OSTRICH: An Optimization Software Tool; Documentation and User's Guide

L. Shawn Matott
State University of New York at Buffalo
Department of Civil, Structural and Environmental Engineering

Version 1.6

Contents

Preface

This document describes the software package known as OSTRICH, a model-independent optimization and calibration tool. The OSTRICH package includes a diverse set of algorithms that can be used either for weighted least-squares calibration of model parameters, for optimization of pump-and-treat and other well-field design problems, or for optimizing a set of model parameters according to a user-defined objective function. Parameters to be calibrated or optimized can be log-transformed using either $\log_{10}$ or $\log_e$ logarithms.

For calibration problems, OSTRICH is also capable of computing an extensive set of statistics, include confidence intervals, parameter correlation, tests of normality and non-linearity, and observation influence and parameter sensitivity measures.

As mentioned, OSTRICH is model-independent, and can be configured to operate with any modeling program that utilizes text-based input and output file formats. Executable versions of OSTRICH are available for both Windows and Linux-based computing environments. A parallel version of OSTRICH, utilizing industry standard MPI interface, is also available. Additionally, OSTRICH can be configured to execute a parallel version of the underlying model executable.

To facilitate integration of OSTRICH into graphical model development tools (e.g. Visual ModFlow, Visual BlueBird), a set of Visual Basic modules is available. These modules encapsulate the OSTRICH input/output file format into a series of Get()/Set() routines, freeing the GUI developer from concerns over file syntax.

Revision Summaries

For those already familiar with OSTRICH, this chapter summarizes the modifications that have been made since the original version 1.0 release. As such, information regarding OSTRICH behavior in this chapter enhances and/or supersedes the descriptions given in the main body of the manual.

Version 1.1

Version 1.1 contains the following modifications:

- Added the option to use a deterministic quad-tree assignment of initial PSO and GA populations in lieu of random assignment. This option is selected by including the following line within the appropriate PSO or GA configuration section:

```
InitPopulationMethod QuadTree
```

- The internal OSTRICH memory checker now reports the line number and source file where a memory allocation failure occurs (if one is detected).
- In the tournament selection routine, a GA elitism bug was fixed. A memory access failure (i.e. NULL pointer exception) could occur if all members of the population had the same fitness.
- A bug was fixed in the routine to calculate matrix determinants (1-parameter arrays were not being handled properly, causing a memory access failure).

Version 1.2

Version 1.2 contains the following modifications:

- Added initial support for combinatorial parameters. The base classes for such parameters are defined and can be initialized from the `ostIn.txt` input file, but the actual optimization/calibration code does not yet utilize these parameters.
- The move limit criterion in the Levenberg-Marquardt algorithm was modified so that moves which take the design outside of the parameter limits are adjusted to move half the distance from the current value to the upper or lower limit, whichever is applicable.
- The parallel model execution code no longer tries to dynamically assess available resources. Instead, number of CPUs and wall-time will have to be entered manually by the user in `ostIn.txt` or will revert to some reasonable defaults. Parallel model execution is not yet fully operational, but is getting closer!
- An experimental hybrid of PSO and Levenberg-Marquardt was added. To use this option, include the following `ProgramType` line:

  ```
  ProgramType     PSO-LevMar
  ```

  and configure the `ParticleSwarm`, `LevMar`, and `MathAndStats` sections as you would in previous versions.
- An option to have Ostrich compute the optimal finite-difference step size was added. To use this option, include the following line in the `MathAndStats` section:

  ```
  DiffRelIncrement    optimal
  ```

- Adjustments were made to make the string parsing of the `ostIn.txt` input file more robust.

Version 1.4

Version 1.4 contains the following modifications:

- Added full support for combinatorial and integer parameters. Combinatorial parameters are parameters that take on a finite number of discrete values (for example on/off parameters), while integer parameters take on a range of integer values. For details see Sections 4.7 and 4.6.

- Added support for tied parameters, parameters that are computed as a function of other parameters. For details see Section 4.8.
- Added support for pump-and-treat optimization (PATO), including response variables (the PATO equivalent of WSSE observations), constraints, and plume geometry. A full description of PATO is given in Chapter 3 and instructions on it's use are given in Chapter 5.
- Added detection of insensitive observations and parameters, based on all-zeroes within a row or column of the Jacobian matrix. Such a condition makes the matrix singular and non-invertible, and therefore causes the Levenberg-Marquardt algorithm to abort.
- Added a check of template files to ensure that all parameters and tied parameters appear in at least one template file.
- Added collection and reporting of algorithm metrics.
- Added user-friendly output for intermediate algorithm loops (lambda trials, generation/swarm evaluation, annealing transitions, and one-dimensional searches).
- Several memory leak and memory fragmentation bugs were discovered and addressed. A memory utilization macro was added to prevent and diagnose future memory problems. As part of RAM fixes, the number of error messages stored in the error log is now limited to 100.
- Added support for user requested abortion of the program. If the user creates a file named "OstQuit.txt", Ostrich will terminate at the end of its current algorithm iteration.
- Reworked the Golden Section search to be more deterministic (max iterations is pre-calculated) and also added some logic to improve Golden-Section performance when it is applied to a multi-modal objective.
- Added explicit consideration of parameter side-constraints to Fletcher-Reeves, Powell and Steepest-Descent algorithms. As with Levenberg-Marquardt algorithm, moves which take the design outside of the parameter limits are adjusted to move half the distance from the current value to the upper or lower limit, whichever is applicable.

- Rewrote simulated annealing algorithm to follow the procedure outlined by Vanderbilt and Louie (1984). These changes have affected some of the SA input parameters, see Section **??** for details.

- Added time and date of build to version output.

- Heuristic algorithms (i.e. GA, PSO, and SA) now have a convergence criteria, which is based on the relative difference between median and minimum objective functions following a given algorithm iteration.

- Modified observation input so that each observation may be assigned a different parsing token. As a result, the `ObsToken` parameter used in previous versions of OSTRICH *is no longer used*. See Section 4.4 for the revised syntax of the Observation Group section of the `OstIn.txt` input file.

Version 1.6

Version 1.6 contains the following modifications:

- Added support for geometry-constrained calibration, including constraints on vertices and insertion of vertices where element geometries overlap. Geometry calibration code was created to support a poster presented at the AGU Fall 2004 Meeting, and the use of the geometry-constrained calibration module is not documented in this manual at the present time. OSTRICH users who are interested in using the module should contact: lsmatott@buffalo.edu

- Added support for generalized constrained optimization (GCOP), including tied response variables. This allows the user considerably more flexibility in incorporating OSTRICH into a general optimization problem, and in many cases will alleviate the need to write a user-supplied driver program. See Chapter 6 for details on how to utilize the GCOP option.

- For non-population based algorithms, added the option to have the design parameters be randomly initialized (this is already possible with population-based algorithms, such as PSO and GA). To use this option, use the keyword `random` in the parameter section in lieu of an actual value for the initial value field. The example below illustrates the syntax, as applied to a set of integer parameters:

```
#Model Parameters
BeginIntegerParams
#name initial lwr upr
C01 random 1 19
C02 random 1 18
C03 random 1 18
EndIntegerParams
```

- Added option to seed some or all of the initial PSO swarm or GA population. This allows the user to incorporate prior information (such as previous optimization results) into the optimization, and may enhance the efficiency and/or effectiveness of the algorithm. To use this option, insert an '`InitParams`' section, which uses the following syntax:

`BeginInitParams`

$$
\begin{array}{cccccccc}
p_{1,1} & p_{2,1} & p_{3,1} & \cdot & \cdot & \cdot & p_{n,1} \\
p_{1,2} & p_{2,2} & p_{3,2} & \cdot & \cdot & \cdot & p_{n,2} \\
\cdot & \cdot & \cdot & & & & \\
\cdot & \cdot & \cdot & & & & \\
p_{1,m} & p_{2,m} & p_{3,m} & \cdot & \cdot & \cdot & p_{n,m}
\end{array}
$$

`EndInitParams`

  where, n is the number of parameters, m is the number of seeds, and $p_{i,j}$ is the j-th seeded value of the i-th parameter (ordered according to the order of the parameters section(s)).

- Added algorithm metrics to Levenberg-Marquardt and Statistics modules. Information about these modules is now collected during program execution and reported upon program completion.

- Added checks for the existence of model executable and model output file(s). Also added more meaningful error messages.

- Added the GRID and BGA algorithms, along with two SA variants. The GRID algorithm can be used to perform an exhaustive search

and/or to generate contours of the objective function surface. The BGA is a binary coded genetic algorithm and the SA variants are a basic SA for continuous parameters, and an SA for discrete parameters. These SA variants are provided in addition to the Vanderbilt-Louie implementation of the version 1.4 OSTRICH release. Information about these new algorithms are available in the following sections:

  - GRID: The GRID algorithm is described in Section 2.4.4, and the input syntax is given in Section 4.10.8.
  - BGA : The OSTRICH implementation of the BGA algorithm is described in Section 2.4.1. The input syntax is described in Section 4.10.5.
  - SAs : The OSTRICH implementation of the SA variants is described in Sections 2.4.2. The input syntax is described in Section 4.10.6.

  The appropriate `ProgramType` setting for selecting these algorithms is provided in Section 4.2.

- Added real-time tracking of program status. This results in the generation of an `OstStatus.txt` file that is updated as the program progresses.

- Added a cost formulation to PATO that corresponds to the cost function defined by Mayer et al. (2002). The mathematical equation for this cost is given in Section 3.3, and the syntax is described in Section 5.2.

- Added support for alternative scientific notation formats when reading in text files. Previous versions would not accept numbers such as 1.0000D-4, due to the use of the 'D' character (as opposed to the usual 'E' character); this has been corrected.

- Added support for parameter-specific relative increments in the calculation of Jacobian and derivatives. That is, rather than applying the same relative increment to all parameters, users may specify such values on a per-parameter basis. To use this option, follow the following syntax for the `DiffRelIncrement` parameter when configuring the `MathAndStats` section of the input file:

```
DiffRelIncrement    p1  p2  .  .  .  pn
```

where `p1` through `pn` are the relative increments to use for each parameter, ordered as they appear in the `Params` section.

- Added support for ON/OFF thresholds in PATO pumping rates. When a pumping rate parameter is assigned a value below the ON/OFF threshold, the well is automatically turned off by assigning a rate of zero. The syntax for this option has been added to the PATO module documentation (Section 5.2).

- Added support for linearly reducing the PSO inertia weight to zero. To select this option, include the following syntax in the `ParticleSwarm` section:

```
InertiaReductionRate    linear
```

Chapter 1

Introduction

This chapter begins by discussing the motivation behind OSTRICH and follows this up with a brief discussion of the differences between calibration and optimization. The chapter concludes with a summary of the various calibration/optimization features that OSTRICH provides.

1.1 Motivation

OSTRICH has been created as a model-independent program that allows researchers and field-practitioners to automate the processes of model calibration and design optimization. For example, consider a hydrologist who desires to model groundwater flow as a first step in modeling contaminant transport. The hydrologist may want to calibrate values of recharge and hydraulic conductivity using head observations taken from wells throughout the model area. Lacking calibration software (such as OSTRICH, UCODE, or PEST), this calibration exercise could be performed manually using a trial-and-error approach.

The trial-and-error approach starts with the hydrologist deciding on a range of possible values for each parameter. This range can be based on previous experience, literature review, or a combination of the two. Next, the hydrologist selects a set of parameter values from the pre-determined range and runs the model. When the model completes, the hydrologist compares the head values predicted by the model with those collected in the field. This process of selecting parameter values, running the model and comparing results with head observations continues until the hydrologist is satisfied that the set of parameter values that best fits the observed data has been discovered.

A major short-coming of the trial-and-error approach is that the hydrologist ends up with a single set of parameters and little more than a gut feeling that these parameters are the best-fit parameters. Conversely, the algorithms used during the automated calibration process yield best-fit parameter estimates along with a variety of statistics that characterize the uncertainty associated with these best-fit estimates. This statistical characterization of the calibration results can help the hydrologist identify ways in which the model may be further improved. For example, influential observations may indicate areas in which additional sampling is desirable.

To understand the motivation for automated optimization, imagine that the hydrologist has completed the groundwater model and has concluded that a pump-and-treat system must be installed to address contaminant transport concerns. Following this assessment, an environmental engineer is tasked with designing the required system of treatment wells. In particular, the engineer must determine the required number of wells, the location where each well should be installed, and the pumping rate of each well. Furthermore, the system must be designed in such a way that it alleviates contaminant transport concerns at the lowest possible cost without incurring excessive draw-down in the aquifer.

The optimization problem described above could be solved via trial-and-error, but the engineer will not be able to give any assurances that the resulting system design is truly optimal. Furthermore, in the process of finding a single feasible design (which may or may not be the optimal design) the engineer tends to wind up mired in the tedium of configuring and analyzing countless manual runs of the model. On the other hand, the optimization strategies employed in OSTRICH alleviate the tedium of manual optimization and are, in principle capable of determining the optimal design, and in practice able to rapidly identify near-optimal designs.

1.2 Calibration vs. Optimization

In general, model calibration is a specific type of optimization wherein the objective is to find the values of certain model parameters such that the resulting sum of squared error between field-measured and model-computed observations is minimized. This so-called least-squares regression problem is different from other optimization problems in that its solution yields optimal parameters along with a set of statistics related to the goodness of fit of the optimal parameter set.

In OSTRICH, another difference between calibration and optimization

is that the use of least-squares error as the objective function for calibration problems is embedded into OSTRICH. Conversely, objective functions for all other optimization problems are not built into OSTRICH. Instead, the user must write a small program that interfaces between OSTRICH and the model executable. This program is called upon by OSTRICH to calculate the value of the objective function for a given set of parameters. Chapter 7 has more information on this procedure.

1.3 Summary of Features

One of the most important features of OSTRICH is that it is model-independent; that is, OSTRICH can be configured to calibrate and/or optimize nearly any modeling program. Chapter 4 discusses the details of preparing an OSTRICH input file that can be used for model calibration. Additional OSTRICH features are summarized in the following sub-sections.

1.3.1 Algorithms

OSTRICH implements classic numerical methods and popular heuristic algorithms, all of which are suitable for both calibration and optimization problems. Additionally, OSTRICH provides an algorithm (Levenberg-Marquardt) tailored to non-linear least-squares calibration problems. Together, these algorithms provide the user with a fair degree of flexibility and enable the present version of OSTRICH to tackle linear and non-linear continuous variable problems. The present version of OSTRICH is capable of handling discrete, mixed-integer and combinatorial problems. The following list briefly describes each of the algorithms provided by OSTRICH. A detailed description of these algorithms can be found in Chapter 2.

- Powell's Method: This zero-order method (i.e. does not require derivative information) utilizes a conjugate direction approach to locate locally optimal parameters.
- Steepest-Descent: This first-order method (i.e. requires derivative information) follows the so-called path of steepest descent to find locally optimal parameter values.
- Fletcher-Reeves: This is a first-order method that uses a conjugate gradient approach to locate locally optimal parameter values.

- Levenberg-Marquardt: This hybrid optimization algorithm combines variable-matric (also called quasi-Newton) and conjugate gradient techniques to efficiently solve non-linear least squares minimization problems; making it an ideal choice for solving many calibration problems.

- Genetic Algorithm: This heuristic algorithm operates on a population of parameter sets. A random survival-of-the-fittest process involving selection, crossover, and mutation, is applied to successive generations of the population. In this manner, the genetic algorithm evolves the initial population in a directed-random search for the globally optimal solution. Two GA variants are available in OSTRICH a binary-coded GA (BGA), suitable for discrete-parameter problems, and a real-coded GA (RGA), suitable for continuous-parameter problems.

- Simulated Annealing: This heuristic method is based on an analogy to the physical process of annealing; wherein a solid is rapidly heated then slowly cooled so as to reach the lowest energy state. In simulated annealing, a highly randomized search algorithm is slowly transitioned into a focused descent onto the globally optimal set of parameters. Three SA variants are available in OSTRICH an SA for discrete parameter problems (DSA), an SA for continuous parameter problems (CSA), developed by the author, and an SA for continuous parameter problems (VSA) developed by Vanderbilt and Louie (1984).

- Particle Swarm Optimization: This heuristic method was developed from attempts to simulate the flocking behavior of birds, fish and other animals. A swarm of particles (analogous to a population in a GA) containing both local and collective knowledge is 'flown' through the parameter space in search of the optimal solution.

1.3.2 Regression Statistics

Regardless of which algorithm is used, OSTRICH can calculate numerous regression statistics following a successful calibration exercise. These statistics are summarized below, while the mathematical formulae used to compute the statistics are discussed in section 2.2. A description of how OSTRICH formats the output of regression statistics is given in Chapter 9.2.

- Estimated Parameters, Error Variance, and Standard Error of the Regression: The most basic statistics computed by OSTRICH are the

estimated parameters along with the error variance and standard error of the regression. These statistics provide a rough indication of the goodness-of-fit of the calibrated model.

- Observation Residuals: At the end of a calibration, OSTRICH outputs a list of observations and the corresponding weighted residuals. Visual inspection of this list allows the user to quickly identify problematic observations. Alternatively, this list can be imported into a spreadsheet, where plots of observations vs. residuals can generated.

- Confidence Intervals: OSTRICH can compute arbitrarily sized confidence intervals (CI) for each parameter. In addition, OSTRICH computes the '*Volume Ratio*'; an indicator of the degree to which the CI block can be treated as a joint confidence region.

- Parameter Variance-Covariance: The variance-covariance matrix for the estimated parameters is used to derive the parameter standard error and parameter correlation coefficient statistics. Parameter standard error is an indicator of the overall goodness-of-fit of respective parameter estimates, while the parameter correlation coefficient is a measure of the dependence between parameters.

- Normality of Residuals: The normality of residuals provides a test of the statistical assumption that residuals are normally distributed over the pool of observations used in the regression.

- Influential Observations: OSTRICH can compute the Cook's D and DFBETAS measures of observation influence. These measure assist the modeler in identifying observations having the greatest impact on the calibration. Cook's D measures overall observation influence, and results in an vector of observation influence values. Meanwhile, DFBETAS measures observation influence with respect to each parameter, resulting in a matrix of influence values (one influence measure for each observation-parameter pair).

- Parameter Sensitivities: OSTRICH computes dimensionless and 1% scaled sensitivities, which are measures of the sensitivity of simulated observation values to changes in parameter values. In addition, OSTRICH computes the composite scaled sensitivity for each parameter, a measure of the overall sensitivity of the model with respect to each parameter. Parameters that have extremely low sensitivity values may

not have a significant impact on model calibration and robust calibration of such parameters may not be straightforward task.

- Model Linearity: Many of the statistics computed by OSTRICH rely on the assumption that the model is approximately linear in the region near the estimated parameter set. To test this assumption, OSTRICH can compute the Linssen and Beale measures of linearity. For models that are linear or moderately non-linear, either measure is appropriate, but Linssen's measure is more accurate for highly non-linear models.

- Matrices: OSTRICH can be configured to output the values of the Jacobian, Normal and Inverse Normal matrices, used during Levenberg-Marquardt regression and/or statistical calculations. These matrices can be used to compute statistics that are not currently offered within OSTRICH.

1.3.3 Supported Platforms

OSTRICH is available for both Linux and Windows platforms and an MPI-parallel version of OSTRICH is available for Linux-based parallel clusters. Additionally, OSTRICH can be configured to execute parallel versions of the underlying model executable, if one is available. Section 8.2 contains detailed instructions for utilizing the parallel options of OSTRICH. Finally, an OSTRICH GUI module has been developed using Visual Basic. This module provides an interface for integrating OSTRICH configuration, execution, and output with model-specific GUIs, such as Visual ModFlow and Visual BlueBird.

Chapter 2

Calibration and Optimization Algorithms

In this chapter, the mathematical and/or algorithmic underpinnings of the optimization and calibration methods implemented within OSTRICH are described in detail. In addition, this Chapter discusses the formulae and statistical theory associated with various regression statistics.

2.1 Levenberg-Marquardt Regression

Levenberg-Marquardt regression is a method, originally developed by Levenberg (1944) and extended by Marquardt (1963), that focuses on the problem of *non-linear least-squares* parameter estimation. The term '*non-linear*' refers to the scenario in which the model output (i.e. the simulated observations) varies non-linearly with the model input (i.e. the model parameters), and the term '*least-squares*' refers to the idea that the optimal parameter set is found by minimizing the sum of the squared differences between field-measured and model-simulated observations. Equation (2.1), gives a mathematical expression for a generic non-linear model, while equation (2.2) is an expression of the weighted least squares objective function.

$$\mathbf{Y}_{sim} = f(\mathbf{X}) \quad \text{where,} \tag{2.1}$$

$$\mathbf{Y}_{sim}^{T} = \begin{bmatrix} y_{sim_1} & y_{sim_2} & . & . & . & y_{sim_m} \end{bmatrix} \quad \text{and,} \tag{2.1a}$$

$$\mathbf{X}^{T} = \begin{bmatrix} x_1 & x_2 & . & . & . & x_n \end{bmatrix} \tag{2.1b}$$

Where $\mathbf{Y}_{sim}$ is a vector of m simulated observations generated by evaluating model $f()$ using a set of n parameters, denoted in vector format as $\mathbf{X}$.

$$\begin{aligned}\Phi &= \sum_{i=1}^{i=m} \left(w_i(y_{obs_i} - y_{sim_i})\right)^2 \\ &= (\mathbf{Y}_{obs} - \mathbf{Y}_{sim})^T \mathbf{Q} (\mathbf{Y}_{obs} - \mathbf{Y}_{sim}) \\ &= (\mathbf{Y}_{obs} - f(\mathbf{X}))^T \mathbf{Q} (\mathbf{Y}_{obs} - f(\mathbf{X})) \quad \text{where,}\end{aligned} \tag{2.2}$$

$$\mathbf{Y}_{obs}^T = \begin{bmatrix} y_{obs_1} & y_{obs_2} & . & . & . & y_{obs_m} \end{bmatrix} \quad \text{and,} \tag{2.2a}$$

$$\mathbf{Q} = \begin{bmatrix} w_1^2 & 0 & . & . & . & 0 \\ 0 & w_2^2 & . & . & . & 0 \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ 0 & 0 & . & . & . & w_m^2 \end{bmatrix} \tag{2.2b}$$

Where Φ is the weighted sum of squared errors objective function, $\mathbf{Y}_{obs}$ is a vector of m measured observation values, w_i is the weight given to the i^{th} observation, and the matrix $\mathbf{Q}$ is an $m \times m$ matrix with diagonal values Q_{ii} equal to the square of the corresponding observation weight (w_i).

