OSTRICH – An Optimization Software Toolkit   
for Research Involving Computational Heuristics

Documentation and User’s Guide

Version 2.0

by

L. Shawn Matott, Ph.D.

State University of New York at Buffalo

Center for Computational Research

Acknowledgements

Funding for the creation of this user manual was provided by Environment Canada under contract number K3D35-14-0487R. The Dynamically Dimensioned Search (DDS) algorithm was developed by Professor Bryan Tolson and implemented in C/C++ code by Professor James Craig. Both James and Bryan are currently at the University of Waterloo. Professor Craig also provided routines for generating normally distributed random variables. Other variants of the DDS family of algorithms were ported to OSTRICH using C/C++ and FORTRAN implementations provided by Professor Tolson’s research group. Hyper volume calculations within the Pareto Archived DDS code have been adapted from original C++ source developed by Nicola Beume at the University of Dortmund. The Shuffled Complex Evolution (SCE) algorithm was ported from the original FORTRAN implementation of Dr. Qingyun Duan. All other algorithm implementations are based on published descriptions and were coded primarily by L. Shawn Matott. The basic object-oriented and model-independent structure of OSTRICH is based off of a code known as MACT (Multi-Algorithm Calibration Tool) that was developed by Vijaykumar Raghavan while at the University at Buffalo and under the supervision of Professor Alan J. Raideau. Portions of the genetic algorithm and simulated annealing implementations in OSTRICH were ported from Mr. Raghavan’s MACT code.

Preface

OSTRICH is a model-independent and multi-algorithm optimization and calibration tool. It can be used for weighted non-linear least-squares calibration of model parameters, or for constrained optimization of a set of design variables according to a user-defined objective or cost function. Both single and multi-objective optimization are supported along with multi-criteria calibration. Parameters to be calibrated or optimized can be log-transformed or computed as functions of other parameters. OSTRICH is also capable of computing an extensive set of post-calibration statistics, include confidence intervals, parameter correlation, tests of normality and non-linearity, and measures of observation influence and parameter sensitivity. OSTRICH can be configured to operate with any modeling program that utilizes text-based input and output file formats. Additional I/O formats that are supported include the MS Access database and netcdf formats. Executable versions of OSTRICH are available for both Windows and Linux-based computing environments. A parallel version of OSTRICH (OstrichMPI), utilizing the industry standard MPI interface, is also available in both Windows and Linux. Linux builds of OstrichMPI are available for both the OpenMPI and Intel-MPI implementations of the MPI standard. The Windows-based OstrichMPI uses a file-based implementation of the MPI standard developed by the author (L. Shawn Matott, lsmatott@buffalo.edu).

Table of Contents

[1. Introduction 4](#_Toc426038684)

[1.1. Calibration and Optimization Algorithms 5](#_Toc426038685)

[1.2. Regression Statistics and Diagnostics 9](#_Toc426038686)

[2. ostIn.txt – the OSTRICH Input File 10](#_Toc426038687)

[2.1. Comments 11](#_Toc426038688)

[2.2. Case Sensitivity 12](#_Toc426038689)

[2.3. ostIn – Basic Configuration 12](#_Toc426038690)

[2.4. ostIn – File Pairs 16](#_Toc426038691)

[2.5. ostIn – Extra Files 17](#_Toc426038692)

[2.6. ostIn – Extra Directories 18](#_Toc426038693)

[2.7. ostIn – Real-valued Parameters 19](#_Toc426038694)

[2.8. ostIn – Integer Parameters 20](#_Toc426038695)

[2.9. ostIn – Combinatorial Parameters 21](#_Toc426038696)

[2.10. ostIn – Tied Parameters 21](#_Toc426038697)

[2.11. ostIn – Special Parameters (pre-emption) 25](#_Toc426038698)

[2.12. ostIn – Initial Parameters 27](#_Toc426038699)

[2.13. ostIn – Parameter Correction 27](#_Toc426038700)

[2.14. ostIn – Observations 30](#_Toc426038701)

[2.15. ostIn – Response Variables 30](#_Toc426038702)

[2.16. ostIn – Tied Response Variables 30](#_Toc426038703)

[2.17. ostIn – Type Conversion (MS Access, netcdf) 30](#_Toc426038704)

[2.18. ostIn – Search Algorithms 30](#_Toc426038705)

[2.19. ostIn – Multi-Objective Search Algorithms 30](#_Toc426038706)

[2.20. ostIn – Math and Stats 30](#_Toc426038707)

[2.21. ostIn – Line Search 30](#_Toc426038708)

[2.22. ostIn – General-purpose Constrained Optimization Platform (GCOP) 31](#_Toc426038709)

[2.23. ostIn – Constraints 31](#_Toc426038710)

[3. Creating Template Files 31](#_Toc426038711)

[4. Running Ostrich 31](#_Toc426038712)

[4.1. Serial Execution 32](#_Toc426038713)

[4.2. Multi-core Parallel Execution in Windows 32](#_Toc426038714)

[4.3. Distributed or Multi-core Parallel Execution in Linux 32](#_Toc426038715)

[4.4. Aborting an Ostrich Run 33](#_Toc426038716)

[4.5. Restarting an Ostrich Run 33](#_Toc426038717)

[5. Ostrich Output Files 33](#_Toc426038718)

[5.1. OstOutput – Main Output File 33](#_Toc426038719)

[5.2. OstOutput – Statistical Output 33](#_Toc426038720)

[5.2.1. Observation Residuals 34](#_Toc426038721)

[5.2.2. Error Variance and Standard Error of the Regression 34](#_Toc426038722)

[5.2.3. Parameter Variance-Covariance and Correlation 34](#_Toc426038723)

[5.2.4. Confidence Intervals 34](#_Toc426038724)

[5.2.5. Model Linearity 34](#_Toc426038725)

[5.2.6. Normality of Residuals 34](#_Toc426038726)

[5.2.7. Influential Observations 34](#_Toc426038727)

[5.2.8. Parameter Sensitivities 34](#_Toc426038728)

[5.2.9. Matrices 35](#_Toc426038729)

[5.3. OstError – OSTRICH Error and Warning Messages 35](#_Toc426038730)

[5.4. OstExeOut – Redirected Model Output 35](#_Toc426038731)

[5.5. OstModel – Model Run Record 35](#_Toc426038732)

[6. Examples 35](#_Toc426038733)

[6.1. Demo #1 – Calibrating SPLIT Groundwater Flow Model 35](#_Toc426038734)

[6.2. Demo #2 – Pump-and-Treat Optimization 35](#_Toc426038735)

[6.3. Demo #3 – BIGFOOT (Benchmarking Interface for Global Function Optimizers and Optimization Toolkits) 35](#_Toc426038736)

[6.4. Demo #4 – TUSWAMP (Tufts University Simple Watershed Modeling Program) 35](#_Toc426038737)

[6.5. Demo #5 – Cantilever Beam Multi-Objective Optimization 35](#_Toc426038738)

[6.6. Demo #6 – Multi-Criteria MODFLOW Calibration 35](#_Toc426038739)

# Introduction

OSTRICH is a model-independent program that automates the processes of model calibration and design optimization ***without requiring the user to write any additional software.*** Typically, users only need to fill out a few required portions of the OSTRICH input file (i.e. ostIn.txt) and create template model input files. Users may also activate and configure a variety of optional features, including: parallel processing, model pre-emption, algorithm restarts, parameter statistics and regression diagnostics, telescoping parameter bounds, predictive uncertainty, non-ASCII model I/O, and user-defined initial parameter sets. Finally, users with skill in code or script development (e.g. C/C++, FORTRAN, Java, R, MATLAB, Python, bash/bat, etc.) may also wish to take advantage of OSTRICH’s parameter correction feature. It allows users to correct candidate parameter sets based on rules-of-thumb or expert judgment that would otherwise be difficult to encapsulate as optimization constraints.

The remainder of this manual describes the configuration and usage of OSTRICH. Sections 1.1 and 1.2 provide brief summaries of the currently supported optimization and calibration algorithms (Section 1.1), and regression statistics and diagnostics (Section 1.2). Sections 2 through 2.23 describe the OSTRICH input file and its various configuration sections. The majority of these sections are optional and others will only be processed when a specific algorithm, objective function (i.e. calibration vs. constrained optimization vs. multi-objective optimization), or feature is activated. Section 3 describes the preparation of template model input files that OSTRICH uses to communicate with a given model. Section 4 provides guidance on running OSTRICH in serial or parallel, and Section 5 describes various output files generated by OSTRICH. Finally, Section 6 reviews the set of examples that accompany the OSTRICH distribution and provides instructions for running them on a Windows-based machine.

