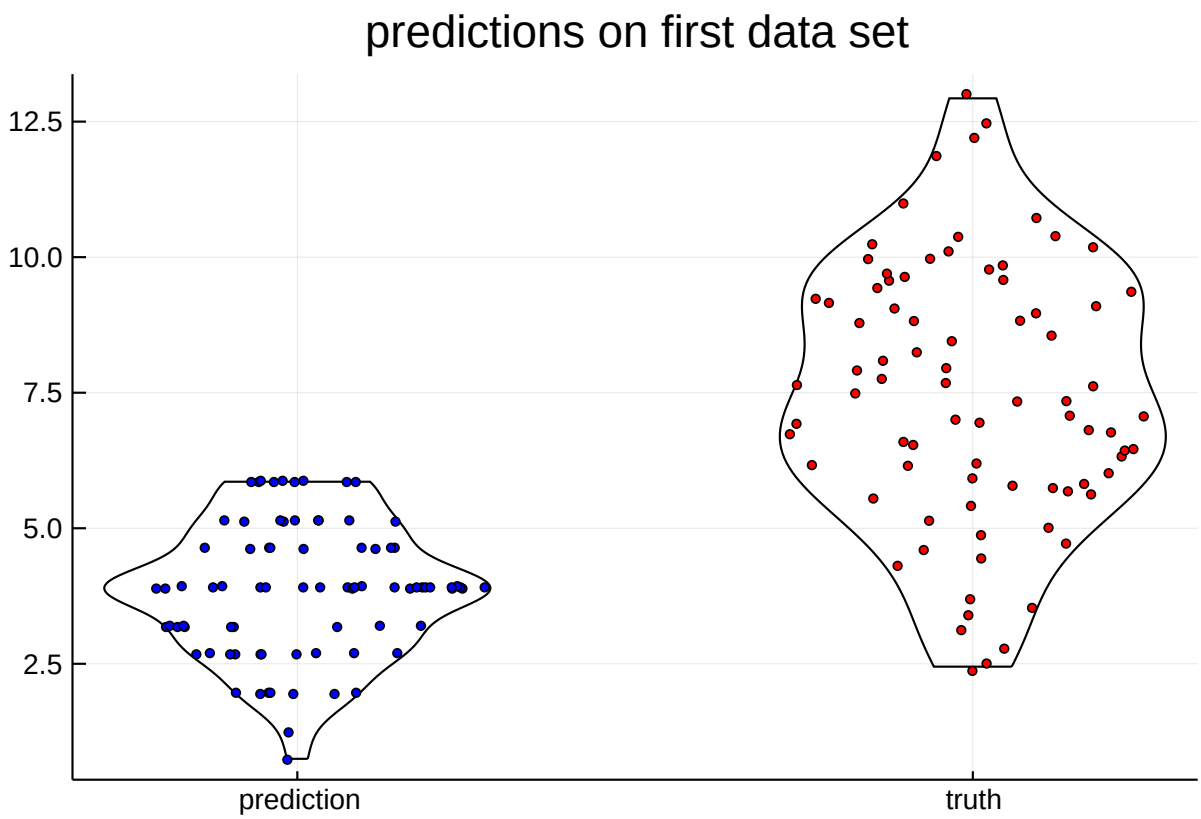
1.3.7 End cylce 1

The first round of the optimisation process is finished. We sample some starting points, fed them to the mode, and used this model to select some new data points. The new constructs are evaluated and can now be used in the second round of the model.

1.3.8 Extra: Evaluation of the model: Cycle 1

The performance of the model is not the priority in the optimisation set up. Still the model needs to able predict the activities to some extend. To avoided rigorous model evaluation, simple visualisation of the model predictions can give a first impression of prediction capacities of the model. This step isn't obligatory in the optimisation procedure. In this case, we see that the prediction are bad, which was expected because only 3 data points were used.

```
truth = map(x->ground_truth(x,Toy_data),Unseen_space);
pred_1 = BOMoD.make_predictions(Model_output,Unseen_space, mod);
gr()
# plot prediciton
violin(["prediction"],pred_1 ,marker=(0.2,stroke(0)),fillalpha = 0,label= "", title
="predictions on first data set")
dotplot!(["prediction"],pred_1,marker = :blue ,markeralpha = 1 ,markersize = 3 ,label =
"")
# plot true values
violin!(["truth"],truth ,marker=(0.2,stroke(0)),fillalpha = 0,label= "", title
="predictions on first data set")
dotplot!(["truth"],truth,marker = :red ,markeralpha = 1 ,markersize = 3 ,label = "")
```



1.3.9 Start Second Cycle round 2 (Still some questions)

Fit Model: Cycle 2 The second round is almost identical to the first round except there is now first sampling step Currently, it is not clear what is the best approach to integrated these new data points. The different options are work out underneath.