The goal of non-linear least-squares regression is to minimize Φ by varying $\mathbf{X}$ subject to the constraint that the parameters take on physically meaningful values (e.g. in the groundwater model described in Section 1.1, the hydraulic conductivity must be a non-negative quantity). Equation (2.3) is a formal optimization problem statement for non-linear least-squares regression.

$$\begin{aligned} &\text{Minimize} \quad \Phi = (\mathbf{Y}_{obs} - f(\mathbf{X}))^T \mathbf{Q} (\mathbf{Y}_{obs} - f(\mathbf{X})) \\ &\text{such that:} \quad \mathbf{X}_L \leq \mathbf{X} \leq \mathbf{X}_U \end{aligned} \tag{2.3}$$

Where $\mathbf{X}_L$ and $\mathbf{X}_U$ are vectors containing the lower and upper parameter bounds, respectively. The Levenberg-Marquardt solution of equation (2.3) begins by considering an initial vector of parameters $\mathbf{X}_0$ and corresponding simulated observations $\mathbf{Y}_{sim0}$. A Taylor series expansion about $\mathbf{X}_0$, yields

a linearized approximation for $\mathbf{Y}_{sim}$:

$$\mathbf{Y}_{sim} \approx \mathbf{Y}_{sim0} + \mathbf{J}(\mathbf{X} - \mathbf{X}_0) \quad \text{where,} \tag{2.4}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial y_{sim_1}}{\partial x_1} & \frac{\partial y_{sim_1}}{\partial x_2} & \cdot & \cdot & \cdot & \frac{\partial y_{sim_1}}{\partial x_n} \\ \frac{\partial y_{sim_2}}{\partial x_1} & \frac{\partial y_{sim_2}}{\partial x_2} & \cdot & \cdot & \cdot & \frac{\partial y_{sim_2}}{\partial x_n} \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & & \cdot & \cdot \\ \frac{\partial y_{sim_m}}{\partial x_1} & \frac{\partial y_{sim_m}}{\partial x_2} & \cdot & \cdot & \cdot & \frac{\partial y_{sim_m}}{\partial x_n} \end{bmatrix} \tag{2.4a}$$

Where $\mathbf{J}$ is the $m \times n$ Jacobian matrix, consisting of the partial derivatives of each simulated observation with respect to each parameter($J_{ij} = \partial y_{sim_i}/\partial x_j$). Substituting the linearized expression for $\mathbf{Y}_{sim}$ into equation (2.2) yields:

$$\Phi \approx (\mathbf{Y}_{obs} - (\mathbf{Y}_{sim0} + \mathbf{J}(\mathbf{X} - \mathbf{X}_0)))^T \mathbf{Q} (\mathbf{Y}_{obs} - (\mathbf{Y}_{sim0} + \mathbf{J}(\mathbf{X} - \mathbf{X}_0))) \tag{2.5}$$

It can be shown that the vector $(\mathbf{X} - \mathbf{X}_0)$ which minimizes equation (2.5) is given by:

$$(\mathbf{X} - \mathbf{X}_0) = (\mathbf{J}^T \mathbf{Q} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{Q} (\mathbf{Y}_{obs} - \mathbf{Y}_{sim0}) \tag{2.6}$$

Since equation (2.4), is an approximation, it follows that the solution given by equation (2.6) is also only an approximation. Therefore, an iterative solution procedure is utilized, wherein equation (2.6) is treated as an upgrade vector ($\mathbf{U}$) for the estimated optimal parameter set ($\mathbf{X}$). Figure 2.1 illustrates the general procedure for iteratively solving the non-linear least-squares regression problem.

The Levenberg-Marquardt method follows the general procedure illustrated in Figure 2.1, except that the formula for the upgrade vector, equation (2.6), is modified slightly:

$$\mathbf{U} = (\mathbf{J}^T \mathbf{Q} \mathbf{J} + \alpha \mathbf{I})^{-1} \mathbf{J}^T \mathbf{Q} (\mathbf{Y}_{obs} - \mathbf{Y}_{sim0}) \tag{2.7}$$

Where α is the Marquardt parameter and $\mathbf{I}$ is the $n \times n$ identity matrix. For large values of α, the revised upgrade vector is approximately equal to the direction of steepest descent (i.e. the negative gradient of the objective function, $-\nabla\Phi$); and as α approaches zero, the upgrade vector in equation (2.7) approaches that of equation (2.6). This formulation of the upgrade vector allows the Levenberg-Marquardt method to smoothly transition between a Steepest-Descent approach far away from the optimal parameter set to a Taylor series approximation close to the optimal parameter set.

1. Compute $\mathbf{Y}_{sim0}$ by evaluating $f(\mathbf{X}_0)$
2. Compute Φ_1 using equation (2.2)
3. Compute $\mathbf{U} = (\mathbf{X} - \mathbf{X}_0)$ using equation (2.6)
4. Compute $\mathbf{X} = \mathbf{X}_0 + \mathbf{U}$
5. Compute $\mathbf{Y}_{sim}$ by evaluating $f(\mathbf{X})$
6. Compute Φ_2 using equation (2.2)
7. Test for convergence by comparing Φ_1 and Φ_2
   (a) If converged, iteration is complete. $\mathbf{X}$ minimizes Φ.
   (b) If not converged, set $\mathbf{X}_0 = \mathbf{X}$ and return to Step 1.

Figure 2.1: General Iterative Procedure for Non-Linear Regression

In many calibration problems, elements within the observation vectors ($\mathbf{Y}_{sim}$ and $\mathbf{Y}_{obs}$) can differ by several orders of magnitude (consider the case of a calibration which combines the use of head[m] and streamflow[m^3/d] observations). Such variation can lead to numerical roundoff error in the computation of the Jacobian. Therefore, a scaling matrix is introduced into the upgrade vector such that the overall equation is mathematically identical to equation (2.7), but numerical errors are avoided:

$$\mathbf{U} = \mathbf{S}(((\mathbf{JS})^T\mathbf{QJS} + \alpha\mathbf{S}^T\mathbf{S})^{-1}(\mathbf{JS}^T)\mathbf{Q}(\mathbf{Y}_{obs} - \mathbf{Y}_{sim0})) \quad \text{where,} \tag{2.8}$$

$$\mathbf{S} = \begin{bmatrix} \frac{1}{J_{11}w_1} & 0 & . & . & . & 0 \\ 0 & \frac{1}{J_{22}w_2} & . & . & . & 0 \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ 0 & 0 & . & . & . & \frac{1}{J_{nn}w_n} \end{bmatrix} \tag{2.8a}$$

Where $\mathbf{S}$ is an $n \times n$ diagonal scaling matrix, with entries $S_{ii} = 1/(J_{ii}w_i)$. Finally, Marquardt introduced the 'Marquardt-lambda', defined as $\lambda = \alpha \times max(S_{ii}^2)$; the largest term in $\alpha\mathbf{S}^T\mathbf{S}$. Adjusting the Marquardt-lambda provides direct control over the weight given to the steepest-descent direction; when λ is very large, the steepest-descent direction will dominate the upgrade vector, whereas the Taylor series approximation will dominate for small values of λ.

Figure 2.2 describes the iteration procedure followed in the Levenberg-Marquardt regression method, where λ is adjusted during each iteration in

search of a value that provides the optimal balance between the competing steepest-descent and Taylor series upgrade vectors. Figure 2.2 captures

1. Initialize λ
2. Compute $\mathbf{Y}_{sim0}$ by evaluating $f(\mathbf{X}_0)$
3. Compute Φ_1 using equation (2.2)
4. Compute $\mathbf{U} = (\mathbf{X} - \mathbf{X}_0)$ using equation (2.8)
5. Compute $\mathbf{X} = \mathbf{X}_0 + \mathbf{U}$
6. Compute $\mathbf{Y}_{sim}$ by evaluating $f(\mathbf{X})$
7. Compute Φ_2 using equation (2.2)
8. Adjust λ
   (a) If # λ adjustments is optimal or exceeds maximum, goto Step 9.
   (b) If($\Phi_2 < \Phi_1$), reduce λ, set $\mathbf{X}_0 = \mathbf{X}$, and return to Step 2.
   (c) If($\Phi_2 \geq \Phi_1$), increase λ and return to Step 4.
9. Test for convergence by comparing Φ_1 and Φ_2
   (a) If converged, iteration is complete. $\mathbf{X}$ minimizes Φ.
   (b) If not converged, set $\mathbf{X}_0 = \mathbf{X}$ and return to Step 1.

Figure 2.2: Iterative Procedure for Levenberg-Marquardt Regression

the essence of the Levenberg-Marquardt algorithm as it is implemented in OSTRICH. Behind the scenes, OSTRICH performs additional steps to: (a) compute the Jacobian matrix using finite-difference derivative approximations; (b) determine optimal step size in the direction of the computed upgrade vector; and (c) restrict parameter changes to fall within user-specified move limits. OSTRICH provides numerous user-configurable parameters that can be adjusted to tune the performance of the Levenberg-Marquardt algorithm to a given calibration exercise. These parameters are discussed in detail in Chapter 4.

2.2 Regression Statistics

This section discusses the mathematical formulas used by OSTRICH to compute various regression statistics.

2.2.1 Observation Residuals

An observation residual is the difference between a field-measured observation and the corresponding model-simulated observation. Individual elements (r_i) of the vector of weighted observation residuals ($\mathbf{r}$) are computed as:

$$r_i = w_i(y_{obs_i} - y_{sim_i}) \tag{2.9}$$

Where w_i is the weight given to the i^{th} observation, and y_{obs_i} and y_{sim_i} are the i^{th} field-observed and model-simulated observations, respectively. Observation residuals that are grossly out of proportion to other residuals are an indicator of gross errors; either in the model, the weighting, or field-observation associated with the given residual. Hill (1998) and Cooley and Naff (1990) suggest the following formula for computing the correlation between weighted residuals and measured observations (R_y):

$$R_y = \frac{\sum_{k=1}^{m}(w_i y_{obs_i} - y_{obs}^{avg})(w_i y_{sim_i} - y_{sim}^{avg})}{\sqrt{\sum_{k=1}^{m}(w_i y_{obs_i} - y_{obs}^{avg})^2 (w_i y_{sim_i} - y_{sim}^{avg})^2}} \tag{2.10}$$

Where m is the number of observations, y_{obs}^{avg} and y_{sim}^{avg} are the averages of the weighted measured and weighted simulated observations, respectively. Cooley and Naff (1990) suggest that $R_y > 0.9$ indicates a good fit between the calibrated model and the measured observations.

2.2.2 Error Variance and Standard Error of the Regression

The error variance and standard error of the regression provide statistical measures of the goodness-of-fit of the calibrated model to the observation data, and are calculated as (Draper and Smith, 1998):

$$s^2 = \frac{\Phi}{(m-n)} \tag{2.11}$$

Where s^2 is the error variance, s is the standard error of the regression, Φ is the weighted sum of squared errors, calculated using the calibrated parameter set, and $(m-n)$ is the number of degrees of freedom, where m and n are the number of observations and number of parameters, respectively. Hill (1998) suggests that if the fitted model is consistent with the observation weights (w_i), then the expected value of s^2 and s should approach 1.0 with large m.

2.2.3 Parameter Variance-Covariance

The $n \times n$ parameter variance-covariance matrix, evaluated using the calibrated parameter set, is computed as (Hill, 1998):

$$\begin{aligned}\mathbf{V} &= s^2(\mathbf{J}^T\mathbf{Q}\mathbf{J})^{-1} \\ &= \begin{bmatrix} Var_1 & Cov_{1,2} & . & . & . & Cov_{1,n} \\ Cov_{2,1} & Var_2 & . & . & . & Cov_{2,n} \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ Cov_{n,1} & Cov_{n,2} & . & . & . & Var_n \end{bmatrix}\end{aligned} \tag{2.12}$$

Where $\mathbf{V}$ is the variance-covariance matrix, s^2 is the error variance, $\mathbf{J}$ is the Jacobian matrix, $\mathbf{Q}$ is a diagonal matrix with elements Q_{ii} equal to the square of the corresponding observation weight (w_i), $(\mathbf{J}^T\mathbf{Q}\mathbf{J})^{-1}$ is known as the inverse normal matrix, Var_i is the variance of the i^{th} parameter, and $Cov_{i,j}$ is the covariance between parameter x_i and parameter x_j. The standard error (se_i) of the i^{th} parameter is defined as (Hill, 1998):

$$se_i = \sqrt{Var_i} \tag{2.13}$$

and is a measure of the reliability of the estimated parameter. Meanwhile, the correlation coefficient between parameters x_i and x_j, defined as (Hill, 1998):

$$Cor_{i,j} = \begin{cases} 1.00 & \text{if } i = j, \\ \frac{Cov_{i,j}}{\sqrt{Var_i Var_j}} & \text{if } i \neq j. \end{cases} \tag{2.14}$$

is a measure of the linear dependence between the two parameters, and can range from -1.0 to +1.0. $Cor_{i,j} \approx 0.0$ indicates no correlation (x_i and x_j are independent), while $Cor_{i,j} \approx -1.0$ indicates negative correlation ($x_i \propto -x_j$), and $Cor_{i,j} \approx +1.0$ indicates positive correlation ($x_i \propto x_j$). In general, highly correlated parameters cannot be uniquely estimated by the model, $f(\mathbf{X})$, used in the given calibration exercise.

2.2.4 Confidence Intervals

Linear confidence intervals are computed using the student t-distribution, which is defined as (Draper and Smith, 1998):

$$F_\nu(a) = \int_{-\infty}^{a} f_\nu(t)dt \quad \text{where,} \tag{2.15}$$

$$f_\nu(t) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left(1+\frac{t^2}{\nu}\right)^{-(\nu+1)/2} \quad \text{and,} \tag{2.15a}$$

$$\Gamma(q) = \int_0^\infty e^{-x}x^{q-1}dx \tag{2.15b}$$

Where $F_\nu(a)$ and $f_\nu(t)$ are the respective cumulative distribution function (cdf) and probability density function (pdf) of the student t-distribution, $\Gamma(q)$ is the gamma function, and ν is the number of degrees of freedom $(m-n)$. A $100(1-\alpha)\%$ confidence interval for parameter x_i is calculated as (Draper and Smith, 1998):

$$x_i \pm t\left(\nu, 1-\frac{\alpha}{2}\right)se_i \tag{2.16}$$

Where $t(\nu, 1-\alpha/2)$ is the inverse cdf of the student t-distribution, α is the significance level, and x_i and se_i are the estimated value and standard error of the parameter, respectively. The confidence interval of a given parameter is a range with probability $100(1-\alpha/2)\%$ of containing the true value of the parameter. As such, narrow confidence intervals are indicative of good parameter estimates.

Because confidence intervals do not take into account parameter correlation, confidence blocks (created by simultaneously considering the confidence intervals for multiple parameters) do not generally representant a joint confidence region for the given parameters. Draper and Smith (1998) suggest a measure, known as the '*volume ratio*', which indicates the degree to which a confidence block approximates the true joint confidence region. The volume ratio is defined as:

$$\frac{E}{R} = \sqrt{c_p} \quad \text{where,} \tag{2.17}$$

$$c_p = \det \mathbf{Cor} = \begin{vmatrix} 1.00 & Cor_{1,2} & . & . & . & Cor_{1,n} \\ Cor_{2,1} & 1.00 & . & . & . & Cor_{2,n} \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ Cor_{n,1} & Cor_{n,2} & . & . & . & 1.00 \end{vmatrix} \tag{2.17a}$$

Where n is the number of parameters, (E/R) is the n-dimensional volume ratio, E is the n-dimensional volume of the true joint confidence ellipsoid, R is n-dimensional volume of the rectangular block, and c_p is the determinant of the parameter correlation coefficient matrix. The volume ratio will vary from 0 to 1; a value of ≈ 1.00 indicates that the confidence block is an acceptable approximation of the joint confidence region, while a value of ≈ 0.00 indicates a poor approximation.

2.2.5 Normality of Residuals

The confidence intervals discussed in Section 2.2.4 are computed under the assumption that the observation residuals are independent and normally distributed. Normal probability plots of the residuals is a means to test these assumptions and detect possibly spurious field-data. A normal probability plot is a graph of the *expected values of the standard normal ordered residuals* (r_{exp_k}) versus the *ordered residuals*(r_{ord_k}). The ordered residuals are formed by taking the weighted observation residuals ($\mathbf{r}$) and sorting them in increasing order, such that r_{ord_k} is the k^{th} smallest residual. If the observation residuals are normally distributed, then the values of r_{ord_k} will be approximately equal to their corresponding expected standard normal values, r_{exp_k}, and the plotted points (r_{exp_k}, r_{ord_k}) will fall on a straight line whose slope is ≈ 1.00. As suggested by Sabo (1999), OSTRICH uses the Snedecor & Cochran approximation to compute values of r_{exp_k}:

$$r_{exp_k} = Z^{-1}\left(\frac{k - 0.375}{m + 0.25}\right) \quad , \; k = 1..m \tag{2.18}$$

Where m is the number of observations, and Z^{-1} is the inverse cdf of the standard normal distribution. Visual inspection of the r_{exp_k} vs. r_{ord_k} plot can reveal outliers; points that lie far from the ideal normal probability line (i.e. a straight line with slope of 1.00). Possible causes of outlier residuals are: (a) the accuracy of associated field data is not properly reflected in the weighting of the residual; (b) the model contains a conceptual error in the vicinity of the associated observation; and (c) the model is affected by non-uniform numerical error near the associated observation.

In addition to providing the user with the (r_{exp_k}, r_{ord_k}) normal probability plot data points, OSTRICH also computes the normal probability

correlation coefficient (R_N^2), defined as (Hill, 1998):

$$R_N^2 = \frac{((\mathbf{r_{ord}} - \mathbf{r_{avg}})^T \mathbf{r_{exp}})^2}{((\mathbf{r_{ord}} - \mathbf{r_{avg}})^T (\mathbf{r_{ord}} - \mathbf{r_{avg}}))(\mathbf{r_{exp}}^T \mathbf{r_{exp}})} \quad \text{where,} \tag{2.19}$$

$$\mathbf{r_{ord}} = \begin{bmatrix} r_{ord_1} & r_{ord_2} & . & . & . & r_{ord_m} \end{bmatrix}^T \quad \text{and,} \tag{2.19a}$$

$$\mathbf{r_{exp}} = \begin{bmatrix} r_{exp_1} & r_{exp_2} & . & . & . & r_{exp_m} \end{bmatrix}^T \quad \text{and,} \tag{2.19b}$$

$$\mathbf{r_{avg}} = \left(\sum_{i=1}^{m} \frac{r_i}{m} \right) \begin{bmatrix} 1.00 & 1.00 & . & . & . & 1.00 \end{bmatrix}^T \tag{2.19c}$$

Where $\mathbf{r_{ord}}$ is the vector of ordered residuals, $\mathbf{r_{exp}}$ is the vector of expected values of the standard normal ordered residuals, and $\mathbf{r_{avg}}$ is a vector with all elements equal to the average value of the weighted residuals. If the residuals are independent and normally distributed, R_N^2 will be close to 1.00.

2.2.6 Influential Observations

OSTRICH can compute both the Cook's D (Cook and Weisberg, 1982) and the DFBETAS (Belsley et al., 1980) measures of observation influence. These measures are useful in identifying observations that had a large effect on the estimated parameter set and can provide additional insight into the behavior of the model. For example, Yager (1998) examined the influential observations of several groundwater flow models and found (a) that each calibration was significantly influenced by a few discharge measurements, and (b) that a recharge zone within one of the models had been assigned an erroneous value.

The Cook's D measure is a vector of m values corresponding to the relative distance between the parameter set, $\mathbf{X}$, estimated using all observations and the parameter set, $\mathbf{X}_{(i)}$, estimated when the i^{th} observation is not used. Individual elements, D_i, of the Cook's D vector are computed as (Cook and Weisberg, 1982):

$$D_i = \frac{1}{n} r_{stu_i}^2 \frac{h_{ii}}{1 - h_{ii}} \quad \text{where,} \tag{2.20}$$

$$r_{stu_i} = \frac{r_i}{s\sqrt{1 - h_{ii}}} \quad \text{and,} \tag{2.20a}$$

$$\mathbf{H} = (\mathbf{Q}^{1/2}\mathbf{J})(\mathbf{J}^T\mathbf{Q}\mathbf{J})^{-1}(\mathbf{Q}^{1/2}\mathbf{J})^T \tag{2.20b}$$

Where n is the number of parameters, r_{stu_i} is the weighted standardized residual of observation i, h_{ii} (known as the leverage of observation i) is a

diagonal element of the hat matrix $\mathbf{H}$, r_i is the weighted residual of observation i (as defined in equation (2.9), s is the standard error of the regression, $\mathbf{J}$ is the Jacobian matrix, and $\mathbf{Q}$ is defined in equation (2.2b).

To decide whether or not a given D_i signifies an influential observation, Rawlings (1988) suggests using a threshold of $4/m$, where m is the number of observations. Therefore, if $D_i > 4/m$, OSTRICH will report observation i as influential with respect to the Cook's D parameter. Yager (1998) also uses the leverage, h_{ii}, as a measure of observation influence, with a threshold value of n/m. Based on this criteria, OSTRICH will report observation i as influential with respect to the leverage if $h_{ii} > n/m$.

The DFBETAS measure is calculated by removing an entire row of observation sensitivity data ($\left[\frac{\partial y_{sim_i}}{\partial x_1} \quad \frac{\partial y_{sim_i}}{\partial x_2} \quad . \quad . \quad . \quad \frac{\partial y_{sim_i}}{\partial x_n}\right]$) from the Jacobian matrix, and calculating the effect that this deletion has on the estimated parameters. This procedure results in a $m \times n$ **DFBETAS** matrix (m is the number of observations and n is the number of parameters, with element $DFBETAS_{ij}$ corresponding to the influence of observation i on parameter j. **DFBETAS** elements are computed in OSTRICH using (Belsley et al., 1980):

$$DFBETAS_{ij} = \frac{c_{ji}}{\sqrt{\sum_{k=1}^{m} c_{jk}^2}} \frac{r_i}{s(i)(1-h_{ii})} \quad \text{where,} \tag{2.21}$$

$$s(i) = \frac{1}{(m-n-1)}\left[(m-n)s^2 - \frac{r_i^2}{1-h_{ii}}\right] \quad \text{and,} \tag{2.21a}$$

$$\mathbf{C} = (\mathbf{J}^T\mathbf{Q}\mathbf{J})^{-1}(\mathbf{J}\mathbf{Q}^{1/2})^T \tag{2.21b}$$

Where $\mathbf{C}$ is the $n \times m$ change matrix, with elements c_{ji}, $s(i)$ is an estimate of the standard error of the regression with observation i removed, r_i is the weighted residual of observation i, h_{ii} is a diagonal element of the hat matrix, $\mathbf{H}$, and s^2 is the error variance. Following the suggestion of Belsley et al. (1980) OSTRICH reports observation i as influential to parameter j if $DFBETAS_{ij} > 2/\sqrt{m}$.

The validity of the Cook's D and DFBETAS measures of observation influence are dependent on the degree of linearity of the model. If the model is effectively linear in the vicinity of the parameter set at which the Cook's D and DFBETAS are calculated, the measures will be accurately computed; otherwise, the measures should be treated as qualitative indicators of observation influence (Yager, 1998).

Even under sufficiently linear conditions, correct interpretation of the

Cook's D and DFBETAS measures generally requires additional investigation. The influential observations may be providing valuable information for the calibration, or they may be sources of error, reducing the reliability of the calibration. To this end, the modeler might pose the following questions regarding the influential observations:

- Are the influential observations subject to larger than average measurement error? If so, the observation weights should be adjusted accordingly.
- Is the model subject to increased numerical error around the influential observations? This could suggest the need to refine the element mesh or finite difference grid, if such a model is being used.
- Is there a logical error in the model configuration which causes the observations to be unduly influential? For example, Yager (1998) used DFBETAS to track down an erroneously assigned recharge zone.
- Is there some physical aspect of the model that causes the observations to be influential? For example, Yager (1998) found that influential head observations were located in areas having steep head gradients and high flow rates.
- Are certain kinds measurement data (e.g. head, flow, or temperature data) disproportionately represented by the influential observations? In the Yager (1998) study, there were 214 observations containing only six discharge measurements; yet five of these six measurements were identified as being influential.
- Are the influential observations clustered in space or time? If influential observations are clustered, the investigator would then seek out the cause. For example, the clustering could indicate an important region of the model, or it might be an artifact of poor data sampling.
- Do the influential observations represent spatial or temporal outliers (i.e. are they distant from other observations)? If so, the calibration exercise may benefit from an additional round of data sampling near the influential observation.