# Calibration and Optimization Algorithms

OSTRICH implements numerous algorithms. Some of these algorithms are deterministic ***local search*** methods, others are heuristic ***global search*** methods that incorporate elements of structured randomness, and others act as ***samplers*** that seek to delineate parameter probability distributions rather than just identifying a single optimal parameter set. While most of these algorithms are suitable for both calibration and optimization problems, one deterministic algorithm (i.e. Levenberg-Marquardt) is tailored to non-linear least-squares calibration problems. Additionally, two algorithms (i.e. Pareto Archive Dynamically Dimensioned Search and the Simple Multi-Objective Optimization Test Heuristic) are suitable for multi-objective optimization or multi-criteria calibration. Finally, several sampling-based algorithms (i.e. Generalized Likelihood Uncertainty Estimation, Rejection Sampling, and Metropolis-Hastings Markov Chain Monte Carlo) are suitable for uncertainty-based calibration. Overall, these algorithms provide the user with a fair degree of flexibility and enable OSTRICH to tackle a variety of linear and non-linear problems. Furthermore, these problems can have continuously varying (i.e. real-valued) parameters, combinatorial parameters, integer parameters, or a mixture of continuous, combinatorial and integer parameters.

***Table 1***summarizes each algorithm implemented in OSTRICH along with appropriate references for detailed descriptions. Algorithms where only contact information is provided are unpublished experimental algorithms and should be used with caution. Some algorithms have been validated against reference implementations. These algorithms include: DDS, PADDS, PSO, GML, BGA, RGA, SA, FLRV, POWL, STPDSC, and SCE. C/C++ programmers will find it straightforward to extend OSTRICH to include additional search algorithms. Doing so involves extension of an abstract base class (AlgorithmABC) that defines a minimum set of required search algorithm functions, including: Calbrate(), Optimize(), WriteMetrics(), WarmStart(), and Destroy(). These functions typically utilize additional classes that encapsulate parameters (i.e. ParameterGroup), models (i.e. ModelABC), and objective functions (i.e. ObjectiveFunction) which are dynamically instantiated based on a user-supplied configuration file (osIn.txt).

**Table 1: Catalog of Algorithms Implemented in OSTRICH**

| Acronym | Algorithm | # Objectives | Serial? | Parallel? | Warm Start? | Pre-Emption? | Parameter Correction? | Initial Parameters? | Math and Stats? | Line Search? | Reference or Contact Information |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Deterministic (Local Search) Algorithms | | | | | | | | | | | |
| BISECT | Bisection Algorithm | 1 |  |  |  |  |  |  |  |  | Matott, Tolson et al (2014) |
| FLRV | Fletcher-Reeves | 1 |  |  |  |  |  |  |  |  |  |
| GML | Gauss-Marquardt-Levenberg | 1 |  |  |  |  |  |  |  |  |  |
| MSGML | Multi-Start GML with Trajectory Repulsion | 1 |  |  |  |  |  |  |  |  |  |
| GRID | Grid-based Exhaustive Search | 1 |  |  |  |  |  |  |  |  | [lsmatott@buffalo.edu](mailto:lsmatott@buffalo.edu) |
| POWL | Powell's Algorithm | 1 |  |  |  |  |  |  |  |  |  |
| STPDSC | Steepest Descent | 1 |  |  |  |  |  |  |  |  |  |
| Heuristic (Global Search) Algorithms | | | | | | | | | | | |
| APPSO | Asynchronous Parallel Particle Swarm Optimization | 1 |  |  |  |  |  |  |  |  |  |
| BEERS | Balanced Exploration-Exploitation Random Search | 1 |  |  |  |  |  |  |  |  | [lsmatott@buffalo.edu](mailto:lsmatott@buffalo.edu) |
| BGA | Binary-coded Genetic Algorithm | 1 |  |  |  |  |  |  |  |  |  |
| CSA | Combinatorial Simulated Annealing | 1 |  |  |  |  |  |  |  |  |  |
| DDDS | Discrete DDS | 1 |  |  |  |  |  |  |  |  |  |
| DDS | Dynamically Dimensioned Search | 1 |  |  |  |  |  |  |  |  |  |
| PDDS | Asynchronous Parallel DDS | 1 |  |  |  |  |  |  |  |  |  |
| PSO | Particle Swarm Optimization | 1 |  |  |  |  |  |  |  |  |  |
| RGA | Real-coded Genetic Algorithm | 1 |  |  |  |  |  |  |  |  |  |
| SA | Simulated Annealing | 1 |  |  |  |  |  |  |  |  |  |
| SCE | Shuffled Complex Evolution | 1 |  |  |  |  |  |  |  |  |  |
| SMPLR | Sampling Algorithm (Big Bang - Big Crunch) | 1 |  |  |  |  |  |  |  |  |  |
| VSA | Vanderbilt-Louie Simulated Annealing | 1 |  |  |  |  |  |  |  |  |  |
| Multi-Objective Optimization and Multi-Criteria Calibration Algorithms | | | | | | | | | | | |
| PADDS | Pareto Archived DDS | 2+ |  |  |  |  |  |  |  |  |  |
| ParaPADDS | Asynchronous Parallel PADDS | 2+ |  |  |  |  |  |  |  |  |  |
| SMOOTH | Simple Multi-Objective Optimization Test Heuristic | 2+ |  |  |  |  |  |  |  |  | [lsmatott@buffalo.edu](mailto:lsmatott@buffalo.edu) |
| Hybrid (Heuristic + Deterministic) Algorithms | | | | | | | | | | | |
| PSO-GML | PSO with GML Polishing | 1 |  |  |  |  |  |  |  |  |  |
| Sampling Algorithms (Uncertainty-based Optimization) | | | | | | | | | | | |
| DDS-UA | DDS for Uncertainty Approximation | 1 |  |  |  |  |  |  |  |  |  |
| GLUE | Generalized Likelihood Uncertainty Estimation | 1 |  |  |  |  |  |  |  |  |  |
| MCMC | Metropolis-Hastings Markov Chain Monte Carlo | 1 |  |  |  |  |  |  |  |  |  |
| RJSMP | Rejection Sampling | 1 |  |  |  |  |  |  |  |  |  |

# Regression Statistics and Diagnostics

When used for model calibration via a weighted sum of squared errors approach, OSTRICH can compute an extensive suite of post-calibration statistics. OSTRICH can also compute a variety of diagnostic measures that test the validity of the underlying assumptions of the aforementioned statistical measures. ***Table 2*** summarizes the regression statistics and diagnostics implemented in OSTRICH along with appropriate references containing more detailed descriptions. The OSTRICH manual for version 1.6 also contains a detailed description of many of the regression statistics and diagnostics listed in ***Table 2***.

**Table 2: Regression Statistics and Diagnostics Implemented in OSTRICH**

| Name of Statistic or Diagnostic | Reference |
| --- | --- |
| Residuals | |
| Autorun Function on Ordered Residuals |  |
| Correlation of Multiple Determination (Ry) |  |
| Normal Probability Plot Correlation Coefficient (R2N) |  |
| Normal Probability Plot Points |  |
| List of Observation Residuals |  |
| Runs Test on Ordered Residuals |  |
| Parameters | |
| Estimated Parameter Values |  |
| Linear Confidence Intervals on Parameters |  |
| Parameter Correlation |  |
| Parameter Standard Deviation |  |
| Parameter Standard Error |  |
| Parameter Variance and Covariance |  |
| Local Sensitivity Analysis | |
| 1% Scaled Sensitivities of Parameters (1%SS) |  |
| Composite Scaled Sensitivity of Parameters (CSS) |  |
| Dimensionless Scaled Sensitivity of Parameters (DSS) |  |
| Jacobian Matrix (J) |  |
| Observation Influence | |
| Cook's D Measure |  |
| DFBETAS Measure |  |
| Observation Leverage |  |
| Predictive Uncertainty Analysis | |
| Linear Confidence Intervals on Predictions |  |
| Model Ranking, Transformation, and Goodness-of-Fit | |
| Akaike Information Criterion (AIC) |  |
| Bayesian Information Criterion (BIC) |  |
| Corrected Akaike Information Criterion (AICc) |  |
| Estimated Box-Cox Transformation |  |
| Hannon-Quinn Information Criterion (HQ) |  |
| Standard Error (s) |  |
| Variance (s2) |  |
| Tests of the Linearity Assumption | |
| Beale's Linearity Measure |  |
| Linssen's Linearity Measure |  |

