Multi-fidelity for "MDO using Gaussian Processes", an extract from "Aerospace System Analysis and Optimization in Uncertainty (3.3)"

Nicolas Garland, Rodolphe Le Riche, Yann Richet, Nicolas Durrande

2021

Disclaimer: This is an extract from the book "Aerospace System Analysis and Optimization in Uncertainty", which provide a generic methodology to build seemingly credible response functions for several standard numerical simulations types: mesh-based, Monte-Carlo and time-step simulators. The whole book is available at [https://doi.org/10.1007/978-3-030-39126-3].

[...] we consider a standard objective function modified for the purpose of studying multifidelity. A slight modification to the Branin function will be the basis of our tests:

$$\begin{aligned} \overline{x_1} &= 15x_1 - 5 \quad , \quad \overline{x_2} = 15x_2 \\ y(x) &= \left(\overline{x_2} - \frac{5\overline{x_1}^2}{4\pi^2} + \frac{5\overline{x_1}}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(\overline{x_1}) + 11 - \exp(-\frac{(\overline{x_1} - 0.5)^2}{15}) \end{aligned} \tag{1}$$

The function is plotted in Figure 1. The modification aims at having only 1 global optimum at at $(0.543, 0.150)$ and 2 local optima. The modified Branin function will soon be further transformed with different perturbations to emulate mesh-based, Monte-Carlo and time-step simulators, which correspond to three different convergence behaviors.

```
branin = function (x) {
  x = .as_x(x)
  x1 <- x[,1] * 15 - 5
  x2 <- x[,2] * 15
  (x2 - 5/(4 * pi^2) * (x1^2) + 5/pi * x1 - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(x1) + 10
}
branin.argmin = matrix(c(0.9616520, 0.15,0.1238946, 0.8166644,0.5427730, 0.15),ncol=2,byrow=TRUE)
branin_mod = function(x) matrix(1+branin(x)-(exp(-(.as_x(x)[,1]-0.5)^2/15)),ncol=1)
branin_mod.argmin = matrix(branin.argmin[ which.min(branin_mod(branin.argmin)),],ncol=2)
f = branin_mod
```

We propose some modifications of this analytical function to build more complex functions that mimic the behavior of real-world simulations response surface. These ,modifications are generic and should be transposed to other analytical functions.

Mesh-based simulations (like finite-elements solvers) are mainly converging smoothly with the mesh size. They yield objectives that evolve continuously with the level of details and tend to an asymptotic objective function. Of course, this characterization of mesh-based simulations neglects possible numerical instabilities that occur with changes in discretization. The smooth convergence is emulated as a ponderation between the asymptotic objective function and a continuous perturbation (a quadratic polynomial). The ponderation is a logarithm of the number of nodes (see Figure 2).

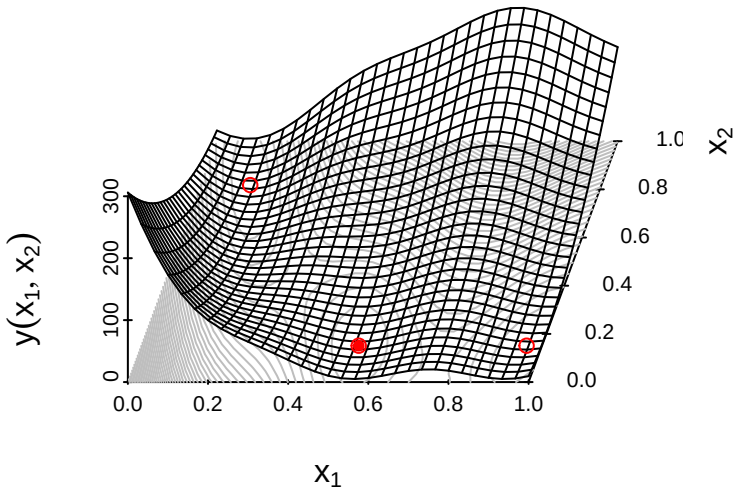

Figure 1: The modified Branin function of formula (1) which serves as the simulation of highest fidelity. The filled bullet is the global optimum, the empty bullets are the local optima.

