User Guide for Leapfrogging Optimization
Deterministic or Stochastic Functions

30 September 2016 – R. Russell Rhinehart – russ@r3eda.com

User Guide for "2016 Generic LF Det or Stoch Function Optimizer 2016-09-30.xlsm"

This guide explains how to use the Excel VBA software provided on the site www.r3eda.com to optimize either a deterministic or a stochastic function using the Leapfrogging optimization algorithm. The software is designed for your investigative testing, so for convenience, it is limited to 20 decision variables; however, the procedure has no limit to the number of DVs. The algorithm is structured to seek a minimum, but could be easily changed to seek a maximum.

This guide explains how to enter VBA code for your model, how to how to select algorithm & convergence options, how to run the program, and how to interpret the results.

After several years of investigation, Leapfrogging (LF) was first published in 2012. See, Rhinehart, R. R., M. Su, and U. Manimegalai-Sridhar, "Leapfrogging and Synoptic Leapfrogging: a new optimization approach", Computers & Chemical Engineering, Vol. 40, 11 May 2012, pp. 67-81. Since then, it has been used on a range of applications including regression modeling, control, and design.

Although LF works for well-behaved applications (those with deterministic linear or quadratic-like responses), for them classic gradient-based or LP optimizers are faster. LF is a multiplayer direct-search algorithm designed to handle application difficulties that confound traditional gradient-based optimizers such as discontinuities, flat spots, integers, multiple optima, stochastic responses, etc.

Is Your Function Deterministic or Stochastic?

A deterministic function returns identically, exactly the same value regardless of when, where, or who runs the procedure. What is 3 times 7? Regardless of whether a computer or human does the calculation, in space or on the Earth, 10 years ago or today, the answer is 21 (base 10 numbers, of course). By contrast, a stochastic function does not return a single value. It returns a value that is a realization of one sample from a distribution. What number results in a roll of a standard die? The answer could be a 1, 2, 3, 4, 5, or 6; and each realization has an equal probability. The outcome is stochastic, not deterministic.

Commonly, engineers use deterministic models to design or analyze devices or processes. However, even though we do so, Nature is mostly stochastic. For instance, in a gas of constant temperature, we accept that the energy of any particular gas molecule continually rises and falls with the vagaries of collisions with other molecules. The molecule energies have a Maxwell distribution, and the "temperature" of any individual molecule does not remain constant in time. Although the natural mechanisms are stochastic when a limited number of elements is observed, when a measurement or response is influenced by millions of elements, the individual variation averages out, and the cause-and-effect relation appears deterministic.

As a familiar example of the averaging impact of large numbers, first roll a limited number of dice (say three) and average the numbers. Although an average of 1 is a possible realization of outcomes, it would be rare to have three 1s. The average will likely be influenced by both low values and high values and

tend in the 3 to 4 range. But, an average of either 1 or 6 when rolling three dice is not impossible; it has a probability of 0.00463.

However, if you roll many dice the possibility of extreme values is diminished. With 10,000 rolls, the average will be 3.5 (with a 95% range of ± 0.029, indicating the expected range of the average will be between 3.471 and 3.529). If you roll a trillion dice, the average will be 3.5 (with a 95% range of ± 0.00000029, indicating the expected range between 3.4999971 and 3.5000029). With classic 3-digit reporting that typifies permissible significant digits in most engineering, the outcome of a trillion rolls will appear to be the ideally expected 3.5 value.

For either truly deterministic functions or those for which averaging effects make it appear so, you can use conventional convergence criteria as appropriate for multiplayer algorithms such as the range of the OF-value of all team players, or the range of each DV within the team. Such metrics asymptotically approach zero as all team members converge on one spot. But, stochastic functions create an issue with determining convergence.

When dealing with queuing, scheduling, reliability, gaming, and forecasting; industrial engineers, business analysists, operations research, financial analysts, and infrastructure design folks commonly employ stochastic models to reveal the distribution of possible outcomes.