# ostIn.txt – the OSTRICH Input File

This section summarizes the input file of the OSTRICH program. On case-sensitive Linux systems, the input file must be named ostIn.txt. On Windows systems the file could also be named OstIn.txt. OSTRICH is a command-line console driven tool and when launched it will look for ostIn.txt in the working directory (i.e. the directory from which OSTRICH is launched). If this file does not exist or if it contains syntax errors, OSTRICH will quickly recognize this and report an error message and close. Windows users will experience this behavior as a brief flash of the DOS console window as it opens and then rapidly closes. In fact, the open-close sequence may happen so fast that all a user notices is a brief flicker on the computer monitor. This does not mean that OSTRICH is not installed correctly! It just means that you didn’t create a valid input file prior to running OSTRICH. The output file named “OstErrors0.txt” will have details on why OSTRICH failed to run.

For OSTRICH to work with a given modeling program, the modeling program must meet the following requirements:

* The modeling program must use a text-based input/output file format. OSTRICH can also work with modeling programs that use the MS Access or NetCDF file formats, but users will need to configure an additional section of the OSTRICH input file. This section is described in Section 2.17 (**Type Conversions**).
* The modeling program must be able to run without prompting for user intervention. This means, for example, that the modeling program cannot prompt the user to enter the name of an input file and the modeling program must not pause for user input at the end of a simulation.
* The output of the modeling program must be in a consistent format that can be reliably parsed. OSTRICH can also work with modeling programs that ***sometimes*** fail to write consistently formatted output. In such cases users should configure the optional “OnObsError” feature described in Section 2.3 (**Basic Configuration**).

OSTRICH utilizes a text-based input file format which specifies that configuration variables be organized on a line-by-line basis using loosely human-readable syntax. Users typically prepare the OSTRICH input file using a text editor like Notepad, Wordpad, VIM, or Emacs. For some sections (e.g. observations and response variables) it may also be helpful to use a spreadsheet program like Excel or Calc and then copy the desired cells from the spreadsheet to the text-based input file.

With a few exceptions (which will be explicitly noted in the following text) the basic format for a line of input in the ostIn.txt file is:

**<variable> <value>**

Where **<variable>** is the name of the configuration variable (e.g. **ProgramType**) and **<value>** is the user-selected value for the variable (e.g. **ParticleSwarm**). The whitespace separating **<variable>** and **<value>** can be any number of spaces or tab characters. Inside **ostIn.txt**, the OSTRICH configuration variables are organized into groups and each group is described below in its own section.

Although the list of **ostIn.txt** configuration groups is rather extensive, most of the groups do not need to be specified, as they are initialized within OSTRICH to reasonable defaults if the user does not set a value for them. Furthermore, many of the configuration groups relate to optional features within OSTRICH and may not be used in a given run of the program. In fact, the only groups that must be configured by the user are: Basic Configuration, File Pairs, and Parameters. You must also include an Observations group if calibrating using OSTRICH’s internal weighted least squares objective function. Otherwise, if using OSTRICH’s general-purpose constrained optimization platform (GCOP), you must include a Response Variables group, a Costs group, and a Constraints group. Sections 2.3 through 2.23 discuss the particular syntax and purpose of the various groups that may be included in the ostIn.txt file.

# Comments

Comment lines in the OSTRICH input files have the ’#’ symbol as the first character. These lines are ignored by the OSTRICH input file parser, and allow the user to supply additional information that can make the input file more readable. Additionally, comments allow the user to disable configuration parameters and/or observations without completely deleting the corresponding lines. A sample comment line is given below in ***Listing 1***. More examples can be found in the demonstration files distributed with the OSTRICH program and these are described in Section 6.

#

# These are some example comment lines. It’s a good

# idea to include comments in the input file to

# describe the intent of your configuration

# choices.

#

**Listing 1: Example Comment Lines**

# Case Sensitivity

Variable **names** and **group tags** in the OSTRICH input file are case sensitive; e.g. using **beginfilepairs** instead of **BeginFilePairs** will result in a parsing error. Meanwhile, **values** of variables are case insensitive; e.g. **GENETICALGORITHM**, **geneticalgorithm**, and **GeneticAlgorithm** will all correctly select the genetic algorithm **ProgramType**.

# ostIn – Basic Configuration

The “Basic Configuration” variables describe the modeling program that is to be optimized or calibrated and identify the optimization (or regression) algorithm that OSTRICH should use. In addition, there are a number of optional basic configuration variables that effect various aspects of the OSTRICH program. ***Listing 2*** summarizes the syntax for the variables that make up the basic configuration group. The third column of text is enclosed in brackets (i.e. “[ ]”) and provides the default settings for each variable. Only the first two columns (i.e. variable and desired value) should be included in an actual input file. An example syntactically correct configuration of the basic group is given in ***Listing 4***. Users can “cut-and-paste” ***Listing 4*** and edit as needed for their particular problem. ***Listing 3*** and ***Listing 5*** provide list of possible values for the TelescopingStrategy and ProgramType variables, respectively. Users interested in the details of the telescoping strategies are referred to the publication by Matott et al [ref].

# essential variables

ProgramType see\_listing\_5 [Levenberg-Marquardt]

ModelExecutable name\_of\_model [no default]

ModelSubdir name\_of\_subdir [.]

ObjectiveFunction wsse/gcop [wsse]

# useful optional variables

PreserveBestModel name\_of\_script [no default]

PreserveModelOutput yes/no [no]

OstrichWarmStart yes/no [no]

NumDigitsOfPrecision val\_from\_1\_to\_32 [6]

TelescopingStrategy see\_listing\_3 [none]

RandomSeed value [randomly assigned]

OnObsError quit/value [quit]

# experimental or less common optional variables

CheckSensitivities yes/no [no]

SuperMUSE yes/no [no]

OstrichCaching yes/no [no]

BoxCoxTransformation value [1.00]

**Listing 2: Basic Configuration Group**

Values for the “ModelExecutable” and “PreserveBestModel” variables can include be fully qualified paths or relative paths and should reference an executable file, batch file, or script file. If a path or file contains spaces the value should be enclosed in double quotes (i.e. “ “).

***Note: the basic configuration group is the only group in the OSTRICH input file that does not have a corresponding “*Begin…*” and “*End…*” group tag. As such these variables can be placed anywhere within the input file. However, since these are the first variable processed by OSTRICH, a good convention to follow is to place these variables at the beginning of the file and avoid mixing them in with the other groups.***

**ProgramType**: This variable tells OSTRICH which algorithm should be used to perform the optimization or calibration.

**ModelExecutable**: Specifies the model executable or driver program or script. If the executable is in the same directory as the working directory from which the program is executed, then the path information may be omitted.

**ModelSubdir**: When running in parallel, users must specify a working subdirectory to prevent parallel runs from clobbering each other’s input and output files. If set to any value other than ’.’ (i.e. the default), the value of ModelSubdir will cause OSTRICH to create unique subdirectories for the model runs of each parallel processor. The subdirectory names are created by concatenating the ModelSubdir value with each processors MPI id number.

**ObjectiveFunction**: The objective function to be optimized, either WSSE (weighted sum of squared error) calibration or GCOP (General-purpose Constrained Optimization Platform).

**PreserveBestModel**: A user-supplied script of executable that is run by OSTRICH every time a new best parameter set is discovered.

**PreserveModelOutput**: If set to "yes" OSTRICH will make copies of files associated with each model run. Preserved files will be stored directories named "runNNN", when NNN is a counter that is incremented after each model run. For example, the files for the first model run will be copied into a directory named run1, and files from the second run copied into a directory named run2, and so on.

**OstrichWarmStart**: If set to "yes" OSTRICH will read the contents of any previously created "OstModel" output files and use the entries therein to restart an optimization or calibration exercise.

