**X-TOOLSS User’s Manual for Version 1.3**

Updated April 18, 2011

**User’s Manual to**

****

**NASA’s Evolutionary Computation Based Design Optimization Toolset**

**Currently being developed by:**

North Carolina Agricultural and Technical State University (NCAT)

Department of Computer Science

NASA Marshall Space Flight Center

ES22 – Life Support, Thermal, & Mechanical Analysis Branch

**NCAT Points of Contact:**

Dr. Gerry Vernon Dozier

Professor & Chair of the Department of Computer Science   
[gvdozier@ncat.edu](mailto:gvdozier@ncat.edu)

Josh Adams

**NASA Points of Contact:**

Dr. Mike Tinker

ES22 Branch Chief and X-TOOLSS Principle Investigator

[mike.tinker@nasa.gov](mailto:mike.tinker@nasa.gov)

Ben DiMiero

X-TOOLSS Researcher

[benjamin.m.dimiero@nasa.gov](mailto:benjamin.m.dimiero@nasa.gov)

**Table of Contents**

PART 1 Introduction to X-TOOLSS 4

History 4

Purpose 4

Applications 5

Demonstration of the possibilities of ECs 5

PART 2: Installation instructions 6

PART 3: X-TOOLSS File Requirements and Application 8

The X-TOOLSS Compatible Simulation Structure: 9

Application components: 13

The Module File Description and Configuration Requirements 14

The Input File Description and Configuration Requirements 21

The Output File Description and Configuration Requirements 21

The Code File Description and Configuration Requirements 22

PART 4: The X-TOOLSS Graphical User Interface 24

Optimization Wizard – X-TOOLSS Application Builder 25

Module File Verification Window 28

Module Execution Window 29

X-TOOLSS Results Window 32

X-TOOLSS Convergence Graph 34

Additional Windows and Screens 40

PART 5 Evolution Results 41

xtoolss.log file: 41

xtoolss.out file: 42

xtoolss.stat file: 42

PART 6: Supported GECs and Parameters 43

1 Generation GA with BLX 43

2 Steady-state GA 44

3 Steady-state GA with BLX 45

4 Steady-generational GA with BLX 46

5 PSO 47

6 Generational DEA 48

7 Steady-state DEA 49

8 Elitist DEA 50

9 Standard EP 51

10 Continuous Standard EP 52

11 Meta-EP 53

12 Continuous Meta-EP 54

PART 7: An X-TOOLSS Example Problem 55

PART 8: Future Anticipated Development 69

Glossary 70

References 71

# PART 1 Introduction to X-TOOLSS

X-TOOLSS is a genetic algorithm based evolutionary computation (EC) toolset that is currently being developed within NASA MSFC/ES22 in conjunction with North Carolina A&T University. This tool makes use of algorithms that, based on the “survival of the fitness” concept, develop optimal solutions for prospective designs by eliminating weak candidate designs. The power of EC techniques lies in their ability to discover unique, innovative, and often non-intuitive designs by thoroughly interrogating the entire design space. Furthermore, ECs drive the whole system to an optimum in a way that no other method can. This unique optimization technique will allow engineers produce superior designs in a more efficient manner.

## History

The X-TOOLSS software package includes, to date, twelve evolutionary computing algorithms that can be used by scientists and engineers to solve optimization problems. The X-TOOLSS algorithms are the result of a project history beginning with NEVOT (Nuclear Exploration Vehicle Optimization Team) and continuing today through the efforts of Dr. Gerry Dozier, his students at North Carolina A&T State University, and the personnel at Marshall Space Flight Center, Oak Ridge National Laboratory, and Arnold Engineering Development Center.

## Purpose

The traditional design process used by engineers today is often tedious, repetitive, and inefficient. Most space system designs cannot be considered truly optimal systems, since they are developed iteratively but without the benefit of optimization technology. Typical techniques that engineers use to investigate a design or develop a process consist of performing trade studies or parametric analyses, which tend to be time consuming iterative processes. X-TOOLSS has the ability to improve the current design process by reducing the required manpower and by automating the iterative design process. Allowing X-TOOLSS to perform automated parametric analyses enables engineers to be more productive and efficient, ultimately producing superior designs with significantly less effort.  These benefits, in turn, reduce the engineering costs associated with the overall design process.

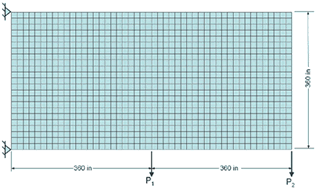
When developing complex designs, the impact of increasing the fidelity of one physical parameter is not realized as well by any one engineer compared to the coupled analysis of X-TOOLSS. As an example, when designing a spacecraft, one engineer will want to make it as powerful as possible, another as lightweight as possible, another as structurally sound as possible, and they will all want to make the project as cost effective as possible. This is typically achieved after performing trade studied and numerous redesigns to sufficiently suit the needs of every physical requirement. The evolutionary computation techniques of X-TOOLSS have the ability to search the entire design space to find a valid design that will suit each physical requirement, while having a minimal impact on the other physical parameters. In this case, X-TOOLSS will search for the spacecraft design that has the maximum available propulsive thrust and structural integrity while having minimal weight and cost.

## Applications

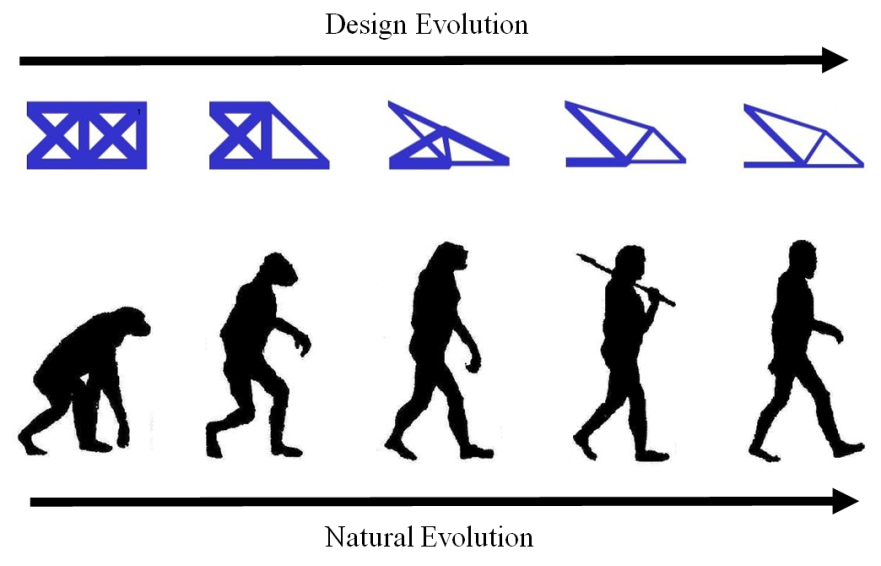
The current optimization process requires the user to write a customized code, specific to the design, that either directly performs the simulation or provides a framework for developing a running simulation model. *This software package can, as it stands, optimize virtually anything that can be modeled using a mathematical representation*. The confining problem with this process is that not everyone has the time or the skills to write custom codes to model their prospective designs, especially when dealing with highly complex systems. A few techniques have been developed that allow engineers to solve generic types of problems, without the need to write a custom code to do so.

## Demonstration of the possibilities of ECs

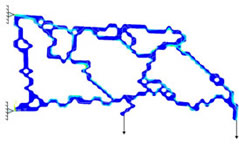
**Example Problem:** Find an optimal truss design, using evolutionary computations, within a given design region that has minimal mass, stress and deflection. The solution must be a single piece in a fabrication ready format.



**Figure 1:** (Left) The predefined design region for an optimal truss design.

****

**Figure 2:** (Right) An analogy between evolutionary computation and natural selection to demonstrate this unique optimization process.

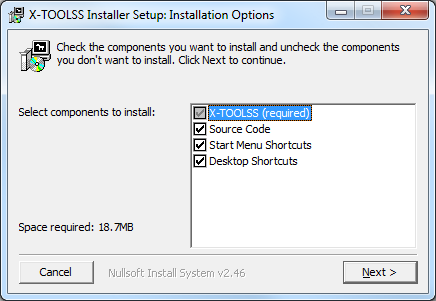
**Figure 3:** (Left) The innovative and unconventional, yet optimal truss design as determined by evolutionary computations

# PART 2: Installation instructions

The current version of X-TOOLSS, Version 1.3, is available for download via the following website:

<http://nxt.ncat.edu/>

To install X-TOOLSS, select the link to a version to download and save the installer to the computer. When you run the installer, you will get a window similar to Figure 4. This screen allows you to select optional components such as source code, start menu and desktop shortcuts. Select the options you would like and click the next button to continue the installation.

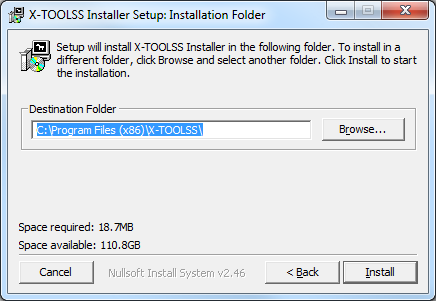


**Figure 4**: X-TOOLSS installation window.

By default all options are checked. The only required option is the X-TOOLSS application. The other options provide the following features:

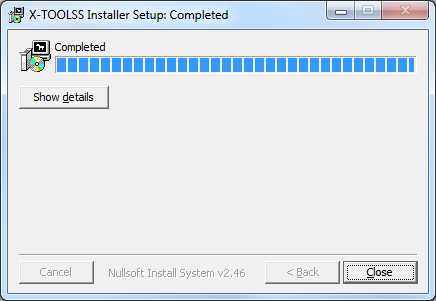
* Source Code – Installs a zip file in the installation directory containing all of the source code required to compile and run X-TOOLSS.
* Start Menu Shortcuts – Provides links in the start menu to the X-TOOLSS application. It also provides a link to this Users Manual.
* Desktop Shortcuts – Provides shortcuts on the user’s desktop to the X-TOOLSS application and to the Users Manual.

After selecting desired options and clicking next, you will be prompted to select the installation directory (Figure 5). Any directory will work fine though the default directory works the best for most installations. Verify the destination folder and click the install button to continue.



**Figure 5**: This window allows manual selection of the X-TOOLSS destination folder.

After clicking install, you will be shown the progress of the installation. This window, Figure 6, allows you to view details of the installation. Once the installation completes, click close.



**Figure 6**: Installation progress window.

Once the installation is complete and you close the installer, you can now start the application. The executable will be located in the directory you selected during installation. If you selected desktop or start menu icons, there will be shortcuts available from those locations.

# PART 3: X-TOOLSS File Requirements and Application

X-TOOLSS itself is a “black box” software package that includes a simple GUI that contains the 12 different EC algorithms. The software acts as a solver that, similar to many other numerical solvers, requires a few specific files in order to operate properly. The simplicity of this software package allows for the maximum amount of flexibility and the ability to wrap any type of numerically modeled design into the optimization process.