2.2.7 Parameter Sensitivities

Whereas influential observations consider the effect of observations on parameter estimates, parameter sensitivity is a measure of parameter influence

on the value of simulated observations. Three parameter sensitivity measures are calculated in OSTRICH: dimensionless scaled sensitivities (**DSS**), composite scaled sensitivities (**CSS**), and one-percent scaled sensitivities (**1%SS**).

The **DSS** measure is a $m \times n$ matrix, where m is the number of observations and n is the number of parameters, and element DSS_{ij} is the dimensionless scaled sensitivity of simulated observation y_{sim_i} with respect to estimated parameter x_j, and is computed as (Hill, 1998):

$$DSS_{ij} = J_{ij} x_j w_i = \frac{\partial y_{sim_i}}{\partial x_j} x_j w_i \tag{2.22}$$

Where J_{ij} is an element of the Jacobian (also known as the sensitivity matrix), and w_i is the weight of observation i, equal to the square root of the i^{th} diagonal element of the **Q** matrix defined in equation (2.2b). Multiplying the sensitivity (J_{ij}) by the parameter value (x_j) has a scaling effect, while multiplication by w_i makes the result dimensionless (assuming that w_i is computed as $1/sd_i$, where sd_i is the standard deviation of the field-measured value y_{obs_i}).

1%SS are computed similar to **DSS**, except that the weighting term is omitted. Thus, elements of **1%SS** that refer to different observation types may have different units, and comparisons of such values is not meaningful. The formula to calculate one-percent scaled sensitivities is (Hill, 1998):

$$1\%SS_{ij} = J_{ij} \frac{x_j}{100} = \frac{\partial y_{sim_i}}{\partial x_j} \frac{x_j}{100} \tag{2.23}$$

Thus, **1%SS** is a $m \times n$ matrix whose elements $1\%SS_{ij}$ estimate the change in simulated observation y_{sim_i} as a result of a 1% increase in parameter x_j.

Composite scaled sensitivities (**CSS**) are an aggregate form of **DSS**, and are calculated as (Hill, 1998):

$$CSS_j = \sqrt{\frac{\sum_{i=1}^{m} (DSS_{ij})^2}{m}} \tag{2.24}$$

Where m is the number of observations. Composite sensitivities measure the overall sensitivity of the model to a given parameter. In general, very low values of CSS_j are an indication that the model is not sensitive to parameter x_j, and the calibration exercise will likely have difficulty obtaining reliable estimates of the parameter. Conversely, very high values of CSS_j

indicate high sensitivity to parameter x_j, and the calibration should be able to reliably estimate the value.

As noted by Madsen and Jacobsen (2001), the utility of the DSS, 1%SS and CSS parameter sensitivity measures is somewhat limited by their dependence on the estimated values of the parameters. For example, if the estimated value of a parameter is ≈ 0, then the scaled sensitivities could appear insensitive, even if the Jacobian value is large. Additional care must be taken if some (or all) of the estimated parameters are log-transformed. Therefore, these sensitivity measures should be evaluated carefully and should be cross-referenced with other measures (such as the Jacobian and standard parameter error).

2.2.8 Model Linearity

Several of the statistical and diagnostic measures discussed in this chapter (namely Cook's D, DFBETAS, and linear confidence intervals) are only accurate if the model in linear, or nearly linear, in the vicinity of the estimated parameter values. Therefore, various measures have been developed to test this linearity assumption. Beale (1960) developed a measure of non-linearity which selects several parameter sets and compares actual simulated observation values (calculated by executing the model) with observation values that are approximated using the assumption of linearity. Differences between actual and linearized values for each parameter set are aggregated and scaled to form a single scalar measure of non-linearity. Linssen (1975) made an adjustment to Beale's measure, correcting the behavior of the Beale measure in severely non-linear problems, while Bates and Watts (1980) developed a measure of non-linearity based on the geometric concept of relative curvature.

The present version of OSTRICH provides an implementation of both the Beale and Linssen (sometimes called the modified Beale) measures of model nonlinearity. Beale's measure (N) is expressed as (Christensen and Cooley, 1999):

$$N = ns^2 \frac{\sum_{p=1}^{q} \big(f(\mathbf{X}_p) - f_l(\mathbf{X}_p)\big)^T \mathbf{Q} \big(f(\mathbf{X}_p) - f_l(\mathbf{X}_p)\big)}{\sum_{p=1}^{q} \Big[\big(f(\mathbf{X}_p) - f(\mathbf{X}_{est})\big)^T \mathbf{Q} \big(f(\mathbf{X}_p) - f(\mathbf{X}_{est})\big)\Big]^2} \tag{2.25}$$

While the square of Linssen's measure (M^2) is computed as (Christensen

and Cooley, 1999):

$$M^2 = ns^2 \frac{\sum_{p=1}^{q} \big(f(\mathbf{X}_p) - f_l(\mathbf{X}_p)\big)^T \mathbf{Q} \big(f(\mathbf{X}_p) - f_l(\mathbf{X}_p)\big)}{\sum_{p=1}^{q} \Big[\big(f_l(\mathbf{X}_p) - f(\mathbf{X}_{est})\big)^T \mathbf{Q} \big(f_l(\mathbf{X}_p) - f(\mathbf{X}_{est})\big)\Big]^2} \tag{2.26}$$

Where $f_l(\mathbf{X}_p)$ is a vector of simulated observations obtained using the following linear approximation (Christensen and Cooley, 1999):

$$f_l(\mathbf{X}_p) = f(\mathbf{X}_{est}) + \mathbf{J}(\mathbf{X}_p - \mathbf{X}_{est}) \tag{2.27}$$

and n is the number of parameters, q is the number of parameter sets ($\mathbf{X}_p$) used to compute the non-linearity measure (N or M^2), s^2 is the error variance, $\mathbf{Q}$ is the weight matrix, as defined in equation (2.2b), $\mathbf{J}$ is the Jacobian matrix, $f(\mathbf{X}_p)$ is the vector of simulated observations obtained by executing the model using parameter set $\mathbf{X}_p$, and $f(\mathbf{X}_{est})$ is the vector of simulated observations corresponding to the regression-estimated parameter set ($\mathbf{X}_{est}$). Note that the equations for M^2 and N are quite similar; in fact, Linssen's modification was to replace the $f(\mathbf{X}_p)$ term in the denominator of N with $f_l(\mathbf{X}_p)$.

Cooley and Naff (1990) suggest that parameter sets ($\mathbf{X}_p$) be taken from points in parameter space that lie on the maximum and minimum 95% joint confidence limits for each parameter, yielding a total of $q = 2n$ parameter sets. The formula for calculating such parameter sets is (Cooley and Naff, 1990):

$$\mathbf{X}_{p_j} = \mathbf{X}_{est} \pm \frac{\sqrt{nF_\alpha(n, m-n)}}{se_j} \mathbf{V_j} \quad \text{where,} \tag{2.28}$$

$$\mathbf{V_j} = \begin{bmatrix} Cov_{1,j} & Cov_{2,j} & . & . & . & Cov_{n,j} \end{bmatrix}^T \tag{2.29}$$

Where $\mathbf{X}_{p_j}$ is the upper or lower (depending on the sign of the equation) joint confidence limit of parameter j, m is the number of observations, α is the significant level corresponding to the desired confidence interval, $F_\alpha(n, m-n)$ is the inverse CDF of an F-distribution containing n and $m-n$ degrees of freedom in the numerator and denominator, respectively, se_j is the standard error of parameter j, and $\mathbf{V_j}$ is the vector corresponding to the j^{th} column of the parameter variance-covariance matrix ($Cov_{j,j} = Var_j$).

Based on thresholds suggested by Beale (1960), OSTRICH reports the following linearity assessment for N and/or M^2:

- $> 1/F_\alpha(n, m-n)$: non-linear
- $< 0.01/F_\alpha(n, m-n)$: effectively linear

Since the Linssen and Beale measures are aggregate values, they are effectively an average measure of non-linearity. Therefore, low N and/or M^2 values suggest, *but do not guarantee*, the linearity of all parameters. Nonetheless, if these measures indicate a linear model, they lend an additional level of credibility to the use of confidence intervals, DFBETAS, and Cook's D measures. Conversely, if N and/or M^2 indicate a non-linear model, the user should regard the Cook's D and DFBETAS measures as qualitative, rather than quantitative, guides; and he/she also might want to consider alternative, non-linear, confidence intervals, such as those described by Diciccio and Romano (1988), Efron and Gong (1983), and Vecchia and Cooley (1987).

2.3 Unconstrained Numerical Optimization

This section describes the three unconstrained numerical optimization methods implemented by OSTRICH. These techniques are appropriate for optimization problems which satisfy the following criteria:

- All parameters are continuously varying.
- The user can bound the parameters such that he/she is confident that there is a single optimal parameter set within these boundaries. If the user cannot guarantee this condition, then the unconstrained optimization procedure can still be used, but should be repeated multiple times using different initial parameter sets. Figure 2.3 illustrates the aforementioned scenarios.
- The only constraints placed on the parameters are the upper and lower boundaries (so-called '*side constraints*'). If other constraints exist, the user may use unconstrained optimization techniques if the objective function can be modified to incorporate constraints via a penalty function approach. Vanderplaats (2001) discusses several penalty function techniques that utilize unconstrained optimization algorithms and fall under the general category of sequential unconstrained minimization techniques (SUMT). Use of SUMT requires iterative solution of more and more restrictive unconstrained optimization problems. At present, it is up to the user to perform these iterative OSTRICH optimizations, either manually or via a separate driver program.

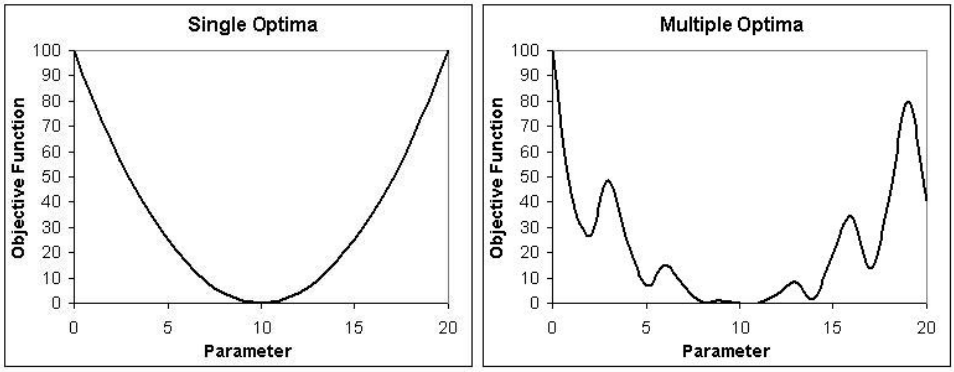

Figure 2.3: Single vs. Multiple Optima Objective Functions

The general algorithm for the numerical solution of unconstrained optimization problems is shown in Figure 2.4, (Vanderplaats, 2001). Where $\mathbf{X}$ is the vector of parameters to be optimized, $\mathbf{X}_0$ is the vector of initial parameter values, Φ_{old} and Φ_{new} are the values of the objective function from the previous and current iterations, respectively, $f(\mathbf{X})$ is computed by executing the model using parameter set $\mathbf{X}$, $\mathbf{S}$ is the search direction, a vector which describes the n-dimensional (where n is the number of parameters) direction in which the algorithm will move next, and α^* is the optimal step which should be taken in direction $\mathbf{S}$. The unconstrained optimization algorithms

1. Initialize algorithm, $\mathbf{X} = \mathbf{X}_0$, and $\Phi_{old} = f(\mathbf{X})$
2. Compute search direction $\mathbf{S}$
3. Compute optimal step (α^*) using one-dimensional search
4. Compute $\mathbf{X} = \mathbf{X} + \alpha^*\mathbf{S}$
5. Evaluate $\Phi_{new} = f(\mathbf{X})$
6. Test for convergence by comparing Φ_{old} and Φ_{new}
   (a) If converged, iteration is complete. $\mathbf{X}$ minimizes $f()$.
   (b) If not converged, set $\Phi_{old} = \Phi_{new}$ and return to Step 2.

Figure 2.4: Iterative Procedure for Unconstrained Optimization

described in the following sections are differentiated by the way in which the search direction is computed (Step 2 of Figure 2.4).

2.3.1 Zero-Order Methods

Zero order methods are a general class of unconstrained optimization methods which do not utilize derivative information. Determination of the search direction (Step 2 in Figure 2.4) is based solely upon evaluation of the objective function.

Powell's Method

Powell's method utilizes the concept of conjugate directions to determine search directions. Two directions, $\mathbf{S_1}$ and $\mathbf{S_2}$, are conjugate if (Vanderplaats, 2001):

$$\mathbf{S_1}^T \mathbf{H} \mathbf{S_2} = 0 \quad \text{where,} \tag{2.30}$$

$$\mathbf{H} = \nabla^2 \Phi = \begin{bmatrix} \frac{\partial^2 f(\mathbf{X})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{X})}{\partial x_1 \partial x_2} & \cdot & \cdot & \cdot & \frac{\partial^2 f(\mathbf{X})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{X})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{X})}{\partial x_2^2} & \cdot & \cdot & \cdot & \frac{\partial^2 f(\mathbf{X})}{\partial x_2 \partial x_n} \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & & \cdot & \cdot \\ \frac{\partial^2 f(\mathbf{X})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{X})}{\partial x_n \partial x_2} & \cdot & \cdot & \cdot & \frac{\partial^2 f(\mathbf{X})}{\partial x_n^2} \end{bmatrix} \tag{2.31}$$

Where $\mathbf{H}$ is the Hessian matrix of 2^{nd} order partial derivatives of the objective function $\Phi = f(\mathbf{X})$ with respect to some n-dimensional parameter set, $\mathbf{X} = [x_1 \; x_2 \; \ldots \; x_n]^T$.

In Powell's method, n orthogonal search directions are initially assigned, such that:

$$\mathbf{S_1} = \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \; \mathbf{S_2} = \begin{bmatrix} 0 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \; \cdot \; \cdot \; \cdot \; \mathbf{S_n} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} \tag{2.32}$$

Powell's method then 'moves' in each of these directions by applying Steps 3-6 in Figure 2.4 for each direction, at which point a new, conjugate, direction is computed: $\mathbf{S_{n+1}} = \mathbf{X_0} + \mathbf{X_n}$, where $\mathbf{S_{n+1}}$ is the new search direction, $\mathbf{X_0}$ is the initial parameter set, and $\mathbf{X_n}$ is the parameter set after moving in the n^{th} search direction. Next, a move is made in the new, $\mathbf{S_{n+1}}$, search

direction and a shift is performed on the search directions, such that:

$$\begin{aligned} \mathbf{S_1} &= \mathbf{S_2} \\ \mathbf{S_2} &= \mathbf{S_3} \\ \mathbf{S_n} &= \mathbf{S_{n+1}} \end{aligned} \tag{2.33}$$

This process of making n moves, computing and moving in conjugate direction $(n+1)$, and shifting the search directions constitutes one complete iteration of Powell's method, as shown in Figure 2.5.

1. Initialize algorithm (equation 2.32), $\mathbf{X} = \mathbf{X}_0$, and $\Phi_{old} = f(\mathbf{X})$
2. Move in each of the n search directions
3. Compute new search direction: $\mathbf{S_{n+1}} = \mathbf{X_0} + \mathbf{X_n}$
4. Move in the $\mathbf{S_{n+1}}$ search direction
5. Shift search directions (equation 2.33)
6. Return to Step 2

Figure 2.5: Iterative Procedure for Powell's Method

2.3.2 First-Order Methods

First order methods are a general class of unconstrained optimization methods that utilize derivative information, such that the search direction is based on the gradient of the objective function, which is approximated in OSTRICH using finite differences.

Steepest-Descent

The steepest-descent method computes the search direction (Step 2 of Figure 2.4) for any given iteration as the negative of the gradient:

$$\mathbf{S} = -\nabla\Phi = -\left[\frac{\partial f(\mathbf{X})}{\partial x_1} \quad \frac{\partial f(\mathbf{X})}{\partial x_2} \quad . \quad . \quad . \quad \frac{\partial f(\mathbf{X})}{\partial x_n}\right]^T \tag{2.34}$$

Where $\nabla\Phi$ is the gradient vector of partial derivatives of the objective function $\Phi = f(\mathbf{X})$ with respect to some n-dimensional parameter set, $\mathbf{X} = [x_1\ x_2\ \ldots\ x_n]^T$. Convergence of the steepest-descent method tends to slow to a crawl as the minimum is approached, and for this reason it is not appropriate for high-quality optimization or calibration. However,

if the user is only interested in ballpark parameter estimates or quick-and-dirty optimization, a few iterations of steepest-descent method can be very effective.

Fletcher-Reeve's Method

The Fletcher-Reeves method is similar to Powell's method in that it utilizes the concept of conjugate directions. However, whereas Powell's method requires n moves to be made prior to calculation of a new conjugate direction (see Section 2.3.1), the Fletcher-Reeves method uses gradient information ($\nabla\Phi$) to compute conjugate directions before every move. After using steepest-descent direction (equation 2.34) as the initial search direction, the Fletcher-Reeves method computes subsequent search directions as (Vanderplaats, 2001):

$$\mathbf{S} = -\nabla\Phi + \beta\mathbf{S_{old}} \tag{2.35}$$

Where, $\mathbf{S}$ and $\mathbf{S_{old}}$ are the revised and previous search directions, respectively, and β is a scalar multiplier. As suggested by Press et al. (1995), OSTRICH uses the Polak-Ribiere variant to compute β as:

$$\beta = \frac{(\nabla\Phi + \nabla\Phi_{old})^T(\nabla\Phi)}{(\nabla\Phi_{old})^T(\nabla\Phi_{old})} \tag{2.36}$$

Where, Φ_{old} is the gradient vector from the previous iteration. The appearance of Φ_{old} in the β calculation allows the Fletcher-Reeves method to utilize information across successive iterations. This feature makes the Fletcher-Reeves method highly efficient, assuming gradients can be reliably computed using finite differences.

2.3.3 One-Dimensional Search Procedures

As shown in Step 3 of Figure 2.4, unconstrained methods utilize a one-dimensional search to determine the optimal step size for a given search direction. Two one-dimensional search algorithms are implemented within OSTRICH: the Golden-Section Method and Brent's Method.

Golden-Section Method

The Golden-Section method uses the so-called '*Golden Ratio*' to subdivide a region that is known to bound the minimum into a smaller region that is also known to bound the minimum. Each subdivision generates a new region that is 38.2% smaller than the previous region. Successive subdivisions are

performed until the region bounding the minimum reaches an acceptably small value (e.g. 1×10^{-4}).

Brent's Method

Brent's Method is a hybrid of polynomial interpolation and Golden-Section techniques. At each iteration, Brent's method attempts to use polynomial interpolation to estimate the location of the minimum. Then, the algorithm computes the true value of the objective function at this estimated location and one side of the minimum bounding region is altered accordingly. If polynomial interpolation fails to reduce the bounding region by more than 38.2%, Brent's method instead uses the Golden-Section technique described previously.

2.4 Heuristic Optimization

Unconstrained numerical optimization methods (Powell, Steepest-Descent, and Fletcher-Reeves) and the Levenberg-Marquardt nonlinear regression method, utilize mathematical properties of the objective function (such as the gradient, the Jacobian matrix, or conjugate direction) to guide the search for the optimum parameter set. Conversely, heuristic optimization techniques are derived from processes that have been observed in the physical world. As such, these techniques do not have mathematically rigorous formulae, but rather are guided by empirical guidelines which attempt to simulate some real world optimization process. Additionally, whereas numerical optimization techniques are deterministic, the heuristic techniques described in this section incorporate elements of randomness, and can be viewed as directed random search techniques.

2.4.1 Genetic Algorithm

Genetic algorithms are based on Darwin's theories of natural evolution, such as inheritable traits and survival of the fittest. Before discussing the mechanics of a genetic algorithm, it is useful to first elaborate on some of the terminology used:

- population: Rather than operating on a single set of parameters, $\mathbf{X}$, during each iteration, genetic algorithms work with a population of parameter sets; $P_{GA} = \{\mathbf{X_1}, \mathbf{X_2}, ...\mathbf{X_p}\}$, where p is the population size.

- generation: Each iteration of the genetic algorithm corresponds to one generation.
- fitness: In a genetic algorithm, different parameter sets are compared with reference to their fitness functions, $F(\mathbf{X})$, such that better parameter sets have greater fitness functions. Since OSTRICH formulates the optimization or regression as a minimization problem, $F(\mathbf{X}) = -f(\mathbf{X})$, where $f(\mathbf{X})$ is the objective function to be minimized.
- selection: Selection is an operation wherein members are selected from the population for the purpose of crossover.
- crossover: The crossover operation is the process by which the parameter values of two population members are combined to create a new (i.e. child) set of parameter values.
- mutation: The mutation operation causes parameter values of a given population member to be randomly altered. Determination of whether a given parameter mutates is random and is based on a user-specified mutation probability.
- encoding: To facilitate crossover and mutation, genetic algorithms typically employ some form of encoding of the parameters; whereby parameters are stored in a form that may be different from the form used in the underlying model executable.

The basic idea behind a genetic algorithm is to iteratively modify a population of solutions by applying selection, crossover and mutation operations, as described in Figure 2.6. These operations are applied in such a way that the population will evolve towards the optimal solution over successive generations.

The following subsections explain the OSTRICH implementation of the selection, crossover and mutation operations along with a discussion of the parameter encoding method and the elitism enhancement.

Selection Operation

The first step in evolving a population from the current generation to the next generation is to select a set of members (known as the mating pool) from the current generation. This mating pool is then used as a basis for producing the next generation of solutions. Applying the concept of survival of the fittest, it is desirable that mating pool selection be biased towards

1. Initialize Population
   The initial population is assigned in a random fashion.
2. Preserve Elites
   Pass elite population members,unchanged, to the next generation.
3. Create Mating Pool
   Perform repeated selections to generate mating pool.
4. Perform Crossover
   Form a child by performing crossover on two members of the mating pool.
5. Perform Mutation
   Subject each parameter of the new child to the possibility of mutation.
6. Update Next Generation
   Pass new child along to the next generation.
7. If next generation is full, continue.
   Else, goto step 4.
8. Report average and best members of the new generation.
9. If current gen. num. $>$ max. gen., stop.
   Else, goto step 2.

Figure 2.6: Iterative Procedure for Genetic Algorithm

more fit members of the population. To this end, two methods are commonly employed: (a) roulette wheel selection and (b) tournament selection. OSTRICH presently implements a two-member tournament selection.

In tournament selection, a number of individuals (i.e. two) are chosen randomly from the population and the best individual from this group (based on comparison of fitness functions) is added to the mating pool. This process is repeated until the mating pool has been filled. OSTRICH uses a mating pool of size $(p - N)$, where p is the population size and N is the number of elites.

Encoding Methods: Real-Coded GA

The encoding method specifies the representation of parameters within the genetic algorithm. Depending on the type of optimization problem, different encoding methods can be employed; and the method chosen results in the utilization of different crossover and mutation operators. The real-encoding method uses double precision real variables to represent parameter values. This coding method allows continuous parameters to be encoded such that no manipulation is required. In this way, the real encoding method can be

thought of as a direct, or transparent, encoding of continuous parameters.