**NumDigitsOfPrecision**: This specifies the precision of values written to OSTRICH output files.

**TelescopingStrategy**: If selected, this optional setting will cause parameter bounds to become incrasingly smaller as an optimization or calibration proceeds.

# Options for TelescopingStrategy

# ===========================================================

#none

#convex-power

#convex

#linear

#concave

#delayed-concave

**Listing 3: Supported Values for the Telescoping Strategy Option**

**RandomSeed**: This variable can be used to control the random seed OSTRICH uses when generating random numbers.

**OnObsError**: This variable controls how OSTRICH behaves when a model fails to generate all of the expected output for a WSSE calibration. If set to "quit", OSTRICH will abort if it ever fails to parse an observation from user-specified output files. If set to a value, OSTRICH will use the value as a placeholder observation value if it can't read a given observation from model output.

**CheckSensitivities**: If this variable is set to "yes", OSTRICH will perform a pre-calibration step to calculate parameter sensitivities (i.e. changes in simulated equivalent observations with respect to changes in parameters).

**SuperMUSE**: If set to "yes", OSTRICH will interface with EPA SuperMUSE tasker-client approach to parallel computing.

**OstrichCaching**: If set to "yes", OSTRICh will examine "OstModel" output files prior to running a given model configuration to see if the associated parameter set has already been evaluated.

**BoxCoxTransformation**: If set to a value other than "1", OSTRICH will apply a Box-Cox power transformation on each calibration residual. The user-supplied value is used as the exponent for the transformation.

# essential variables

ProgramType ParticleSwarm

ModelExecutable “C:\My Folder\My\_Model.exe”

ModelSubdir mod

ObjectiveFunction GCOP

# useful optional variables

PreserveBestModel “C:\My Folder\Save\_Best.bat”

PreserveModelOutput no

OstrichWarmStart yes

NumDigitsOfPrecision 8

TelescopingStrategy none

RandomSeed 100

OnObsError quit

# experimental or less common optional variables

CheckSensitivities yes

SuperMUSE no

OstrichCaching no

BoxCoxTransformation 1.00

**Listing 4: Example of a Syntactically Correct Basic Configuration Group**(note: variables set to default values could be omitted or commented out)

# Options for ProgramType Algorithm Description

# ============================ ===============================

# GeneticAlgorithm See Table 1 (RGA)

# BinaryGeneticAlgorithm See Table 1 (BGA)

# ShuffledComplexEvolution See Table 1 (SCE)

# BisectionAlgorithm See Table 1 (BIS)

# SamplingAlgorithm See Table 1 (BBBC)

# ParticleSwarm See Table 1 (PSO)

# APPSO See Table 1 (APPSO)

# PSO-GML See Table 1 (PSO-GML)

# SimulatedAnnealing See Table 1 (CSA)

# DiscreteSimulatedAnnealing See Table 1 (DSA)

# VanderbiltSimulatedAnnealing See Table 1 (VSA)

# Levenberg-Marquardt See Table 1 (GML)

# GML-MS See Table 1 (MSGML)

# Powell See Table 1 (POWL)

# Steepest-Descent See Table 1 (STPDSC)

# Fletcher-Reeves See Table 1 (FLRV)

# RegressionStatistics Compute regression statistics

# Jacobian Compute Jacobian matrix

# Hessian Compute Hessian matrix

# Gradient Compute Gradient information

# ModelEvaluation Process InitParams group

# GridAlgorithm See Table 1 (GRID)

# DDS See Table 1 (DDS)

# ParallelDDS See Table 1 (PDDS)

# DiscreteDDS See Table 1 (DDDS)

# GLUE See Table 1 (GLUE)

# RejectionSampler See Table 1 (RJSMP)

# MetropolisSampler See Table 1 (MCMC)

# SMOOTH See Table 1 (SMOOTH)

# PADDS See Table 1 (PADDS)

# ParaPADDS See Table 1 (ParaPADDS)

# BEERS See Table 1 (BEERS)

**Listing 5: Supported Values for the Program Type Option**

# ostIn – File Pairs

A file pair consists of a template file and a corresponding model input file. The contents of the template file should be identical to the paired model input file except that values of optimization (or calibration) parameters are replaced with unique parameter names defined in the Parameters section. During optimization, OSTRICH uses the template files to create syntactically correct model input files in preparation of running the model at different parameter values. Section 3 describes this process in detail. The general syntax for the File Pair group is given in Listing 6 and a concrete example is given in Listing 7.

BeginFilePairs

<template1><sep><input1>

<template2><sep><input2>

.

.

.

<templateN><sep><inputN>

EndFilePairs

**Listing 6: General Format for the File Pairs Group**

As shown in Listing 6, **BeginFilePairs** and **EndFilePairs** are parsing tags that wrap a list of file name pairs such that <template1> ... <templateN> are the names of the template files corresponding to the <input1> ... <inputN> model input files, and <sep> is a separator that tells OSTRICH when one filename ends and the next begins. Valid file name separators are the semi-colon character ’;’ and the TAB character. Spaces are not valid separator characters because OSTRICH allows spaces within file names.

BeginFilePairs

Wells.tpl ; Ledom.wel

kvalues.tpl ; Ledom.lpf

recharge.tpl ; Ledom.rch

EndFilePairs

**Listing 7: Example Implementation of the File Pairs Group**

# ostIn – Extra Files

Extra files are model input files not used by OSTRICH, but required for proper execution of the model. In parallel environments, OSTRICH needs to know about these extra input files so that it can copy them to each processor’s working directory (see ModelSubdir in Section 2.3, above). Sharing a working directory among parallel processors is not recommended because it can result in multiple processors trying to write to the same file at the same time. The general syntax for the Extra Files group is given in Listing 8 and a concrete example is given in Listing 9.

BeginExtraFiles

<file1>

<file2>

.

.

.

<fileN>

EndExtraFiles

**Listing 8: General Format for the Extra Files Group**

As shown in Listing 8, **BeginExtraFiles** and **EndExtraFiles** are parsing tags that wrap a list of extra model input files. Extra files must be identified if the model is to be executed in a dynamically generated subdirectory (as specified by the ModelSubdir variable), so that OSTRICH knows to copy them to the subdirectory. For serial algorithms, creation of a dynamic subdirectory is unnecessary and specification of the extra files section is optional. However, this section is required if running a parallel algorithm to avoid aforementioned processor I/O conflicts.

BeginExtraFiles

Ledom.nam

Ledom.bas

Ledom.dis

Ledom.pcg

EndExtraFiles

**Listing 9: Example Implementation of the Extra Files Group**

# ostIn – Extra Directories

Extra directories are directories containing model input files not used by OSTRICH, but required for proper execution of the model. In parallel environments, OSTRICH needs to know about these extra directories so that it can copy them (and all files and subdirectories contained within) to store in each processors working directory (as specified by the ModelSubdir variable). Sharing a working directory among parallel processors is not recommended because it can result in multiple processors trying to write to the same file of the same directory at the same time. Listings 10 and 11 contain the general syntax and a concrete example of the Extra Directories group, respectively. As shown in Listing 10, **BeginExtraDirs** and **EndExtraDirs** are parsing tags that wrap a list of extra model input directories.

BeginExtraDirs

<dir1>

<dir2>

.

.

.

<dirN>

EndExtraDirs

**Listing 10: General Format for the Extra Directories Group**

BeginExtraDirs

HUC\_001

HUC\_002

HUC\_003

HUC\_004

HUC\_005

HUC\_006

EndExtraDirs

**Listing 11: Example Implementation of the Extra Directories Group**(e.g. for a hypothetical model that processes 6 watersheds)

# ostIn – Real-valued Parameters

This configuration group describes the parameters to be calibrated or optimized. Parameter configuration variables include names, initial values, lower and upper bounds, input, output and internal transformations, and (optionally) fixed format printing codes. Parameters in this section are real and continuously varying. Listing 12 provides the general format for the parameters group and Listing 13 gives a concrete example.

BeginParams

<name1> <init1> <lwr1> <upr1> <txIn1> <txOst1> <txOut1> <fmt1>

<name2> <init2> <lwr2> <upr2> <txIn2> <txOst2> <txOut2> <fmt2>

. . .

