ANALYTIC ELEMENT MODELING OF THE HIGH PLAINS AQUIFER: NON-LINEAR MODEL OPTIMIZATION USING LEVENBERG-MARQUARDT AND PARTICLE SWARM ALGORITHMS

by

ANDY ALLEN

B.S., Kansas State University, 2009

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Civil Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2012

Approved by:

Major Professor
David R. Steward

Copyright

Abstract

Accurate modeling of the High Plains Aquifer depends on the availability of good data that represents and quantifies properties and processes occurring within the aquifer. Thanks to many previous studies there is a wealth of good data available for the High Plains Aquifer but one key component, groundwater-surface water interaction locations and rates, is generally missing. Without these values accurate modeling of the High Plains Aquifer is very difficult to achieve. This thesis presents methods for simplifying the modeling of the High Plains Aquifer using a sloping base method and then applying mathematical optimization techniques to locate and quantify points of groundwater-surface water interaction. The High Plains Aquifer has a base that slopes gently from west to east and is approximated using a one-dimensional stepping base model. The model was run under steady-state predevelopment conditions using readily available GIS data representing aquifer properties such as hydraulic conductivity, bedrock elevation, recharge, and the predevelopment water level. The Levenberg-Marquardt and particle swarm algorithms were implemented to minimize error in the model. The algorithms reduced model error by finding locations in the aquifer of potential groundwater-surface water interaction and then determining the rate of groundwater to surface water exchange at those points that allowed for the best match between the measured predevelopment water level and the simulated water level. Results from the model indicate that groundwater-surface water interaction plays an important role in the overall water balance in the High Plains Aquifer. Findings from the model show strong groundwater-surface water interaction occurring in the northern basin of the aquifer where the water table is relatively shallow and there are many surface water features. In the central and southern basins the interaction is primarily limited to river valleys. Most rivers have baseflow that is a net sink from groundwater.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I wish to express gratitude to Prof. David Steward for his continuous guidance throughout my studies and in this research project. I would like to gratefully acknowledge financial support provided by the National Science Foundation (grants EPS0553722 and GEO0909515) and the United States Department of Agriculture/Agriculture Research Service (Ogallala Aquifer Initiative). I would also like to thank my committee members Prof. Stephen Welch and Prof. Saugata Datta for providing supplemental advice and accepting to review my thesis and supervise my Oral Examination.

Chapter 1

Introduction

1.1 Overview

Groundwater is a critically important water resource. Of all the fresh water on Earth, approximately 30% is stored in aquifers while less than 1% is stored in surface water bodies such as lakes and rivers. The remaining 69-70% is frozen in glaciers and the polar ice caps [Hornberger, 1998]. Groundwater provides a valuable sorce of drinking and irrigation water for humans and it also provides water for important natual environments. Many rivers and wetlands depend on groundwater to supply the water necessary to sustain their aquatic ecosystems.

Groundwater flow and storage are impacted by both human and climatic stresses. In the absence of human stresses, large aquifers like the High Plains Aquifer shown in blue in Figure 1.1, will usually achieve a state of equilibrium where discharge from the aquifer equals recharge into the aquifer. This steady-state is disturbed when groundwater wells are installed and discharge from the aquifer begins to exceed recharge.

This study examines the groundwater flow through the High Plains Aquifer under steady state conditions. The High Plains Aquifer is a nationally important water resource that underlies an area of nearly 174,000 mi^2 (450,658 km^2) covering parts of eight states in the Midwestern United States. The aquifer is relied upon heavily to support large agricultural production with more than one-fourth of the Nation's agricultural production coming from

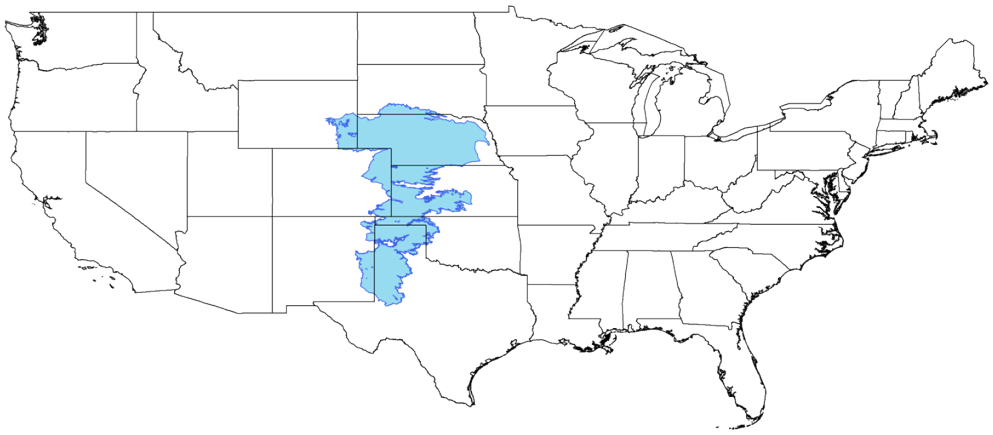

Figure 1.1: *Location of High Plains Aquifer*

this region [McMahon et al., 2007]. This land is primarily irrigated from the High Plains Aquifer which supplies nearly 30% of the nations total groundwater used for irrigation. In addition to irrigation, the aquifer also provides drinking water for 82% of the people who live within the aquifer boundary [Dennehy, 2000]. There are also several major river systems that cross the High Plains from west to east. These river systems include the Platte, Republican, Arkansas, Cimarron, and Canadian Rivers (figure 1.2). These major rivers as well as many smaller streams are hydraulically connected to the High Plains Aquifer [Dennehy, 2000]. During low flow periods, water in these rivers may be almost entirely derived from groundwater discharge from the High Plains Aquifer.

Given the importance of the High Plains Aquifer it is essential that it is well managed. In order to support proper management a good understanding of inputs (recharge) and outputs (pumping and natural discharge) within the aquifer are necessary. Good knowledge on the aquifer's physical properties (hydraulic conductivity, bedrock elevation, specific yield) is also vital. Data is available for the aquifer properties such as hydraulic conductivity, bedrock elevation, and recharge. However, little data is available that pinpoints and quantifies areas of natural groundwater discharge or areas where recharge exceeds published estimates due to surface water bodies transmitting large volumes of runoff water down to the water table.

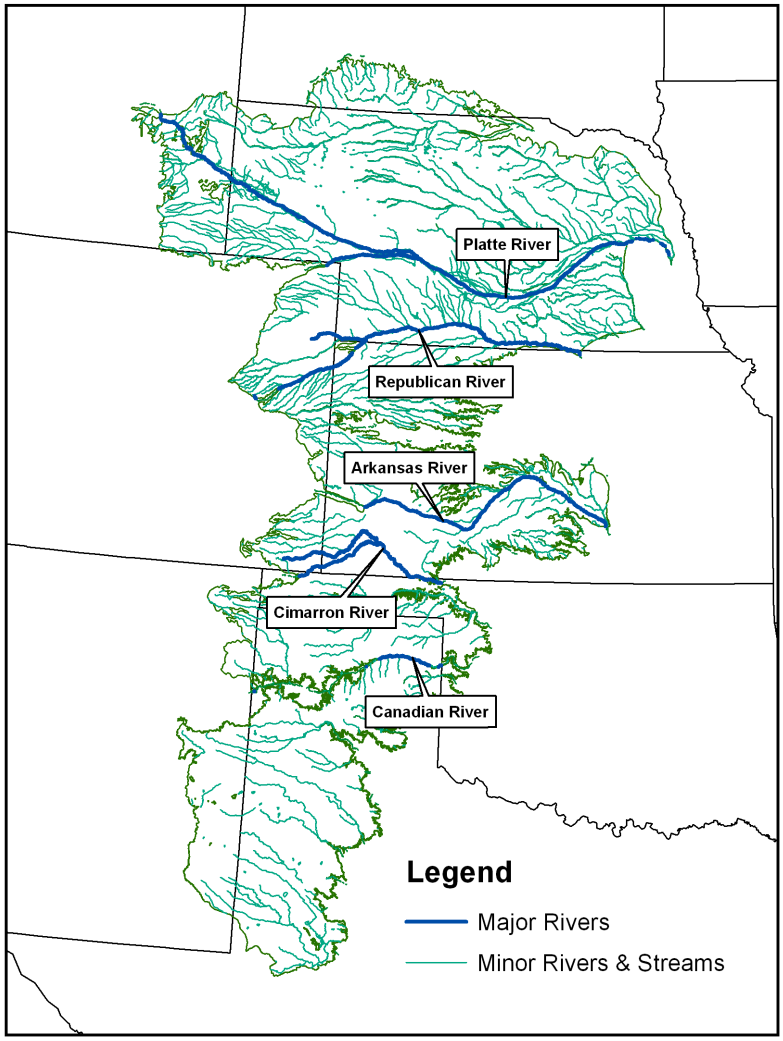

Figure 1.2: *Major Rivers of the High Plains Aquifer*

These groundwater-surface water interactions are important to better understanding the water balance in the High Plains Aquifer.

This study aims to improve our understanding of groundwater flow, especially in areas of groundwater-surface water interaction in the High Plains Aquifer during predevelopment conditions (pre-1940's). This is achieved through the use of a one-dimensional groundwater model which uses a stepping base approach to approximate flow through the aquifer. Predevelopment groundwater heads are simulated and conservation of mass is achieved by balanc-

ing groundwater recharge and discharge. Locations and quantities of groundwater-surface water interaction are determined through the use of the numerical optimization techniques known as the Levenberg-Marquardt algorithm and Particle Swarm Optimization.

Chapter 2

Background

2.1 High Plains Aquifer

The High Plains Aquifer (Figure 1.1) is located in the central part of the United Sates and covers 174,000 square miles in parts of Kansas, South Dakota, Wyoming, Nebraska, Colorado, New Mexico, Oklahoma, and Texas. It is the principal source of water in one of the Nation's major agricultural areas. It provides drinking water for 82 percent of the people living within its boundaries, and one third of the nation's water used for irrigation is pumped from the High Plains aquifer.

The High Plains aquifer is made up mostly of hydraulically connected geologic units. The Ogallala Formation is the largest unit in the High Plains aquifer covering 134,000mi^2 (347,000 km^2). The Ogallala consists of a mixture of clays, silts, sands, and gravel deposits that were laid down by streams that flowed eastward from the mountains. On the regional scale, the High Plains aquifer is an unconfined aquifer consisting primarily of near surface sand and gravel deposits. About 66% of the water storage is in Nebraska and about 12% is in Texas [Gutentag et al., 1984]. The saturated thickness (shown in Figure 2.1) averages about 200 ft with a maximum of nearly 1,250 ft in parts of Nebraska. Hydraulic conductivity depends on sediment types which are highly variable throughout the aquifer. Consequently, hydraulic conductivity is also highly variable. Hydraulic conductivity ranges from 25 to 300 feet per day with an average of 60 ft/day (18.3 m/day). Groundwater flow is generally east

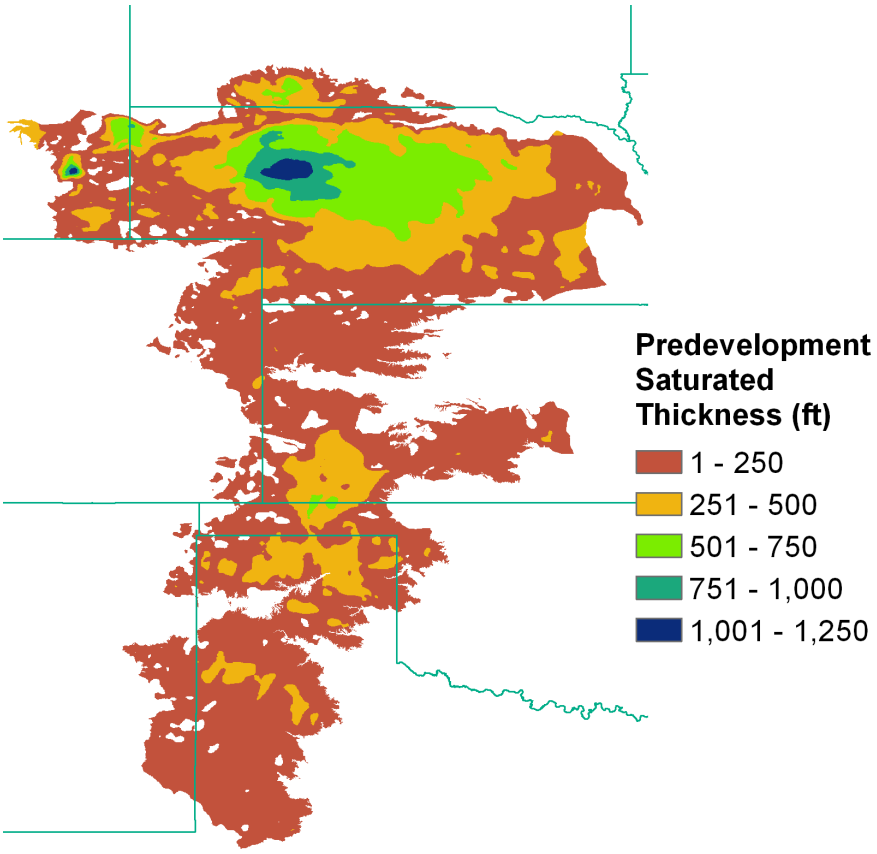

Figure 2.1: *High Plains Predevelopment Saturated Thickness*

to west with with an average rate of about 1 foot per day. This flow discharges naturally into springs, streams, and directly into the atmosphere by evapotranspiration. The main source of recharge is precipitation. Prior to development of the aquifer which began in the 1940's and 1950's the water table was near equilibrium and the quantity of water stored in the aquifer was nearly constant with changes occurring only in response to variations in annual precipitation, streamflow, and vegetation.

2.2 Recharge and Discharge

Under steady-state conditions the High Plains' groundwater storage mass balance is represented simply by change in $Storage = Recharge - Discharge$. Under predevelopment

conditions groundwater in the aquifer would flow from areas of recharge to areas of discharge. Discharge from the aquifer would occur in multiple ways such as flow to streams, lakes, and springs; water usage by phreatophytes; evaporation from playas and areas of very shallow water table; and leakage to adjacent aquifers [Sophocleous, 2005].

Many studies have been conducted to determine rates of recharge in the High Plains aquifer. One major method that has been used to quantify recharge is the regional climatic soil-water balance method. The regional climatic soil-water balance method has been used by the USGS in multiple studies. JT Dugan [1985] used this method to determine recharge rates for the Central Midwest Regional Aquifer Systems Analysis (CMRASA) in parts of Arkansas, Colorado, Kansas, Missouri, Nebraska, New Mexico, South Dakota, and Texas but excluded the land areas overlaying the High Plains aquifer. Hansen [1991] used the methods established by Dugan and Peckenbaugh to calculate potential natural recharge for the entire state of Kansas. The potential natural recharge was defined as the amount of water that is deep enough in the soil profile to be beyond the influence of evapotranspiration and therefore available to move down to the water table. The soil water balance used is represented by the following equation:

$$R = ASW + P - SRO - AET - SC \tag{2.1}$$

where:

R	=	potential recharge
ASW	=	antecedent soil water within the root zone
P	=	precipitation
SRO	=	surface runoff
AET	=	actual evapotranspiration
SC	=	total available soil water storage capacity of the root zone

Dugan and Zelt [2000] expanded the CMRASA study and Hansen [1991] to include the entire High Plains aquifer region. All data used in their study (precipitation, temperature, solar radiation, etc...) to calculate potential recharge was was taken from the years 1951-1980 and averaged. In addition to calculating potential recharge rates based on equation

(2.1) they also determined potential recharge rates including the impacts of irrigation on recharge. The soil water balance equation used for those calculations is as follows:

$$R = ASW + P + I - SRO - AET - SC \tag{2.2}$$

where I = irrigation and all other terms are the same as shown in (2.1).

2.3 Groundwater-Surface Water Interaction

Groundwater-surface water interaction is a key component in understanding the mass balance of the High Plains aquifer. According to Sophocleous [2005] in order to properly manage groundwater resources it is critical to have accurate information about the inputs (recharge) and outputs (discharge) within a groundwater basin. Without good estimates of recharge groundwater models become unreliable.

Changes in the water table will have an impact on surface water in areas where the two are connected. Decline of groundwater levels near surface water bodies can capture some groundwater that would have otherwise discharged as baseflow to the surface water. As the water table drops, baseflow from the groundwater may stop completely and the surface water will begin to flow into the aquifer. Sophocleous [2002] calls this process *induced infiltration* or *induced recharge*. The result of reduced baseflow and induced infiltration is streamflow depletion.

The studies referenced in section 2.2 use average rainfall over a large geographic location to calculate average recharge rates but other factors could have large effects on recharge to the High Plains aquifer. Transmission losses from rivers during high precipitation events can contribute greatly to the recharge of the aquifer. Gillespie and Slagle [1972] estimated the recharge from Wet Walnut Creek near located in west central Kansas and found that recharge to the aquifer during periods of low flow was only around 0.5 cubic feet per mile of stream channel. However, they also found that recharge from high flows can be substantial. They noted water-level rises of 4 to 14 feet in observation wells following periods of high flow.

In a similar project Jordan [1977] studied transmission losses of flood flows in ephemeral channels in western Kansas. He estimated the average transmission loss to be 1.3 percent of flow per mile of stream channel. Although ephemeral streams are intermittent, it is likely, based on the above research, that high flows in such streams can be a principal source of recharge to the aquifer.

One area of intense groundwater-surface water interaction in the High Plains is the Sand Hills of Nebraska shown in figure 2.2. The Sand Hills is a region of mixed-grass prairie on grass-stabilized sand dunes that covers just over one quarter of the state. It is the largest sand dune formation in the Western Hemisphere and contains thousands of ponds which recharge the High Plains aquifer [Szilagyi et al., 2011]. The aquifer in turn discharges into creeks and rivers throughout the region. According to Sniegocki [1959], the water table is so close to land surface in many areas in the Sand Hills that groundwater discharge by evapotranspiration may equal or exceed that of baseflow discharging to streams. This region is an example of how important groundwater-surface water interaction can be in the High Plains.

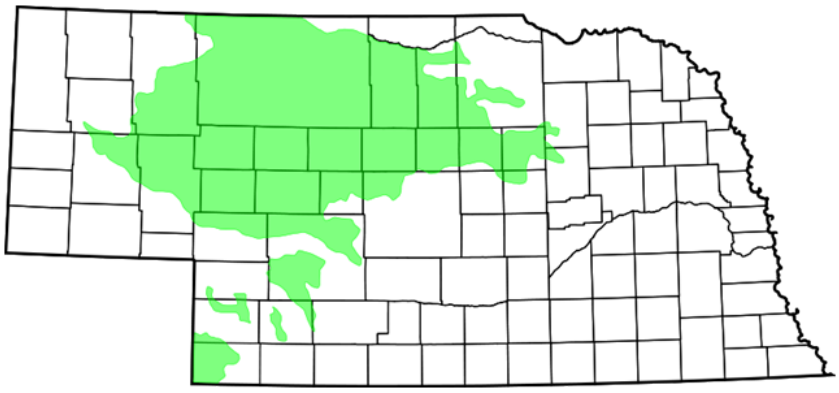

Figure 2.2: *Nebraska Sand Hills*

2.4 Groundwater Modeling

2.4.1 Modeling Methods

Groundwater models are used to represent the flow of groundwater in an aquifer. These models can be used to simulate and predict aquifer conditions under different scenarios. Two popular methods of mathematical modeling are the finite difference method (FDM) and the finite element method (FEM). The FDM method has been used extensively by the groundwater community and is the basis of the popular groundwater modeling software MODFLOW. Both methods depend of the use of a grid to represent the area being modeled. The finite difference method is limited to rectangular meshes while the finite element method can use triangular or deformed rectangular meshes.

The analytic element method (AEM) is another form of mathematical modeling used to model groundwater flow that was initially developed by O.D.L. Strack at the University of Minnesota [Strack, 2003]. The governing differential equation in the AEM method is linear and this allows many elementary solutions (elements) to be superimposed to obtain increasingly complex solutions. Each analytic element function is designed to simulate the effect of a discrete physical feature in the aquifer such as a well, surface water body, or a change in conductivity. Unlike FDM and FEM, AEM is grid independent and the flow solutions are continuous over the model domain. This allows features to be represented by their exact coordinates rather than being approximated or aggregated within a grid cell. Another benefit of AEM is that the computational effort required depends on the number of elements being modeled and their discretization level, not the spatial extent of the model domain. This allows for the modeling of the main features of a very large geographic area at high resolution without excessive computation time.

The AEM has been applied to many different types of groundwater problems. Large multiaquifer problems have been solved through the use of AEM. Bakker et al. [1999] applied the AEM method to the Yucca Mountain project. The model simulated an area larger than 450,000 km^2 demonstrating the effectiveness of AEM for large scale models. AEM

has also been used to simulate groundwater flow over most of the Netherlands in a model known as NAGROM. Meij and Minnema [1999] used the country-sized multiaquifer model to investigate the effects of sea-level rise, salt water intrusion, and land subsidence in individual provinces.

Groundwater-surface water interaction has been modeled extensively through the use of AEM. Hunt et al. [2003] developed a two-dimensional model to simulate the groundwater flow systems in La Crosse County, Wisconsin. The models provided estimates of the locations and amounts of groundwater flow into the Mississippi River as well as other local streams and lakes. Feinstein et al. [2005] used GFLOW (AEM based software) to develop a model of the St. Croix River Basin in Wisconsin to better understand the groundwater flow system and its relation to stream drainage in the St. Croix.

AEM has also been used to evaluate different well types and wellhead protection. Steward and Jin [2001] created a three-dimensional analytic model of an isolated horizontal well. The model was used to investigate and quantify losing sections of the horizontal well. The concern was that contaminated water could be captured by a gaining section of the well and then be injected into and uncontaminated portion of the aquifer in a losing section of the well. Drinking water source protection areas were modeled using AEM by Raymond et al. [2006]. GFLOW2000 was used to delineate capture zones in areas where other more widely accepted methods had already been used and demonstrated that GFLOW2000 was capable of producing capture zones similar to the other methods. GFLOW2000 was also used in place of other simpler volumetric methods used by some states and was found to produce much more accurate and detailed capture zones than the more simplistic delineation methods.

2.4.2 Sloping Base Analysis

The approximate analysis of flow of groundwater over an impermeable bed was initiated by Dupuit [1863] and then expanded upon by Boussinesq [1904]. This subject has received

much attention in recent years, especially in regards to the situation in which the bed is sloping. This sloping base scenario is well represented by the High Plains aquifer which has a base that slopes gently from west to east. Sloping base analysis assumes that the lateral extent of the aquifer is very large compared with its thickness and therefore the flow is constrained to directions that are parallel to the bed [Childs, 1971]. Regions of rapidly changing hydraulic gradient cannot be included in the sloping base approximation.

The impact of transients in a sloping base aquifer was examined by Steward et al. [2009]. Transient response functions were developed through the use of dimensionless variables and regression of model results. These transient response functions were used to reconstruct groundwater response to historical water use practices and to predict future changes in saturated thickness for several possible alternative water use scenarios.

Solutions for steady and transient flow using physical and numerical models for recharge-induced groundwater flow over a sloping bed was studied by Chapman [2005]. Outflow hydrographs for groundwater under conditions of steady uniform recharge were studied in viscous fluid model tests. Data from the experiments were compared with the nonlinear Boussinesq model and it was shown that for a given bed slope there is a nearly linear relationship between outflow and storage raised to a power n, where n ranged from almost 2 for zero slopes to slightly more than 1 at a gradient of 0.3.

Daly and Porporato [2004] used the sloping base concept along with the non-linear Boussinesq equation to model groundwater flow along a hillslope. Through the use of a traveling wave coordinate transformation he was able to write the Boussinesq equation as if the flow occurred on a horizontal impermeable bed. Although the coordinate transformation allowed for simplification of the problem, it also limits his approaches applicability because the same transformation has to be applied to the initial and boundary conditions.

Steward [2007] examined the transient response of groundwater storage induced by extraction or injection of water over a small region. In formulating equations for a sloping model, the slope of the base was incorporated directly into the hydraulic conductivity and

aquifer diffusivity terms which allowed for consistency between formulations using $s - n$ coordinates (s being tangent to the base) and $x - z$ coordinates (x being horizontal). A stepping base model based on the Analytic Element Method was also developed for both steady and transient flow. Results from the stepping model were compared with the sloping model and it was found that the stepping model approaches the exact solutions of the sloping model as the number of steps increased.

2.4.3 Model Calibration

Effective management of groundwater resources requires accurate estimation of aquifer parameters which are key inputs to groundwater flow models. The process of discerning parameters from field data (calibration) is critical to the modeling process. Model calibration can be done manually or automatically. Manual calibration may be acceptable when the model is quite simple but most models tend to have a level of complexity that makes manual calibration a very undesirable method. For example, it took 500 man hours and 80 simulation runs to calibrate transmissivity at 222 nodes in a finite element model for the Cortaro Aquifer in southern Arizona using a manual trial and error technique [Power, 1993]. So clearly automated calibration techniques are necessary for complex models. There are a number of different techniques available to perform automated calibration but two of particular interest to this study are particle swarm optimization and the Levenberg-Marquardt algorithm.

Levenberg-Marquardt

The Levenberg-Marquardt algorithm is a gradient based method and is one of the most popular methods for model calibration [Piotrowski, 2011]. Levenberg-Marquardt operates by starting from a given point in a parameter space and then moving down gradient until it converges to a minimum.

Power [1993] applied the Levenberg-Marquardt nonlinear leas-squares algorithm to an AEM based steady-state regional aquifer model of Bemidji, Minnesota. In his study Power

found the analytic element method to be very stable and attractive for use in automated calibration routines. The suitability of AEM to automated calibration was related to the fact the analytic element method uses analytic functions to describe hydraulic features. This approach focuses the attention on the critical geologic and hydraulic features of the aquifer rather than on the discretization of the aquifer into a grid. His study also demonstrated the effectiveness of the automated calibration routine at making groundwater modeling feasible for a wider variety of projects where limited budgets and lack of tools would otherwise make such modeling inappropriate.