Unlike many commercial solvers, X-TOOLSS does not have a specific pre and post processors that can be used to set up jobs for submission to X-TOOLSS. Therefore each of the files required to run an optimization with X-TOOLSS are typically written by hand. Depending on the problem this could be either a very simple or difficult task, fortunately a variety of tutorials have been developed that explain how to easily set up and optimize models built in popular commercial design and analysis software with X-TOOLSS. The pool of analysis tools and applications for optimization problems is too large and diverse to tie to X-TOOLSS universally, which is why these individual tutorials have been developed to demonstrate X-TOOLSS’s capabilities with commonly used analysis tools. If a tutorial does not exist that integrates designs built with your modeling tool of choice, contact the current X-TOOLSS support staff on how to integrate it. Contact information can be found at <http://nxt.ncat.edu/contact.html>.

As stated above, X-TOOLSS has the ability optimize any design as long as it is modeled with a numerical representation. Any specific design optimization needs to first be structured in such a way to be compatible with X-TOOLSS iterative optimization process. A minimum of two different files are required to set up an optimization properly with X-TOOLSS; these files will be described in detail in the “Application Components” section below. The work horse of this process is given the name the “code file”, which is either a java script or executable application that acts as a framework to build and simulate every step required to analyze any given design. This file can be as simple as a compiled script that does literally anything you program it to do. The requirement to design a custom java or executable is the achilles heelof the process, since not every engineer has the programmer experience to manually develop an algorithm to model some desired desire, but this is also what makes X-TOOLSS so incredibly flexible. Again, a variety of the X-TOOLSS tutorials explain simple techniques to allow an engineer without a lot of programming experience to set up very complex optimization models.

Optimization is an iterative process, and in the most generic sense is a means to automate the process followed manually by engineers on every design. X-TOOLSS essentially uses the code file as a framework to build prospective designs that vary from one another by some predefined design variables, iterating across the entire design space. The user is able to define the aspects of a design that they would like to investigate and provide those variables with a range of values to simulate prospective designs with. From generation to generation X-TOOLSS will improve a population of prospective design candidates by eliminating the weaker models while continuously building and testing new candidate models in search for better designs. This process is computationally intensive, but completely automated.

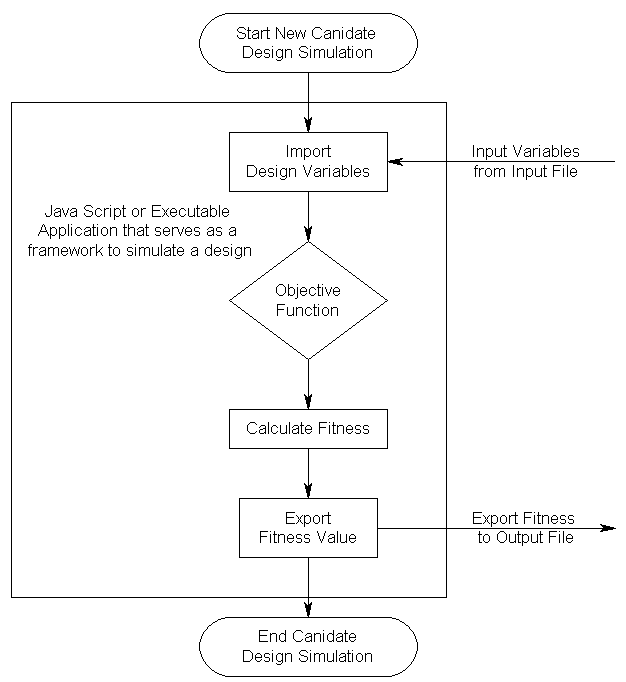
## The X-TOOLSS Compatible Simulation Structure:

The objective function is the foundation of any evolutionary computation. An objective function is essentially a mathematical representation of the model under question; it performs the sequential calculations in order to solve for a predetermined, desired result. Typically this is done with a custom code that models the physical system, in our case an X-TOOLSS code file acts as a framework that can be customized to perform the desired physical calculations and tasks to fully analyze a design.

The prerequisite for an objective function is a technique to vary problem specific design variables within the design’s simulation. These desired design variables can be varied between separate analyses to analyze separate but similar models; any individual model with its own unique design variables is known as a candidate model. X-TOOLSS compares one candidate model to another by a value, known as a fitness value, which is assigned to each candidate model. The fitness value is a measure of a model’s ability to suit the desired design requirements; furthermore the calculated fitness value itself is the objective of an X-TOOLSS objective function. This value is a user defined numerical output that is reduced to a single number that represents the quality of a candidate model.

X-TOOLSS searches for an optimal design based upon the fitness value assigned to any specific candidate design. Lower fitness values are considered to be better candidates than those with higher fitness values, essentially X-TOOLSS searches for a fitness value that approaches negative infinity. Fitness calculations should be catered to suit this characteristic. For example, when searching for the maximum value of something, it can be used as the fitness calculations by simply multiplying it by negative 1, which will make the fitness calculation compatible with X-TOOLSS. After the fitness value is calculated, X-TOOLSS requires that it be exported or written to an output file that can be read and analyzed by the genetic algorithms within X-TOOLSS.

X-TOOLSS requires that simulations be in the form of either a custom java scripts or standalone executable applications that directly simulates a design. Within these two simulation modes, X-TOOLSS requires that it be able to import data from an input file, perform the objective function, calculate a fitness value as a result of the simulation, and then export that value to an output file. Figure 7 below displays the required structure for any simulation to operate with X-TOOLSS, and Figure 8 below displays how this framework is integrated into an X-TOOLSS evolution.



**Figure 7:** A flowchart of the basic structure for any simulation to be X-TOOLSS compatible.

The custom java scripts or standalone executable applications are designed to allow X-TOOLSS to push and pull data from the objective function contained within the script. Both X-TOOLSS and the code file use files to communicate information. X-TOOLSS would first decide what variables it wants to run the model with, load those values into an input file (typically named “input.txt”), and the executable or script would then extract that information from the input file. The candidate model would run with the specific input variables and at the end of the analysis the value of the user-defined fitness function would be pushed into another .txt file (typically named “fitness.txt”). X-TOOLSS would then judge the ability for the respective candidate model to fit the desired solution based upon the fitness value and rank it against subsequent candidate models. This process essentially automates parametric analyses, eliminating the need for the engineer to build and retest multiple new models.



**Figure 8:** The process flow of how custom design simulations are integrated into the optimization process of X-TOOLSS.

## Application components:

There are two particular files that must be initially available to X-TOOLSS for any analysis to run: a module file and a code file. These two files must be in the same directory on the computer. The following sections explain each of these components, their configurations, and their roles. Figure 9 demonstrate how these files tie into the optimization process with X-TOOLSS.



**Figure 9:** This diagram shows how the module file is associated with X-TOOLSS.

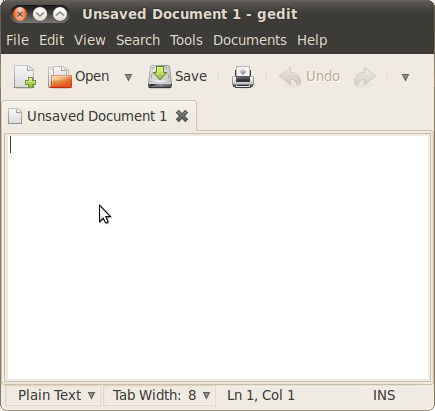
## The Module File Description and Configuration Requirements

The module file is the building block of an analysis for X-TOOLSS. It provides the critical instructions needed in order to perform an X-TOOLSS analysis. A module is simply a text file that has been saved with an .xts file extension. This file must include the following information:

* input variables
  + Input variables are the values the code file uses to calculate the fitness. For each iteration of the optimization, X-TOOLSS will calculate values for these variables and write them to the input file. Each input variable must have a variable name, range, and type.
* input file name
  + The input file name specifies the name of the file where X-TOOLSS writes the values of the input variables. This is also the file that the code file reads its input values from.
* code file name and type
  + The code file name and type specify the name and type of the executable file that reads in the input variables from the input file, calculates and writes the fitness to the output file.
* name of the variable output by the code file (fitness variable)
  + The output variable name specifies how X-TOOLSS describes the fitness of the candidate solution.
* output file name
  + The output file name specifies the name of the file where the code file writes the fitness values. This is also the file X-TOOLSS reads to associate a candidate solution to a fitness value.

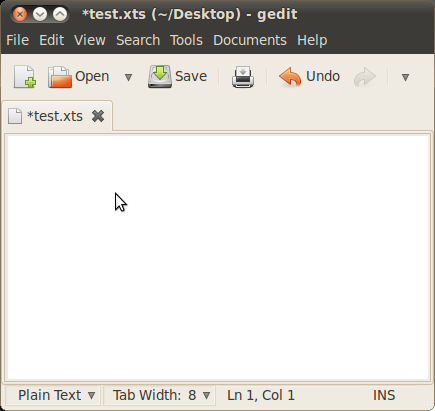
In the section below demonstrates how to properly create an xts file to be used for execution in an X-TOOLSS session.

**Step 1:**  To create an xts file, start simply by opening your systems text editor. An example of this is shown in figure 10.



**Figure 10**: An example of a text editor.

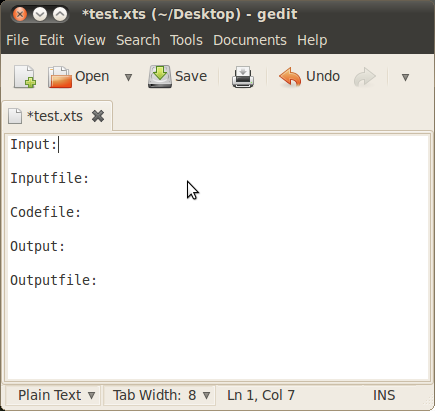
**Step 2:** Save the blank file, giving a meaningful name with an extension of “.xts”.\* This file should be saved in the directory of the code file or in the directory where the code file will be.



**Figure 11**: This figure shows the text editor after the file has been saved.

\*Note: When saving it may be necessary to save as “All Files” in order to avoid your operating system appending .txt to the file name.

An xts file consists of 5 parts: Input, Inputfile, Codefile, Output, Outputfile. These five parts are developed by using their names followed by a “:” on a single line (ie. “Input:”, “Inputfile:”, “Codefile:”, “Output:”, “Outputfile:”). These parts are not case sensitive.

