10-20-2016

Differential Evolution: A Tool for Global Optimization

Andrey D. Ukhov
Cornell University School of Hotel Administration, au53@cornell.edu

Differential Evolution: A Tool for Global Optimization

Abstract

This report describes a tool for global optimization that implements the Differential Evolution optimization algorithm as a new Excel add-in. The tool takes a step beyond Excel's Solver add-in, because Solver often returns a local minimum, that is, a minimum that is less than or equal to nearby points, while Differential Evolution solves for the global minimum, which includes all feasible points. Despite complex underlying mathematics, the tool is relatively easy to use, and can be applied to practical optimization problems, such as establishing pricing and awards in a hotel loyalty program. The report demonstrates an example of how to develop an optimum approach to that problem.

Keywords

hospitality firms, optimization, customer loyalty programs, evolution strategies, statistical estimation, customer portfolio management

Disciplines

Hospitality Administration and Management

Comments

CENTER FOR HOSPITALITY RESEARCH

Differential Evolution

A Tool for Global Optimization

by Andrey D. Ukhov

EXECUTIVE SUMMARY

This report describes a tool for global optimization that implements the Differential Evolution optimization algorithm as a new Excel add-in. The tool takes a step beyond Excel's Solver add-in, because Solver often returns a local minimum, that is, a minimum that is less than or equal to nearby points, while Differential Evolution solves for the global minimum, which includes all feasible points. Despite complex underlying mathematics, the tool is relatively easy to use, and can be applied to practical optimization problems, such as establishing pricing and awards in a hotel loyalty program. The report demonstrates an example of how to develop an optimum approach to that problem.

ABOUT THE AUTHORS

Andrey D. Ukhov is an assistant professor of finance in the School of Hotel Administration. Ukhov is an expert on a wide range of investments, including preferred stocks, warrants, derivative securities, and convertibles. His research papers have been published in *Management Science, Journal of Financial and Quantitative Analysis, Review of Finance, Quantitative Finance, Economic History Review,* and *Journal of Real Estate Research.*

Ukhov is a frequent presenter and discussant at international finance conferences. Prior to joining the School of Hotel Administration, he taught both undergraduate and graduate finance courses at Kelley School of Business; Indiana University; and Kellogg School of Management, Northwestern University. He has received numerous teaching awards at Cornell, Indiana, and Northwestern.

Prior to becoming an economist, Professor Ukhov studied applied mathematics, mathematical physics, and computer science. He received two U.S. patents for technology inventions. Ukhov earned a bachelor's degree in economics with distinction, an MA, MPhil, and PhD in financial economics, all from Yale University

Acknowledgments. The author would like to thank Dmitry Zeliakh for numerous productive discussions, computational assistance, and extensive testing of the tool; Brett Massimino, Nagesh Gavirneni, and Gary Thompson for helpful conversations on global optimization and evolutionary algorithms, Adrian Tang for research assistance, and Dmitry Zeliakh and Adrian Tang for extensive testing of the tool prior to its release. I also would like to thank former CHR director Michael Sturman and two anonymous referees for helpful comments and suggestions.

Differential Evolution:

A Tool for Global Optimization

by Andrey D. Ukhov

Hospitality executives have always faced optimization problems of one type or another, including revenue management pricing, establishing room blocks for groups, and making the most of a corporate loyalty program. While the availability of big data can contribute to a manager's decision-making process, many managers need simple but robust global optimization algorithms for solving the problems that are fundamental to their daily work. In short, big data comes with new opportunities for management, but at the same time it requires new types of skills, knowledge, and tools.[1]

[1] For a discussion of big data issues in the hospitality industry see, for example: J. Bruce Tracey, "Hospitality HR and Big Data: Highlights from the 2015 Roundtable," *Cornell Labor and Employment Law Report*, August 2015, Vol. 15, No. 12 (Cornell Institute for Hospitality Labor and Employment Relations).

This report demonstrates the use of a new tool, known as Differential Evolution, that can help address optimization issues. Although the underlying mathematics is complicated, solving a complex optimization problem should not itself be difficult, provided the individual has a solid understanding of the problem. For example, a hotel chain marketing executive may have an expert knowledge of the principles of reward programs, such as how and at what properties loyalty points are accumulated and when and at what properties loyalty points tend to be used. That person should not also have to be an expert in optimization theory just to improve the program's design. In addition to being easy to use, a global optimization algorithm should also be powerful enough to reliably converge to the true optimum. Furthermore, the computer time spent searching for a solution should not be excessive. Thus, a genuinely useful global optimization method should be simple to implement, easy to use, reliable, and fast.

Differential Evolution is such a method. Since its inception in 1995, Differential Evolution has earned a reputation of an effective global optimizer.[2] It has been used to solve difficult problems in mathematics,[3] materials science (where it was used to study silicon-hydrogen, Si-H, clusters), robotics, and real-world engineering problems. The Differential Evolution method has also been used to determine hypocenters of earthquakes, in three-dimensional medical image processing, and in the design of digital circuits. These are challenging problems, and differential evolution has proven to be a powerful and useful global optimization tool.[4]

This report describes how to implement the Differential Evolution algorithm as an add-in tool for Microsoft Excel. This tool is designed to be as easy to use as another optimizing add-in tool, Solver, although Differential Evolution has a broader application. The goal of this report is to make this powerful global optimization algorithm available to Excel users.

This report begins with a brief discussion of Solver's limitations, followed by a concise overview of the Differential Evolution algorithm. If you're interested in applying the tool, feel free to skip these two sections and proceed directly to the descriptions of how to install and use the tool. I illustrate how to use the tool by giving an example relating to loyalty programs that includes screenshots, complete with suggestions for other uses for Differential Evolution.

I offer supporting information in the appendices. This includes description of four standard test functions that are used to test capabilities and performance of global optimization algorithms, and a detailed description and further details of the Differential Evolution algorithm. I also provide installation instructions for installing Differential Evolution on Excel 2007 and on later versions of Excel (for example, Excel 2013). Finally, I discuss how Differential Evolution could apply to the optimization of a customer loyalty program, and I conclude with answers to frequently asked questions, including instructions on how to use the tool to solve integer programming problems and how to use the tool to solve constrained optimization problems.

Why Another Tool?

But let's start with the basic question of why one might need an additional optimizer tool. Microsoft has included the Solver optimizer as an add-in designed to solve minimization and maximization problems. As part of Excel, Solver is user friendly, but its ability to find a global optimum of a difficult optimization problem is limited.[5] Instead, Solver often finds only a local minimum (or maximum), not the global one. This may be sufficient in some cases, but in many cases the business problem calls for the global solution.

I illustrate this issue in Appendix A, which describes four standard test functions, which represent difficult-to-solve minimization problems. These functions, which are used to test and evaluate optimization methods, present a challenge for Solver in the form of many local minima. (The Salomon function is particularly challenging, for instance). In contrast, when given enough ability to explore the function (that is, a sufficiently large population and number of generations), Differential Evolution is able to reliably and consistently find the optimum for each of these challenging optimization problems.

While Solver is convenient and may be sufficient in most instances, today's managers are better off having several global optimization tools at their disposal. Differential Evolution has been shown to be indispensable in many engineering and natural science applications. Its applications in management have been limited until now by the lack of availability of a convenient tool, such as an Excel add-in.

---

[2] Rainer Storn and Kevin V. Price (1995). Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI. See also: Rainer Storn and Kenneth Price, "Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, December 1997

[3] For example, DE has no difficulty determining the coefficients of the 33-dimensional Chebyshev polynomial, considered by many researchers to be a difficult task for a general-purpose optimizer.

[4] For additional details on applications of Differential Evolution to various problems, see: Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Springer-Verlag, 2005, Chapter 7.

[5] Solver has two methods for searching for a global optimum. The first method is the Multistart option where the user instructs Solver to conduct search starting from multiple initial locations (the locations are picked at random by Solver). Solver has a minimum of 10 and maximum of 200 starting points when using Multistart. For challenging problems 200 starting points may not be enough to explore the function. The second option is Solver's implementation of an Evolutionary Algorithm, a method different from Differential Evolution that is described in this report

A Brief Overview of the Differential Evolution Algorithm

Differential Evolution was developed to be a reliable and versatile global optimizer for functions of many variables that is also easy to use. The first written publication on Differential Evolution appeared as a technical report in 1995.[6] Since then, Differential Evolution has proven itself in competitions, such as IEEE's International Contest on Evolutionary Optimization (ICEO) in 1996 and 1997 and in the real world on a broad variety of applications. For an interested reader, the algorithm is described in detail in Appendix B. Only a brief overview is provided here.

Like nearly all evolutionary algorithms (EAs), Differential Evolution is a population-based optimizer. It starts by sampling the objective function at multiple, randomly chosen initial points. The objective function f(x) is a function of D variables, $x = (x_0, x_1, \ldots, x_{D-1})$, $f(x) = f(x_0, x_1, \ldots, x_{D-1})$. The initial population has Np vectors x. The variable Np represents the number of points (vectors) in the population (it is not a product of N and p). Each vector x_i is indexed with a number i from 0 to Np-1. Each vector represents an initial trial solution and the objective function is evaluated at each vector. The domain from which the Np initial vectors are chosen is defined by the preset parameter bounds.

Like other population-based methods, Differential Evolution generates new points that are perturbations of existing points. Differential Evolution perturbs vectors with the scaled difference of two randomly selected population vectors. To produce a (new) trial vector, Differential Evolution adds the scaled, random vector difference to a third randomly selected population vector. In the selection stage, the trial vector competes against the population vector of the same index. The vector with the better objective function value (e.g., lower for minimization) is marked as a member of the next generation. The procedure repeats until all Np population vectors have competed against a randomly generated trial vector. Once the last trial vector has been tested, the survivors of the Np pairwise competitions become parents for the next generation in the evolutionary cycle.

It should be noted that in the operations research and numerical methods literature there is insight into how and why Differential Evolution works, including conditions for a convergence proof. There is also evidence on favorable performance comparisons with other global optimization algorithms.

How to Use the Excel Tool: Installation

Installation consists of 4 easy steps.

Step 1. Download the file that contains the tool. Filename: DEv1.1_AddIn.xlsm

Step 2. Open the file in Excel.

Step 3. Use "Save As" to save the file as Excel Add-In in the Add-In directory.

Step 4. Install the Add-In.

The installation needs to be done only once.

As described in detail in the appendices, the steps are the same for various Excel versions, but the screen shots are different because of the minor differences in Excel user interface.

How to Use the Excel Tool: Running the Tool

As I said, the tool is for performing global optimization, which means finding the minimum (or maximum) of a function of several variables. Again, while Solver is suitable for this purpose in many cases, it handles local optimization better than it handles global optimization.

About Evolution Algorithms

Evolution strategies (ESs) were developed by Rechenberg (1973) and Schwefel (1994), while genetic algorithms (GAs) are attributed to Holland (1962) and Goldberg (1989). Both approaches attempt to evolve better solutions through recombination, mutation, and survival of the fittest. Because they mimic Darwinian evolution, ESs, GAs, DE and their ilk are often collectively referred to as evolutionary algorithms, or EAs.

Like other multi-start algorithms, an ES samples the objective function landscape at many different points, but unlike the multi-start approach where each base point evolves in isolation, points in an ES population influence one another through recombination. Beginning with a population of parent vectors, the ES creates a child population of vectors by recombining randomly chosen parent vectors. Recombination can be discrete (some parameters are from one parent, some are from the other parent) or intermediate (e.g., averaging the parameters of both parents). Once parents have been recombined, each of their children is "mutated" by the addition of a random deviate, that is typically a zero mean, normally distributed random variable. After mutating and evaluating all children, the ES selects the best children to become the next generation's parents. Alternatively, the ES populates the next generation with the best vectors from the combined parent and child populations.

See: Rechenberg I (1973), *Evolutionsstrategie*. Frommann-Holzboog, Stuttgart. Schwefel H-P (1994); *Evolution and optimum seeking*. Wiley, New York. Holland JH (1962); Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery* 3:297-314; and Goldberg DE (1989), *Genetic algorithms in search optimization and machine learning*. Addison-Wesley, Reading, MA

[6] Storn and Price, *op.cit.*

EXHIBIT 1

Summary of control parameters, Differential Evolution

Option	What this option does	Suggested value	Why higher values are useful	Why lower values are useful
Population size	A positive integer number between 4 and 25,000. Tells the algorithm how many trial points to maintain in each generation.	10 • D, where D is the number of variables. For functions with many local optima (multi-modal functions), the size may need to be 100 • D or larger. Population size may need to be larger for higher Cr values.	Larger population size allows to explore the function better and gives more chances of finding the global optimum. For higher values of Cr, larger population sizes are better.	Faster computation time, especially if the objective function is difficult to evaluate (takes a long time for each evaluation).
Differentiation constant *(F)*	This controls how radically DE "mutates" your decisions from one iteration to the next.	0.9 is recommended; For higher values of *Cr* (say, *Cr* = 0.9), larger values of *F* (*F* > 0.8) work better.	Larger values work better with larger values of the crossover constant.	Values in the range (-1, 0) should only be used to localize the search.
Crossover constant *(Cr)*	A number between 0 and 1. A probability that a parameter will be inherited from a mutant to the next generation.	0.5 in general; In the range 0 to 0.2 for simple functions; Close to 1 for functions with mutual dependence of parameters (variables).	High values provide extra diversity to the pool of possible trial solutions. In the general case of parameter-dependent functions, the value should be close to 1.	Low values, between 0 and 0.2 work well for decomposable objective functions.
Maximum number of iterations	A positive integer between 1 and 20 million. This limits the number of generations the algorithm is working through. Puts a limit on total computing time.	A minimum of 1,000 is recommended. For difficult objective functions with many local optima, tens of thousands of iterations are recommended (50,000 or more).	Gives the algorithm an adequate opportunity for the evolutionary dynamics to kick in and to find the global optimum.	Faster computation time. A user is OK with a "good enough" solution and does not need the global solution.
Low and high bounds	Cells in the worksheet that give low (and high) bounds for each of the variables.	The values are dictated by the problem being solved. For example, for positive prices the low bound is 0.0.	Not applicable.	Not applicable.

I'll describe how to proceed here and then show how it works.

(1) *Target cell*. To begin, the user should formulate the problem and set up the spreadsheet accordingly. There should be one cell that contains the function that needs to be optimized (the function whose minimum or maximum the user would like to find). This cell must contain a formula (that is, it cannot be a numerical value, such as 77). This is the Target Cell.

(2) *Changing cells*. There should be a group of cells (a region) that contains the variables that the function depends on. These are the variables of the problem and these are the cells that will be varying in value during the search for the optimum. These cells should be in one contiguous region; several disjointed regions are not accepted.

(3) *Low bound constraints cells*. There should be a group of cells (a region) that contains low bounds for the variables. These cells also should be one contiguous region. There must be as many cells in this region as there are variables (that is, as there are changing cells), and these cells should not be empty. They should have numerical values or formulas that evaluate to numerical values.

(4) *Upper bound constraints cells*. There should be a group of cells (a region) that contains upper bounds for the variables. As with the others, these cells should be one contiguous region, and

there must be as many cells in this region as there are changing cells). Again, these cells also should not be empty, and like the low-bound constraint cells, they should have numerical values or formulas that evaluate to numerical values.

(5) To be clear, the regions (cells) for the Target Cell, the Changing Cells, Low Bound Constraints Cells, and High Bound Constraints Cells should occupy four distinct areas on the worksheet and must not be overlapping.

(6) Each set of cells must have the same number of cells. That is, the Changing Cells, Low Bound Constraints Cells, and High Bound Constraints Cells must all have the same number of cells.

(7) Although this should go without saying, all selected ranges (the Target Cell, Changing Cells, Low Bound Constraints Cells, and Upper Bound Constraints Cells) must be located on the same Active Worksheet.

(8) During the computation, user can use ESC or Ctrl-Break key to interrupt calculations. A dialog box will appear that displays the current iteration, the total computing time until that moment, the expected time until completion, and the optimal (minimum or maximum) value found so far. The user is given an option to continue calculations (the computations will proceed without losing any information) or to stop computations. If the calculations are stopped, then the best solution so far is retained in the worksheet.

(9) The Excel status bar (lower left corner of the Excel window) will provide progress updates at 1%, 5%, 10%, and then 5% increments to 90%, 95%, and 99% of completion.

(10) If not interrupted, computations will stop after all iterations are complete. The number of iterations is set by the user as the maximum number of generations (iterations), given in the user interface before starting calculations. If calculations are interrupted at any time, one can examine the intermediate result and discontinue further calculations if that value is deemed satisfactory.

Parameter Values

You also select values of the parameters for the algorithm. Four parameters determine how the algorithm searches for the optimum: **(1)** population size, which is the number of trial solution points in each generation; **(2)** differentiation constant, which determines how radically the algorithm "mutates" candidate solutions from one generation (iteration) to the next; **(3)** crossover constant, a number that controls how the algorithm inherits values from a mutant to the next generation; and **(4)** maximum number of generations (iterations), which limits the number of generations the algorithm is working through. Let's look at each of these in greater detail, with control parameters summarized in Exhibit 1.

(1) *Population size (Np)* is the number of points that make up the current generation. Population size is an important factor. It should not be too small to avoid stagnation of the algorithm, and to provide opportunity for sufficient exploration of the surface of the objective function. The increase in *Np* results in the increase in the number of function evaluations, increasing computing time. *Np* is related to the constant of differentiation, *F*. Intuitively, a large *Np* requires a small *F*. That is, the larger the size of a population, the more densely the individuals fill the search space, so less amplitude in their movement is needed. Typically, the recommended size of the population is ten times the number of variables: $Np = 10 \cdot D$. The minimum population size is 4 and the maximum population size in the current implementation is 25,000.

(2) *Differentiation constant (F)*. This number controls how radically Differential Evolution "mutates" candidate solutions from one iteration to the next. At the first attempts at the problem, the user should keep the default value of 0.9. The default value should work for the majority of optimization problems. If the user experiments with differentiation constant and with the crossover constant, it is useful to keep in mind that smaller values of *F*, say $F = 0.5$ work better with smaller values of the crossover constant *Cr*, say with $Cr = 0.2$. As *Cr* increases, *F* usually needs to increase as well. For example, if $Cr = 0.9$, then $F \geq 0.8$ will be more likely to give regular convergence than will $F = 0.5$.

Exhibit 2

Technical summary of the implementation.

Population size: Integer between 4 and 25,000
Differentiation constant: [-1, 0) or (0, 2.5]
Constant of crossover: [0, 1]
Maximum number of generations: Integer between 1 and 20,000,000 (20 million)

Strategy: DE/rand/1/bin (Classic Differential Evolution).
Crossover: DE Uniform crossover (one randomly chosen parameter from the mutant is always chosen for the trial vector, so the trial vector will not replicate the target; the remaining parameters are crossed over one at a time, with probability Cr of the parameter coming from the mutant).
Selection: Immediate selection (one array is maintained for the population).
Note on Performance: During calculations when evaluating the objective function, the tool calculates all open workbooks. To improve performance and to speed up calculations, keep open only the workbooks that are necessary to compute the objective function and close all other workbooks.