The Levenberg-Marquardt algorithm was also used by Kambhammettu [2010] to estimate the transmissivity and storage coefficient of a confined aquifer. Kambhammettu demonstrated one of the strengths of the Levenberg-Marquardt method being that it can converge very rapidly to the minimum in comparison with other methods. He also showed that this rapid convergence depends quite strongly on a good initial guess for the starting point and that without a good initial guess convergence is much slower.

Particle Swarm Optimization

While gradient based algorithms like Levenberg-Marquardt are quite popular they have a weakness in that they can produce local optimal values rather than the global optimal solution. One way to overcome this is to use the multi-start technique where the Levenberg-Marquardt optimization routine is run multiple times starting from a different point in the parameter space each time. This becomes inefficient though as the parameter space increases in size and dimension. To overcome this problem global search methods like particle swarm optimization can be used. Particle swarm optimization operates by having multiple "particles" moving around the parameter space with each particle representing a potential solution to the optimization problem. Having multiple particles searching the parameter space at the same time helps to prevent convergence to a local minimum.

Particle swarm optimization was used by Gaur [2011] to assist in solving groundwater management problems. An AEM model was coupled with the particle swarm technique to

determine the maximum pumping from an aquifer in the Dore river basin in France and also to determine the minimum cost to develop a new pumping well system. The AEM-particle swarm model was found to be efficient in identifying the optimal location and discharge of the pumping wells. The study also used a penalty function approach which is used to help drive the particles away from undesirable solutions.

Chau [2007] used split step particle swarm optimization to train an artificial neural network used for real-time forecasting of water levels at in the Shing Mun River near Hong Kong. In Chau's split step approach particle swarm optimization was used for a predetermined number of iterations before switching over to the Levenberg Marquardt algorithm. With this approach Chau was able to combine the advantages of the global search capability of the particle swarm algorithm with the fast local convergence of the Levenberg-Marquardt algorithm. Results showed that this approach was able to achieve higher accuracy in shorter time periods than just using particle swarm optimization alone.

Piotrowski [2011] also applied particle swarm optimization to the problem of river stage forecasting. Piotrowski compared the usage of particle swarm optimization to train artificial neural networks versus the use of the Levenberg-Marquardt algorithm for training. His conclusion was that the Levenberg-Marquardt method was best because of its shorter convergence time but acknowledged that Levenberg-Marquardt had problems with converging to local minima. His suggested solution to this was simply to use the multi-start approach. This, however, overlooks the limitations of the multi-start approach discussed earlier in this section.

Chapter 3

Motivation

Modeling of groundwater is often accomplished through the use of the software package MODFLOW. For this study, a steady-state model of the High Plains Aquifer was created using MODFLOW with the standard data sets obtained from the the USGS (see Appendix C). The only data that was not available from previous studies was river conductance where river conductance is a measure of how much water a river bed can transmit to or from an aquifer.

$$C = \frac{A}{r} \tag{3.1}$$

where:

C = conductance ($[L]^2/[T]$)
A = riverbed area ($[L]^2$)
r = resistance ($1/[T]$)

and

$$r = \frac{t}{K} \tag{3.2}$$

where:

t = riverbed thickness ($[L]$)
K = riverbed conductivity ($[L]^2/[T]$)

Resistances of 0.1/day, 1.0/day, 10.0/day, and 100.0/day were used to test different

river conductance values for the High Plains Aquifer. The resulting error of these model runs are shown in figures 3.1-3.4. Predevelpment groundwater contours were obtained from Ceaderstrand and Becker [Cederstrand and Becker, 1999] which were then turned into a raster using the Topo to Raster tool in ArcGIS. The MODFLOW model's output head values were used to create rasters of the simulated head. The errors are presented as the predevelopment head minus the simulated head.

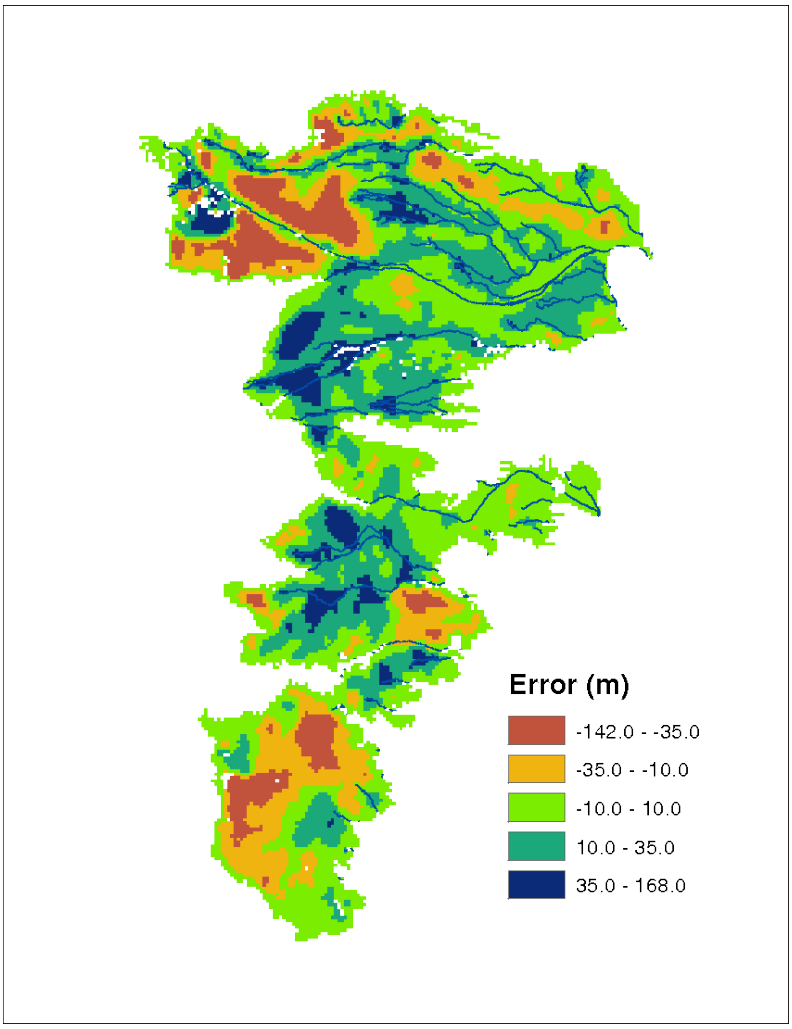

Figure 3.1: *MODFLOW Results with River Resistance = 0.1/day*

These results show that the rivers have a large impact on the model which indicates that groundwater-surface water interaction is an integral part of the water balance in the

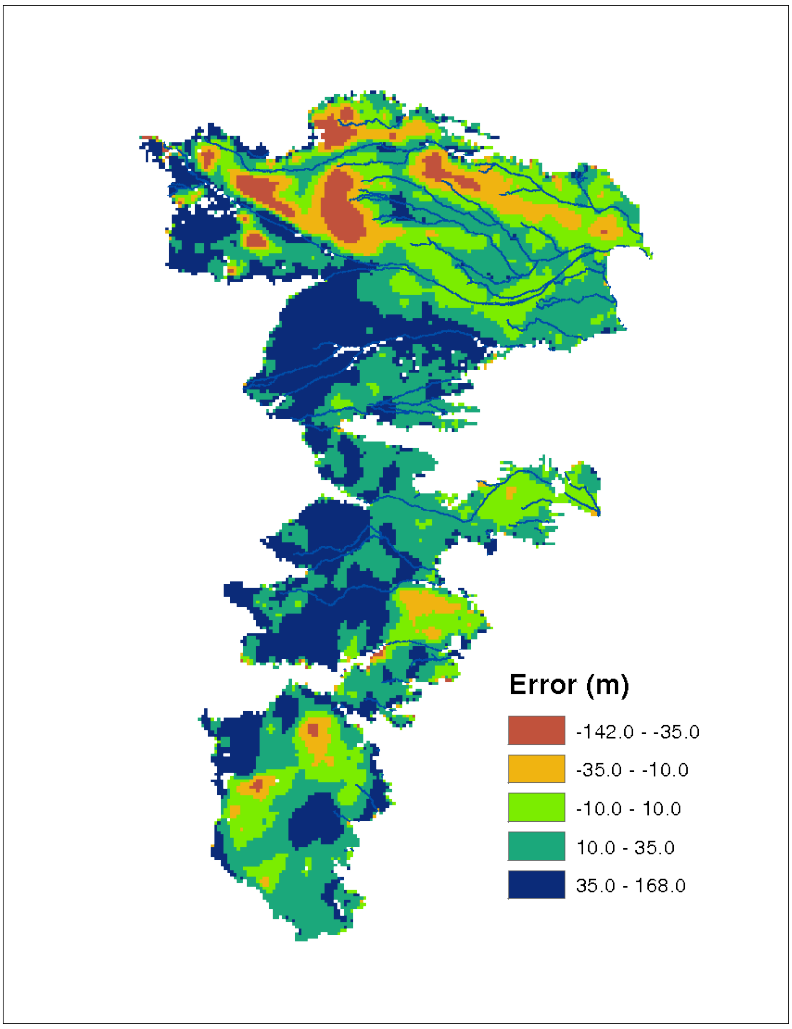

Figure 3.2: *MODFLOW Results with River Resistance = 1.0/day*

High Plains Aquifer. The results also indicate that a good model can not be produced with such simple representations of the groundwater-surface water interaction. Given the size and heterogeneity of the High Plains Aquifer it is probable that rates of interaction throughout the aquifer will vary. Areas with large conductivities and shallow water tables will have higher rates of interaction versus areas with low conductivities and deep water tables. It is also probable that points of interaction will occur in areas other than just river beds. Areas where the water table is shallow will likely have springs, ponds, lakes, and wetland areas that will provide many points of groundwater-surface water interaction. The

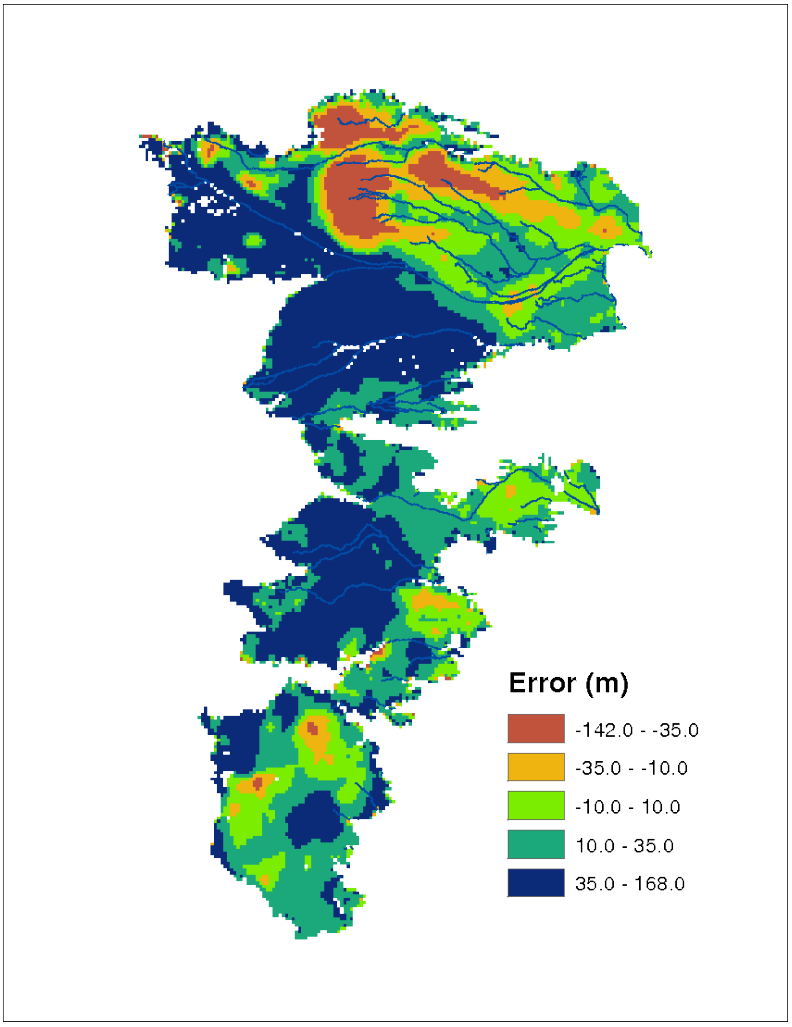

Figure 3.3: *MODFLOW Results with River Resistance = 10.0/day*

variability in groundwater-surface water interaction in the High Plains Aquifer makes large scale modeling of the aquifer very difficult using the current standard modeling techniques. When trying to represent all the groundwater-surface water interaction in a MODFLOW model it will be necessary to know the geographic locations of every single surface water feature that is interacting with the aquifer before the model is run.

Also, representation of surface water features will require that many different parameters be known. For rivers values for riverbed conductances or river stages will be required. Lakes will require data for the lakebed hydraulic properties as well as maximum and minimum lake

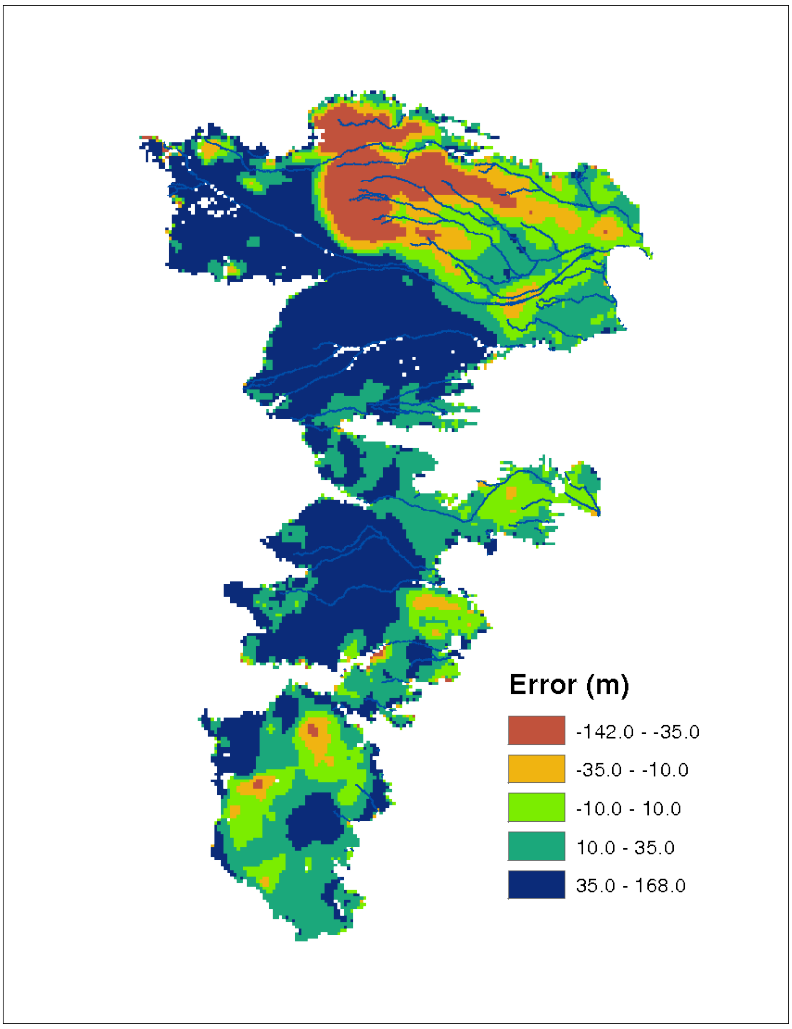

Figure 3.4: *MODFLOW Results with River Resistance = 100.0/day*

stages. These requirements make determination of groundwater-surface water interaction in the High Plains Aquifer using a modeling environment such as MODFLOW very complex and difficult to achieve.

These difficulties in determining groundwater-surface water interaction show the need for a modeling technique that reduces model complexity while accurately representing groundwater flow in the High Plains Aquifer and allowing for the location and quantification of points of groundwater-surface water interaction. The lack of good data on groundwater-surface water interaction in the High Plains Aquifer and the inability to locate and quantify

these points of interaction through the use of standard modeling techniques are the problems to be addressed by this study.

Chapter 4

Methods

4.1 Stepping Base Model

4.1.1 Groundwater Flow Equations

Our understanding of groundwater flow mechanics is based on a constitutive relation discovered by Darcy [1856] known as Darcy's law. Through experimentation, Darcy was able to establish an empirical relationship that described the flow of fluids through a porous medium by relating the specific discharge (q_i) to the head (ϕ).

$$q_x = -k\frac{\partial \phi}{\partial x} \tag{4.1}$$

$$q_y = -k\frac{\partial \phi}{\partial y} \tag{4.2}$$

$$q_z = -k\frac{\partial \phi}{\partial z} \tag{4.3}$$

In 1863 Jules Dupuit established the use of the assumption (known as the Dupuit assumption) that groundwater head does not vary in the z direction meaning all flow is horizontal.

$$\frac{\partial \phi}{\partial z} = 0 \tag{4.4}$$

This assumption allows for simplification of the groundwater flow equations for both confined and unconfined aquifers. For this study, the High Plains Aquifer was assumed to be unconfined and one-dimensional.

A potential function is developed here to simplify later mathematics. The discharge per unit width (Q_x) is arrived at by performing the following integration:

$$Q_x = \int_0^h q_x dz \tag{4.5}$$

where h is the saturated thickness of the aquifer. If the head is measured from the base of the aquifer then

$$\phi = h \tag{4.6}$$

and since our model makes use of the Dupuit assumption, q_x does not vary over the the height of the aquifer and so (4.5) reduces to

$$Q_x = \phi q_x \tag{4.7}$$

and substituting Darcy's law gives

$$Q_x = \phi \left[-k \frac{\partial \phi}{\partial x} \right] \tag{4.8}$$

To help the derivation, a new function known as the discharge potential ($\Phi(x, y)$) is introduced and (4.8) is rewritten as

$$Q_x = -\frac{\partial \Phi}{\partial x} \tag{4.9}$$

where $\Phi(x, y)$ is defined as

$$\Phi = \frac{1}{2} k \phi^2 \qquad (\phi < h) \tag{4.10}$$

for an unconfined aquifer so that (4.7) and (4.8) are identical. Equation (4.10) is used to calculate the head from a known potential (Φ) and is rewritten as

$$\phi = \sqrt{\frac{2\Phi}{k}} \tag{4.11}$$

A conservation of mass equation is obtained using the inflows and outflows through an elementary aquifer volume are shown in figure 4.1. The volume is a column of soil and water representing the full saturated aquifer height and the recharge (R) is assumed to be constant over time. From continuity of flow we know that

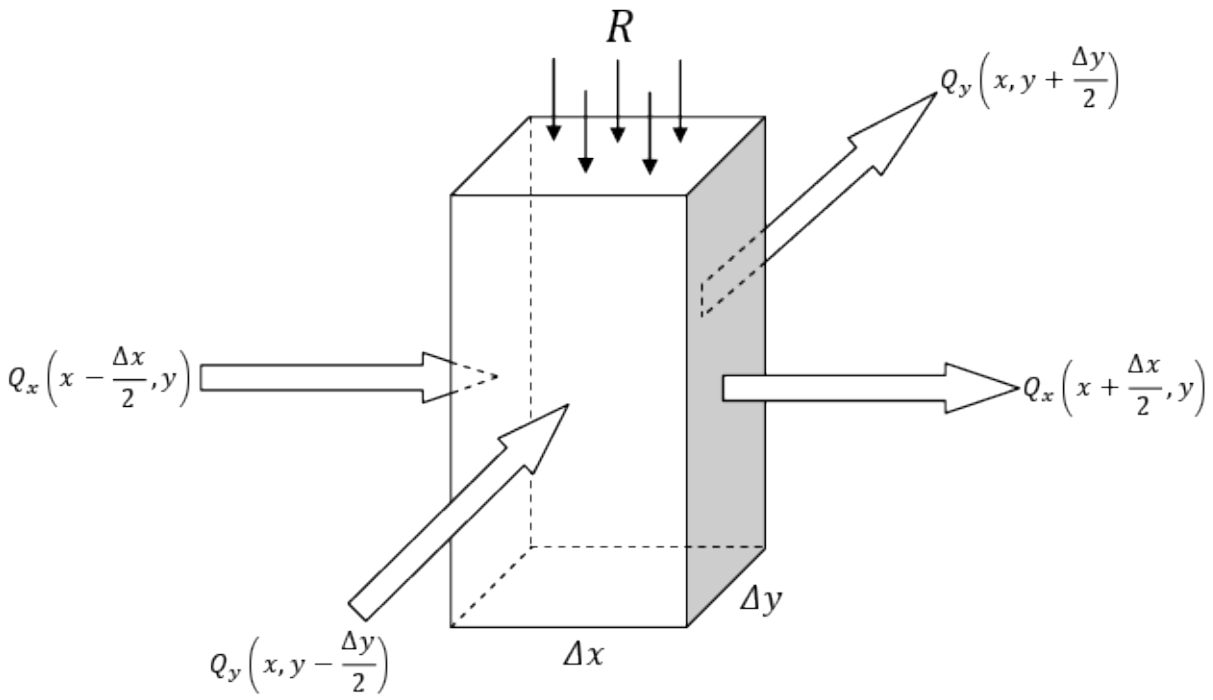

Figure 4.1: *Continuity of flow diagram with areal recharge*

$$outflow - inflow = 0 \tag{4.12}$$

and combining terms from Figure 4.1 gives

$$\begin{aligned}\left[Q_x\left(x+\frac{\Delta x}{2}, y\right) - Q_x\left(x-\frac{\Delta x}{2}, y\right)\right]\Delta y \quad &+ \\ \left[Q_y\left(x, y+\frac{\Delta y}{2}\right) - Q_y\left(x, y-\frac{\Delta y}{2}\right)\right]\Delta x \quad &+ \\ -R\Delta x\Delta y \quad &= \quad 0\end{aligned} \tag{4.13}$$

Dividing both sides of equation (4.13) by $\Delta x\Delta y$ and then passing through the limit for $\Delta x \to 0$ and $\Delta y \to 0$ and moving the recharge term to the right hand side of the equation yields

$$\frac{\partial Q_x}{\partial x^2} + \frac{\partial Q_y}{\partial y} = R \tag{4.14}$$

Again, since the model considered in this study is one-dimensional the y terms can be dropped and then, by combining (4.14) with the derivative of (4.9), we arrive at the following

differential equation for groundwater flow with areal recharge:

$$\frac{d^2\Phi}{dx^2} = -R \tag{4.15}$$

A conceptual model of one dimensional flow with recharge is shown in figure 4.2 where the bedrock elevation is constant throughout the transect, as is the recharge (R) and the hydraulic conductivity (k). The general solution to (4.15) which allows us to solve for the potentials is

$$\Phi = -\frac{R}{2}x^2 + Ax + B \tag{4.16}$$

where A and B are chosen so that at the following boundary conditions are met:

$$\Phi(x) = \begin{cases} \Phi_1, & \text{for } x = 0 \\ \Phi_2, & \text{for } x = L \end{cases}$$

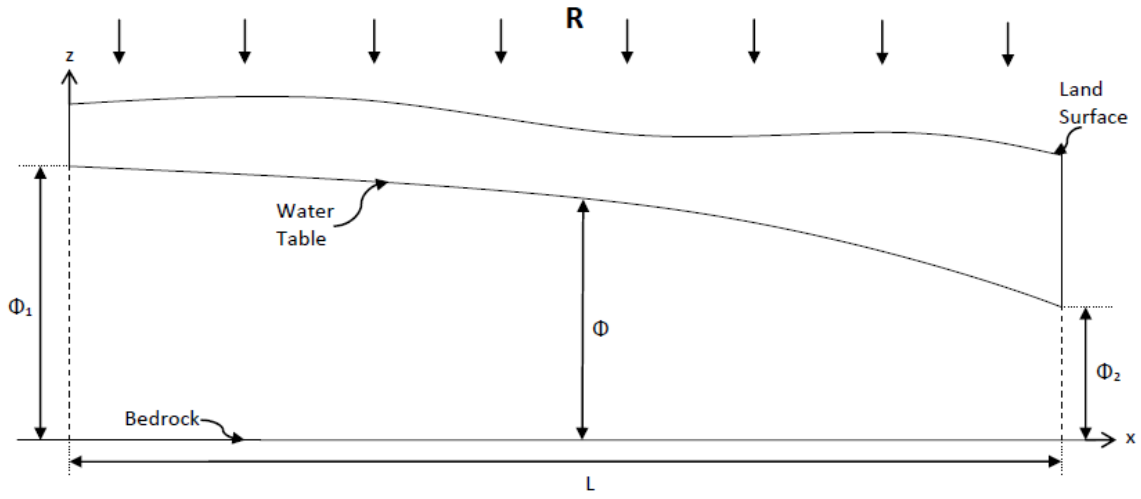

Figure 4.2: *One dimensional flow with recharge*

The values of A and B that satisfy the boundary conditions are

$$A = \frac{R}{2}L + \frac{\Phi_2 - \Phi_1}{L} \qquad B = \Phi_1 \tag{4.17}$$

Substituting these constants into (4.16) gives

$$\Phi = -\frac{R}{2}x\,(x - L) + \frac{\Phi_2 - \Phi_1}{L}x + \Phi_1 \tag{4.18}$$

Equation (4.18) is used to solve for the potential at any point in the aquifer transect shown in figure 4.2. Once the potential is calculated at a point, equation (4.11) is used to calculate the head at that location.

4.1.2 Stepping Base Approach

The equation (4.18) discussed in the previous section, is useful to calculate groundwater heads, but requires simplified conditions. The values of head or potential must be known at both ends of the transect and constant values are required on a transect for bedrock elevation, hydraulic conductivity, and recharge rates. In real world scenarios bedrock, hydraulic conductivity, and recharge rates vary over study areas. So approaches are needed to deal with changes in these conditions in order to model large aquifers. The stepping base approach [Steward, 2007] is one method to deal with varying aquifer properties and is effective in modeling aquifers with a base that gently slopes in one direction. This section presents the stepping base approach to modeling one dimensional steady state groundwater flow in an unconfined aquifer with a sloping base. The methods presented here were implemented in the Python programming language and the final scripts are shown in Appendix B.