Crossover Operation: Real-Coded GA

Three crossover methods are typically considered for real encoded parameters. The first crossover method utilizes a biased weighting scheme that favors the more fit of the two members, and is expressed mathematically as:

$$\begin{aligned} x_1^c &= (0.5 + rnd_1)x_1^{p1} + (0.5 - rnd_1)x_1^{p2} \\ x_2^c &= (0.5 + rnd_2)x_2^{p1} + (0.5 - rnd_2)x_2^{p2} \\ &\cdot \\ &\cdot \\ &\cdot \\ x_n^c &= (0.5 + rnd_n)x_n^{p1} + (0.5 - rnd_n)x_n^{p2} \end{aligned} \tag{2.37}$$

Where, $\mathbf{X^c} = [x_1^c \quad x_2^c \quad . \quad . \quad . \quad x_n^c]^T$ is a vector of child parameter values, $\mathbf{RND} = [rnd_1 \quad rnd_2 \quad . \quad . \quad .rnd_n]^T$ ia a vector of randomly generated values, each in the range of $(0.00, 0.50)$, and $\mathbf{X^{p1}} = [x_1^{p1} \quad x_2^{p1} \quad . \quad . \quad . \quad x_n^{p1}]^T$ and $\mathbf{X^{p2}} = [x_1^{p2} \quad x_2^{p2} \quad . \quad . \quad . \quad x_n^{p2}]^T$ are vectors of parameter values of the most and least fit of the two members undergoing crossover, respectively.
The second crossover method uses a non-biased weighting scheme, whereby no favoritism is provided to more-fit members:

$$\begin{aligned} x_1^c &= rnd_1 x_1^{p1} + (1 - rnd_1)x_1^{p2} \\ x_2^c &= rnd_2 x_2^{p1} + (1 - rnd_2)x_2^{p2} \\ &\cdot \\ &\cdot \\ &\cdot \\ x_n^c &= rnd_n x_n^{p1} + (1 - rnd_n)x_n^{p2} \end{aligned} \tag{2.38}$$

Where the variables are as defined previously, except that the range of random values in **RND** is $(0.00, 1.00)$ instead of $(0.00, 0.50)$.
The final crossover type is a yes/no weighting scheme, whereby all of the value of a given parameter comes from just one member. For this calculation, equation (2.38) is used, except that random values in **RND** are either 1 or 0. In the current version of OSTRICH, the non-biased weighting scheme is used.

Mutation Operation: Real-Coded GA

Mutation is implemented on a per-parameter basis. For each parameter in a population member, a random number between 0 and 1 is generated and compared against the user-defined mutation probability (also called mutation rate). If the random number is less than the mutation probability, then the parameter is assigned a uniformly distributed random value chosen from the range of possible values for the given parameter.

Encoding Methods: Binary-Coded GA

The binary-encoding method encodes parameters into a string of binary bits. This coding method allows efficient GA manipulation of discrete parameters. An example of binary encoding of two parameters is given below:

$$P_1 = 156 \rightarrow 010011100$$
$$P_2 = 43 \rightarrow 000101011$$

Crossover Operation: Binary-Coded GA

For binary-coded variables, the crossover operation operates by performing between-parent bit swapping for each parameter string. First, a crossover bit location is randomly selected. Then the parents exchange the bits at this location and all subsequent locations in the given string. Finally, one of newly formed strings is chosen to represent the 'child' population member. A 2-parameter example of crossover between parents is given below:

$$P_{1,1} = 156 \rightarrow 0100111|00$$
$$P_{2,1} = 043 \rightarrow 0001010|11$$
$$C_1 = 040 \rightarrow 000101000$$

$$P_{1,2} = 037 \rightarrow 00001|00101$$
$$P_{2,2} = 085 \rightarrow 00010|10101$$
$$C_2 = 069 \rightarrow 0001000101$$

Mutation Operation: Binary-Coded GA

Mutation of binary-coded parameters is implemented on a per-parameter basis. If a random number is less than the mutation probability, then the bits in a randomly selected section of the string are 'flipped' (i.e. converted

from 1 to 0, or vice versa). An example of the mutation of two paramters is given below:

$$\begin{aligned} P_1 &= 156 \rightarrow 010011|100 \\ P_1^* &= 155 \rightarrow 010011011 \\ \\ P_2 &= 037 \rightarrow 00001|00101 \\ P_2^* &= 058 \rightarrow 0000111010 \end{aligned} \tag{2.39}$$

Elitism

The OSTRICH implementation of the genetic algorithm contains an enhancement known as elitism. Under the elitism enhancement, the top N most-fit members of the population are identified at the beginning of each generation; where N is a user specified parameter. These top N members are passed, unchanged, onto the next generation of solutions. In this way, the genetic material of the most elite solutions are protected from being lost or degraded by the selection, crossover and mutation operations.

2.4.2 Simulated Annealing

Simulated annealing is an algorithm derived from an analogy to the physical annealing of metals. In physical annealing, a metal is liquified by heating to a very high temperature. In this state, the molecules move about randomly and are unordered and highly energized. The liquid metal is then cooled very slowly, reducing thermal mobility such that a completely ordered, pure crystalline lattice is formed. It is important to cool the material very slowly, otherwise the molecules will not be given sufficient time to form an orderly lattice. If cooled slowly enough, the resulting crystal lattice represents the minimum possible energy state for the molecules. Simulated annealing generalizes this minimization process to arbitrary, numerically computed, objective functions.

Simulated annealing mimics physical annealing by first 'melting' the design space; whereby the objective function is evaluated at random locations so as to estimate the initial temperature of the system (which is based on the variance of the objective function). Next, the optimization undergoes equilibration, where a set of transitional moves is made while maintaining a constant 'temperature'; where 'temperature' is a control parameter that dictates the amount of randomness in the algorithm.

The idea behind equilibration is to find the best possible objective function at a given temperature. Once the system is equilibrated, the temperature is reduced slightly (10% is a typical reduction value) and the equilibration process is repeated at the new temperature. The process of equilibration and gradual temperature reduction is repeated until a user-specified maximum number of iterations is exceeded. Figure 2.7 illustrates how the various processes of simulated annealing are utilized in an iterative optimization algorithm.

1. Perform Melting Operation
   (a) Generate random parameter set ($\mathbf{X}$)
   (b) Evaluate and store $f(\mathbf{X})$
   (c) If num. melts > max. melts, goto Step 2. Else, return to (a).
2. Initialize Temperature (based on variance of melting operation)
3. Equilibrate at Current Temperature
   (a) Make transitional move via Metropolis (Figure 2.8)
   (b) If all moves made Select best and goto Step 4
   (c) Else return to (a)
4. Reduce Temperature
5. Report current parameter set.
6. If current iteration > max. iterations, stop. Else, goto Step 3

Figure 2.7: Iterative Procedure for Simulated Annealing

The equilibration process of simulated annealing is depicted in Step 3 of Figure 2.7. During equilibration, a series of transitional moves are made. These transitional moves are generated by successive applications of the Metropolis algorithm. The Metropolis algorithm randomly perturbs the current parameter values and decides whether to move to the location of the new parameter set or remain at the current location and try another perturbation. Within OSTRICH the three SA variants (Vanderbilt, Continuous, and Discrete) are differentiated by the manner in which the range of these perturbations are assigned:

- Vanderbilt: In this SA implementation, the range of perturbations is estimated based on the monte-carlo statistical approach of Vanderbilt and Louie (1984). When applied to problems involving continuous parameters, the method can be expected to speed SA convergence.

- Continuous and Discrete: In these SA implementations, the amount of parameter change induced by a given transitional move is conditioned to fall within a 'neighborhood' of adjacent solutions. For the Continuous SA implementation, the neighborhood is defined as +/-10% of the parameter range. For the Discrete SA, the neighborhood is defined as a parameter change of +/-1.

After all transitional moves have been made, the equilibration process chooses the best transitional move as the equilibrated parameter set and returns control back to the main simulated annealing control loop.

The Metropolis algorithm (shown in Figure 2.8) defines two criteria for accepting a given move: (a) if the move results in a decrease in the objective function, it is always accepted; and (b) the move is accepted if a randomly generated number between 0 and 1, $rand()$, satisfies: $rand() < exp(-\Delta f()/T)$, where $\Delta f()$ is the difference between the objective functions of the parameter set under consideration and the parameter set at the current location, and T is the temperature. Because of the 2^{nd} criteria, the Metropolis algorithm allows the equilibration process to accept "uphill" moves, or moves that can *increase* the objective function. This uphill movement gives the simulated annealing algorithm a measure of protection against being trapped in areas of local minima.

Inspection of the exponential term in the acceptance criteria of the Metropolis algorithm reveals that low values of temperature correspond to less randomness in the system (i.e. the left-hand-side of the acceptance test is driven to zero). At the limits, a temperature of 0 results in a completely deterministic algorithm (one that cannot make uphill moves), and a temperature of ∞ results in an undirected random search (one that accepts all moves). If the cooling process is too rapid, the algorithm may switch to a deterministic method too soon and will be unable to escape from a local minima. Therefore, like physical annealing, rapid cooling in simulated annealing is undesirable.

2.4.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was introduced by Kennedy and Eberhart (1995) as an outgrowth of their attempts to simulate the cooperative-

1. Generate random perturbation ($\Delta\mathbf{X}$)
2. Evaluate $f(\mathbf{X} + \Delta\mathbf{X})$
3. If $f(\mathbf{X} + \Delta\mathbf{X}) < f(\mathbf{X})$
   (a) Accept move ($\mathbf{X} = \mathbf{X} + \Delta\mathbf{X}$)
   (b) Return to equilibration process
4. Generate random number ($rand(0, 1)$)
5. Compute $\Delta f() = f(\mathbf{X} + \Delta\mathbf{X}) - f(\mathbf{X})$
6. If $rand() < exp(-\Delta f()/T)$
   (a) Accept move ($\mathbf{X} = \mathbf{X} + \Delta\mathbf{X}$)
   (b) Return to equilibration process
7. If num. attempts > max. attempts
   (a) Reject move
   (b) Return to equilibration process
8. Reject move and goto step 1.

Figure 2.8: Metropolis Algorithm

competitive nature of social behavior; they were attempting to replicate the flocking behavior of birds as they fly about in search of food. After successfully accomplishing this task, the simulation program was simplified into a general optimization algorithm. The PSO algorithm consists of a set of solutions known as the 'particle swarm'; each particle corresponds to one possible solution, and the entire set of particles makes up the swarm. During iteration, the particles 'fly' through the design space, using the following calculations to determine their new location (Beielstein et al., 2002):

$$\begin{aligned} v_j^{i+1} &= \chi(wv_j^i + c_1 r_1^i(p_{l_j}^i - x_j^i) + c_2 r_2^i(p_{g_j}^i - x_j^i)) \quad j = 1..n \\ x_j^{i+1} &= x_j^i + v_j^{i+1} \quad j = 1..n \end{aligned} \tag{2.40}$$

Where, i is the iteration number, n is the number of parameters, x_j, and v_j are the value and velocity of parameter j, respectively, χ is the constriction factor, w is the inertia weight, r_1 and r_2 are independent and uniformly distributed random numbers, c_1 is the cognitive parameter, the weight of a particles own experience, c_2 is the social parameter, the weight of the combined experience of the swarm, p_l is the parameter value corresponding to the best solution ever personally visited by the given particle, and p_g is the parameter value corresponding to the best solution ever visited by any particle (the current global best). Examination of equation (2.40) reveals three components for updating a particle: (a) the previous velocity, (b)

the current local best, and (c) the current global best. The particle will retain some fraction (w) of its previous velocity while moving in the general direction of the global (p_g) and local (p_l) best solutions. These directions are randomly weighted (r_1 and r_2) and scaled by their respective cognitive (c_1) and social (c_2) parameters. Figure 2.9 illustrates the particle swarm optimization iterative procedure.

1. Initialize Particle Swarm
   The initial swarm is assigned in a random fashion.
2. Move each particle using equation (2.40)
3. For each particle, revise current local best (p_l), if necessary.
4. Revise current global best p_g, if necessary.
5. Report average and globally best particle of the revised swarm.
6. If current iteration > max. iterations, stop.
   Else, goto step 2.

Figure 2.9: Iterative Procedure for Particle Swarm Optimization

Since its introduction, the PSO algorithm has been studied rather intensively and has been applied to a wide variety of applications. One of the appeals of PSO is that it has relatively few configurable parameters (compare PSO with the GA, which has a multitude of configuration parameters along with several choices for selection, crossover and mutation methods). Furthermore, PSO is straightforward to implement (just a few lines of vector computation) and, like the GA, is readily parallelized.

2.4.4 Exhaustive Search (GRID)

OSTRICH provides a means by which the optimization design space may be explored (possibly exhaustively) in a deterministic fashion via a gridding procedure. The GRID algorithm instructs OSTRICH to evaluate the objective function at regularly spaced parameter intervals and write the ensemble of results to a comma-separated output file (`OstGrid.csv`). The density of the multi-dimensional grid of evaluation points is selected by the user, and if the grid is sufficiently refined, the algorithm effectively operates as an exhaustive search. Alternatively, a courser grid can be useful for contouring the objective function surface and evaluating the non-linearity of the problem.

Chapter 3

Pump-and-Treat Optimization (PATO)

This chapter discusses the OSTRICH implementation of pump-and-treat optimization, an example of constrained optimization which has been studied extensively in recent years. The PATO module can be used to optimize various pump-and-treat strategies, including plume containment, mass removal, and even a combination of mass-removal and plume-containment. In fact, the PATO module is not limited to pump-and-treat remediation and is suitable for solving a variety of well-field design problems (i.e. problems that seek to identify optimal number, placement and pumping rates of wells).

3.1 Introduction to PATO

Remediation of sites containing contaminated groundwater is a continuing problem for the industrialized world. At many contaminated sites, a pump-and-treat system containing a series of extraction and injection wells is installed to prevent further plume migration and/or remove contaminant mass. Designers of such systems determine the number, location, and rates of extraction and injection wells such that the remediation goal is realized at the lowest cost. Given a modeling program which can adequately simulate plume development, the OSTRICH PATO module automates the process of pump-and-treat system design.

3.2 PATO Constraints

OSTRICH uses the output of the modeling program to determine whether or not the pump-and-treat constraints are being met. Constraints supported in the PATO module are:

- Capacity: Capacity constraints limit total pumping so that the PAT system does not overload an established treatment facility. Alternatively, capacity constraints can be used to enforce a lower limit on total pumping. For example, a lower limit may be used to ensure that the pump-and-treat system meets some minimum extraction rate established by a regulatory agency.

- Drawdown: Drawdown constraints prevent aquifer dewatering by limiting the drop in water table elevation caused by pumping of the wells.

- Plume Containment: These constraints ensure that a given pump-and-treat design prevents further plume migration. Depending on the type of modeling performed (recall that OSTRICH is a model independent package), different plume containment indicators may or may not be available. For example, if only flow modeling is used (e.g. via MODFLOW), then a standards compliance constraint will not be appropriate (such a constraint would require a mass-transport model, such as MT3D). The PATO module supports the following plume containment constraints:

  - Particle Capture: Particle tracking considers the advection of a series of particles, initially located inside or along the perimeter of the plume. Plume containment is suggested if all particles remain within plume boundaries at the conclusion of the remediation time frame.
  - Hydraulic Gradient Control: Plume containment using hydraulic gradient control is accomplished by specifying a series of control locations along the perimeter of the plume. When the head gradient at all control locations is oriented toward the interior of the plume, plume containment is presumed to occur.
  - Other: Other plume containment constraints (e.g. net flux out of the plume and/or standards compliance at monitoring points) can be realized via the general constraint format (see below).

- General Constraints: Mass removal constraints, well proximity constraints and any other constraints not listed previously can be implemented in the PATO module as General Constraints, so long as the modeling program outputs the required constraint information in a format parsable by OSTRICH.

3.3 Cost Formulations

PATO cost formulations can be divided into three categories: i) total pumping rate as a surrogate for cost, ii) operational costs only (i.e. energy, disposal and labor costs), iii) both operational and capital costs, and iv) the cost formulation of Mayer et al. (2002). These cost formulations, as implemented in the PATO module are described below.

3.3.1 Total Pumping Rate

$$C_{TOTQ} = \alpha_{ext} \sum_{i=1}^{n_{ext}} |Q_{i,ext}| + \alpha_{inj} \sum_{i=1}^{n_{inj}} |Q_{i,inj}| \tag{3.1}$$

where,
C_{TOTQ} is the cost as a function of total pumping rate
α_{ext} : cost conversion factor [\$ per L^3/T] for the total extraction rate
α_{inj} : cost conversion factor [\$ per L^3/T] for the total injection rate
n_{ext} : the number of extraction wells
n_{inj} : the number of injection wells
$Q_{i,ext}$: rate of extraction of well i.
$Q_{i,inj}$: rate of injection of well i

In the Ostrich input file, the user specifies the cost conversion factors (α_{ext} and α_{inj}).

3.3.2 Operational Costs

This cost formulation considers the operational costs of extraction and injection wells. Formulations for operational costs within OSTRICH are based on Mayer et al. (2002)(for energy costs) and RS Means ECHOS (2004)(for all other operational cost components).

$$C_{OPER} = [C_L + C_E + C_A + C_D + C_M]\, T \tag{3.2}$$

where,
C_{OPER} is the operational cost
C_L : annual labor cost
C_E : annual energy cost
C_A : annual analytic cost
C_D : annual disposal cost
C_M : annual maintenance cost
T : remediation time frame (in years)

Labor Cost

Labor costs account for the hourly rate charged by professionals who must maintain and operate the pump-and-treat system.

$$C_L = R_L\left(110\sqrt{\frac{NW}{3}}\right) \tag{3.3}$$

where,
R_L : the aggregate rate ($/hour) for all professional labor
NW : the number of active wells (both extraction and injection)

In the OSTRICH input file, the user specifies the labor rate (R_L).

Energy Cost

Energy costs reflect the electrical costs associated with operating the pumps of extraction and injection wells. For extraction wells, these costs are affected by the amount of lift required to pump the water out of the ground.

$$C_E = \beta_0 \sum_{i=1}^{n_{ext}} |Q_{i,ext}| \left(h_i - Z_{gs,i}\right) + \beta_1 \sum_{i=1}^{n_{inj}} |Q_{i,inj}| \tag{3.4}$$

where,
β_0 : annual energy cost conversion factor for extraction
β_1 : annual energy cost conversion factor for injection
h_i : the head at extraction well i
$Z_{gs,i}$: the ground surface elevation at the ith extraction well

In the OSTRICH input file, the user specifies the annual cost conversion factors (β_0 and β_1).

Analytic Cost

Analytic costs account for groundwater monitoring and sampling activities that are performed during the course of the remedial action.

$$C_A = R_A F_A \, (10 * NW) \tag{3.5}$$

where,
R_A : the analytic cost [\$/sample]
F_A : the sampling frequency [samples/year]

In OSTRICH, the user must specify the analytic cost rate (R_A) and the sample frequency (F_A).

Disposal Cost

Disposal costs are incurred when extracted groundwater is discharged to publicly operated treatment works (POTW). Because POTW charges are based on the volume of discharged water, disposal costs can become significant in high flow rate systems.

$$C_D = R_D\big(Q_{tot,ext} - Q_{tot,inj}\big) \tag{3.6}$$

where,
R_D : the disposal rate [\$ per year per L^3/T of net discharge]
$Q_{tot,ext}$: total extraction rate
$Q_{tot,inj}$: total injection rate

Note that this formulation assumes that injected water is recycled from extracted water, and such recycled water is not subject to disposal charges. In OSTRICH, the user specifies R_D, the annual per-volume disposal charge.

Maintenance Cost

Maintenance costs are associated with replacement of pumps, treatment components and other system parts that require regular service.

$$C_M = R_M \sum C_{CAP} F_Y \tag{3.7}$$

where,
R_M : the maintenance complexity factor
C_{CAP} : the capital cost

F_Y : a year-specific adjustment factor from RS Means ECHOS (2004)

Note that unless the user selects a cost formulation that includes capital costs (C_{OPER+}), C_M will be taken as zero. In OSTRICH, the user must specify the R_M and F_Y parameters.

3.3.3 Operational and Capital Costs

This cost function adds consideration of capital costs to the operational costs described previously (i.e. $C_{OPER+} = C_{OPER} + C_{CAP}$). Capital costs are a function of the cost of drilling a well and (for extraction wells) the cost of a pump capable of pumping at the desired rate and depth.

$$\begin{aligned} C_{OPER+} = C_{OPER} + \beta_2(NW) + \beta_3 \sum_{i=1}^{NW} \left(Z_{gs,i} - b_i\right) \\ + \sum_{i=1}^{n_{ext}} LKUP\Big(F_{C,Q}\left|Q_{i,ext}\right|, F_{C,H}\left(Z_{gs,i} - h_i\right)\Big) \end{aligned} \tag{3.8}$$

where,
C_{OPER+} is the operational and capital cost
β_2 : the per-well fixed construction cost
β_3 : the per unit length drilling cost
b_i : the aquifer base at well i
$LKUP$: a lookup table of pump costs
$F_{C,Q}$: units conversion factor for rate
$F_{C,H}$: units conversion factor for lift

The lookup table contains ranges of puming rate and lift along with associated pump costs. In this way, continuously varying lift and puming rate values can be mapped to cost estimates of discrete pump sizes. The OSTRICH software provides a default lookup table (in units of m^3/day for rate and meters for lift), based on the RS Means ECHOS (2004) publication. Users also have the option of explicity defining the table in the input file, by including a "LookupTable" section. Alternatively, users may supply appropriate values for $F_{C,Q}$ and $F_{C,H}$ to convert the OSTRICH table to an alternative set of units.

3.3.4 Mayer Costs

Mayer et al. (2002) provides a set of benchmark PATO problems, and the suggested cost formulation has been included in the OSTRICH PATO module, and is a combination of drilling and pumping costs.

$$C_{MAYER} = CF_{DRILL} \times NW + CF_{PUMP} \times n_{ext} \tag{3.9}$$

where,
CF_{DRILL} is the cost of drilling a well, and CF_{PUMP} is the cost of pumping (extraction wells only).

3.3.5 Time Value of Money

The operational (C_{OPER}), and operational and capital (C_{OPER+}) cost formulations do not account for the time-value of money. If the user wishes to include the time-value of money into cost calculations, then the time-frame term (T) is replaced with the following expression:

$$\frac{1 - (1 - R_I)^{-T}}{R_I} \tag{3.10}$$

Where, R_I is the interest rate (in decimal units) and T is the remediation time frame, in years. In OSTRICH, the user supplies values for both R_I and T. If R_I is set to zero (the default value), then the time-value of money will not be incorporated into the cost function.

3.4 Penalty Functions

The pump-and-treat objective function is mathematically formulated as a combination of the system cost function (either C_{TOTQ}, C_{OPER}, C_{OPER+}, or C_{MAYER}) and a penalty function, P_{TOTAL}, which accounts for the cost of various constraint violations. Equations for P_{TOTAL} and associated constraints are given below.

$$P_{TOTAL} = P_{CPCY} + P_{DRAW} + P_{PLUME} + P_{GEN} \tag{3.11}$$

where,
P_{TOTAL} is the total penalty due to various constraint violations.
P_{CPCY} : Penalty due to capacity constraint violations
P_{DRAW} : Penalty due to drawdown constraint violations
P_{PLUME} : Penalty due to plume capture constraint violations

P_{GEN} : Penalty due to general constraint violations

The following subsections detail the penalty functions for the various constraints.

