[Simulation Metamodeling](https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-1153-7_957" \t "_blank) – building and using surrogate models that can approximate results from more complicated simulation models – is an interesting approach to analyze results from complicated, computationally expensive simulation models. Metamodels are useful because they can yield good approximations of the original simulation model response variables using less computational resources. For an introduction to Metamodeling, refer to (Barton [2015](https://www.pedronl.com/post/des-metamodeling-splines-r-arena/#ref-Barton2015)).

To my knowledge, no Discrete-Event Simulation (DES) software provides metamodeling capabilities, and guidance on how to actually execute metamodeling [is scarce](https://www.google.com/search?q=discrete+event+simulation+metamodeling&oq=discrete+event+simulation+metamodeling). In this post, I’ll build a [Spline](https://en.wikipedia.org/wiki/Spline_(mathematics))-based simulation metamodel. This tutorial should be useful to advanced users of [Arena Simulation](https://www.arenasimulation.com/) who would be willing to give metamodeling a try.

**Why Splines?**

In my [previous post](https://www.pedronl.com/post/making-sense-of-models-with-metamodels-low-order-polynomialss-with-arena-and-r/), I briefly described the motivation for using metamodels to approximate simulation models results. Splines are among the useful techniques for metamodeling because: (i) they are relatively simple (they are piecewise-defined polynomials), and (ii) Unlike low-order polynomials, you can generally use them with a global sampling strategy (Barton and Meckesheimer [2006](https://www.pedronl.com/post/des-metamodeling-splines-r-arena/#ref-Barton2006)), meaning you can just sample a wide range of input values of your control variable and your model will still have a decent fit.

**Data Wrangling**

Before developing our metamodel, let’s first load the simulation data and do some data wrangling. For the details on this step, please refer to my [previous post](https://www.pedronl.com/post/making-sense-of-models-with-metamodels-low-order-polynomialss-with-arena-and-r/).

library(arena2r)

library(dplyr)

library(ggplot2)

library(readr)

sim\_results = arena2r::get\_simulation\_results(source = "2019-03-metamodeling/")

sim\_results$BatchSize = readr::parse\_number(as.character(sim\_results$Scenario))

sim\_results = subset(sim\_results, Statistic == "Entity 1.NumberOut")

head(sim\_results)

## Scenario Statistic Replication Value BatchSize

## 51 BatchSize200 Entity 1.NumberOut 1 9200 200

## 52 BatchSize200 Entity 1.NumberOut 2 9368 200

## 53 BatchSize200 Entity 1.NumberOut 3 9322 200

## 54 BatchSize200 Entity 1.NumberOut 4 9039 200

## 55 BatchSize200 Entity 1.NumberOut 5 9255 200

## 56 BatchSize200 Entity 1.NumberOut 6 9400 200

**Trying Splines**

You can build a spline model with the R’s standard linear model lm function. Instead of using the standard Y ~ X formula, we just have to use the bs() function from the splines package. Thus, our formula for our spline metamodel will be Y ~ bs(X).

## Now using Splines:

library(splines)

# Building a Spline Model:

spline\_model <-lm(Value ~ bs(BatchSize),data = sim\_results)

summary(spline\_model)

##

## Call:

## lm(formula = Value ~ bs(BatchSize), data = sim\_results)

##

## Residuals:

## Min 1Q Median 3Q Max

## -221.646 -30.896 1.011 46.969 268.354

##

## Coefficients:

## Estimate Std. Error t value Pr(>|t|)

## (Intercept) 9256.49 30.15 306.99 < 2e-16 \*\*\*

## bs(BatchSize)1 1312.17 102.27 12.83 < 2e-16 \*\*\*

## bs(BatchSize)2 1042.28 88.29 11.80 1.61e-15 \*\*\*

## bs(BatchSize)3 961.00 42.95 22.38 < 2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Residual standard error: 96.04 on 46 degrees of freedom

## Multiple R-squared: 0.9465, Adjusted R-squared: 0.943

## F-statistic: 271.1 on 3 and 46 DF, p-value: < 2.2e-16

Once you have your spline\_model, you can use the predict function to estimate the expected value of the response variable. Estimating what will be the Expected value of the Output variable with a Batch Size of 200 units is easy as:

predict(spline\_model,

newdata = data.frame(BatchSize = 200))

## 1

## 9256.489

**“Optimizing” with Splines**

Now that we have a spline model that can approximate our model results, we will use this model to find an “optimal” Batch Size which maximizes our Output Variable.

## Defining limits:

batchlims <- range(sim\_results$BatchSize)

# Generating Test Data

batch.grid<-seq(from=batchlims[1], to = batchlims[2])

# Using the metamodel:

spline\_data = data.frame(BatchSize = batch.grid,

Value = predict(spline\_model,

newdata = list(BatchSize=batch.grid))

)

# What is the Batch Size which "optimizes" the Output?

Optimum\_BatchSize <- spline\_data$BatchSize[which.max(spline\_data$Value)]

Output\_Value <- spline\_data$Value[which.max(spline\_data$Value)]

Optimum\_BatchSize

## [1] 331

The suggested batch size is 331. Is this a reasonable guess, based on our simulation runs? Let’s figure this out by plotting the simulation data, the spline function and the optimum value found.

# Let's plot again with the optimum batch size:

ggplot(sim\_results, mapping = aes(x = BatchSize, y = Value)) +

geom\_point() +

stat\_smooth(method = lm, formula = y ~ splines::bs(x)) +

geom\_vline(xintercept = Optimum\_BatchSize) +

geom\_text(aes(x=Optimum\_BatchSize,

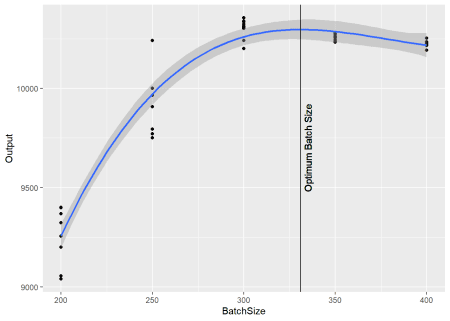
label="\nOptimum Batch Size", y=9700),

angle=90,

text=element\_text(size=11)

) +

labs(y = "Output")



Yes, definitely this is a good estimate! This plot encourages one to avoid going below 300 units, and suggests that going 350 and above is not a good idea either. The interesting pattern that the spline curve suggests is that increasing Batchsize not always increases Output, and that the output loss is not symetric.

Acknowledging these non-linear relationships is one of the outcomes I value at the end of a simulation project, and I hope that metamodeling will be an useful tool to you as well. Splines are a straightforward option to interpolate results from a simulation model, but there are other options out there. Future posts might explore other alternatives such as kriging metamodels, neural nets, and other techniques.

**References**

Barton, Russell R, and Martin Meckesheimer. 2006. “Metamodel-Based Simulation Optimization” 13 (06). <https://doi.org/10.1016/S0927-0507(06)13018-2>.

Barton, Russel R. 2015. “Tutorial: Simulation Metamodeling.” In *Proceedings of the 2015 Winter Simulation Conference*, 1765–79.