By contrast, design and analysis of aerospace, chemical, mechanical, electrical processes and devices has typically presumed that deterministic models of the undergraduate engineering programs are appropriate. However, the presumption that the basis for an application (perhaps corrosion rate, price of electricity, hours of use, environment temperature, weight of a person, barometric pressure, duty, raw material composition, spring stiffness, etc.) are both known and invariant over time or with successive use instances is not defensible. There is uncertainty associated with all of the “givens”; and progressively, engineering practice and business decisions are designing within uncertainty.

A common way to do this is to evaluate the function used for design or analysis with a realization value for each given. The realization value is randomly sampled from the uncertainty range expected for that variable. This is termed a Monte Carlo method, in reference to the procedure origins in gaming simulation. This takes what appears to be a deterministic function, and makes it stochastic. Any replicate evaluation will be subject to the vagaries of the sampling of the given values.

The issue in optimizing a stochastic function is that the function value is not reproducible. Even, with DV values unchanged, replicate realizations will return worse or better values than are representative. These misrepresentations can send the optimizer to a “best” solution that is an improbable outcome of a rare confluence of sampling realizations from the given value ranges, a phantom best, actually a bad decision.

A trick to minimize this problem is to run thousands of stochastic function realizations and observe the distribution of outcomes. Then have the optimizer seek the best average value, or minimize the 95% worst value, etc. However, this means that every trial solution needs thousands of function realizations, which wastes computer time for trial solutions that are not in the vicinity of the optimum, which is most of the optimization search. Further, although thousands of trials will temper the variability of a result, as the dice example revealed, that does not remove the stochastic attribute.

The solution for optimizing stochastic functions offered here is based on the publication Rhinehart, R. R., “Convergence Criterion in Optimization of Stochastic Processes”, Computers & Chemical Engineering, Vol.

68, 4 Sept 2014, pp 1-6: Use a multiplayer optimizer (such as Leapfrogging). Resample the OF value of the player that claims to be best. If it remains best use it; if not, use the second best. Claim convergence when the OF value of the worst player appears to be at steady state w.r.t. (with respect to) iteration.

If your function is deterministic, and you believe that there is no uncertainty on either the givens or the model representation of the application reality, then accept the best-of-N trials as the optimum and the DV* values to make a decision. However, I cannot imagine that such a contrived idealization represents any practical application. If there is uncertainty in either the givens or in the model parameter representation of the cause-and-effect DV to OF relations, then you should explore the impact of this uncertainty on the result.

This can be done several ways. You could accept the deterministic function DV*, and then explore the range of OF values that it would create with a range of realizations for the givens and parameter values. However, the deterministic DV* may not represent a best decision within uncertainty. A better approach is to include the uncertainty in your function. At each call, initialize the function with a realization of givens and model parameter values selected from sampling the expected distribution of those values. This creates a stochastic function. If you run enough optimizer trials, the best few will collectively represent realistic DV* and OF* values. The best of all of them just happens to represent one trial realization, not the whole truth about the application. I suggest reporting the best of N, but also report the range indicated by (perhaps the best 10) equivalently good solutions.

Entering Your Code

Whether deterministic or stochastic, you need to enter your function in the VBA function "OF_Value". Additionally, you need to edit the "Assign" and "Convergence_Test" subroutines as appropriate, and declare any variables that are specific to your function. If you are new to Excel VBA, visit this primer from my web site.
http://www.r3eda.com/wp-content/uploads/2016/06/r3eda-site-VBA-Primer-2016-06-21.pdf

There are two modules in the VBA project. The function and subroutine codes for your application are in Module "A_User_Routines". You'll need to edit code and variable declarations in Module A. The core algorithms for LF and display management are in Module "B_Main_and_Auxiliary". You should not need to edit these, but you are free to do so. They are all open to your view and editing.

I choose to use the "Option Explicit" declaration in the header in Modules "A_…" and "B_…". This means that all variables need to be explicitly declared. Accordingly, if you decide to use a model with different variable or coefficient names you need to declare them in the header of module "A_…". This is an aggravation, but it is a useful artifice in minimizing code development errors, and maximizing subsequent understanding and editing.