3.4.1 Capacity Constraint Penalty Function

Capcity constraints limit the total system pumping rate so that the treatment system is not overloaded, and also so that some minimum pumping rate is enforced. Therefore, the penalty is computed as either: i) the difference between the total rate and the maximum allowable rate, or ii) the difference between the minimum rate and the total rate. This penalty is then scaled via a conversion factor to represent dollar units.

$$P_{CPCY} = \begin{cases} \beta_{CPCY}\left(Q_{min} - Q_{tot}\right) & \text{if } Q_{tot} < Q_{min} \\ \beta_{CPCY}\left(Q_{tot} - Q_{max}\right) & \text{if } Q_{tot} > Q_{max} \\ 0 & \text{if } Q_{min} \leq Q_{tot} \leq Q_{max} \end{cases} \tag{3.12}$$

where,
β_{CPCY} : converts the units of capcity violation (L^3/T) to dollars ($)
Q_{tot} : total pumping rate
Q_{max} : maximum allowable pumping rate
Q_{min} : minimum allowable pumping rate

In OSTRICH, the user specifies β_{CPCY}, Q_{max} and Q_{min}. Conversion factors are generally set large so that optimization algorithms will avoid pump-and-treat configurations that violate the constraint.

3.4.2 Drawdown Constraint Penalty Function

Drawdown constraints are applied to every active well in the PAT system. The summation of all drawdown constraint violations are multiplied by a penalty factor which converts drawdown units (L) to dollars ($).

$$\begin{aligned} P_{DRAW} &= \beta_{DRAW} \sum_{i=1}^{NW} P_{DRAW,i} \\ P_{DRAW,i} &= \begin{cases} \left(dh_i - dh_{max}\right) & \text{if } dh_i > dh_{max} \\ 0 & \text{if } dh_i \leq dh_{max} \end{cases} \end{aligned} \tag{3.13}$$

where,
β_{DRAW} is the penalty factor

$P_{DRAW,i}$ is the drawdown penalty at the ith well
dh_{max} is the maximum allowable drawdown ith well
dh_i is the drawdown at the ith well

In OSTRICH, the user specifies β_{DRAW} and dh_{max}. This formulation of drawdown requires OSTRICH to first run the groundwater model with no active wells, so that drawdown (defined as the change in head induced by pumping) is explicitly computed. Such a formulation is useful if there is an externally driven hard requirement on drawdown. Alternatively, the general constraint format can be used to limit drawdown. This formulation is useful if the only drawdown consideration is that the pumps must be fully submersed in order to operate properly. The general constraint format uses head at the wells (not the change in head), referenced against some minimum allowable head. The equation below illustrates this alternative drawdown formulation:

$$\begin{aligned} P_{DRAW} &= \beta_{DRAW} \sum_{i=1}^{NW} P_{DRAW,i} \\ P_{DRAW,i} &= \begin{cases} (h_{min} - h_i) & \text{if } h_i < h_{min} \\ 0 & \text{if } h_i \geq h_{min} \end{cases} \end{aligned} \tag{3.14}$$

where, h_{min} is the minimum allowable head a given well. In this formulation, the user configures OSTRICH with β_{DRAW} and h_{min}.

3.4.3 Plume Capture Constraint Penalty Function

The PATO mdule provides special constraint penalty formulations for hydraulic gradient control and particle capture plume containment strategies. Other plume containment constraint penalties can be incorporated as a general constraint penalty function.

$$P_{PLUME} = \beta_{HGRAD} \sum_{i=1}^{m} P_{HGRAD,i} + \beta_{PCAP} \sum_{i=1}^{n} P_{PCAP,i} \tag{3.15}$$

where,
β_{HGRAD} : penalty factor for gradient control constraints
β_{CAP} : penalty factor for particle capture constraints
m : number of gradient control constraints
n : number of particle capture constraints
$P_{HGRAD,i}$: violation of the ith gradient control constraint
$P_{PCAP,i}$: violation of the ith particle capture constraint

Hydraulic Gradient Control Penalty Function

The hydraulic gradient control penalty function examines the difference in head between control pairs, typically located along the perimeter of the plume. Each control pair is associated with outside and inside head values, and the constraint is that the inside head be less than the outside head (causing flow inward to the plume).

$$P_{HGRAD,i} = \begin{cases} \left(h_{in,i} - h_{out,i}\right) & \text{if } h_{iin,i} > h_{out,i} \\ 0 & \text{if } h_{iin,i} \leq h_{out,i} \end{cases} \tag{3.16}$$

where,
$h_{in,i}$: inside head at the ith control point
$h_{out,i}$: outside head at the ith control point

In OSTRICH, the user must specify β_{HGRAD} and the locations of the inside and outside control points of each control pair.

Particle Capture Penalty Function

Particle capture constraint violations are computed as the squared distance from a particles final resting position to the nearest plume boundary.

$$P_{PCAP,i} = \begin{cases} d_i^2 & \text{if particle outside plume} \\ 0 & \text{if particle inside plume} \end{cases} \tag{3.17}$$

where, d_i is the distance from the final position of the particle (following T years of particle tracking) to the plume boundary. The user must specify the starting coordinates of each particle, and provided a model executable that is capable of particle tracking (for exmple, a batch file that runs MODFLOW followed by MODPATH).

3.4.4 General Constraint Penalty Function

The constraints discussed in previous sections (i.e. gradient control, particle capture, drawdown, and capacity) are unique in that there is code built into the PATO module which assists in computing the penalty function. Examples of such unique built-in computation are:

- running the model with no active wells, for computation of drawdown
- summation of the parameters included in a cpacicty contraint

- computation of gradient differences for gradient control pairs
- computation of particle distances

Therefore, these constraint formulations calculate information about the pump-and-treat system which are not normally outputs of the underlying flow or transport model. Rather, the values of these special constraints are *derived* from nomal model output. Conversely, general constraints have values which can be read *directly* from model output. That is, general constraint penalties are computed as the difference between a model-output value and a user-specified target value. In this way, any numerical output from the model may be used to constrain the pump-and-treat optimization.

$$
\begin{aligned}
P_{GEN} &= \sum_{i=1}^{g} \beta_{GEN,i} P_{GEN,i} \\
P_{GEN,i} &= \begin{cases} (G_i - G_{max,i}) & \text{if } G_i > G_{max,i} \\ (G_{min,i} - G_i) & \text{if } G_i < G_{min,i} \\ 0 & G_{min,i} < G_i < G_{max,i} \end{cases}
\end{aligned}
\tag{3.18}
$$

where,
g is the number of general constraints
$\beta_{GEN,i}$ is the penalty factor for the ith general constraint
$P_{GEN,i}$ is the violation of the ith general constraint
G_i is the value of the constraint, read directly from model output
$G_{max,i}$ is the maximum allowable value of G_i
$G_{min,i}$ is the minimum allowable value of G_i

In OSTRICH, the user specifies the penalty factor and the minimum and maximum allowable values for each general constraint.

3.5 Objective Function

The OSTRICH PATO module offers several techniques for combining cost (C_{TOTQ}, C_{OPER}, C_{OPER+}, or C_{MAYER}) and P_{TOTAL} 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, as applied in the PATO module, are

given below:

$$\begin{aligned} F_{APM}(NW, \mathbf{Q}, \mathbf{X}, \mathbf{Y}) &= Cost + P_{TOTAL} \\ F_{MPM}(NW, \mathbf{Q}, \mathbf{X}, \mathbf{Y}) &= \max(Cost, P_{TOTAL})(1 + P_{TOTAL}) \\ F_{EPM}(NW, \mathbf{Q}, \mathbf{X}, \mathbf{Y}) &= \max(Cost, P_{TOTAL}) exp(P_{TOTAL}) \end{aligned} \tag{3.19}$$

where,
F_{APM} : objective function using APM
F_{MPM} : objective function using MPM
F_{EPM} : objective function using EPM
$\mathbf{Q}$: vector ($\mathbf{Q} = [Q_1, Q_2, ...Q_{NW}]^T$) of pumping rates
$\mathbf{Y}$: vector($\mathbf{Y} = [Y_1, Y_2, ...Y_{NW}]^T$) of y-coordinates
$\mathbf{X}$: vector($\mathbf{X} = [X_X, X_2, ...X_{NW}]^T$) of x-coordinates
$Cost$: one of C_{TOTQ}, C_{OPER}, C_{OPER+} or C_{MAYER}

In OSTRICH, the user provides ranges for the elements of $\mathbf{Q}$, $\mathbf{X}$, and $\mathbf{Y}$ in the parameter section of the input file. Additionally, the user can choose which method (APM, MPM, or EPM) should be used to combine the cost and penalty functions.

Chapter 4

Creating an OSTRICH Input File

This chapter discusses the input file syntax of the OSTRICH program. If OSTRICH is to be used for least-squares calibration, then the information in this chapter will provide the user with all the information needed to setup OSTRICH. If OSTRICH is to be used for general optimization, then the user should also read Chapter 7 for information on how to prepare a driver program to interface OSTRICH and the modeling program that is to be optimized. Furthermore, if OSTRICH is to be used for pump-and-treat optimization, then the user should also read Chapter 5. In any case, for OSTRICH to work with a given modeling program, the modeling program must meet the following requirements:

1. The modeling program must use a text-based input/output file format.
2. The modeling program must be able to run without prompting for user intervention (e.g. the modeling program cannot prompt the user to enter the name of an input file).
3. The output of the modeling program must be in a consistent format that can be reliably parsed.

4.1 Input File Organization

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, where each line of text in the input file is assumed to be no

longer than 160 characters. With a few exceptions (which will be explicitly noted in the following text) the basic format for a line of input is:

```
<variable>    <value>
```

Where `<variable>` is the name of the configuration variable and `<value>` is the user-selected value for the variable. The whitespace separating `<variable>` and `<value>` can be any number of spaces or tab characters, so long as the resulting line does not exceed 160 characters.

Aside from any required model template files, there is only one input file for OSTRICH, and it must be named `ostIn.txt` (Linux users: take note of the case). Inside `ostIn.txt`, the OSTRICH configuration variables are organized into groups, as follows:

- Basic Configuration: These variables describe the modeling program that is to be optimized or calibrated and identify the optimization (or regression) algorithm that OSTRICH should use.

- 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 regression) 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. Figure 4.1 illustrates this process.

- Extra Files: Extra files are model input files not used by OSTRICH, but required for proper execution of the model. In some parallel environments, OSTRICH will need to know about these extra input files.

- Observations: For calibration problems, the *Observations* group is used to list the observation names, values, and weights, along with parsing instructions for reading simulated observations from model output files.

- Parameters: This configuration group describes the parameters to be calibrated or optimized. Parameter configuration variables include names, initial values, lower and upper bounds, and input, output and internal transformations. Parameters in this section are real and continuously varying.

1. Given a set of parameters ($\mathbf{X}$), evaluate the objective function, $f(\mathbf{X})$
   (a) Use template files to generate model input files having desired parameters ($\mathbf{X}$)
   (b) Execute model program (or driver program, if optimizing)
   (c) Read model output files and compute the objective function, $f(\mathbf{X})$
      i. If calibrating: use observation variables to read simulated observations ($\mathbf{Y_{sim}}$) from model output files and compute WSSE objective function.
      ii. If using PATO, use response variables to read simulated response data from model output files and compute cost and penalty functions.
      iii. If optimizing: read objective function value from the driver program output file.
2. Proceed to next step of algorithm.

Figure 4.1: OSTRICH Usage of File Pairs

- 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.

- 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.

- Tied Parameters: Tied parameters are parameters which are computed as a function of integer, real or combinatorial parameter values.

$$X_{tied} = f_{tied}(X_1, X_2, ...X_n, c_1, c_2, ...c_m) \tag{4.1}$$

  Where, X_{tied} is the tied parameter value which is a function of n non-tied parameters (X_1,X_2,...X_n) and a set of m coefficients (c_1,c_2,...c_m), which depend on the functional form of $f_{tied}()$. Tied parameter configuration variables include: i) the name of the tied parameter, ii) a

list of the names of non-tied parameters used in the computation of the tied-parameter value, iii) a specification of the functional form of $f_{tied}()$, and iv) a list of coefficients used in the evaluation of $f_{tied}()$.

- Algorithms: Each algorithm has its own configuration group, wherein the user can specify the values for various algorithm variables.
- Math and Statistics: The variables in this group describe the finite difference method that OSTRICH will employ, if the chosen algorithm requires such computation. If calibration is being performed, additional variables in this group are used to request various statistical output.
- One-Dimensional 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 each of the unconstrained numerical optimization procedures described in Section 2.3.

Although the list of 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. Groups containing variables that must be configured by the user are: *Basic Configuration*, *File Pairs*, *Observations* (if calibrating), and *Parameters*. The following sections discuss the particular syntax required for each of the groups that may be included in the `ostIn.txt` file.

4.2 Basic Configuration

The following variables make up the basic configuration group:

- `ProgramType`: This variable tells OSTRICH which algorithm should be used to perform the optimization or calibration. The syntax is:

  ```
  ProgramType     <value>
  ```

  Where `<value>` can be any one of the following:

  - `GridAlgorithm`
  - `GeneticAlgorithm` : (this is the Real-coded GA)
  - `BinaryGeneticAlgorithm` : (this is the Binary-coded GA)
  - `ParticleSwarm`

  - `SimulatedAnnealing` : (this is the Continuous SA)
  - `DiscreteSimulatedAnnealing` : (this is the Discrete SA)
  - `VanderbiltSimulatedAnnealing` : (this is the Vanderbilt SA)
  - `Levenberg-Marquardt` : (this is the default)
  - `Powell`
  - `Steepest-Descent`
  - `Fletcher-Reeves`
  - `RegressionStatistics` : Computes regression statistics at the location specified by the initial parameter values.

- `ModelExecutable` : Specifies the model (or driver program, if optimizing) executable. The syntax is:

  `ModelExecutable  <value>`

  Where `<value>` is the full path and filename of the executable. On Windows-based PC's the path and filename may contain spaces; on Linux machines, spaces are problematic. 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` : In some parallel environments, creation of a dynamic subdirectory prevents parallel runs from clobbering each others input and output files. The syntax is:

  `ModelSubdir  <value>`

  If set to any value other than '`.`' (which is 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` : Allows the user to select the objective function to be optimized. The syntax is:

  `ObjectiveFunction  <value>`

  Where `<value>` can be any one of the following:

  - `WSSE` : Weighted sum of squared error calibration (this is default)

  - `PATO` : Pump-and-treat optimization
  - `GCOP` : General Constrained Optimization (see Chapter 6)
  - `USER` : User-defined objective function (see Chapter 7)

- `ParallelModelExec` : Tells OSTRICH to run the model in parallel. If this line is present in the `ostIn.txt` file, OSTRICH will check for available parallel resources and assemble and submit a PBS script to the parallel cluster whenever the model needs to be run. If the parallel cluster is unavailable or if needed PBS script commands (e.g. `qsub`, `showbf`) are unavailable, then OSTRICH will report the error and abort.

4.3 File Pairs

The syntax for this section is:

```
BeginFilePairs
<template1><sep><input1>
<template2><sep><input2>
.
.
.
<templateN><sep><inputN>
EndFilePairs
```

Where `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.

4.4 Observations

Note:The observations section is only needed when the optimization objective is calibration. The syntax for this section is:

```
BeginObservations
<name1>  <value1>  <weight1>  <file1><sep><keyword1>  <line1>  <col1>  <tok1>
<name2>  <value2>  <weight2>  <file2><sep><keyword2>  <line2>  <col2>  <tok2>
.
.
.
<nameN>  <valueN>  <weightN>  <fileN><sep><keywordN>  <lineN>  <colN>  <tokN>
EndObservations
```

Where `BeginObservations` and `EndObservations` are parsing tags that wrap a list of observations, which are made up of the following variables:

- `<name>` : The name of the observation, each observation should have a unique name.
- `<value>` : The field-measured value of the observation.
- `<weight>` : The weight assigned to the observation. See Section 4.4.1 for guidelines to assigning observation weights.
- `<file>` : The model output file where the simulated value of the observation will be stored following execution of the modeling program.
- `<keyword>`, `<line>`, and `<col>`: These variables tell OSTRICH how to extract model simulated observation values from the model output file. First, OSTRICH Positions the output file parser at the first line in `<file>` containing `<keyword>`. If OSTRICH should begin parsing at the beginning of the file, then `<keyword>` should be set to `OST_NULL`. Next, the parser uses the `<line>` and `<col>` values to locate the position of the desired observation value. This value is then extracted and converted to a double precision number. The parsing process is repeated until all observation values are read.

  The `<line>` variable tells OSTRICH how many lines must be skipped, starting from the line containing `<keyword>`, before the line containing the desired observation value is reached. Therefore, if the observation

value is on the same line as <keyword>, then <line>=0; if the observation value is on the line immediately following <keyword>, then <line>=1, and so on.

The <col> variable tells OSTRICH which column in the line contains the desired observation value; where each column is separated by the <tok> variable, and column numbering begins at 1.

The <sep> term is the filename separator described in the File Pairs section (Section 4.3), and the <tok> variable is used to specify an alternative column separator to use when parsing the model output file; the default column separator is whitespace (any number of space or TAB characters). If model output uses an alternative format (such as comma separated values), then the <tok> variable should be set accordingly.

Figure 4.4 illustrates the parse procedure using an example observation list (Figure 4.2) and observation output file (Figure 4.3).

```
#Observation Configuration
ObsToken    ,
BeginObservations
#name  value  weight  file         keyword   line  col
obs1   68.23  1       headerr.dat  computed  2     3
obs2   68.10  1       headerr.dat  computed  3     3
obs3   68.23  1       headerr.dat  computed  4     3
EndObservations
```

Figure 4.2: Example Observation Group

4.4.1 Assigning Observation Weights

There are several ways to assign observation weights, and the appropriate method depends on the particulars of the calibration problem. The most straightforward method is to assign all observations an equal value (typically 1), making the calibration an *un-weighted* sum of squared errors. This technique is commonly employed when all observations are of the same type (e.g. head observations) and the investigator is confident that all observations are of equal validity (i.e. there is no reason to suspect that some observations are more or less accurate than others).

```
OUTPUT OF ACME GROUP GROUNDWATER FLOW PROGRAM
LIST OF COMPUTED VERSUS MEASURED HEADS
-------------------------------------
x , y , computed head , measured head
-------------------------------------
17.64 , 77.24 , 68.3652 , 68.23
24.43 , 77.12 , 67.9723 , 68.10
17.32 , 70.64 , 68.3618 , 68.23
-------------------------------------
```

Figure 4.3: Example Model Output

1. General Parse Algorithm
   (a) Locate `<keyword>`
   (b) Skip `<line>` lines
   (c) Read and convert `<col>`th column
   (d) If num. obs. read = total num. obs., stop.
   Else, return to (a).
2. Application of Parse Algorithm to Figures 4.2 and 4.3
   (a) 1^{st} Iteration (Read `obs1`)
   Locate '`computed`' → '`computed head , measured head`'
   Skip 2 lines → '`17.64 , 77.24 , 68.3652 , 68.23`'
   Read and convert 3^{rd} column → obs1 value = 68.3652

   (b) 2^{nd} Iteration (Read `obs2`)
   Locate '`computed`' → '`computed head , measured head`'
   Skip 3 lines → '`24.43 , 77.12 , 67.9723 , 68.10`'
   Read and convert 3^{rd} column → obs2 value = 67.9723

   (c) 3^{rd} Iteration (Read `obs3`)
   Locate '`computed`' → '`computed head , measured head`'
   Skip 4 lines → '`17.32 , 70.64 , 68.3618 , 68.23`'
   Read and convert 3^{rd} column → obs3 value = 68.3618

Figure 4.4: OSTRICH Parse Algorithm for Model Output

When observations are of mixed types (e.g. head and flow observations), or if they are not of equal validity, then the weights should be assigned to reflect these discrepancies. Hill (1998) suggests assigning weights equal to $1/sd_{obs_i}$, where sd_{obs_i} is the standard deviation of the i^{th} observation and is

a measure of the error or uncertainty inherent in the observation. Units of sd_{obs_i} are the same as obs_i, so a weight of $1/sd_{obs_i}$ results in a dimensionless WSSE objective function; resolving the problem of mixed observation types. Additionally, a weighting of $1/sd_{obs_i}$ causes observations with low uncertainty to be more influential than observations that have a high amount of uncertainty, thus compensating for varying levels of observation uncertainty.

One problem with the $1/sd_{obs_i}$ weighting technique is that the investigator may not have sd_{obs_i} readily available. Hill (1998) suggests some techniques for estimating sd_{obs_i} based on professional judgement. The general idea is to estimate a range (R) of possible values for the observation in question and treat this range as a 95% confidence interval. If the observation is assumed to be normally distributed, then this 95% CI can be used to obtain an estimate of standard deviation:

$$sd_{obs_i} = \frac{R}{2Z_{0.95}^{-1}} \tag{4.2}$$

Where $Z_{0.95}^{-1}$ is the inverse cdf of a normally distributed variable, computed at the 95% confidence level ($Z_{0.95}^{-1} \approx 1.96$). The following list contains some suggestions for estimating the range:

- If multiple observations of the same type (e.g. multiple head observations) are available, then the combined range of these observations can be used to approximate R. For example, if the investigator has 100 head observations ranging from 100 to 300 meters, then $R = 300 - 100 = 200$ meters.
- If the investigator knows from past experience that the observations are accurate $\pm$ some value, then twice this value could be used as an estimate of the range.
- If the investigator has a sense of the possible high and low values (due to seasonal fluctuations, for example) of a given observation, the difference could be used as a range estimate.
- If measurements are taken from published sources of known data quality (e.g. USGS digital elevation maps), the data quality information can be used to estimate the range.

Sometimes observations are of the same type, but have dramatically different magnitudes. For example, in aqueous and soil chemistry, it is not uncommon to have some species with concentrations on the order of [mg/L],

while other species may be on the order of [μg/L] or even [pg/L]. In such cases, calibration using consistent units and uniform weighting can mute the influence of the low-concentration species, which are typically the species greatest interest. Therefore, two alternatives may be considered: (a)express concentrations in different units, but use uniform weighting; or (b) use consistent units, but assign greater weight to low-concentration species.

4.5 Parameters

The syntax for this section is:

```
BeginParams
<name1>  <initVal1>  <lowBnd1>  <uprBnd1>  <txIn1>  <txOst1>  <txOut1>
<name2>  <initVal2>  <lowBnd2>  <uprBnd2>  <txIn2>  <txOst2>  <txOut2>
.  .  .
<nameN>  <initValN>  <lowBndN>  <uprBndN>  <txInN>  <txOstN>  <txOutN>
EndParams
```

Where `BeginParams` and `EndParams` are parsing tags that wrap a list of 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).
- `<initVal>` : Initial value of the parameter, in units specified by `txIn`.
- `<lowBnd>` : Lower bound of the parameter, in units specified by `txIn`.
- `<uprBnd>` : Upper bound of the parameter, in units specified by `txIn`.
- `<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. log_{10} or log_e) 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-covariances, (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.

4.6 Integer Parameters

The syntax for this section is:

```
BeginIntegerParams
<name_1>  <init_1>  <low_1>  <upr_1>
<name_2>  <init_2>  <low_2>  <upr_2>
.  .  .
<name_N>  <init_N>  <low_N>  <upr_N>
EndIntegerParams
```

Where `BeginIntegerParams` and `EndIntegerParams` are parsing tags that wrap a list of integer 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).
- `<init>` : Initial value of the parameter.
- `<low>` : Lower bound of the parameter.
- `<uprBnd>` : Upper bound of the parameter.

