An Exposé on Surrogate Problems in Mathematical Optimization

Alexander Iannantuono

November 2020

1 An Introduction to Surrogates

In the first chapter of the textbook [1], some of you might have been asked to answer exercise 1.4 which has as a question:

$$\min_x \left\{ \sum_{i=1}^{n} |x_i| : x \in \mathbb{R}^n, c(x) \leq 0 \right\} \tag{1}$$

where $c(x) \in \mathcal{C}^1$. For those of you that were required to solve it, recall that the trick is to reformulate the problem in such a way that the objective function will become smooth (in comparison to the above which is not). This is an example of a surrogate function, since it is not the original problem, but it shares some similarities.

That being said, not all surrogate functions are simple (or complex) reformulations of the original problem. Unfortunately, some problems cannot be reformulated in such a way that we can capture all the information that is present while making the problem *nicer*[1] to solve. In the example above, we introduced two constraints for each x_i for $i = 1, \ldots, n$. Doing so replaces the need for the absolute value function for each x_i, as the constraints essentially impose the same 'positive-ness' that the absolute value does.

While the example in equation 1 captures the essence of why a surrogate problem is sometimes needed, the course textbook provides a similar definition:

Definition 1 (Surrogates)**.** *The problem*

$$\min_{x \in X \subset \mathbb{R}^n} \{\tilde{f}(x) : \tilde{c}(x) \leq 0\},$$

is said to be a surrogate for an optimization problem

$$\min_{x \in X \subset \mathbb{R}^n} \{f(x) : c(x) \leq 0\},$$

if $\tilde{f} : X \to \mathbb{R} \cup \{\infty\}$ and $\tilde{c} : X \to (\mathbb{R} \cup \{\infty\})^m$ share some similarities with f and c but are faster to evaluate. The functions $\tilde{f}$ and $\tilde{c}$ are said to be surrogates of the true functions f and c.

While the textbook claims that the vagueness of *sharing some similarities* is intentional, it intuitively makes sense why this is so: providing a rigid definition of similarities would limit the scope of the usage of surrogate functions. This could be in circumstances where one *might* not know what the best approach would be in terms of developing a surrogate. I say this since, as we will see, there are many ways of developing a surrogate due to the vague definition.

In the example from the homework problem, there is clearly similarities to the problem, but is the new problem technically faster to solve? Indeed, since now that the problem is smooth, there are many more off-the-shelf solves that we can use to solve the problem efficiently.

[1] This term has various relative meanings that change, so take this with a grain of salt.

1.1 Developing a Surrogate

When faced with a certain problem, especially a real-world problem, there is an assumption that there is an underlying understanding of the optimization problem structure. Since this is the case, those tasked with solving it would know what can be modified (and to what degree) while still keeping the surrogate problem relevant to its original counterpart.

The example shown in equation 1 is indeed a surrogate, but there are many others examples that exist. Another *relatively simple* example of possible surrogate problem is to use approximations of functions, which is the main study of our project. Since an approximation function evidently, as its name suggests, *approximates* a function, then one would assume that it must share some similarities with the original function that it intends to model via apprixmation.

2 Our Work

The main task of our project is to explore the topic of surrogates using the NOMAD software framework developed at GERAD at L'École Polytechnique in Montréal, Québec. It is a framework that specializes in black-box optimization, and can be interfaced with using the command line, MATLAB, Python or any programming language of your choice (as long as it can perform arithmetic operations). In this project, we implemented some test problems used by Müller et al. in [2].

To summarize blackbox optimization, a blackbox is a function in which we do not know its internal structure or representation. We can only query the 'box' (or function) with an input, and receive the outputs of that given input. Optimization of the said blackbox is simply finding the set of inputs in which the function is optimized, whether this be maximal or minimal.

To reiterate, surrogate functions, on the other hand, are functions that share some similarities with the blackbox functions defining problem but are cheaper to compute. In other words, they are useful when the true blackbox functions are costly to evaluate.

Two types of surrogates may be distinguished:

- Non-adaptive surrogates are defined by the user at the beginning of an optimization and do not evolve during the algorithm execution.
- Adaptive surrogates, on the contrary, are automatically computed and updated during the execution from past evaluations.