Be sure to not use variable names already used by the main and auxiliary program. See the global declarations of variable names in Module "B_…". The global declaration means that variables and values are common to all modules.

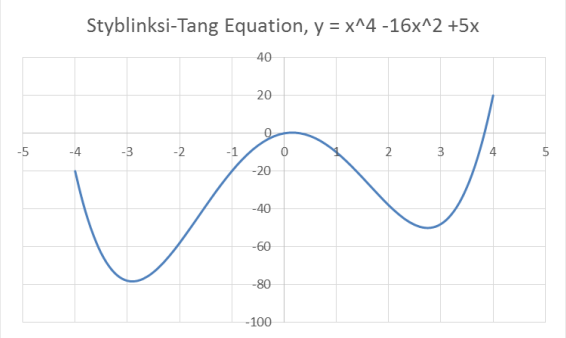
Save the file frequently while you are editing it. While editing, you'll run tests. Sometimes you can create a loop with no escape, and the program will not respond to your keyboard or mouse commands. If you

force-close an un-saved file, Excel will return to a past auto-saved version; and you may have lost recent edits.

This LF optimizer is set to seek a minimum. If you wish to find a maximum, the simplest approach is to set the function "OF_Value" as the negative of your objective function value. Alternately, you could edit the optimizer code to make it leap the lowest over the highest, to test convergence on the appropriate players, to output the appropriate player, etc.

Some constraints are functions of the decision variables. When they are, place the relations in the subroutine "Constraint_Test". However, some constraints are only recognized during model computation. For instance, a constraint may be related to an execution error such as square root or log of a negative number. Or for instance, a constraint may be related to an event along a path in space or time that is not encountered until the model integrates along to that spot. But, such non-obvious constraints are still hard constraints, even if there is no known relation that could specify the constraint, *a priori*, from the DV values. In such a case, include code in your "OF_Value" function that 1) sets Constraint = "FAIL" and 2) exits the function prior to encountering the execution error. Example code is included in the "2016 Generic LF Det or Stoch Function Optimizer 2016-09-17.xlsm" file.

The function in the current version is based on the Styblinski-Tang equation. $f(x) = x^4 - 16x^2 + 5x$. It is a simple and commonly used relation that has a local minimum of about -80 at about $x = -2.9$, and a local minimum of about -50 at about $x = +2.7$. Here is a graph of the equation. The function in the program is the sum of 6 individual relations.



$$OF = \sum f(x_i) = \sum ({x_i}^4 - 16{x_i}^2 + 5x_i)$$

This is not a particularly difficult challenge problem. But, it is non-quadratic, has inflections, and has two minima for each variable. Over an initialization x-range of -5 to +5 the region of attraction to either minima using a steepest-descent type algorithm from a random point is about 50%. As a result, a single TS (trial solution) steepest-descent optimizer, randomly initialized, will have a 50% chance of going to the global on any one trial. Then, for a 6-DV application, the probability that any randomly initialized single trial solution steepest-descent algorithm will find the global is $0.5^{\,6} \cong 1.6\%$. However, if any LF player lands on an x-value that has an OF less than about -50 (lands in the region of the global minima) then it will gather all players to the global. That portion of the range for a single x-variable is about 20%. If there are 10 players per dimension, then the probability that one of them is randomly initialized within the region guaranteed to find the 1-DV global is $p = 1 - (1 - 0.2)^{10} \cong 90\%$, not 50%. However, initial leap-overs permit additional full-range exploration. If the effective initial exploration includes both initialization and the first 10 iterations of leap-overs, then $p \cong 1 - (1 - 0.2)^{(10+10)} \cong 99.\%$. For a 6-DV global the probability of LF initialization placing a player in the vicinity of the global can be crudely estimated as $p \cong 1 - (1 - 0.2^6)^{60} \cong 0.4\%$. (This assumes the hyper-volume containing the global attractors is a hyper-rectangle. I think a hyper-sphere may be more realistic.) However, since initial leap-overs continue surface exploration, the probability of finding the global is increased. My trials reveal that the basic LF algorithm finds the global about 15% of the trials, which is ten times the probability that a steepest-descent optimizer will find it.