The stepping base approach assumes an aquifer whose bed slopes gently in one direction. The High Plains aquifer satisifes this assumption quite well. Figure 4.3 shows the elevation of the bedrock and figure 4.4 shows the elevation of the predevelopment water table in the High Plains aquifer. Both figures clearly indicate a gentle west to east gradient that suits the stepping base model well.

The stepping base approach used in this study is shown conceptually in figure 4.5. In the figure an aquifer transect is split into $n = 4$ steps with each step having constant values of recharge (R_i), extraction (E_i) , bedrock elevation (B_i), and hydraulic conductivity (k_i) throughout the length (x) of the step (i). Extraction represents the groundwater-surface water interaction that is not included in the recharge term. This interaction could be groundwater discharging into rivers and springs or it could also represent areas of enhanced

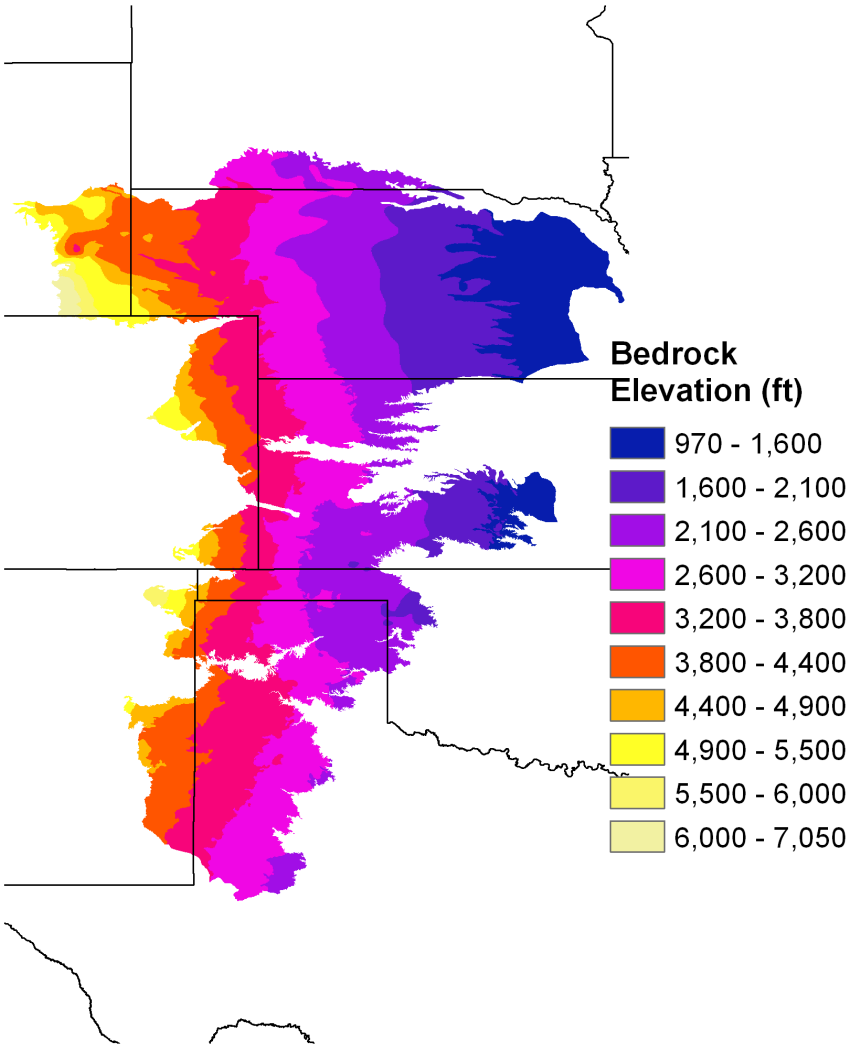

Figure 4.3: *High Plains Bedrock Elevation*

recharge where surface water is percolating down to the groundwater table. Flowrates presented as discharge per width of the aquifer ($Q(m^2/day)$) are also considered in figure 4.5 where

$$Q_i = \sum_{i}^{n} (W_i)\, x \tag{4.19}$$

and

$$W_i = R_i - E_i \tag{4.20}$$

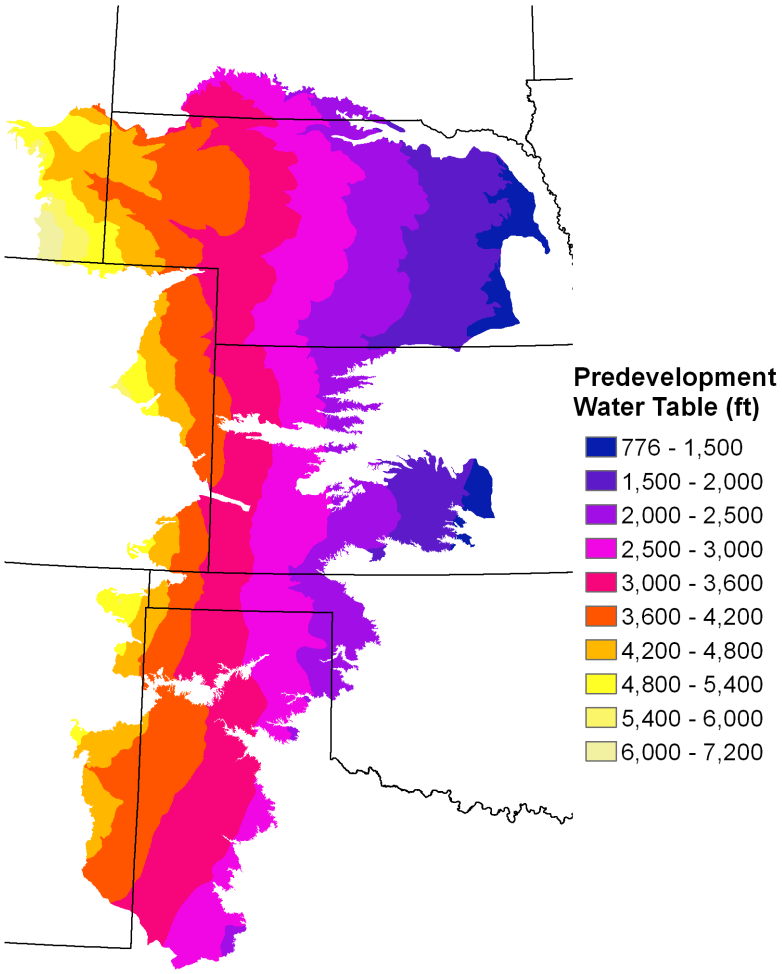

Figure 4.4: *High Plains Predevelopment Water Table Elevation*

In order to apply the stepping base approach to an aquifer, the aquifer must be split into transects or rows of constant width. The groundwater elevations are solved iteratively from right to left (lower elevation to higher elevation) in the transect [Steward, 2007]. This solution is achieved using equations formulated from (4.16). A solution to (4.16) was shown in section 4.1.1 beginning on page 25, however, a certain set of boundary conditions were used in that solution which are different from the boundary conditions used for the stepping base model. The boundary conditions for a single step in the stepping base model are as

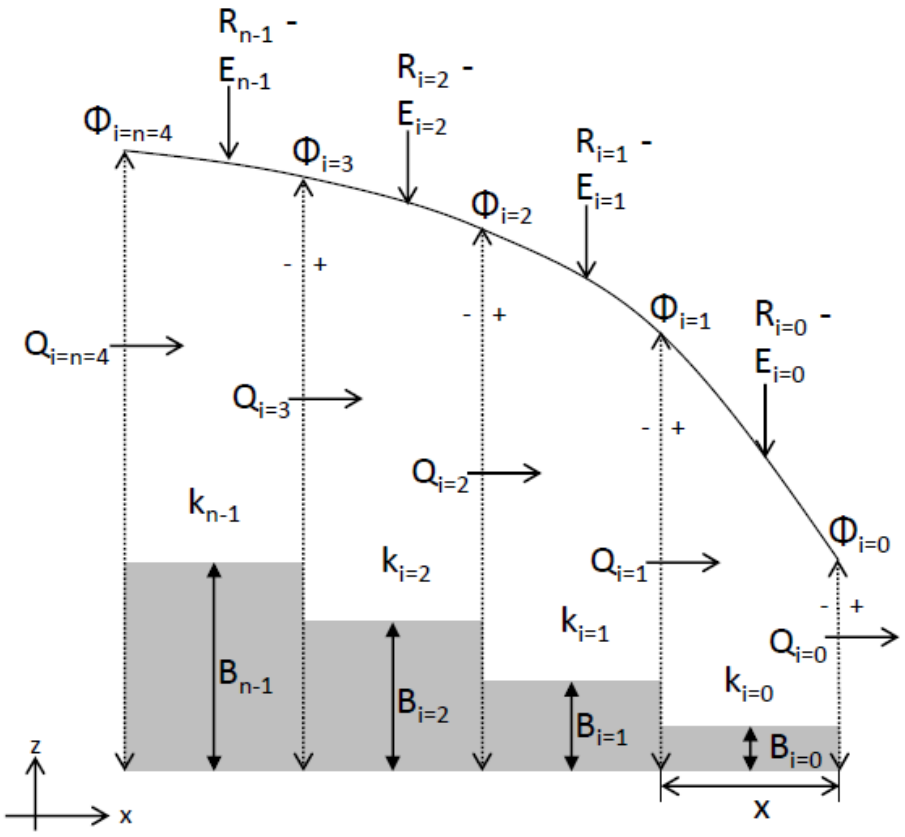

Figure 4.5: *Stepping Base Conceptual Diagram*

follows:

$$\Phi(x) = \Phi_0, \quad \text{for } x = 0$$

$$Q(x) = Q_0, \quad \text{for } x = 0$$

Figure 4.6 illustrates these conditions. The coefficients that satisfy these boundary conditions are

$$A = -Q_0 \qquad B = \Phi_0 \tag{4.21}$$

which gives the following equation used to solve for the potential:

$$\Phi = -\frac{W}{2}x^2 - Q_0 x + \Phi_0 \tag{4.22}$$

In order to use this solution the groundwater elevation must be known at the right-hand edge of the aquifer transect being modeled. This known value of head is used to initialize

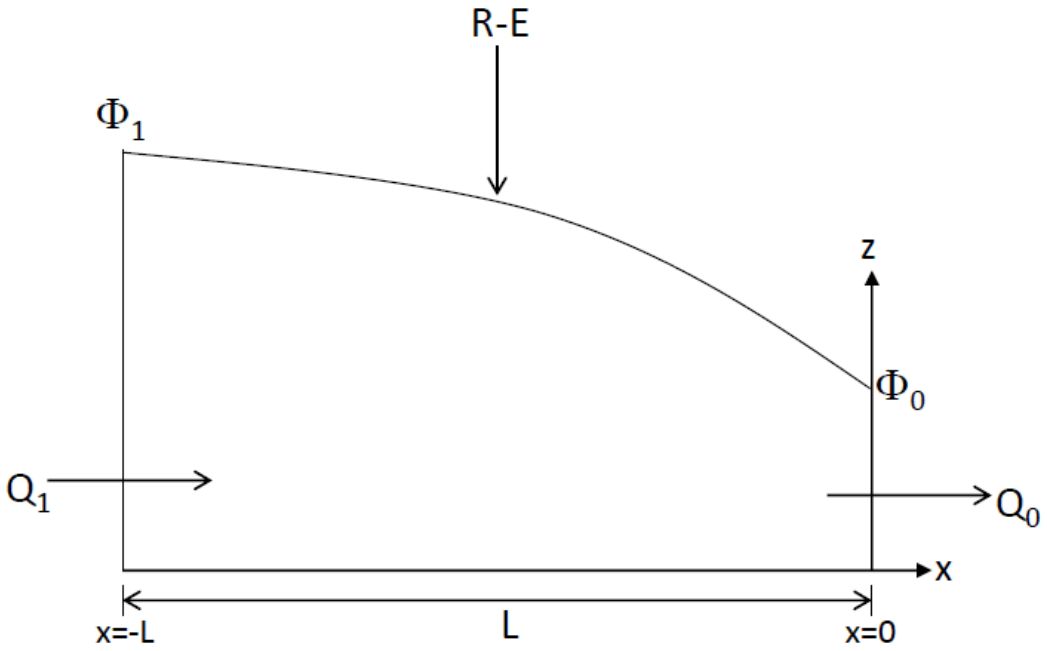

Figure 4.6: *Stepping Base Boundary Conditions*

the solution and all other values are calculated using (4.22) and the groundwater heads are obtained through the use of (4.11). The following equations are the primary equations used in the stepping base solution and have already been presented but are rewritten here to follow the notation shown in Figure 4.5:

$$\Phi_i^+ = -\frac{W_i}{2}x^2 - Q_i x + \Phi_i^- \tag{4.23}$$

$$\phi_i^+ = B_i + \sqrt{\frac{2\Phi_i^+}{K_i}} \tag{4.24}$$

$$\Phi_i^- = \frac{1}{2}K_i\left(\phi_{i-1}^+ - B_i\right)^2 \tag{4.25}$$

The following are the general steps to calculating the heads in a transect and follows the notation of Figure 4.5:

1. Calculate Φ_0^- from known ϕ_0^-

2. Calculate potential at left-hand side of first step ($\Phi_{i=0}^+$) using (4.23)

3. Then calculate the head ($\phi^{+}_{i=0}$) using (4.24)

4. Then calculate the potential on the other side of the step boundary ($\Phi^{-}_{i=1}$) using (4.25)

5. Repeat steps 2 through 4 incrementing i by 1 for each subsequent step in the model until all cells in a transect have been simulated.

As the transect is being modeled, it is possible that the jump in bedrock elevation from one cell to the next could result in the simulated groundwater head being below the bedrock elevation. If this occurs, the head at the beginning of the next step is set equal to the bedrock elevation of that step as illustrated in figure 4.7 [Steward, 2007].

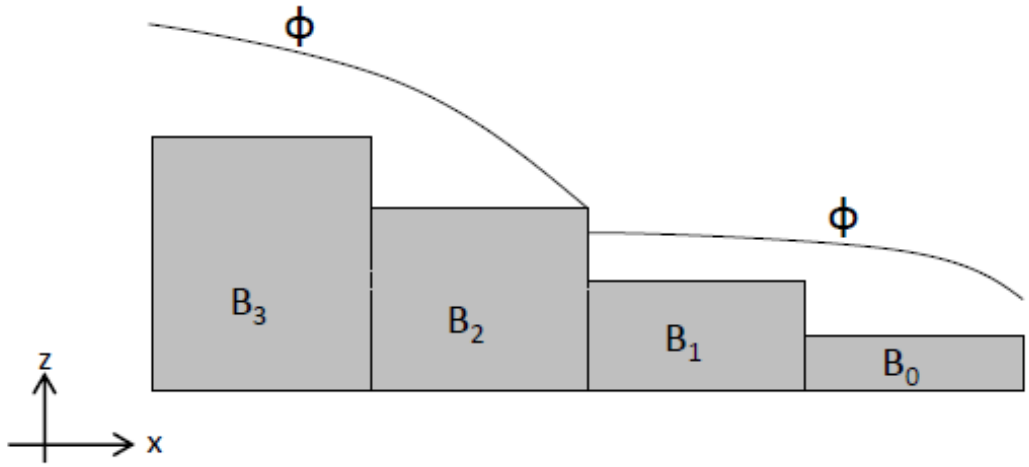

Figure 4.7: *Simulated Head Below Bedrock Elevation*

4.1.3 Data Preparation

Running the stepping base model requires input data that is aggregated across a grid. Both ArcGIS 9.3 and ArcGIS 10 were used to facilitate the preparation of input files for the stepping base model. The stepping base model requires six input data sets: bedrock elevation, predevelopment water table elevation, land surface elevation, recharge rate, hydraulic conductivity, and the aquifer extents. These data sets are represented in ArcGIS as shapefiles or rasters. A shapefile is a digital vector storage format for storing geometric location data

and its associated attribute information. Figure 4.8 is an example of a shapefile storing hydraulic conductivity data for the High Plains aquifer. Each unique hydraulic conductivity value is represented by a polygon of a certain color associated with a unique value. Bedrock elevation, predevelopment water table elevation, recharge rate, hydraulic conductivity, and the aquifer extents are all initially represented as shapefiles in ArcGIS. Detailed information on all input data sets is presented in Appendix C.

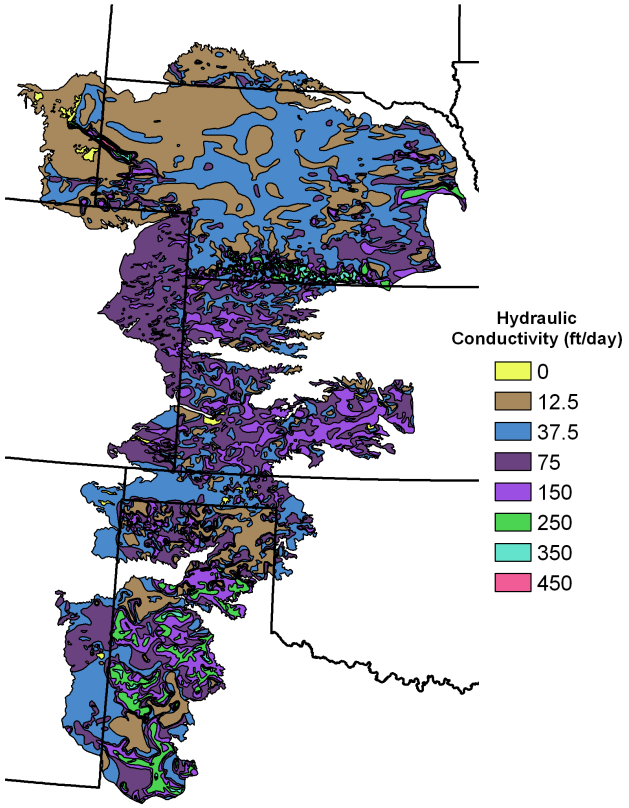

Figure 4.8: *Shapefile Example (Hydraulic Conductivity)*

A raster is a matrix of cells organized into rows and columns forming a grid. Each cell contains a value representing information at a particular geographic location. Figure 4.9 shows an example of the land surface elevation raster used in the sloping base model. In the box on the right of the figure some individual cells can be made out, each cell is 1000m^2 and represents the average land surface elevation in that area.

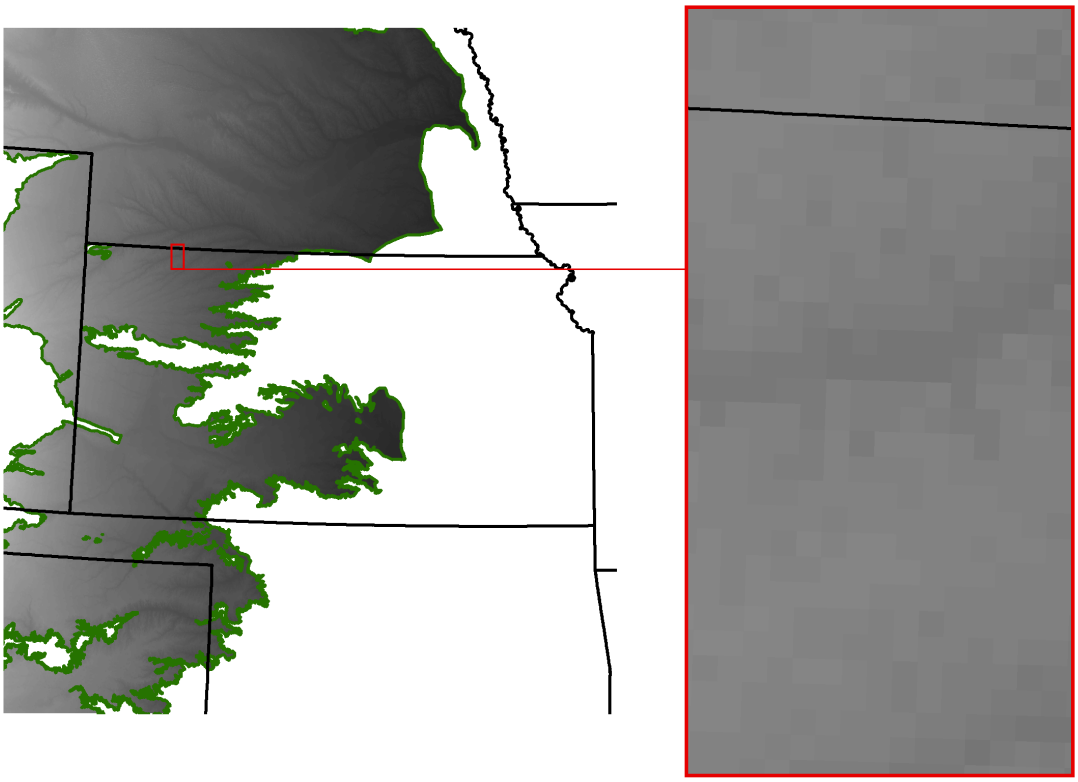

Figure 4.9: *Raster File Example (Land Surface Elevation)*

As the stepping base model simulates across a transect it will take steps of equal width. This requires that all data for a row have a common resolution. To satisfy this requirement, all input data will be aggregated into grids with cells of equal widths. Once the data is in the grid format the stepping base model will simulate one transect at a time and the results will be stored so that all transect results can be viewed at once.

Efficient execution of the sloping base model requires the data to be in a format that is easily consumed by the Python computer code in the same computer language as the sloping base model. In order to get the aquifer data into a usable format some data preprocessing must be carried out. ArcGIS has many tools that were used to process the aquifer data sets and get them to a format that is usable with the sloping base model. ArcGIS tools can be used manually within the ArcGIS user interface or they can be accessed through

the Python programming language. In this study very large datasets were used and many different models were tested requiring extensive data processing. In order to facilitate the building of several different models, the data processing routines were written in Python so that the basic processes could be easily reused without having to go into ArcGIS and utilize each separate tool manually to create a new model. Appendix A contains more detailed information on these data processing routines as well as the Python code used to create them.

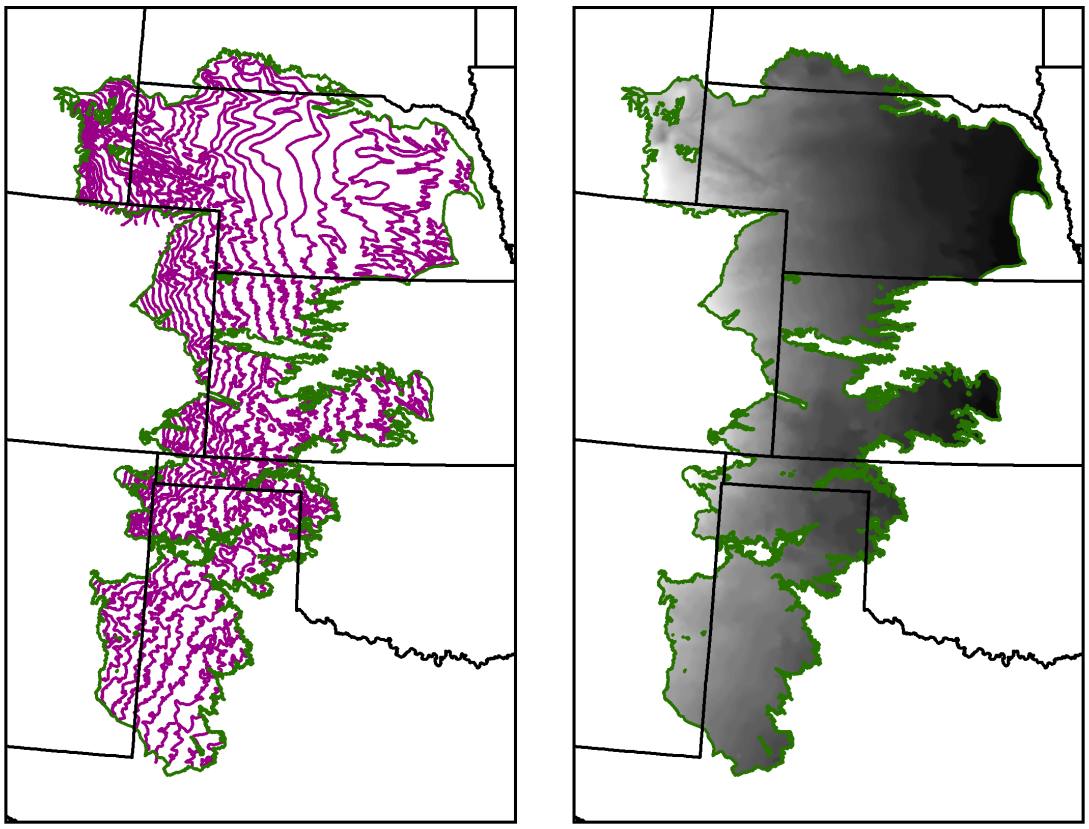

Figure 4.10: *Raster Representation of a Shapefile*

The easiest way to represent the different aquifer properties is as two-dimensional arrays with each row in the array representing a single transect of the aquifer. Rasters are already in a grid format (rows and columns) which allows them to be easily transformed into a 2D array of values. Because the shapefiles represent non-rectangular geometries they are not

usable in a grid format so they are converted to rasters in ArcGIS. Figure 4.10 shows a shapefile representing bedrock elevation contours which are interpolated to produce a raster representation of the data. When creating raster datasets it is important to know the cell size at which the model will be run. Raster are made up of square cells of constant size so whatever cell size is desired for the sloping base model, that is the cell size that should be used to generate the rasters in ArcGIS. Once the rasters with appropriate cell sizes have been created they can be exported to text files using the Raster To ASCII tool within ArcGIS. This creates a separate text file for each aquifer data set that can then be easily loaded into a 2D array in Python.