NOMAD currently exploits only the non-adaptive surrogates. Surrogates are used in the poll step: Before the poll trial points are evaluated on the true functions, they are evaluated on the surrogates. This list of points is sorted according to the surrogate values so that the most promising points are evaluated first during the true evaluations.

2.1 Reliability-Redundancy Problems

Our perusal of the internet for examples led us to [2] by the aforementioned author. This paper presents some problems used in blackbox optimization that were of particular interest to us. Their mathematical formulations as well as some slight discussion are presented below.

We would like to note that all of these problems are maximization problems, so we make use of the duality of maximization and minimization problems by finding the minimizer of $-f$ where f is our original objective value function.

2.1.1 The 19th Problem: A Bridge System

Suppose we are given a set of $n \in \mathbb{N}$ components $\mathcal{C} = \{C_i : i = 1, \ldots, n\}$ that are to be part of a system. An example layout of these components can be seen in Figure 1. For all intents and purposes, this abstraction of the problem seems to best be understood as if it were an electrical circuit where we need to go from one side of the circuit to the other.

In this specific problem, the components that are required to function together to keep the system running can be represented as a set of subsets of the component set

$$\{\{C_1, C_2\}, \{C_3, C_4\}, \{C_1, C_4, C_5\}, \{C_2, C_3, C_5\}, \{C_1, C_2, C_3, C_4, C_5\}\}.$$

Sticking to the analogy of a circuit, scenarios in which the system fails due to the inability to have 'current' flow is also a set of subsets of $\mathcal{C}$

$$\{\{C_1, C_2, C_3, C_4\}, \{C_1, C_2, C_3, C_5\}, \{C_1, C_2, C_4, C_5\}, \{C_1, C_3, C_4, C_5\}, \{C_2, C_3, C_4, C_5\}\}.$$

To keep the system running, we are permitted to add additional components of type i in parallel for each $i = 1, \ldots, 5$. In the optimization problem that we are looking to solve, this is the integer valued u_i for $i = 1, \ldots, 5$. However, there is a trade-off of redundancy to cost. These considerations can be taken care of in the upper bounds of the $u \in \mathbb{Z}^5$ variable. As a note, in a real-world scenario, the space that a component requires in the system might also be something to give thought to. One can assume that this has been implicitly taken into consideration through the upper bound on the u_i's.

The second variable is the reliability of the parallelized subsystem which is what $x_i \in \mathbb{R}$ denotes. For instance, we have the function $R(x_i, u_i)$ to be

$$R(x_i, u_i) = 1 - (1 - x_i)^{u_i}$$

Thus, we can see that if we were to fix x_i, as u_i increases (and even in the asymptotic case $u_i \to \infty$), we have that

$$\lim_{u_i \to \infty} (1 - x_i)^{u_i} = 0 \implies R(x_i, u_i) \to 1.$$

Despite this small digression, I think it provides more insight into the fact that as we increase the u_i variable, the parallelized i-th subsystem becomes more reliable, and if we look at x_i as the variable, and u_i fixed, a larger x_i is more desirable, as this would mean that $1 - x_i$ is closer to 0. I invite you to determine why this would make the subsystem more reliable as well (hint: it's very similar to the above)!

Written out mathematically, the problem is defined as follows:

$$\begin{aligned} \max_{x,u} f(x, u) = \max_{x,u} \quad & R_1R_2 + R_3R_4 + R_1R_4R_5 + R_2R_3R_5 + 2R_1R_2R_3R_4R_5 \\ & -R_1R_2R_3R_4 - R_1R_2R_3R_5 - R_1R_2R_4R_5 - R_1R_3R_4R_5 - R_2R_3R_4R_5 \end{aligned} \tag{19a}$$

such that

$$\sum_{i=1}^{5} p_i u_i^2 \leq P \tag{19b}$$

$$\sum_{i=1}^{5} c_i(x_i) \left(u_i + \exp\left(\frac{u_i}{4}\right)\right) \leq C \tag{19c}$$