<nameN> <initN> <lwrN> <uprN> <txInN> <txOstN> <txOutN> <fmtN>

EndParams

**Listing 12: General Format for the Real-valued Parameters Group**

In Listing 12, **BeginParams** and **EndParams** are parsing tags that wrap a list of *N* model parameters made up of the following variables:

**name**: The name of the parameter, parameter names must be unique and correspond identically to the names used in the template file(s) (see Section 2.4 and Section 3).

**init**: Initial value of the parameter, in units specified by the **txIn** variable. Alternatively, the keywords “**random**” or “**extract**” may be used instead of specifying a value. OSTRICH will assign a randomly generated initial value if the “**random**” keyword is used. OSTRICH will extract the initial value from existing model input files if the “**extract**” keyword is used.

**lwr**: Lower bound (i.e. minimum value) of the parameter, in units specified by the **txIn** variable.

**upr**: Upper bound (i.e.. maximum value) of the parameter, in units specified by the **txIn** variable.

**txIn**, **txOst**, and **txOut**: These specify the type of transformation units that OSTRICH should use. Transformations allow the user to take advantage of any linearity relationships that exist between a transformed parameter value (e.g. log10 or loge) and the underlying model. Three kinds of transformations are provided so that the user can work with input and output transformations that are different than the internal transformation. Typically, the user will request no input and output transformation (so that input and output values are the native units of the parameter), while instructing OSTRICH to perform a transformation internally. This approach allows the algorithm to take advantage of a transformed relationship without requiring manual conversion of input and output values. However, it should be noted that some statistical output is reported in terms of **txOst** units, regardless of the value of **txOut**; namely (a) parameter variance-covariance, (b) observation influence, (c)parameter sensitivity, (d) model linearity, and (e) matrices. OSTRICH supports the following transformation values:

– **none**: no transformation.

– **log10**: log base 10 transformation.

– **ln**: natural logarithm transformation.

**fmt**: A format code that OSTRICH will use when writing model input files. This is provided so that OSTRICH can support modeling programs which expect fixed format inputs (i.e. when values in the input file are expected to take up an exact number of characters). For example, many programs written in legacy FORTRAN (e.g. F77) expect fixed format. Use a **fmt** value of “**free**” if using a modeling program that is not bound by fixed format requirements. Otherwise, use a format code of “**Fw.d**” for decimal values (e.g. 3.4567) where “**w”** is the total number of characters and “**d”** is the number of characters following the decimal. For example, to represent the value of Pi to 6 significant digits you would use a format code of **F8.6**, resulting in a value of “3.141593”. Use a format code of “**Ew.d**” or “**Dw.d**” for scientific notation, where “**w**” is the total number of characters and “**d**” is the number of significant digits. For example, applying a format code of **E10.3** to the value of 1/12 would result in “ 8.333E-02”. For fixed decimal notation “**w**” should be at least equal to “**d**”+2 and for fixed scientific notation “**w**” should be at least equal to “**d**”+7.

BeginParams

\_DIAM\_ random 10.0 50.0 none none none free

\_LEN\_ random 200.0 1000.0 none none none free

EndParams

**Listing 13: Example of the Real-valued Parameters Group**

# ostIn – Integer Parameters

This configuration group describes those parameters to be calibrated or optimized which can take on only integer values. Like their real-parameter counterparts, integer parameter configuration variables include names, initial values, and lower and upper bounds. However, format codes and unit transformations are not supported for integer parameters. Listings 14 and 15 provide the general syntax and a concrete example of the integer parameters group, respectively.

BeginIntegerParams

<name1> <init1> <lwr1> <upr1>

<name2> <init2> <lwr2> <upr2>

. . .

<nameN> <initN> <lwrN> <uprN>

EndIntegerParams

**Listing 14: General Format for the Integer Parameters Group**

BeginIntegerParams

N\_INJ\_WELLS 2 0 20

N\_EXT\_WELLS 6 0 50

EndIntegerParams

**Listing 15: Example of the Integer Parameters Group**

# ostIn – Combinatorial Parameters

This configuration group describes those parameters to be calibrated or optimized which can take on a discrete set of values, which can be in the form of real, integer or string (text) values. Like integer and real parameters, combinatorial parameter configuration variables include names and initial values; but instead of lower and upper bounds, the user must supply a complete list of the discrete values that may be assigned to the parameter. Furthermore, format codes and unit transformations are not supported for combinatorial parameters. Listing 16 provides the general syntax of the combinatorial parameters group.

BeginCombinatorialParams

<name1> <type1> <init1> <N1> <v1,1> <v1,2> ... <v1,N1>

<name2> <type2> <init2> <N2> <v2,1> <v2,2> ... <v2,N2>

. . .

<nameM> <typeM> <initM> <NM> <vM,1> <vM,2> ... <vM,NM>

EndCombinatorialParams

**Listing 16: General Format for the Combinatorial Parameters Group**

In Listing 16, the “**type**” field should be either “**real**”, “**integer**”, or “**string**” and should correspond to the type of values in the subsequent combinatorial list. Furthermore, the “**N1**” through “**NM**” values specify the number of entries in the combinatorial list, which is generically represented in Listing 14 as **vm,n** for the nth discrete value that can be taken on by the mth parameter. Listing 17 provides a concrete example of the combinatorial parameters group.

BeginCombinatorialParams

COLOR string blue 5 red orange yellow green blue

BOLTS real 0.25 4 0.0625 0.125 0.25 0.5

PRIME integer 1 10 1 3 5 7 11 13 17 19 23 29

EndCombinatorialParams

**Listing 17: Example of the Combinatorial Parameters Group**

# ostIn – Tied Parameters

Tied parameters are parameters which are computed as a function of integer, real or combinatorial parameter values. They may also be functions of other tied parameters.

 (1)

Where, *Xtied* is the tied parameter value which is a function of *n* non-tied parameters (*X1*,*X2*,...*Xn*) and a set of *m* coefficients (*c1*,*c2*,...*cm*), which depend on the functional form of *ftied*(). Tied parameter configuration variables include: the name of the tied parameter; a list of the names of tied or non-tied parameters used in the computation of the tied-parameter value; a specification of the functional form of *ftied*(); and a list of coefficients used in the evaluation of *ftied*(). Listing 18 provides the general syntax for the tied parameters group.

BeginTiedParams

<name1> <np1> <pname1,1> <pname1,2> ... <pname1,np1> <type1> <type\_data1>

<name2> <np2> <pname2,1> <pname2,2> ... <pname2,np2> <type2> <type\_data2>

. . .

<nameN> <npN> <pnameN,1> <pnameN,2> ... <pnameN,npN> <typeN> <type\_dataN>

EndTiedParams

**Listing 18: General Format for the Tied Parameters Group**

In Listing 18, **BeginTiedParams** and **EndTiedParams** are parsing tags that wrap a list of tied model parameters made up of the following variables:

**name**: The name of the tied parameter, parameter names must be unique and correspond identically to the corresponding name used in the template file(s).

**np** : The number of non-tied parameters used in the calculation of the tied parameter value. Valid values for **np** depend on the choice of functional relationship, specified in the **type** field.

**pname1** … **pnamenp**: A list of parameter names that are used in the computation of the tied-parameter.

**type**: The type of functional relationship ,*ftied*(), between the tied parameter and the list of named parameters (i.e. **pname1** … **pnamenp)**. Valid values for **type** are:

**linear**: Selects a linear relationship for *ftied*(). If this choice is selected, the value of **np** must be either 1 or 2.

**exp**: Selects an exponential relationship for *ftied*(). If this choice is selected, the value of **np** must be 1.

**log**: Selects a log relationship for *ftied*(). If selected, the value of **np** must be 1.

**dist**: The tied parameter is the distance between two (x,y) coordinates, where these coordinates are parameters of the optimization/calibration. If selected, the value of **np** must be 4 and the ordering of parameter names should correspond to (x1,y1),(x2,y2).

**wsum**: The tied parameter is the weighted sum of the listed parameters.

**ratio**: The tied parameter is the ratio of a linear combination of parameters. If selected, the value of **np** must be 2 or 3.

**constant**: The tied parameter is a constant. If selected, the value of **np** must be 0.

**type\_data**: Depending on the choice of **type**, the syntax of this field varies, as described below. The syntax for **type\_data** includes a format specifier – see the description of the **fmt** variable in Section 2.7.