Running the Program

You enter data in the green-colored fields on the "Main" worksheet.

Column D, the 4th column – Algorithm Choices:

1. In Cell(4,4) enter the number of decision variables for your application. This needs to be an integer between 1 and 20 (inclusive).
2. In the following two cells [(5,4) and (6,4)] enter the first and last trial number. Normally, the count would start with the trial number 1, but if you stop the program on the 7th trial, and wish to restart it without losing the prior data output on the Main sheet, start on the 8th. However, data will be lost on the Results sheet. The optimizer will run as many trials as you would like, but I've limited the clear-old-data to 2,000 sets. So, the end trial number should be $\leq 2{,}000$. Cell(5,15) provides an estimate of the desired number of trials you should specify.
3. In Cell(7,4) enter the upper limit on the number of iterations. An iteration is defined as the number of feasible leap-overs equal to the number of dimensions. This limit will stop the optimizer without claiming convergence if it seem to be lost and unable to find a minimum.
4. Cell(8,4) is for the leap-to window size. Enter either the letter A, or a number between about 0.25 and 2.5. The nominal leap-to window is the same size as the leap-from window, with a Cell(8,4) value of 1. However, a larger value promotes more surface exploration and increases the likelihood of locating the global, but it makes convergence of the team of players take longer. A smaller value accelerates convergence, but sacrifices exploration. In the extreme of 0, all players will leap to the initial best, and converge there with no function exploration. If a value is greater than 2.718… (the base of the natural logarithm), the team will diverge. If you enter the letter A (for automatic) the algorithm starts with a leap-to window size value of 2, then after enough exploration iterations, as indicated in Cell(1,15), it shifts to a value of 1 and also shifts the leap-to window a quarter-distance back to the leap-from spot, to permit exploration in the vicinity of the team leader (not just on the other side). Mostly, I find the nominal 1 provides a reasonable balance of exploration and convergence speed. I recommend 1.
5. Enter the number of players in Cell(9,5). Nominally, it is the 10 times the number of DVs. You can use the default cell formula, or enter your own value. Higher numbers, such as 20 times the DV number, provide greater surface exploration, but take longer to have all players converge to a common spot. Conversely, lower numbers run faster, but with less probability of finding a global. Both the number of players and the leap-to window size affect both the probability of finding the global and the number of function evaluations for the team to converge. The nominal values of $window\ size\ = 1$ and $Nplayers = max\{20, 5N_{DV}\}$, or equivalently $window\ size\ = 0.5$ and $Nplayers = 10N_{DV}$ seem a best universal balance for efficiency and simplicity.
6. One approach to finding the global is to run many optimizer trials from randomized initializations, and use the best-of-N trials as the global. But, this means many optimization trials, and does not guarantee finding the global. An alternate is to initially search the OF response with many randomized DV locations, hopefully enough to locate the vicinity of the global, then use the optimizer to start from the best of all the explorations. If you enter a value in Cell(10,5) the optimizer will initialize with that many extra players, but it will start the optimization with the best subset of Nplayers defined in Cell(9,5). Cell(6,15) provides a recommended value for initializations.
7. In Cell(11,5), enter an integer to select the convergence criterion. The available options are listed in Column 11, Rows 4-9. The options for a deterministic function are "1" OF range (worst minus best), "2" the root-mean-square value of the OF of all players, and "3" the rms OF value relative

to the best OF. Since DVs can have a wide range of values and units, I did not include a convergence criterion based on DV values. However, you could include any sort of convergence criterion test by adding your code to the "Convergence_Test" subroutine of Model "B_...". For stochastic functions, the two tests provided are based on the approach to steady state w.r.t. iteration of either the worst OF value or the range of the best to worst, options "4" or "5". The method to determine steady state is that of Cao, S., and R. R. Rhinehart, "An Efficient Method for On-Line Identification of Steady-State," Journal of Process Control, Vol. 5, No 6, 1995, pp. 363-374.