Repeat 2.5 The most stride froward option, repeat the same model 2.4 but with more data, which implies that the same prior is used as in 2.4. Is assumed to be correct doesn't recycle the calculation work done is the first step.

```
Model_output_2a = BO_Lin_wrapper(df.constructs,df.μ,df.σ,mod);
```

Update the prior The posterior of first-round becomes the prior of the second round. Only the new data points are fed to the model to updated the posterior. This step seems the most logical step and prevents that all previous sample data points need to be recalculated. Speciality with large datasets and more complex models this could speed up the optimisation. Still not sure if this is the correct way to go.

```
#Model_output_2b =
Update_BO_Lin_wrapper(Model_output,df_new.constructs,df_new.μ,df_new.σ,mod)
```

1.3.10 Combination of a and b

Finally, the third option is the combination of the two previous options. We update prior of the model based on the first model evaluation. Then all current now data points are fed to the model.

```
#Model_output_2c = Update_BO_Lin_wrapper(Model_output,df.constructs,df.μ,df.σ,mod)
```

The next steps are identical to the first round. To complete the second round, they are explicitly given. We used option 3.4.a just to complete this first version. The best of the above section will be used eventually.

1.3.11 Sampling step: Cycle 2

```
Unseen_space = filter!(x-> !(x in constructs) , Unseen_space);
new_constructs = Linear_TS_sampling(Model_output_2a,mod,Unseen_space,n_samples);
```

1.3.12 Evaluation: Cycle 2

Evaluated the new proposed construct in the lab and join to the data frame.

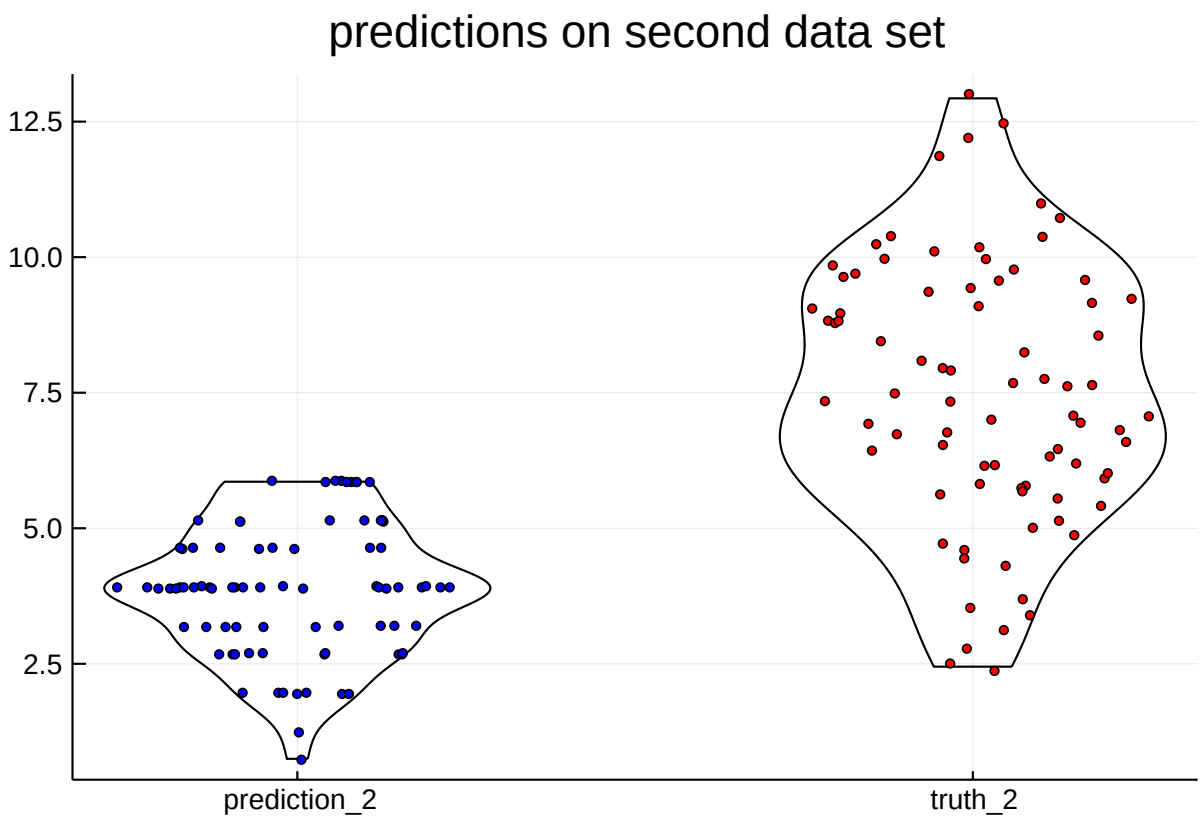
```
lab_μ,lab_σ = go2lab(rng,new_constructs,Toy_data);
new_df = DataFrame(constructs = new_constructs,μ = lab_μ, σ = lab_σ);
append!(df,new_df);
```