1. If **type** = ”linear” and **np** = "1" : The functional relationship is linear and has the form:

Xtied = (c1 × X) + c0

where Xtied is the tied-parameter value, c0 and c1 are coefficients, X is the non-tied parameter value, and **type\_data** should be replaced with the following syntax:

<c1> <c0> <fmt>

1. If **type** = ”linear” and **np** = "2" : The functional relationship has the form:

Xtied = (c3 × X1 × X2) + (c2 × X2) + (c1 × X1) + c0

where Xtied is the tied-parameter value, c0, c1, c2, and c3 are coefficients, X1 and X2 are the non-tied parameter values, and **type\_data** should be replaced with the following syntax:

<c3> <c2> <c1> <c0> <fmt>

1. If **type** = ”exp” : The functional relationship has the form:

Xtied = c2 × b(c1 × X) + c0

where Xtied is the tied-parameter value, c0, c1 and c2 are coefficients, b is the exponent base, X is the non-tied parameter value, and **type\_data** should be replaced with the following syntax:

<base> <c2> <c1> <c0> <fmt>

where **base** can be a numerical value, or “exp” if the natural base is to be used.

– If **type** = ”log”: The functional relationship has the form:

Xtied = c3 × loga(c2 × X + c1) + c0

where Xtied is the tied-parameter value, c0, c1, c2 and c3 are coefficients, a is the logarithm base, X is the non-tied parameter, and **type\_data** should be replaced with the following syntax:

<base> <c3> <c2> <c1> <c0> <fmt>

where **base** can be a numerical value, or “ln” if the natural logarithm is to be used.

– If **type** = ”dist”: The **type\_data** field should contain the desired **fmt** specification.

– If **type** = ”wsum”: The **type\_data** field should list the values of each weight, using the

same ordering as the named list of parameters, followed by the desired **fmt** specification.

– If **type** = ”ratio” and **np** = “2”: The functional relationship has the form:

Xtied = (c3 × X1 + c2) / (c1 × X2 + c0)

where Xtied is the tied-parameter value, c3, c2, c1 and c0 are coefficients, X1 and X2 are non-tied parameters, and **type\_data** should be replaced with the following syntax:

<c3> <c2> <c1> <c0> <fmt>

– If **type** = ”ratio” and **np** = “3”: The functional relationship has the form:

Xtied = [ (n7 × X1 × X2 × X3) + (n6 × X1 × X2) + (n5 × X1 × X3) +   
 (n4 × X2 × X3) + (n3 × X1) + (n2 × X2) + (n1 × X3) + n0 ] /   
 [ (d7× X1 × X2 × X3) + (d6 × X1 × X2) + (d5 × X1 × X3) +   
 (d4 × X2 × X3) + (d3 × X1) + (d2 × X2) + (d1 × X3) + d0 ]

where Xtied is the tied-parameter value, n7 … n0 and d7 … d0 are coefficients, X1 … X3 are non-tied parameters, and **type\_data** should be replaced with the following syntax:

n7 n6 n5 n4 n3 n2 n1 n0 d7 d6 d5 d4 d3 d2 d1 d0 fmt

– If **np** = “0”: The tied parameter is assigned a constant value. No **type** field is required and the **type\_data** field must contain the parameter value followed by a format specifier (**fmt**).

Listing 19 provides concrete examples of the different tied parameter types.

BeginTiedParams

# 1-parameter linear (TLIN = 2\*XVAL)

TLIN 1 XVAL linear 2.00 0.00 free

# 2-parameter linear (TLN2 = 2\*XVAL + YVAL)

TLN2 2 XVAL YVAL linear 0.00 2.00 1.00 0.00 free

# exponent, base e (TEXP = exp(-XVAL))

TEXP 1 XVAL exp exp 1.00 -1.00 0.00 free

# exponent, base 10 (TX2P = 10^(-XVAL))

TXP2 1 XVAL exp 10.0 1.00 -1.00 0.00 free

# logarithm, natural log (TLOG = 2\*LN(XVAL))

TLOG 1 XVAL log ln 2.00 -1.00 0.00 0.00 free

# logarithm, base 2 (TLG2 = log2(XVAL/2)+1)

TLG2 1 XVAL log 2.00 1.00 0.50 0.00 1.00 free

# distance

TDST 4 X1VAL Y1VAL X2VAL Y2VAL dist free

# weighted sum (TSUM = (1/3)\*(XVAL+YVAL+ZVAL))

TSUM 3 XVAL YVAL ZVAL wsum 0.33 0.33 0.33 free

# 2-parameter ratio (TRAT = (XVAL / YVAL))

TRAT 2 XVAL YVAL ratio 1.00 0.00 1.00 0.00 free

# 3-parameter ratio (TRT3 = (XVAL\*YVAL)/(ZVAL+1))

TRT3 3 XVAL YVAL ZVAL ratio 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 free

# constant (Pi)

TPI 0 3.1415 free

EndTiedParams

**Listing 19: Example of the Tied Parameters Group**

# ostIn – Special Parameters (pre-emption)

Certain models are capable of monitoring the progress of a simulation and aborting further processing if some threshold cost or constraint is exceeded. OSTRICH provides the “SpecialParams” group to support such models. Special parameters are cost and constraint thresholds that are tracked by selected algorithms in OSTRICH (see the relevant column in ***Table 1***, above) and written to input files using the same template mechanism as regular calibration/optimization parameters. In this way OSTRICH can pass the most up to date threshold values on to the pre-emptive model. Pre-emption is described in detail by [ref]. The general syntax for the SpecialParams group is given below in Listing 20 and a concrete example is given in Listing 21.

BeginSpecialParams

<name1> <init1> <type1> <con\_type1> <con\_name1>

<name1> <init1> <type1> <con\_type1> <con\_name1>

.

.

.

<name1> <init1> <type1> <con\_type1> <con\_name1>

EndSpecialParams

**Listing 20: General Format of the Special Parameters Group**

In Listing 20, **BeginSpecialParams** and **EndSpecialParams** are parsing tags that wrap a list of model pre-emption parameters made up of the following variables:

**name**: The name of the pre-emption parameter, parameter names must be unique and correspond identically to the corresponding name used in the template file(s).

**init**: The initial value of the pre-emption parameter. This should be set to a value that will **NOT** trigger pre-emption.

**type**: The type of pre-emption parameter. This should be set to either “**BestCost**” or “**BestConstraint**” depending on the nature of pre-emption (i.e. model pre-emption based on exceeding the cost function or model pre-emption based on violation of a constraint threshold).

**con\_type**: For “**BestConstraint**” pre-emption parameters the “**con\_type”** value should be either “**upper**” or “**lower**”. Set the value to “**upper**” if the model should pre-empt if it’s internally computed constraint exceeds the value of the constraint specified by “**con\_name**”. Set the value to “**lower**” if the model should pre-empt if it’s internally computed constraint is less than the value of the constraint specified by “**con\_name**”. For “**BestCost**” pre-emption parameters, the “**con\_type**” and “**con\_name**” fields are ignored and should be set to “**n/a**”.

**con\_name**: The name of the constraint whose violation should trigger pre-emption. Constraints are defined in the Constraints group which, in turn, require specification of a Response Variable group --- see Sections 2.15 and 2.23, below.

BeginSpecialParams

# initial special upper or cons-

#name value parameter lower? traint

OST\_COST 1E60 BestCost n/a n/a

# pre-emption based on violation of MyPen

OST\_MASS 1E60 BestConstraint upper MyPen

EndSpecialParams

BeginConstraints

# require -9E99 < (PenRV\*100) < 10

#name type conv.fact lower upper resp.var

MyPen general 100 -9E99 10.00 PenRV

EndConstraints

BeginResponseVars

# tells OSTRICH how to extract values of PenRV from model output

#name filename keyword line col token

PenRV Simple.out ; OST\_NULL 0 3 ' '

EndResponseVars

**Listing 21: A Concrete Example of the Special Parameters Group**(for completeness, example Constraint and Response Variable groups are also provided)

# ostIn – Initial Parameters

As indicated in ***Table 1***, users of certain algorithms can optionally seed some or all of the initial search entries with predefined parameter sets. This allows the user to incorporate prior information (such as previous optimization results or expert judgement) into the optimization, and may enhance the efficiency and/or effectiveness of the algorithm. To use this option, insert an “**InitParams**” group, which uses the general syntax given in Listing 22.

BeginInitParams

p1,1 p2,1 p3,1 . . . pn,1

p1,2 p2,2 p3,2 . . . pn,2

. . .