8. Each convergence criterion requires a threshold value. Enter it in Cell(12,5). If you choose Criterion "1" or "2" your entry for the threshold value will be dependent on the value of the OF (which also depends on the units used). Alternately Criterion "3", relative rms value is dimensionless. Perhaps 0.000001 is a practicable value, which would claim convergence when the variation in the OF of all players is 6 orders of magnitude smaller than the OF value. However, if the optimum OF value is zero, this test will probably not claim convergence. Also, dimensionless is the threshold choice for either of the steady state Criteria "4", or "5". A value of 1.0 indicates moderately low confidence that SS has been attained. A value of 0.8 means a very high confidence, but it will require more iterations to achieve that confidence. I recommend a SS convergence value of 0.9 for exploratory trials, and 0.8 for high confidence.
9. Finally in this section, enter an integer of 0, 1, 2, … in Cell(13,5) to indicate the number of replicates that the optimizer uses to affirm the OF value of the best player. If the function is deterministic, enter 0, no replicates are needed. If the function is stochastic, you do not want it to follow an improbable path of best values to a phantom improbable best location. So, enter the number of replicates that you desire, the number of re-evaluations of the best player to keep it representing the best (which will be the worst of several replicates). Large values will steer the optimizer away from improbable occurring bad spots, but take additional function evaluations. If you believe it is easy to generate a bad value then a 1 or 2 here is a good value. Alternately, if you think a bad value is not likely to be obtained in just a few samples, or if you desire a high confidence that a bad value will be recognized you could enter 5 or 10, or more, for replicates. What is a right number? On any trial solution, if you want to be 90% confident ($c = 0.90$) that the replicates will find one of the worst quartile values ($f = 0.25$) then calculate the desired number of replicates as $N_{repllicates} = Round[\ln(1-c)/\ln(1-f)] - 1$. For example $c = 0.90$ and $f = 0.25$ results in 7 as the number of replicates. However, if the best remains the best it would be evaluated 7 more times. It is not unusual to have the best remain the best for several iterations. So, my experience is that a 1 or 2 replicates is adequate. Excessive replicates just makes the program take longer to run.

Columns H, J, & K, the 7th, 9th, and 10th columns – DV Choices:

1. Enter the names of your Decision Variables in Column H, the 7th column. The program will duplicate these variable names in the data output sections. The labels here do not need to match exactly the corresponding variable names in the VBA code.
2. Enter the expected range for each variable in Columns J and K, the 9th and 10th columns. Perhaps these should be what you expect the feasible range for each to be. It is best to enter ranges that will include the global optimum; but, if the function is relatively well-behaved this is not necessary. If all players are initialized on the side of a hill, the team will move downward to the single optimum. A large range of values will take the algorithm longer to converge, but will provide greater exploration, and a higher probability of detecting the global.

Column N, the 15th column – Probability Choices:

1. Cell(2,15) is for a characteristic value of the portion of any DV range that you think will draw the optimizer to the global best. I think that $f = 0.7$ is a good, nominal, value. For well-behaved functions, any initial DV value will lead to the optimum. Then $f = 1.0$. For others, perhaps only $f = 0.1$ fraction of the range will lead to the global. If there are numerical discretization issues then the vicinity of the global may be easy to find, but the exact minimum within the local ridges or pockets created by numerical artifacts may be a very small portion of the DV range, perhaps $f = 0.0001$. But, in this case the function does not represent the true minimum of the situation, just a trap in its vicinity due to numerical implementation choices. Specify values within (not including) the range of $0 < f < 1$.
2. Enter the desired confidence in Cell(3,15) that you wish the desired fraction to be found in at least one of N trials. The value is within the range $0 < c < 1$. Nominally reasonable values are $0.90 < c < 0.999$.
3. After multiple optimizer trials, you hope that the best-of-N solutions represents one from the best fraction of all possible solutions. Specify that desired fraction in Cell(4,15). I think that $f = 0.05$ is a good, nominal, value. For some functions any initial value will lead to the optimum, here $f = 1$. For others, in which the optimum is a difficult to find pinhole perhaps only 0.001 fraction of solutions will represent the global. Again, specify values within the range of $0 < f < 1$. However, recognize that this f is different from what you are specifying in Cell(2,15). The number of optimizer trials needed to be confident that at least one trial finds an OF value within one of the best fraction is calculated with the formula $N = \ln(1-c)\,/\ln(1-f)$. See Iyer, M. S., and R. R. Rhinehart, "A Method to Determine the Required Number of Neural Network Training Repetitions," IEEE Transactions on Neural Networks, Vol. 10, No. 2, March, 1999, pp. 427-432.