$$y_{mesh}(x, n_{nodes}) = y(x) \times min(1, n_{nodes}) + 100 \sum_{i=1}^{n} (x_i - 0.5)^2 \times (1 - min(1, n_{nodes}))$$

```
#' @test x = matrix(runif(10),nrow=5); f_mesh(x,5)[1]; f_mesh(x[1,],5)
#' @test x = matrix(runif(10),nrow=5); f_mesh(x,1E8)/f(x)
f_mesh = function(x,nodes_number) {
  x = .as_x(x)
  nodes_dist = log10(nodes_number)/(2^3) # for 3D finite-difference code
  a = pmin(1,nodes_dist) # threshold of 1E8 nodes
  return(f(x) * a + 100*rowSums((x-.5)^2) * (1-a))
}
```

Monte Carlo-based simulations are converging with an added white noise which decreases with the size of the random sample used. This noise, standing for the simulation error, follows the central limit theorem and its variance decreases in 1/(number of samples). Moreover, the white noise has no correlation in the space of the parameters (x_1, x_2). The effect of such fidelity levels is illustrated in Figure 3.

$$y_{MC}(x, n_{particles}) = y(x) + 100 \times \mathcal{N}(0, 1/\sqrt{n_{particles}})$$

```
#' @test x = matrix(runif(10),nrow=5); set.seed(1); f_mc(x,5)[1]; set.seed(1); f_mc(x[1,],5)
#' @test x = matrix(runif(10),nrow=5); f_mc(x,1E8)/f(x)
f_mc = function(x,mc_sample_size) {
  x = .as_x(x)
  sd = 1/sqrt(mc_sample_size)
  return(f(x) + 100*rnorm(nrow(x),0,sd))
}
```

In the two previous examples, the simulation fidelity describes the degree of convergence of a simulation. Cokriging is also useful for discrete iterative simulations where the results of one step give the boundary conditions of the next one. Time or space iterated solvers are typical cases. Such *time dependent simulations* (like a discrete-time iterative MDO solver) are not converging toward an asymptotic solution, in the sense that the ending "time" (which may also be another dimension) is not approximated by previous times with