$$\sum_{i=1}^{5} w_i u_i \exp\left(\frac{u_i}{4}\right) \leq W \tag{19d}$$

$$0 \leq x_i \leq 1 - 10^{-6} \quad i = 1, \ldots, 5 \tag{19e}$$

$$u_i \in \{1, 2, \ldots, 10\} \quad i = 1, \ldots, 5 \tag{19f}$$

With $P = 110, W = 200, C = 175$ and the cost reliability relation $c_i(x_i)$ function is described as

$$c_i(x_i) = \alpha_i \left(\frac{-t}{\log x_i} \right)^{-\beta_i} \quad i = 1, \ldots, 5.$$

The parameter t describes the time that the system needs to work and in the case of these problems are set to $t = 1000$. The $\alpha_i, \beta_i, w_i, p_i$ are all parameters that are given in the Table 1. For this problem, the paper states that the best known objective function value is 0.999659. An interesting thing to note is that $x_i \geq 0$ but $x_i \to 0 \implies \log x_i \to -\infty$, so we had to modify this in our formulation of the problem, and we set a lower bound to $x_i \geq 10^{-6}$.

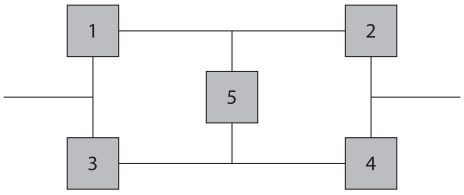

Figure 1: An abstract representation of the bridge system from Problem 19.

i	$\alpha_i \times 10^5$	p_i	w_i	b_i
1	2.330	1	7	1.5
2	1.450	2	8	1.5
3	0.541	3	8	1.5
4	8.050	4	6	1.5
5	1.950	2	9	1.5

Table 1: Data for the bridge system problem

2.1.2 The 20th Problem: An Overload Protection System

This problem is described as a protective system for a gas turbine that is modeled as a seres system with four components. In this case, it has four components that in the real-world scenario are valves. If there is a detection of overspeed from some other system, these valves must shut off the supply of fueld to the gas turbine. Thus, if one valve shuts, they all do. This can be defined as

$$\max_{x,u} f(x,u) = \max_{x,u} \prod_{i=1}^{4} R_i \tag{20a}$$

such that

$$\sum_{i=1}^{5} p_i u_i^2 \leq P \tag{20b}$$

$$\sum_{i=1}^{5} c_i(x_i) \left(u_i + \exp\left(\frac{u_i}{4}\right)\right) \leq C \tag{20c}$$

$$\sum_{i=1}^{5} w_i u_i \exp\left(\frac{u_i}{4}\right) \leq W \tag{20d}$$

$$0.5 \leq x_i \leq 1 - 10^{-6} \quad i = 1, \ldots, 5 \tag{20e}$$

$$u_i \in \{1, 2, \ldots, 10\} \quad i = 1, \ldots, 5 \tag{20f}$$

With $P = 250, W = 500, C = 400$ and the cost reliability relation $c_i(x_i)$ function is described as

$$c_i(x_i) = \alpha_i \left(\frac{-t}{\log x_i}\right)^{-\beta_i} \quad i = 1, \ldots, 4.$$

In this problem, $t = 1000$ again and R_i is as above. Note that the lower bound on the x_i's are The $\alpha_i, \beta_i, w_i, p_i$ are all parameters that are given in the table 2. The best known objective function value is 0.999889.

i	$\alpha_i \times 10^5$	p_i	w_i	b_i
1	1.0	1	6	1.5
2	2.3	2	6	1.5
3	0.3	3	8	1.5
4	2.3	2	7	1.5