Exhibit 3

Possible security warning with Differential Evolution add-in

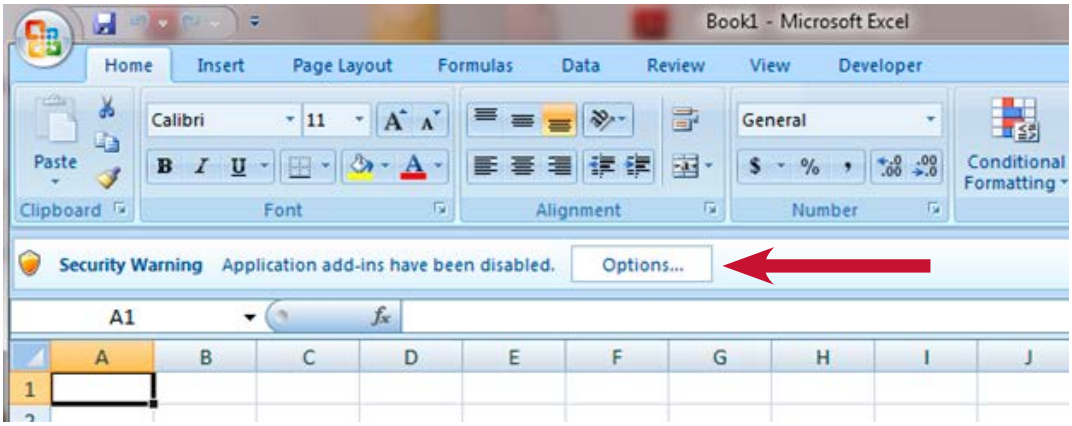


Exhibit 4

Enabling content in the event of a security warning with Differential Evolution add-in

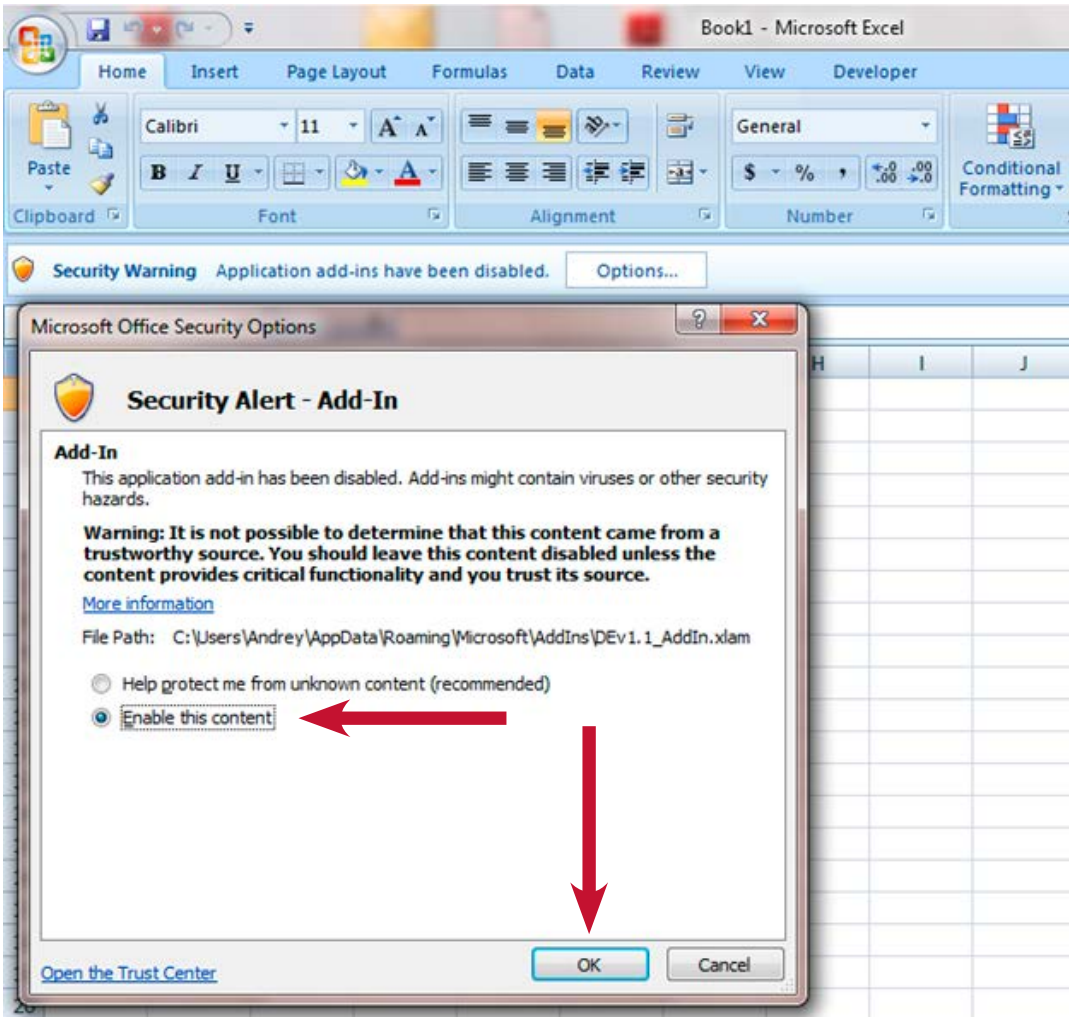
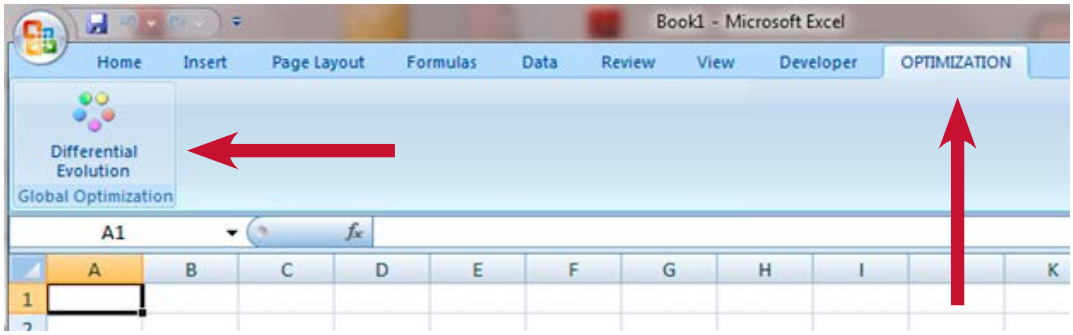


EXHIBIT 5

Optimization menu with Differential Evolution add-in



(3) *Crossover constant (Cr)*. Parameter *Cr* can be thought of as a mutation rate, or the (approximate) probability that a parameter will be inherited from a mutant into the next generation. A low *Cr* corresponds to a low mutation rate. Previous research has discovered that for some functions low values of *Cr* tend to be the most effective. For other functions, high values of *Cr* were effective. In extensive testing by prior researchers, different functions were solved with either $0 \leq Cr \leq 0.2$ or $0.9 \leq Cr \leq 1$. That said, in the first attempts at the problem, the user should keep the default value of 0.5. If the user decides to experiment with different parameter values to explore their problem, the following information may be useful. A low value of the crossover constant, $Cr = 0.2$ may work well for separable functions or those that exhibit limited parameter dependence (low degree of mutual dependence among the function variables). The values $Cr = 0.9$ or $Cr = 0.95$ are appropriate for functions that have parameter dependence (mutual dependence between the variables). Population size *Np* may have to be increased as *Cr* increases. Setting $Np = 5 D$ *Cr* is usually a low-end default setting, but for highly multi-modal, parameter-dependent functions, *Np* may need to be $10 \cdot D$ or higher.

(4) *Maximum number of generations (iterations)*. This is an integer that determines how many generations will be used. At the minimum, 1,000 generations should be used, but 5,000 is better for the first attempt at a problem. If the function to be optimized (the objective function) takes a long time to evaluate, start with a smaller number of generations (e.g., 500) to limit computing time. However, for the productive properties of the algorithm to take effect, it is important to have a large number of generations. For difficult objective functions with many local optima, tens of thousands of iterations are recommended.

Detailed Example

The tool has many potential applications. Whenever a business problem can be formulated as a maximization or a minimization problem and the objective function is non-linear, a robust tool is needed to find a global maximum or minimum.

First, the tool needs to be installed as an Excel Add-In (as described above). After you install the tool as an add-in, whenever you start Excel, depending on your security settings within Excel, you may see a security warning, as shown in Exhibit 3. If you see a security warning, click Options next to the warning and in the dialog box that shows select, Enable this content and click OK. This will allow the Visual Basic code for the Differential Evolution optimizer to run.

After the Add-In is enabled, you will see a new menu, Optimization. Clicking on the Optimization menu will show a ribbon with Differential Evolution button.

Now set up the business problem. As an example, you can use the file "Example_DEv1.1" that has a model illustrating application of the tool. Download the file and open it (remember to enable Add-In content, as described above).[7]

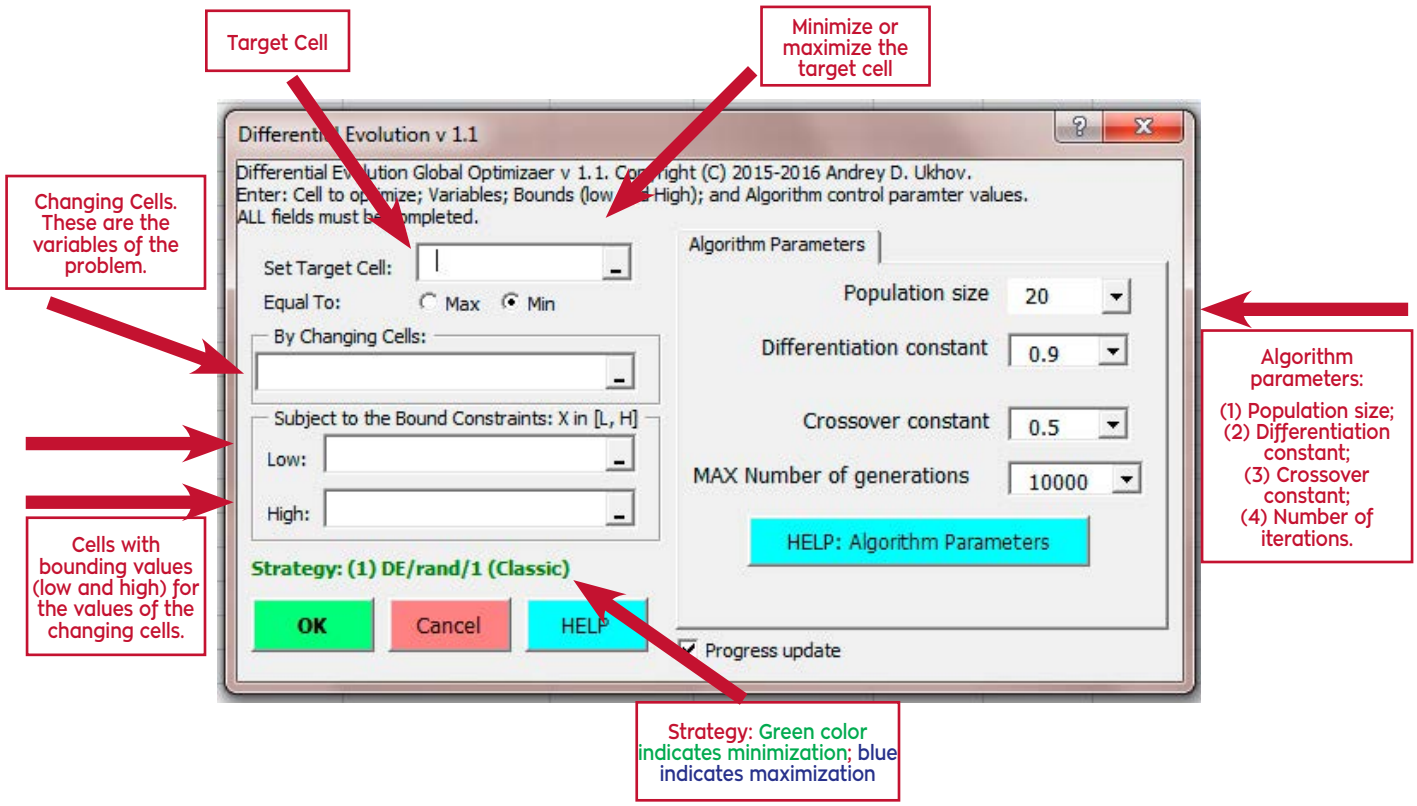
This example illustrates how to use the Differential Evolution global optimization add-in to find an optimal value for a formula in one cell (called the "target cell"). The nature of the example will be familiar to hotel operators. The add-in works with a group of cells that are related, either directly or indirectly, to the formula in the target cell. The Add-In adjusts the values in the changing cells that you specify to produce the specified result (a minimum or a maximum) from the target cell formula. You also specify the lower and upper bounds on the values that the adjustable cells may take, as I described above.

In this example, which resembles a hotel operation, a company operates 5 business locations. Each location has a fixed cost, plus variable cost (per unit of product sold). Fixed and variable costs are different for the five locations, and prices likewise are different. The demand for the product depends on price (demand decreases as price increases) and on advertising spending

[7] This is a relatively simple example that does not require a sophisticated optimization algorithm to be solved. The purpose of the example is to show how to use the tool, not to showcase the DE algorithm

EXHIBIT 6

User interface



(higher spending on advertisements increases demand). Total expenses in each location are calculated as follows:

Total Expenses = Fixed Cost + (Variable Cost * Demand) + Advertising

Total revenues in each location are essentially a gross operating income value:

Total Revenues = Price * Demand

Profits are revenues minus expenses (essentially, EBITDA), and total profits for the firm equal the sum of profits at all 5 locations.

In this example, the level of advertising at each location affects the number of units sold, indirectly determining the amount of sales revenues, the associated expenses, profit at each location, and total profit. The optimizer can change the allocations to advertisement, each changing cell remaining within the bounds set by the low and high bounds, until the value for the total profit reaches the maximum. The values in the adjustable cells are used to calculate the profit for each location, adding up to the total profit.

To run the Differential Evolution add-in select the Optimization menu and click the "Differential Evolution" button on the Excel ribbon. A user interface of the add-in will appear, and you will need to specify the Target Cell, the area for the Changing Cells, the areas for the Low Bound cells and the High (Upper) Bound cells. You also specify that this is a Maximization problem by selecting the button "Max."

This is where you specify the parameters that control the algorithm: **(1)** Population size; **(2)** Differentiation constant; **(3)** Crossover constant; and **(4)** Number of iterations. For the first run, you may leave the default values for the algorithm parameters. For problems with more variables (more changing cells), increase population size and the number of iterations. For complicated problems, you should experiment with the other control parameters.

Here's a description of the screen shot with the user interface shown in Exhibit 6.

(1) Select the Target Cell. **(2)** Specify that the target is to be maximized (select "Max" button and watch the Strategy field change color from green indicating minimization to blue indicating maximization). **(3)** Select the range that contains changing cells; the values in these cells will be changing as the optimizer is searching for the maximum. **(4)** Select the range for cells that contain low bound values. **(5)** Select the range for cells that contain high bound values.

For the first attempt at the problem you may keep the default values for algorithm parameters. Click OK to start calculations.

Exhibit 7

User interface with completed fields

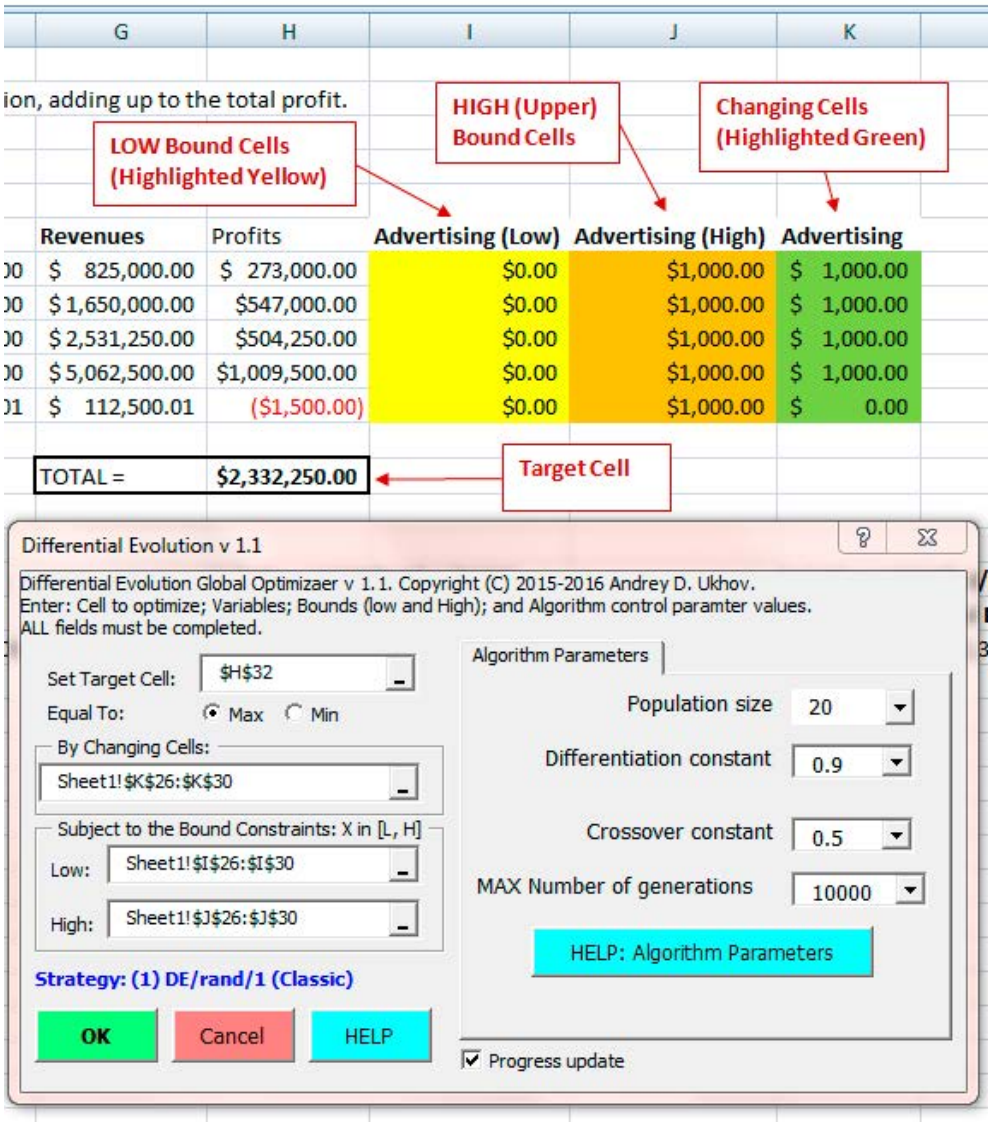
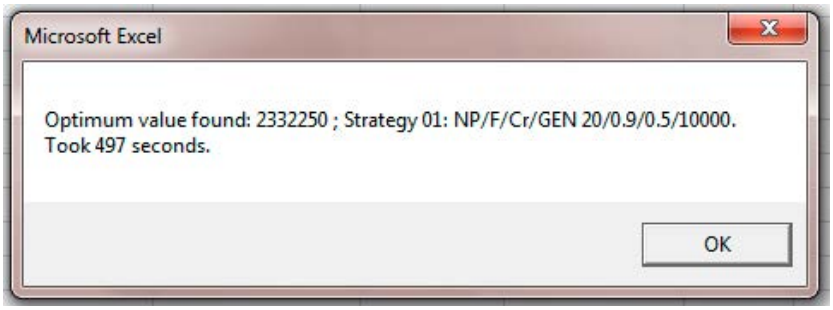


Exhibit 8

Completed calculation notice



Shown in Exhibit 7 is the user interface with completed fields for this example. When the computations are complete the message shown in Exhibit 8 will be displayed:

Click OK and the optimal values will be written into the changing cells. Once again, you can interrupt the calculations and resume them as needed.

You can experiment with the add-in by changing algorithm parameters. The Excel file "Example_DEv1.1" has a second example, where the firm can choose product prices and advertising expenses at the 5 locations (the total of 10 variables) to maximize profits. The description of this example and instructions on running the add-in for the second example are given in the Excel file "Example_DEv1.1". The tool has many applications. Enjoy!

Discussion and Conclusion

As I said at the outset, one application for a global optimization tool might involve optimization of a customer loyalty program. For a large organization, such as a hotel chain, the propensity to earn and redeem points may be different for different hotels and with respect to different services (rooms vs. other services). Given the data on the propensity to earn a redeem points, a manager may wish to optimize a customer loyalty program by searching for point award rules and point redemption rules that maximize profitability. Appendix E contains additional discussion of this potential application.