. . .

p1,m p2,m p3,m . . . pn,m

EndInitParams

**Listing 22: General Format of the Initial Parameters Group**

Where “**BeginInitParams**” and “**EndInitParams**” are parsing tags that wrap a list of initial parameters, and **n** is the number of parameters, **m** is the number of entries in the initial parameters group, and **pi,j** is the **j**-th initial value of the **i**-th parameter (ordered according to the order of the parameters section(s)). A concrete example of the “**InitParams**” group is given in Listing 23.

BeginParams

xval 0 -20.0 +20.0 none none none

yval 0 -20.0 +20.0 none none none

zval 0 -20.0 +20.0 none none none

EndParams

BeginInitParams

#xval yval zval

0.0 0.0 0.0

0.0 0.0 10.0

0.0 10.0 0.0

10.0 0.0 0.0

20.0 20.0 20.0

EndInitParams

**Listing 23: Example of the Initial Parameters Group**(with parameters group included for completeness)

# ostIn – Parameter Correction

The “**ParameterCorrection**” group and corresponding “**Corrections**” sub-group allows users to interface OSTRICH with an external program or script that makes adjustments to a candidate parameter set that has been calculated by an OSTRICH search algorithm but not yet evaluated. These corrections allows users to incorporate expert judgment or other information into the search procedure while still using one of the algorithms already implemented within OSTRICH. As an example, consider an optimization problem that seeks to install a well in an optimal location for extracting contaminated groundwater. Parameter correction can be used to adjust candidate well locations if they are found to be outside the boundaries of the contaminated plume. To use this option, insert a “**ParameterCorrection**” group, which uses the general syntax given in Listing 24 and which includes a “**Corrections**” sub-group.

BeginParameterCorrection

Executable <name\_of\_exe>

Template <tpl\_name> ; <inp\_name>

BeginCorrections

<name1> <outfile1> ; <keyword1> <line1> <col1> '<sep1>'

<name2> <outfile2> ; <keyword2> <line2> <col2> '<sep2>'

.

.

.

<nameN> <outfileN> ; <keywordN> <lineN> <colN> '<sepN>'

EndCorrections

EndParameterCorrection

**Listing 24: General Syntax for the Parameters Correction  
Group and Corrections Sub-Group**

Where “**BeginParameterCorrection**” and “**EndParameterCorrection**” are parsing tags that wrap the configuration variables of the “**ParameterCorrection**” group and “**BeginCorrections**” and “**EndCorrections**” are parsing tags that wrap the “**Corrections**” sub-group. Configuration variables are described below:

**name\_of\_exe**: The name (including path, if desired) of the external correction program or script that implements user-defined parameter corrections.

**tpl\_name**: The name of the template file that mimics the input file used by the external parameter correction program (i.e. “**name\_of\_exe**”). The template file must contain the names of all parameters that are to be subjected to possible correction by the external program.

**inp\_name**: The name of the input file read by the “**name\_of\_exe**” parameter. OSTRICH will create this file by replacing the parameter names listed in the “**tpl\_name**” template file with actual candidate values under consideration by the search algorithm.

**name**: The name of a correctable parameter listed in the template file (i.e. “**tpl\_name**”). Each correctable parameter must be included in the **Corrections** sub-group.

**outfile**: The name of the file that will be created by the external correction program and which will contain the possibly corrected value of the parameter specified by the corresponding “**name**” field.

**keyword**: A keyword that is search for within “**outfile**” prior to extracting the possibly corrected value of the parameter specified by the corresponding “**name**” field. If no keyword search is desired, set the value of this variable to “**OST\_NULL**”.

**line**: The line number to advance to within “**outfile**” prior to extracting the possibly corrected value of the parameter specified by the corresponding “**name**” field. If “**keyword**” is set to “**OST\_NULL**” the line number is relative to the beginning of the file, otherwise the line number is relative to the first line containing the specified keyword. A line number of “0” indicates the same line as the keyword, a line number of “1” indicates the first line after the keyword, a line number of “2” indicates the second line after the keyword, and so on.

**col**: The column number within the specified line of the “**outfile**” that will contain the possibly corrected value of the parameter specified by the corresponding “**name**” field. A column number of “1” indicates the first column, a column number of “2” indicates the second column, and so on, where each column is separated by the separator character given in the “**sep**” field.

**sep**: A character that separates each column. This variable should be enclosed in single quotes (e.g. ' ' for space-separated, ',' for comma-separated, etc.).

A concrete example of the “**ParameterCorrection**” group and accompanying “**Corrections**” sub-group is given in Listing 25.

#=================================================================

# interface with an external program that will make corrections to

# the candidate parameter sets generated by the search algorithm.

# Executable --- the external program

# Template --- the template pair for creating the input file

# Corrections --- instructions for parsing the output file

#=================================================================

BeginParameterCorrection

Executable "C:\Program Files\MakeWellCorrections.bat"

Template Corrector.tpl ; input.csv

BeginCorrections

XQ01 output.csv ; OST\_NULL 0 1 ','

YQ01 output.csv ; OST\_NULL 0 2 ','

RQ01 output.csv ; OST\_NULL 0 3 ','

EndCorrections

EndParameterCorrection

**Listing 25: Example of the Parameters Correction Group and Corrections Sub-Group**

# ostIn – Observations

For calibration problems that use the internal OSTRICH weighted sum of squared errors (WSSE) objective function, the Observations group is used to list the observation names, values, and weights, along with parsing instructions for reading simulated equivalent observations from model output files. The general syntax for the Observations group is given in Listing 26.

A concrete example of the Observations group is given in Listing 27.

# ostIn – Response Variables

In this section, the user specifies the response variables that OSTRICH should read from model output files prior to evaluating costs and constraints. The syntax is very similar to the observations group used in model calibration, and includes variable name, output file name (from which the value of the variable is read), and parsing instructions for retrieving the value of the variable from the given model output file. The Constraints and GCOP sections (see below) build upon the Response and Tied Response Variable groups by associating response variables with a constraint or cost variable.

# ostIn – Tied Response Variables

In this section, the user specifies ’tied’ response variables; variables whose values are computed by OSTRICH as functions of one or more response variables and/or parameters.

# ostIn – Type Conversion (MS Access, netcdf)

Models that generate input or output files in MS Access or netcdf format can be interfaced with OSTRICH via specification of a corresponding “TypeConversion” group.

# ostIn – Search Algorithms

Each algorithm has its own configuration group, wherein the user can specify the values for various algorithm control variables.

# ostIn – Multi-Objective Search Algorithms

# ostIn – Math and Stats

The variables in this group describe the finite difference method that OSTRICH will employ, if the chosen algorithm requires such computation. These algorithms are identified in ***Table 1*** by the shaded entries in the “Math and Stats?” column. If calibration is being performed, additional variables in this group are used to request various statistical and diagnostic output.

# ostIn – Line Search

This group is used for configuration of the method (Brent or Golden-Section) and convergence criteria of the one-dimensional search algorithm that underlies an unconstrained numerical optimization procedure. Such algorithms are identified in ***Table 1*** above by the shaded entries in the “Line Search?” column.

# ostIn – General-purpose Constrained Optimization Platform (GCOP)

In this group users can specify the response variable (or variables if performing multi-objective optimization or multi-criteria calibration) that will serve as the cost function (or functions) for the general constrained optimization platform. Each “Cost Function” identifies a single response variable or tied response variable that represents a system cost (*CSYS*) to be minimized by the optimizer. The overall GCOP objective function (*FSYS*) is a combination of the system cost (*CSY S*) and a penalty function, *PTOTAL*, which accounts for the cost of all constraint violations. The OSTRICH GCOP module offers several techniques for combining *CSYS* and *PTOTAL* to form the objective function; namely the additive penalty method (APM), the multiplicative penalty method (MPM), and the exponential penalty method (EPM). The functional form of the three techniques are given below:

FAPM(X) = CSY S + PTOTAL

FMPM(X) = max(CSY S, PTOTAL)(1 + PTOTAL)

FEPM(X) = max(CSY S, PTOTAL)exp(PTOTAL)

(6.1)

where, FAPM : objective function using APM; FMPM : objective function using MPM; FEPM : objective function using EPM; and X : vector( X = [X1,X2, ...XN]T ) of design parameters

# ostIn – Constraints

In the Constraints group, the user supplies information about the various constraints that are to be placed on a general constrained optimization problem. Any number and combination of constraints are supported. The configuration syntax for constraints consists of: constraint name, constraint type, conversion factor, and names of relevant response (or tied-response) variables.