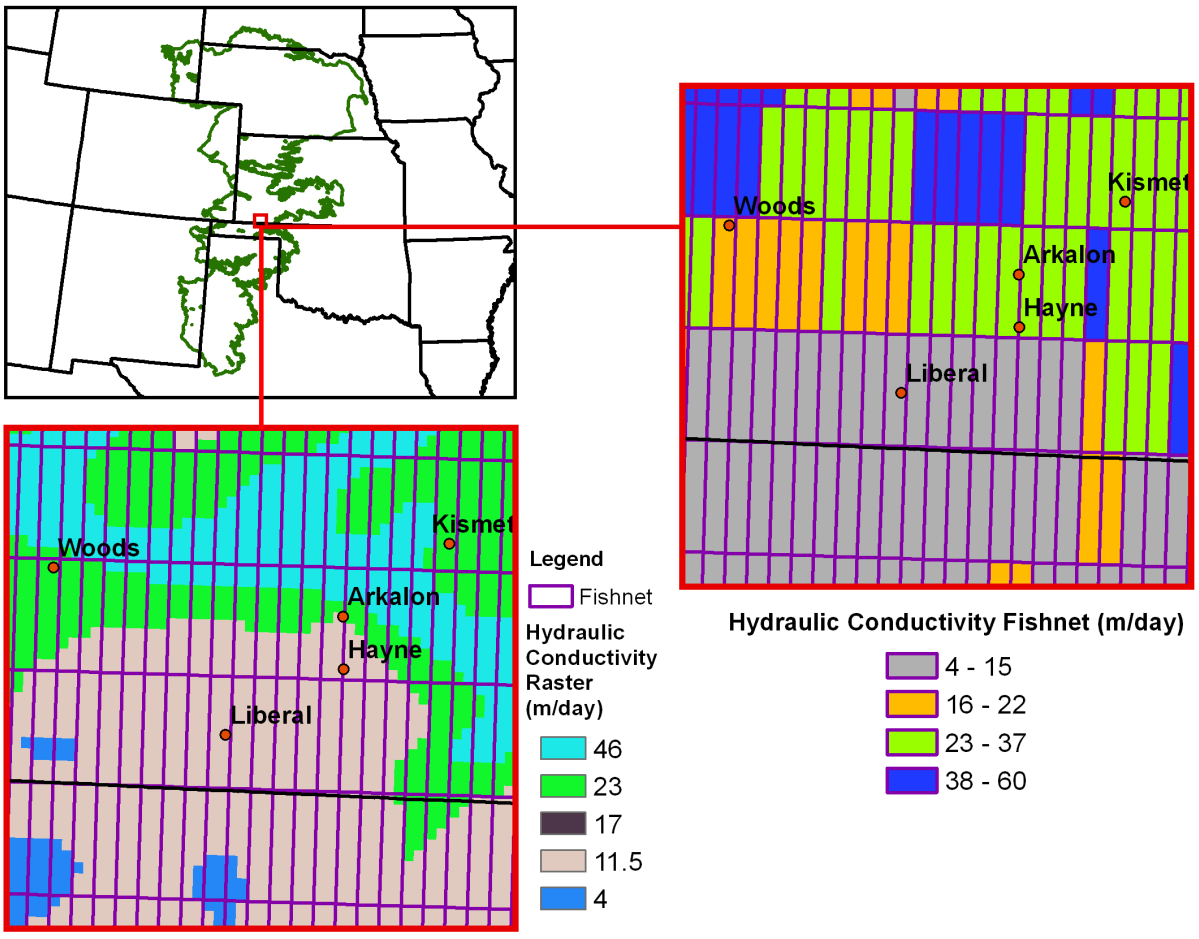

Figure 4.11: *Example of Grid Setup*

One limitation in the use of rasters to generate files for use in the sloping base model is that raster cells are always square. This makes it impossible to only use rasters if the desired model cell size is rectangular. Rectangular cells allow for data to be aggregated more in one direction than another. For example, in this study rectangular cells with lengths (north to south) of 10 kilometers and widths (west to east) of 2 kilometers were used (similar to Steward et al. [2009]). In order to create grids with rectangular cells one more step after creation of the rasters is needed before exporting to a text file. The value of each cell within the grid is calculated from the rasters by taking the average of all the raster cells that fall within the grid cell. Figure 4.11 shows an example of a 2km by 10km grid along with a raster representing hydraulic conductivity on the bottom left and the resulting conductivity values aggregated up to the grid size on the right. Each grid cell contains a single value that represents the hydraulic conductivity for the entire area within the cell. Once the grids have been created for each aquifer data set, they can be exported to text files which are read into Python as 2D arrays.

4.2 Levenberg-Marquardt Optimization

The Levenberg-Marquardt algorithm is a technique for minimizing both linear and non-linear functions over a space of parameters for the function. It combines Newton's method which is used near a minimum and the gradient descent method which is used when at large distances from a minimum. In this study Levenberg-Marquardt is used to minimize the following objective function (F):

$$F = \sum_{m=1}^{M} \left[\phi\left(v_m, E\right) - \phi_m\right]^2 \tag{4.26}$$

where ϕ_m are the measured or known values of head and $\phi\left(v_m, E\right)$ are the simulated values of head. The Levenberg-Marquardt routine is carried out row by row following the grid setup discussed in section 4.1.3.

The parameters to be optimized are represented by E which is a set of groundwater-

surface water interaction rates $[Length^2/time]$ for the aquifer row being simulated. Each value in E represents the average groundwater-surface water interaction rate for a particular cell in the row being simulated, and the values of E can be positive or negative. A negative value represents water being transmitted from a surface water body to groundwater and a positive value represents groundwater discharging into a surface water body. Other parameters (hydraulic conductivity, bedrock elevation) that are used to calculate groundwater heads (see 4.1.2) are represented by v_m. These parameters v_m are not adjusted at any point in the optimization process.

To arrive at the Levenber-Marquardt method, the steepest descent method and Newton's method provide a foundation. The steepest descent method uses the q^{th} estimate of the parameters $E|_q$ to obtain the next iterate $E|_{q+1}$ using

$$E|_{q+1} - E|_q = -\frac{\Delta x}{|g|_q} g|_q \tag{4.27}$$

The gradient vector of the objective function (F) with respect to the unknown coefficients is calculated using the coefficients $E|_q$ and is given by

$$\mathbf{g} = \begin{bmatrix} \frac{\partial F}{\partial E_1} \\ \frac{\partial F}{\partial E_2} \\ \vdots \\ \frac{\partial F}{\partial E_N} \end{bmatrix} \quad , \quad \frac{\partial F}{\partial E_n} = 2 \sum_{m=1}^{M} \frac{\partial \phi\left(v_m, E\right)}{\partial E_n} \left[\phi(v_m, E) - \phi_m\right] \tag{4.28}$$

Each iterate in the steepest descent method adjusts the coefficients by taking a step Δx in the direction of the gradient of F. Newton's method determines its next step using

$$\mathbf{H}|_q \left(E|_{q+1} - E|_q\right) = -\mathbf{g}|_q \tag{4.29}$$

where $\mathbf{H}$ is the Hessian matrix composed of the second derivatives given by

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 F}{\partial E_1 \partial E_1} & \frac{\partial^2 F}{\partial E_1 \partial E_2} & \cdots & \frac{\partial^2 F}{\partial E_1 \partial E_N} \\ \frac{\partial^2 F}{\partial E_2 \partial E_1} & \frac{\partial^2 F}{\partial E_2 \partial E_2} & \cdots & \frac{\partial^2 F}{\partial E_2 \partial E_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 F}{\partial E_N \partial E_1} & \frac{\partial^2 F}{\partial E_N \partial E_2} & \cdots & \frac{\partial^2 F}{\partial E_N \partial E_N} \end{bmatrix} \quad , \quad \frac{\partial^2 F}{\partial E_n \partial E_k} = 2 \sum_{m=1}^{M} \frac{\partial \phi}{\partial E_n} \frac{\partial \phi}{\partial E_k} + \frac{\partial^2 \phi}{\partial E_n \partial E_k} \left(\phi - \phi_m\right) \tag{4.30}$$

Combining these two methods produces the following equation used in the Levenberg-Marquardt method:

$$\left(\mathbf{H}|_q + \lambda \mathbf{I}\right)\left(E|_{q+1} - E|_q\right) = -\mathbf{g}|_q \tag{4.31}$$

In (4.31) $\mathbf{I}$ is the identity matrix and λ is a coefficient that is adjusted in each iteration. The Levenberg-Marquardt method may also be written in terms of the Jacobian matrix $\mathbf{J}$ as

$$\left(\mathbf{J}^T\mathbf{J}|_q + \lambda \mathbf{I}\right)\left(E|_{q+1} - E|_q\right) = -\mathbf{J}^T\boldsymbol{\phi}|_q \tag{4.32}$$

where

$$\mathbf{J} = \begin{bmatrix} \frac{\partial\phi(v_1,E)}{\partial E_1} & \frac{\partial\phi(v_1,E)}{\partial E_2} & \dots & \frac{\partial\phi(v_1,E)}{\partial E_N} \\ \frac{\partial\phi(v_2,E)}{\partial E_1} & \frac{\partial\phi(v_2,E)}{\partial E_2} & \dots & \frac{\partial\phi(v_2,E)}{\partial E_N} \\ \dots & \dots & \dots & \dots \\ \frac{\partial\phi(v_M,E)}{\partial E_1} & \frac{\partial\phi(v_M,E)}{\partial E_2} & \dots & \frac{\partial\phi(v_M,E)}{\partial E_N} \end{bmatrix}, \qquad \boldsymbol{\phi} = \begin{bmatrix} \phi\left(v_1, E\right) - \phi_1 \\ \phi\left(v_2, E\right) - \phi_2 \\ \vdots \\ \phi\left(v_M, E\right) - \phi_M \end{bmatrix} \tag{4.33}$$

These are the typical steps followed in the Levenberg-Marquardt algorithm:

1. Set $\lambda = 0.01$
2. evaluate the objective function F
3. solve for $E|_{q+1}$ and reevaluate F
4. if F increases set $\lambda = \lambda\nu$, otherwise set $\lambda = \lambda/\nu$ where $\nu > 1$
5. continue solving and adjusting λ for each iteration until $E|_q - E|_{q+1}$ is small or some other user defined stoppage criteria, such as a maximum number of iterations, is met.

The Levenberg-Marquardt optimization technique was implemented in the Python programming language and the code is shown in Appendix B. For each transect in the High Plains Aquifer, potential locations of groundwater-surface water interaction were identified using a depth to water criteria. A maximum depth to water (DTW_{max}) was selected and any cell in a transect with a depth to water less than DTW_{max} was selected as a potential extraction point and added to the parameter set E. An arbitrary range of 6m - 10m was used

when setting the value for DTW_{max} in this study. To initialize the optimization routine, all parameters in parameter set E were set equal to some initial value which varied depending on the cell size being used to model the High Plains Aquifer.

4.3 Particle Swarm Optimization

Particle swarm optimization (PSO) is a population based optimization technique developed by Kennedy and Eberhart [1995] where the population is a set of potential solutions known as particles. In this study PSO was used to optimize (4.26). The PSO routine was written in the Python programming language and the code is presented in Appendix B. PSO is initialized by creating random sets of E with the number of sets being equal to the desired population size (n). As a general rule of thumb a ratio of 8 particles per parameter in parameter set E_n was maintained as much as possible throughout all optimization runs. Difficulty in following this ratio arises when the number of parameters becomes large meaning $\phi(v_m, E_n)$ must be evaluated many times resulting in excessively long run times. The number of parameters in E_n was determined by the same method described in section 4.2 where parameter locations in a transect were determined by DTW_{max}.

The sets of E are initialized by giving an initial guess value for each parameter in a parameter set E_n. The initial guess is a random number between a lower bound (xmin) and an upper bound (xmax). The values of xmin and xmax are determined by the user. Once all sets of E are ititialized $\phi(v_m, E_n)$ for each particle (E_n) is evaluated and the fitness of each solution is checked by evaluating the objective function (4.26). The location of each particle in the parameter space is then updated. Two values are kept track of as the particles move from iteration to iteration. The first value is the best overall solution that has been found so far which is known as the global best (*gbest*). The second value is the location of the best solution found by each individual particle which is known as the personal best (*pbest*). Every particle is aware of the global best and it's own personal best value, and these are the two values that are used to direct the movement of the particles through the parameter

space. The following are the equations used to update the location of an individual particle:

$$x_{k+1} = x_k + v_{k+1} \tag{4.34}$$

where x is the vector containing the coordinates of a particle in the parameter space, k is the iteration number, and v_{k+1} is known as the velocity vector of the particle which is calculated with the following:

$$v_{k+1} = v_k + a\,(pbest - x_k) + b\,(gbest - x_k) \tag{4.35}$$

where a and b are coefficients that weight the amount of influence that *pbest* and *gbest* have on a particle's movement. Both of these coefficients are selected randomly in each iteration from a set range of values. The range used for a was 0-1.5 and for b the range was 0-2.5. In order to keep v_{k+1} from growing too large, a maximum velocity value (v_{max}) can be set and any component of the vector $v_{k+1} > v_{max}$ will be reduced to v_{max}.

PSO also allows for the use of a penalty function. A penalty function provides the ability to increase the objective function when certain user defined criteria are met. With a penalty function, the user can attempt to drive the solution in a desired direction. For example, if a maximum value is known for a parameter then a line of code can be added to the PSO algorithm that adds some amount to the objective function for each parameter that exceeds the known maximum. This will drive the solution away from parameter values that are unrealistically large.

Chapter 5

Application

This chapter discusses how the sloping base model and optimization methods discussed in Chapter 4 were implemented and the results obtained from them. The data sets used in the models are presented in Appendix C.

5.1 Model Results and Discussion

The sloping base model was first tested with a no groundwater-surface water interaction scenario. Under this scenario there are constant recharge values for each cell in the aquifer but the only point of discharge is at the eastern most cell of each transect. A plot of the results for one of these transects is shown in figure 5.1. In the figure it can be seen that the simulated head in blue is greatly overestimated. This is expected because the recharge from each cell along the transect is building up and has no points at which it can discharge other than the end of the transect. The aquifer can not physically move that amount of water through the entire transect without some of it being discharged to surface water bodies.

The next step was to incorporate groundwater-surface water interaction into the model. Three different grid sizes were used throughout the modeling process. Initially 1km x 1km and and lastly a 2km x 10km grid was used. These grids contained 1,095,888 cells and 55,080 cells respectively. In order to efficiently search the aquifer for locations of groundwater-surface water interaction and then quantify them the Levenberg-Marquardt algorithm was initially used on the $1km^2$ grid.

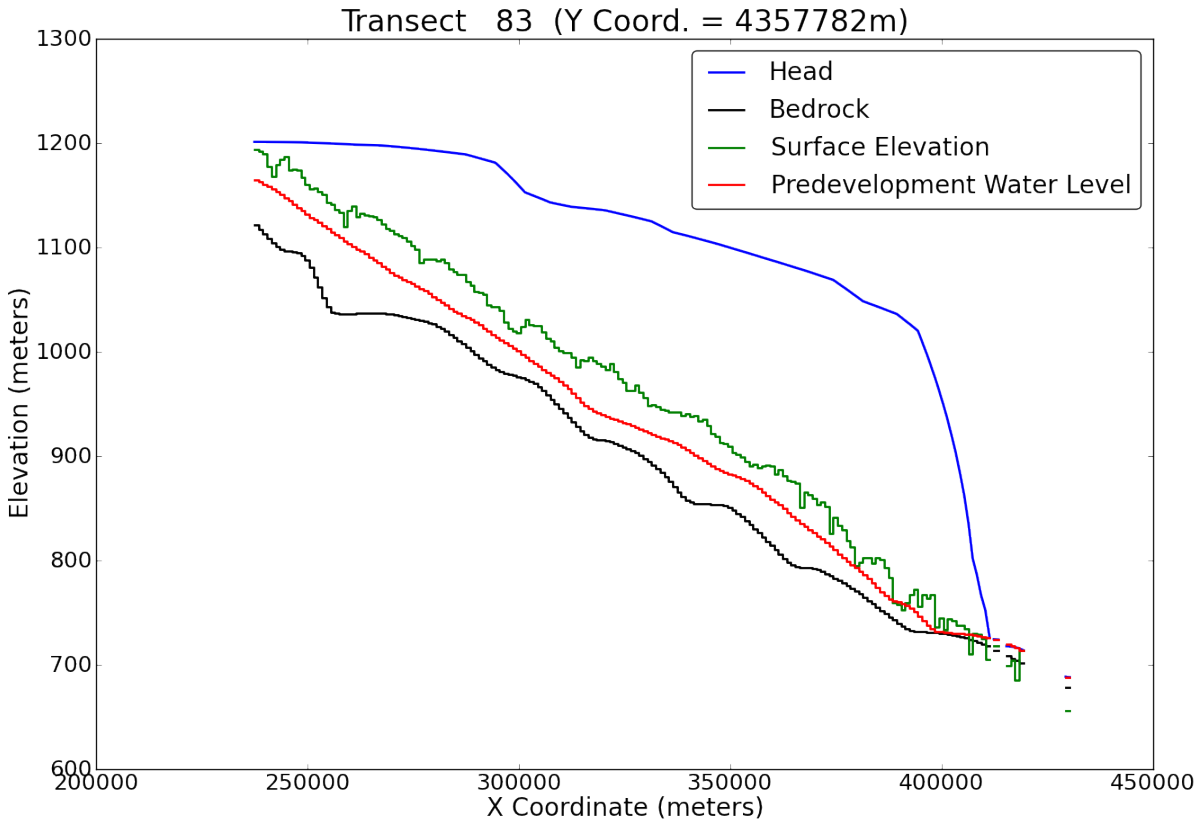

Figure 5.1: *Simulated Transect Results with no Groundwater-Surface Water Interaction*

The first step in initializing the Levenberg-Marquardt optimization runs was to pick the cells in the grid that would be considered as possible points of interaction. These points were selected using two primary methods, the interval method and the depth to water method. The interval method simply selected points at a given interval and the depth to water method selected points based on a maximum depth to water. The depth to water method should give the most realistic results as potential for groundwater-surface water interaction will decrease with increasing depth to water. The interval method pays no attention to the depth of the groundwater table when selecting points of interaction.

Once the points of interaction were selected they were given an initial guess for their discharge value in m^2/day. Since the Levenberg-Marquartd algorithm is a gradient descent

method the initial guess had a significant impact on the model results. If a good initial guess was not given the Levenberg-Marqaurdt optimization would converge to a local minimum and not a global minimum. Convergence to a local minimum would produce highly erroneous results. Much trial and error determined the best initial guess values given in table 5.1.

Table 5.1: *Extraction Initialization*

Cell Size (m)	Extraction (m^2/d)
1000x1000	0.01
5000x5000	0.5

5.1.1 Initial Levenberg-Marquardt Results

Using the initial guess value for the 1000m^2 grid shown in table 5.1 produced the results shown in figures 5.2 and 5.3. Model errors were calculated by subtracting the simulated water level from the measured predevelopment water level.

The Levenberg-Marquardt routine eliminated most of the error in the simulated heads as shown in figure 5.2 and 5.4. The transect shown in figure 5.4 is typical of all the transects simulated with the optimized solution represented by the green line almost perfectly following the predevelopment water level measurements shown by the red x's. Although this solution has little error in the simulated heads it has much error in the simulated extraction. The Levenberg-Marquardt optimization algorithm used did not take into account error's which are represented by things like unrealistically large extraction rate values or points of groundwater-surface water interaction occurring in areas where the groundwater is very deep. The results shown in these figures were only achievable though the use of the interval method in selecting interaction points. An interval of 1 was used so that every single cell in the model was a potential point of interaction. The results of this can be seen in figure 5.3 where every single point in the aquifer has some rate of interaction occurring. In reality, however, there are many locations in the aquifer where no groundwater-surface water interaction is occurring.

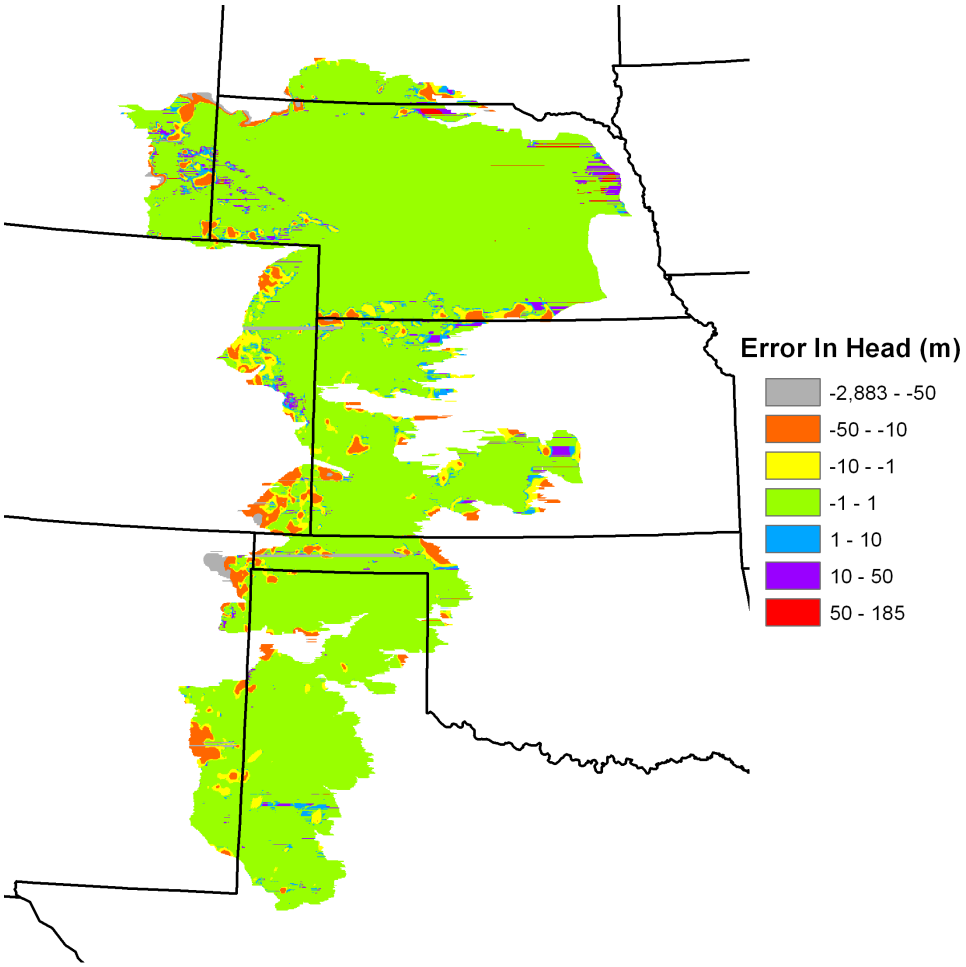

Figure 5.2: *Sloping Base Levenberg-Marquardt Optimization Results: Head Error (1km by 1km grid cells)*

In order to more accurately simulate groundwater-surface water interaction in the High Plains Aquifer the depth to water method of locating points of interaction was used. This method was intended to help eliminate the error of having interaction points in unrealistic locations. This method was tested out in the Kansas portion of the aquifer using the 1km x 1km grid and the initial guess given in table 5.1. A maximum depth to water (DTW_{max}) of 6m was used meaning any cell with a depth to water of 6m or less was considered a potential point of interaction. The results are shown in figures 5.5 and 5.6.

The errors resulting from the use of the depth to water method of locating points of

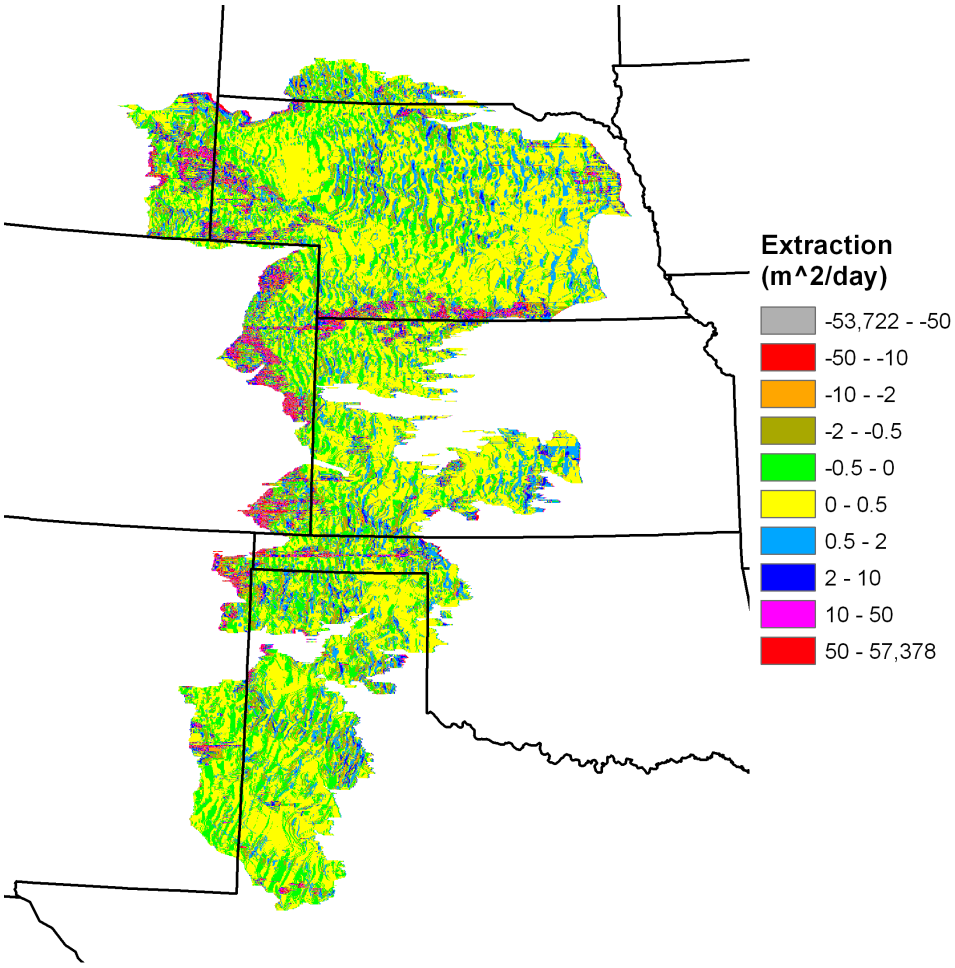

Figure 5.3: *Sloping Base Levenberg-Marquardt Optimization Results: Interaction Rates (1km by 1km grid cells)*

interaction were quite large as can be seen in figure 5.5. The points of interaction were limited to more realistic locations (figure 5.6) like river valleys but the Levenberg-Marquardt routine was never able to find a combination of extraction rates that produced good model results. Initial guesses other than $0.01\text{m}^2/\text{day}$ were tried but the results were the same and the models were unable to match predevelopment measurements of head.