Another application may be in customer portfolio management. This idea was developed in several research papers by Michael Johnson and Fred Selnes.[8] Customer portfolio management is a process of creating value across a company's customer relationships with an emphasis on balancing closer customer relationships with weaker ones. The idea is to focus on the management of an entire portfolio of customers who are at different relationship stages in a dynamic framework. One important insight from this research is that a key to the creation of value through closer relationships (e.g., repeat customers and loyal customers) lies in bringing weaker relationships into a portfolio in the first place (e.g., first time customers, churn customers). Customer portfolio management is directly related to optimal resource allocation by a firm, as the firm decides how it should invest in an entire portfolio of relationships. From a customer portfolio standpoint, the fundamental resource-allocation trade-off is as follows. Should resources be allocated to create simple transactions and growth as a means of leveraging economies of scale, or should resources be allocated to build relationships through cooperation to increase the lifetime value of customers? Thus, customer portfolio management yields a global optimization problem, that takes into account costs and benefits of different customer categories, as well as transition of customers from one category to another.

Another application of global optimization is in statistical estimation. One of the methods widely used in estimation of statistical models is Maximum Likelihood Estimation. The idea is to write down a likelihood function that describes the probability of observing the data and to find values of the parameters (say, regression coefficients and variances of error terms) to maximize the likelihood function. For example, Maximum Likelihood Estimation can be applied when simultaneously estimating supply and demand equations in a given market. In another example, Maximum Likelihood Estimation can be used to estimate the impact of a new entrant (supply shock) on the performance of existing hotels in a market. In a large data set generated from a complex business environment, a likelihood function that captures interesting properties of the data can be complicated and may depend on a large number of parameters of interest. Because the goal is to find parameters that maximize the value of such a likelihood function, these statistical problems may present challenging global optimization problems.

Whatever the applications, the tool described in this reports gives a user a new Excel Add-In to help search for a global optimum. ■

---

[8] Michael D. Johnson and Fred Selnes, "Customer Portfolio Management: Toward a Dynamic Theory of Exchange Relationships," *Journal of Marketing*, 68, no. 2 (April 2004): 1-17, develops and simulates a customer portfolio lifetime value model. Michael D. Johnson and Fred Selnes, "Diversifying Your Customer Portfolio," *MIT Sloan Management Review*, 46, no. 3 (Spring 2005): 11-14, discusses the need for a disciplined approach to managing customers as a portfolio. And R. Dhar and R. Glazer, "Hedging Customers," *Harvard Business Review*, 81, no. 5 (May 2003): 86-92, emphasizes the need to hedge a customer portfolio by including different types of relationships.

Appendix A

Test Functions

This appendix contains four test functions. These are some of the functions that have been used in the numerical methods literature to test performance of optimization algorithms. All functions are in D dimensions. The plots are presented for the case of two variables, D=2. Each function definition includes a set of upper and lower initial parameter bounds. A function's parameter bounds apply to all of its parameters. Each test problem description also lists the minimum objective function value.*

Exhibit A1

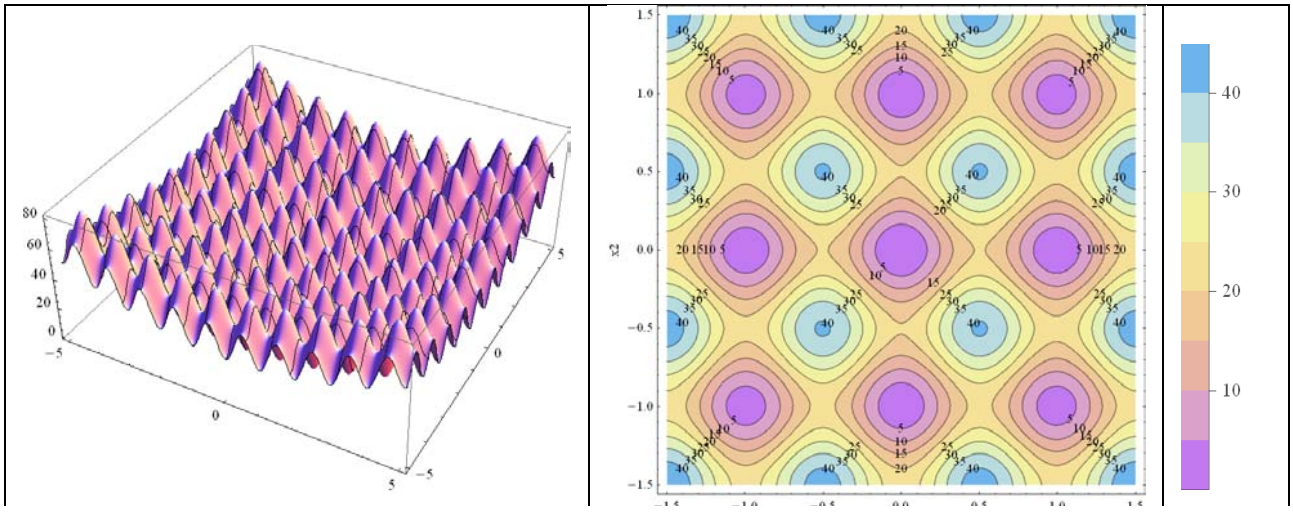
Rastrigin's function in two dimensions (left) and its contour plot

Rastrigin. Rastrigin's function has many local optima arrayed on the side of a larger bowl-shaped depression. This is an example of a highly multimodal search space. It has several hundred local optima in the interval of consideration. This function is separable as written and is easily solved by methods that can exploit decomposable functions. It is much harder to solve when rotated. Rastrigin's function is a generalization of a two-dimensional function to D dimensions. The function is symmetric about its solution. Optimizers that search the vicinity of the mean population vector will do well on this symmetric function because, like the local minima, the population will also be symmetrically distributed.

$$f(\boldsymbol{x}) = \sum_{j=0}^{D-1} \left(x_j^2 - 10 \cdot cos\left(2\pi \cdot x_j\right) + 10\right)$$

$$-5.12 \leq x_j \leq 5.12, j = 0,1, \cdots, D-1$$

Solution: $f(\boldsymbol{x}^*) = 0, x_j^* = 0.$



* These test problems are discussed in: Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer-Verlag, 2005. Test functions for evaluating optimization algorithms are also discussed in Z. Michalewicz and M. Shoenauer, 1996, "Evolutionary algorithms for constrained parameter optimization problems." Evolutionary Computation 4(1): 1-32 and in X. Yao and Y. Liu, 1997, "Fast evolution strategies.: In: P.J. Angeline, R.G. Reynolds, J. R. McDonnell, R. Eberhart (eds) Evolutionary Programming IV, Lecture notes in computer science 1213, Springer, Berlin, pp. 151-161

Exhibit A2

Salomon function (left) and its contour plots (right)

Salomon. The landscape of this parameter-dependent function resembles a pond with ripples. Because this function is symmetric, methods that search the vicinity of the population mean vector will likely perform well.

$$f(\boldsymbol{x}) = -cos(2\pi \cdot \|\boldsymbol{x}\|) + 0.1 \cdot \|\boldsymbol{x}\| + 1,$$

$$\|\boldsymbol{x}\| = \sqrt{\sum_{j=0}^{D-1} x_j^2}, \quad -100 \le x_j \le 100, \ \ j = 0,1,\cdots,D-1.$$

Solution: $f(\boldsymbol{x}^*) = 0, x_j^* = 0.$

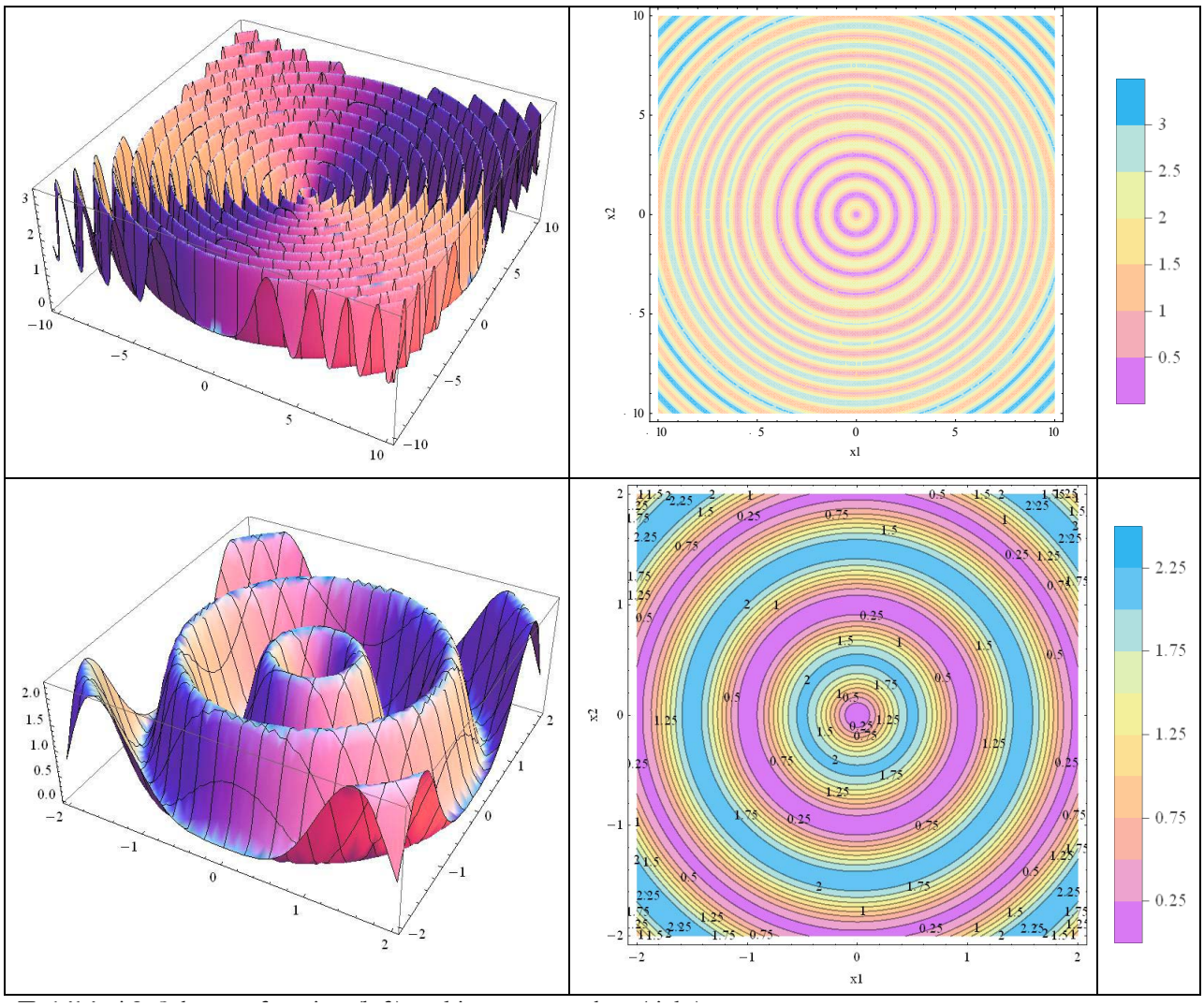


EXHIBIT A3

Schwefel function (left) and its contour plots (right)

Schwefel. This classic test function has a solution that lies on a coordinate system diagonal. In this version, the objective function is normalized by D so that f(x^*) is the same regardless of dimension. Success here can depend heavily on how bound constraints are handled. This function is separable.

$$f(\boldsymbol{x}) = -\frac{1}{D}\sum_{j=0}^{D-1} x_j \cdot sin\left(\sqrt{|x_j|}\right),$$

$$-500 \le x_j \le 500, \quad j = 0,1,\cdots,D-1$$

Solution: $f(\boldsymbol{x}^*) = -418.983, x_j^* = 420.968746.$

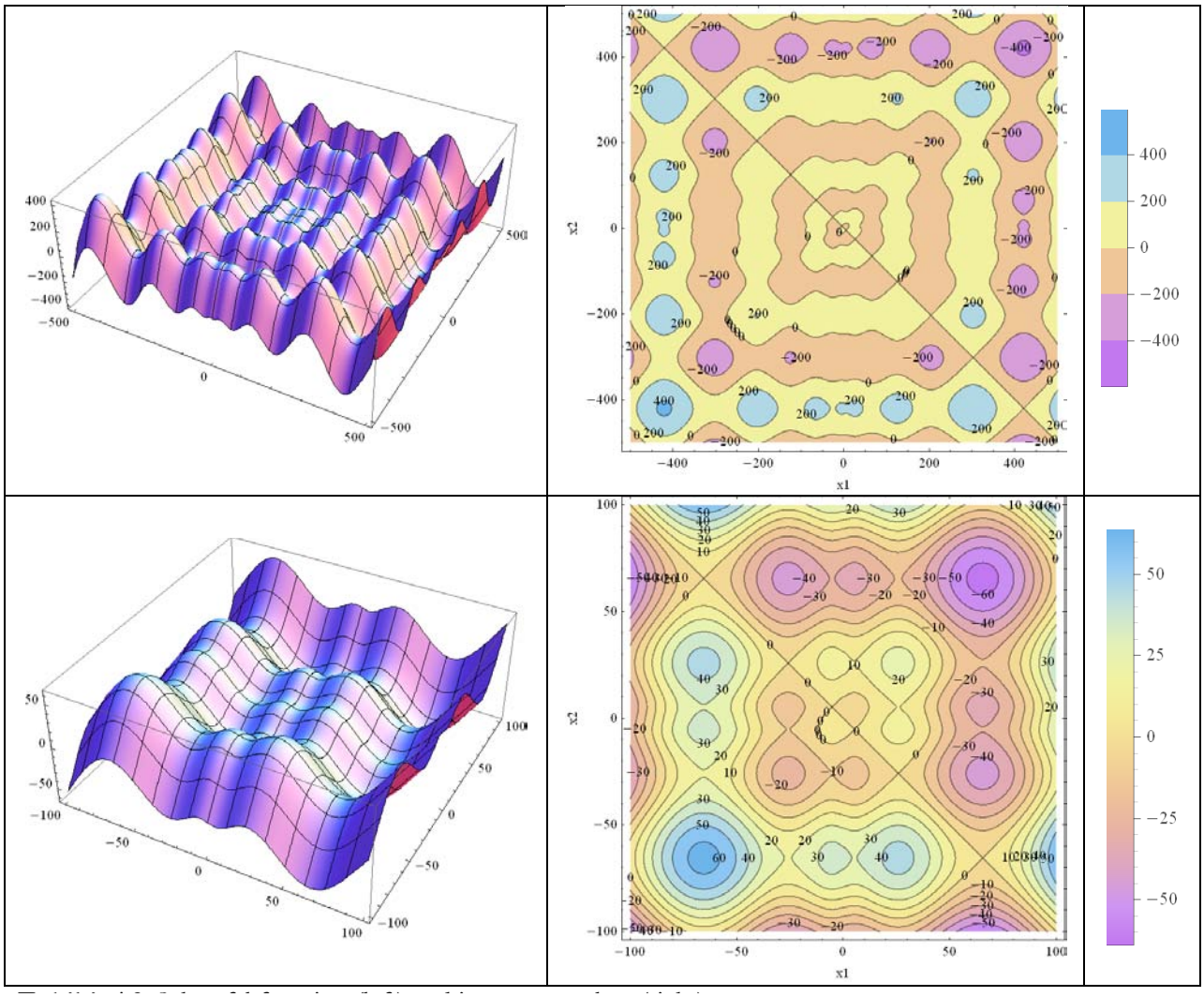


EXHIBIT A4

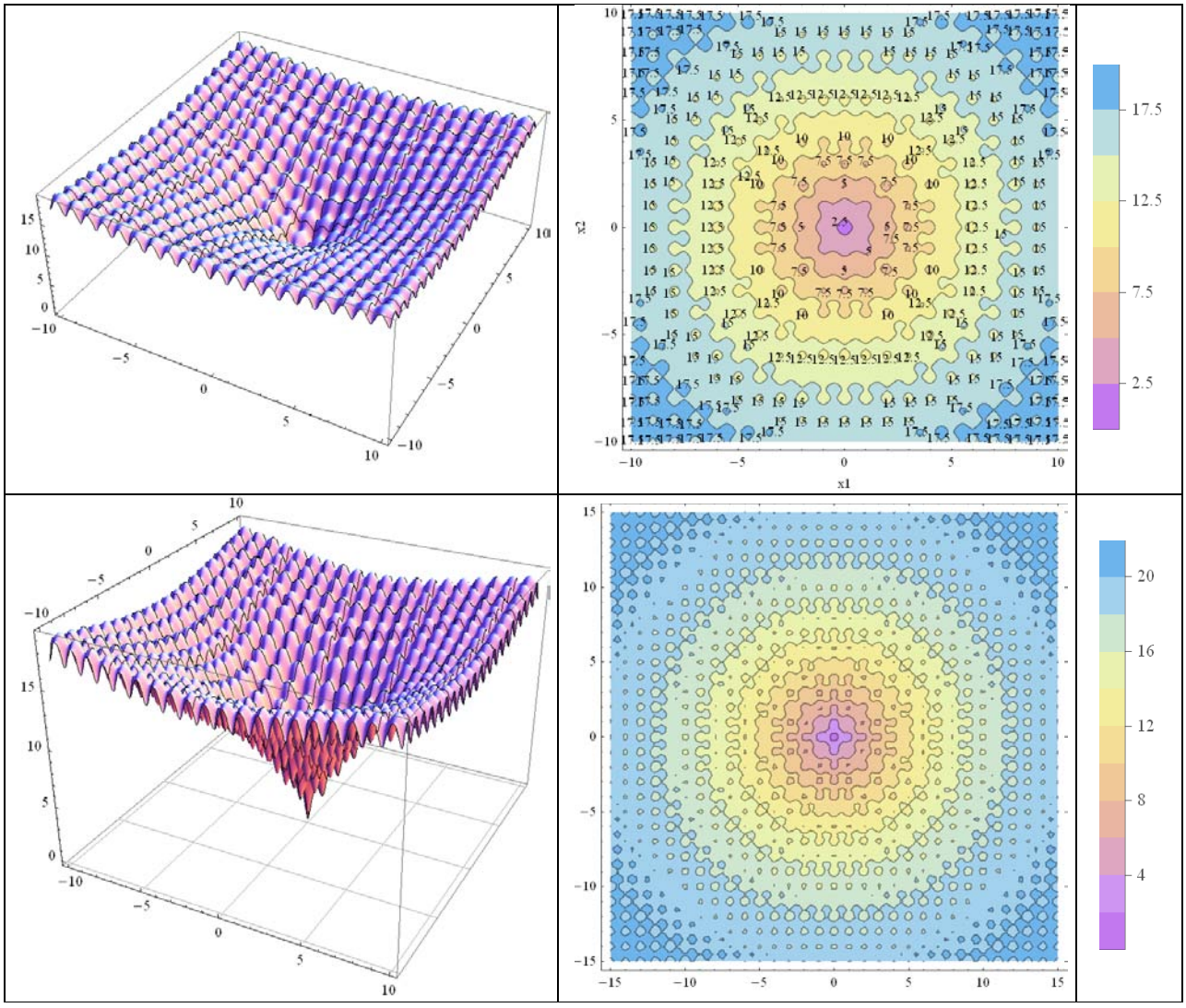
Ackley function (left) and its contour plots (right)

Ackley's function. Ackley's function is one of the most commonly cited multi-modal test functions. At high dimensions (D≥30), care must be taken with computer code to ensure precise result. For example, the constant e=2.71828... is best implemented as *e = Exp*[1].

$$f(\boldsymbol{x}) = -20 \cdot Exp\left(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{j=0}^{D-1} x_j^2}\right) - Exp\left(\frac{1}{D} \cdot \sum_{j=0}^{D-1} cos(2\pi \cdot x_j)\right) + 20 + e^1,$$

$$-32.768 \leq x_j \leq 32.768, \quad j = 0,1,\cdots,D-1.$$

Solution: $f(\boldsymbol{x}^*) = 0, x_j^* = 0.$



Appendix B

Diffential Evolution Algorithm

The goal of an optimization problem for a function f of D variables, $f: \Omega \subseteq \mathbb{R}^D \to \mathbb{R}, \Omega \neq \emptyset$, where f is called the *objective function* (or *fitness*, or *cost function*), is to find a vector

$$X^* = (x_0^*, x_1^*, \cdots, x_{D-1}^*) \in \Omega$$

such that $\forall X \in \Omega: f(X) \geq f(X^*) = f^*$, where f^* is called a *global minimum*; X^* is the *minimum location* (*point* or *set*). Often, bound constraints are imposed:

$$L \leq X \leq H \ : L, H \in \mathbb{R}^D.$$

Because $max\{f(X)\} = -min\{f(X)\}$, the restriction to minimization is without loss of generality. In general, the optimization task is complicated by the existence of nonlinear objective functions with multiple local optima. A local minimum $\hat{f} = f(\hat{X})$ is defined as:

$$\exists \epsilon > 0 \; such \; that \; \forall X \in \Omega : \ \left\| X - \hat{X} \right\| < \epsilon \ \Rightarrow \ \hat{f} \leq f(X).$$

Differential Evolution is one of the numerical methods aimed at finding global optima.