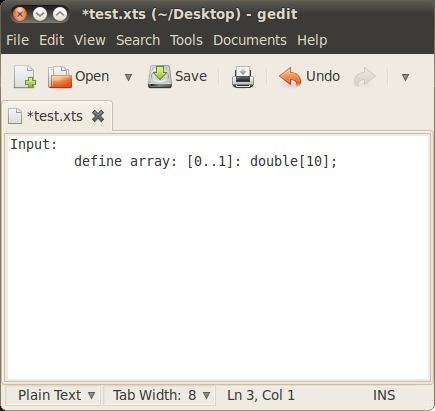


**Figure 12**: This figure shows each of the required sections

Now define the input variables. In this part we will define the input data type (ie. What the genetic algorithm will be evolving and writing to a file to be evaluated by your evaluation function). This is done using the keyword “define” followed by the following syntax:

define N: [R]: D;

Where N is the variable name (in this example array), R is the range of values that can be stored in said variable [Min..Max] (in this example the minimum value is 0 and the maximum is 1), and lastly D defining the data type for our variable (in this example a double array of size 10).



**Figure 13**: Text editor with input section defined.

The following is a list of data types supported by XTOOLSS:

**int** - Integers are represented by natural numbers and should be used when enumerating objects. For example, you would use integers to represent different types of objects such as elements. The difference between element 1 and element 4 is the same as the difference between element 2 and element 3.

**ord** - Ordinals represent natural numbers and should be used when the difference between different magnitudes effects the problem being solved. For example, when determining how many workers are needed for a project, the differences between 6 and 7 workers is smaller than the difference between 6 and 10 workers.

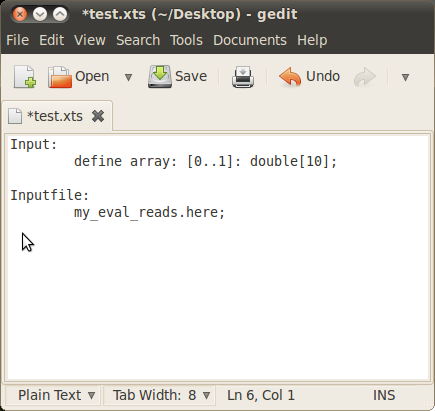
**double -** Doubles represent real numbers and are a double precision floating point format. A typical 32-bit x86 system will use 8 bytes to store a double precision floating point value.

**float -** Floats represent real numbers and real numbers and are a single precision floating point format. This format uses half of the amount of memory allocated for a double.

The only data structure defined in X-TOOLSS is the array which is defined as follows:

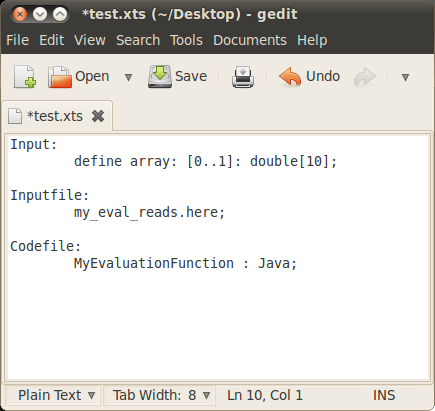
**array -** Arrays provide a convenient way of duplicating variables. If an evaluation function requires 1,000 variables of type double, an array should be used instead of an xts file with 1,000 different variable names.

The next step is define the input file. This is simply the name of the file your evaluation function is expected to read from (in this example our evaluation function reads from a file called “my\_eval\_reads.here”).\*\*



**Figure 14**: This figure shows the module file with the input file.

Next we define the code file. This is simply the name of the evaluation function and the language used to generate the executable.\*\*



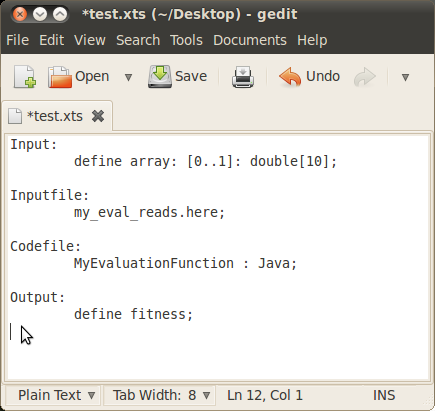
**Figure 15**: This example shows the module file with the code file specified.

The following is a ist of executable types supported by X-TOOLS:

* java (.class files) [executes “java NAME”]
  + This functionality is executing the compiled java byte code (.class) file.
* matlab (.mat files) [executes “matlab –nodisplay < NAME”]
  + This functionality executes matlab to run the specified .mat file.
* C/C++ (.exe files) [executes “./NAME”]
  + This functionality will run any file that can be executed from the command line with the above command. This can be specified in the module file (.xts file) by the keywords “c”, “c++”, or “exe”. An alternate example would be “a.out : exe;”.

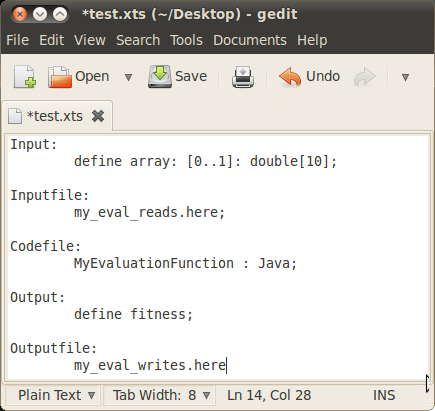
\*\*Note: The defined input file will be created in the working directory, which is the directory that contains the module, code, input and output files. Each running optimization has a unique working directory.

The next step is to define the name we would like to use to refer to the value returned by your evaluation function or codefile.



**Figure 16**: This example shows the output variable defined.

Finally we define the output file. This is where we need X-TOOLSS to read in order to access the value returned by your evaluation function or code file.

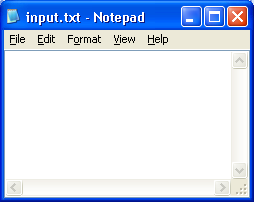


**Figure 17**: The complete module file.

These are all the steps needed to create a xts file which is needed to perform an X-TOOLSS optimization. Figure 17 shows the completed module file.

## The Input File Description and Configuration Requirements

The input file is a text file (.txt extension typically) whose purpose is to channel input variables from X-TOOLSS to the code file. The filename is user defined and completely interchangeable; the only requirement for the filename is that it must be consistent within both the code and module files. This can be edited using any text editor. In this example we will be using Microsoft Notepad. The figure below displays an example of an empty .txt file with the filename “input.txt” used. When X-TOOLSS in run, the input file will be created using the input file name and variables specified in the module file. In this example, the input file name specified is input.txt.



**Figure 18**: This figure shows Microsoft Notepad editing the input file.

## The Output File Description and Configuration Requirements

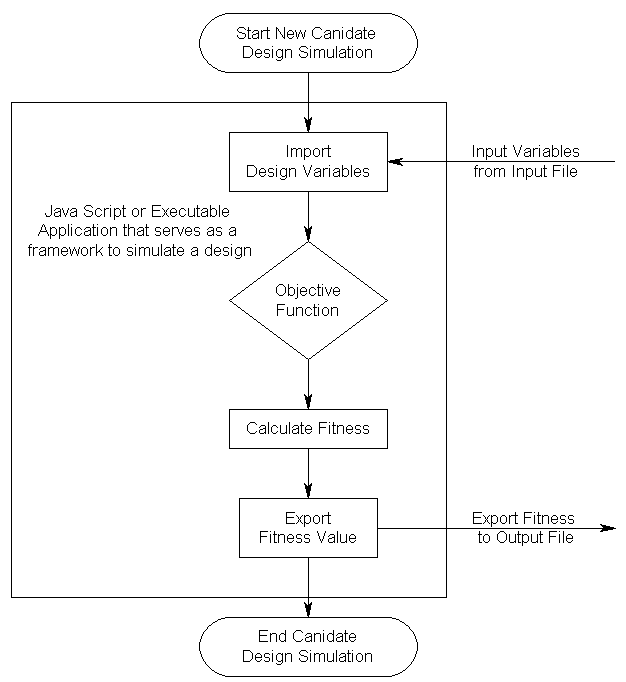
As with the input file, the output file is an arbitrarily named text file that must be consistent with the file name used in the module and code files. In most cases the code file will generate the output file if it has not already been created in the working directory; therefore it may not be necessary to create this file separately. Figure 19 displays an example of an empty .txt file with the filename “fitness.txt” used representing an output file. It is usually given the title “fitness” because this is the file that stores the fitness value after a simulation is executed, but as stated above, this is not a requirement.



**Figure 19**: This figure shows Microsoft Notepad with the output file loaded.

## The Code File Description and Configuration Requirements

The code file must be either a java script or an executable application file. The code file is the file that does the majority of the analysis work in every X-TOOLSS evolution; it serves as a framework that simulates candidate designs that differ only by each generation’s different design variables. The requirements for the code file are that it must be able to read the variables in from the input file, perform the design’s objective function, calculate a fitness for the solution, and output that fitness to the output file. The language used to write the code file, the terms of the fitness function, and any manipulation of the input variables are entirely flexible and at the disposal of the user.



**Figure 20:** A flowchart of the basic structure for any simulation to be X-TOOLSS compatible.

The code file is the code intensive requirement of an X-TOOLSS analysis. You can use any scripting language to simulate whatever it is you are trying to design as long as it follows the structure displayed in the figure above and in either an .exe or .java format. Refer to the X-TOOLSS tutorials that explain how to use different techniques to develop a code file and integrate models built in commercial design and analysis software. As X-TOOLSS continues to be developed the code intensive requirement has been becoming less and less of a burden.

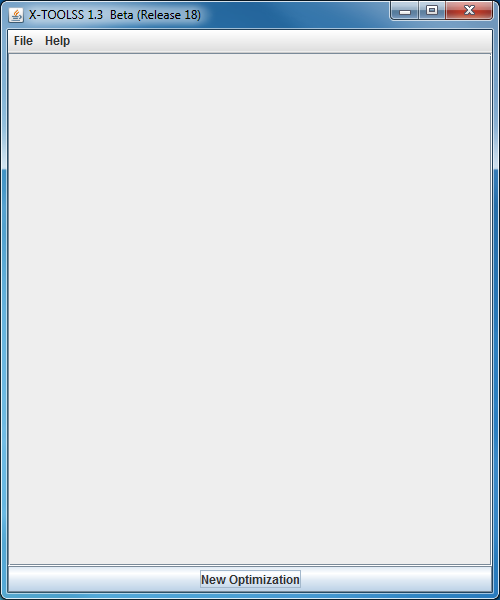
# PART 4: The X-TOOLSS Graphical User Interface

X-TOOLSS can be opened from either the Window’s “Start” Menu or by clicking on the Desktop icon, which is shown below in Figure 21.



**Figure 21**: X-TOOLSS Icon

When starting X-TOOLSS, the “X-TOOLSS Optimization Frame” will pop up. This window is the X-TOOLSS v1.3 graphical user interface (GUI) itself.