Table 2: Data for the overload protection problem

2.1.3 The 21st Problem: Revisiting the Bridge System

In this last problem, we have a system that contains a parallel sub-subsystem within one of its parallel branches. This can be viewed in 2. The formulation of the optimization problem to be solved is as follows

$$\max_{x,u} f(x,u) = \max_{x,u} 1 - (1 - R_1R_2)(1 - (1 - R_3)(1 - R_4)R_5) \tag{21a}$$

such that

$$\sum_{i=1}^{5} p_i u_i^2 \leq P \tag{21b}$$

$$\sum_{i=1}^{5} c_i(x_i) \left(u_i + \exp\left(\frac{u_i}{4}\right)\right) \leq C \tag{21c}$$

$$\sum_{i=1}^{5} w_i u_i \exp\left(\frac{u_i}{4}\right) \leq W \tag{21d}$$

$$0 \leq x_i \leq 1 - 10^{-6} \quad i = 1, \ldots, 5 \tag{21e}$$

$$u_i \in \{1, 2, \ldots, 10\} \quad i = 1, \ldots, 5 \tag{21f}$$

With $P = 180, W = 100, C = 175$ and the cost reliability relation $c_i(x_i)$ function is before once again. The parameter is $t = 1000$ again. The new $\alpha_i, \beta_i, w_i, p_i$ parameters are given in Table 3 for this problem. The paper states that the best known objective function value is 0.999725. In this problem, the same issue with the lower bound on the x_i's is present, so once again, we set the lower bound to $x_i \geq 10^{-6}$.

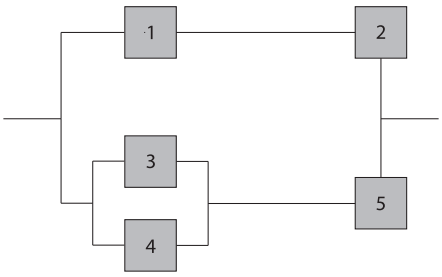

Figure 2: An abstract representation of the series-parallel bridge system from Problem 21.

2.2 Implementation of Sample Problems

For our study, we decided to implement the problems in both MATLAB and Python. This was done in order to verify correctness as well as explore the different interfaces that the NOMAD framework provides.

The surrogate that we used was to simplify the constraints in each problem that made use of $\exp(\frac{x_i}{4})$. To do this, we took a order-3 approximation of the function and evaluated that instead of the exponential. This is

i	$\alpha_i \times 10^5$	p_i	w_i	b_i
1	2.500	2	3.5	1.5
2	1.450	4	4.0	1.5
3	0.541	5	4.0	1.5
4	0.541	8	3.5	1.5
5	2.100	4	4.5	1.5

Table 3: Data for the series-parallel problem

due to the fact that the Maclaurin series for e^x is

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \implies \exp\left(\frac{x}{4}\right) = \sum_{k=0}^{\infty} \frac{\left(\frac{x}{4}\right)^k}{k!} = \sum_{k=0}^{\infty} \frac{x^k}{4^k k!}.$$

Another way to do this, is simply just use the third order approximation of e^x and divide the input by 4 as an input. This can be done since the function does not know otherwise!

Code is avaliable on GitHub if you care to take a look, any constructive comments, stylistically or otherwise, are always welcome.

2.2.1 MATLAB

A MATLAB version for NOMAD that was developed by Mark Abramson is also available. In MATLAB, when defining a surrogate in a function we pass one value in the options provided to the `nomad` function while computing the objective function, we pass a value to the parameter `has_sgte` (1 if the surrogate is available). The blackbox will first check that whether the variable that was passed in the function has surrogates available or not. If it is available, the surrogate function is then computed. Otherwise, the original optimization function is computed.

2.2.2 Python

In the Python implementation, there was a substantial use of the `numpy` Python package for numerical computations as it makes things more concise and 'pythonic' (which is an actual term that is used). The NOMAD framework does provide a Python interface (API), but it was not used. The problems were coded to comply with the NOMAD format. More can be read in the NOMAD user guide provided by the authors of the software.

There are also some uses of functional programming techniques such as 'reduce' for those familiar. Essentially, the reduce technique is to start with a list of elements, take the first two and do some operation with that results in one new object. We now have one less object to deal with in the list and now we take this new object with the next element and continue this process until we only end up with one of them. Most of these techniques are not really necessary, but taps into the more computer science side of things, and provides a different way of representing the problems.