As other evolutionary algorithms, DE works with *generations* (indexed by g). Each generation has a *population* of potential solutions. For a given generation g, the population $\mathbb{P}$ has Np vectors, so called *individuals* of the population. Np is the population size. Each such individual represents a potential optimal solution.

Let $f(\boldsymbol{x}) = f(x_0, x_1, \cdots, x_{D-1})$ be a function of D variables that we are interested in optimizing. An individual that is a potential solution is a point

$$X = (x_0, x_1, \cdots, x_{D-1}) = \{x_j\}, \quad j = 0, 1, \cdots, D-1.$$

Because each individual belongs to a population, we need to index the individuals within a population, so we have a second index, i:

$$X_i = (x_{i,0}, x_{i,1}, \cdots, x_{i,D-1}) = \{x_{i,j}\}, \quad j = 0, 1, \cdots, D-1; \quad i = 1, \cdots, Np.$$

But also, there are different generations. And population belongs to a generation. So we need one more index to keep track of generations, This index is g:

$$X_i^g = (x_{i,0}^g, x_{i,1}^g, \cdots, x_{i,D-1}^g) = \{x_{i,j}^g\}, \ j = 0,1, \cdots, D-1; \ i = 1, \cdots, Np.$$

The process is initialized by creating population $\mathbb{P}^0$ of randomly generated individuals within the boundary constraints:

$$\mathbb{P}^0 = \{x_{i,j}^0\} = \{rand_{i,j} \cdot (h_j - l_j) + l_j\},$$

where the $rand$ function generates random numbers uniformly distributed in the interval $[0,1]$, and l_j and h_j are the low and high bounds, respectively, $l_j \leq x_j \leq h_j$. Before the population can be initialized, both upper and lower bounds for each parameter must be specified. Once initialization bounds have been specified, a random number generator assigns each parameter of every vector a value from within the prescribed range.

Then, for each generation, all individuals in the population are updated according to a reproduction scheme. For each individual ind in the current population, a set π of other individuals is randomly selected from the population. To produce a new individual the operations of *differentiation* and *crossover* are performed one after another. Next, *selection* is used to choose the best. We now consider these operations in more detail.

1. Differentiation (Mutation). First, a set of randomly selected individuals $\pi = \{\xi_1, \xi_2, \cdots, \xi_n\}$ is necessary to perform differentiation. These individuals form the basis for various *strategies*. A strategy involves a *base vector* β and a *difference vector* δ. The result of differentiation is a so-called trial individual ω, given by

$$\omega = \beta + F \cdot \delta,$$

where F is the *constant of differentiation*. A typical strategy is to randomly extract three different individuals from the population.[1] The trial individual is equal to

$$\omega = \xi_3 + F \cdot (\xi_2 - \xi_1)$$

with the base vector being $\beta = \xi_3$ and the difference vector $\delta = (\xi_2 - \xi_1)$. This is a classic strategy. This strategy is implemented as follows. *Differential mutation* adds a scaled, randomly sampled, vector difference to a third vector. To create a mutant vector, $\boldsymbol{v}_{i,g}$, three different, randomly chosen, vectors are recombined as

$$\boldsymbol{v}_{i,g} = \boldsymbol{x}_{r0,g} + F \cdot \left(\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g}\right)$$

The scale factor, F, is a real number that controls the rate at which the population evolves. It is usually suggested that F is positive, $F \in (0,1+)$. While there is no upper limit on F, effective values are rarely greater than 1.0. Recently, Feoktistov suggested that negative values in the region $[-1,0)$ may be useful sometimes. The implementation of the Excel Tool limits acceptable values to [-1, 0) or (0, 2.5]. The *base vector* index, $r0$, can be determined in a variety of ways, but in the classic strategy it is assumed to be a randomly chosen vector index that is different from the *target vector* index, i. Except for being distinct from each other and from both the base and target vector indices, the *difference vector* indices, $r1$ and $r2$, are also randomly selected once per mutant.

Other strategies for differentiation are possible and have been suggested in the literature on DE.

Another strategy is "either/or algorithm."[2] The idea is to implement a search by defining a mutation probability such that trial individuals (trial vectors) that are pure mutants occur with probability p_F and those that are pure recombinants occur with probability $(1 - p_F)$,

$$\omega = \boldsymbol{u}_{i,g} = \begin{cases} \boldsymbol{v}_{i,g} = \boldsymbol{x}_{r0,g} + F \cdot \left(\boldsymbol{x}_{r1,g} - \boldsymbol{x}_{r2,g}\right) & if\ rand_i(0,1) < p_F \\ \boldsymbol{w}_{i,g} = \boldsymbol{x}_{r0,g} + K \cdot \left(\boldsymbol{x}_{r1,g} + \boldsymbol{x}_{r2,g} - 2\boldsymbol{x}_{r0,g}\right) & otherwise \end{cases}$$

This scheme accommodates functions that are best solved by either mutation only ($p_F = 1$) or recombination only ($p_F = 0$), as well as generic functions that can be solved by randomly interleaving both operations ($0 < p_F < 1$). Price, Storn, and Lampinen recommend $K = 0.5 \cdot (F + 1)$ as a good first choice for K given F. C-style pseudo-code for this strategy is given in **Exhibit B1** below.

[1] This strategy is proposed by Rainer Storn and Kenneth Price. "Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces." Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[2] Kenneth V. Price, Rainer M. Storn, Jouni A. Lampinen. *Differential Evolution. A Practical Approach to Global Optimization*. Springer-Verlag, 2005, Section 2.6.5.

Exhibit B1. C-style Pseudo-code for "either/or" strategy.

```
...
if( rand_i(0,1) < p_F)                                  /* mutate or recombine */
{
      u[i] = x[r0] + F*(x[r1] - x[r2]);                 /* mutate */
}
else
{
      u[i] = x[r0] + K*(x[r1] + x[r2] - 2*x[r0]);       /* recombine */
}
...
```

2. Crossover. Once the trial individual ω is generated, it is recombined with the target individual ind. In a *crossover*, a new trial individual inherits components of the target with some probability,

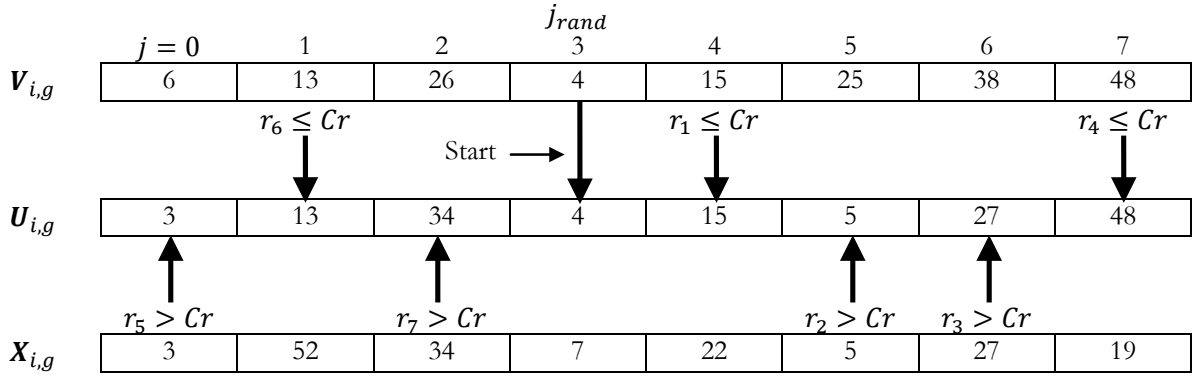
$$\omega_j = \begin{cases} \omega_j & if\ rand_j \le Cr \\ ind_j & otherwise \end{cases},$$

where $j = 0,1,\cdots,D-1$, $rand_j \in [0,1)$ and $Cr \in [0,1]$ is the constant of crossover. The crossover described above is *combinatorial crossover*. Other types of crossover may be used: for example, uniform (binomial) crossover.

DE's version of uniform crossover begins by taking a randomly chosen parameter from the mutant so that the trial vector will not replicate the target vector. Comparing Cr to $rand_j(0,1)$ determines the source for each remaining trial parameter. If $rand_j(0,1) \le Cr$, then the parameter comes from the mutant; otherwise, the target is the source. The process is illustrated in **Exhibit B2**.[3] The number of inherited mutant parameters follows a binomial distribution, because parameter origins are determined by a finite number of independent trials with two outcomes with constant probabilities.

Exhibit B2. Uniform crossover.

Once an initial, randomly chosen parameter is inherited from the mutant (for example, $j_{rand} = 3$), $D-1$ independent trials are conducted to determine the source of the remaining parameters. If $rand_j(0,1) \le Cr$, the mutant donates a parameter value; otherwise, parameters are copied from the target.



[3] This is the crossover implemented in the Excel tool code.

3. *Selection*. Selection is performed by comparing the objective function values of the target and trial individuals. If the trial individual has a lower value of the objective function, then it replaces the target individual. This is the case of *elitist* or so-called "greedy" selection.

$$ind = \begin{cases} \omega & if\ f(\omega) \le f(ind) \\ ind & otherwise \end{cases}$$

Selection can happen immediately, or can be delayed until all are mutated. With immediate selection, one array is maintained for the population, and the trial immediately replaces the ancestor.[4] With delayed replacement, two population arrays are maintained (one for the old and one for the new population). After all mutations are performed, the individuals are copied into the new population based on their fit.

The three steps above repeat from generation to generation. For stopping conditions, the user usually fixes the number of generations, g_{max}. Another stopping condition can be arrival within a desirable precision within a specified value VTR (*value-to-reach*).[5]

Exhibit B3 presents C-style pseudo-code for classic DE. The vector indices $r0, r1$, and $r2$ are all different and distinct from the target index, i. In the shown version, selection is delayed until the trial population is complete.

Exhibit B3. C-Style Pseudo Code for Classic DE.

```
/* Initialize ... */
do
{
      for(i=0; i<Np; i++) {   /* r0 != r1 != r2 != i */
            do r0=floor(rand(0,1)*Np); while (r0 == i);
            do r1=floor(rand(0,1)*Np); while (r1==r0 or r1 == i);
            do r2=floor(rand(0,1)*Np); while (r2==r1 or r2 == r0 or r2 == i);
            jrand=floor(D*rand(0,1));

            for(j=0; j<D; j++) {
                  /* Generate a trial vector */
                  if ( rand(0,1)<=Cr or j==jrand ){
                        u[j,i] = x[j, r0] + F*( x[j,r1]-x[j,r2] );
                        /* Check for out of bounds */
                  }
                  else {
                        u[j,i] = x[j, i];
                  }
            } /* for j<D loop */

      } /* for i<Np loop */

      /* Select the next generation */
      for (i=0; i<Np; i++){
            if( f(u[i]) <= f(x[i]) ) x[i] = u[i];
      } /* for i<Np loop */
} while (termination criterion not met)
```

[4] This is the selection that is implemented in the Excel tool.

[5] The stopping condition implemented in the tool is the maximum number of generations. The VTR rule is not implemented. During the computation, user can use ESC (twice) or Ctrl-Break to interrupt calculations (an option to resume or interrupt computations will be given).

This completes the description of DE. A brief discussion follows.

Discussion.

Notice that there are three control parameters in this algorithm:

1. Np = population size
2. F = constant of differentiation
3. Cr = constant of crossover.

DE applies variation (differentiation and crossover) sequentially to each individual in the population. The result of a variation is a child, called a trial individual. Two replacement strategies are possible. In the first one, the trial immediately replaces its ancestor in the population if its fitness is better than or equal to its ancestor's fitness. In the second approach, all mutations are performed first and after that the individuals are copied to the next population based on their fit.

Notation. The technical name for the method described above is "DE/rand/1/bin" because the base vector is randomly chosen, 1 vector difference is added to it, and because the number of parameters donated by the mutant vector closely follows a binomial distribution. Usually, this case is referred to as the "classic DE."

A comment on the *Crossover constant*. Lampinen and Zelinka have shown that the number of possible trial vectors, n_{trial}, that can be created with DE's uniform (binomial) crossover is

$$n_{trial} = \begin{cases} Np^3 - 3Np^2 + 2Np & if\ Cr = 1 \\ D \cdot Np \cdot (Np^3 - 3Np^2 + 2Np) & if\ Cr = 0 \\ 2^D \cdot Np \cdot (Np^3 - 3Np^2 + 2Np) & otherwise \end{cases}$$

The number of possible trial vectors is constant when $0 < Cr < 1$, but uniform crossover has a distributional bias, because not all configurations are equally likely. DE does not eliminate distribution bias but relies on Cr to provide the means for controlling it. A low value, $Cr \approx 0.0$ minimizes disruption by incrementally altering just a few parameters of a vector at a time. The values $Cr \approx 1.0$ favor exploration by drawing most trial vectors directly from the mutant population. The value $Cr = 1 - 1/D$ allows to inherit the minimal number of the newly created individual's genes.

Parameter Cr can be thought of as a *mutation rate*, i.e. the (approximate) probability that a parameter will be inherited from a mutant. In DE, the average number of parameters mutated for a given Cr depends on the specific crossover model (e.g., exponential or binomial) but in each, a low Cr corresponds to a low mutation rate. Previous research has discovered that for some functions (e.g., the Rastrigin function, as well as other) low values of Cr to be the most effective. For other functions, high values of Cr were effective. Different functions in the extensive testing by Storn and Price[6] were solved with either $0 \leq Cr \leq 0.2$ or $0.9 \leq Cr \leq 1$. The reason for the bifurcation of the crossover control space was understood when researchers realized that functions solvable with low Cr were decomposable, while those requiring $Cr \approx 1$ were not.

[6] Kenneth V. Price, Rainer M. Storn, Jouni A. Lampinen. *Differential Evolution. A Practical Approach to Global Optimization*. Springer-Verlag, 2005, Section 2.6.2.

Decomposable functions. A decomposable function $f(x_0, x_1, \cdots, x_{D-1})$ of D variables can be written as a sum of D one-dimensional functions (not necessarily all the same):

$$f(\boldsymbol{x}) = \sum_{j=0}^{D-1} f_j(x_j).$$

Decomposability simplifies the task of optimization because each parameter can be optimized independently, allowing the task of optimizing a single D-dimensional function to be broken up into D one-dimensional problems. Once the optima of the D one-dimensional functions have been located, they can be combined to specify the optimum of the original D-dimensional function,

$$f(\boldsymbol{x}^*) = f(x_0^*, x_1^*, \cdots, x_{D-1}^*), \;\; min\left[f_j(x_j)\right] = f_j(x_j^*), \;\; j = 0,1, \cdots, D-1.$$

For such functions, changing only one parameter (e.g., $Cr = 0$) before each evaluation can be viewed as a single step in an independent, one-dimensional optimization. If the parameter being modified is randomly selected, then the D one-dimensional optimizations proceed as arbitrarily sequenced tasks.

The role of the crossover constant Cr is to provide opportunity to exploit decomposability (if it is present), and to provide extra diversity to the pool of possible trial vectors, especially when $Cr \approx 1$. In the general case of parameter-dependent functions, Cr should be close to 1.

A comment on *Differentiation* (search strategies). The differentiation operation can be realized by many different search strategies. A strategy involves a *base vector* β and a *difference vector* δ. The result of differentiation is a so-called trial individual ω, given by

$$\omega = \beta + F \cdot \delta,$$

where F is the *constant of differentiation*. Strategies may differ in terms of how they select the base vector, how they construct the difference vector, and whether they keep F as a constant or randomize F.[7]

With respect to the *base vector* selection, several base vector selection schemes have been proposed that bias selection toward better vectors. For example, the algorithm DE/best/1/bin proposed by Rainer Storn[8] always selects the best-so-far vector (best) as the base vector, adds a single (1) scaled vector difference to it, then creates a trial vector by uniformly crossing (bin) the resulting mutant with the target vector. In this algorithm, the best vector always has the lowest objective function value in the current population,

$$r0 = best, if \; \forall i \in (0,1, \cdots, Np-1), f\left(\boldsymbol{x}_{best,g}\right) \leq f\left(\boldsymbol{x}_{i,g}\right).$$

When compared to random base vector selection at the same Np, best-so-far base vector selection usually speeds convergence, reduces the odds of stagnation, and lowers the probability of success.

[7] For additional discussion of strategies see Vitaliy Feoktistov, *Differential Evolution: In Search of Solutions*, Springer, 2006, Chapter 3.

[8] Storn R (1996). On the usage of differential evolution for function optimization. In: Smith MH, Lee MA, Keller J, Yen J (eds) Proceedings of the 1996 biennial conference of the North American fuzzy information processing society - NAFIPS, June 19-22, Berkeley, CA, USA. IEEE Press, New York, pp 519 - 523.

Two other schemes that bias selection toward better vectors without creating the intense selection pressure that the "best" method applies have been proposed. Kenneth Price[9] suggested that a base vector's objective function value must be less than or equal to that of the target vector, $\boldsymbol{x}_{i,g}$:

$$r0 = better, if\ f(\boldsymbol{x}_{better,g}) \leq f(\boldsymbol{x}_{i,g}).$$

The other method, DE/target-to-best/1/bin (called "rand-to-best" in Rainer Storn[10], 1996), uses arithmetic recombination to generate a base vector that lies on a line between the target vector and the best-so-far vector:

$$\boldsymbol{x}_{r0,g} = \boldsymbol{x}_{i,g} + k \cdot (\boldsymbol{x}_{best,g} - \boldsymbol{x}_{i,g}),\ \ k \in [0,1] = \ constant.$$

The constant, k, controls the bias toward the best-so-far solution.

Values of control parameters: Rules of Thumb.

A low value of the crossover constant, $Cr = 0.2$ should be the default value for separable functions or those that exhibit limited parameter dependence. The values $Cr = 0.9$ or $Cr = 0.95$ are appropriate for functions that have parameter dependence.

When $Cr = 0.2$, F may be as small as 0.3, but $F = 0.5$ will be a better first choice. As Cr increases, F usually needs to increase as well. For example, if $Cr = 0.9$, then $F \geq 0.8$ will be more likely to give regular convergence than will $F = 0.5$.

Like F, population size Np may also have to be increased as Cr increases. Setting $Np = 5 \cdot D \cdot Cr$ is usually a low-end default setting, but for highly multi-modal, parameter-dependent functions, Np may need to be $10 \cdot D$ or higher.

[9] Price K V (1997) Differential evolution vs. the functions of the second ICEO. In: Proceedings of the 1997 IEEE international conference on evolutionary computation, Indianapolis, Indiana, USA. IEEE Press, New York, pp. 153 - 157.

[10] Storn R (1996). On the usage of differential evolution for function optimization. In: Smith MH, Lee MA, Keller J, Yen J (eds) Proceedings of the 1996 biennial conference of the North American fuzzy information processing society - NAFIPS, June 19-22, Berkeley, CA, USA. IEEE Press, New York, pp 519 - 523.

Appendix C

Installation Instructions for Excel 2007

The tool needs to be installed only once. After this, it will be available when you open Excel. Installation consists of **Four** easy steps.