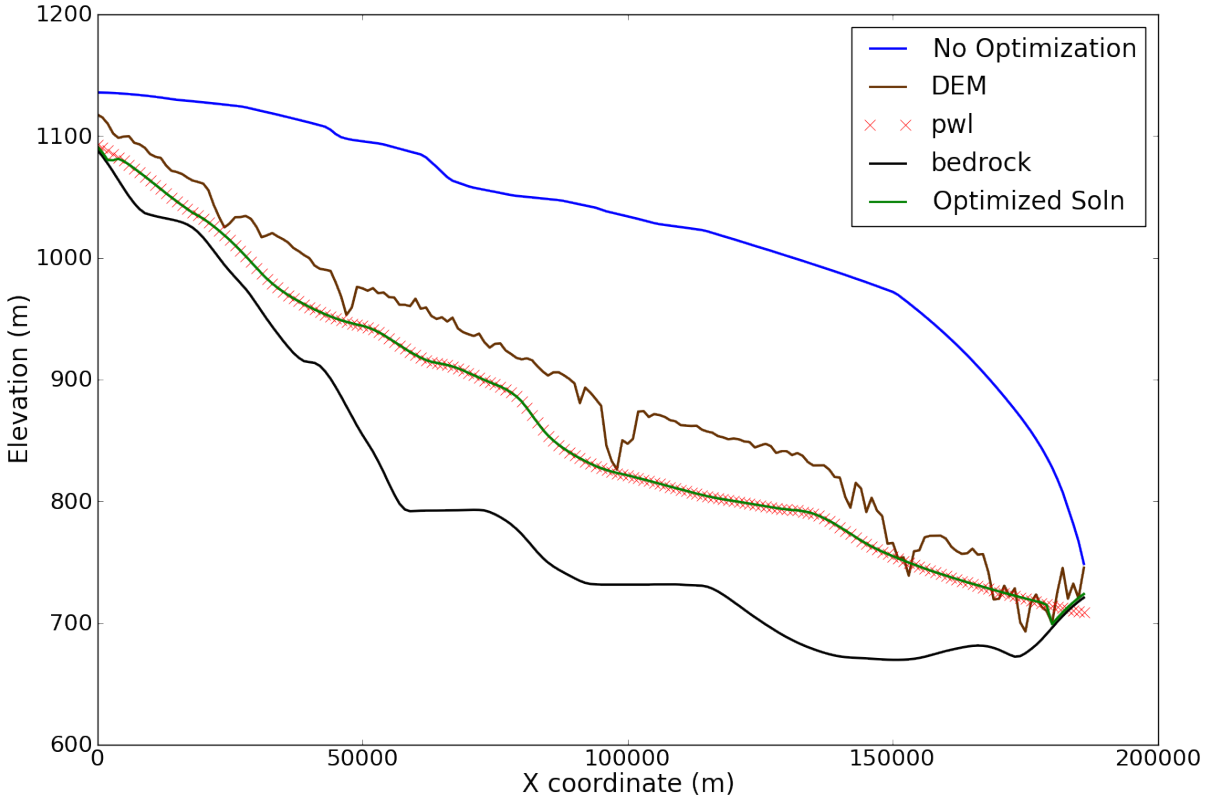

Figure 5.4: *Simulated Transect Results with Levenberg-Marquardt Optimization (1km by 1km grid cells)*

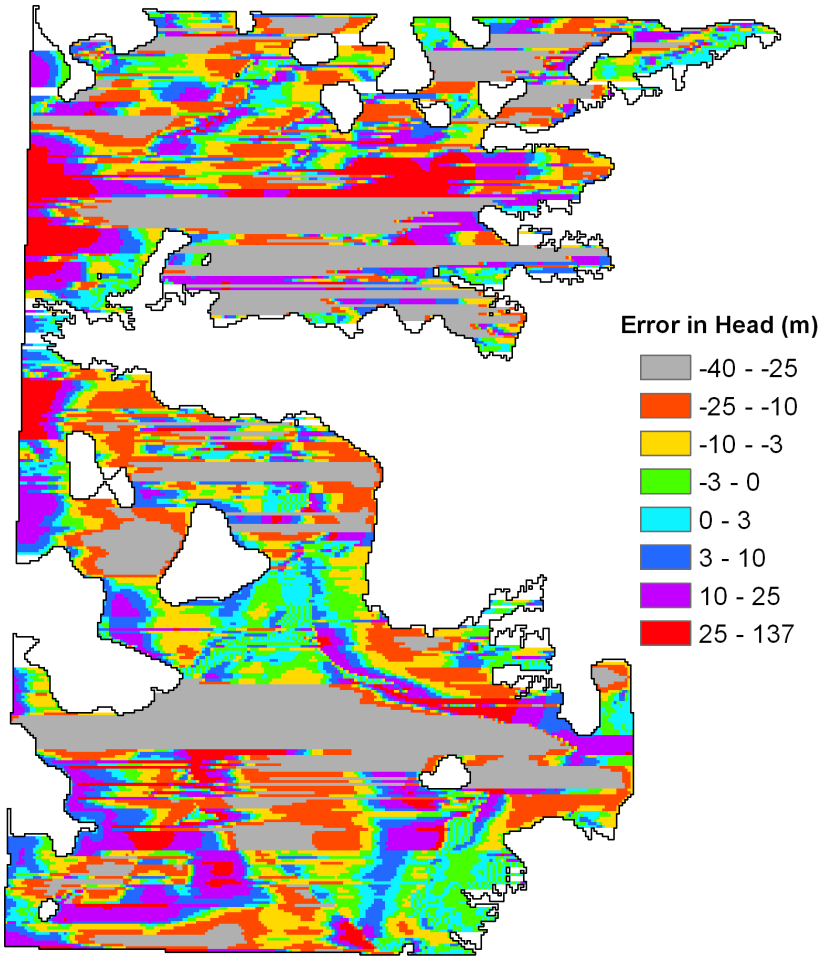

Figure 5.5: *Sloping Base Levenberg-Marquardt Optimization Results: Head Error (1km by 1km grid cells, DTW_{max}=6m)*

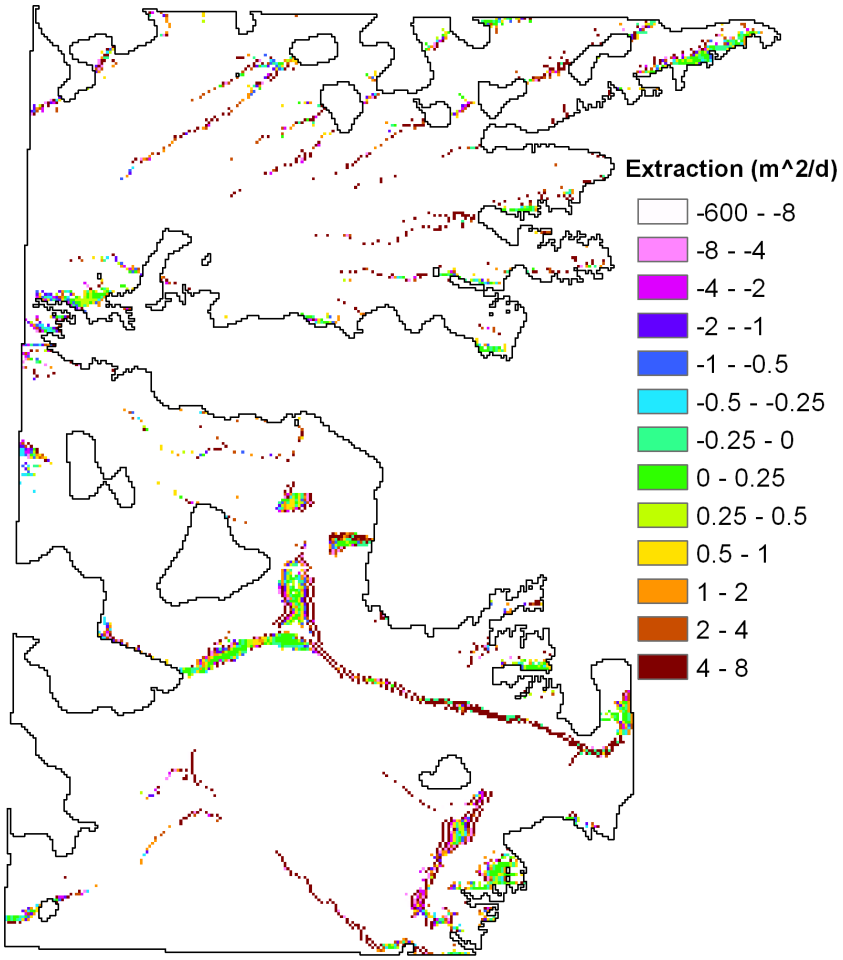

Figure 5.6: *Sloping Base Levenberg-Marquardt Optimization Results: Interaction Rates (1km by 1km grid cells, DTW_{max}=6m)*

5.1.2 Particle Swarm Results

One possible reason for the models inability to find a good combination of realistic interaction points and interaction rates was the tendency of Levenberg-Marquardt to converge to local minima. To overcome this the Particle Swarm Optimization technique was used. Because Particle Swarm Optimization (PSO) tests many different solutions at a time it is less likely to get stuck in a local minima. Also in order to simplify the model and reduce the total simulation time required the data used in the sloping base model was aggregated up to a 2km x 10km grid cell size.

The PSO routine has several parameters that need to be set before running the model. The parameters used in the PSO routine are as follows:

UseMinElevation If this value is set to yes in the code (Appendix B) then the minimum elevation in a grid cell is used instead of the average elevation.

mode Can use the depth to water mode (DTW) or the interval mode.

maxDTW Maximum depth to water value to use if using the depth to water method of finding points of interaction.

xmin The minimum value of the initial guess (see section 4.3)

xmax The maximum value of the initial guess (see section 4.3)

xmaxEnforce The maximum value of extraction allowed before the objective function is penalized

enforceXMAX If this value is set to yes then xmaxEnforce is used if it is set to no then no penalty will be enforced for values larger than xmaxEnforce

xminEnforce The minimum value of extraction allowed before the objective function is penalized

enforceXMIN If this value is set to yes then xminEnforce is used if it is set to no then no penalty will be enforced for values smaller than xminEnforce

popsize The number of particles used

maxIter The maximimum number of iterations before the optimization terminates

Vmax The maximum velocity (see section 4.3)

amax Parameter movement of a particle (see section 4.3)

bmax Parameter movement of a particle (see section 4.3)

penalty Description of penalty functions used if any (see section 4.3)

The PSO routine was used for several model runs. Figures 5.7 and 5.9 show the error and extraction maps of the best results achieved through usage of PSO. Table 5.2 shows the parameters used to achieve these results.

Table 5.2: *Parameters used to Obtain Best PSO Results*

Parameter Name	Parameter Value
useMinElevation	Yes
mode	DTW
maxDTW	10
xmin	0
xmax	0.0005
xmaxEnforce	2.5
enforceXMAX	Yes
xminEnforce	-10
enforceXMIN	Yes
popsize	200
maxIter	350
Vmax	0.0001
amax	1.5
bmax	2.5
penalty	+50000 for ea. E > xmaxEnforce & +3000 for ea. E not = 0

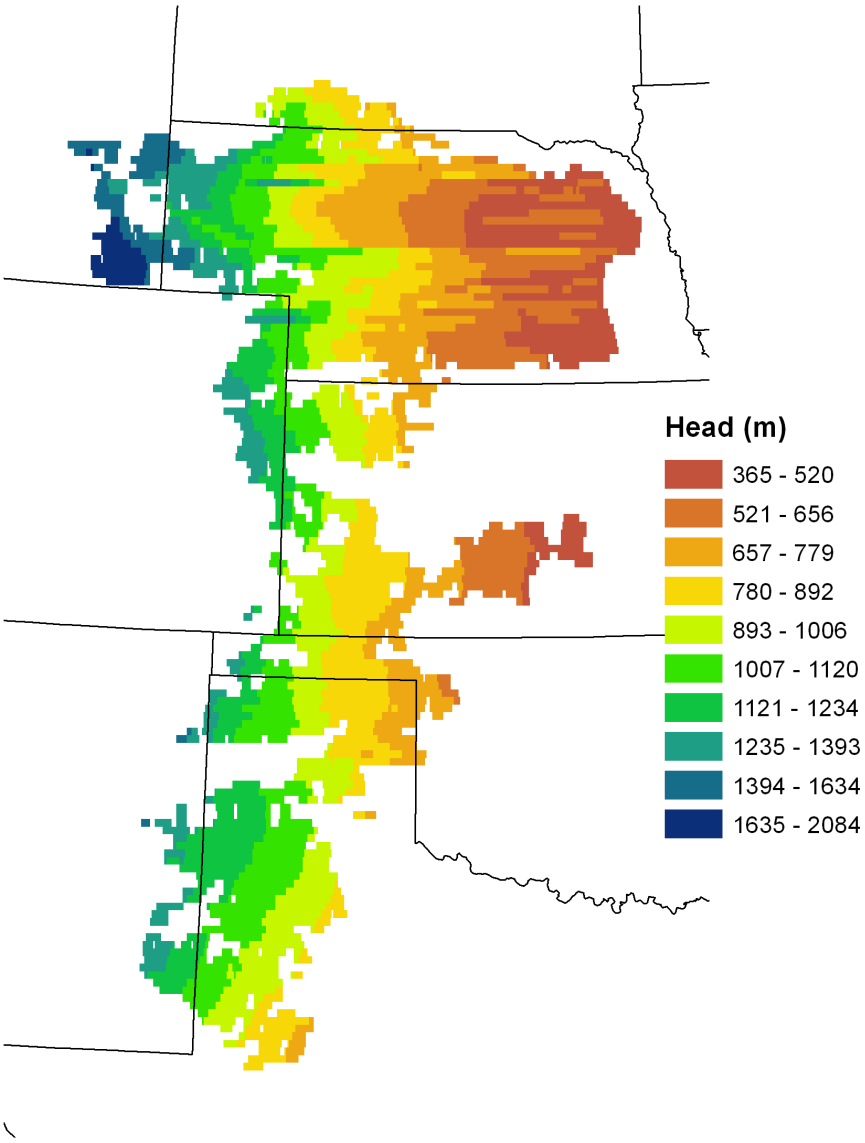

Figure 5.7: *Sloping Base PSO Results: Heads (2km by 10km grid cells)*

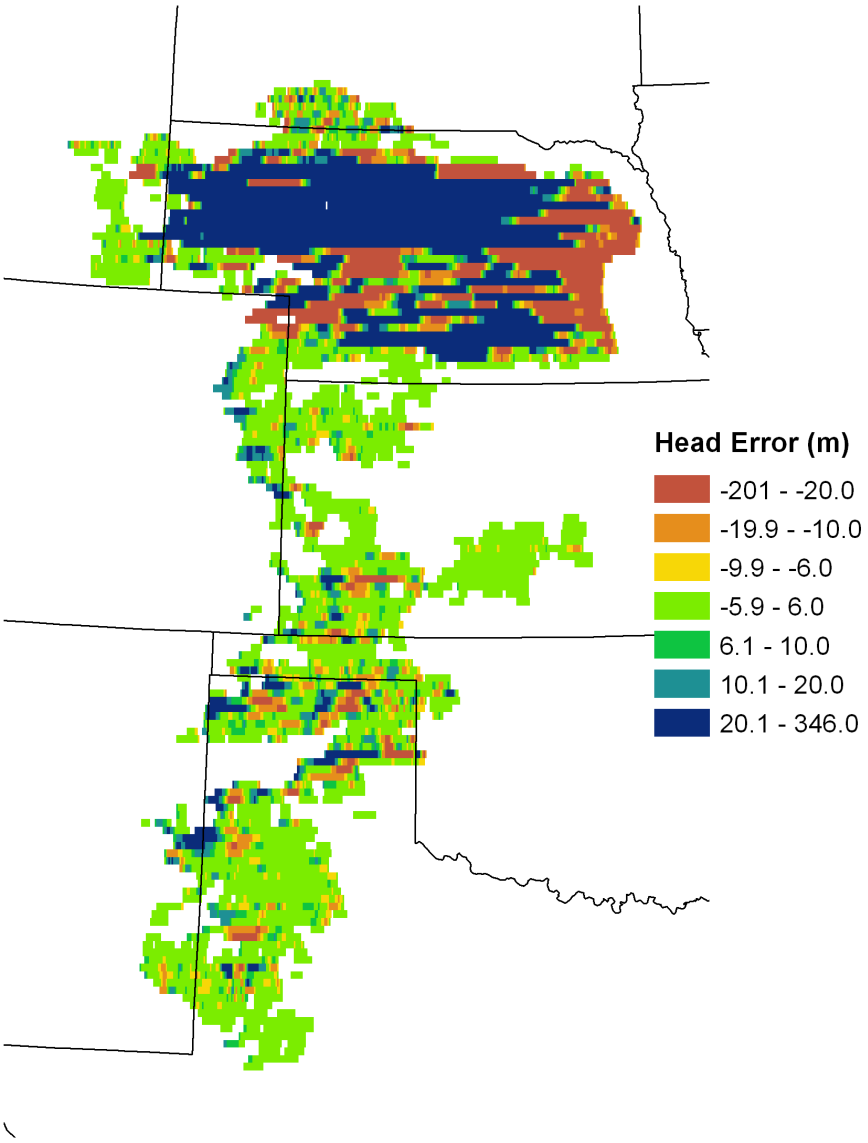

Figure 5.8: *Sloping Base PSO Results: Simulated Head Error (2km by 10km grid cells)*

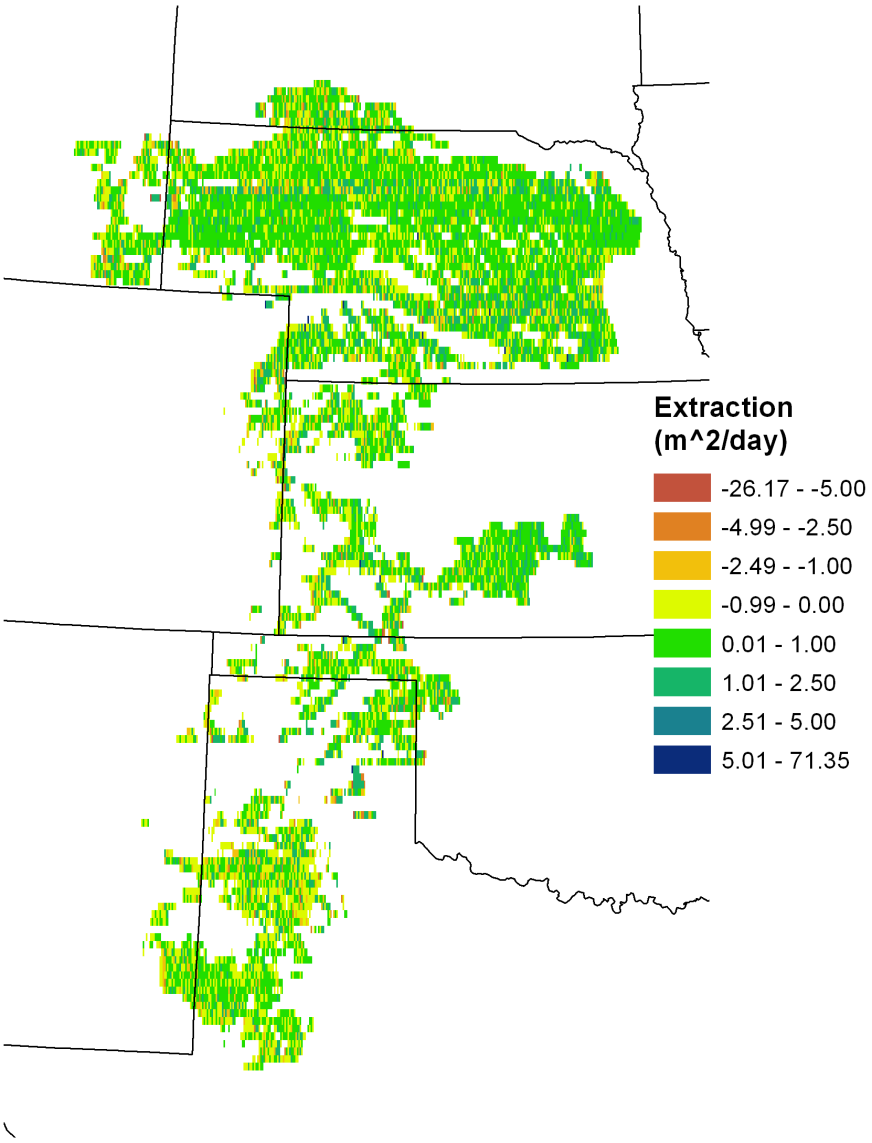

Figure 5.9: *Sloping Base PSO Results: Interaction Rates (2km by 10km grid cells)*

The model did well in simulating heads in the southern two thirds of the High Plains Aquifer but struggled in Nebraska. This is likely due to the high level of complexity present in this basin. In Nebraska the aquifer transects are quite long west to east compared to other portions of the aquifer. Also there are many rivers and many areas where the water table is near to the land surface. This results in a transect have a very high number of possible points of interaction. This can be seen in the map in figure 5.9 where there is interaction occurring in most cells in Nebraska. While PSO is much better at avoiding local minima than Levenberg-Marquardt it will still struggle when the parameter space becomes too complex. Many of the transects in Nebraska have around 250 possible points of interaction. This means the PSO routine has to search a 250 dimensional parameter space to try and find the best combination of interaction rates. In such a complex parameter space it is probable that the PSO routine is simply unable to perform an effective search. The search can be made more effective by increasing the number of particles and the maximum number of iterations but the search time becomes prohibitive as the population size and maximum number of iterations grows too large.

Greater numbers of particles and increased maximum number of iterations were tried to see if the solution in Nebraska could be improved. Results showed that increasing the population size and maximum number of iteration had a negligible impact on the quality of the results while drastically increasing the simulation time required to achieve those results. Figures 5.10-5.13 show the impact of increased populations on a transect in Nebraska. A maximum iteration of 400 was used for all figures. The results shown were typical for other transects in the area as well, increasing the population size up to 2500 resulted in little to no improvement in the final results.