1.3.13 End cycle 2

The second round of the optimisation process is finished. This procedure should be repeated multiple times. The exact number of repetitions depends on the goal of the experiment and the available budget.

1.3.14 Extra Evaluation of the model: 2

```
truth_2 = map(x->ground_truth(x,Toy_data),Unseen_space);
pred_2 =  BOMoD.make_predictions(Model_output,Unseen_space, mod);
gr()
# plot prediciton
violin(["prediction_2"],pred_1 ,marker=(0.2,stroke(0)),fillalpha = 0,label= "", title
="predictions on second data set")
dotplot!(["prediction_2"],pred_1,marker = :blue ,markeralpha = 1 ,markersize = 3 ,label
= "")
# plot true values
violin!(["truth_2"],truth ,marker=(0.2,stroke(0)),fillalpha = 0,label= "")
dotplot!(["truth_2"],truth,marker = :red ,markeralpha = 1 ,markersize = 3 ,label = "")
```



1.4 Part 3: Example

In this section, the process is run in multiple cycles, and the convergence is evaluated. To know if the best construct is found, all constructs should be evaluated, but this is what we try to avoid in the first place. Still, the convergence plot can help out. I no improvement are found in the last couple of cycles the convergence plot flattens which indicates that a good variant is found. Still, This conversion doesn't guarantee the optimal is found, and the model can be stuck in a local maximum.

1.4.1 Setting of the example

Changing the setting below allows you to evaluate different design space with differents sampling setting.

```
length_constructs = 4;
pre_mod = ["a","b","c","d","e","f","g","h","i","j","k","l"];
order = false;
n_first = 3;
n_samples = 3;
n_cycles = 20;
mod = group_mod(pre_mod);
design = construct_design(mod,length_constructs,order = order);
```

1.4.2 Update toy data

Updates the toy data for the given settings.

```
rng  = MersenneTwister();
activities = rand(rng,Uniform(0,5),length(mod));
Toy_data = Dict(mod .=> activities);
```