Step 1. Download the file that contains the tool. Filename: **DEv1.1_AddIn.xlsm**
Step 2. Open the file in Excel.
Step 3. Use "Save As" to save the file as Excel Add-In in the Add-In directory.
Step 4. Install the Add-In.

We now go through these steps in more detail.

Step 1. Download the file **Dev1.1_AddIn.xlsm** -- this is Excel file that contains the Visual Basic code for Differential Evolution optimization. You may place this file in any folder that is convenient.

Step 2. Open the downloaded file in Excel. Depending on the security level of your Excel set up, Excel may give you a Security Warning because the file has macros. In this case, you need to enable Macros. Click "**Options...**" that is visible next to the security warning. In the dialog that shows, select "Enable this content" and click **OK**. If you do not see Security Warning, this means that your Excel installation already allows macros.

Note that a new menu item, "OPTIMIZATION" will show. Clicking on the "OPTIMIZATION" menu will show a ribbon that has the "Differential Evolution" button.

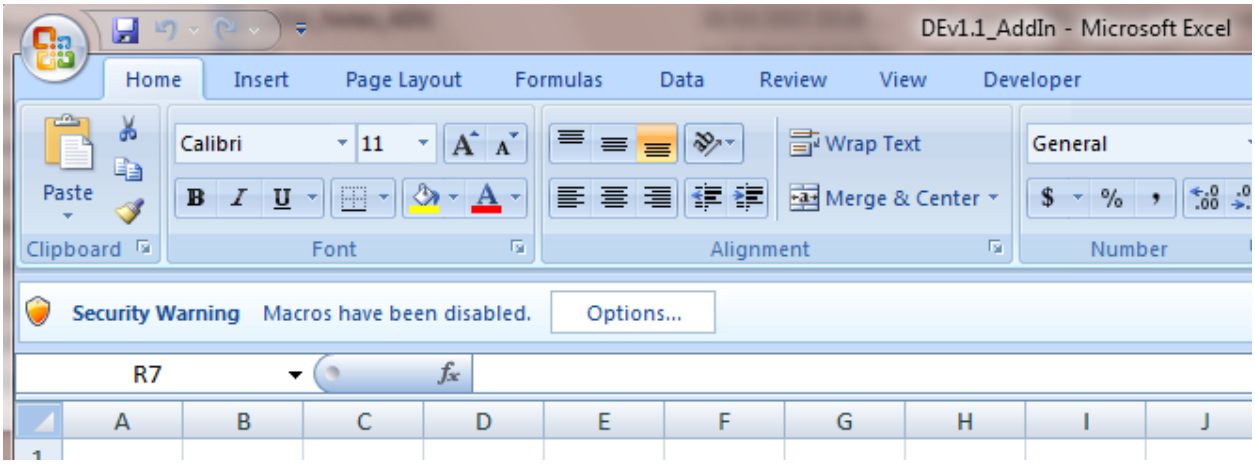


Exhibit C1. Security Warning (Excel 2007).

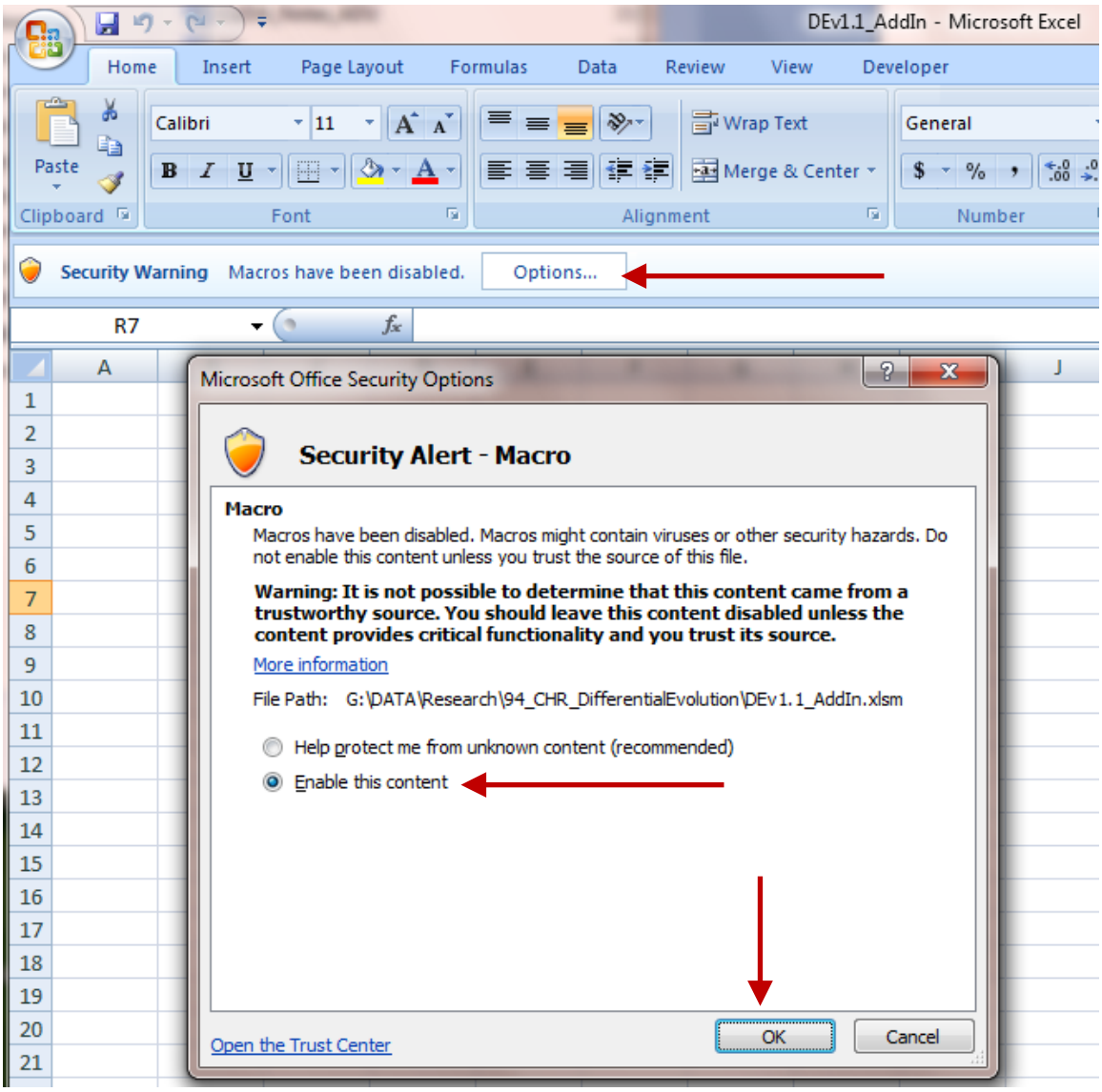


Exhibit C2. Enable Macros (Excel 2007)

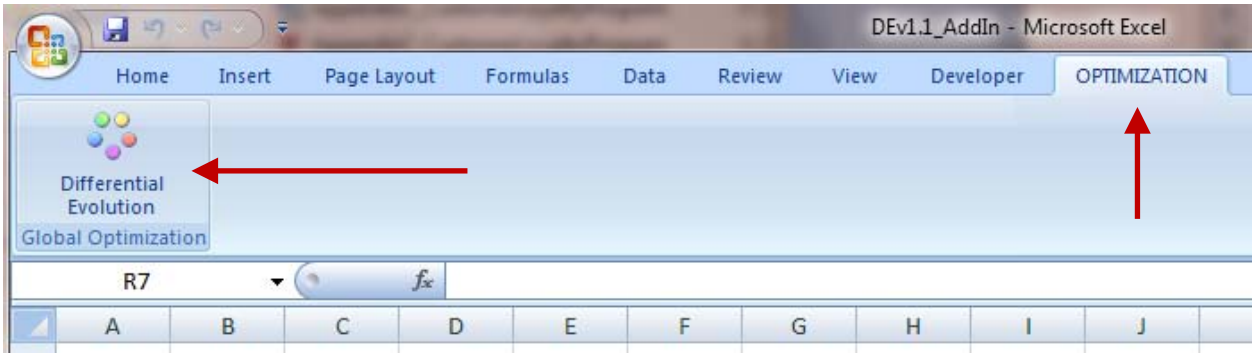


Exhibit C3. New "OPTIMIZATION" menu with a button for Differential Evolution optimizer (Excel 2007).

Step 3. Now save the opened file as an Excel Add-In. Excel will automatically put the saved add-in in the correct folder. To do this, follow instructions for this step.

Click **Office Button** in the upper-left corner of Excel and select "**Save As**", "**Other Formats**".

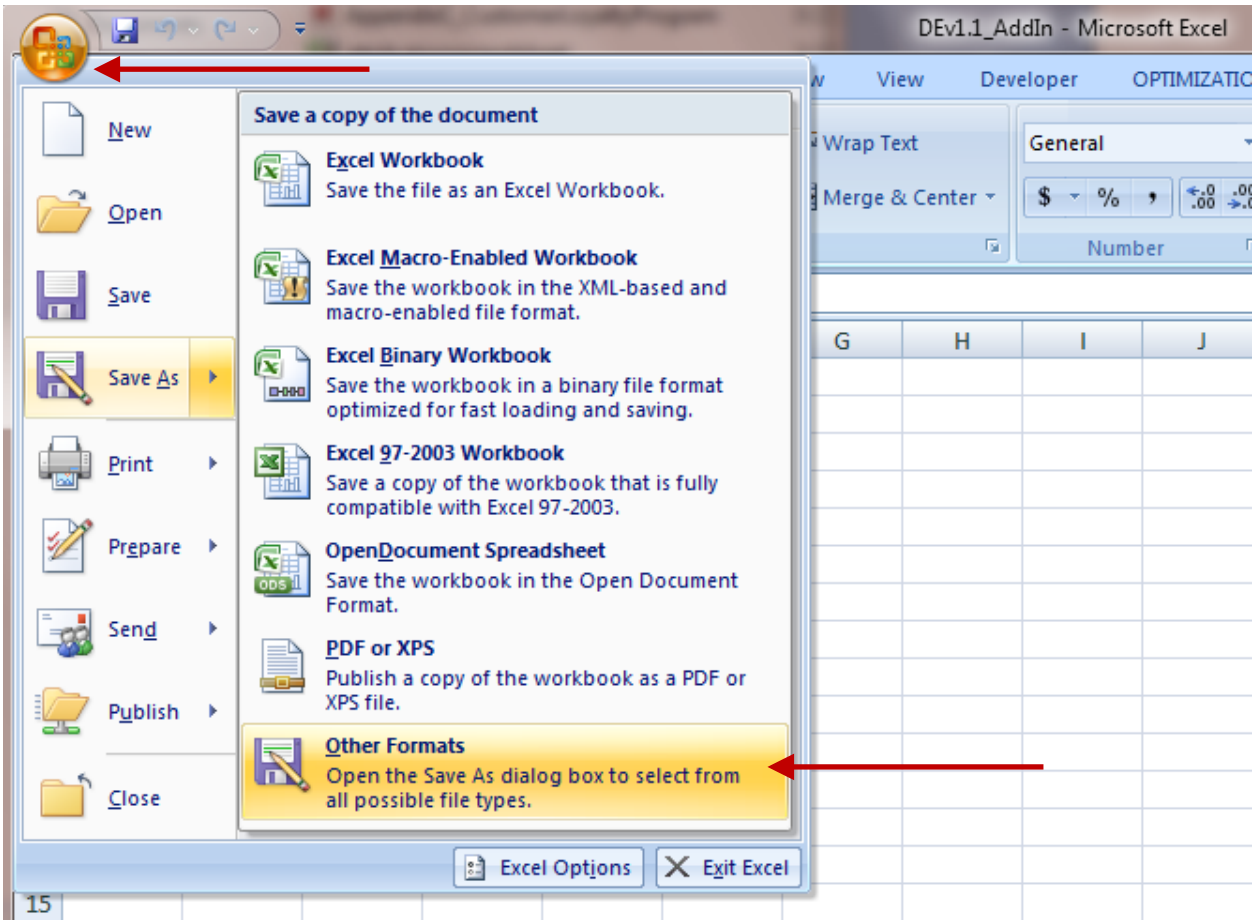


Exhibit C4. "Save As" interface (Excel 2007).

When you select "Other Formats", Excel dialog for saving files will appear.

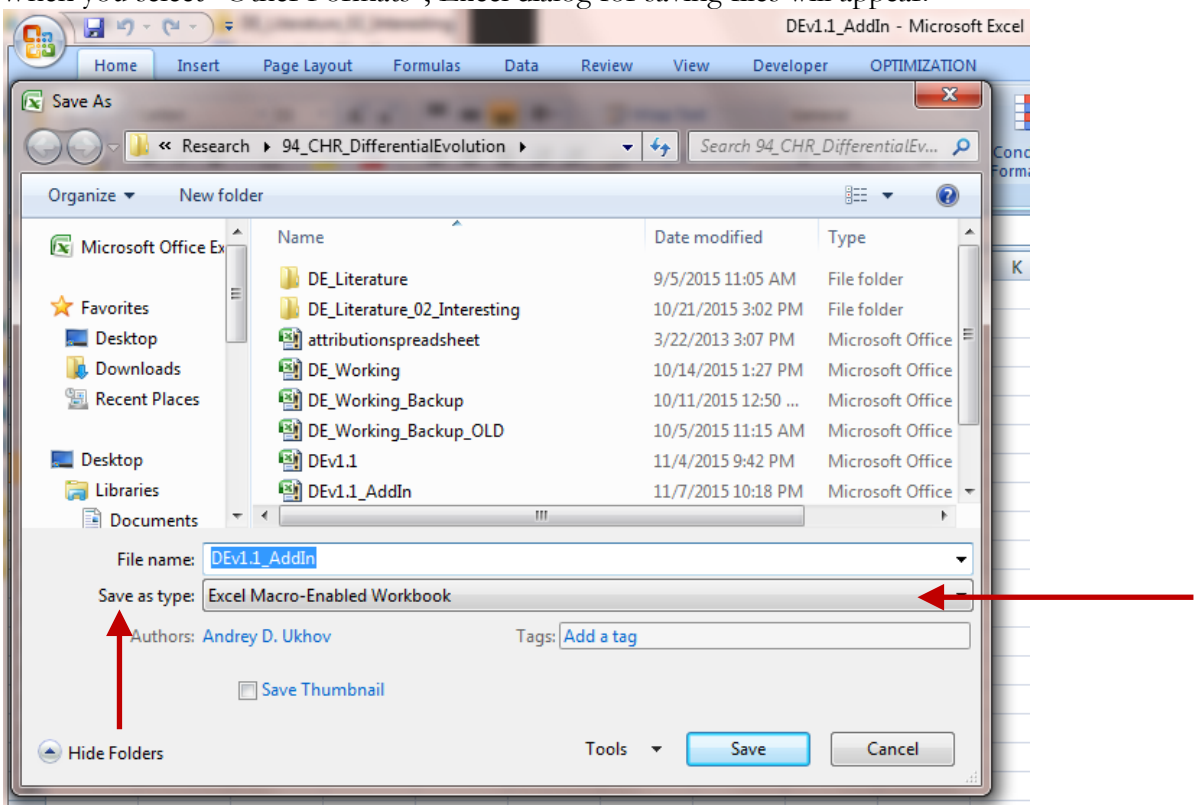
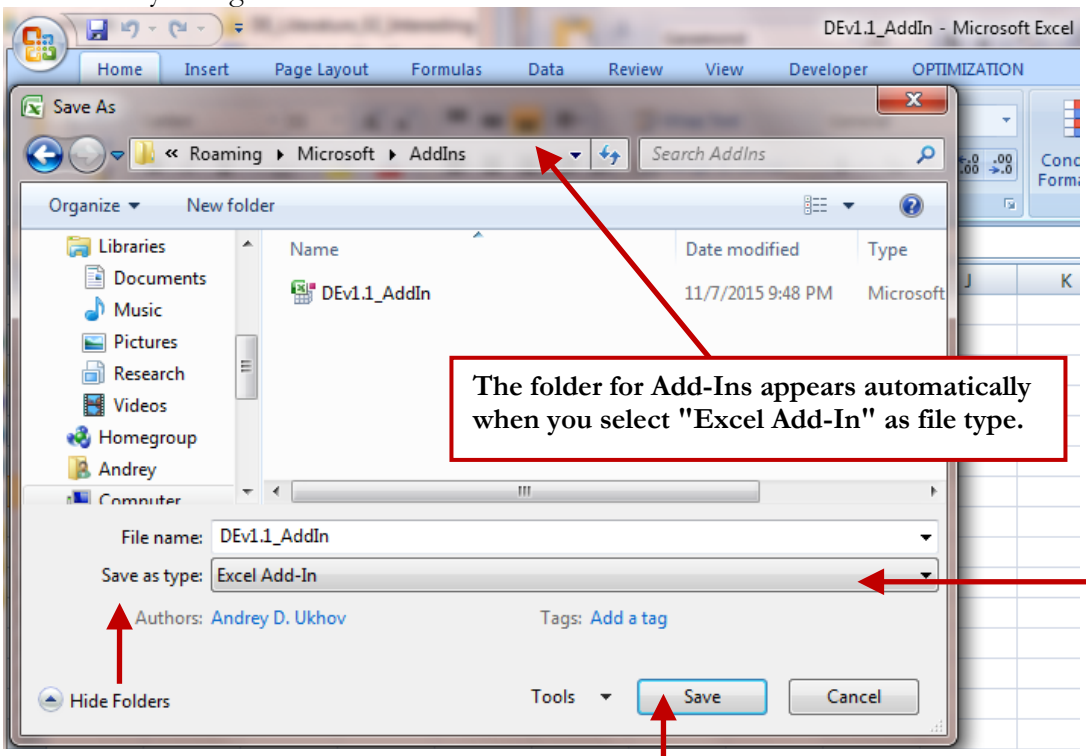


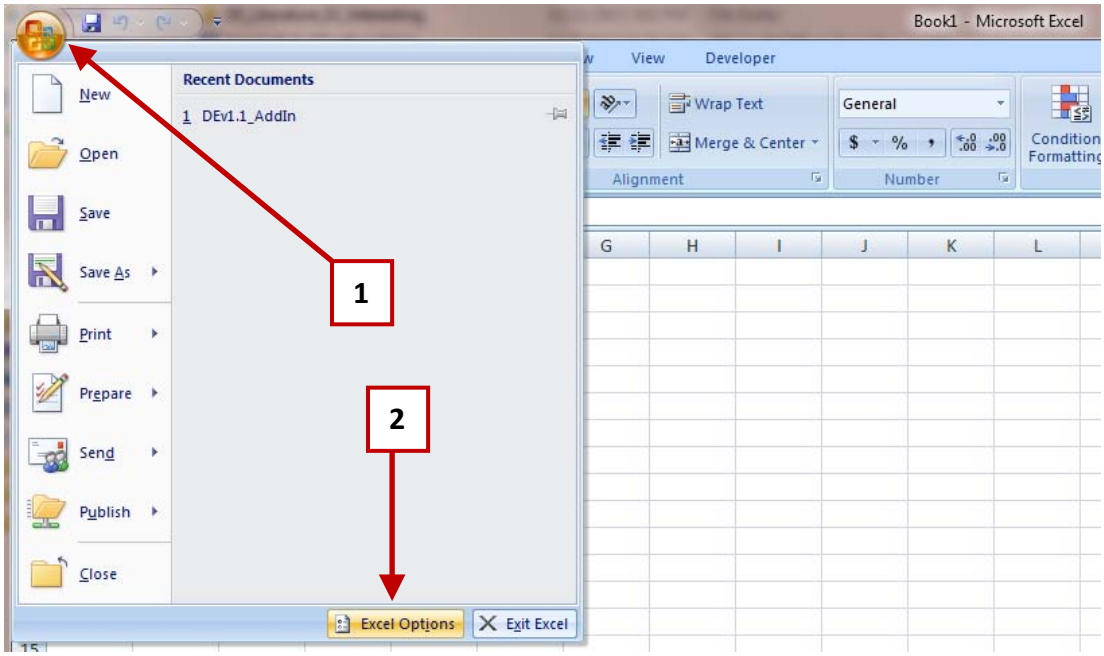
Exhibit C5. Excel Dialog for saving files.