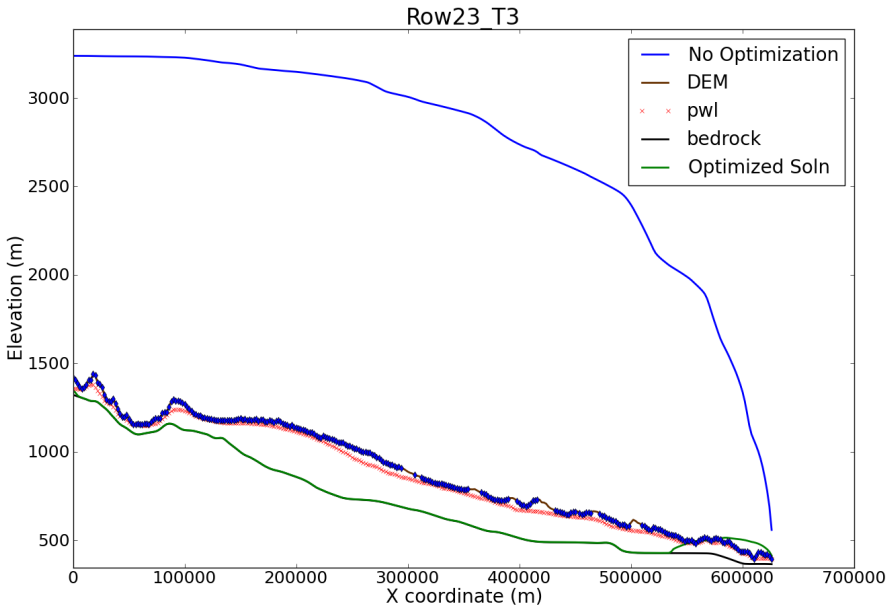

Figure 5.10: *PSO Transect Results with Population of 500*

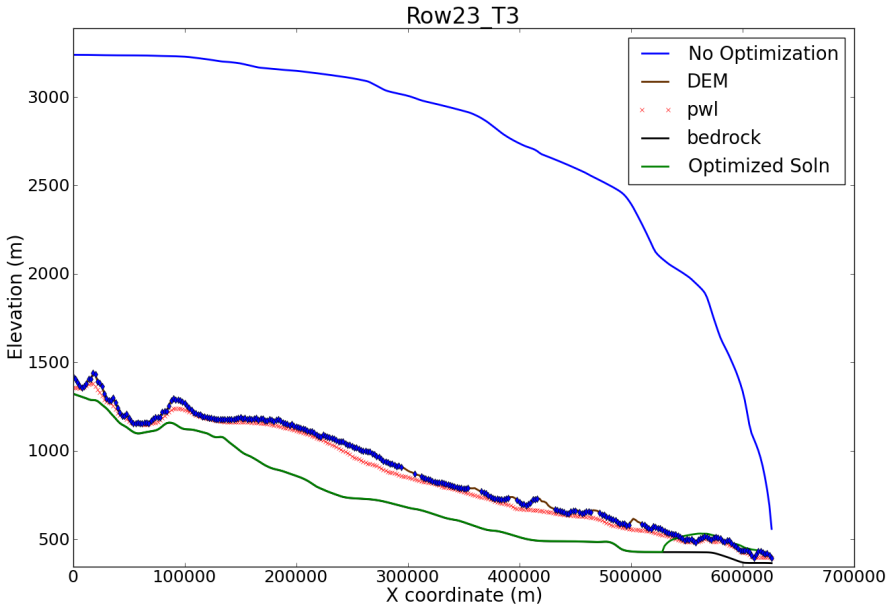

Figure 5.11: *PSO Transect Results with Population of 1000*

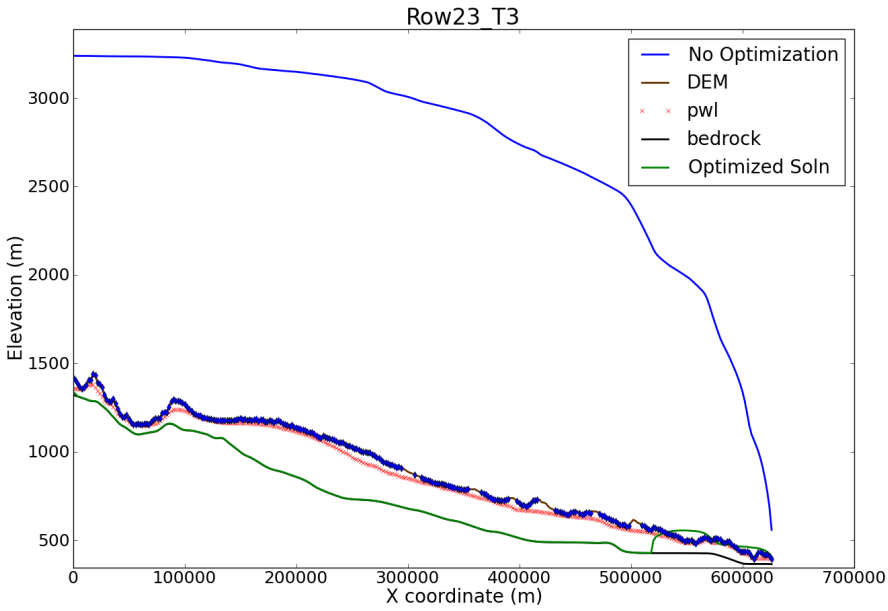

Figure 5.12: *PSO Transect Results with Population of 2000*

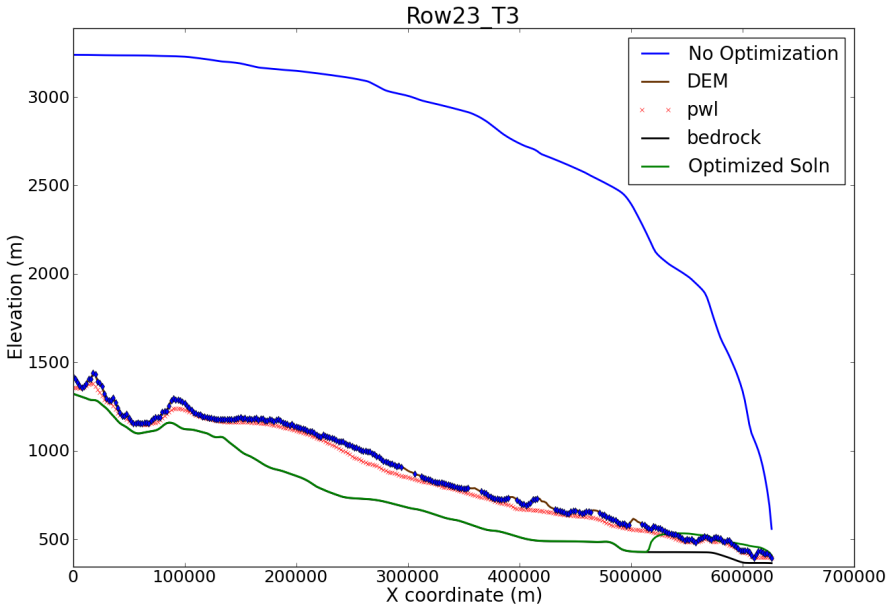

Figure 5.13: *PSO Transect Results with Population of 2500*

Increasing the maximum number of iterations was also tried. Here the objective function of the best particle and the worst particle in the swarm were plotted to see if an increased number of iterations would improve convergence of the solution. Figures 5.14-5.17 show the results. A population size of 350 was used for all figures. This increase also had almost no effect on the final outcome of the transect and the results shown here were typical for other transects tested as well. It can be seen in the objective function plots that the particles converge quite early in the optimization routine. This was true of all the Nebraska transects tested. The failure of these attempts to improve the PSO results in Nebraska point to the likelihood that the parameter space is too complex for the PSO routine to handle with a reasonable population size and maximum number of iterations.

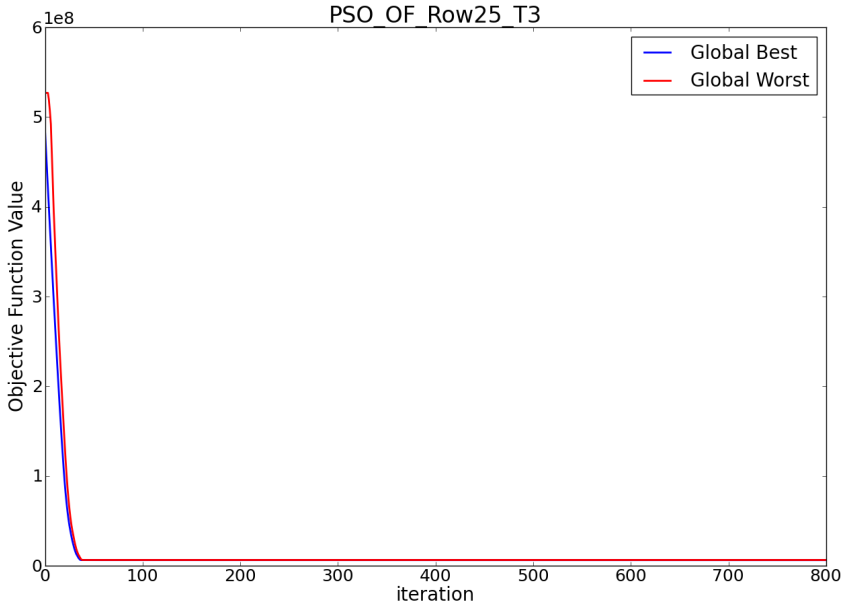

Figure 5.14: *PSO Transect Objective Function with Maximum Iterations of 800*

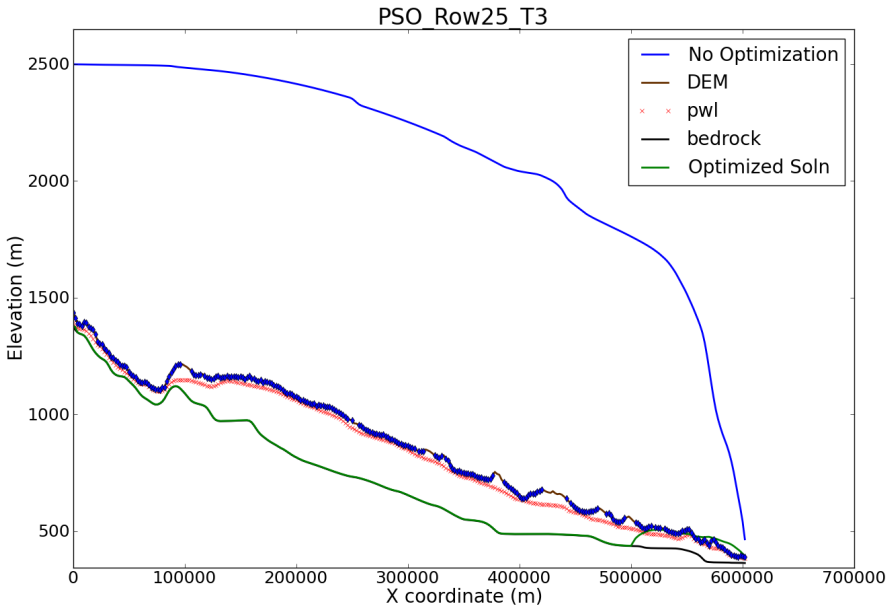

Figure 5.15: *PSO Transect Results with Maximum Iterations of 800*

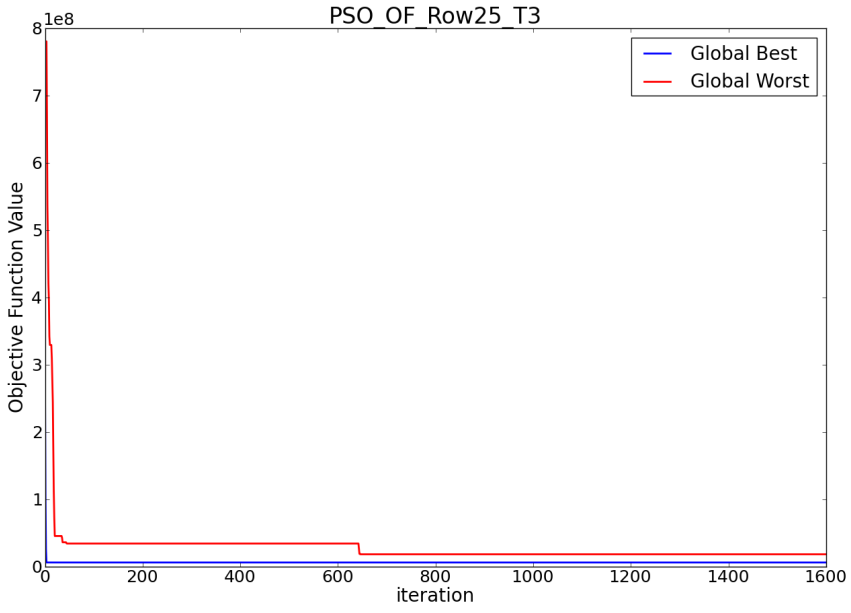

Figure 5.16: *PSO Transect Objective Function with Maximum Iterations of 1600*

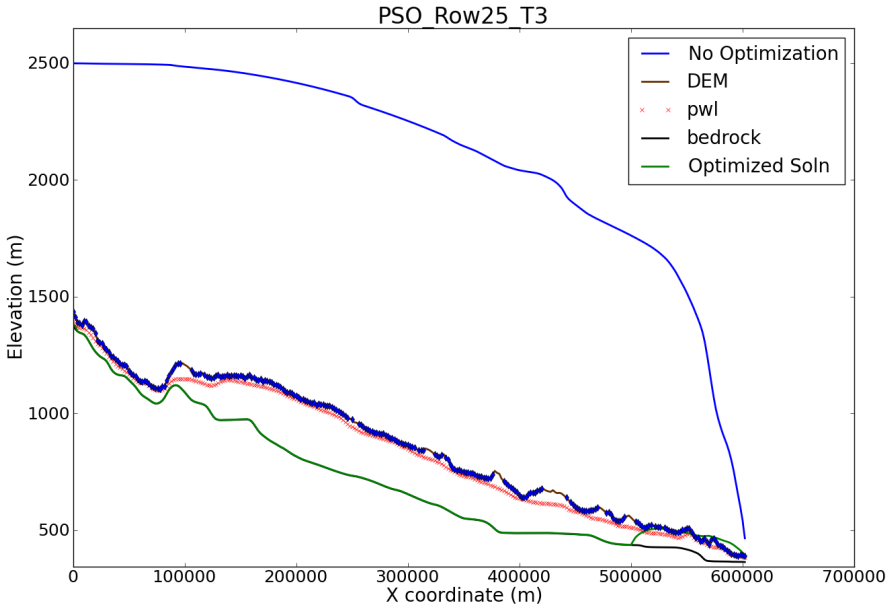

Figure 5.17: *PSO Transect Results with Maximum Iterations of 1600*

5.1.3 Levenberg-Marquardt Results (2km x 10km grid cells)

The PSO routine failed to find a good solution in Nebraska likely due to an overly complex parameter space. The Levenberg-Marquardt algorithm showed the ability to minimize the objective function when using a very large number of parameters (figure 5.2). Because the Levenberg-Marquardt algorithm showed a strength in finding solutions that minimize the objective function when there are a large number of parameters involved it was used again to run simulations on the 2km x 10km grid to see if it could find a good solution for the aquifer in Nebraska. Figures 5.18-5.20 show the results obtained from these simulations.

The Levenberg-Marquardt results were not much better than the PSO results. There are a couple of long transects with errors of ±6m but the majority of the results in Nebraska are very poor. These results were obtained by using an initial guess of $0.2\text{m}^2/\text{day}$. Because the Levenberg-Marquardt method depends so heavily on the initial guess it is possible that further experimentation with different initial guesses could yield improved results.

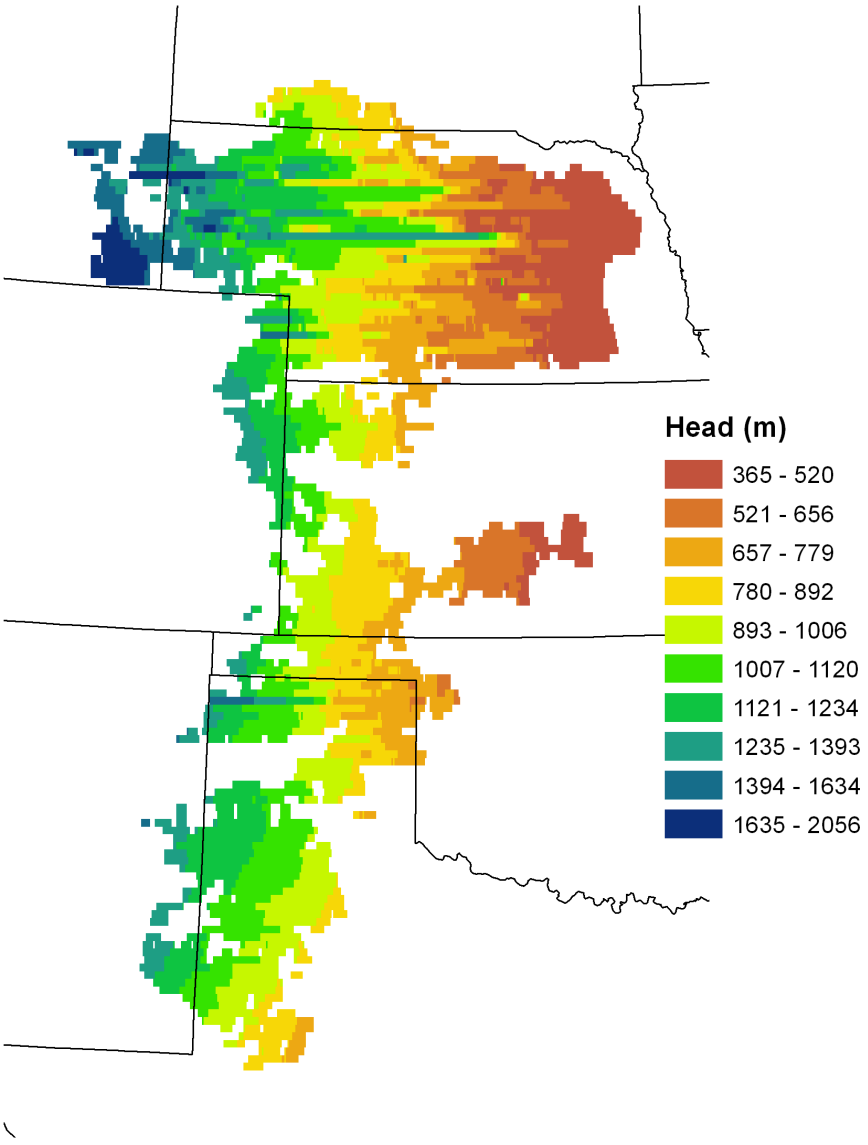

Figure 5.18: *Sloping Base Levenberg-Marquardt Optimization Results: Heads (2km by 10km grid cells, DTW_{max}=10m)*

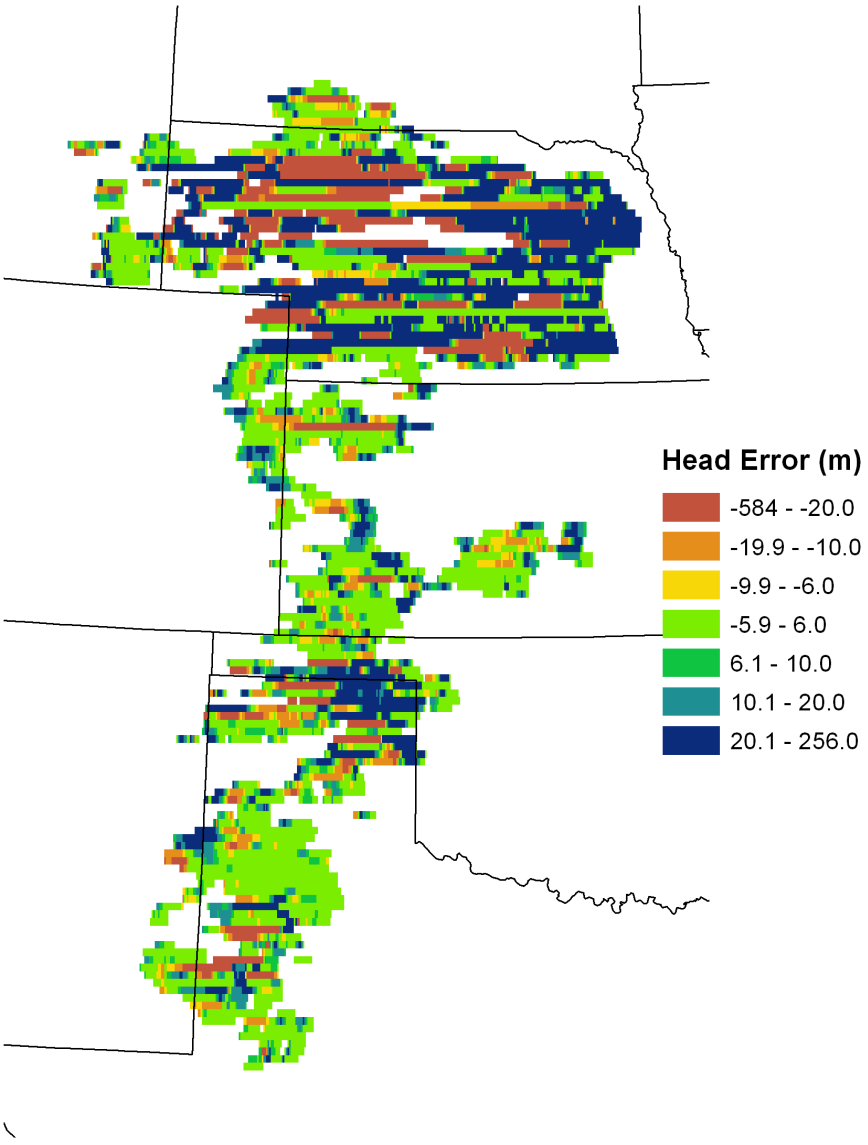

Figure 5.19: *Sloping Base Levenberg-Marquardt Optimization Results: Simulated Head Error (2km by 10km grid cells, DTW_{max}=10m)*

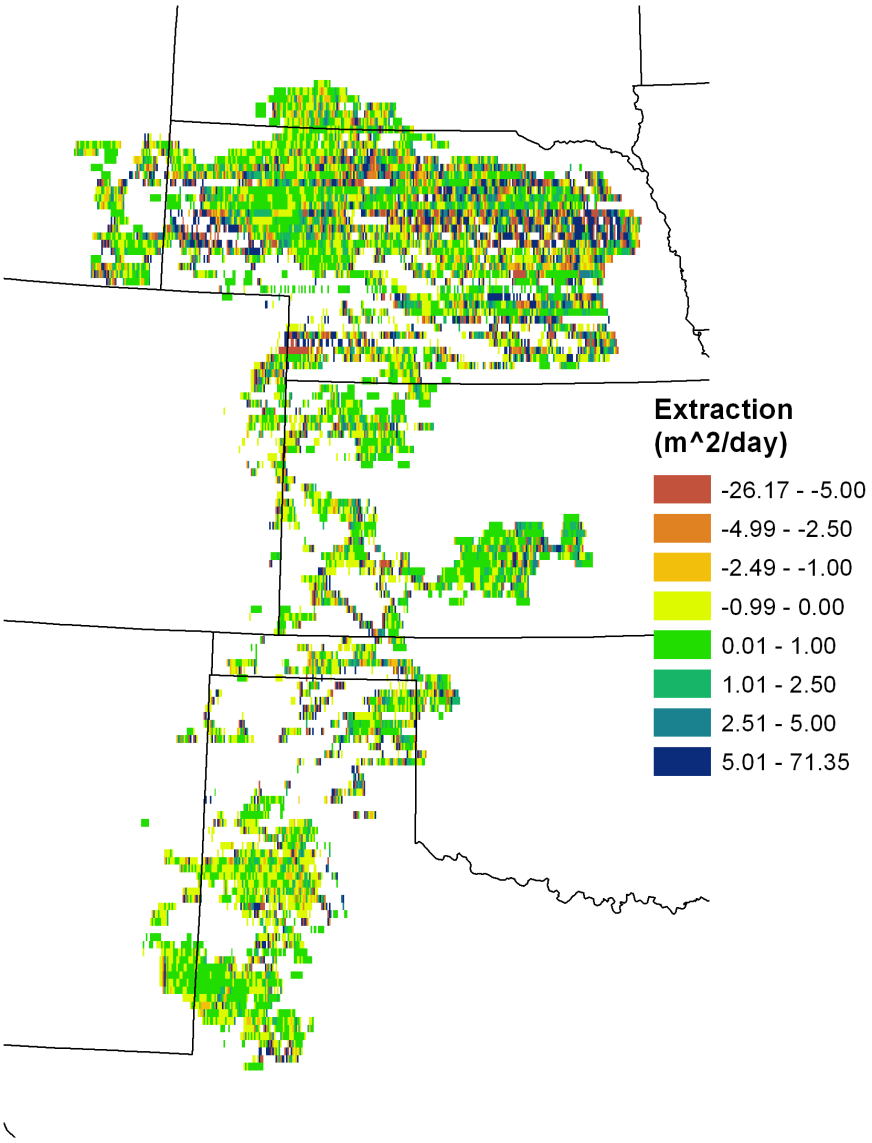

Figure 5.20: *Sloping Base Levenberg-Marquardt Optimization Results: Interaction Rates (2km by 10km grid cells)*

5.1.4 Analysis of Groundwater-Surface Water Interaction

The PSO results shown in figures 5.8-5.9 and the Levenberg-Marquardt results shown in figures 5.19-5.20 were analyzed to determine the amount of groundwater being discharged to rivers (baseflow). There are many rivers and streams in the High Plains Aquifer so only the Canadian River, Cimarron River, Arkansas River, Republican River, and Platte River were considered in this analysis (figure 1.2). The sum of the interaction rates obtained from the optimized sloping base models were summed along each river to determine the total contribution to stream flow. Historical stream gauge data (figure 5.21) was obtained from the USGS and the percentage of streamflow derived from groundwater discharge (baseflow) is presented for each river. The percentages were calculated based on the monthly average stream flows obtained from the USGS.

All rivers except the Canadian had multiple gauging stations. This chapter only presents the cumulative results at the final gauging station for each river. Results for segments of rivers up to gauging stations other than the final gauging station are presented in Appendix D. Discharges were computed for the northern, central, and southern basins of the High Plains Aquifer and presented in this section along with the model error for these basins.

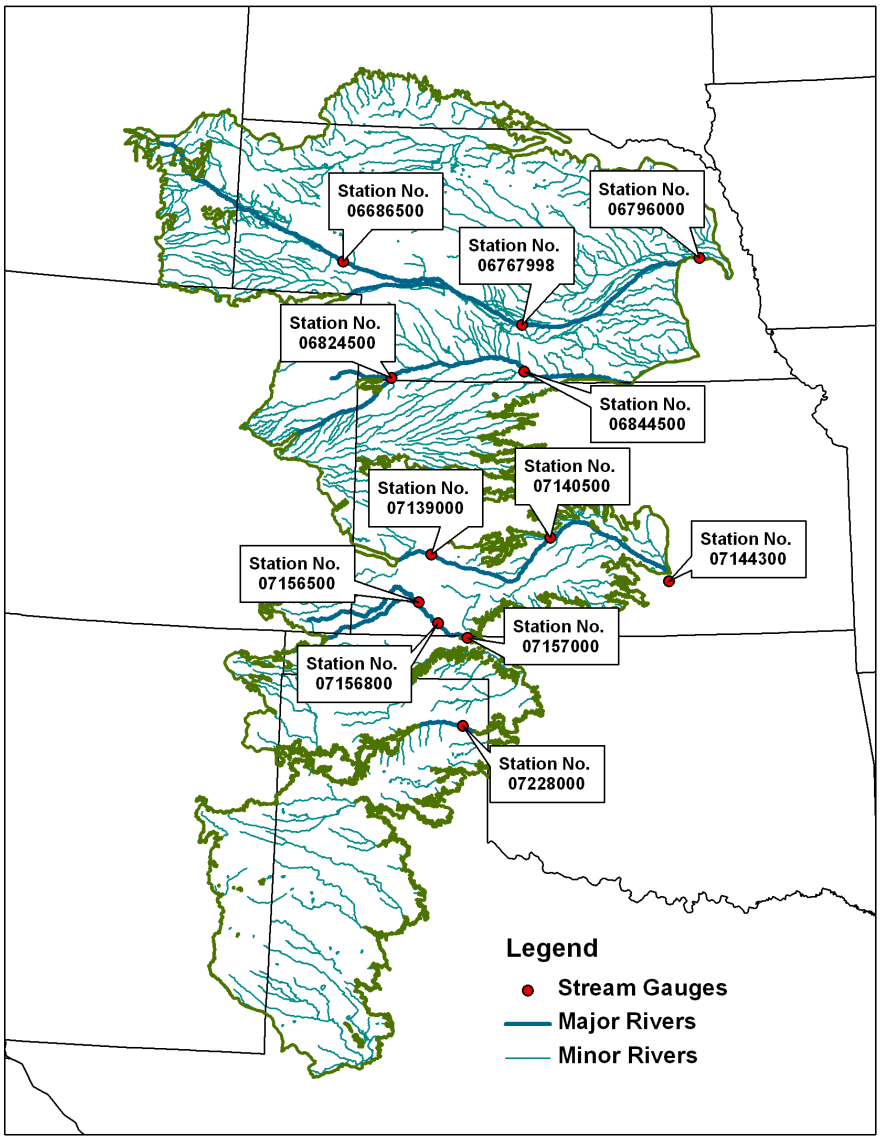

Figure 5.21: *Rivers of the High Plains Aquifer and USGS Stream Gauges*

Canadian River

Table 5.3 summarizes the results from the sloping base model. Monthly stream gauge data is shown in Appendix D and table 5.4 shows the percentage of the streamflow that comes from groundwater discharge. Figures 5.22 & 5.23 show the maps of the groundwater-surface water interaction.