4.7 Combinatorial Parameters

The syntax for this section is:

BeginCombinatorialParams
<name$_1$> <type$_1$> <init$_1$> <n$_1$> <val$_1$><TAB><val$_2$><TAB>...<val$_{n_1}$>
<name$_2$> <type$_2$> <init$_2$> <n$_2$> <val$_2$><TAB><val$_2$><TAB>...<val$_{n_2}$>
. . .
<name$_N$> <type$_N$> <init$_N$> <n$_N$> <val$_N$><TAB><val$_N$><TAB>...<val$_{n_N}$>
EndCombinatorialParams

Where BeginCombinatorialParams and EndCombinatorialParams are parsing tags that wrap a list of combinatorial 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).
- <type> : The type used in representing the combinatorial parameter. Valid type values are:
  - real : If this type is selected, values of init, and <val$_1$>,<val$_2$>,...,<val$_n$> must be real, and input in either decimal or scientific notation. This type would be appropriate if the model parameter could take on a finite number of real values (for example, the thickness of a material may only be available in 1/16, 1/8, 1/4 and 1/2 inch sizes).
  - integer : If this type is selected, values of init, and <val$_1$>,<val$_2$>,...,<val$_n$> must be entered as integers.
  - string : If this type is selected, values of init, and <val$_1$>,<val$_2$>,...,<val$_n$> must be entered as text strings. For example, a combinatorial model parameter made up of string types might be used to enable and disable different aspects of a model by inserting appropriate text in the model input file.
- <init> : Initial value of the parameter.
- <n> : Number of combinations to consider.
- <val$_1$>,<val$_2$>,...,<val$_n$> : A list of the n combinatorial values that are valid for the given parameter. Values in the list must be separated by <TAB> characters.

4.8 Tied Parameters

The syntax for this section is:

BeginTiedParams
<name$_1$> <np$_1$> <pname$_1$> <pname$_2$>...<pname$_{np_1}$> <type$_1$> <type_data$_1$>
<name$_2$> <np$_2$> <pname$_1$> <pname$_2$>...<pname$_{np_2}$> <type$_2$> <type_data$_2$>
. . .
<name$_N$> <np$_N$> <pname$_1$> <pname$_2$>...<pname$_{np_N}$> <type$_N$> <type_data$_N$>
EndTiedParams

Where 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 names 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.
- <pname$_1$>,<pname$_2$>...<pname$_n$p> : The list of non-tied parameter names that are used in the computation of the tied-parameter.
- <type> : The type of function relationship between the tied and non-tied parameters. Valid values for <type> are:
  - linear : A linear relationship between the tied and non-tied parameter(s). If this choice is selected, the value of <np> must be either 1 or 2.
  - exp : An exponential relationship between the tied and non-tied parameter. If this choice is selected, the value of <np> must be 1.
  - log : A log relationship between the tied and non-tied parameter. If this choice is 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 this choice is selected, the value of <np> must be 4, and the ordering of parameter names should correspond to (x_1,y_1),(x_2,y_2).

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

- `<type_data>` : Depending on the choice of `<type>`, the syntax of this field varies, as described below:

  - If `<type>` = "`linear`" and `<np>` = `"1"` : The functional relationship is linear and has the form:

$$X_{tied} = c_0 + c_1 X \tag{4.3}$$

    where, X_{tied} is the tied-parameter value, c_0 and c_1 are coefficients and X is the non-tied parameter value. `<type_data>` should be replaced with the following syntax:

`<c1>` `<c0>`

  - If `<type>` = "`linear`" and `<np>` = `"2"` : The functional relationship has the form:

$$X_{tied} = c_3 X_1 X_2 + c_2 X_2 + c_1 X_1 + c_0 \tag{4.4}$$

    where, X_{tied} is the tied-parameter value, c_0, c_1, c_2, and c_3 are coefficients, and X_1 and X_2 are the non-tied parameter values. `<type_data>` should be replaced with the following syntax:

`<c3>` `<c2>` `<c1>` `<c0>`

  - If `<type>` = "`exp`" : The functional relationship has the form:

$$X_{tied} = c_2 b^{(c_1 X)} + c_0 \tag{4.5}$$

    where, X_{tied} is the tied-parameter value, c_0, c_1 and c_2 are coefficients, b is the exponent base, and X is the non-tied parameter value. `<type_data>` should be replaced with the following syntax:

`<base>` `<c2>` `<c1>` `<c0>`

    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:

$$X_{tied} = c_3 log_a(c_2 X + c_1) + c_0 \tag{4.6}$$

where, X_{tied} is the tied-parameter value, c_0, c_1, c_2 and c_3 are coefficients, a is the logarithm base, and X is the non-tied parameter value. <type_data> should be replaced with the following syntax:

<base> <c_3> <c_2> <c_1> <c_0>

where <base> can be a numerical value, or ln if the natural logarithm is to be used.

– If <type> = "dist" : If the distance type is selected, the <type_data> field is not parsed and may be omitted.

– If <type> = "wsum" : If the weighted sum type is selected, the <type_data> field should list the values of each weight, using the same ordering as the preceding named list of parameters.

4.9 Extra Files

The syntax for this section is:

```
BeginExtraFiles
<file1>
<file2>
.
.
.
<fileN>
EndExtraFiles
```

Where BeginExtraFiles and EndExtraFiles are parsing tags that wrap a list of extra model input files. If the model is to be executed in a dynamically generated subdirectory (as specified by the ModelSubdir variable), then extra files must be identified so that OSTRICH can copy them to the subdirectory. For serial execution, creation of a dynamic subdirectory is unnecessary, and specification of the extra files section is not required. However, in some parallel environments (such as a dual-processor environment

with shared file space), creation of a dynamic subdirectory (and therefore specification of extra files) is needed so that parallel model runs don't clobber each others input and output files.

4.10 Algorithms

The following sub-sections describe the configuration of the various algorithms that are available within OSTRICH. When OSTRICH parses the input file, only the algorithm section which matches the `ProgramType` variable will be evaluated. Also, OSTRICH will use reasonable default values for any algorithmic parameters that are not specified by the user in the algorithm section.

4.10.1 Levenberg-Marquardt

The syntax for this section is:

```
BeginLevMar
InitialLambda                <value>
LambdaScaleFactor            <value>
MoveLimit                    <value>
AlgorithmConvergenceValue    <value>
LambdaPhiRatio               <value>
LambdaRelReduction           <value>
MaxLambdas                   <value>
MaxIterations                <value>
EndLevMar
```

Where `BeginLevMar` and `EndLevMar` are parsing tags that wrap a list of configuration variables:

- `InitialLambda` : Initial Marquardt λ (default is 10.00)
- `LambdaScaleFactor` : Marquardt λ scale factor; the factor by which λ is multiplied or divided during λ adjustment. (default is 1.10)
- `MoveLimit` : Parameter move limits; the maximum adjustment of a parameter (relative to the range of the parameter) that is allowed in a single iteration. (default is 0.10)

- `AlgorithmConvergenceValue` : Algorithm convergence value; regression will stop when the relative reduction in Φ over two iterations is less that this value. (default is 1E-4)
- `LambdaPhiRatio` : Φ reduction criteria for deciding on optimal λ adjustments; when relative reduction in Φ is greater than this value, λ adjustment for the given iteration is complete. (default is 0.30)
- `LambdaRelReduction` : Φ reduction criteria for abandoning λ adjustment; when relative reduction in Φ is less than this value, λ adjustment for the given iteration is halted. (default is 0.01)
- `MaxLambdas` : Max. λ adjustments per iteration. (default is 10)
- `MaxIterations` : Max. iterations in overall method. (default is 30)

4.10.2 Powell's Method

The syntax for this section is:

```
BeginPowellAlg
ConvergenceVal    <value>
MaxIterations     <value>
EndPowellAlg
```

Where `BeginPowellAlg` and `EndPowellAlg` are parsing tags that wrap a list of configuration variables:

- `ConvergenceVal` : Algorithm convergence value; optimization will stop when the relative reduction in Φ over three iterations is less that this value. (default is 1E-6)
- `MaxIterations` : Max. iterations in overall method. (default is 20)

4.10.3 Steepest-Descent

The syntax for this section is:

```
BeginSteepDescAlg
ConvergenceVal    <value>
MaxIterations     <value>
EndSteepDescAlg
```

Where `BeginSteepDescAlg` and `EndSteepDescAlg` are parsing tags that wrap a list of configuration variables:

- `ConvergenceVal` : Algorithm convergence value; optimization will stop when the relative reduction in Φ over two iterations is less that this value. (default is 1E-6)
- `MaxIterations` : Max. iterations in overall method. (default is 20)

4.10.4 Fletcher-Reeves

The syntax for this section is:

```
BeginFletchReevesAlg
ConvergenceVal    <value>
MaxIterations     <value>
EndFletchReevesAlg
```

Where `BeginFletchReevesAlg` and `EndFletchReevesAlg` are parsing tags that wrap a list of configuration variables:

- `ConvergenceVal` : Algorithm convergence value; optimization will stop when the relative reduction in Φ over three iterations is less that this value. (default is 1E-6)
- `MaxIterations` : Max. iterations in overall method. (default is 20)

4.10.5 Genetic Algorithm

The syntax for this section is:

```
BeginGeneticAlg
PopulationSize    <value>
MutationRate      <value>
Survivors         <value>
NumGenerations    <value>
ConvergenceVal    <value>
EndGeneticAlg
```

Where `BeginGeneticAlg` and `EndGeneticAlg` are parsing tags that wrap a list of configuration variables:

- `PopulationSize` : The population size. (default is 50)
- `MutationRate` : Mutation rate for child members. (default is 0.05)
- `Survivors` : Number of elites who pass unchanged to next generation. (default is 1)
- `NumGenerations` : Number of generations in optimization. (default is 10)
- `ConvergenceVal` : This is the convergence value for the algorithm. If the relative difference between the current minimum and the median of the latest generation is less than or equal to this value, the algorithm will halt. (default value is 0.0001)

4.10.6 Simulated Annealing

The syntax for this section is:

```
BeginSimulatedAlg
NumInitialTrials          <value>
TemperatureScaleFactor    <value>
OuterIterations           <value>
InnerIterations           <value>
ConvergenceVal            <value>
EndSimulatedAlg
```

Where `BeginSimulatedAlg` and `EndSimulatedAlg` are parsing tags that wrap a list of configuration variables:

- `NumInitialTrials` : This is the number of uphill moves that are attempted in the melting process. Larger values will result in more accurate estimates of the initial temperature, but at the expense of additional model runs. (default is 100)
- `TemperatureScaleFactor` : After each (outer) iteration, the temperature is reduced by multiplying by this value (therefore, `TemperatureScaleFactor` < 1.00). (default value is 0.90)
- `OuterIterations` : This is the number of iterations in the overall algorithm, where one outer iteration corresponds to one temperature reduction. (default value is 20)

- `InnerIterations` : This is the number of iterations in each temperature equilibration, where one inner iteration corresponds to a single transitional move. (default value is 10)
- `ConvergenceVal` : This is the convergence value for the algorithm. If the relative difference between the current minimum and the median of the latest series of equilibration moves is less than or equal to this value, the algorithm will halt. (default value is 0.001)

4.10.7 Particle Swarm Optimization

The syntax for this section is:

```
BeginParticleSwarm
SwarmSize               <value>
NumGenerations          <value>
ConstrictionFactor      <value>
CognitiveParam          <value>
SocialParam             <value>
InertiaWeight           <value>
InertiaReductionRate    <value>
ConvergenceVal          <value>
EndParticleSwarm
```

Where `BeginParticleSwarm` and `EndParticleSwarm` are parsing tags that wrap a list of configuration variables:

- `SwarmSize` : The size of the particle swarm. (default is 20)
- `NumGenerations` : The number of generations in the PSO. (default is 50)
- `ConstrictionFactor` : The value of χ in the PSO algorithm. Setting χ less than 1.00 will restrict the searchable design space after each iteration and accelerate convergence, but can lead to entrapment in local minima. (default is 1.00)
- `CognitiveParam` : The weight given to the local knowledge of each particle. High values (relative to the `SocialParam`) will cause particles to bias their search to the area surrounding each particles local best. (default is 2.00)

- `SocialParam` : The weight given to the global (social) knowledge of each particle. High values (relative to the `CognitiveParam`) will cause particles to bias their search to the area surrounding the global best.(default is 2.00)

- `InertiaWeight` : The weight (w) given to the velocity used in each particles previous generation of movement. High values tend to cause particles to 'overshoot' their destination, which is desirable in initial generations because it allows for more complete exploration of the design space. (default is 1.2)

- `InertiaReductionRate` : Relative reduction rate for w. As the optimization proceeds, w is reduced by `InertiaReductionRate` $* 100\%$ of its current value. This reduces overshoot over successive generations such that late-generation searches are clustered around the global best solution. If this value is set to `linear`, then the inertia weight is linearly reduced from it's initial value to a final value (i.e. at the last generation) of zero.(default value is 0.10)

- `ConvergenceVal` : This is the convergence value for the algorithm. If the relative difference between the current minimum and the median of the latest swarm evaluation is less than or equal to this value, the algorithm will halt. (default value is 0.001)

4.10.8 GRID Algorithm

The syntax for this section is:

```
BeginGridAlg
Dimensions    <d1>  <d2>  .  .  .  <dn>
EvalsPerIter  <value>
EndGridAlg
```

Where `BeginGridAlg` and `EndGridAlg` are parsing tags that wrap a list of configuration variables:

- `Dimensions` : The size of the multi-dimensional grid, where `<di>` is the dimension of the i-th parameter.

- `EvalsPerIter` : The number of model evaluations per 'iteration' of the GRID algorithm. If running in parallel, setting this value equal to the number of available processors will achieve the maximum speedup.

4.11 Math and Statistics

The syntax for this section is:

```
BeginMathAndStats
DiffType            <value>
DiffRelIncrement    <value>
CI_Pct              <value>
StdDev
StdErr
CorrCoeff
NormPlot
Beale
Linssen
CooksD
DFBETAS
Matrices
Confidence
Sensitivity
EndMathAndStats
```

Where `BeginMathAndStats` and `EndMathAndStats` are parsing tags that wrap a list of configuration variables:

- `DiffType` : Selects the method of computing finite differences. OSTRICH supports the following values for `DiffType`:
  - `forward` : Forward difference; finite diff. is computed using current parameter (x) and a forward step ($x + \Delta x$). (this is the default method)
  - `outside` : Central difference; finite diff. is computed using a forward step ($x + \frac{1}{2}\Delta x$) and a backward step ($x - \frac{1}{2}\Delta x$).
  - `parabolic` : Parabolic interpolation; finite diff. is computed from a parabola ($f = ax^2 + bx + c$) that is fitted through the current parameter, and forward and backward steps.
  - `best-fit` : Least-squares fit; finite diff. is computed via the least squares fit of a line ($f = ax + bx$) through points at the current parameter, and forward and backward parameter steps.

- `DiffRelIncrement` : Relative increment (or decrement) to use as the finite difference step size (Δx). Value is relative to the range of each parameter. (default is 0.01)

- `CI_Pct` : Confidence interval percentage level. Probability of the desired confidence interval. (default is 95.00)

- `StdDev` : If this line is present, OSTRICH will report the error variance and the standard error of the regression.

- `StdErr` : If this line is present, OSTRICH will report the parameter variance-covariance matrix and the standard error of each parameter.

- `CorrCoeff` : This line causes OSTRICH to report parameter correlation coefficients.

- `NormPlot` : This line causes OSTRICH to report a list of normalized residuals along with the R_N^2 correlation coefficient.

- `Beale` : This will cause OSTRICH to report Beale's measure of nonlinearity.

- `Linssen` : If this line is present, OSTRICH will report Linssen's measure of nonlinearity.

- `CooksD` : This line causes OSTRICH to report Cook's D measures of observation influence.

- `DFBETAS` : This line causes OSTRICH to report the DFBETAS measures of observation influence.

- `Matrices` : This line causes OSTRICH to output the Jacobian, Normal and inverse Normal matrices.

- `Confidence` : This will cause OSTRICH to report linear confidence intervals for each parameter along with the Volume Ratio (E/R) comparison of joint confidence ellipsoids and confidence blocks.

- `Sensitivity` : If present, OSTRICH will report parameter sensitivities.

4.12 One-Dimensional Search

The syntax for this section is:

```
Begin1dSearch
1dSearchConvergeVal     <value>
1dSearchMethod          <value>
End1dSearch
```

Where `Begin1dSearch` and `End1dSearch` are parsing tags that wrap a list of configuration variables:

- `1dSearchConvergeVal` : Convergence value for the one-dimensional search. (default is 1E-4)
- `1dSearchMethod` : Search method, either `Brent` or `GoldenSection` (`Brent` is the default)

4.13 Comment Lines

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. Sample comment lines can be found in the examples throughout Chapter 10.

4.14 Case Sensitivity

Variable names and section headings 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`.

Chapter 5

Using OSTRICH for Pump-and-Treat Optimization

This chapter discusses the various input file sections that must be configured to use OSTRICH for solving pump-and-treat optimization problems (see Chapter 3). When using OSTRICH for PATO, start by following the guidelines in Chapter 4 to fill out the following configuration groups:

- Basic Configuration
  - `ObjectiveFunction` : Be sure and set this variable to `PATO`.
  - `ModelExecutable` : Be sure to specify a model (or batch file) that will generate the necessary output for OSTRICH to evaluate constraint violations. If more than one executable is needed, (as in a MODFLOW/MODPATH particle tracking scenario) a batch file should be created and supplied as the model executable.
- File Pairs : Treat the template file pairs just as you would any optimization or calibration exercise. There should be a template file pair for any input files containing well parameters (i.e. pumping rate, x- and y- coordinates).
- Extra Files : Treat the extra files section just as you would any optimization or calibration exercise.
- Observations : The observations group is not used in PATO. Instead an analogous group, the *Response* group, is used. The Response group

is filled in which information on how to parse the model output files to retrieve information (such as particle locations) that is required for computation of the various penalty functions described in Section 3.4.

- Parameters : Include information on initial, upper and lower values for PATO parameters. Typically there will be three parameters for each well, corresponding to pumping rate and x- and y- coordinates. Coordinates may be continuous, integer or combinatorial parameters; the appropriate choice depends to some extent on the type of spatial discretization used by the underlying modeling program (e.g. analytic, finite difference or finite element). Pumping rates are typically treated as continuous variables, but integer or combinatorial parameterization could also be used.

- Tied Parameters : Set up any desired tied parameters as you would for any optimization/calibration exercise. Typical uses for tied parameters in PATO are:

  - Head at a Well : Some modeling programs cannot compute head at the exact location of a well. In such cases, tied parameters can be used to slightly offset the well location for the purposes of computing head at the well.
  - Conversion from global to local coordinates. Some modeling programs (such as MODFLOW) use a grid-based row-column indexing scheme for well-location, whereas user's may be more comfortable working in global coordinates. Thus, a tied parameter could be configured to convert from grid coordinates to global coordinates.

- Algorithms : Set up the desired optimization algorithm as you would any optimization/calibration exercise.

- Math and Statistics : Statistical output for PATO objective is not meaningful, but if a gradient-based algorithm is used, this section is useful for configuring finite-difference variables.

- One-dimensional Search : If appropriate for the chosen optimization algorithm, set up the one-dimensional search parameters as you would any optimization/calibration exercise.

Having fully configured the standard OSTRICH groups using the guidelines described previously, the next step is to configure a series of groups that are specific to the PATO module. These groups are:

- Response Variables : In this section, the user specifies the response variables that OSTRICH should read from model output files prior to evaluating PATO 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 Candidate Wells sections build upon the Response Variable group by associating individual response variables with a constraint or well parameter. Examples of PATO Response variables include:
  - Simulated Head Values : These may be computed at (or near) a well, to facilitate drawdown and lift calculations, or they may be located at inside and outside gradient control locations.
  - Particle Locations : These response variables would be used in the evaluation of particle capture constraints.
  - Other Model Output : could also be specified as response variables, if they are used for computation of a general constraint, as described in Section 3.4.4. For example, contaminant concentrations at compliance points could be specified as response variables, assuming such data is computed by the modeling program. These response variables could then be used in a general constraint that specifies some maximum concentration value at the compliance point.
- Pump-and-Treat : In the pump-and-treat section, the user specifies:
  - Cost Function : One of C_{TOTQ}, C_{OPER}, C_{OPER+}, or C_{MAYER}, described in Section 3.3
  - Penalty Function Method : One of APM, MPM or EPM, described in Section 3.5
  - Cost Coefficients associated with the chosen cost function.
- Constraints : In this section, the user supplies information about the various constraints that are to be placed on the pump-and-treat system. Any number and combination of the following five constraint types are currently supported: capacity, drawdown, hydraulic gradient, particle capture and general. The configuration syntax for constraints consists of: constraint name, constraint type, conversion factor (β-value), names of relevant response variables, and other constraint-specific information.

- Candidate Wells : The candidate wells sections is a list that associates individual well parameters (e.g. rate, and (x,y) coordinates) and related response variables (e.g. head and base elevation at the well) with a specific well.
- Plume Geometry : The plume geometry sections is required for PATO problems in which particle tracking constraints are used. This section contains a list of (x,y) vertices which describe the shape of one or more plumes.

Configuration syntax for the PATO-specific groups is described in detail in the following sections.

5.1 Response Variables

The general syntax for the response variables section is:

```
BeginResponseVars
<name_1>    <file_1><sep><key_1>    <line_1>    <col_1>    <tok_1>
<name_2>    <file_2><sep><key_2>    <line_2>    <col_2>    <tok_2>
.   .   .
<name_n>    <file_n><sep><key_n>    <line_n>    <col_n>    <tok_n>
EndResponseVars
```

Where `BeginResponseVars` and `EndResponseVars` are parsing tags that wrap a list of response variables, which are made up of the following variables:

- `<name>` : The name of the response variable, each should have a unique name.
- `<file>` : The model output file where the simulated value of the response variable will be stored following execution of the modeling program.
- `<key>`, `<line>`, `<col>`, and `tok`: These variables tell OSTRICH how to extract model simulated response variable values from the model output file. The parsing procedure is identical to that used in extracting Observation group data (see Section 4.4 for details).

5.2 Pump-and-Treat

The general syntax for the pump-and-treat section is:

```
BeginPumpAndTreat
CostFunction      <val>
PenaltyFunction   <val>
OnOffThreshold   <val>
#TOTQ Cost Coefficients
ExtRateCF       <val>
InjRateCF       <val>
#OPER/OPER+ Cost Coefficients
TimeFrame       <val>
InterestRate    <val>
LaborRate       <val>
ExtEnergyRate   <val>
InjEnergyRate   <val>
AnalyticRate    <val>
SampleFreq      <val>
DisposalRate    <val>
MaintFactor     <val>
#OPER+ Cost Coefficients
FixedWellCF     <val>
DepthDepWellCF  <val>
RateUCF         <val>
LiftUCF         <val>
Mayer Cost Coefficients
MayerDrillCF
MayerPumpCF
EndPumpAndTreat
```

Where `BeginPumpAndTreat` and `EndPumpAndTreat` are parsing tags that wrap a list of pump-and-treat cost configuration variables:

- `CostFunction` : The value of this parameter selects from the C_{TOTQ}, C_{OPER}, C_{OPER+}, and C_{MAYER} cost functions described in Section

3.3. As such, there are four options for this variable (default is `PumpRate`):

  - `PumpRate` : Selects C_{TOTQ} cost function
  - `OpCost` : Selects C_{OPER} cost function
  - `Cap&OpCost` : Selects C_{OPER+} cost function
  - `Mayer` : Selects C_{MAYER} cost function

  Note that if C_{OPER} or C_{OPER+} are selected, then OSTRICH will use the head at each well to compute the lift required to pull water from the bottom of the well to the ground surface. Therefore, if C_{OPER} or C_{OPER+} are selected, then the head at each well must be included in the response variables section.