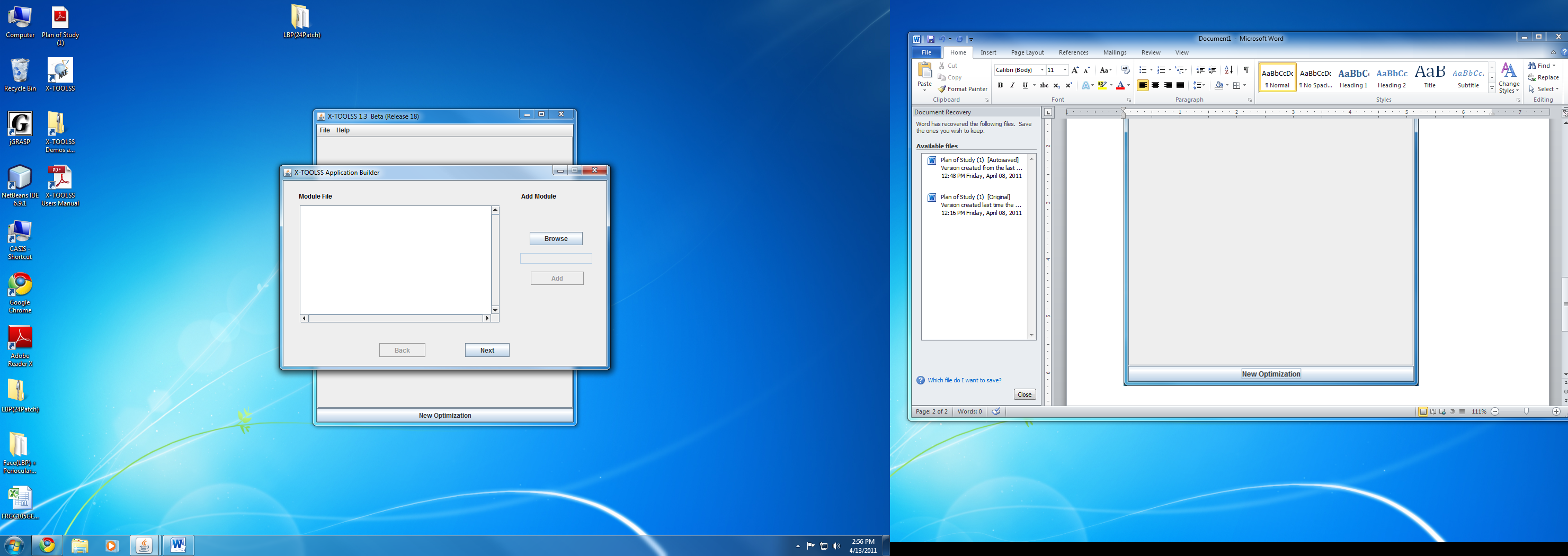


**Figure 22**: X-TOOLSS Optimization Frame

The X-TOOLSS optimization frame allows the user to start new optimizations with the “New Optimization” button at the bottom. The user can also start optimizations through the file menu. The help menu provides versioning formation and help documentation.

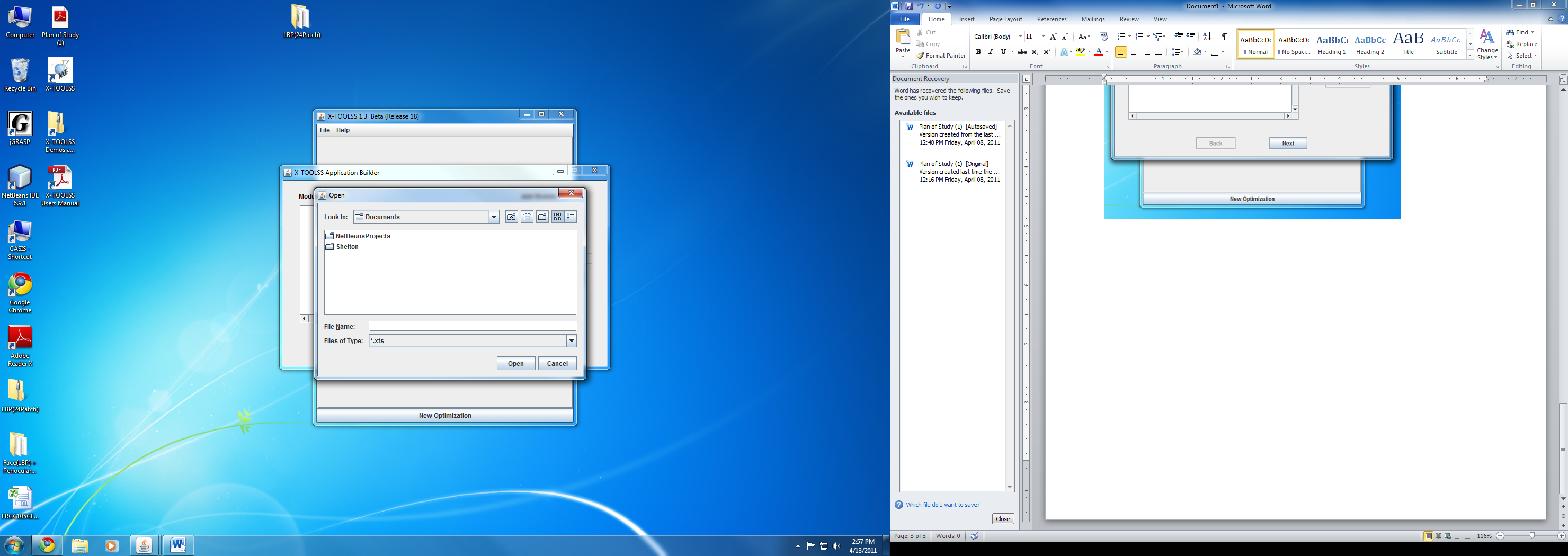
## Optimization Wizard – X-TOOLSS Application Builder

Module Browser and Module Selection



**Figure 23**: New Optimization

When the user creates a new optimization, they will be prompted to browse for the module file. This process is shown in figures 23. When the user selects browse, they will be prompted to select the module file with a file selection dialog window which is shown in figure 24.



**Figure 24**: File selection dialog window.



Up One Level Button:

This button takes the file dialog window to the parent directory.



Desktop Button:

This button takes the file dialog window to the user’s desktop directory.



Create New Folder Button:

This button creates a new folder in the current directory.



List Button:

This button changes the current view of the file selection dialog window to a list view.



Details Button:

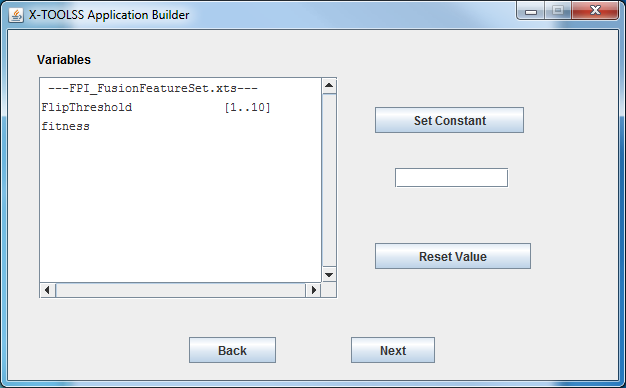
This button changes the current view of the file selection dialog window to a detail view.

Once the file is selected and the user clicks open, the file selection dialog will disappear and the module file path will populate the module file text box. The user then clicks the add button to add that module file to the module file list. This is shown in figure 25.



**Figure 25**: Module file selection window after module file is selected.

When the user clicks next, they will be shown the module file verification window which is shown in figure 26.



**Figure 26:** Module file verification window. This example shows one input variable (FlipThreshold) with ranges between 1 and 10. It also specifies one output variable named “fitness”.

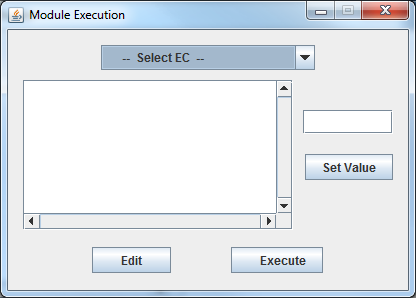
## Module File Verification Window

The module file verification window provides the user with variable information from the module file. It includes the module file name along with all input and output variables.

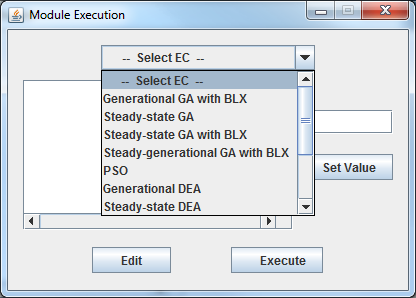
* Set Constant (Currently Broken) – This button allows the user to specify constant values for any of the input variables.
* Reset Value (Currently Broken) – This button allows the user to reset the value of any input variable back to the range defined in the module file.
* Next Button – Takes user to the Module Execution window.
* Back Button – Takes user back to the Module File Selection Window.

## Module Execution Window

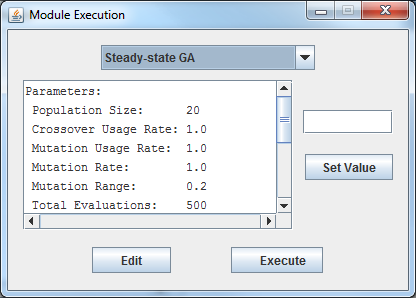
This window allows the user to select the genetic algorithm along with specifying its associated parameters. As of this update, there are 12 different genetic algorithms provided. More information on those algorithms will be provided later in this manual. Figures 27 through 31 show an example of selected a genetic algorithm and modifying a parameter. Once all parameters are set, the user can click the execute button to start the optimization.



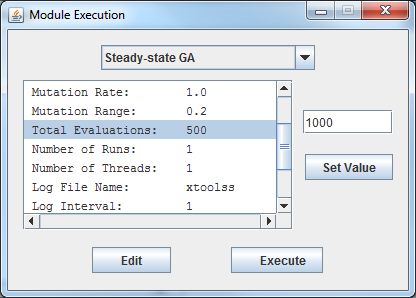
**Figure 27**: The Module Execution Window



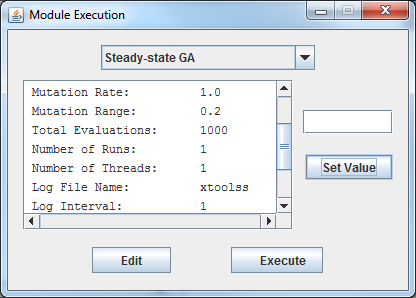
**Figure 28**: Selecting a GA.