Several indications result from this data. If there are N number of DVs, and random initialization desires that at least one of M initializations will locate a player in the region that draws to the global with a desired confidence (perhaps $c = 0.99$) then $M_{initializations} = \ln(1-c)\,/\ln(1-f^{N_{DV}})$. The value is reported in Cell(6,15) and could be used to specify the number of initializations in Cell(10,4). See Manimegalai-Sridhar, U., A. Govindarajan, and R. R. Rhinehart, "Improved Initialization of Players in Leapfrogging Optimization", Computers & Chemical Engineering, Vol. 60, 2014, 426-429.

Also, with this data entered, Cell(5,15) calculates the number of trials needed to be c-confident that at least one of M trials identifies an OF value within the best f-fraction of all possible values. $N = \ln(1-c)\,/\ln(1-f)$. And, Cell(1,15) indicates the number of iterations required for the automated adjustment of the leap-to window to switch form an amplitude of 2 and base shift of 0 to an amplitude of 1 and base shift = 0.25. The formula is $N_{iteration\ to\ switch} = \left[\frac{\ln(1-c)}{\ln\left(1-f^{N_{DV}}\right)} - M_{players}\right]/N_{DV}$.

As a multiplayer algorithm, LF has a high probability of finding the global optimum. However, no optimization algorithm (even those named global optimizers) can guarantee that they will find the global on every run. There are many things that you can specify to improve the probability that the optimizer will find the global. Obvious options are: Option 1 – increase the number of players in Cell(9,4). And Option 2 – increase the window size factor in Cell(8,4). Both of these obvious solutions work, but both undesirably extend the number of iterations in the end-game (convergence of the team when an optimum has been found).

The Excel/VBA program provides three other options to enhance the probability that the optimizer finds the global, while seeking to minimize the end-game work. In Option 3 run the optimizer from N independent initializations, and for deterministic functions take the best of N as the global. For stochastic functions report the range of the best few. In this option the user specifies the confidence [Cell(3,15)] that in at least one of the N trials the result will belong to one of the best fraction [Cell(4,15)] of all possible OF values. The N calculated in Cell(5,15) would be the number of trials entered in Cell(4,6). In Option 4, place the letter A in Cell(8,4), which directs the program to start the LF algorithm with a leap-to window size that is twice that of the leap-from window. This provides area exploration. Then after sufficient number of iterations indicate that at least one player should have landed in the region of the global attractor, then the window size factor returns unity and the window shifts ¼ of the way back from the best toward the leap-from spot. Both actions accelerate end-game convergence. The iteration to switch modes from exploration to convergence is calculated in Cell(1,15). Finally, in Option 5, specify the desired number of initializations in cell(10,4). Then the optimizer will explore that many extra initial trial solutions, and take the best of all initial and extra players as the set of players to use in the LF algorithm. Here, since the LF team is initially located in the vicinity of the best (not all over the DV range), convergence is faster; but initialization takes longer.

Of course, you could have the program do all of the five options. But, I think best (considering maximizing the probability of finding the global with minimizing the number of function evaluations with little *a priori* function knowledge) is to use the standard for the number of players and window location, and either Option 3, 4, or 5.

My trials indicate that Option 4 (large initial leap-to window for exploration, then reduce for convergence) is a good plan, but even so, one trial may not find the global optimum, so also implement Option 3, and see the best-of-N trials.