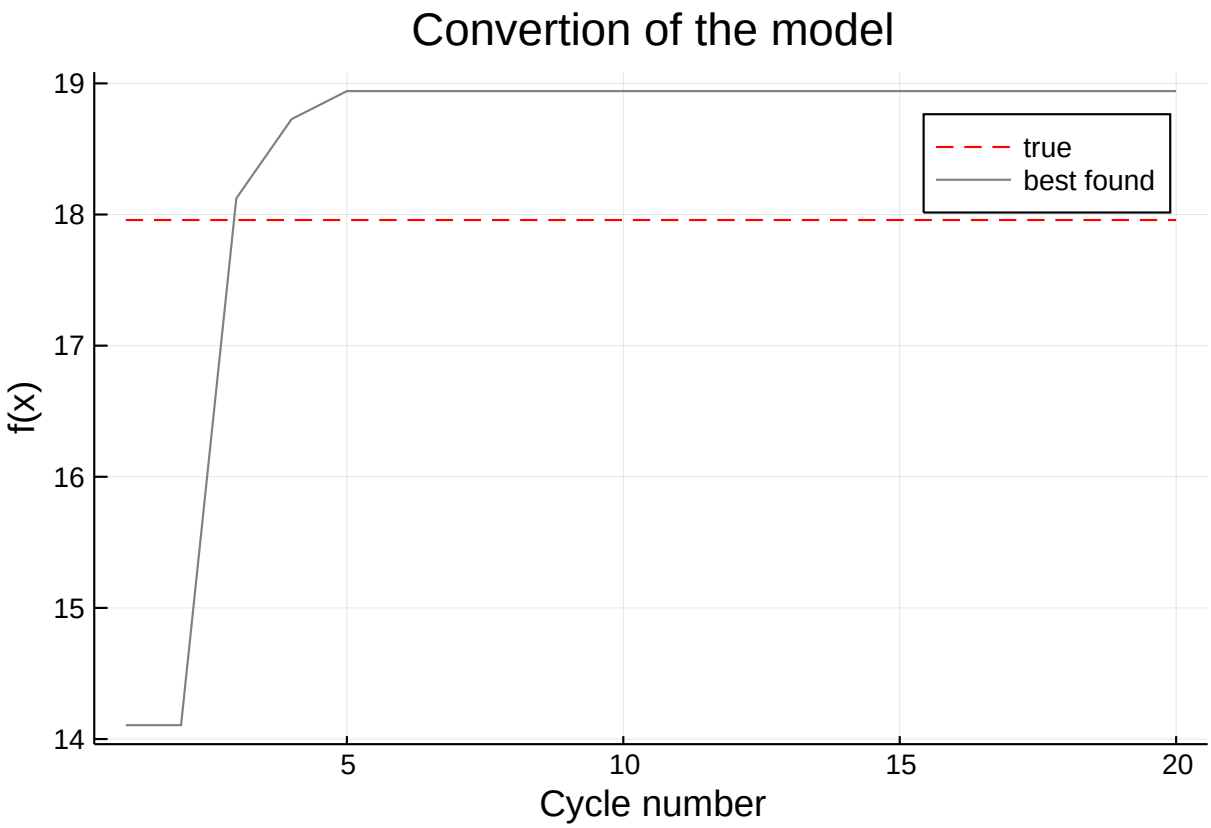
1.4.3 Exection of the example

Run the given example

```
max_cycle = model_linear(design,n_first,n_samples,Toy_data,n_cycles);
real_truth = sort(collect(values(Toy_data)),rev= true)[1:length_constructs] |> sum;
```

1.4.4 Generating plot

```
gr()
plt = plot(xaxis = "Cycle number",yaxis = "f(x)")
plot!(plt,collect(1:n_cycles),repeat([real_truth],n_cycles),linestyle = :dash,color =
:red,label ="true")
plot!(plt,collect(1:n_cycles),max_cycle,linestyle = :solid,color = :grey,title
="Convertion of the model",label ="best found")
```



The plot converges to a maximum value as expected. The fact that the best-found value is above the ground-truth is due to variance on the measurements with isn't plotted. In a real experiment, the red line is unknown.