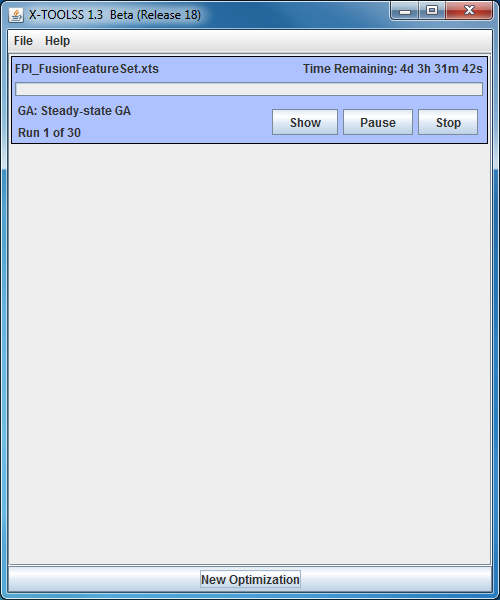
**Figure 29**: The module execution window after selection a GA.



**Figure 30**: Selecting a parameter to update.



**Figure 31**: After setting a parameter value.



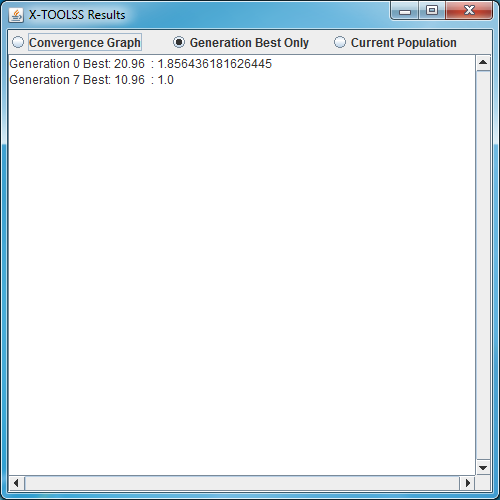
**Figure 32**: Example of the X-TOOLSS optimization frame once an optimization has been started.

Figure 32 shows the result of the X-TOOLSS optimization wizard. The optimization panel (the blue section inside the optimization frame) shows the progress of a single thread running in X-TOOLSS. Included in the optimization panel is the following information:

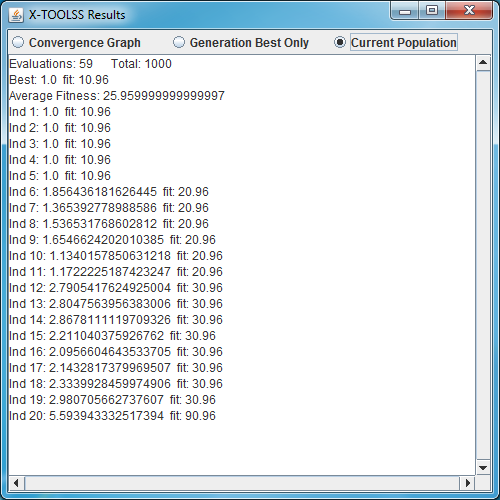
* Module File Name
* Estimated time remaining
* Progress bar
* Selected GA
* Number of optimizations completed
* Show Button
* Opens the X-TOOLSS Results window which provides the user with additional information for that optimization.
* Pause Button
* Allows the user to pause or resume an optimization.
* Stop Button
* Allows the user to stop the optimization. One the optimization is stopped, this button will allow the user to remove that panel from the X-TOOLSS optimization frame.

## X-TOOLSS Results Window

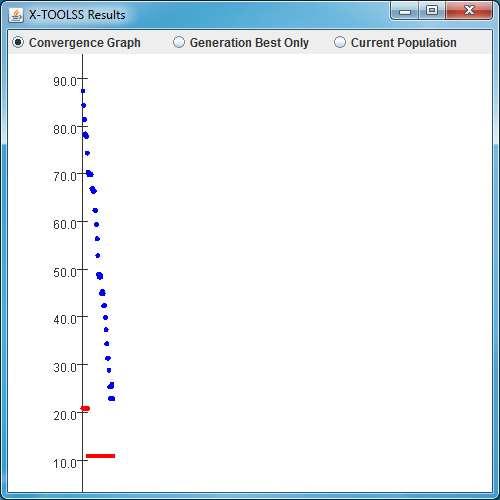
When the show button is pressed in the optimization panel, a results window for that optimization will be shown to the user. This results window has three major display functionalities. These views include the generation best, the current population and the convergence graph of the population. Figures 33 through 35 show these different views.



**Figure 33**: The Generation Best Only view. This view only updates when the optimization finds a new best solution.



**Figure 34**: The Current Population View. This view updates with each iteration of the optimization and includes the number of function evaluations completed, the best candidate solution in the population, the average fitness of the population and each individual and it’s associated fitness.

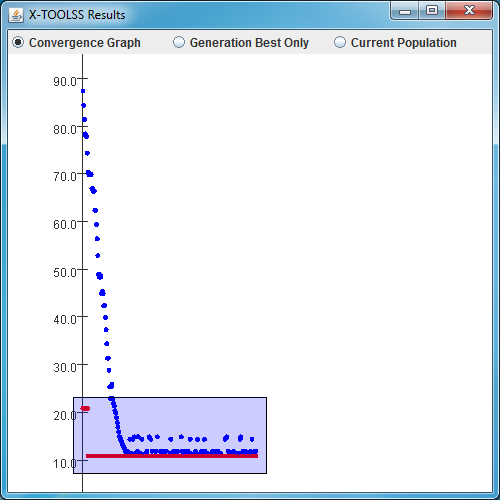


**Figure 35**: The Convergence Graph view. This view updates with each iteration of the optimization. The x-axis represents the enumerated function evaluation while the y-axis represents the fitness value. Blue dots indicate the average of the population’s fitness for the associated function evaluation. The red dots indicate the best fitness in the population for the same function evaluation. There is both a red dot and a blue dot for every function evaluation.

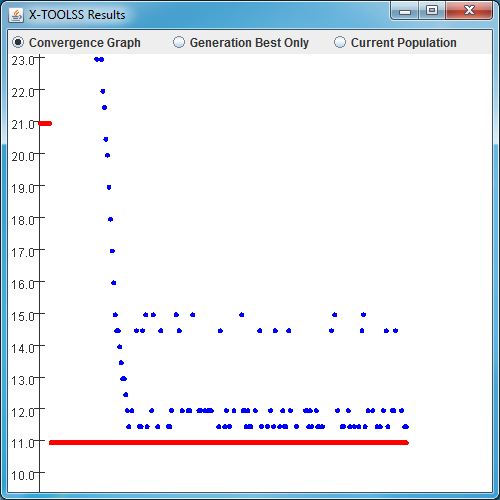
## X-TOOLSS Convergence Graph

By default, the convergence graph will automatically scale to fit all the dots on the screen. If this automatic feature does not provide the desired information to the user, additional functionality is provided to the user for navigation. Figures 36 through 40 provide details on the following navigation features:

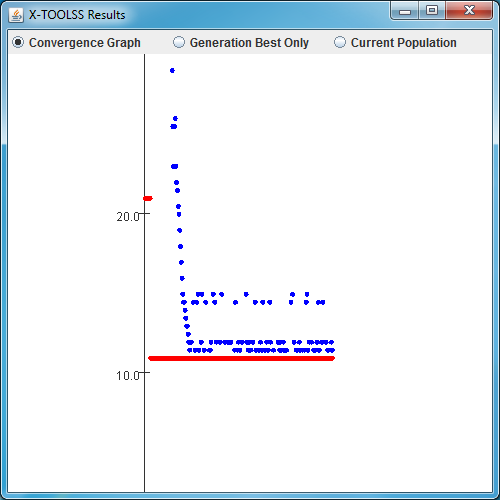
* X-Translation and Y-Translation
  + The arrow keys can be used to translate the axis up, down, left and right.
* Zoom
  + The “z” key can be used to zoom in.
  + “Shift z” can be used to zoom out.
  + Clicking and dragging the mouse over any area in the graph will zoom into that area.
* Reset
  + The “r” key will reset the view to it’s automatic default.



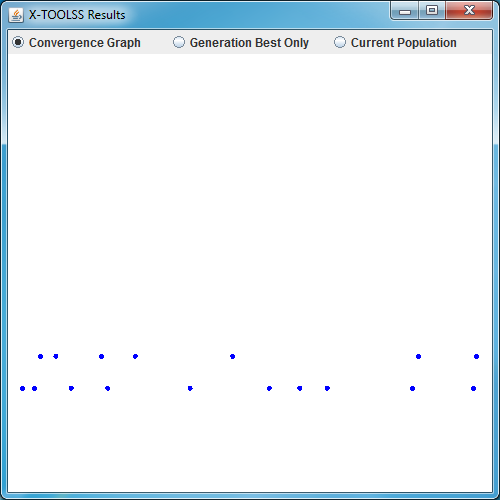
**Figure 36**: This figure shows the drag-zoom feature when using the mouse. As the user drags the cursor over the graph, the blue shaded region is drawn. When the user releases the mouse button, the graph will be zoomed into the shaded area.



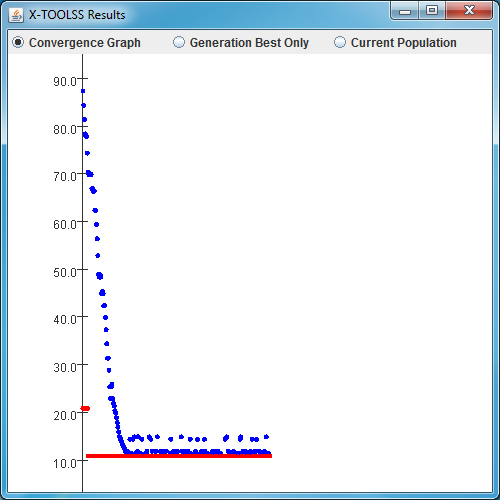
**Figure 37**: This figure shows the graph after zooming into the shaded are from figure 36.



**Figure 38**: This figure shows zoom out feature (shift-z). This view is zoomed out from figure 37.



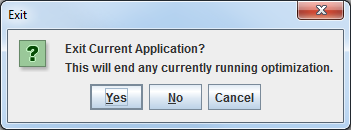
**Figure 39**: This figure shows the zoom-in feature (z). This view is zoomed in from figure 38.



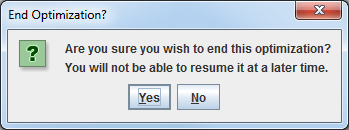
**Figure 40**: This figure shows the reset feature. This is the result of pressing the r-key from figure 39.

## Additional Windows and Screens

The results window can be opened and closed without affecting the optimization. The X-TOOLSS Optimization Frame however will end any running optimizations if closed. The attempting to close the Optimization Frame, the user will be prompted before closing the application. This prompt is shown in figure 41. When the user clicks the stop button in the Optimization Panel, the user will be prompted to confirm the action. This prompt is shown in figure 42.