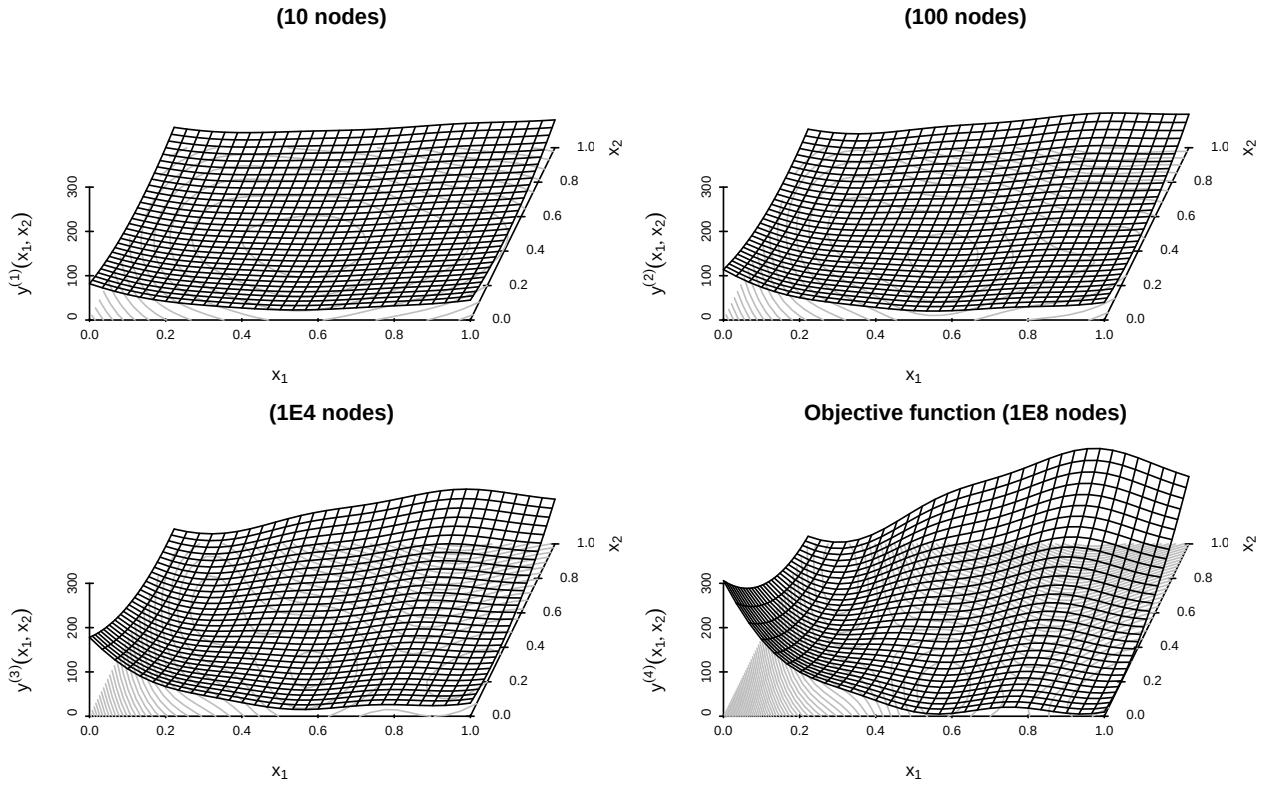

Figure 2: Emulation of mesh-based multifidelity simulations by continuous perturbations of the Branin function.

an "error" that decreases with steps. Nevertheless, the intrinsic Markovian behavior of such simulation is well suited to cokriging models. A third analytical test case is made of 4 steps of an autoregressive process (AR1) whose last iteration is the "high-fidelity" Branin objective function (see Figure 4).

$$y_{AR}(x,t) = y(x) - 80 \times \sum_{i=4}^{t}(4-i) \times \sum_{i=1}^{n}(x_i - 0.5)^2 - 80$$

```
#' @test x = matrix(runif(10),nrow=5); f_time(x,2)[1];  f_time(x[1,],2)
#' @test x = matrix(runif(10),nrow=5); f_time(x,4)/f(x)
f_time = function(x,time_step) {
  x = .as_x(x)
  eps = function(t) 80*(4-t) * (rowSums((x-.5)^2) ) -80 #+ (-1)^(t) * 60
  return(f(x) - rowSums(matrix(sapply(4:time_step,eps),ncol=length(4:time_step))) -80)
}
```

Add-on: iterative regularization

$$y_{swell}(x,s) = y(x^s)$$

```
#' @test x = matrix(runif(10),nrow=5); set.seed(1); f_swell(x,5)[1]; set.seed(1); f_swell(x[1,],5)
#' @test x = matrix(runif(10),nrow=5); f_swell(x,4)/f(x)
f_swell = function(x,swell) {
  x = .as_x(x)
```