A sixth option, not yet installed in the program is to use a coarse convergence criterion on the N trials, then take the best-of-N as the global and do a fine search on it. You could implement this easily by specifying N trials with a coarse convergence to have a high probability that the best-of-N is in the vicinity of the global. Then run one more trial initialized with the former best player as one team member [enter a Y in cell(9,15)] with the tight convergence threshold.

A seventh option, also not yet installed, is to run a sequence of randomly initialized trials; and, when a Bayesian probability analysis reveals a strong confidence that the global has been repeatedly found, end the number of trials. Snyman and Fatti (1987) reveal a method to determine when to stop new trials.

Column N, the 15th column – Observation Choices:

1. Enter a Y or N in Cell(8,15) to watch the progress. If Y, each time a new player becomes a best, the display is updated. If N, only the end-of-trial results are displayed. Choose Y if you wish to observe progress; but, realize that updating the display makes it take a longer time to complete a run.
2. If you wish the optimizer to initialize with the former best solution, then enter Y in Cell(8,15). If you think you have a reasonable idea of the best then this will accelerate convergence. But, if you do not know the vicinity of the global, then enter N, because a Y here may draw the optimizer repeatedly to the same local optimum.

3. If you wish to observe all players as the worst leaps over the best and acquires new DV values, then enter Y in Cell(9,15). I've found this useful when debugging and learning, but the program runs faster with a N.

Running the Programs

The button "Start Optimizer" initializes the data and starts the N optimization trials. If you wish to stop execution once started, use either the break key or the escape key. I find that the computer is more immediately responsive when I use the break key. For me, it is the key combination Fn-Break, but the pattern depends on your keyboard layout.

If you wish to enter your own DV values in Column I, the 8th column, to see what the OF is, do so, and press the "My Model" button.

If you wish to see the best of the N models listed on the "DataRESULT" sheet, press the "Best Model" button. When the program has finished all N trials, what is displayed is the last trial completed, not the best of all N trials.

Seeing and Interpreting the Results

Worksheet "Main" reveals trial solution and OF values, trial and iteration numbers, and convergence criterion value with each iteration that creates an improved TS, a new best player. These are displayed in the Rows 3 to 22 section of Columns 6 through 8.

Worksheet "Main" also reveals the trial OF* and DV* results, listed in Rows 24 and below, sequentially by trial number, shown in Column 1. Column 2 contains the number of iterations to claim convergence. But, more important as an indicator of optimizer work, is the number of function evaluations (NOFE) in Column 3. The yellow highlighted fields above the data in Column 3 are the average and standard deviation of the trial-to-trial NOFE values. Note that this average and standard deviation are for all trial data, not just those at the global optimum. If a particular trial did not converge, but was stopped due to excessive iterations, there is no entry in this output data section of the "Main" sheet. Final OF* and DV* values for each trial are listed in Columns 4 and after.

For a deterministic function, take the best-of-N results as the DV*. But, you should explore the impact of uncertainty on the givens and model structure.

For a stochastic function, the best-of-N will simply be the fortuitous best. I think it might be wiser to look at all DV* values from all of the solution sets that appear to represent the global. They should all be indicating the same DV* vicinity. Perhaps take the average as the DV* you report to others. Perhaps take a set that represents a middle of the DV ranges. Perhaps just report the centroid and range of each DV.

The graph "Progress w.r.t. Iterations" reveals best and worst player numbers and their OF values. You should see the worst player number continually change with each iteration. The best player number should frequently change, but may hold constant for a few iterations if leap-overs do not find a better spot. If the best player number does not change for an extended number of iterations, it could indicate that there is a flat spot on the surface; alternately, it could indicate a problem with the formulation in a

section of code. In deterministic functions, the best OF value should asymptotically approach a final value, and the worst OF value should relax toward the best. However, for stochastic functions, both the best and worst should drop from the initial values to a noisy steady state. For either deterministic or stochastic functions, the best OF trace on the graph may take several drops as a local optima or a gently sloping region is found, then left for better locations.

If the best and worst OF values always approach their final values in a first-order manner (one exponential-like drop of OF w.r.t. iteration), this indicates a well-behaved function; and perhaps gradient based optimizers will find the optimum faster. However, if the OF values show relatively flat periods and sequential drops during the path to the optimum surface, this indicates saddle-like or relatively flat regions in the OF response.