**Figure 41**: Prompt to confirm closing the X-TOOLSS Optimization Frame.



**Figure 42**: Prompt to confirm stopping the optimization.

# PART 5 Evolution Results

The data from any X-TOOLSS evolution is not only displayed in the “X-TOOLSS Results” window, the software also stores this data in three different output files. Each thread, or simultaneously running X-TOOLSS optimization, creates its own working directory by copying everything in the module file directory into a subdirectory named .\X-TOOLSS\_RUNS\[DATE]\_[RUN#]. For example, if your module file had the following address:

C:\myOptimization\myModuleFile.xts

If you were to start an optimization performing 30 runs with 2 threads on January 1st, 2011, the resulting working directory would be:

C:\myOptimization\X-TOOLSS\_RUNS\20110101\_001\

C:\myOptimization\X-TOOLSS\_RUNS\20110101\_002\

The last number of the working directory path would be incremented for each run on that day. Because you are performing 30 runs using 2 threads, each working directory would hold the results of 15 runs each. Within each working directory there is an X-TOOLSS\_LOGS folder that contains the log, stat and out files.

## 

## xtoolss.log file:

The plain text file xtoolss.log houses a list of the best individual in each generation. The data includes the values of all the input variables separated by spaces and followed by “||” with the fitness of the individual at the end of each entry.

Example of the format:

**Candidate 1: Variable1 Variable2...VariableN || Fitness Value**

**Candidate 2: Variable1 Variable2...VariableN || Fitness Value**

**.**

**.**

**.**

**Candidate N: Variable1 Variable2...VariableN || Fitness Value**

**Figure 43**: Example format for the log file.

## xtoolss.out file:

The xtoolss.out file can also be opened in any text editor and is the complete output file for the application. It contains the variable values and fitness for every generation and every individual. The population, total evaluations, best solution, and average fitness for each generation are also displayed.

This file is only updated on the log interval. For example, if your optimization is set for 500 function evaluations with a log interval of 100, the xtoolss.out file will 5 sets of the following format:

Example of the format:

**Number of Generations Completed Total Number of Generations To Occur**

**Current Generations Best: Variable1 Variable2...VariableN Fitness Value**

**Average Fitness Value**

**Ind 1: Variable1 Variable2...VariableN Fitness Value**

**Ind 2: Variable1 Variable2...VariableN Fitness Value**

**.**

**.**

**.**

**Ind N: Variable1 Variable2...VariableN Fitness Value**

**Figure 44**: Example format for the out file.

## xtoolss.stat file:

The STAT file, xtoolss.stat, can be opened in any text editor and displays only the best (minimum) fitness value of the optimization. If your optimization is set to perform 15 runs, the STAT file will contain 15 fitness values.

Example of the format:

**Evolutuon’s Best Fitness Value**

**Figure 45**: Example format for the stat file.

# PART 6: Supported GECs and Parameters

X-TOOLSS uses the GENEVOT library, which currently supports 12 different genetic algorithms. While each GEC evolves a population of candidate solutions to find an optimum solution, the process in which each one does this differs. The effectiveness of each GEC largely depends on the problem it is optimizing. The following is a list of supported GECs and their respective parameters.

## 1 Generation GA with BLX

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Number of Elites The number of best individuals that will survive to the next generation.

Crossover Usage Rate The probability that an individual will undergo crossover (i.e., recombination).

BLX-alpha The alpha parameter for blend crossover. The values for alpha should lie in the interval [0.0, 0.5]. Alpha values greater than 0.0 cause blend crossover to also introduce a “mutation” as well, and the higher the alpha value, the greater this mutation becomes.

Mutation Usage Rate The probability that an individual will undergo mutation.

Mutation Rate The probability that an allele (i.e., a component of an individual) will undergo mutation.

Mutation Range The percentage of an allele’s total range that is used as the standard deviation for the Gaussian mutation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss”.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 2 Steady-state GA

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Crossover Usage Rate The probability that an individual will undergo crossover (i.e., recombination).

Mutation Usage Rate The probability that an individual will undergo mutation.

Mutation Rate The probability that an allele (i.e., a component of an individual) will undergo mutation.

Mutation Range The percentage of an allele’s total range that is used as the standard deviation for the Gaussian mutation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 3 Steady-state GA with BLX

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Crossover Usage Rate The probability that an individual will undergo crossover (i.e., recombination).

BLX-alpha The alpha parameter for blend crossover. The values for alpha should lie in the interval [0.0, 0.5]. Alpha values greater than 0.0 cause blend crossover to also introduce a “mutation” as well, and the higher the alpha value, the greater this mutation becomes.

Mutation Usage Rate The probability that an individual will undergo mutation.

Mutation Rate The probability that an allele (i.e., a component of an individual) will undergo mutation.

Mutation Range The percentage of an allele’s total range that is used as the standard deviation for the Gaussian mutation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 4 Steady-generational GA with BLX

**Parameters**:

Population Size The number of individuals that exist in the population at any one time.

Crossover Usage Rate The probability that an individual will undergo crossover (i.e., recombination).

BLX-alpha The alpha parameter for blend crossover. The values for alpha should lie in the interval [0.0, 0.5]. Alpha values greater than 0.0 cause blend crossover to also introduce a “mutation” as well, and the higher the alpha value, the greater this mutation becomes.

Mutation Usage Rate The probability that an individual will undergo mutation.

Mutation Rate The probability that an allele (i.e., a component of an individual) will undergo mutation.

Mutation Range The percentage of an allele’s total range that is used as the standard deviation for the Gaussian mutation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 5 PSO

**Parameters:**

Number of Particles The number of individuals that exist in the population at any one time.

Neighborhood Size The number of best individuals that will survive to the next generation.

Constriction Coefficient? The probability that an individual will undergo crossover (i.e., recombination).

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 6 Generational DEA

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Phi The number of best individuals that will survive to the next generation.

Mutation Usage Rate The probability that an individual will undergo mutation.

Mutation Rate The probability that an allele (i.e., a component of an individual) will undergo mutation.

Mutation Range The percentage of an allele’s total range that is used as the standard deviation for the Gaussian mutation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 7 Steady-state DEA

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Number of Elites The number of best individuals that will survive to the next generation.

Phi The probability that an individual will undergo crossover (i.e., recombination).

Mutation Usage Rate The probability that an individual will undergo mutation.

Mutation Rate The probability that an allele (i.e., a component of an individual) will undergo mutation.

Mutation Range The percentage of an allele’s total range that is used as the standard deviation for the Gaussian mutation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 8 Elitist DEA

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Number of Elites The number of best individuals that will survive to the next generation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 9 Standard EP

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 10 Continuous Standard EP

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 11 Meta-EP

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Eta Mutation Rate The number of best individuals that will survive to the next generation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

## 12 Continuous Meta-EP

**Parameters:**

Population Size The number of individuals that exist in the population at any one time.

Eta Mutation Rate The number of best individuals that will survive to the next generation.

Total Evaluations The number of function evaluations to perform before termination.

Number of Runs The number of independent runs to perform.

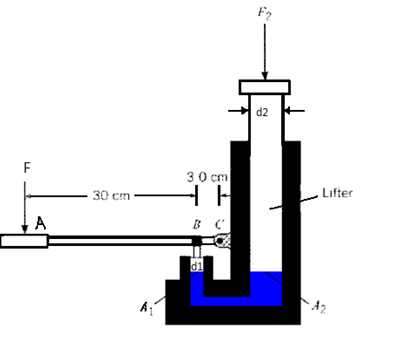
Number of Threads The number of threads to divide the runs into.

Log File Name The name of the log files to be saved. The default value is “xtoolss” so the three log files that will be saved by default are: xtoolss.log, xtoolss.stat, and xtoolss.out.

Log Interval The number of function evaluations to execute between screen/file output. If this value is set to anything greater than the Total Evaluations, then only the final generation of each run will be output.

# PART 7: An X-TOOLSS Example Problem

Consider a hydraulic jack with a 100 N force, F, exerted on the handle of the jack, as seen in Figure 46.



**Figure 46:** Hydraulic jack schematic.

Under this load, a force is exerted on the small piston with diameter. We can solve for this force by taking the moments about C.

This equation can easily be solved for . Because the small piston is in equilibrium, this force is equal to the pressure force on the piston, or

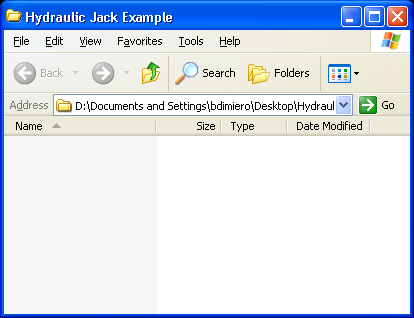
where is the pressure on the face of the small piston and is the area of the face of the small piston. This equation can easily be solved for , the pressure of the liquid. Knowing this pressure we can solve for the force on the large piston. Since surfaces 1 and 2 are at the same elevation, , we know

To optimize the design of hydraulic jack, diameters of the small and large pistons must be specified to provide the largest load the jack can support. Analysis of the combination of previous equations provides a simple understanding of how these dimensions will affect the load.

Since the area of the large piston is in the numerator of the equation, a maximum area will result in a maximum supported load. Alternately, the area of the small piston is in the denominator; therefore, a minimum area will result in maximum supported load. With this expectation in mind, we can allow the X-TOOLSS algorithms to optimize the design and judge its results.

**Step 1: Create a new optimization directory**

X-TOOLSS requires all of its application components all be located within the same directory. So for any new X-TOOLSS analysis, create a separate module directory. **For this example, create a new module directory titled “Hydraulic Jack Example”**. Figure 47 below displays this new, empty directory.

****

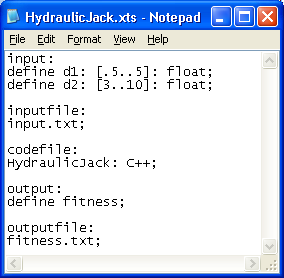
**Figure 47:** The new optimization directory.

**Step 2: Create a model file**

The module file is the building block of an analysis for X-TOOLSS. It provides the critical instructions needed in order to perform an X-TOOLSS analysis. A module is simply a text file that has been saved with an .xts file extension. This file must include the input variables (variable name, range, and type), the input file name, the code file name and type, the name of the variable output by the code file (fitness variable), and the output file name.

For the hydraulic jack, the design variables used to optimize the system will be the diameters of the big and small pistons, d1 and d2. They will range from 0.5 to 5 meters and 3 to 10 meters respectively. Both variables will be designated as floats to accept decimal places. The input file can have any name, but must match the file name in the code file, “input.txt in” this case. The code file was written in C++ and compiled to an executable named “HydraulicJack.exe”. This code file will calculate the fitness value for any given candidate design and export that value to the output, which was given the name “fitness.txt”. The text from HydraulicJack.xts that was used in this example is shown in the text box below.