Table 5.3: *Simulated Groundwater-Surface Water Interaction along Canadian River up to Gauging Station 07228000*

PSO Results		L-M Results	
No. of Interaction Points	30	No. of Interaction Points	30
Cumulative Contribution to Baseflow (cfs)	19.6	Cumulative Contribution to Baseflow (cfs)	29.0
Average Interaction Rate per 2km Cell (cfs)	0.65	Average Interaction Rate per 2km Cell (cfs)	0.98

Table 5.4: *Simulated Groundwater-Surface Water Interaction along Canadian River up to Gauging Station 07228000 (Mean Discharge from 1960-1980)*

	Canadian River Baseflow											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	78.0	86.4	136.9	107.9	228.1	148.9	25.3	25.6	53.0	34.8	101.2	77.5
% Derived from Groundwater Discharge (PSO)	25%	23%	14%	18%	9%	13%	77%	77%	37%	56%	19%	25%
% Derived from Groundwater Discharge (L-M)	37%	34%	21%	27%	13%	20%	115%	114%	55%	83%	29%	37%

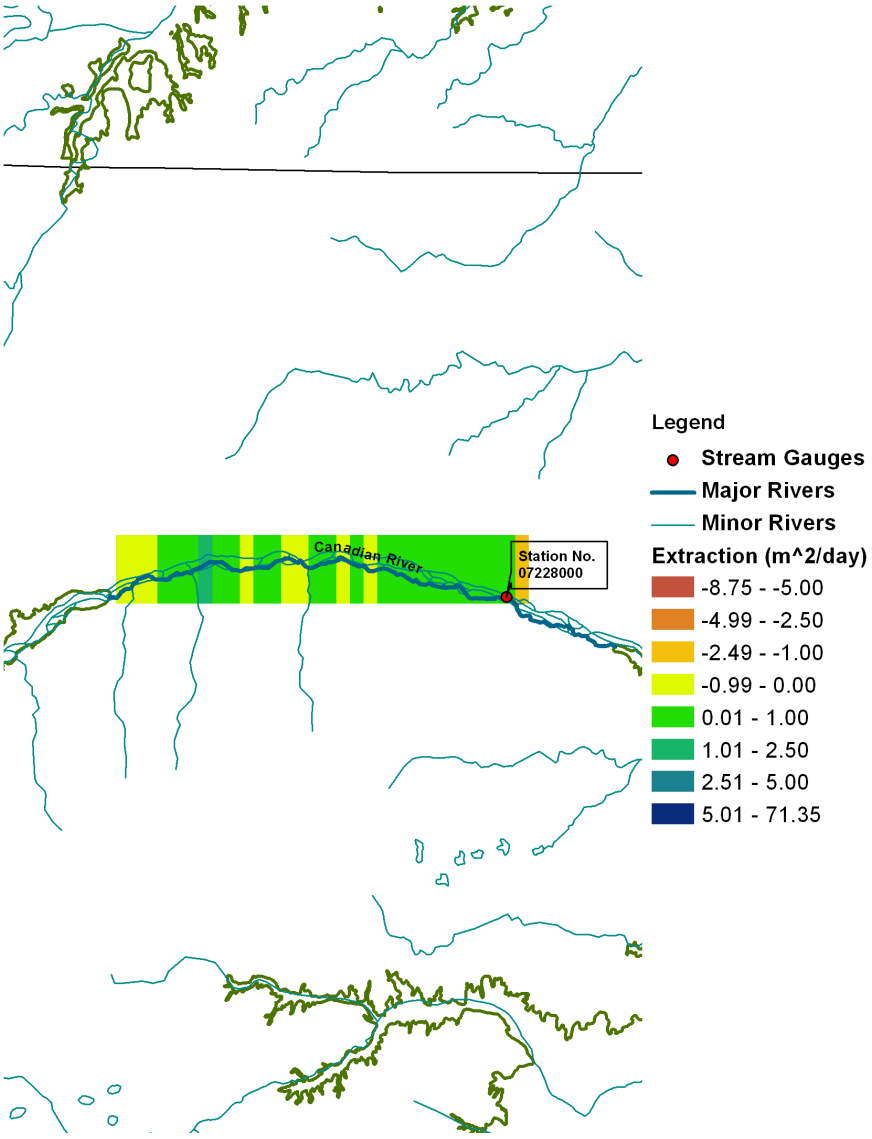

Figure 5.22: *Groundwater-Surface Water Interaction Along the Canadian River (PSO results)*

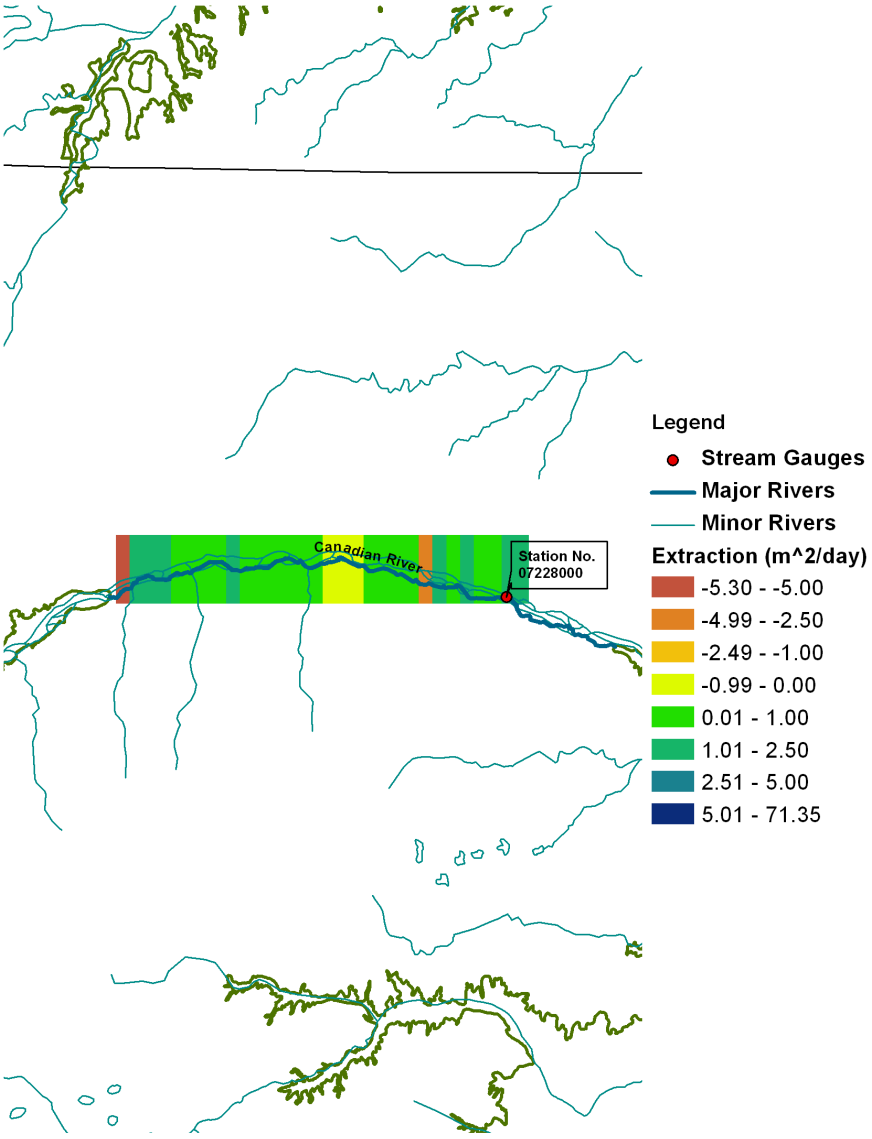

Figure 5.23: *Groundwater-Surface Water Interaction Along the Canadian River (L-M results)*

Cimarron River

Table 5.5 summarizes the results obtained from the sloping base model. Monthly stream gauge data is shown in Appendix D and table 5.6 shows the percentage of the streamflow that comes from groundwater discharge. Figures 5.24 & 5.25 show the maps of the groundwater-surface water interaction.

Table 5.5: *Simulated Groundwater-Surface Water Interaction along Cimarron River up to Gauging Station 07157000*

PSO Results	
No. of Interaction Points	138
Cumulative Contribution to Baseflow (cfs)	124
Average Interaction Rate per 2km Cell (cfs)	0.9

L-M Results	
No. of Interaction Points	115
Cumulative Contribution to Baseflow (cfs)	577
Average Interaction Rate per 2km Cell (cfs)	5.0

Table 5.6: *Simulated Groundwater-Surface Water Interaction along Cimarron River up to Gauging Station 07157000 (Mean Discharge from 1942-1965)*

	Cimarron River Baseflow (up to station 07157000)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	77.0	83.0	77.0	85.0	187.0	156.0	107.0	123.0	76.0	84.0	78.0	75.0
% Derived from Groundwater Discharge (PSO)	160%	149%	160%	145%	66%	79%	115%	100%	163%	147%	158%	165%
% Derived from Groundwater Discharge (L-M)	749%	695%	749%	678%	308%	370%	539%	469%	759%	687%	739%	769%

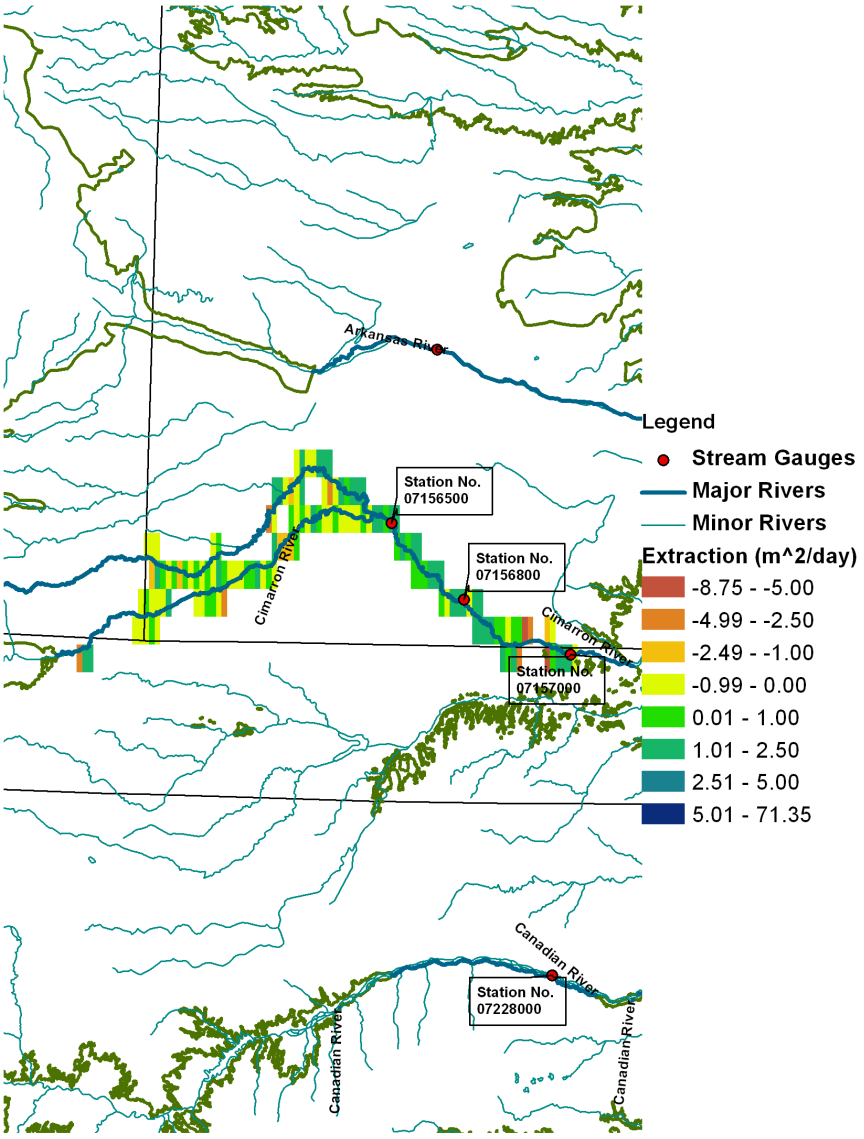

Figure 5.24: *Groundwater-Surface Water Interaction Along the Cimarron River (PSO results)*

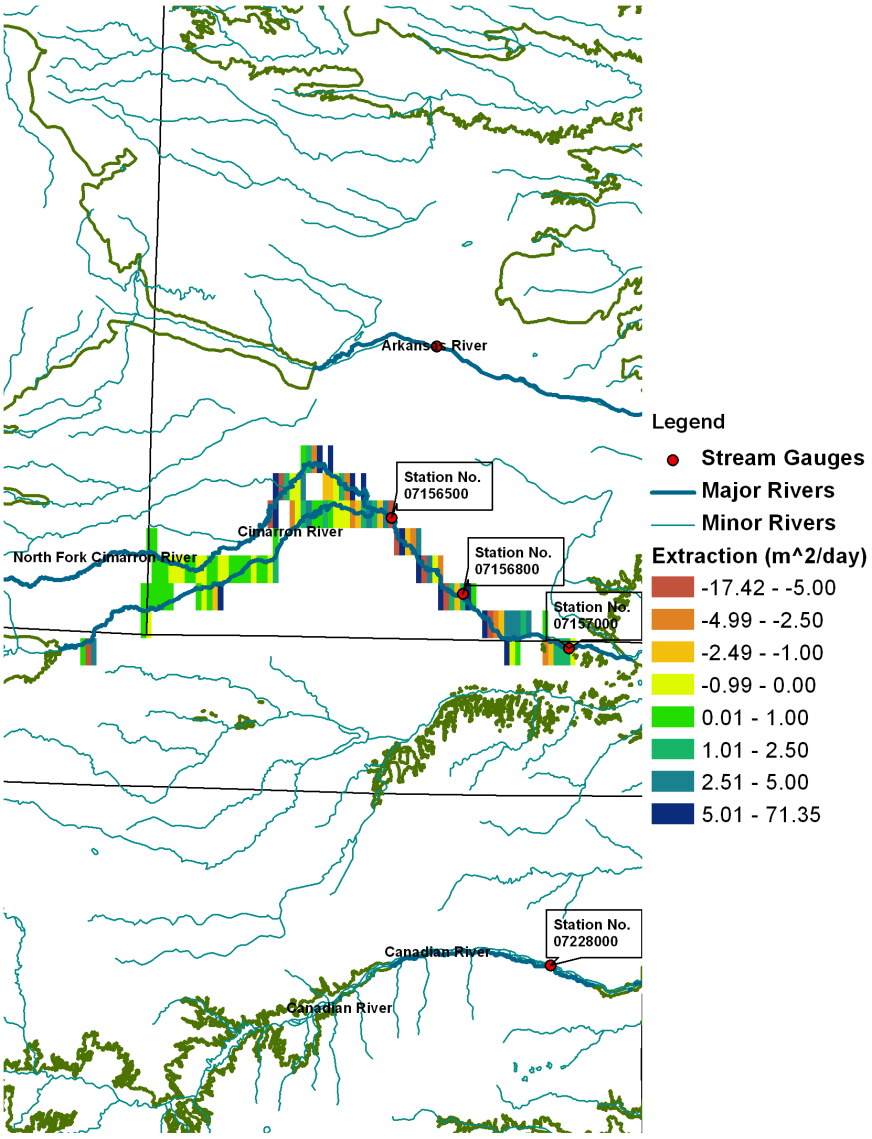

Figure 5.25: *Groundwater-Surface Water Interaction Along the Cimarron River (L-M results)*

Arkansas River

Table 5.7 summarizes the results from the sloping base model. Monthly stream gauge data is shown in Appendix D and table 5.8 shows the percentage of the streamflow that comes from groundwater discharge. Figures 5.26 & 5.27 show the maps of the groundwater-surface water interaction.

Table 5.7: *Simulated Groundwater-Surface Water Interaction along Arkansas River up to Gauging Station 07144300*

PSO Results	
No. of Interaction Points	217
Cumulative Contribution to Baseflow (cfs)	324
Average Interaction Rate per 2km Cell (cfs)	1.47

L-M Results	
No. of Interaction Points	192
Cumulative Contribution to Baseflow (cfs)	309
Average Interaction Rate per 2km Cell (cfs)	1.6

Table 5.8: *Simulated Groundwater-Surface Water Interaction along Arkansas River up to Gauging Station 07144300 (Mean Discharge from 1934-1950)*

	Arkansas River Baseflow (up to station 07144300)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	599	860	1187	1519	2326	2267	2396	1452	1077	979	522	549
% Derived from Groundwater Discharge (PSO)	54%	38%	27%	21%	14%	14%	14%	22%	30%	33%	62%	59%
% Derived from Groundwater Discharge (L-M)	52%	36%	26%	20%	13%	14%	13%	21%	29%	32%	59%	56%

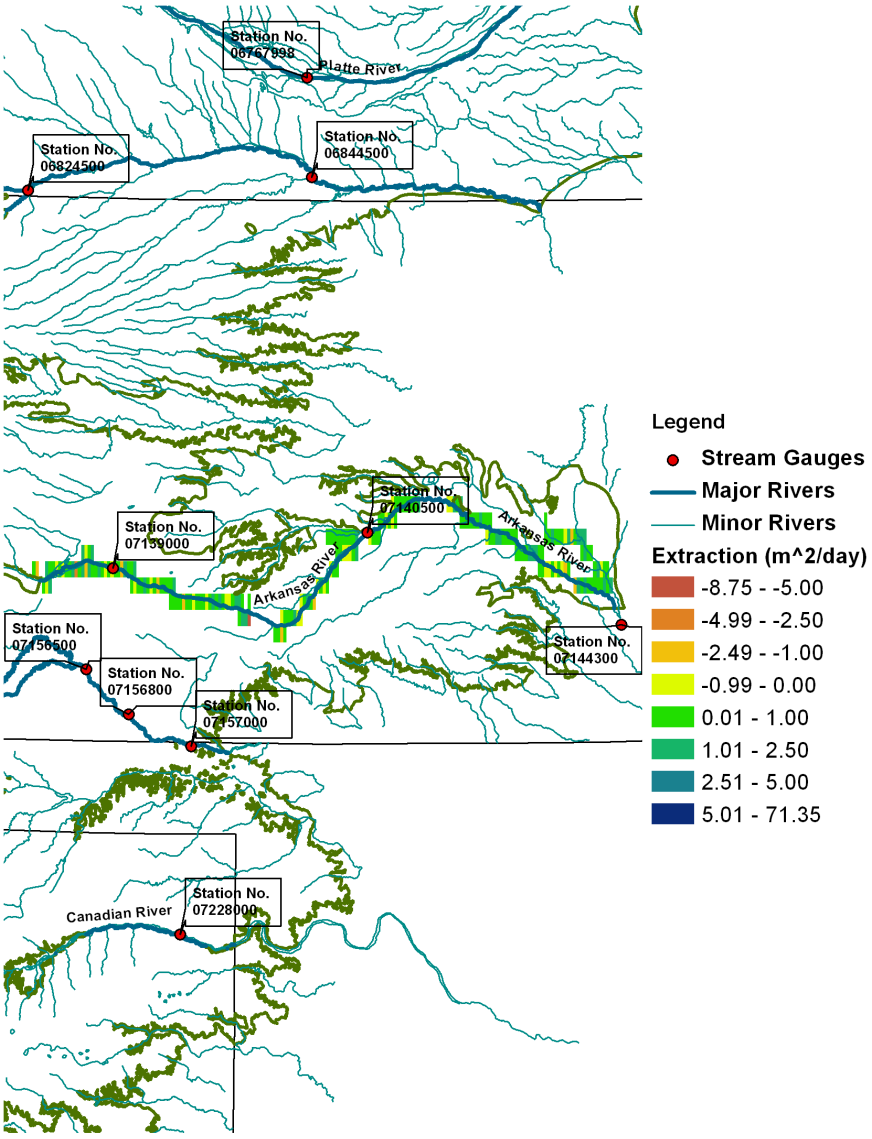

Figure 5.26: *Groundwater-Surface Water Interaction Along the Arkansas River (PSO results)*

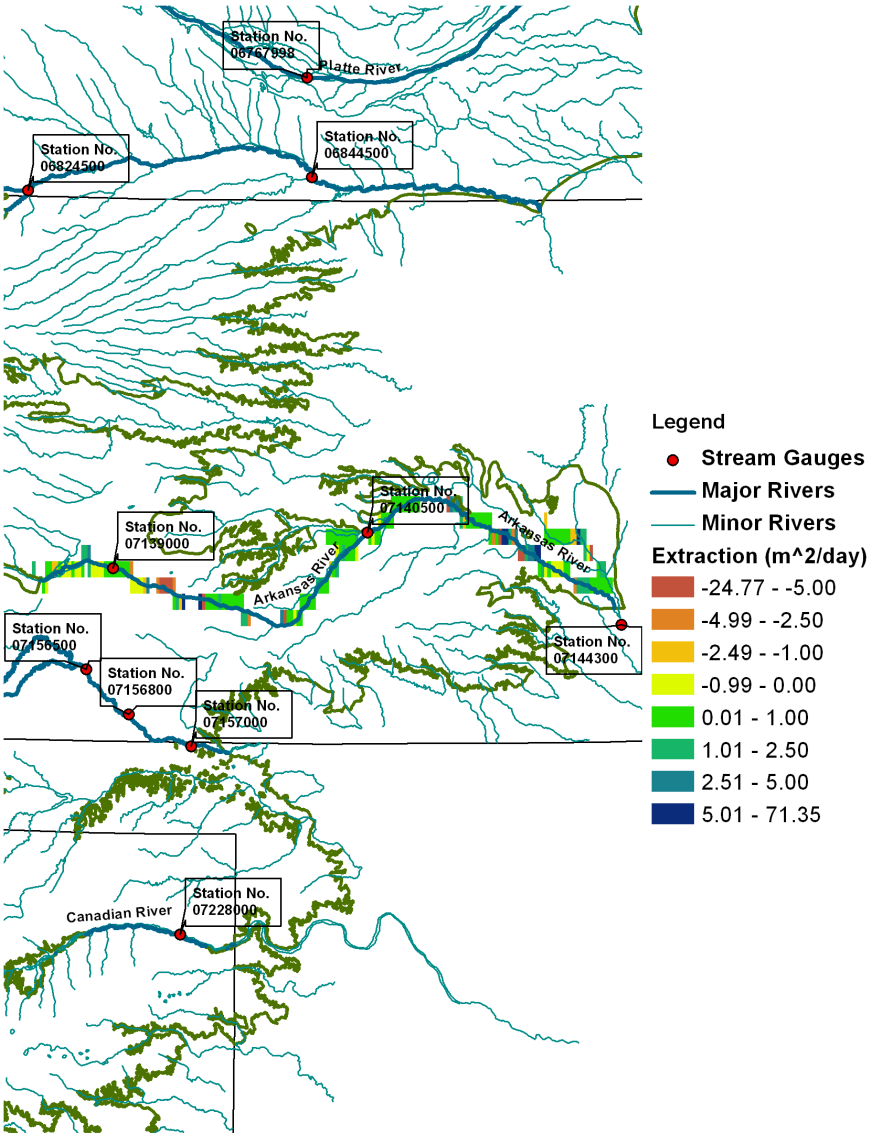

Figure 5.27: *Groundwater-Surface Water Interaction Along the Arkansas River (L-M results)*

Republican River

Table 5.9 summarizes the results from the sloping base model. Monthly stream gauge data is shown in Appendix D and table 5.10 shows the percentage of the streamflow that comes from groundwater discharge. Figures 5.28 & 5.29 show the maps of the groundwater-surface water interaction.