1.5 Part 4: Evaluation

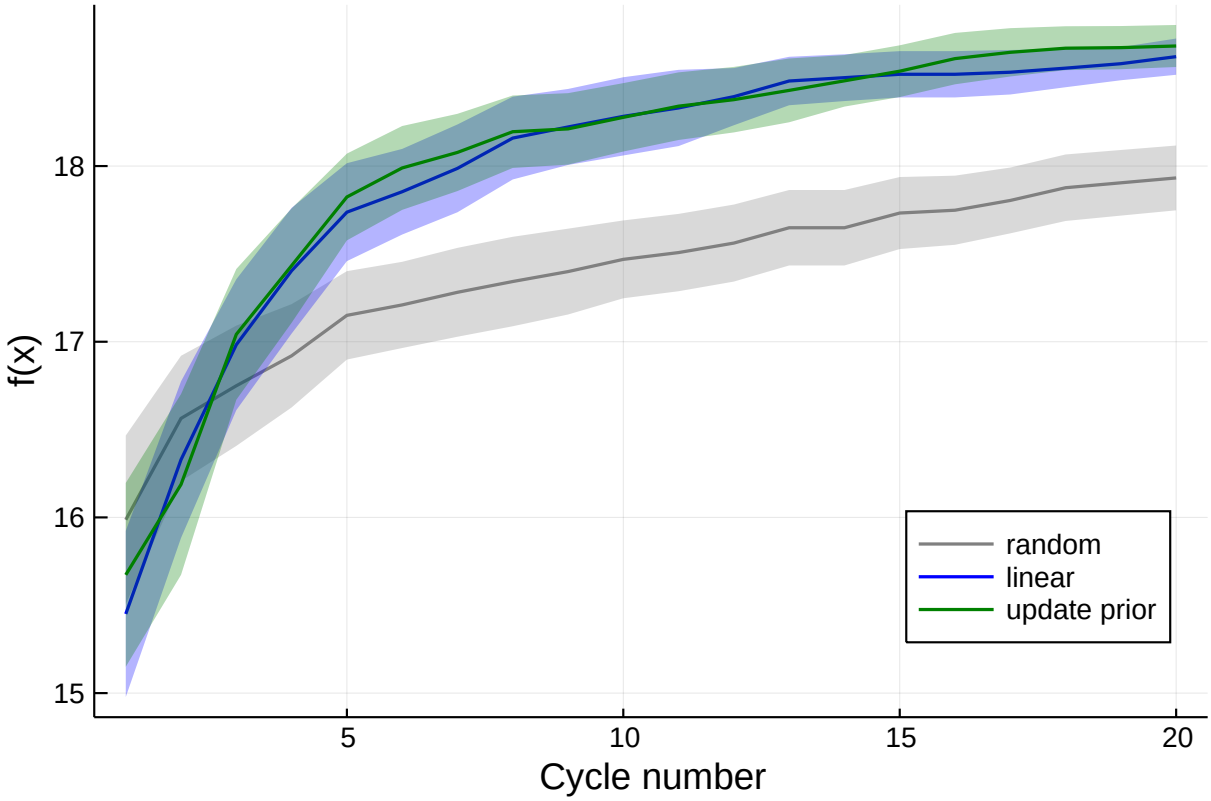
We made a great effort to make an entire optimisation process. The unanswered question is these efforts were useful. In this section, we compear the proposed model to a simple random baseline. The process is repeated multiple times because the performance depends on the starting constructs.

1.5.1 Run evaluation

```
n_iter = 50
n_cycles = 20
max_random = Matrix{Float64}(undef, n_iter, n_cycles)
max_linear = Matrix{Float64}(undef, n_iter, n_cycles)
max_update_pior = Matrix{Float64}(undef, n_iter, n_cycles)
for i in 1:n_iter
  max_random[i,:] = model_random(design,n_first,n_samples,Toy_data,n_cycles);
  max_linear[i,:] = model_linear(design,n_first,n_samples,Toy_data,n_cycles);
  max_update_pior[i,:] =
update_prior_model_linear(design,n_first,n_samples,Toy_data,n_cycles)
end
```

1.5.2 Plot results

```
gr()
plt = plot(xaxis = "Cycle number",yaxis = "f(x)")
models = [max_random,max_linear,max_update_pior]
names = ["random","linear","update prior"]
collors = [:grey , :blue ,:green]
for (m,n,c) in zip(models,names,collors)
  make_plot(plt,m,label=n,color = c)
end
plot(plt,legend =:bottomright)
```



The mean value of every cycle plotted in bold. The colour region indicates 2 times the stander error on this mean. **Question** is it correct to use the stander error instead of the standard deviation?

The curve chose that the model outperforms random sampler. The model that updates the prior and only use new data points to fit the model performance as good as the basic model, which was expected.

1.6 Conclusion

This was a first introduction in the BOMoD package wich visualises the concepts behind the package. All different step to set up your own project where covered and some additional features and evaluation were shown. If linear models aren't sufficient to solve your problem, then check out the other tutorials using Gaussian process as underlying model.