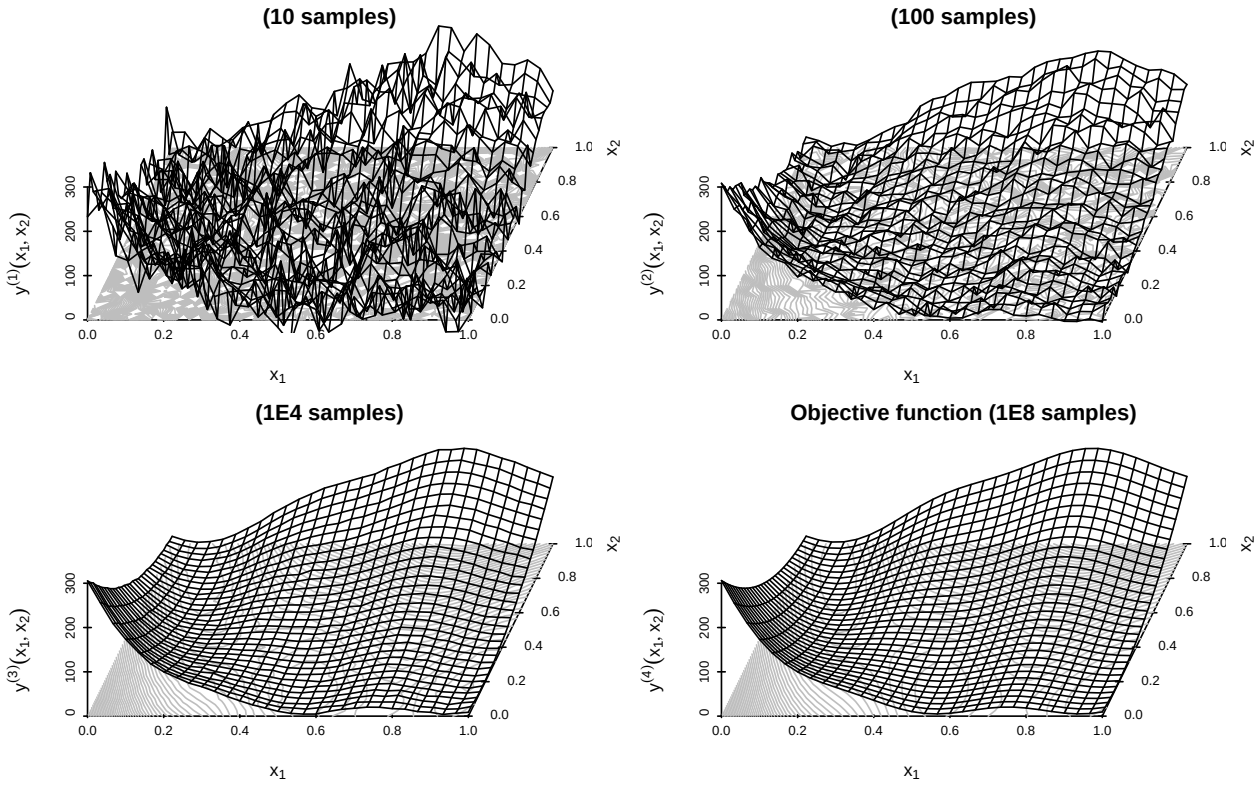

Figure 3: Emulation of Monte-Carlo based multifidelity simulations by added white noise perturbations of the Branin function.