3 Results

3.1 Some Data with Analysis

Given that the surrogate problem is not the true optimization problem in our case, it is expected that our results deviate from the results obtained in [2]. However, it is also the case that using the surrogate problem, the approximation of the constraint functions (via the exponential) the objective function values can be both better or worse. This is the case in our experiments.

As seen in table 4, there is an overall trend of a better objective function values as n increases. This is the case in Problem 21 with $n = 100$ blackbox evaluations in the surrogate problem, where the reported mean value over 30 trials for this same problem is 0.8132.

On the other hand, we see that for say, Problem 20, at $n = 200$ blackbox, our testing yields a mean objective value of 0.9992, while the reported mean for that problem is 0.9971. While, this does give a relative percent

	Problem 19				Problem 20				Problem 21			
	Python		MATLAB		Python		MATLAB		Python		MATLAB	
n	$\bar{x}$	s^2	$\bar{x}$	s^2	$\bar{x}$	s^2	$\bar{x}$	s^2	$\bar{x}$	s^2	$\bar{x}$	s^2
100	0.8948	4.46E-02	0.7267	3.65E-02	0.9788	3.27E-03	0.7719	7.41E-02	0.7621	9.80E-01	0.8246	1.80E-02
200	0.9854	1.09E-03	0.7610	3.65E-02	0.9992	5.99E-07	0.8371	8.63E-02	0.9796	1.36E-03	0.8907	4.23E-03
300	0.9987	2.08E-06	0.8143	7.64E-02	0.9997	4.24E-07	0.9000	2.17E-02	0.9969	4.46E-05	0.9096	2.19E-03

Table 4: Data obtained over $N = 30$ runs for Problems 19, 20, and 21

error of 0.0021%, the sample standard deviation is too small to have substantial significance in claiming that the randomness of the Mesh Adaptive Direct Search (MADS) algorithm that was used through seeding had much of an effect on the sample mean.

Another note to make is that a point that is feasible by our surrogate is not necessarily feasible to the original problem, thus this is of great importance when using surrogates. In our case, since we are using an order-3 approximation of e^x, the chances that this occurs is small, but still possible.

A final point to keep in mind is modification of the lower bounds on x in problems 19 and 21. Both allow for $x = 0$, but in the constraint contain $\log x_i$, which is an issue in itself. Taking the preventative measure of increasing the lower bound to 10^{-6}, our data shows that some of the solutions outputted by NOMAD outputs some feasible solutions whose variables have $x_i = 10^{-6}$ for some i. From here, one must make the decision of keeping that value, or perhaps making the argument that it should *technically* be zero. So as long as it satisfies the constraints, we could change those entries to zero. Of course, this is assuming that doing so would yield a higher objective function value.

3.2 Insights, Comments and Conclusive Remarks

When it comes to solving these problems, there are several ways one can conceive their measure of "speed" i.e. how fast can this problem be solved numerically using, in this case, the NOMAD framework. One crucial point to make it highly dependent on a few factors. First, it is important to decide how the blackbox is being called. In this project, the Python script was invoked (or triggered) by NOMAD via the command line and treated as a true black box. All NOMAD knows is what it provided as an input and what it received as an output.

However, in this case, it is very slow in reference to clock time. This has to do with how the command line works, but mainly how Python works. Python is an *interpreted* programming language. This means that upon running a script, the code inside is *parsed* by a program called a scanner, then a lexer, and then is eventually converted to machine code to be executed. This is done for every black box evaluation, which is really costly in time! On the other hand, a language like MATLAB, which is also an interpreter compiles code to C/C++ which is very fast. Without delving further into the topic of programming languages and compilers, this is something to keep in mind when using a framework like this. If this has piqued your interest on the topic, feel free to reach out for resources, or check out a course on compilers (or a textbook).

References

[1] Charles Audet and Warren Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing, 2017.

[2] Juliane Müller, Christine A. Shoemaker, and Robert Piché. "SO-MI: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems". In: *Computers & Operations Research* 40.5 (2013), pp. 1383–1400. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2012.08.022. URL: http://www.sciencedirect.com/science/article/pii/S0305054812001967.