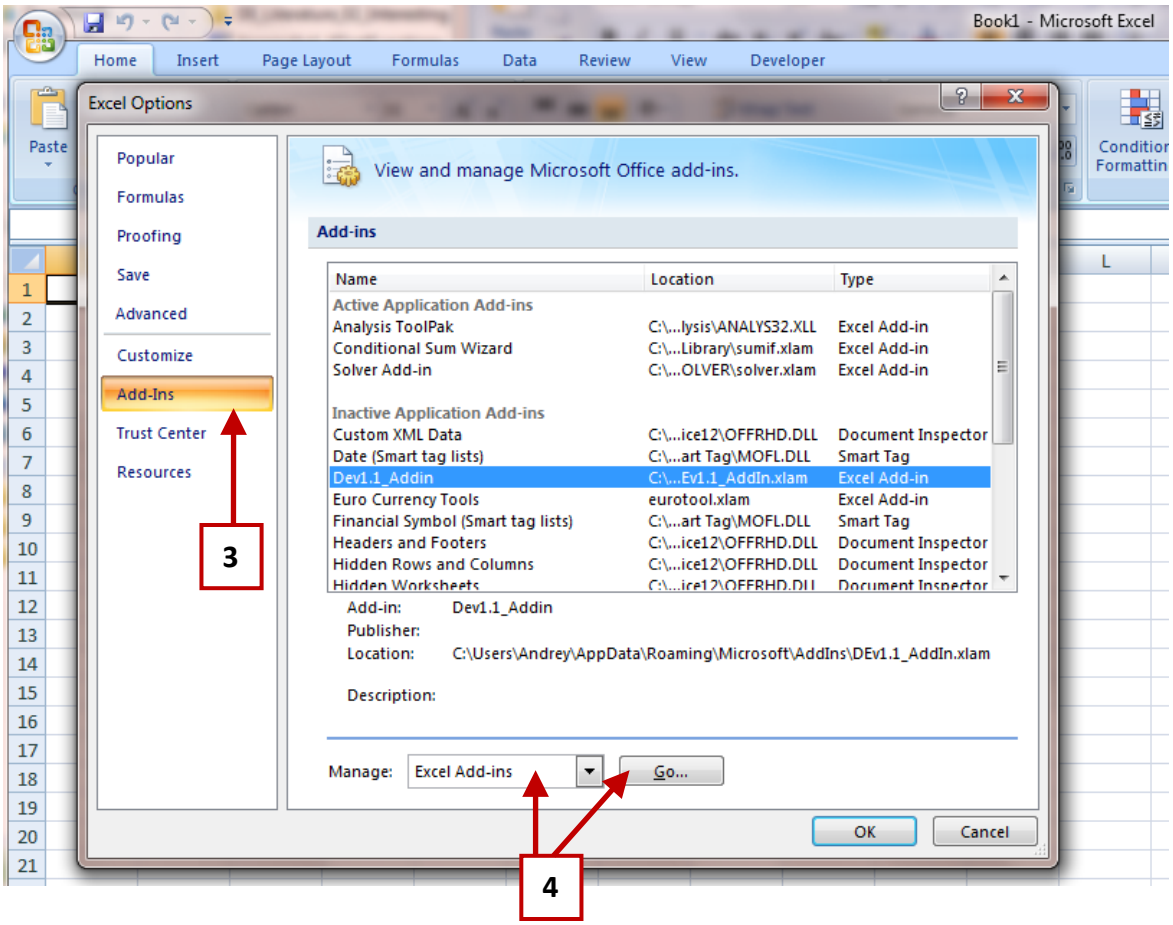
Select 'Excel Add-In' from the drop-down 'Save as type' menu and click "SAVE". Excel will automatically change the folder to the folder where the add-ins are stored.



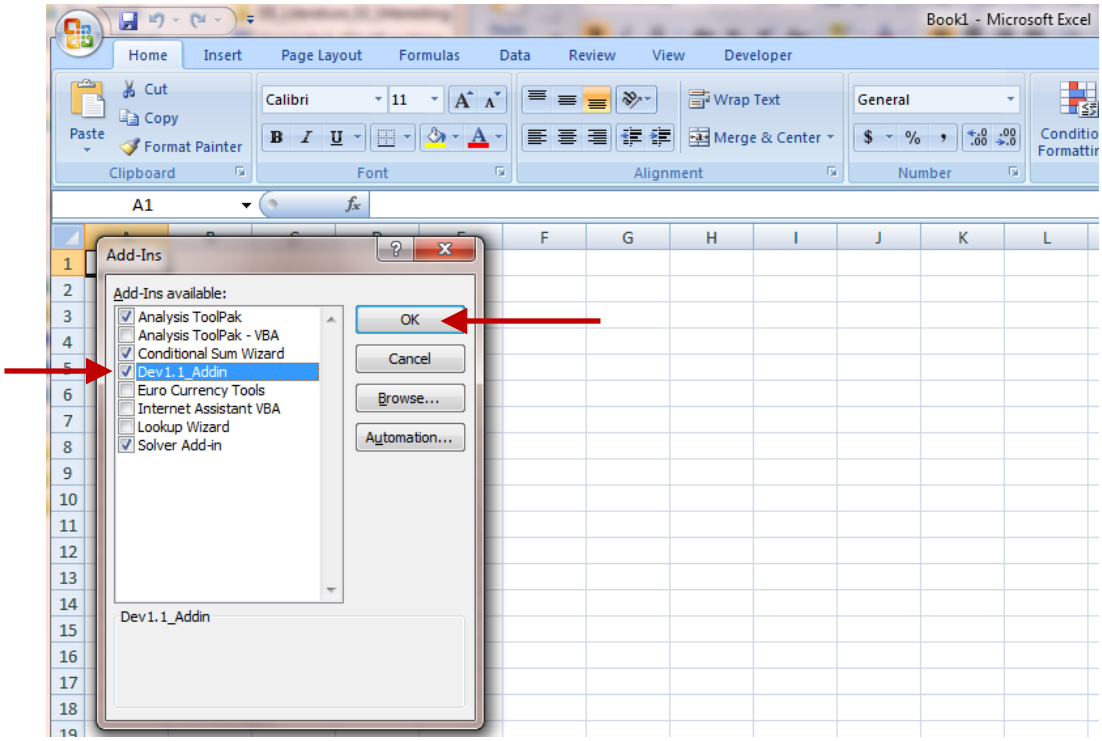
Step 4. In this step, you will add the new Add-In so it is available whenever you start Excel.
Exit Excel.
Start Excel, without opening any files.

1. Click the **Microsoft Office Button** .
2. Then click **Excel Options**.
3. Click the **Add-Ins** category.
4. In the **Manage** box, click **Excel Add-ins**, and then click **Go**.

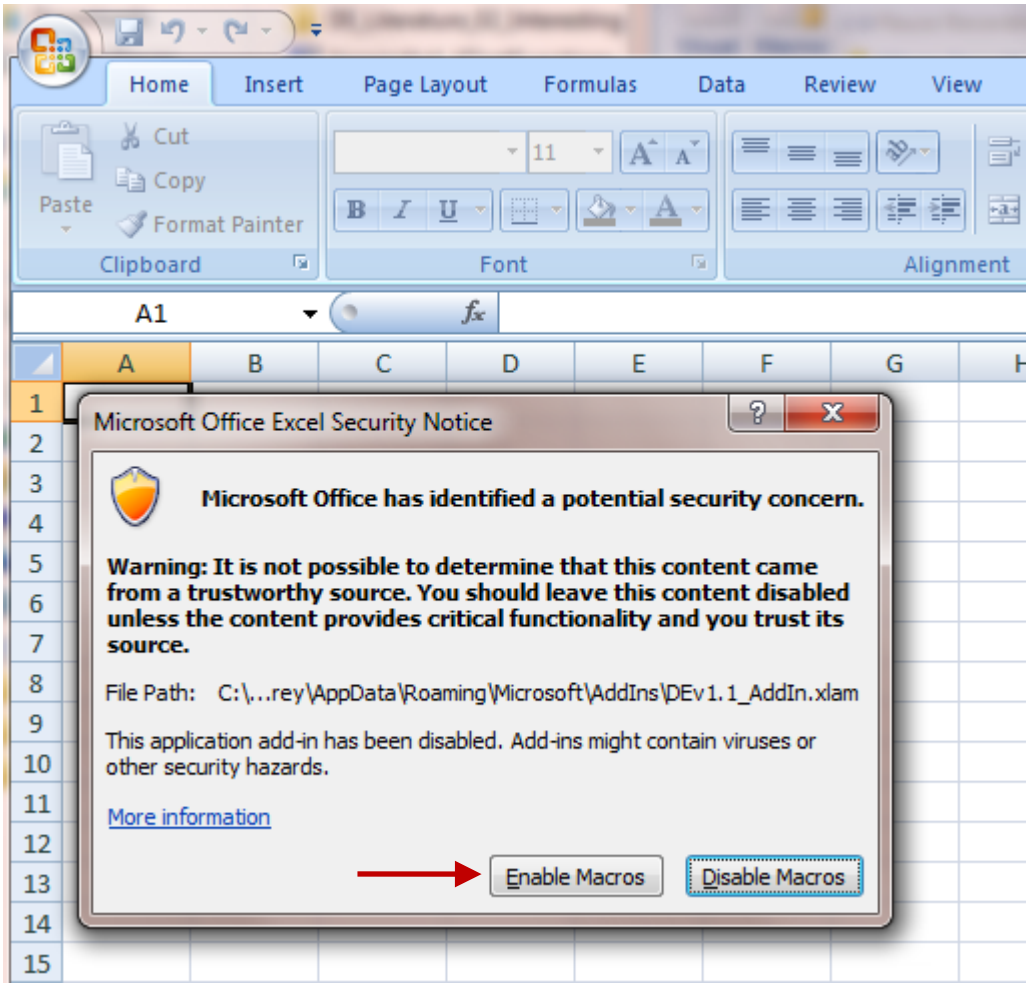




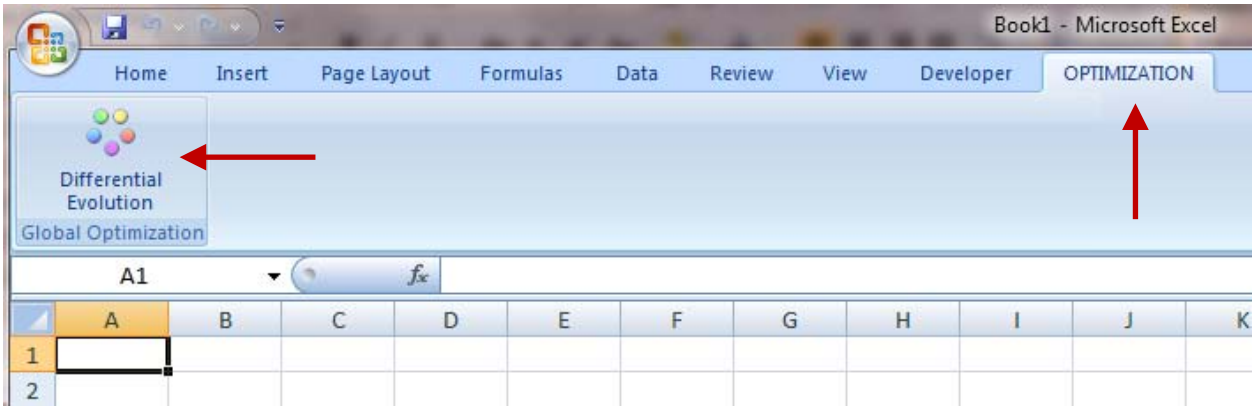
5. To load the add-in, in the **Add-Ins available** box, select the check box next to the **DEv1.1_AddIn** and then click **OK**.



Depending on your Excel installation, you may see a Security Warning (shown below). Click "**Enable Macros**" to finish the installation. If you do not see the Security Warning, this means that your Excel installation already allows macros and installation will complete.



The Add-In is now installed. You will see that the new "OPTIMIZATION" menu appeared with a button for Differential Evolution optimizer. The installation is complete.



Appendix D

Installation Instructions for Excel Versions Since 2007

The tool needs to be installed only once. After this, it will be available when you open Excel. Installation consists of **Four** easy steps.

Step 1. Download the file that contains the tool. Filename: **DEv1.1_AddIn.xlsm**
Step 2. Open the file in Excel.
Step 3. Use "Save As" to save the file as Excel Add-In in the Add-In directory.
Step 4. Install the Add-In.

We now go through these steps in more detail.

Step 1. Download the file **Dev1.1_AddIn.xlsm** -- this is Excel file that contains the Visual Basic code for Differential Evolution optimization. You may place this file in any folder that is convenient.

Step 2. Open the downloaded file in Excel. Depending on the security level of your Excel set up, Excel may give you a Security Warning because the file has macros. In this case, you need to enable Macros. Click "**ENABLE CONTENT**" that is visible next to the security warning. This will enable macros. If you do not see Security Warning, this means that your Excel installation already allows macros.

Note that a new menu item, "OPTIMIZATION" will show. Clicking on the "OPTIMIZATION" menu will show a ribbon that has the "Differential Evolution" button.

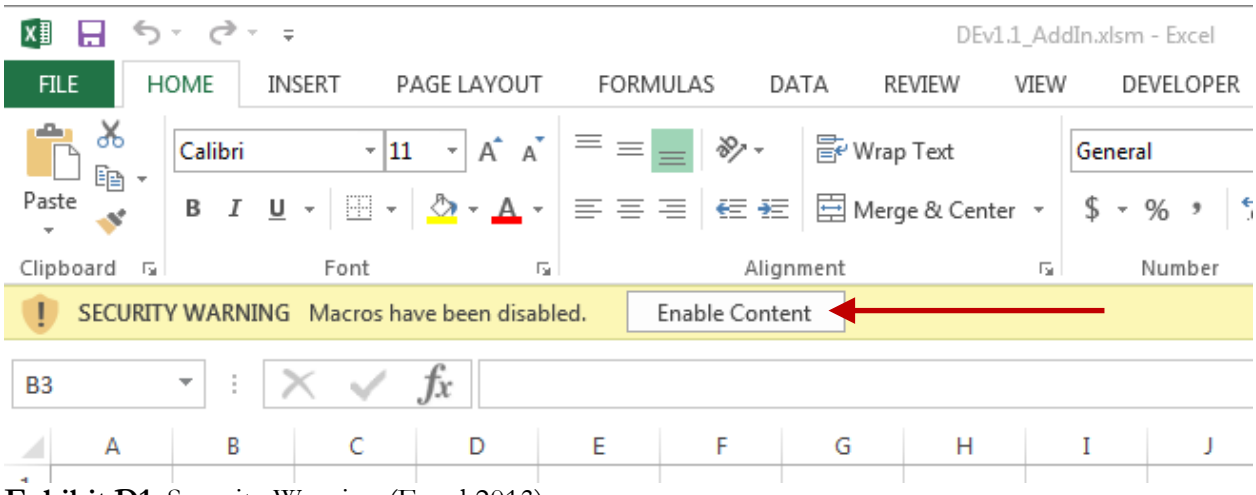


Exhibit D1. Security Warning (Excel 2013).

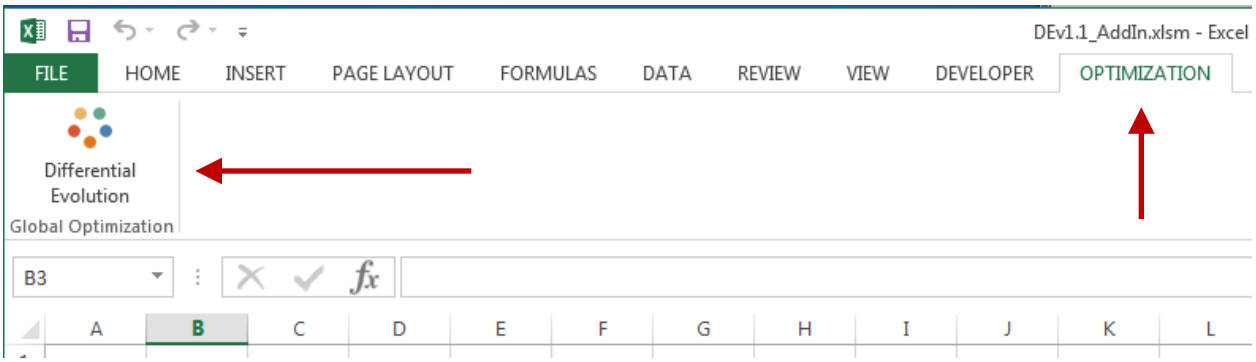


Exhibit D2. New "OPTIMIZATION" menu with a button for Differential Evolution optimizer (Excel 2013).

Step 3. Now save the opened file as an Excel Add-In. Excel will automatically put the saved add-in in the correct folder. To do this, follow instructions for this step.

Click **FILE** in the upper-left corner of Excel and select "**Save As**", "**Browse**".

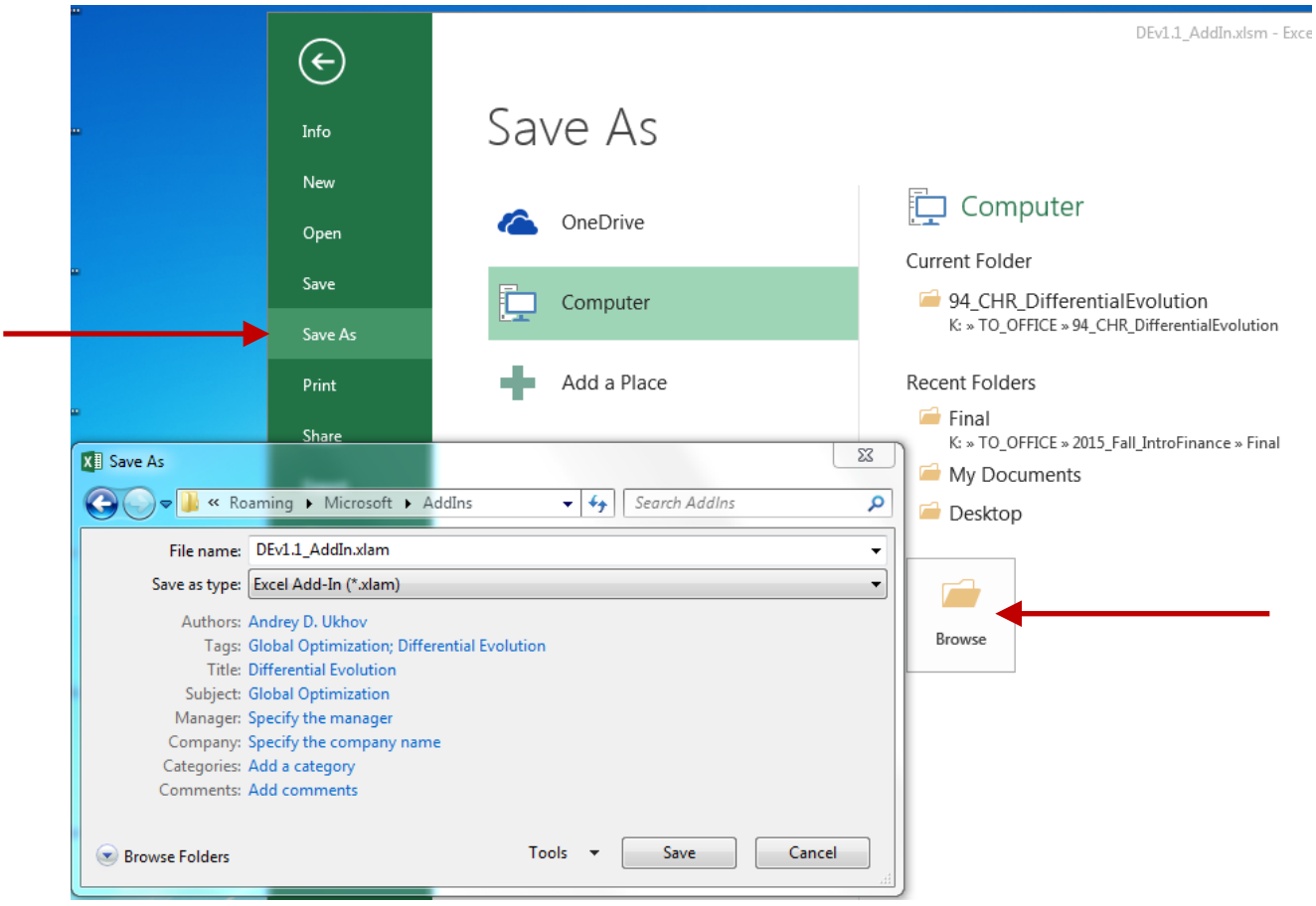


Exhibit D3. "Save As" interface (Excel 2007).