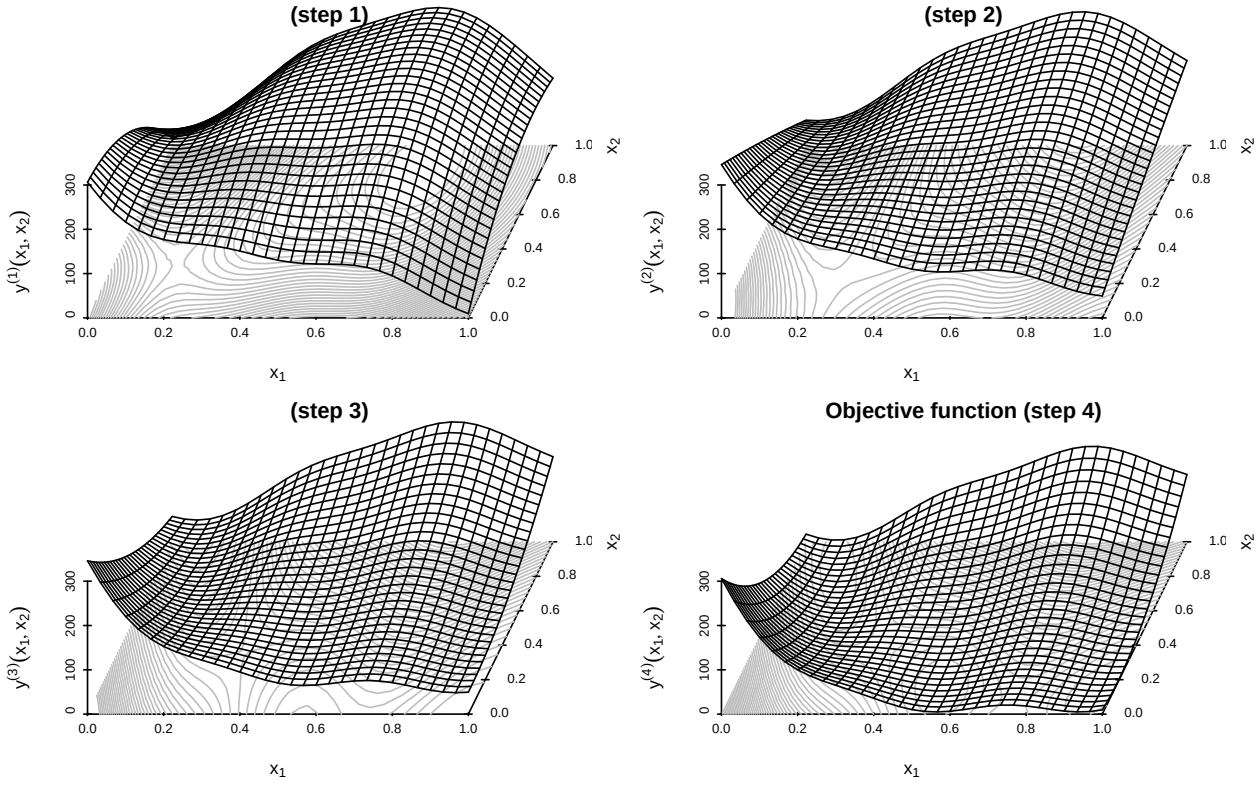

Figure 4: Illustration of the 4 AR time steps that make the third analytical test case.

```
  return(f(x^swell))
}
```

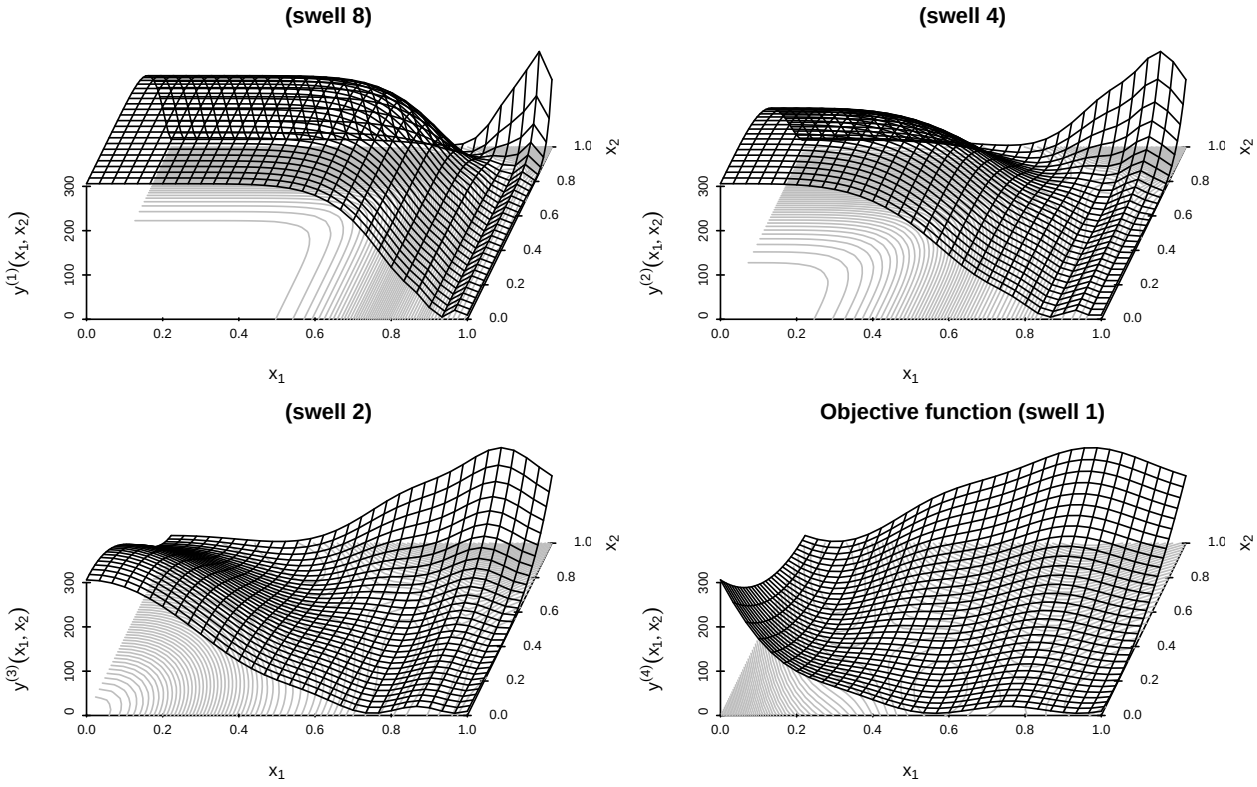

Figure 5: Emulation of 'iterative regularization' based simulations by non-isotropic shifting of the Branin function.