**Copy the text from the text box below into a blank text editor and save the file as “HydraulicJack.xts” to the module directory.**



input:

define d1: [.5..5]: float;

define d2: [3..10]: float;

inputfile:

input.txt;

codefile:

HydraulicJack: C++;

output:

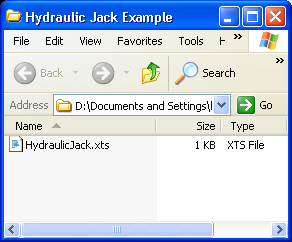
define fitness;

outputfile:

fitness.txt;

**Figure 48:** Module file for hydraulic jack example.

The optimization directory should now look like the screenshot below.

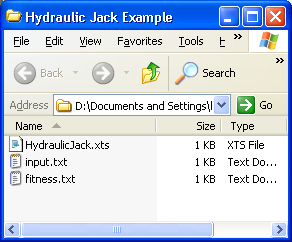


**Figure 49:** The current module directory.

**Step 3: Create the input and output files**

**This step is only required if applications such as X-TOOLSS or your code file do no have permissions to create files. This varies based on operating system and operating system configuration. These files should be created automatically in the optimization process.**

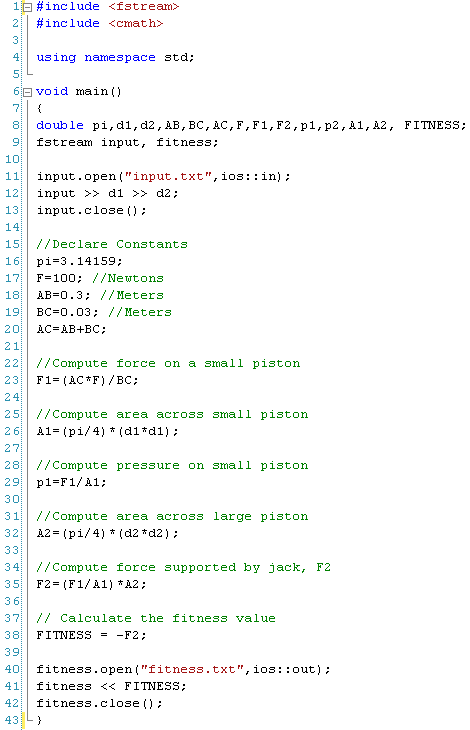
The input file is a text file (.txt extension) whose purpose is to channel input variables from X-TOOLSS to the code file. Similarly, the output file is a text file used to store the fitness value after a simulation completes its analysis. **For this example, create and save two empty text files named “input.txt” and “fitness.txt” to the same folder as the HydraulicJack.xts module file.** This can be done by using any text editor; typically Microsoft’s Notepad is used to do this. The current module directory should now look like the screenshot below.

****

**Figure 50:** The current module directory.

**Step 4: Develop the code file**

The code file can be either a MatLab script, compiled Java byte-code (.class) or an executable file. Be aware that if you did not write/compile the executable file, it may need to be recompiled or reinstalled on your system. The only requirements for the code file are that it must be able to read the variables in the input file, use them to calculate a fitness for the solution, and output that fitness to the output file. The language used to write the code file, the terms of the fitness function, and any manipulation of the input variables are entirely flexible and at the disposal of the user. The screenshot below displays the C++ script use to compile the executable that will be used as the code file.

****

In-Stream

Objective

Function

Fitness Calculation

Out-Stream

**Figure 51:** The code file script written in C++ which was compiled to an executable.

The instructions on lines 11 through 13 open the designated file (the input file), extract the data from within it, and assign that data to variables within the script. These few lines of code complete the first requirement of any code file, that is, to import the design variable data.

The objective function consist of the analysis written in the code from lines 15 to 35, as required by X-TOOLSS a fitness calculation needs to be made after the objective function and data streams need to be placed before the objective function and after the fitness calculation. The fitness calculation is written on line 38, which sets the value of the supported force by the jack equal to a variable named “FITNESS”. The reason the value of this force is converted to a negative is because we are searching for the maximum value of F2 while X-TOOLSS searches for a minimum value of fitness. Finally lines 40 through 42 contain the commands to export the value of the fitness function to the output file named “fitness.txt”.

Creating a C++ script and compiling it into an executable is only one of many different possible ways to create a code file. Refer to the others X-TOOLSS tutorials that demonstrate a variety of different techniques on how to do this**. For the purposes of this tutorial the compiled code file and its source code should have been provided with the User’s Manual**. **Place “HydraulicJack.exe” into the optimization directory.**

However you choose to design the code file it must follow the structure outlined in Part 3 of the Users Manual. The key is that the fitness function reads from the input file specified in module file to calculate the fitness, and that it writes to the output file specified in the module file. The current module directory should now look like the screenshot below.



**Figure 52:** The current module directory.

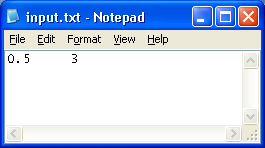
**Step 6: Test the code file for proper functionality**

In order to test if the code file runs properly, do the following:

1. Delete the output file (Named fitness.txt in this example).
2. Open the input file, delete whatever is in it, and enter tab delimited data corresponding to the variable ranges defined within the program. In this case just try entering the minimum limits of each variable, but any numerical value within each variables range will suffice.

**Enter the following into “input.txt” and then save the file:**

**0.5 3**



**Figure 53:** A screenshot of the input file with the minimum limits entered into it.

1. Save the input file, and double click the code file (HydraulicJack.exe).
2. The command prompt should flash onto the screen and then close, the output file should then be created within the directory containing the executable.
3. Open the newly created output file, fitness.txt, to find the value of the fitness within this text file, to confirm it is correct you can do a hand calculation of the objective code with the variables you chose.

If you entered 0.5 and 3 into the input file, than the calculated fitness value should be

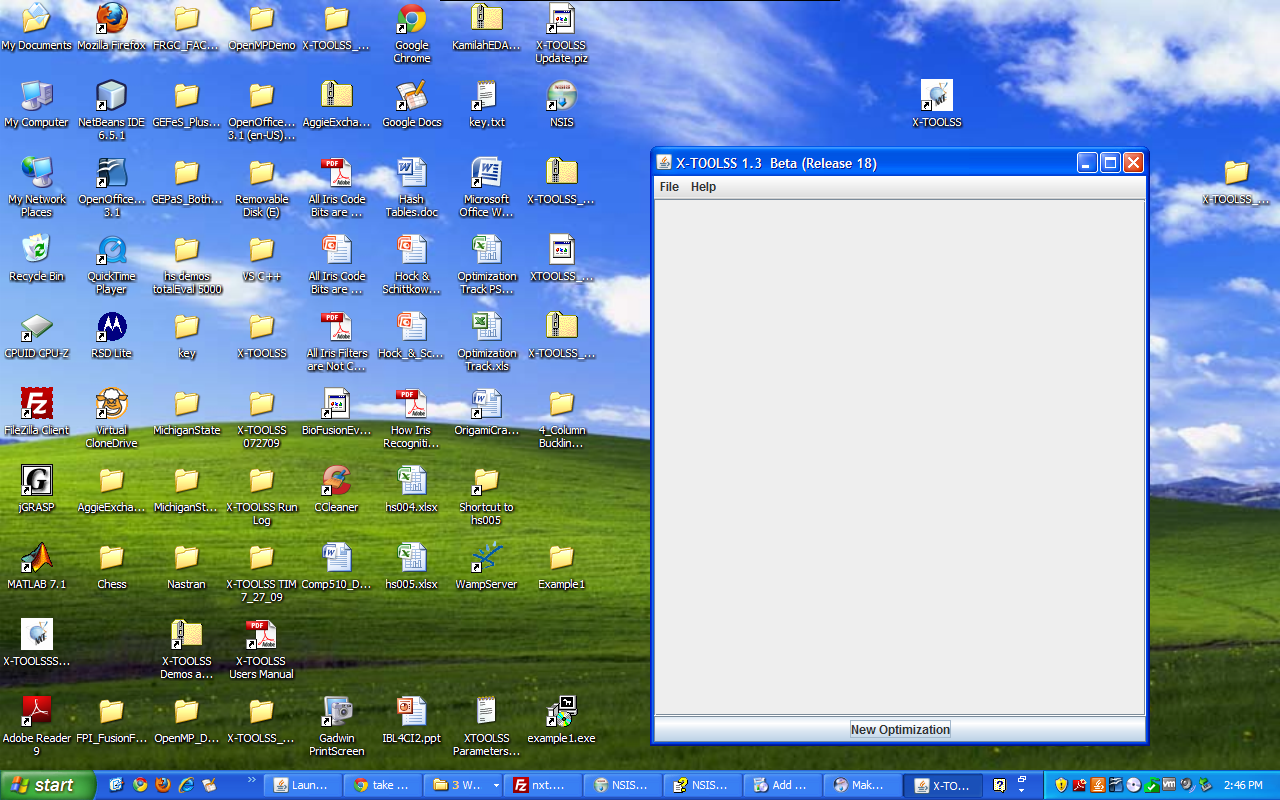
-39599.999999999993.

**NOTE:** *Hand calculation will tell you that the answer is -39600; the actual results may have insignificant rounding errors as a result of the binary calculations done via the CPU. These errors are too small to have any impact on the results.*

1. If this worked properly, than you are ready to run an X-TOOLSS analysis.

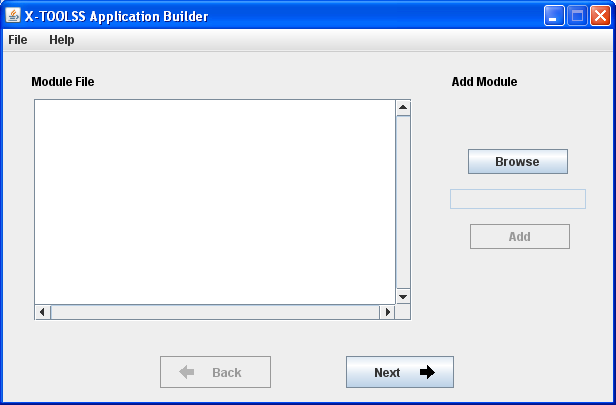
**Step 6: Run X-TOOLSS with the Newly Created Module**

To run a module, first open the X-TOOLSS Application from the Start menu or the Program File (C:/Program Files/X-TOOLSS if the default directory was used). The first screen of X-TOOLSS is the module importation screen shown in Figure 54.



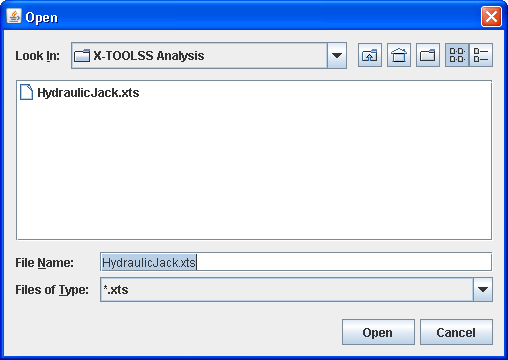
**Figure 54**: Initial X-TOOLSS Screen, also called the Optimizations Frame.