When you select "Browse", Excel dialog for saving files will appear.

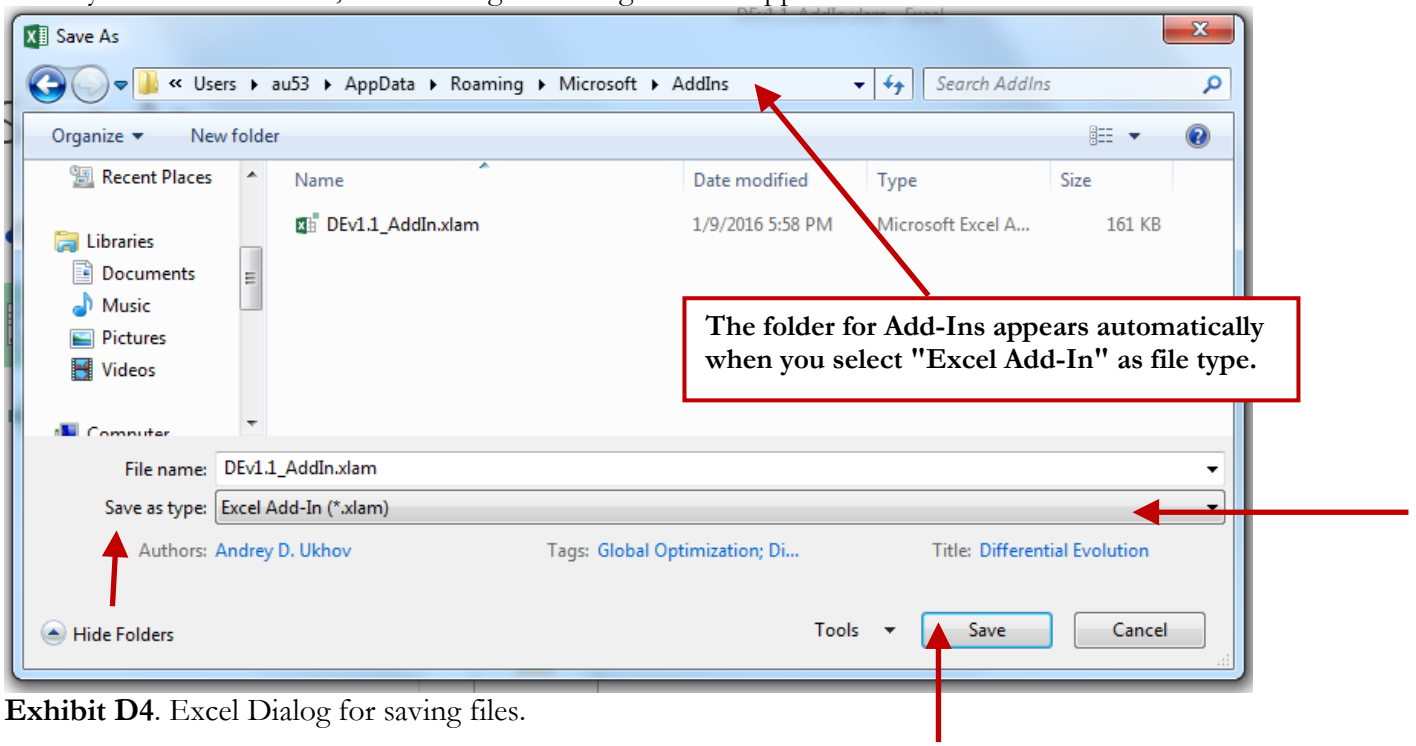
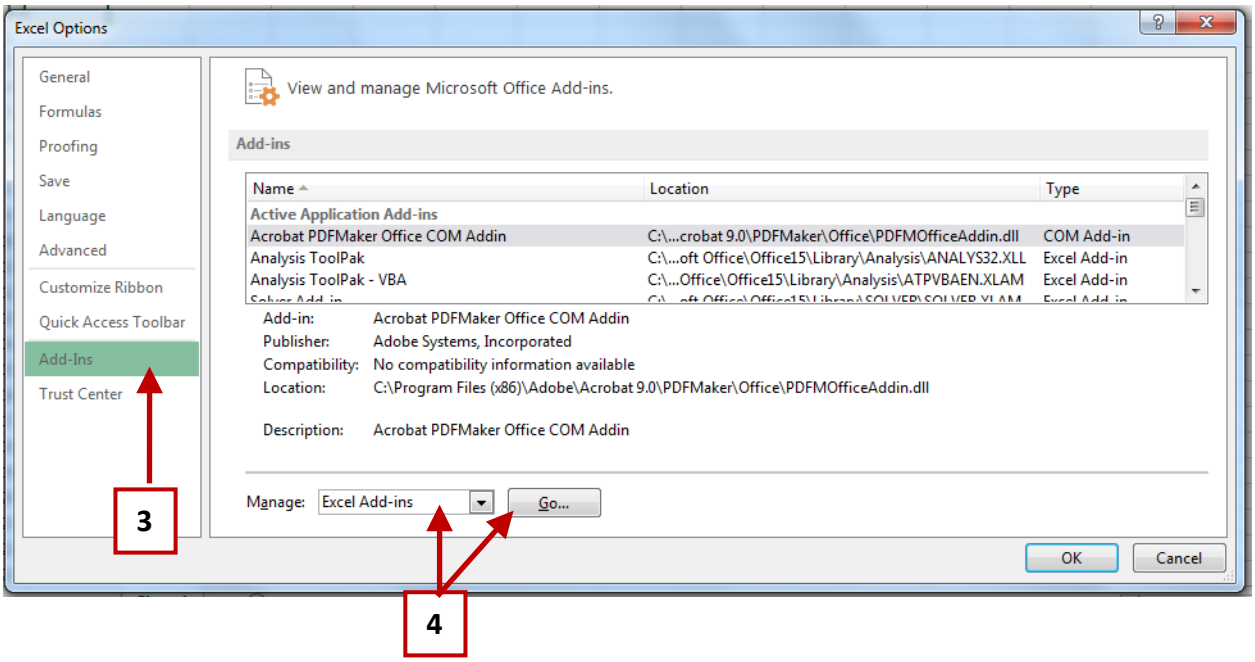


Exhibit D4. Excel Dialog for saving files.

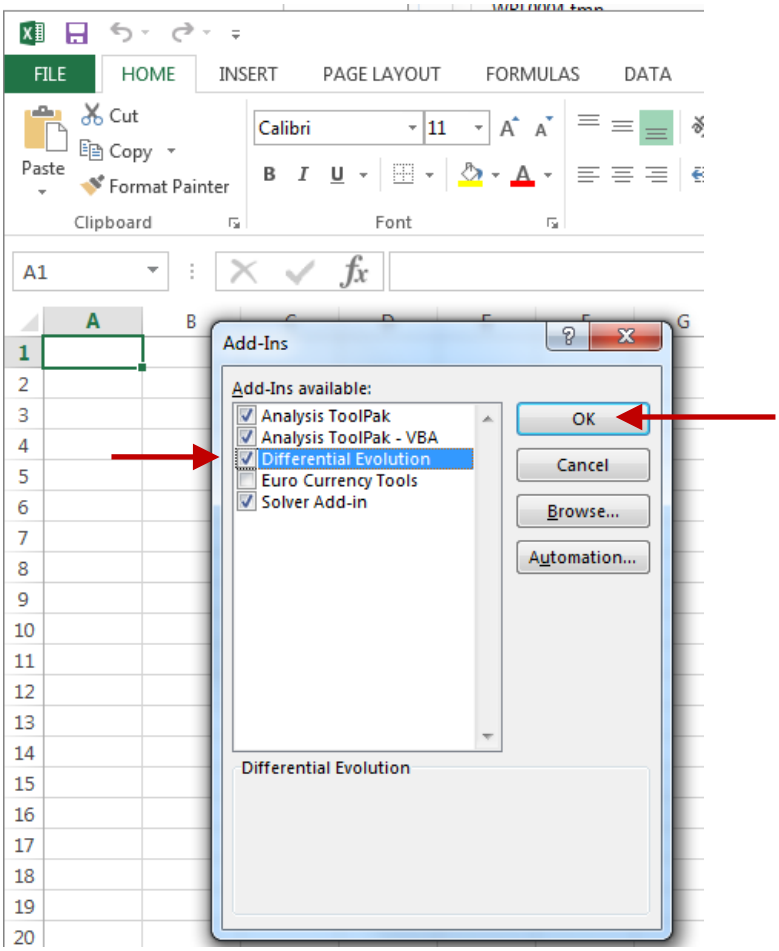
Select 'Excel Add-In' from the drop-down 'Save as type' menu and click "SAVE". Excel will automatically change the folder to the folder where the add-ins are stored.

Step 4. In this step, you will add the new Add-In so it is available whenever you start Excel.
Exit Excel.
Start Excel, without opening any files.

1. Click the **FILE**.
2. Then click **Options**.
3. Click the **Add-Ins** category.
4. In the **Manage** box (located toward the bottom of the interface), click **Excel Add-ins**, and then click **Go**.

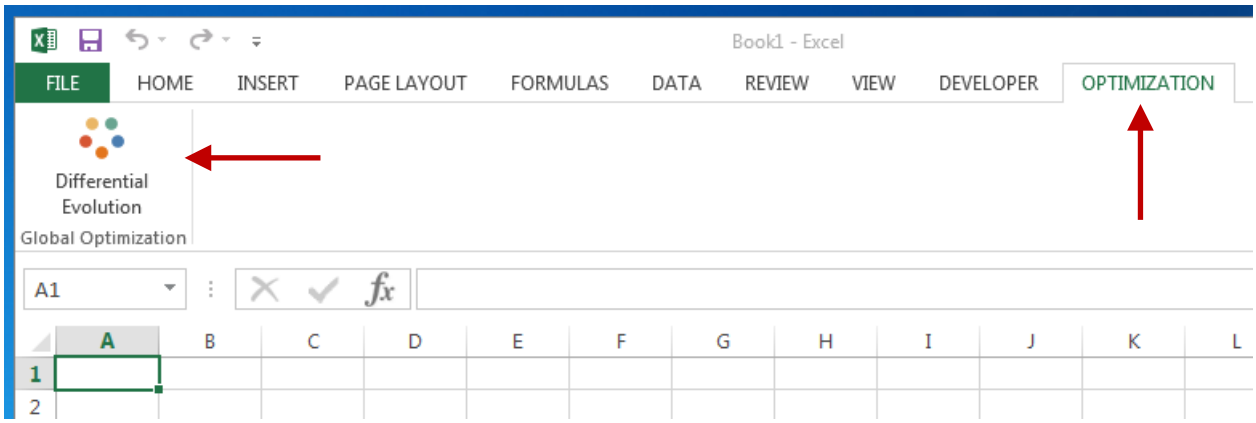


To load the add-in, in the **Add-Ins available** box, select the check box next to the **Differential Evolution** and then click **OK**.



Depending on your Excel installation, you may see a Security Warning. Click **"Enable Macros"** to finish the installation. If you do not see the Security Warning, this means that your Excel installation already allows macros and installation will complete.

The Add-In is now installed. You will see that the new "OPTIMIZATION" menu appeared with a button for Differential Evolution optimizer. The installation is complete.



Appendix E

Differential Evolution Application: Optimization of Customer Loyalty Program

Consider the following problem. A hotel chain with multiple property locations is designing a customer loyalty program. The objective is to incorporate the loyalty program into the overall revenue management scheme. The idea is to maximize profitability while taking into account the effects of the loyalty program and accounting for the costs of administering the loyalty program.[1]

The chain operates in M markets, indexed by $m = 1, \cdots, M$. The loyalty program allows participants to accumulate loyalty points (with purchases of rooms and other services) and to redeem loyalty points as payments for rooms and for other services.

The markets may be different in terms of the elasticity of demand with respect to prices and with respect to the ability to accumulate loyalty points with purchases and the ability to redeem them. For example, customers may be prone to accumulate points in the markets where they travel on business, and to redeem points in the markets where they travel on vacation. In general, demand for rooms in market m, $\mathcal{D}R_m$, is a function of: room prices; prices of other services; price *elasticity* with respect to room and services prices; ability to earn loyalty points from room purchases and purchases of other services; *elasticity* with respect to the ability to earn points; ability to redeem loyalty points as room payments and as payments for other services, and *elasticity* with respect to the ability to redeem points as room payments and payments for other services.

$$\mathcal{D}R_m = \mathbb{D}R(pR_m, pS_m, ERpR_m, ERpS_m, \pi R_m, \pi S_m, ER\pi R_m, ER\pi S_m, \rho R_m, \rho S_m, ER\rho R_m, ER\rho S_m)$$

Similarly to room demand, the demand for other services in market m, $\mathcal{D}S_m$, is a function of prices, price elasticity, ability to earn loyalty points (and corresponding elasticity), and the ability to redeem loyalty points (and corresponding elasticity).

$$\mathcal{D}S_m = \mathbb{D}S(pR_m, pS_m, ESpR_m, ESpS_m, \pi R_m, \pi S_m, ES\pi R_m, ES\pi S_m, \rho R_m, \rho S_m, ES\rho R_m, ES\rho S_m)$$

Profitability[2] from room sales in market m, $\mathcal{P}R_m$, is a function of room demand, $\mathcal{D}R_m$, fixed costs, $\mathcal{F}R_m$, and variable costs, $\mathcal{V}R_m$:

$$\mathcal{P}R_m = \mathbb{P}R(\mathcal{D}R_m, \mathcal{F}R_m, \mathcal{V}R_m)$$

The profitability is measured in terms of dollars and thus would not include payments made by customers with the loyalty points. Still, profitability will depend on the propensity of customers to use loyalty points for payment, to the extent that such payments substitute currency payments. Also, profitability depends on the loyalty program parameters through the demand variable.

[1] A caveat: the objective of this discussion is not to attempt to build the most comprehensive revenue management model, but to illustrate the method, so the forgoing discussion should be judged in this light. What follows is a fairly general formulation of the problem.

[2] Profitability is based on an estimated model of the firm and estimated supply and demand in the market. The model can also incorporate such boundary conditions as 100% occupancy when price is zero, and other, more realistic, conditions.

Profitability from other services in market m, $\mathcal{PS}_m$, is a function of demand, $\mathcal{DS}_m$, fixed costs associated with other services, $\mathcal{FS}_m$, and variable costs, $\mathcal{VS}_m$, associated with other services:

$$\mathcal{PS}_m = \mathbb{PS}(\mathcal{DS}_m, \mathcal{FS}_m, \mathcal{VS}_m)$$

Overall profitability is the sum across all markets of market room and service profitability, minus the total cost of administration of the loyalty program:

$$Total\mathcal{P} = \sum_{m=1}^{M} (\mathcal{PR}_m + \mathcal{PS}_m) - Cost\mathcal{L}$$

Let

$$\boldsymbol{\beta}_m = (ERpR_m, ERpS_m, ESpR_m, ESpS_m,$$
$$ER\pi R_m, ER\pi S_m, ES\pi R_m, ES\pi S_m, ER\rho R_m, ER\rho S_m, ES\rho R_m, ES\rho S_m)$$

be the vector of parameters that describe consumer behavior in market m. The description of each variable is given in **Exhibit E2**.

Let $\boldsymbol{\theta}_m = (pR_m, pS_m, \pi R_m, \pi S_m, \rho R_m, \rho S_m)$ be the vector of choice variables for market m. The description of the variables is given in **Exhibit E1**.

Cost of the loyalty program administration depends on the choice variables. The ability to earn (and redeem) loyalty points affects customer participation rates. There are various administrative costs associated with participation: Customer service, program-related mailings and distribution, loyalty cards, database management, etc. To the extent that these costs are a function of customer participation, and participation is affected by the loyalty program parameters, we have:

$$Cost\mathcal{L} = \mathbb{CL}(\cdot; \boldsymbol{\beta}_1, \cdots, \boldsymbol{\beta}_M; \boldsymbol{\theta}_m, \cdots, \boldsymbol{\theta}_M)$$

The optimization problem then becomes: given the parameters that describe consumer behavior, $\boldsymbol{\beta}_m$; given the profitability functions $\mathcal{PR}_m = \mathbb{PR}(\cdot; \boldsymbol{\beta}_m; \boldsymbol{\theta}_m)$ and $\mathcal{PS}_m = \mathbb{PS}(\cdot; \boldsymbol{\beta}_m; \boldsymbol{\theta}_m)$; and given the cost function for the loyalty program, $Cost\mathcal{L} = \mathbb{CL}(\cdot; \boldsymbol{\beta}_m; \boldsymbol{\theta}_m)$, maximize total profitability with respect to the choice variables $\boldsymbol{\theta}_m$ across all markets, $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_M$. The optimization problem is then:

$$\max_{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_M} Total\mathcal{P} = \max_{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_M} \sum_{m=1}^{M} \left(\mathbb{PR}(\cdot; \boldsymbol{\beta}_m; \boldsymbol{\theta}_m) + \mathbb{PS}(\cdot; \boldsymbol{\beta}_m; \boldsymbol{\theta}_m)\right) - Cost\mathcal{L}$$

Constraints may also be imposed. For example, constraints that prices must be positive, as well as non-negative loyalty point accumulation and redemption parameters.

Clearly, for a large chain operating in many markets this is an optimization problem of high dimension. The objective function has many parameters and variables and may be analytically cumbersome to track, requiring numerical methods. A robust global optimization tool is useful here.[3]

[3] Two caveats. First, to implement the model, elasticity parameters need to be estimated from data on customer behavior. Second, the model as describes is a steady-state model. It does not include a time dimension, describing the

Exhibit E1. Choice variables for market m, $\boldsymbol{\theta}_m$.

pR_m	Pricing for Rooms in market m.
pS_m	Pricing for Other Services in market m.
πR_m	Ability to earn loyalty points from room (R) purchases in market m (a customer may earn 1 point for every $1 spent on the room rate, or in select markets there may be a special promotion where a customer may earn 2 points for every $1 spent on the room rate).
πS_m	Ability to earn loyalty points from purchases of other services (S) in market m (a customer may earn 2 points for every $1 spent at a spa, for example, or in select markets there may be a special promotion of 3 points for every $1 spent at the spa).
ρR_m	Ability to redeem loyalty points as payment for rooms in market m.
ρS_m	Ability to redeem loyalty points as payment for other services in market m.

Exhibit E2. Parameters that describe consumer behavior in market m, $\boldsymbol{\beta}_m$.

$ERpR_m$	Elasticity of room demand with respect to room prices in market m.
$ERpS_m$	Elasticity of room demand with respect to pricing of other services in market m.
$ER\pi R_m$	Elasticity of room demand with respect to ability to earn loyalty points from room payments.
$ER\pi S_m$	Elasticity of room demand with respect to ability to earn loyalty points from payments for other services.
$ER\rho R_m$	Elasticity of room demand with respect to ability to redeem loyalty points as room payments.
$ER\rho S_m$	Elasticity of room demand with respect to ability to redeem loyalty points as payments for other services.
$ESpR_m$	Elasticity of other services demand with respect to room prices in market m.
$ESpS_m$	Elasticity of other services demand with respect to pricing of other services in market m.
$ES\pi R_m$	Elasticity of other services demand with respect to ability to earn loyalty points from room payments.
$ES\pi S_m$	Elasticity of other services demand with respect to ability to earn loyalty points from payments for other services.
$ES\rho R_m$	Elasticity of other services demand with respect to ability to redeem loyalty points as room payments.
$ES\rho S_m$	Elasticity of other services demand with respect to ability to redeem loyalty points as payments for other services.

system in steady state. Thus, the model does not explicitly discuss expiration of loyalty points over time. This feature can be incorporated into the model in several ways (for example, as a steady-state propensity to use points). Again, the point of this appendix is to give an example of a type of problem where a robust global optimizer is needed to solve a business problem. The objective here is not to develop the most comprehensive model, but to outline a potential application of the tool.