- `PenaltyFunction` : The value of this parameter selected from the APM, MPM or EPM penalty function methods discussed in Section 3.5, and the three ooptions for this variable are `APM`, `MPM` and `EPM`. (default is `MPM`)

- `OnOffThreshold` : This optional line specifies a minimum active pumping rate. If a well is assigned a rate below `<val>`, the well is turned off by assigning a rate of zero.

- `ExtRateCF` : This cost factor applies when C_{TOTQ} is used, and corresponds to α_{ext} in Equation 3.1.(default is 0.00)

- `InjRateCF` : This cost factor applies when C_{TOTQ} is used, and corresponds to α_{inj} in Equation 3.1.(default is 0.00)

- `TimeFrame` : This is the remediation time frame (T) in years, and applies when either C_{OPER} or C_{OPER+} is used. (default is 0.00)

- `InterestRate` : This is the interest rate (R_I), if the time value of money is to be accounted for (see Equation 3.10), and applies when either C_{OPER} or C_{OPER+} is used. (default is 0.00)

- `LaborRate` : Corresponds to R_L in Equation 3.3 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)

- `ExtEnergyRate` : Corresponds to β_0 in Equation 3.4 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)

- `InjEnergyRate` : Corresponds to β_1 in Equation 3.4 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)

- `AnalyticRate` : Corresponds to R_A in Equation 3.5 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)
- `SampleFreq` : Corresponds to F_A in Equation 3.5 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)
- `DisposalRate` : Corresponds to R_D in Equation 3.6 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)
- `MaintFactor` : Corresponds to R_M in Equation 3.7 and is applicable for C_{OPER} and C_{OPER+} cost functions. (default is 0.00)
- `FixedWellCF` : Corresponds to β_2 in Equation 3.8 and is applicable for the C_{OPER+} cost function. (default is 0.00)
- `DepthDepWellCF` : Corresponds to β_3 in Equation 3.8 and is applicable for the C_{OPER+} cost function. (default is 0.00)
- `RateUCF` : Corresponds to $F_{C,Q}$ in Equation 3.8 and is applicable for the C_{OPER+} cost function. (default is 0.00)
- `LiftUCF` : Corresponds to $F_{C,H}$ in Equation 3.8 and is applicable for the C_{OPER+} cost function. (default is 0.00)
- `MayerDrillUCF` : Corresponds to CF_{DRILL} in Equation 3.9 and is applicable for the C_{MAYER} cost function. (default is 0.00)
- `MayerPumpUCF` : Corresponds to CF_{PUMP} in Equation 3.9 and is applicable for the C_{MAYER} cost function. (default is 0.00)

5.3 Constraints

The general syntax for the constraints section is:

```
BeginConstraints
#Particle Capture Constraints
<pname_1>   partcap   <PCF>   <pxloc_1>   <pyloc_1>   <plume_1>
<pname_2>   partcap   <PCF>   <pxloc_2>   <pyloc_2>   <plume_2>
...
<pname_n>   partcap   <PCF>   <pxloc_n>   <pyloc_n>   <plume_n>
#Hydraulic Gradient Constraints
<hname_1>   hydgrad   <HCF>   <hlwr_1>   <hupr_1>   <hout_1>   <hin_1>
<hname_2>   hydgrad   <HCF>   <hlwr_2>   <hupr_2>   <hout_2>   <hin_2>
...
<hname_m>   hydgrad   <HCF>   <hlwr_m>   <hupr_m>   <hout_m>   <hin_m>
#Capacity Constraints
<cname_1>   capacity   <CCF>   <clwr_1>   <cupr_1>   <param_1,1,param_1,2,...,param_1,N>
<cname_2>   capacity   <CCF>   <clwr_2>   <cupr_2>   <param_2,1,param_2,2,...,param_2,N>
...
<cname_c>   capacity   <CCF>   <clwr_c>   <cupr_c>   <param_c,1,param_c,2,...,param_c,N>
#Drawdown Constraints
<dname_1>   drawdown   <DCF>   <clwr_1>   <cupr_1>   <head_1>
<dname_2>   drawdown   <DCF>   <clwr_2>   <cupr_2>   <head_2>
...
<dname_d>   drawdown   <DCF>   <clwr_d>   <cupr_d>   <head_d>
#General Constraints
<gname_1>   general   <GCF_1>   <glwr_1>   <gupr_1>   <resp_1>
<gname_2>   general   <GCF_2>   <glwr_2>   <gupr_2>   <resp_2>
...
<gname_g>   general   <GCF_g>   <glwr_g>   <gupr_g>   <resp_g>
EndConstraints
```

Where `BeginConstraints` and `EndConstraints` are parsing tags that wrap a list of pump-and-treat constraints, which can be any number of and combination of five types (specified in the second field of each constraint), whose

syntax are described below:

- `Particle Capture Constraints` : If the second field in the constraint definition is `partcap`, then the constraint is interpreted as a particle capture constraint, with the following fields:
  - `<pname>` : A unique name for the constraint (e.g. `PCAP023`).
  - `<PCF>` : The cost factor (β_{PCAP} in Equation 3.15).
  - `<pxloc>` : The name of the response variable corresponding to the x-coordinate of the particle.
  - `<pyloc>` : The name of the response variable corresponding to the y-coordinate of the particle.
  - `<plume>` : The name of the plume, defined in the Plume Geometry section (Section 5.5), within which the particle should reside after T years of remedial activity.
- `Hydraulic Gradient Constraints` : If the second field in the constraint definition is `hydgrad`, then the constraint is interpreted as a hydraulic gradient control constraint, with the following fields:
  - `<hname>` : A unique name for the constraint (e.g. `HGRAD053`).
  - `<HCF>` : The cost factor (β_{HGRAD} in Equation 3.15).
  - `<hlwr>` : Users should set this value to 0.
  - `<hupr>` : Users should set this to a large value (e.g. 1,000).
  - `<hout>` : The name of the response variable corresponding to the head at the outside location of the gradient control pair ($h_{out,i}$ in equation 3.16).
  - `<hin>` : The name of the response variable corresponding to the head at the inside location of the gradient control pair ($h_{in,i}$ in equation 3.16).
- `Capacity Constraints` : If the second field in the constraint definition is `capacity`, then the constraint is interpreted as a capacity constraint, with the following fields:
  - `<cname>` : A unique name for the constraint (e.g. `TOTALQ`).
  - `<CCF>` : The cost factor (β_{CPCY} in Equation 3.12).
  - `<clwr>` : This is the lower limit on capacity, and corresponds to Q_{min} in Equation 3.12.

  - `<cupr>` : This is the upper limit on capacity, and corresponds to Q_{max} in Equation 3.12.
  - `<param`$_1$`>,<param`$_2$`>,...,<param`$_N$`>` : This is a comma-separated list of parameter names, whose values are summed up to form Q_{tot} in Equation 3.12.

- `Drawdown Constraints` : If the second field in the constraint definition is `drawdown`, then the constraint is interpreted as a drawdown constraint, with the following fields:
  - `<dname>` : A unique name for the constraint (e.g. `DRAW005`).
  - `<DCF>` : The cost factor (β_{DRAW} in Equation 3.13).
  - `<clwr>` : Users should set this value to 0.
  - `<cupr>` : Max allowable drawdown (dh_{max} in Equation 3.13).
  - `<head>` : The name of the response variable corresponding to the head at the drawdown location.

- `General Constraints` : If the second field in the constraint definition is `general`, then the constraint is interpreted as a general constraint, with the following fields:
  - `<gname>` : A unique name for the constraint (e.g. `CONC002`).
  - `<GCF>` : The cost factor ($\beta_{GEN,i}$ in Equation 3.18).
  - `<glwr>` : The lower constraint limit ($G_{min,i}$ in Equation 3.18).
  - `<gupr>` : The upper constraint limit ($G_{max,i}$ in Equation 3.18).
  - `<resp>` : The name of the response variable used to evaluate the constraint (corresponding to G_i Equation 3.18).

5.4 Candidate Wells

The general syntax for the candidate wells section is:

```
BeginCandidateWells
<name1>  <xloc1>  <yloc1>  <rate1>  <head1>  <surface1>  <base1>
<name2>  <xloc2>  <yloc2>  <rate2>  <head2>  <surface2>  <base2>
...
<namen>  <xlocn>  <ylocn>  <raten>  <headn>  <surfacen>  <basen>
EndCandidateWells
```
(5.1)

Where `BeginCandidateWells` and `EndCandidateWells` are parsing tags that wrap a list of pump-and-treat wells, whose rates and locations are to be optimized. The syntax of a given candidate well is described below:

- `<name>` : A unique name for the well (e.g. `WELL005`).
- `<xloc>` : The name of the parameter which represents the x-location of the given well.
- `<yloc>` : The name of the parameter which represents the y-location of the given well.
- `<rate>` : The name of the parameter which represents the rate of the given well.
- `<head>` : The name of the response variable corresponding to the head at the given well. Note that if C_{TOTQ} is used as the cost function, this variable is not read by OSTRICH and may be omitted.
- `<surface>` : The name of the response variable corresponding to the ground surface elevation at the given well. Alternatively, if this value is constant throughout the given model, no response variable is necessary and the user should enter the constant value instead. Note that if C_{TOTQ} is used as the cost function, this variable is not read by OSTRICH and may be omitted.
- `<base>` : The name of the response variable corresponding to the aquifer base elevation at the given well. Alternatively, if this value is constant for the given model, no response variable is necessary and the user should enter the constant value instead. Note that if C_{TOTQ} is used as the cost function, this variable is not read by OSTRICH and may be omitted.

5.5 Plume Geometry

The general syntax for the plume geometry section is:

```
BeginPlumeGeometry
PlumeName <plume_1>
BeginPlumeCoords
<x_{1,1}>   <y_{1,1}>
<x_{1,2}>   <y_{1,2}>
...
<x_{1,n}>   <y_{1,n}>
EndPlumeCoords
#Next Plume
PlumeName <plume_2>
BeginPlumeCoords
<x_{2,1}>   <y_{2,1}>
<x_{2,2}>   <y_{2,2}>
...
<x_{2,n}>   <y_{2,n}>
EndPlumeCoords
...
#Next Plume
PlumeName <plume_N>
BeginPlumeCoords
<x_{N,1}>   <y_{N,1}>
<x_{N,2}>   <y_{N,2}>
...
<x_{N,n}>   <y_{N,n}>
EndPlumeCoords
EndPlumeGeometry
```

(5.2)

Where `BeginPlumeGeometry` and `EndPlumeGeometry` are parsing tags that wrap a list of plume geometries. The syntax of a plume geometry is described below:

- `PlumeName` : A unique name for the given plume (e.g. TCE_PLUME)
- `BeginPlumeCoords` : A tag that indicates the beginning of the list of vertices for the plume.
- `<x>` : The x-coordinate of a plume vertex.
- `<y>` : The y-coordinate of a plume vertex.
- `EndPlumeGeometry` : A tag that indicates the end of the list of vertices for the plume.

This section is required if particle tracking is used, otherwise the Plume Geometry section may be omitted.

Chapter 6

Using OSTRICH for General Constrained Optimization

If OSTRICH is to be used for anything other than least-squares calibration or pump-and-treat optimization, the user has two options: i) generate a driver program (see Chapter 7) or ii) incorporate the alternative objective function into the OSTRICH input file via the GCOP module. This chapter discusses the various input file sections that must be configured to use OSTRICH for solving such GCOP problems. When using OSTRICH for GCOP, start by following the guidelines in Chapter 4 to fill out the following configuration groups:

- Basic Configuration
  - `ObjectiveFunction` : Be sure and set this variable to `GCOP`.
  - `ModelExecutable` : Be sure to specify a model (or batch file) that will generate the necessary output for OSTRICH to evaluate system cost and any constraint violations. If more than one executable is needed a batch/script file should be created and supplied as the model executable.
- File Pairs : Treat the template file pairs just as you would any optimization or calibration exercise. There should be a template file pair for any input files containing design parameters.
- Extra Files : Treat the extra files section just as you would any optimization or calibration exercise.
- Observations : The observations group is not used in GCOP. Instead an analogous group, the *Response* group, is used. The Response group

is filled with instructions for parsing model output files to retrieve (and compute) constraint and cost information.

- Parameters : Include information on initial, upper and lower values for GCOP parameters. There can be as many parameters as necessary, and they may be continuous, integer and/or combinatorial parameters; the appropriate choice is application specific.

- Tied Parameters : Set up any desired tied parameters as you would for any optimization/calibration exercise. A typical uses for tied parameters in GCOP is to convert a discrete-valued design parameter (used by the optimization) to its continuous analog (used by the simulation).

- Algorithms : Set up the desired optimization algorithm as you would any optimization/calibration exercise.

- Math and Statistics : Statistical output for GCOP is not meaningful, but if a gradient-based algorithm is used, this section is useful for configuring finite-difference variables.

- One-dimensional Search : If appropriate for the chosen optimization algorithm, set up the one-dimensional search parameters as you would any optimization/calibration exercise.

Having fully configured the standard OSTRICH groups using the guidelines described previously, the next step is to configure a series of groups that are specific to the GCOP module. These groups are:

- 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 build upon the Response and Tied Response Variable groups by associating response variables with a constraint or cost parameter.

- 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.

- GCOP : In the GCOP section, the user specifies:

- Cost Function : The Cost Function identifies a single response variable or tied response variable that represents the overall system cost (C_{SYS}), which is to be minimized by the optimizer.
- Penalty Function Method : The overall GCOP objective function is a combination of the system cost (C_{SYS}) and a penalty function, P_{TOTAL}, which accounts for the cost of all constraint violations. The OSTRICH GCOP module offers several techniques for combining C_{SYS} and P_{TOTAL} 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:

$$\begin{aligned} F_{APM}(\mathbf{X}) &= C_{SYS} + P_{TOTAL} \\ F_{MPM}(\mathbf{X}) &= \max(C_{SYS}, P_{TOTAL})(1 + P_{TOTAL}) \\ F_{EPM}(\mathbf{X}) &= \max(C_{SYS}, P_{TOTAL})exp(P_{TOTAL}) \end{aligned} \tag{6.1}$$

where,
F_{APM} : objective function using APM
F_{MPM} : objective function using MPM
F_{EPM} : objective function using EPM
$\mathbf{X}$: vector($\mathbf{X} = [X_1, X_2, ...X_N]^T$) of design parameters

- Constraints : In this section, the user supplies information about the various constraints that are to be placed on the user-defined 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.

Configuration syntax for the GCOP-specific groups is described in detail in the following sections.

6.1 Response Variables

The general syntax for the response variables section is:

```
BeginResponseVars
<name_1>    <file_1><sep><key_1>    <line_1>    <col_1>    <tok_1>
<name_2>    <file_2><sep><key_2>    <line_2>    <col_2>    <tok_2>
.   .   .
<name_n>    <file_n><sep><key_n>    <line_n>    <col_n>    <tok_n>
EndResponseVars
```

Where `BeginResponseVars` and `EndResponseVars` are parsing tags that wrap a list of response variables, which are made up of the following variables:

- `<name>` : The name of the response variable, each name should be unique.
- `<file>` : The model output file where the simulated value of the response variable will be stored following execution of the modeling program.
- `<key>`, `<line>`, `<col>`, and `tok`: These variables tell OSTRICH how to extract model simulated response variable values from the model output file. The parsing procedure is identical to that used in extracting Observation group data (see Section 4.4 for details).

6.2 Tied Response Variables

The general syntax for the tied response variables section is:

```
BeginTiedRespVars
<name_1>  <np_1>   <pname_1>  <pname_2>...<pname_np_1>   <type_1>   <type_data_1>
<name_2>  <np_2>   <pname_1>  <pname_2>...<pname_np_2>   <type_2>   <type_data_2>
.   .   .
<name_N>  <np_N>   <pname_1>  <pname_2>...<pname_np_N>   <type_N>   <type_data_N>
EndTiedRespVars
```

Where `BeginTiedRespVars` and `EndTiedRespVars` are parsing tags that wrap a list of tied response variables. The parameters in this section are

identical to those in the Tied Parameters (see Section 4.8), except that the list of non-tied items (used in the calculation of the tied response variable) may be parameters and/or response variables.

6.3 GCOP

The general syntax for the GCOP section is:

```
BeginGCOP
CostFunction      <val>
PenaltyFunction   <val>
EndGCOP
```

Where `BeginGCOP` and `EndGCOP` are parsing tags that wrap a list of GCOP cost configuration variables:

- `CostFunction` : The value of this parameter must be the name of a response variable (or tied response variable) and corresponds to the system cost (C_{SYS}).
- `PenaltyFunction` : The value of this parameter selected from the APM, MPM or EPM penalty function methods, and the three options for this variable are `APM`, `MPM` and `EPM`. (default is `MPM`)

6.4 Constraints

The general syntax for the constraints section is given below. Note that if no constraints exist, this section may be omitted.

```
BeginConstraints
<name_1>   general   <CF_1>   <lwr_1>   <upr_1>   <resp_1>
<name_2>   general   <CF_2>   <lwr_2>   <upr_2>   <resp_2>
...
<name_g>   general   <CF_g>   <lwr_g>   <upr_g>   <resp_g>
EndConstraints
```

Where `BeginConstraints` and `EndConstraints` are parsing tags that wrap a list of general constraints. The syntax of a given constraint is described below:

- `<name>` : A unique name for the constraint.
- `<CF>` : A cost factor that is multiplied by the amount of constraint violation. This converts a constraint violation into a penalty cost.
- `<lwr>` : The lower constraint limit (g_{min}). If the actual constraint value (g) is less than g_{min}, a penalty of $P = CF \times (g - g_{min})$ will be added to P_{TOTAL}.
- `<upr>` : The upper constraint limit (g_{max}). If the actual constraint value (g) is greater than g_{max}, a penalty of $P = CF \times (g_{max} - g)$ will be added to P_{TOTAL}.
- `<resp>` : The name of the response variable (tied or non-tied) used to evaluate the constraint.

Chapter 7

Using OSTRICH for General Optimization

Rather than configuring the GCOP input sections described in Chapter 6, some users may find it easier to generate a driver program which executes the simulation model and computes the objective function on behalf of OSTRICH, as shown in Figure 7.1. The driver program must write the objective

1. Given a set of parameters ($\mathbf{X}$), objective function, $f(\mathbf{X})$, is desired.
2. OSTRICH prepares model input file(s) using the template file(s).
3. OSTRICH runs the driver program.
   (a) Driver runs model program.
   (b) Driver reads model output and computes $f(\mathbf{X})$
   (c) Driver writes $f(\mathbf{X})$ and any errors to standard output (stdout).
   (d) Driver exits.
4. OSTRICH redirects stdout of driver to `OstExeOut.txt`.
5. OSTRICH reads $f(\mathbf{X})$ and any errors from `OstExeOut.txt`
6. OSTRICH proceeds to next step of algorithm.

Figure 7.1: OSTRICH and Driver Program Interface

function value and model error to stdout using the following syntax:

```
OST_ObjFuncVal      <value>
OST_ModelErrCode    <value>
```

The lines containing `OST_ObjFuncVal` and `OST_ModelErrCode` can occur anywhere in the output file (beginning, middle or end), so long as they are unique; otherwise, OSTRICH will parse the first occurrences of the required syntax. The value of `OST_ObjFuncVal` must be in integer, decimal (e.g. 68.345) or scientific (6.835E1) format and must not use a thousands separator (e.g. use 12345.678 instead of 12,345.678). Meanwhile, the `OST_ModelErrCode` value should be a single line of text which describes any errors that may have occurred. If there were no errors, the value of `OST_ModelErrCode` should be `no_errors`.

Chapter 8

Running OSTRICH

This chapter describes the execution of OSTRICH from the command line and parallel computing environments. A total of three OSTRICH executables are available: a serial version which 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:

1. `ostIn.txt` : Main configuration file, created using the syntax described in Chapter 4.
2. Template File(s) : File(s) that OSTRICH uses to create a syntactically correct model input file(s), so as to evaluate some set of model parameters ($\mathbf{X}$).
   (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.
3. 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.

8.1 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.

8.2 Parallel Execution

To run OSTRICH in parallel, or to have OSTRICH run a parallel version of the modeling program the user must have access to a parallel computing environment that supports the MPI (Message Passing Interface) libraries. After logging on to such a system the user can follow the general guidelines given below to run the desired parallel job.

1. Create working directory and OSTRICH component files as described previously.

2. Create script file for parallel job.

3. Submit parallel job to scheduler.

The details of creating and submitting parallel jobs depends on the configuration of the parallel cluster. Section 10.4 provides an example using PBS script files and the `qsub` command, run on the Clearwater cluster at the Center for Computational Research (CCR) at the University at Buffalo (UB).

8.2.1 Running OSTRICH in Parallel

The present version of OSTRICH contains a parallel implementation of the Genetic Algorithm. While any of the other algorithms can be successfully run in a parallel environment, doing so will not result in any performance improvement unless the underlying model is to be run in parallel (see Section 8.2.2). Future revisions of OSTRICH will provide parallel implementations of Particle Swarm Optimization and Simulated Annealing, along with parallelized computation of finite differences and regression statistics.

8.2.2 Running Model in Parallel

Running a parallel model from within OSTRICH is accomplished by having OSTRICH dynamically assemble and submit parallel runs of the model executable whenever a run of the model is needed. After submitting such a job, OSTRICH then uses job monitoring commands to wait for the parallel run to complete. To utilize this feature, the following requirements must be satisfied:

1. The model must be MPI-parallel; meaning that it must be parallelized using the MPI interface.
2. The model must be compiled for a Linux parallel cluster.
3. The cluster must support PBS (portable batch system) scripts.
4. The cluster must support the following commands:
    - `qsub` : To submit the parallel model run.
    - `qstat` : To monitor the job.
    - `sleep` : To pause OSTRICH while waiting for the parallel run to complete.

If all of these requirements are met, then running OSTRICH with a parallel model can be accomplished as follows: (a) select parallel model execution by include the `ParallelModelExec` line in the `ostIn.txt` input file, (b) follow the guidelines in 8.2 to submit a one (or two, if required by the cluster environment) processor OSTRICH job.

Chapter 9

Evaluating OSTRICH Output Files

Upon completion, OSTRICH will have generated four output files per processor `N`, where `N=0` for serial runs:

- `OstOutputN.txt` : Main output file, contains an optimization (or regression) record along with statistical output (if applicable).
- `OstErrorsN.txt` : Error file, any errors encountered by OSTRICH are stored in this file.
- `OstModelN.txt` : A sequential record of every model run is stored in this file.
- `OstExeOut.txt` : The output of Model runs are redirected to this file.

The next few sections describe these files in more detail using example output.