# Creating Template Files

# Running Ostrich

This chapter describes the execution of OSTRICH from the command line and parallel computing environments. Four OSTRICH executables are available: a serial version which runs on Windows, a multi-core parallel version that runs on Windows, a serial version which runs on Linux, and a parallel version which runs on Linux-based parallel clusters. Regardless of which version of OSTRICH is used, the following components must be created and stored in a working directory before running OSTRICH:

**ostIn.txt**: This is the main configuration file which should be created using the syntax described in Sections 2.3 through 2.23.

**Template File(s)**: These are file(s) that OSTRICH uses to create a syntactically correct model input file(s), so as to evaluate some set of model parameters (X).

(a) Create a template file by making a copy of the corresponding model input file.

(b) Edit the template file by replacing parameter values with the corresponding parameter names.

(c) Check that template file has been listed in the FilePairs section of ostIn.txt.

(d) Check that parameter names in Params section of ostIn.txt are consistent with those used in template file.

(e) [optional] If including a ParameterCorrections group, prepare and include appropriate template files for parameter corrections (see Section 2.13)

(f) [optional] If including a ParameterCorrections group, check that parameter names listed in the Corrections sub-group are consistent with the Params section as well as the template files for parameter corrections.

**Extra Model Input Files**: Any model input files not required by OSTRICH (i.e. there is no corresponding template file), but needed by the model.

# Serial Execution

Once all files have been created and placed in the working directory, serial execution of OSTRICH is straightforward: open a command line prompt, change directory (cd) to the working directory, and run OSTRICH by typing:

**/<path>/Ostrich** (if using Linux)

or

**<path>\Ostrich.exe** (if using Windows),

Where <path> is the path to the location of the OSTRICH executable (e.g. **C:\Program Files\Ostrich** or **/home/usr/bin**). When run in serial, an optimization run record is printed for each iteration of the chosen algorithm.

# Multi-core Parallel Execution in Windows

# Distributed or Multi-core Parallel Execution in Linux

To run OSTRICH in parallel on a Linux machine or Linux-based cluster of machines, the Linux environment must provide MPI (Message Passing Interface) libraries. Any MPI implementation will suffice and pre-compiled RedHat/CentOS OSTRICH binaries compatible with various versions of OpenMPI and Intel-MPI are provided with the OSTRICH distribution. For systems not supported by the pre-compiled binaries it is necessary to re-compile OSTRICH source so that it can link with a supported MPI implementation. Please contact L. Shawn Matott via e-mail ([lsmatott@buffalo.edu](mailto:lsmatott@buffalo.edu)) to obtain the source code.

Currently supported parallel algorithms implemented in OSTRICH are listed in ***Table 1***. While any of the other algorithms can be successfully run in a parallel environment, doing so will not result in any performance improvement. A typical command line for launching OSTRICH in parallel on a Linux machine or cluster is given below:

# Aborting an Ostrich Run

# Restarting an Ostrich Run

# Ostrich Output Files

Upon completion, OSTRICH will have generated various output files. The name of each file will contain the id of the corresponding processor *N*, where *N*=0 is the supervisor/master processor responsible for gathering and reporting the majority of the output. Output files generated by subordinate processors are generally only useful for troubleshooting.

* **OstOutputN.txt**: The main output file, it contains an optimization (or regression) record along with statistical output (if applicable).
* **OstErrorsN.txt**: Any errors or warnings encountered by processor *N* are stored in this file.
* **OstModelN.txt**: A sequential record of every model run evaluated by processor *N* is stored in this file.
* **OstExeOut.txt**: The standard output and standard error of Model runs are redirected to this file. For a given processor, only the output from the most recent evaluation is retained.
* **OstStatusN.txt**: This file is periodically updated with the current progress (i.e. percent complete and number of model evaluations) of the selected search algorithm.
* **OstGcopOut**: This file keeps track of the cost and constraint information for each model evaluation when the General-purpose Constrained Optimization Platform (GCOP) is used.
* **OstNonDomSolutions**: When a multi-objective search is performed, this file is periodically updated with the current list of non-dominated solutions that have been discovered by the selected search algorithm.

The following sub-sections describe selected OSTRICH output files in more detail.

# OstOutput – Main Output File

The main output file always contains the following elements (i) a GNU Public License disclaimer; (ii) a summary of the basic configuration variables; (iii) an OSTRICH run record detailing each iteration of the optimization algorithm; and (iv) the resulting optimal parameter set(s) and objective function value. The run record contains the parameter and objective function values at each iteration along with an algorithm-dependent value that indicates progress of the algorithm toward convergence. For most algorithms, the parameter and objective function values for the current best solution are reported.

# OstOutput – Statistical Output

Various statistical measures may be reported if calibration using the WSSE objective function is being performed. The statistics that are reported will depend on whether or not they were selected in the input file (see the “Math and Stats” group described in Section 2.20). The following sub-sections describe the output of these statistics.

## Observation Residuals

Observations residuals are reported automatically at the end of every WSSE calibration. Also included in the observation residual output is the correlation between measured and simulated observations (*Ry*).

## Error Variance and Standard Error of the Regression

Including the **StdDev** option in the MathAndStats group will cause the error variance (*s2*) and standard error of the regression (*s*) to be reported.

## Parameter Variance-Covariance and Correlation

Including **StdErr** in the MathAndStats group will cause OSTRICH to output the parameter variance-covariance matrix along with the standard error of each parameter. Furthermore, including **CorrCoeff** in the MathAndStats group will cause OSTRICH to output the parameter correlation matrix.

## Confidence Intervals

Including **Confidence** in the MathAndStats groups triggers the reporting of linear confidence intervals (CI) for each parameter. OSTRICH will use the value of the **CI\_PCT** variable as the corresponding confidence level (e.g. CI\_PCT = 0.95 selects a 95% confidence interval).

## Model Linearity

Including either Beale or Linssen in the MathAndStats group will trigger reporting of the corresponding non-linearity measures and linearity thresholds.

## Normality of Residuals

Inclusion of the **NormPlot** variable will cause OSTRICH to report a list of normalized residuals and the corresponding normal probability correlation coefficient (*R2N*).

## Influential Observations

When either the **CooksD** or **DFBETAS** variable (or both) is set, OSTRICH will generate and output the corresponding measures of observation influence, along with an assessment of which observations are influential, based on influence thresholds suggested in the literature.

## Parameter Sensitivities

Including the **Sensitivity** variable in the MathAndStats group causes OSTRICH to report parameter sensitivity measures. These are measures of local parameter sensitivity and are centered on the optimal parameter set.

## Matrices

OSTRICH can be configured to output matrices used for various statistical calculations by including the **Matrices** variable in the MathAndStats group.

# OstError – OSTRICH Error and Warning Messages

During execution, OSTRICH logs errors and warnings to the file ostErrors*N*.txt, where *N* corresponds to the processor number. Some errors are severe and will cause OSTRICH to abort while others serve as warnings that the results may not be optimal even though OSTRICH is able to proceed. Error codes and brief descriptions are given below in Table XXX.

[table xxx goes here]

# OstExeOut – Redirected Model Output

OSTRICH redirects the standard error and standard output of each model run to a file named ’OstExeOut.txt’. The output of each new model run will overwrite the output of the previous model run so that OstExeOut.txt always contains the standard error and standard output of the most recent model run.

# OstModel – Model Run Record

OSTRICH maintains a file named ’OstModel*N*.txt’, where *N* is the processor number, containing the parameter set and objective function of each model run. This list of model runs is numbered in increasing sequential order, with the highest value corresponding to the most recent model run.

# Examples

# Demo #1 – Calibrating SPLIT Groundwater Flow Model

# Demo #2 – Pump-and-Treat Optimization

# Demo #3 – BIGFOOT (Benchmarking Interface for Global Function Optimizers and Optimization Toolkits)

# Demo #4 – TUSWAMP (Tufts University Simple Watershed Modeling Program)

# Demo #5 – Cantilever Beam Multi-Objective Optimization

# Demo #6 – Multi-Criteria MODFLOW Calibration