Appendix F

Frequently Asked Questions

The calculation is taking a long time: Can I stop it?

The criteria for stopping the calculations is the maximum number of generations. You specify this number before running the algorithm. You can interrupt the calculations at any time by pressing ESC (twice) or Ctrl-Break. A user dialog will show, displaying the current generation of the maximum number (for example, "Generation: 930 of 1000") and the optimum value found so far. At this point you can resume calculations (by clicking "Yes") or stop the calculations (by clicking "No"). If you click "Yes" the calculations will resume at the point where they were interrupted, no work is lost. If you click "No" the current best solution will be given in the "Changing Cells." So, yes, you can stop the calculation at any time and you will retain the best solution found so far.

The calculation is taking a long time. Why is that?

It is true that generally we do not like to wait. Some things, however, are worth waiting for. Searching for a global optimum (a global minimum or a global maximum) is a difficult task for a computer. The search involves: (a) Selecting values for variables; (b) Computing the value of the function you are optimizing given these variable values; (c) Comparing the value found to the best-so-far solution; (d) Selecting new values for the variables. All these steps are performed in a loop. By selecting different values for the variables (different points) the computer is *exploring* the function you are optimizing. Depending on how many variables are in the problem and how complex the function is, the exploration can take a fairly long time. If your objective function involves a lot of computations on the Worksheet, this will take longer. Keep in mind, that thorough exploration (visiting many points) is a key to finding the global optimum.

How do I know the status of the calculation?

The tool provides updates on Excel Status Bar (look at the lower left corner of Excel window). The status bar displays a message, such as "Optimizing (35% done). Please be patient..." Updates at the status bar are provided at 1%, 5%, 10%, ... (at 5% increments)... 90%, 95%, 99%. You can also interrupt the computations at any time by pressing ESC (twice) or Ctrl-Break. A dialog box will appear that displays the current iteration, the total computing time until now, the expected time until completion, and the optimal (minimum or maximum) value found so far. The user is given an option to continue calculations (the computations will proceed without losing any information) or to stop computations. If the calculations is stopped, then the best solution so far is retained in the worksheet.

What are some limitations of the Differential Evolution algorithm?

The is no universal optimization algorithm that is guaranteed to find global optimum for any problem that is given. DE has limitations. One limitation is the problem where the variables take *integer* values. Actually, there is a special area in Operations Research that studies algorithms specifically designed to work with problems

where the variables take integer values (this area is called *integer programming*). DE is not such a method. It has no specialized capabilities to work through such problems. However, this does not mean that DE will never find the optimum. This means that DE is likely to be not the best method to solve an integer programming problem. If you have an integer programming problem your best route is to use a specialized algorithm, especially if the problem is of high dimension (you have, say 100 variables).

If I wanted to use DE to solve an integer programming problem, how should I set it up?

You have an objective function (the function that you are minimizing or maximizing) that depends on variables that take *integer* values. An example could be the number of customer service stations, or the number of reservations at a restaurant. Let $f(x_1, \cdots, x_k, z_1, \cdots z_m)$ be the objective function and let $x_1, \cdots, x_k$ be the continuous variables and let $z_1, \cdots z_m$ be the integer variables. You should set up your spreadsheet as follows.

1. Set up an area for the *changing cells* (one continuous area). These are the cells that the program will be changing while searching for the solution. For example, let cells A1 through A5 represent the continuous variables $x_1, \cdots, x_5$ and let the cells A6 through A12 represent the integer variables $z_1, \cdots z_7$.

2. Set up an area for the variables of the objective functions. This area should be separate from the changing cells variables. These cells will be referenced by your objective function: to compute the value of the objective function you will refer to the cells in this area, not in the changing cells area. For example, let the cells B1 through B12 represent this area.

3. Now you need to set up the references between the area that has the variables for the objective function (in our example, cells B1 through B12) and the changing cells (in our example, cells A1 through A12). The cells that represent continuous variables are referenced directly, without any changes. In this example, in the cell B1 you enter the formula "=A1", in the cell B2 you enter the formula "=A2", and continue this way until B5 (the formula in the cell B5 is "=A5"). The cells that represent integer variables need a different formula. When working, the algorithm will be trying different continuous values in the changing cells, but the objective function requires the variables $z_1, \cdots z_m$ to be integer. For these variables, you set up the spreadsheet to take a continuous variable from the changing cells area and turn it into an integer. You use Excel function **INT()** to do this. The formula for B6 is "=INT(A6)", for B7 "=INT(A7)" and so forth until B12 contains "=INT(A12)".

4. Set the region in the spreadsheet for the **low bounds** of the variables. If the lowest acceptable value for an integer variable is negative, say $-n$, then the low bound you set should be that value, $-n$. For example, if the lowest acceptable value for an integer variable is -11, then the low bound for that variable should be set as -11. If the lowest acceptable value for an integer variable is positive, say n, then the lower bound you set should be that value, n. For example, if the lowest acceptable integer value is 14, then the low bound should be set as 14. If the lowest acceptable integer value is zero, then set the low bound at zero.

5. Set the region in the spreadsheet for the **upper bounds** of the variables. If the largest acceptable value for an integer variable is negative, say $-n$, then the upper bound you set should be just below the negative integer that is immediately larger than $-n$; a good choice would be $(-n + 1).01$. For example, if the largest acceptable integer value is -17, set the upper bound to -16.01. The Excel function =INT(x) will take negative non-integer values and convert them to an integer less than or equal to x, so when the integer value is passed

to the objective function it will be within the acceptable bound. For example, =INT(-16.1) returns the value of -17. If the largest acceptable integer value is zero, then set the upper bound at 0.999. If the largest acceptable integer value is positive, say n, then the upper bound you set should be just below the positive integer that is immediately larger than n. For example, if the largest acceptable integer value is 18, set the upper bound to 18.99. The Excel function =INT(x) will take non-integer values close to the upper bound and convert them to an integer that is less than or equal to x, so when the integer value is passed to the objective function it will be within the acceptable bound. For example, =INT(18.99) returns the value of 18.

6. Run the tool. The tool will be changing values in the changing cells are, which in turn will be converted by the =INT() function in the area that is referenced by the objective function calculation.

What if the variables of the objective function take discrete (not continuous) values?

There are cases when the objective function depends on one or several *discrete variables*. A discrete variable Z takes its values from a discrete set $\{z_1, z_2, \cdots, z_{k-1}, z_k\}$. For example, a variable can take the values from the set $\{-7.25, -1, 0, 0.01, 10, 12, 17.85\}$. The set of values can always be ordered from small to large, so that $z_i < z_{i+1}$. The idea is to set up the spreadsheet so that during optimization the tool varies the *indexes* i of the discrete values. For the evaluation of the objective function the discrete value itself is looked up based on the value of the index and is passed on to the objective function for the evaluation. The set up of the spreadsheet is described below.

Set up the area for the *changing cells*. The tool will be varying the values of these cells and will treat them as continuous variables (they can take any value between the set low and upper bounds). Set up another area where the values from the changing cells are converted into integer indexes for the discrete values (this area will use =INT() function to convert value from a changing cell into an integer). Set up an area where the integer indexes are turned into values of discrete variables. In this area, an integer index for a variable is used to look up a value from the discrete set of values for that variable. This looked up value is then used in the objective function evaluation.

Set up low bounds area and upper bounds are. These areas are needed to keep the indexes within acceptable range. The low bound for all indexes is 1, so the low bound area is populated by 1's. For a variable Z that can take k discrete values, $1, \cdots, k$, the upper bound is a number just below the integer that is immediately larger than k. For example, if a variable can take 8 discrete values, the upper bound can be set as 8.999 (this will work because =INT(8.999) = 8).

At this point you can run the tool.

Why should I try this tool?

This tool is not a replacement for Solver. This tool is a complement, an *addition*, to your Excel tool set. You should try this tool if the function you are optimizing has many local optima (visualize the crate that is used to hold a dozen of eggs in the supermarket, and imagine that the peaks have slightly different heights and the valleys have slightly different depths). You should try this tool if you are not satisfied with the solutions you

are obtaining with Solver or if you would like to check the robustness of Solver solutions to your model. In many cases solving an optimization problem is a process that requires a lot of trial and error. Having access to several tools for this process is beneficial.

Where can I read more on Differential Evolution?

If you are interested in learning technical details of the algorithm there are two excellent books.

1. Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Springer-Verlag, 2005.

2. Vitaliy Feoktistov, *Differential Evolution: In Search of Solutions*, Springer, 2006.

These books also contain numerous references to scientific articles on the relevant topics. The first book contains detailed discussion of the application of DE to different problems in science, mathematics, and engineering. The books also contain detailed discussions on the performance of the DE algorithm relative to other numerical optimization methods. In addition to these books, and interested reader may consult Appendix B of this report where the details of the DE algorithm are discussed.

What systems have the tool been tested on?

The tool is written in Excel Visual Basic. The tool is written with the maximum compatibility in mind. There are no system requirements in addition to those for Microsoft Office Excel. The tool has been extensively tested in Microsoft Office Excel 2007, Microsoft Office Excel 2010, and Microsoft Office Excel 2013, under both 32-bit and 64-bit Windows operating systems. The tool was not tested on Apple computers running Excel.

A Useful Tip 1

Before running the tool (before clicking on the Excel ribbon to show the tool's user interface), make the cell where you have the objective function to be the *active cell* in Excel. When the user interfaces starts, the tool identifies the cell that is currently selected in Excel and puts the address of this cell into the Target Cell field of the user interface.

I do not Know What Bounds to Set for the Variables. What Should I do?

The bounds for the variables are determined by the nature of the problem being solved. The researcher or the analyst working on the problem is the most knowledgeable person to specify the bounds on variables. For example, a natural low bound on the number of restaurant reservations for a given time slot is zero, because

user. If there are no natural bounds, the user can specify a very wide range. All the variables are implemented as the **Double** data type. **Double** (double-precision floating-point) variables are stored as IEEE 64-bit (8-byte) floating-point numbers ranging in value from **-1.79769313486231E308** to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to **1.79769313486232E308** for positive values. User can set the bounds to be close to the smallest valid number for the low bound and to the largest valid number for the High (Upper) bound. Of course, the larger the search area (the wider are the bounds), the harder it is to find global optimum.

Can the Tool be used to Solve a Constrained Optimization Problem?

As programmed, the tool handles boundary constraints. However, the spreadsheet model can be build by a user to handle general constraints using the penalty function method. Below is the description of how to do this.

Let $f(x_1, \cdots, x_n) = f(\boldsymbol{x})$ be the objective function (the function you are optimizing) and let $\boldsymbol{x} = (x_1, \cdots, x_n)$ be the variables. *Boundary constraints* represent low and high limits on the values that each individual variable can take:

$$L_i \leq x_i \leq H_i, \qquad i = 1, \cdots, n.$$

Constraint functions represent general constraints placed on the variables, $g_k(x_1, \cdots, x_n) \leq 0, k = 1, \cdots, m.$

In the general form, a constrained optimization problem is written as

$$\text{find } \boldsymbol{x}^*: f(\boldsymbol{x}^*) = \min_{(x_1, \cdots, x_n)} f(\boldsymbol{x})$$

Subject to

$$\text{boundary constraints } L_i \leq x_i \leq H_i, \qquad i = 1, \cdots, n$$

$$\text{constraint functions } g_k(x_1, \cdots, x_n) \leq 0, \qquad k = 1, \cdots, m.$$

The tool handles boundary constraints that are specified by the user.

Constraint functions can also be handled, but user needs to build this into the spreadsheet. One method, called *penalty function method*, for doing so is described below. The idea is to set up a cell that adds up all constraint violations and add this value to the objective function in the case of a minimization problem, or subtract this value from the objective function in the case of a maximization problem.

1. For each constraint function, $g_k(x_1, \cdots, x_n) \leq 0$, set up a cell that computes the value of the constraint function. If the constraint is met, the value in this cell will be negative or zero. If the constraint is violated the value of this cell will be positive. For example, let the cell that contains constraint function $g_1(x_1, \cdots, x_n)$ be the cell **D7**. Set up another cell that returns =Max(ConstraintCell, 0). This cell will contain constraint violation vale (if the constrain is violated) and zero if the constraint is met. For example, the cell **E7** will contain the formula =Max(D7, 0).

2. Repeat step (1) above for all constraint functions.

3. Set up a cell to compute weighted sum of constraint violations. The weights are positive numbers, $w_k > 0$, for all k.

$$G(\boldsymbol{x}) = w_1 \cdot Max[g_1(\boldsymbol{x}), 0] + \cdots + w_k \cdot Max[g_k(\boldsymbol{x}), 0]$$

A simple choice of weights to start with is to keep them all equal to one, $w_k = 1.0$, for all k. The weights determine the relative importance of violations of different constraints.

4. For a Minimization Problem: Set up new Target Cell. The cell needs to take the objective function (the function that you are minimizing), $f(\boldsymbol{x})$, and add to it the weighted sum of constraint violations multiplied by a positive scaling constant, $s > 0$. The Target Cell should contain:

$$f(\boldsymbol{x}) + s \cdot G(\boldsymbol{x})$$

The scaling constant should be a number sufficiently large so that it penalizes the function $f(\boldsymbol{x})$. For example, if the expected value at the minimum of the objective function $f(\boldsymbol{x})$ is approximately in the -5,000 to -6,000 range, the scaling constant s may be set at +10,000 or even +50,000.

5. For a Maximization Problem: Set up new Target Cell. The cell needs to take the objective function (the function that you are maximizing), $f(\boldsymbol{x})$, and add to it the weighted sum of constraint violations multiplied by a negative scaling constant, $s < 0$. The Target Cell should contain:

$$f(\boldsymbol{x}) + s \cdot G(\boldsymbol{x})$$

The scaling constant should be a number sufficiently large in absolute value so that it penalizes the function $f(\boldsymbol{x})$. For example, if the expected value at the maximum of the objective function $f(\boldsymbol{x})$ is approximately in the 5,000 to 6,000 range, the scaling constant s may be set at -10,000 or even -50,000.

6. At this point you can run the tool to optimize the Target Cell set up in (4) or (5) above.

Center for Hospitality Research
Publication Index
chr.cornell.edu

2016 Reports

Vol. 16 No. 23 Short-term Trading in Long-term Funds: Implications for Hospitality Financial Managers, by Pamela C. Moulton, Ph.D.

Vol. 16 No. 22 The Influence of Table Top Technology in Full-service Restaurants, by Alex M. Susskind, Ph.D., and Benjamin Curry, Ph.D.

Vol. 16 No. 21 FRESH: A Food-service Sustainability Rating for Hospitality Sector Events, by Sanaa I. Pirani, Ph.D., Hassan A. Arafat, Ph.D., and Gary M. Thompson, Ph.D.

Vol. 16 No. 20 Instructions for the Early Bird & Night Owl Evaluation Tool (EBNOET) v2015, by Gary M. Thompson, Ph.D.

Vol. 16 No. 19 Experimental Evidence that Retaliation Claims Are Unlike Other Employment Discrimination Claims, by David Sherwyn, J.D., and Zev J. Eigen, J.D.

Vol. 16 No. 18 CIHLER Roundtable: Dealing with Shifting Labor Employment Sands, by David Sherwyn, J.D.

Vol. 16 No. 17 Highlights from the 2016 Sustainable and Social Entrepreneurship Enterprises Roundtable, by Jeanne Varney

Vol. 16 No. 16 Hotel Sustainability Benchmarking Index 2016: Energy, Water, and Carbon, by Eric Ricaurte

Vol. 16 No. 15 Hotel Profit Implications from Rising Wages and Inflation in the U.S., by Jack Corgel, Ph.D.

Vol. 16 No. 14 The Business Case for (and Against) Restaurant Tipping, by Michael Lynn, Ph.D.

Vol. 16 No. 13 The Changing Relationship between Supervisors and Subordinates: How Managing This Relationship Evolves over Time, by Michael Sturman, Ph.D. and Sanghee Park, Ph.D.

Vol. 16 No. 12 Environmental Implications of Hotel Growth in China: Integrating Sustainability with Hotel Development, by Gert Noordzy, Eric Ricaurte, Georgette James, and Meng Wu

Vol. 16 No. 11 The International Hotel Management Agreement: Origins, Evolution, and Status, by Michael Evanoff

Vol. 16 No. 10 Performance Impact of Socially Engaging with Consumers, by Chris Anderson, Ph.D., and Saram Han

Vol. 16 No. 9 Fitting Restaurant Service Style to Brand Image for Greater Customer Satisfaction, by Michael Giebelhausen, Ph.D., Evelyn Chan, and Nancy J. Sirianni, Ph.D.

Vol. 16 No. 8 Revenue Management in Restaurants: Unbundling Pricing for Reservations from the Core Service, by Sheryl Kimes, Ph.D., and Jochen Wirtz, Ph.D.

Vol. 16 No. 7 Instructions for the Food Preparation Scheduling Tool v2015, by Gary Thompson, Ph.D.

Vol. 16 No. 6 Compendium 2016

Vol. 16 No. 5 Executive Insights on Leader Integrity: The Credibility Challenge, by Tony Simons, Ph.D., with Kurt Schnaubelt, John Longstreet, Michele Sarkisian, Heather Allen, and Charles Feltman

Vol. 16 No. 4 Authenticity in Scaling the Vision: Defining Boundaries in the Food and Beverage Entrepreneurship Development Cycle, by Mona Anita K. Olsen, Ph.D., and Cheryl Stanley

Vol. 16 No. 3 Communication Planning: A Template for Organizational Change, by Amy Newman

Vol. 16 No. 2 What Guests Really Think of Your Hotel: Text Analytics of Online Customer Reviews, by Hyun Jeong "Spring" Han, Ph.D., Shawn Mankad, Ph.D., Nagesh Gavirneni, Ph.D., and Rohit Verma, Ph.D.

Vol. 16 No. 1 The Role of Service Improvisation in Improving Hotel Customer Satisfaction, by Enrico Secchi, Ph.D., Aleda Roth, Ph.D., and Rohit Verma, Ph.D.

CREF Cornell Hotel Indices

Vol. 5 No. 3 Second Quarter 2016: Slowdown for Large Hotels Continues: Small Hotels Have Now Slowed as Well, by Crocker Liu, Ph.D., Adam D. Novak, Ph.D., and Robert M. White, Jr.

Vol. 5 No. 2 First Quarter 2016: Second Verse, Same as the First, by Crocker Liu, Ph.D., Adam D. Novak, Ph.D., and Robert M. White, Jr.

2015 Reports

Vol. 15 No. 22 Have Minimum Wage Increases Hurt the Restaurant Industry? The Evidence Says No!, by Michael Lynn, Ph.D., and Christopher Boone, Ph.D.

Vol. 15 No. 21 Hotel Brand Conversions: What Works and What Doesn't, by Chekitan S. Dev, Ph.D.

CHR Advisory Board

Cornell Hospitality Report
Vol. 16, No. 24 (October 2016)

Cornell Hospitality Report is produced for the benefit of the hospitality industry by The Center for Hospitality Research at Cornell University.

Christopher K. Anderson, Director
Carol Zhe, Program Manager
Glenn Withiam, Executive Editor
Kate Walsh, Acting Dean, School of Hotel Administration

Center for Hospitality Research
Cornell University
School of Hotel Administration
389 Statler Hall
Ithaca, NY 14853

607-254-4504
chr.cornell.edu