9.1 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 and objective function value. An example of elements (ii)-(iv) is shown in Figure 9.1. In Figure 9.1, the run record contains the parameter and objective function values at each iteration along with the value of an algorithm-dependent (e.g. λ) parameter.

```
Ostrich Setup
Model                  : Split.exe > OstExeOut.txt
Algorithm              : Levenberg-Marquardt
Objective Function     : WSSE
Number of Parameters   : 3
Number of Observations : 25

Ostrich Run Record
iter  obj. function  Kback          K2             K1             lambda
0     2.419552E+002  1.100000E+000  5.500000E+000  1.570000E+001  1.000000E+001
1     1.812830E-001  9.973903E-001  5.226647E+000  1.563847E+001  9.090909E+000
2     6.332663E-004  9.999858E-001  5.103630E+000  1.560503E+001  8.264463E+000
3     4.618933E-004  1.000013E+000  5.025882E+000  1.556041E+001  6.830135E+000

Optimal Parameter Set
Objective Function : 4.614216E-004
Kback              : 1.000008E+000
K2                 : 5.026354E+000
K1                 : 1.553661E+001
```

Figure 9.1: Sample Optimization Output

For population based algorithms (i.e. particle swarm optimization and the genetic algorithm), the parameter and objective function values for the current best solution are reported. Algorithm-dependent parameters reported in the run record are:

- Genetic Algorithm : The average objective function for the population.
- Particle Swarm Optimization : The average objective function for the population.
- Simulated Annealing : The temperature.
- Levenberg-Marquardt : The Marquardt λ.
- Powell : The change in objective function.
- Steepest Descent : The change in objective function.
- Fletcher-Reeves : The change in objective function.

9.2 Statistical Output

If regression is being performed, then various statistical measures may be reported, depending on whether or not they were selected in the input file (see Section 4.11). The following sub-sections describe this output of these statistics.

9.2.1 Observation Residuals

Observations residuals are reported automatically at the end of every regression. An example list of observation residuals is given in Figure 9.2. Also included in the observation residual output is the correlation between measured and simulated observations (R_y).

```
Observation Residuals
Observation    Measured       Simulated      Weighted
Residual
obs1           6.809246E+001  6.809238E+001  +7.555000E-005
obs2           6.570953E+001  6.570943E+001  +9.844000E-005
obs3           6.166319E+001  6.166321E+001  -1.707000E-005
obs4           6.807076E+001  6.807044E+001  +3.207400E-004
obs5           5.936214E+001  5.936215E+001  -7.620000E-006
obs6           6.449409E+001  6.449413E+001  -4.088000E-005

Correlation between measured and simulated observations
Ry : 0.923
```

Figure 9.2: Sample Observation Residuals

9.2.2 Error Variance and Standard Error of the Regression

If the user includes `StdDev` in the `MathAndStats` section of the input file, then the error variance (s^2) and standard error of the regression (s) will be reported. Figure 9.3 provides an example of resulting output file syntax.

9.2.3 Parameter Variance-Covariance

Including `StdErr` in the `MathAndStats` section of the input file will cause OSTRICH to output the parameter variance-covariance matrix along with the standard error of each parameter, as shown in Figure 9.4. Including

```
Error Variance and Standard Error of the Regression
S^2 : 2.097371E-005
S   : 4.579706E-003
```

Figure 9.3: Example of Basic Statistics

```
Parameter Variance-Covariance
_               Kback           K2              K1
Kback          +1.856038E-010  -1.428247E-008  +3.418409E-008
K2             -1.428247E-008  +5.830411E-006  -4.090841E-006
K1             +3.418409E-008  -4.090841E-006  +7.805281E-005

Parameter Standard Error
Kback           : 1.362365E-005
K2              : 2.414624E-003
K1              : 8.834750E-003
```

Figure 9.4: Sample Parameter Statistics

`CorrCoeff` in the input file will cause OSTRICH to output the parameter correlation matrix, as shown in Figure 9.5.

```
Parameter Correlation
_              Kback   K2      K1
Kback          +1.000  -0.434  +0.284
K2             -0.434  +1.000  -0.192
K1             +0.284  -0.192  +1.000
```

Figure 9.5: Sample Parameter Correlation

9.2.4 Confidence Intervals

If the user includes `Confidence` in the `MathAndStats` section of the input file, then OSTRICH will use the `CI_Pct` configuration variable to compute linear confidence intervals (CI) for each parameter along with the volume ratio (a comparison between CI block and ellipsoidal joint confidence regions). Figure 9.6 illustrates OSTRICH output for a 95% confidence interval.

```
Linear Confidence Intervals (95.00%)
Parameter      Lower Limit     Upper Limit
Kback          +9.999433E-001  +1.000073E+000
K2             +4.968730E+000  +5.084646E+000
K1             +1.489478E+001  +1.620610E+001
Volume Ratio : 0.861
```

Figure 9.6: Sample Confidence Interval Output

9.2.5 Model Linearity

Including either `Beale` or `Linssen` in the `MathAndStats` section of the input file will trigger computation and output the corresponding non-linearity measures. An example, where both `Beale` and `Linssen` have been included is shown in Figure 9.7.

```
Non-Linearity Measures
Beale (N)     : 1.056831E-004
Assessment    : Linear

Linssen (M^2) : 1.070575E-004
Assessment    : Linear

Thresholds for N and/or M^2
Non-linear : > 3.279620E-001
Linear     : < 2.951658E-002
```

Figure 9.7: Example of Non-Linearity Measures

9.2.6 Normality of Residuals

Inclusion of the `NormPlot` variable will cause OSTRICH to report a list of normalized residuals and the corresponding correlation coefficient (R_N^2), as illustrated in Figure 9.8.

9.2.7 Influential Observations

When either the `CooksD` or `DFBETAS` variable (or both) is set, OSTRICH will generate and output the corresponding measures of observation influ-

```
Normalized Residuals
r_expected      r_ordered
-1.964216E+000  -5.833990E-003
-1.519197E+000  -5.208840E-003
etc.
+1.519197E+000  +6.445870E-003
+1.964216E+000  +1.501312E-002
Normal probability correlation coefficient
R2N :  0.859
```

Figure 9.8: Example of Normalized Residuals Output

ence, along with an assessment of which observations are influential, based on influence thresholds suggested in the literature. Figure 9.9 contains an example of influential observation measures for a hypothetical 2-parameter problem.

9.2.8 Parameter Sensitivities

If the `Sensitivity` variable is found in the `MathAndStats` input file section, OSTRICH will report parameter sensitivity measures. An example of such output is shown in Figure 9.10.

9.2.9 Matrices

OSTRICH can be configured to output matrices used for various statistical calculations, using the `Matrices` variable. An example of such output is shown in Figure 9.11.

9.3 OSTRICH Error Messages

During execution, OSTRICH logs errors to the file `ostErrorsN.txt`, where N corresponds to the processor number. The most common error encountered is the violation of parameter boundaries; which occurs when the optimization algorithm (or statistical calculation) tries to set a parameter value that is greater than the upper bound or less than the lower bound of the given parameter. OSTRICH reports such errors as `PARAMETER BOUNDS` errors and continues running, but uses the upper (or lower) limit of the parameter instead of the requested value.

```
Measures of Observation Influence
Cook's D
Observation    Leverage   infl.?  Di         infl.?
obs1           8.72E-001  yes     9.15E+001  yes
obs2           8.02E-002  no      9.16E-005  no
etc.
Number of Influential Leverage : 2
Number of Influential Di       : 1
Thresholds for Cook's D
Di       > 6.67E-001
Leverage > 3.33E-001

DFBETAS
Observation    param_1     infl.?  param_2     infl.?
obs1           -6.24E-002  no      -2.06E-001  no
obs2           +2.06E-001  no      -1.30E-002  no
etc.
Number of Influential DFBETAS : 0
Threshold for DFBETAS
|DFBETASij| > 8.16E-001
```

Figure 9.9: Example of Influential Observation Measures

Parameter boundary violations can frustrate the optimization and even invalidate optimization results, especially if they occur during statistics calculations or during the execution of a numerical algorithm. Heuristic algorithms are more tolerant of boundary violations since internal checks within each algorithm prevent random perturbations from violating parameter boundaries.

The obvious technique for handling boundary violation errors is to adjust the parameter limits (e.g. reduce or increase by a factor of 10) and re-run the optimization. If a derivative-based scheme is being used (or if statistics are being calculated), note that adjusting parameter limits will affect the finite difference step size, and `DiffRelIncrement` may also need to be revised.

If OSTRICH encounters a singular matrix during Levenberg-Marquardt regression or during statistical calculation, the program will abort and a `SINGULAR MATRIX` error will be reported. Such an error could be an indication that some of the parameters in the calibration are linearly dependent or have a very high parameter correlation.

```
Parameter Sensitivities
Dimensionless Scaled Sensitivities
Observation    param_1          param_2
obs1           -1.44737E-002  -2.26310E-001
obs2           +7.82751E-002  -4.12052E-002
etc.
1-Percent Scaled Sensitivities
Observation    param_1          param_2
obs1            -1.44739E-004  -2.26319E-003
obs2            +7.82754E-004  -4.12052E-004
etc.
Composite Scaled Sensitivities
param_1  :  8.961414E-002
param_2  :  2.223486E-001
```

Figure 9.10: Example of Parameter Sensitivity Output

```
Matrices

Jacobian Matrix
Observation  param_1          param_2
obs1         -1.232016E-002  -3.236455E-001
obs2         +6.662821E-002  -5.892727E-002
etc.
Normal Matrix
+3.491185E-002  -2.196855E-002
-2.196855E-002  +6.066642E-001
Inverse Normal Matrix
2.931148E+001   1.061429E+000
1.061429E+000   1.686795E+000
```

Figure 9.11: Example of Matrices Output

If OSTRICH encounters an error when reading the input file or the model output file(s), a `FILE I/O ERROR` will be reported. Depending on the nature of the error, OSTRICH may be able to continue execution or may have to abort. For example, if OSTRICH is unable to locate the algorithm setup section of `ostIn.txt`, default values will be used. On the other hand, if

OSTRICH cannot parse a model output file, it will abort.

If OSTRICH encounters the end of a line of input unexpectedly, a `COULDN'T PARSE INPUT` error will be reported and the program will exit. These errors can occur if the observation, parameter or file pair lists in `ostIn.txt` are improperly formatted (e.g. a variable has been left out), or if the model output file containing observations is not formatted according the parsing instructions provided to OSTRICH.

If there are problems running a driver program (see Chapter 7) or running a parallel model (see Section 8.2), OSTRICH will report a `MODEL EXECUTION ERROR` along with a description of the error that occurred.

During simulated annealing, if OSTRICH is unable to perform the number of melt operations specified by the `NumInitialTrials` variable, then a `MAX NUMBER OF TRIALS` error will be reported and OSTRICH will begin accepting all melting trials, even if they decrease the objective function.

9.4 Redirected Model Output

OSTRICH redirects the standard error and standard output of all model runs to the file '`OstExeOut.txt`'. The output of each new model run will overwrite the contents of the file, such that `OstExeOut.txt` always contains the standard error and standard output of the most recent model run.

9.5 Model Run Record

OSTRICH maintains a file named '`OstModelN.txt`', where N is the processor number (0 for serial runs of OSTRICH), 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. Figure 9.12 shows and example `OstModel0.txt` file:

Run	obj.func.(WSSE)	K1	K2	Kback
1	8.230901E+004	5.000000E+001	5.000000E+001	5.000000E+001
2	8.230900E+004	5.002303E+001	5.000000E+001	5.000000E+001
etc.				

Figure 9.12: Example of *OstModel0.txt*

Chapter 10

Example Exercises

This chapter provides a set of examples illustrating the usage of OSTRICH in a variety of water resources and environmental engineering applications. Two examples deal with model calibration, one example illustrates a solution to a pump-and-tread optimization problem and the final example describes parallel processing applied to a calibration exercise that utilizes particle swarm optimization.

10.1 3-Parameter Groundwater Model Calibration

A hypothetical groundwater model was created using the analytical element method (AEM) based program known as Split. The Split executable and instruction manual are available for download from:

www.groundwater.buffalo.edu/software/software.html.

The model, a two-dimensional unconfined aquifer with no recharge and eastward regional uniform flow, contains three parameters that are to be calibrated: two-inhomogeneities (K1 and K2), and the background conductivity (Kback). Twenty-five synthetically generated head measurements are used as the observation data, with head values in the range of 65-70 meters. Figure 10.1 illustrates the setup of the model. The `Demo1` folder included with the OSTRICH distribution contains the required input and template files, along with the Split model executable. The `head.dat` file is a required Split input file which instructs Split to output simulated head values in a file named `headerr.dat`. The `Observations` section of the `ostIn.txt` input file contains the required parsing information for OSTRICH to extract these simulated heads.

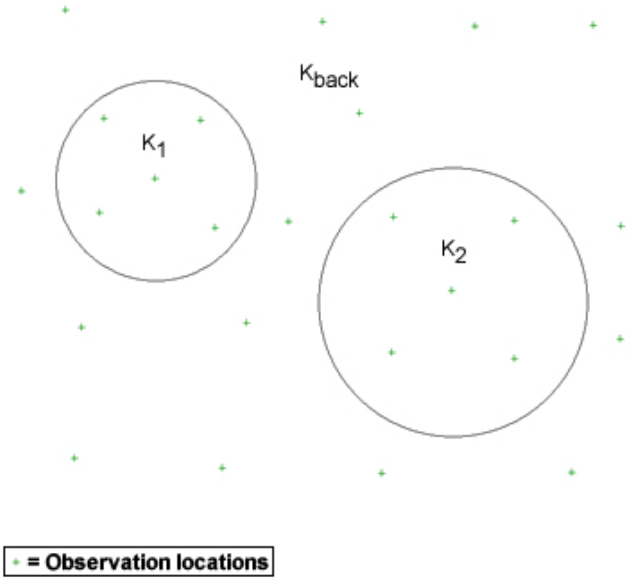

Figure 10.1: 3-Parameter Groundwater Model Setup

For this example, OSTRICH has been configured to use the Levenberg-Marquardt algorithm and since the parameters are conductivities, a log transformation has been applied within the `Params` section of the input file. Finally, the `AllStats` line in the `MathAndStats` section of the input file instructs OSTRICH to calculate and output the entire suite of regression statistics.

To run the calibration, open a command line prompt, change directory to the `Demo1` folder and execute OSTRICH from the command line. During execution, the OSTRICH run record displays the progress of the calibration and finally settles on the following parameter values: K1=15.6, K2=5.0 and Kback=1.0. The statistical output provides the following useful insights: i) as indicated by the Linssen and Beale measures, the model is linear in the vicinity of the calibrated parameter set, ii) the narrow confidence intervals indicate that the parameter values for this model have been estimated with a high degree of certainty, and iii) K2 and Kback exhibit a possibly significant negative correlation.

10.2 6-Parameter Diffusion Model Calibration

10.3 6-Parameter Pump-and-Treat Optimization

A hypothetical groundwater contamination problem was created and modeled using the Split modeling engine. The Split executable and instruction manual are available for download from:

www.groundwater.buffalo.edu/software/software.html.

The problem setup is illustrated in Figure 10.3; wherein plume capture is desired so as to prevent migration to a nearby surface water body. The site model is bounded by no-flow boundaries in the north and south and by constant head boundaries in the east and west, resulting in easterly regional flow. Also shown in Figure 10.3 are 11 hydraulic gradient control pairs and 50 particles, which are used as plume capture constraints in the OSTRICH PATO module. Also included are the initial locations of two pump-and-treat wells. The (x,y) coordinates and pumping rates (Q) of these wells are the PATO parameters to be optimized; for a total of 6 parameters (i.e. x_1,y_1,Q_1,x_2,y_2, and Q_2). Additional constraints for the pump-and-treat system are:

- The minimum head at each well is limited to 10 meters, corresponding to a roughly 20 meter maximum drawdown.
- Well rates are between ± 200 m^3/day
- Well (x,y) coordinates are between (0,0) and (335,243)
- Maximum total Q is 400 m^3/day
- Minimum total Q is 1 m^3/day (reflecting a desire to have more extraction than injection)

The pump-and-treat section of the PATO module is configured to use the C_{TOTQ} cost function, with α_{inj}=0.53 and α_{ext}=21.17, and the F_{APM} objective function. For this example, OSTRICH has been configured to use the Fletcher-Reeves algorithm, with a Golden-Section 1-D search and forward difference with 0.1% step size.

The `Demo2` folder included with the OSTRICH distribution contains the required input and template files, and the Split model executable is located in the `Demo1` folder. To run the PATO, open a command line prompt, change directory to the `Demo2` folder and execute OSTRICH from the command line.

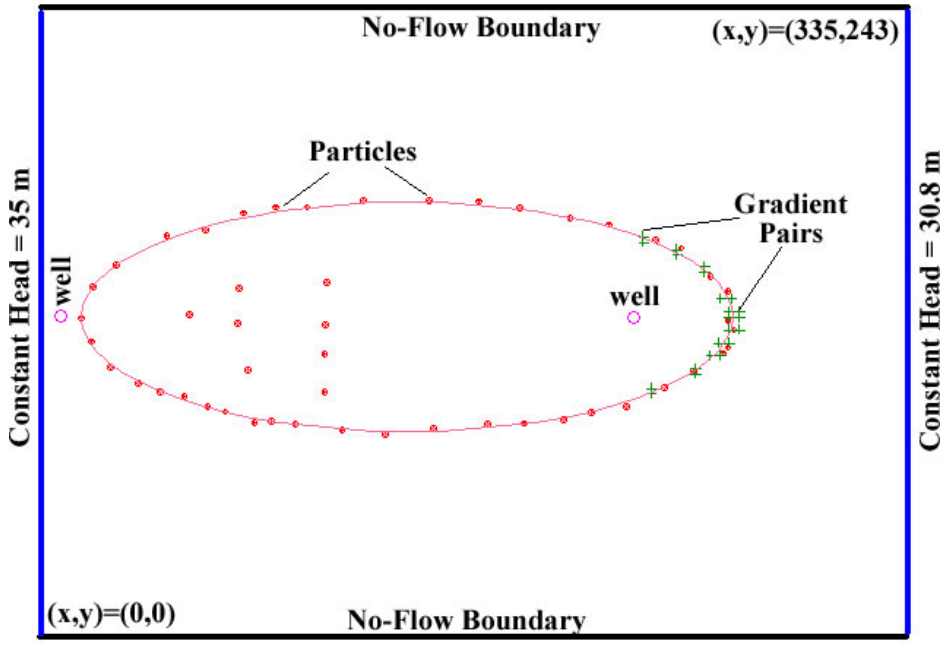

Figure 10.2: PATO Problem Setup

Due to the particle tracking operation, execution of the split model may be time consuming. During execution, the OSTRICH run record displays the progress of the optimization and finally settles on the following parameter values:

- x_1 : 0.00 m
- y_1 : 125 m
- Q_1 : -44 m^3/day (injection)
- x_2 : 230 m
- y_2 : 116 m
- Q_2 : 40 m^3/day (extraction)

The optimal cost found using the Fletcher-Reeves algorithm is \$875 with a penalty of \$44, incurred because there is a violation on the minimum pumping capacity (more injection than extraction). The corresponding total absolute pumping rate is 84 m^3/day. Note that the Fletcher-Reeves algorithm

is ineffective at varying the locations of the wells. User's are encouraged to try other algorithms to see how they perform in comparison.

10.4 Parallel Processing Example

Bibliography

Bates, D. M. and Watts, D. G. 1980. Relative curvature measures of non-linearity, *Journal of the Royal Statistical Society, Series B*, vol. 42, no. 1, pg. 1-25.

Beale, E. M. 1960. Confidence Regions in Non-linear Estimation, *Journal of the Royal Statistical Society, Series B*, vol. 22, no. 1, pg. 41-88.

Belsley, D., Kuh, E. and Welsch, R. 1980. *Regression Diagnostics: Identifying Influential Data and Sources of Colinearity*, John Wiley & Sons, Inc., New York, NY.

Beielstein, T., Parsopoulos, K. E., and Vrahatis, M. N. 2002. "Tuning PSO parameters through sensitivity analysis". Technical Report, Reihe Computational Intelligence CI 12402. Collaborative Research Center (Sonderforschungsbereich) Department of Computer Science, University of Dortmund.

Christensen, S. and Cooley, R. L. 1999. Evaluation of confidence intervals for a steady-state leaky aquifer model, *Advances in Water Resources*, vol. 22, no. 8, pg. 807-817.

Cook, R. and Weisberg, S. 1982. *Residuals and Influence in Regression*, Monographs of Statistics and Applied Probability, vol. 18, Chapman and Hall, New York, NY.

Cooley, R. L. and Naff, R. L. 1990. Regression modeling of ground-water flow. *U.S. Geological Survey Techniques of Water Resources Investigations*, Book 3, Chapter B4.

Diciccio, T. J. and Romano, J. P. 1988. A Review of Bootstrap Confidence Intervals, *Journal of the Royal Statistical Society, Series B*, vol. 50, no. 3, pg. 338-354.

Draper, N. and Smith, H. 1998. *Applied Regression Analysis: Third Edition*, John Wiley & Sons, Inc., New York, NY.

Efron, B. and Gong, G. 1983. A Leisurely look at the Bootstrap, the Jackknife, and Cross-Validation, *The American Statistician*, vol. 37, pg. 36-48.

Hill, M. 1998. Methods and guidelines for effective model calibration, *U.S. Geological Survey Water Resources Investigation Report 98-4005*, Denver, CO.

Kennedy, J. and Eberhart, R. C. 1995. Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ. pg. 1942-1948.

Levenberg, K. 1944. A Method for the Solution of Certain Problems in Least Squares, *Quarterly of Applied Mathematics*, vol. 2, pg. 164-168.

Linssen, H. N. 1975. Nonlinearity measures: A case study, *Statistica Neerlandica*, vol. 29, pg. 93-99.

Madsen, H. and Jacobsen, T. 2001. "Automatic calibration of the MIKE SHE integrated hydrological modelling system", 4th DHI Software Conference, 6-8 June, Scanticon Conference Centre, Helsingr, Denmark.

Marquardt, D. 1963. An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM Journal on Applied Mathematics*, vol. 11, pg. 431-441.

Mayer A. S., Kelley C. T., Miller C. T. 2002. Optimal design for problems involving flow and transport phenomena in subsurface systems, *Advances in Water Resources*, vol. 12, g. 1233-1256.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. 1995. *Numerical Recipes in C, Second Edition*, Cambridge University Press, New York, NY.

Rawlings, J. 1988. *Applied Regression Analysis*, Wadsworth and Brooks, Pacific Grove, CA.

RS Means ECHOS. 1999. *Environmental Remediation Cost Data Assemblies*, R.S. Means Company, Kingston, MA.

Sabo, D. 1999. "Normal Porbability Plots", British Columbia Institute of Technology (BCIT), BC, Canada.

Vanderbilt, D. and Louie S.G. 1984. A Monte Carlo Simulated Annealing Approach to Optimization over Continuous Variables, *Journal of Computational Physics*, vol. 56, pg. 259-271.

Vanderplaats, G. N. 2001. *Numerical Optimization Techniques for Engineering Design*, Vanderplaats Research & Development, Colorado Springs, CO.

Vecchia, A. V. and Cooley, R. L. 1987. Simultaneous Confidence and Prediction Intervals for Nonlinear Regression Models with Application to a Groundwater Flow Model, *Water Resources Research*, vol. 23, no. 7, pg. 1237-1250.

Yager, R. 1998. Detecting influential observations in nonlinear regression modeling of groundwater flow, *Water Resources Research*, vol. 34, no. 7, pg. 1623-1633.