Once the application opens, click on the New Optimization button. The Application Builder window will open. This is shown in figure 55.



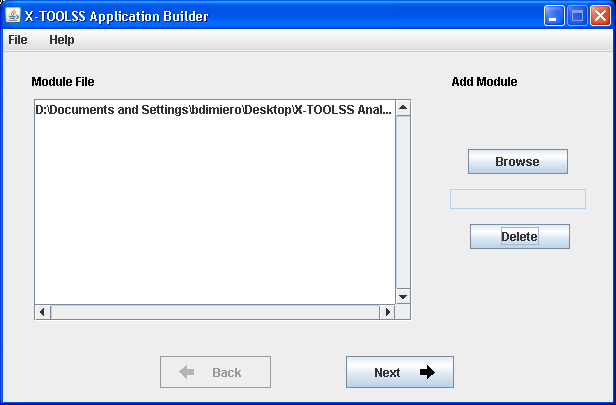
**Figure 55:** X-TOOLSS opening screen.

Clicking the “Browse” button will allow the addition of a module. Find the HydraulicJack.xts module file and click “Open” to load it. The HydraulicJack.xts file selection is shown in Figure 56.



**Figure 56:** Module addition window.

Once the module has been opened, X-TOOLSS returns to the Application Builder window. The module added will appear in the right hand text box. Click “Add” to load that module to the application. It will appear in the Module File window on the left as shown in Figure 57.



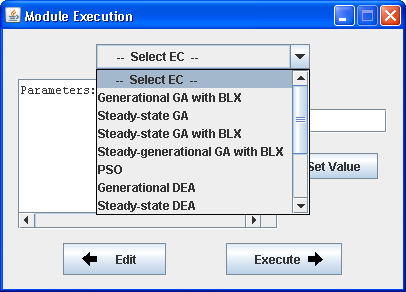
**Figure 57:** Opened module HydraulicJack.xts ready to be added to the application.

Once the module is loaded, click Next to move to the Variables window. In this window, shown in Figure 58, the module currently added to the application is shown with the input and output variables named and defined.



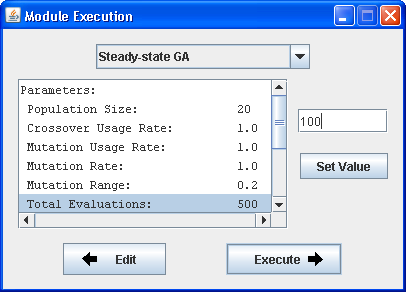
**Figure 58:** X-TOOLSS variable manipulation window.

Clicking “Next” on the Variables window brings up the EC selection window. Click the drop-down menu to view the selection of EC algorithms, as shown in Figure 59.



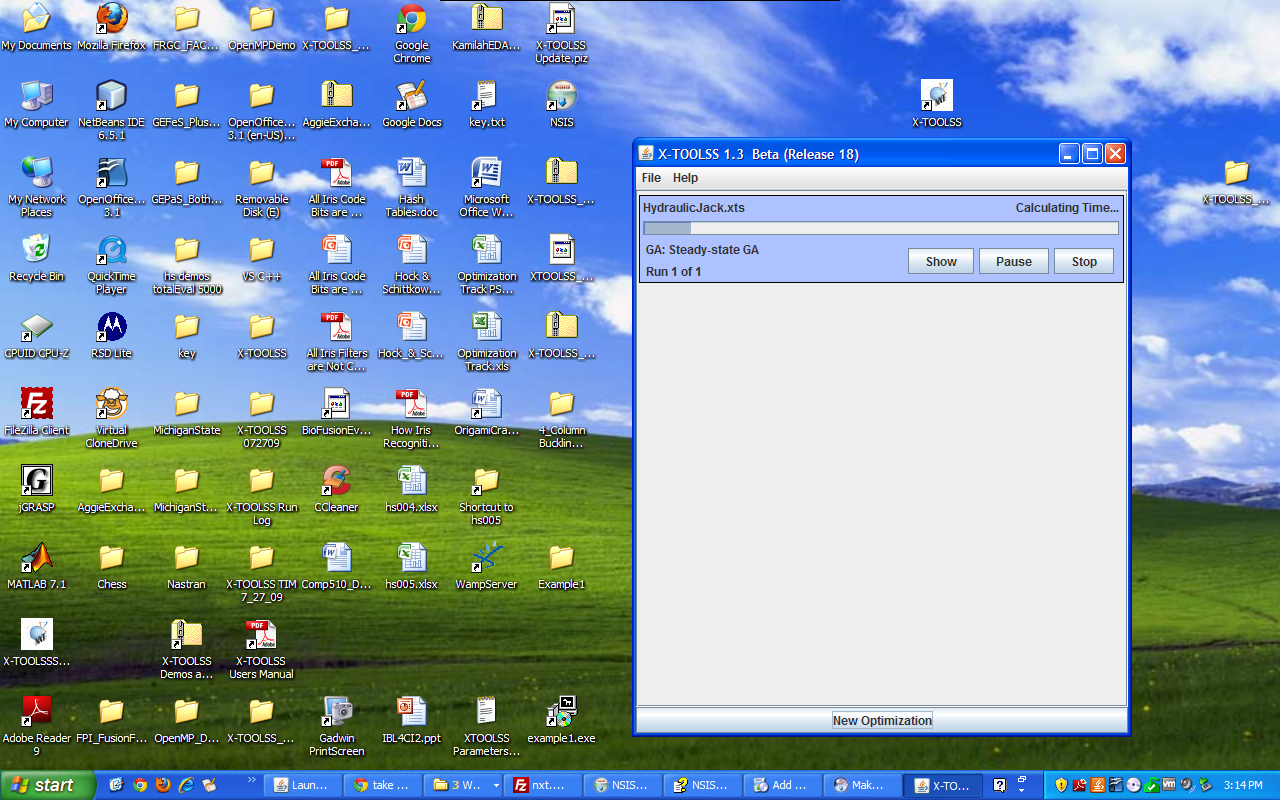
**Figure 59:** Selection of evolutionary computing algorithm.

Once an algorithm is selected, the parameters for that particular algorithm are shown in the window and can be edited. Each algorithm’s parameters are reviewed in the Help Menu of the Application Builder window. Figure 60 shows the steady-state genetic algorithm selected. The Total Evaluations parameter is in the process of being changed from 500 to 100. To change a parameter, select it in the window, input the new value in the text box to the right, and click “Set Value.”



**Figure 60:** EC selection and parameter modification.

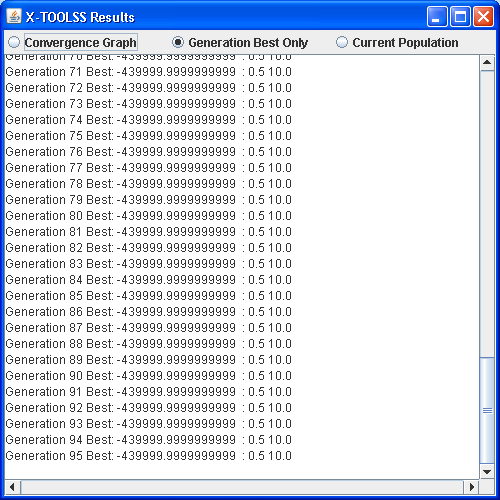
After all parameters have been reviewed, clicking “Execute” will start the application. The Module Execution and X-TOOLSS Application Builder windows will close and the Optimizations Frame will be visible. This is shown in Figure 61.



**Figure 61**: Optimization Frame running the Hydraulic Jack optimization.

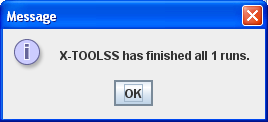
The Optimization Panel within the Optimization Frame provides an overview of the progress of the optimization. It also includes buttons to control the optimizations process along with getting additional information. You can start simultaneous optimizations by repeating the previous process. To get additional information on a particular optimization, click the show button. This button opens the X-TOOLSS Results window shown in figure 62.

This X-TOOLSS Results window provides a number of viewing options. Selecting “Convergence Graph” will show a plot of the fitness of each evaluation, the “Generation Best Only” option shows the best fitness solution for each generation, and the “Current Population” option will show all of the solutions in the current generation.



**Figure 62:** X-TOOLSS Results window.

Once the current application has finished running all runs (a parameter that can be set) for that algorithm, it will display the message in Figure 63 notifying completion.



**Figure 63:** Run completion notification window.

**Step 7: Analyze the Results**

It is important to note that no two simulations will likely ever be the same. There is no test to find however many generations are needed in order to find the optimal solution since every evolution will be different. The more the model is allowed to evolve will increase the chances that X-TOOLSS will find better solutions. In the case of the hydraulic jack problem, the optimal solution was known before even running a model. The optimal candidate model would have had the smallest possible d1 value and largest possible d2 value in order to produce the largest supported load. Again, this was only an example to demonstrate the capabilities for X-TOOLSS to search for optimal designs. Other examples exist where an optimal candidate design does not have an obvious, straightforward answer. Refer to the other X-TOOLSS tutorials for more examples and different techniques used to optimize designs developed in commercial design and analysis tools.

# PART 8: Future Anticipated Development

In order for NASA to fully utilize design optimization with X-TOOLSS, easy-to-use methods need to be developed to optimize specific types of problems. The goal for its development would be to advance X-TOOLSS to the point where it was user-friendly enough that an engineer will not have to invest a considerable amount of time learning a process and writing a custom code in order to run beneficial types of optimization simulations.

X-TOOLSS can now, as it stands, optimize complex designs; it has the potential of significantly changing and improving the conventional design process. Genetic algorithm based optimization tools like X-TOOLSS have not only the potential of benefiting NASA, but the scientific and engineering industry as a whole. Evolutionary computing has the ability to optimize designs very early in the design phase, where it is the most beneficial, since at this point in the design process the cost associated with change is the lowest.

# Glossary

|  |  |
| --- | --- |
| **Candidate Design** | A candidate solution is a possible solution to problem being optimized. |
| **Candidate Solution** | See Candidate Design. |
| **Evolution** | The change in population alleles over time. In the case of genetic algorithms, evolution is the change in the population over the specified number of function evaluations. |
| **EC** | Evolutionary Computation |
| **GA** | Genetic Algorithm |
| **GEC** | Genetic and Evolutionary Computation |
| **Population** | A set of candidate solutions for a problem. |
| **Generation** |  |
| **Wrapper** |  |
| **Script** |  |

# References

NCAT

* Josh Adams
* Kelvin Bryant
* Gerry Dozier

NASA MSFC

* Mike Tinker
* Andrew Schnell
* Patrick Hull
* Jesse Hugeut
* Ben DiMiero
* Erin Moore
* Julianna
* Matt
* Adam

Original Developers

* Aaron Garrett