The CDF of all OF values will be in the graph so labeled. For deterministic functions: Desirably, the best OF will be precisely and repeatedly discovered. Then the CDF(OF) curve will make a vertical jump at the global OF. However, if there is only one optimum, then the CDF(OF) curve will represent the variation in OF values in the vicinity of the optimum due to the tolerance specified by the convergence test. Also, if numerical discretization creates discontinuities and local pockets in the vicinity of the optimum, then the CDF(OF) curve will reveal the dispersion of OF* values created by the numerical discretization, not by the application that the model is supposed to represent. Use this graph for diagnostics and estimating the p(global).

For stochastic functions: The CDF(OF) curve will not rise sharply, but will have an S-shape due to the trial-to-trial variation. Consider whether the curve is suggesting whether there is a single optimum, or several. Use the data in the best optimum to represent the global. Consider how the uncertainty in DV* and OF* values represents the anticipated situation. If the variation is both representative of what you expect to see in the application and acceptable for making decisions, report a characteristic OF* and DV* value and/or a range of DV*s and associated OF* values from that global set. If unacceptable, reconsider the uncertainties on the givens and the model coefficients and obtain better values as appropriate.

Cell(1,12) and Cell(2,12) show the high and low player OF values from the initialization. This is one indication of how steep the function is. Close values that are also close to the converged optimum indicate gentle gradients, which suggest that it is easy to find the global. By contrast, large differences between initialized high and low OF values and the converged OF value indicate steep contours, and possibly an application with steep valleys or other features making it difficult to find the optimum.

Worksheet “DataRESULT” also shows the converged values, but these are sorted by OF* value to reveal the best-of-N (at the top of the list) and to construct the CDF(OF*) curve. Unlike the Main worksheet, if the optimizer is stopped due to excessive iterations, the ending results are shown in DataRESULT.

In worksheet “DataRESULT” above Column 11, which reports the iteration results from each trial, Rows 2 and 5 report the maximum and average number of iterations of up to 500 trials. If the maximum matches the maximum permissible iteration from “Main” cell(7,4) then you might not want to use that data. Above Column 12, which reports the NOFE results from each trial, Rows 2 and 5 report the standard deviation and average of up to 500 trials. This is useful in quantifying a key efficiency measure in evaluating optimizer choices.

Also in worksheet “DataRESULT” above the columns that report the OF* and DV* results from each trial, Rows 2 and 5 report the standard deviation and average of the best 10 trials. This is useful in quantifying

results of stochastic functions. No one particular solution is the truth about the optimum. I think the best 10 (or so) from 100 (or so) trials can be used to indicate a reasonable proximity of the optimum. For stochastic functions consider reporting the average and a 95% probable range of 2-sigma on the OF* and each DV*. If the function is deterministic then, ideally, the best of all trials represents the true global optimum; however, there is imperfection in reporting the optimum due to the convergence tolerance. If the best 10 of all N trials each represent the global, the standard deviation of the best few should reveal the impact of the convergence criterion threshold.

Worksheet "Progress" records the values for the "Progress w.r.t. Iterations" graph.

Worksheets "Doodle_1" and "Doodle_2" are for you to use.

Disclaimer

Optimization is not the truth about Nature. Optimization is a numerical procedure that characterizes your embodiment of the mathematical model you use to represent Nature. If you use results from optimization to make decisions in the real world (decisions that can impact the happiness and welfare of yourself and others), then you need to be sure that your choices (of the model, the code that represents the model, the constraints, the convergence criterion and threshold, the method to find the global, the method to report uncertainty of results, etc.) are all representative of the real situation. The mental concept, especially its first embodiment, is not the reality!

The user accepts all responsibility for the use of this program and the use of results from this program. The code is wholly visible for your benefit. To the best of my knowledge it provides correct results. But, I have not tested every possible situation, nor can I accept accountability for the user's code changes, the user's optimizer or convergence choices, the user's representation of the function, Excel/VBA functions, etc.