Table 5.9: *Simulated Groundwater-Surface Water Interaction along Republican River up to Gauging Station 06844500*

PSO Results	
No. of Interaction Points	90
Cumulative Contribution to Baseflow (cfs)	72.4
Average Interaction Rate per 2km Cell (cfs)	0.82

L-M Results	
No. of Interaction Points	72
Cumulative Contribution to Baseflow (cfs)	84.3
Average Interaction Rate per 2km Cell (cfs)	1.2

Table 5.10: *Simulated Groundwater-Surface Water Interaction along Republican River up to Gauging Station 06844500 (Mean Discharge from 1947-1967)*

	Republican River Baseflow (up to station 06844500)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	198	364	485	419	503	683	356	214	139	159	190	185
% Derived from Groundwater Discharge (PSO)	37%	20%	15%	17%	14%	11%	20%	34%	52%	46%	38%	39%
% Derived from Groundwater Discharge (L-M)	43%	23%	17%	20%	17%	12%	24%	39%	61%	53%	44%	46%

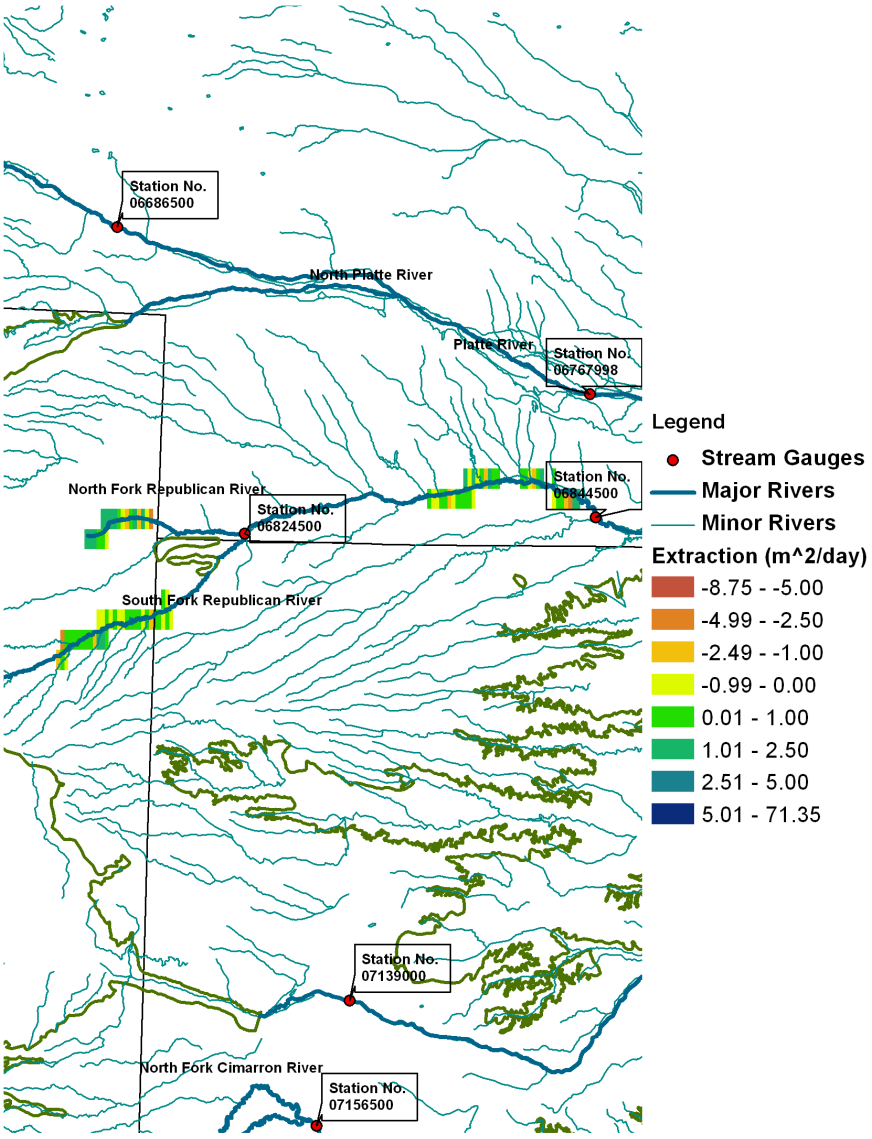

Figure 5.28: *Groundwater-Surface Water Interaction Along the Republican River (PSO results)*

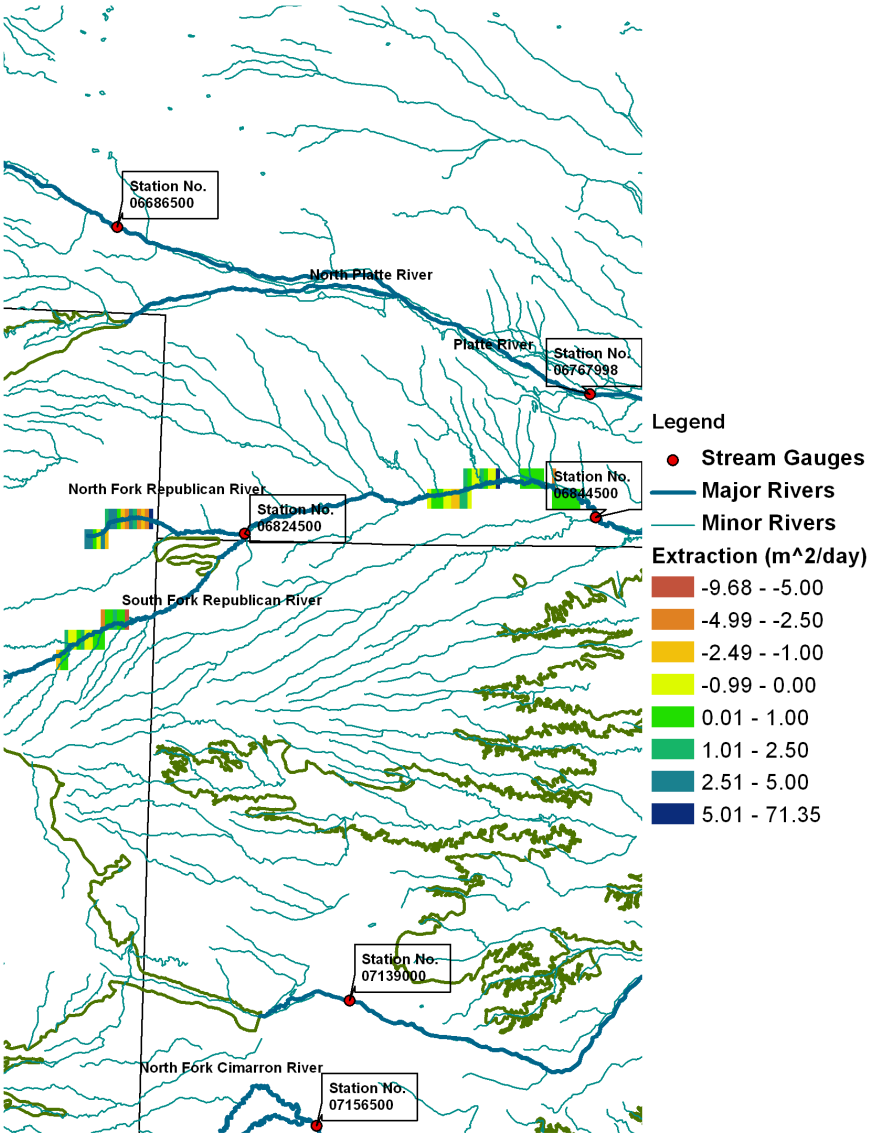

Figure 5.29: *Groundwater-Surface Water Interaction Along the Republican River (L-M results)*

Platte River

Table 5.11 summarizes the results from the sloping base model. Monthly stream gauge data is shown in Appendix D and table 5.12 shows the percentage of the streamflow that comes from groundwater discharge. Figures 5.30-5.35 show the maps of the groundwater-surface water interaction.

Table 5.11: *Simulated Groundwater-Surface Water Interaction along Platte River up to Gauging Station 06796000*

PSO Results	
No. of Interaction Points	431
Cumulative Contribution to Baseflow (cfs)	739
Average Interaction Rate per 2km Cell (cfs)	1.7

L-M Results	
No. of Interaction Points	216
Cumulative Contribution to Baseflow (cfs)	1145
Average Interaction Rate per 2km Cell (cfs)	5.3

Table 5.12: *Simulated Groundwater-Surface Water Interaction along Platte River up to Gauging Station 06796000 (Mean Discharge from 1949-1969)*

	Platte River Baseflow (up to station 06796000)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	2516	4113	6454	4927	4305	5823	3024	1927	2417	3144	3451	2746
% Derived from Groundwater Discharge (PSO)	29%	18%	11%	15%	17%	13%	24%	38%	31%	24%	21%	27%
% Derived from Groundwater Discharge (L-M)	46%	28%	18%	23%	27%	20%	38%	59%	47%	36%	33%	42%

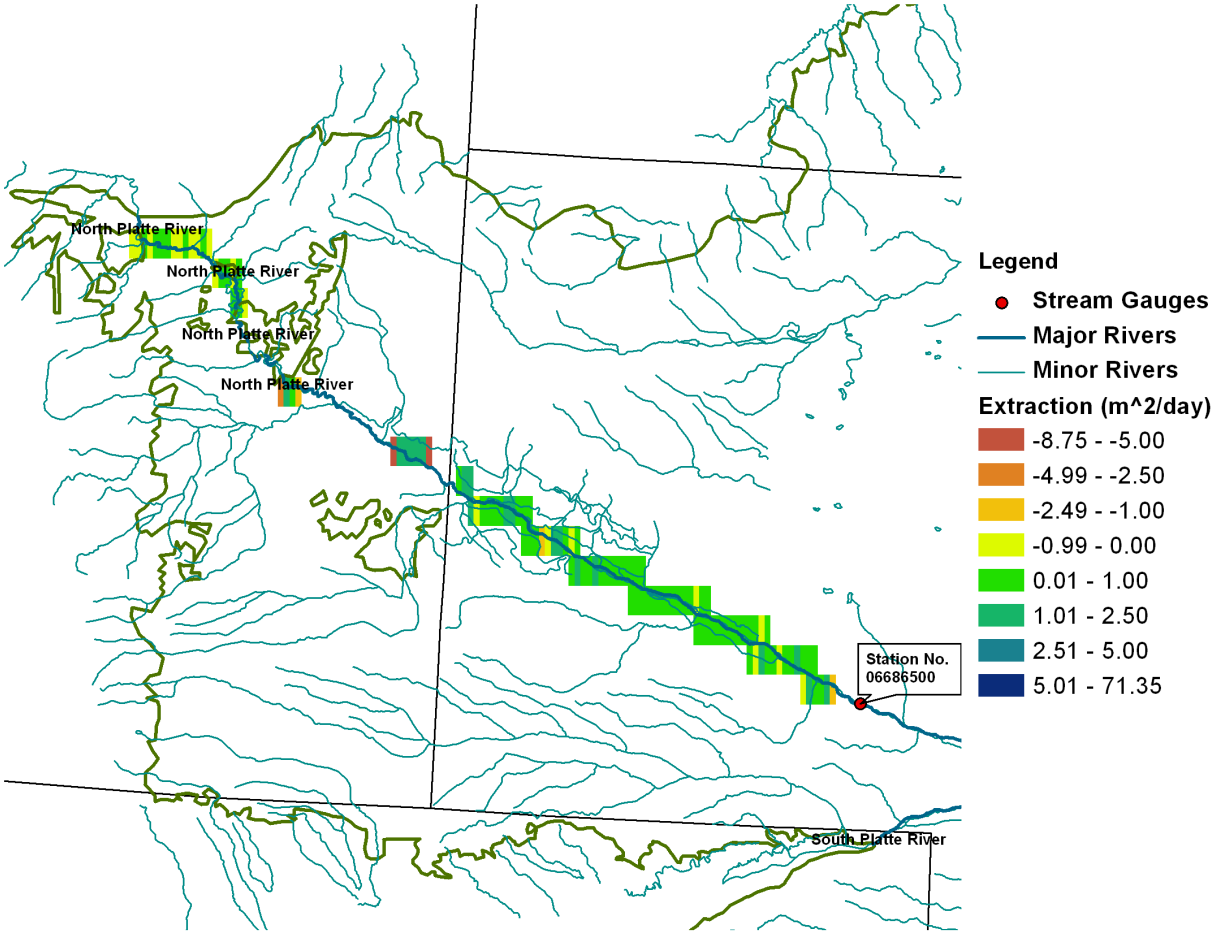

Figure 5.30: *Groundwater-Surface Water Interaction Along the Platte River: First Segment (PSO results)*

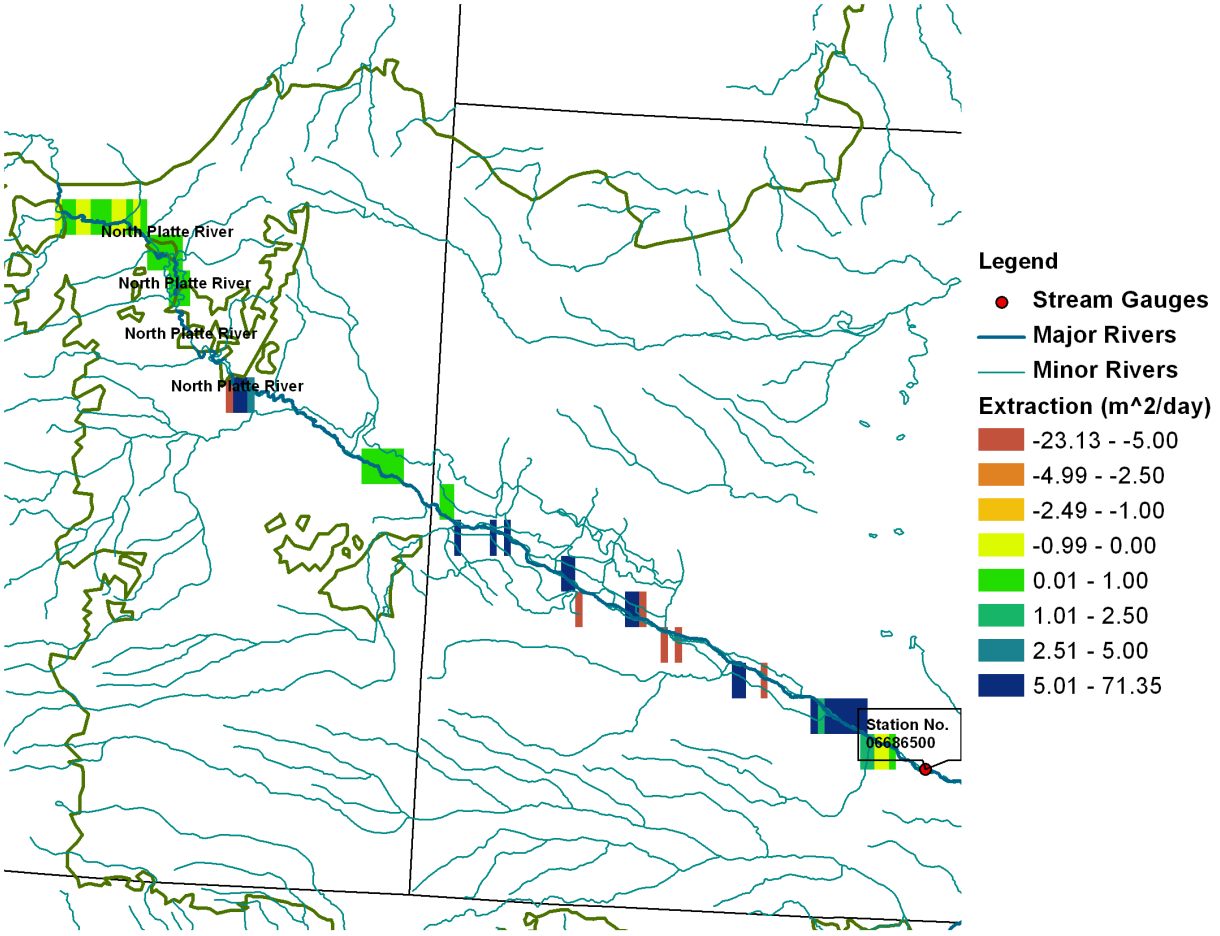

Figure 5.31: *Groundwater-Surface Water Interaction Along the Platte River: Second Segment (L-M results)*

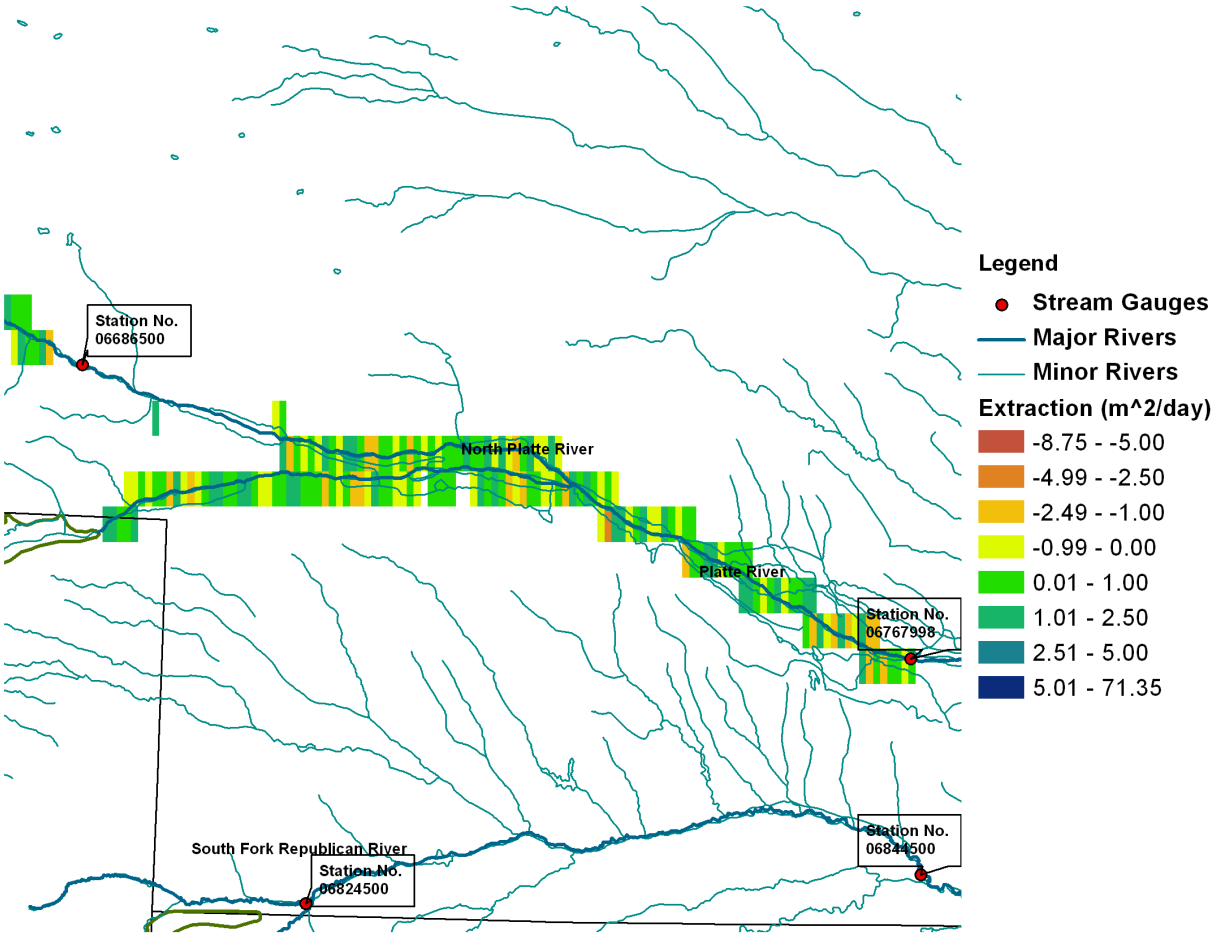

Figure 5.32: *Groundwater-Surface Water Interaction Along the Platte River: Second Segment (PSO results)*

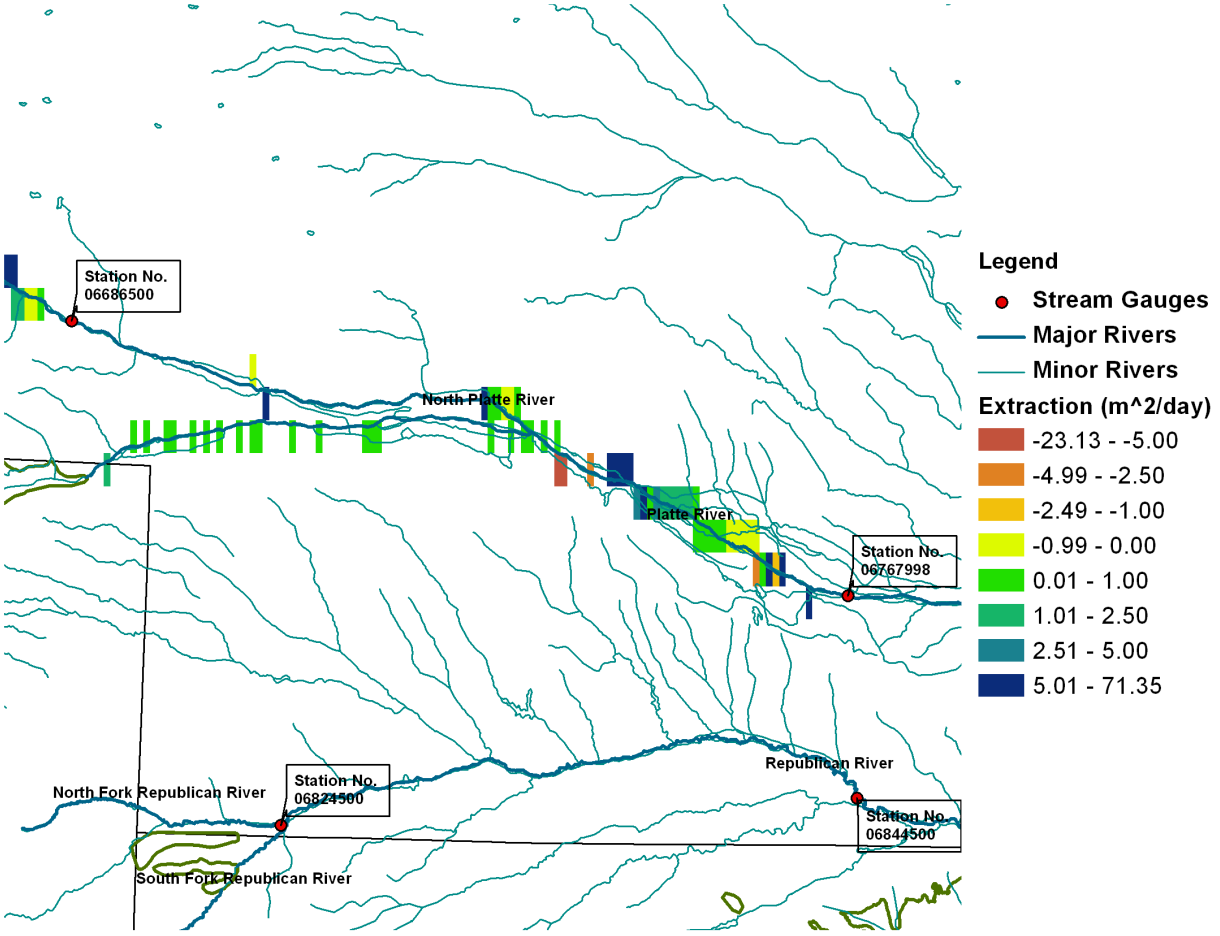

Figure 5.33: *Groundwater-Surface Water Interaction Along the Platte River: Second Segment (L-M results)*

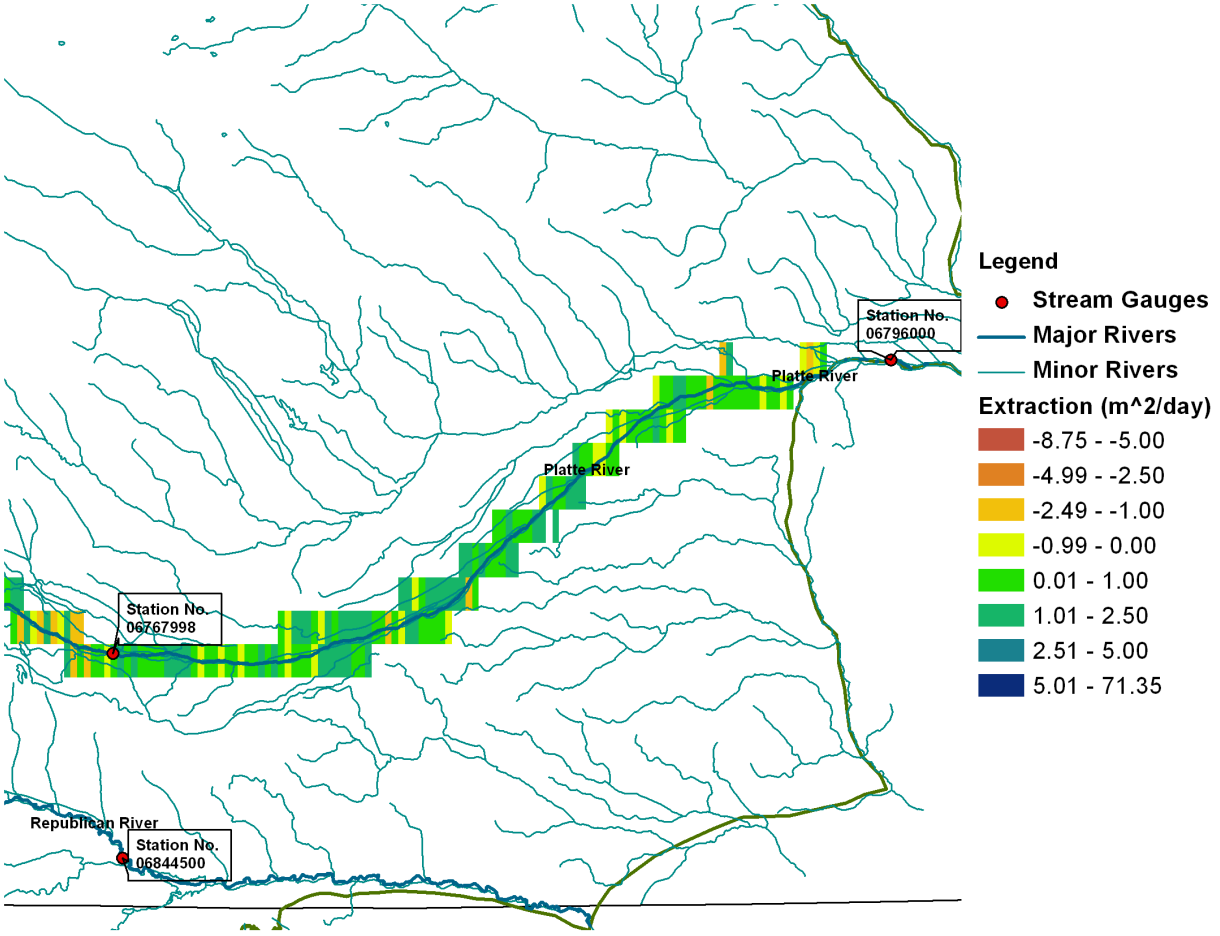

Figure 5.34: *Groundwater-Surface Water Interaction Along the Platte River: Third Segment (PSO results)*

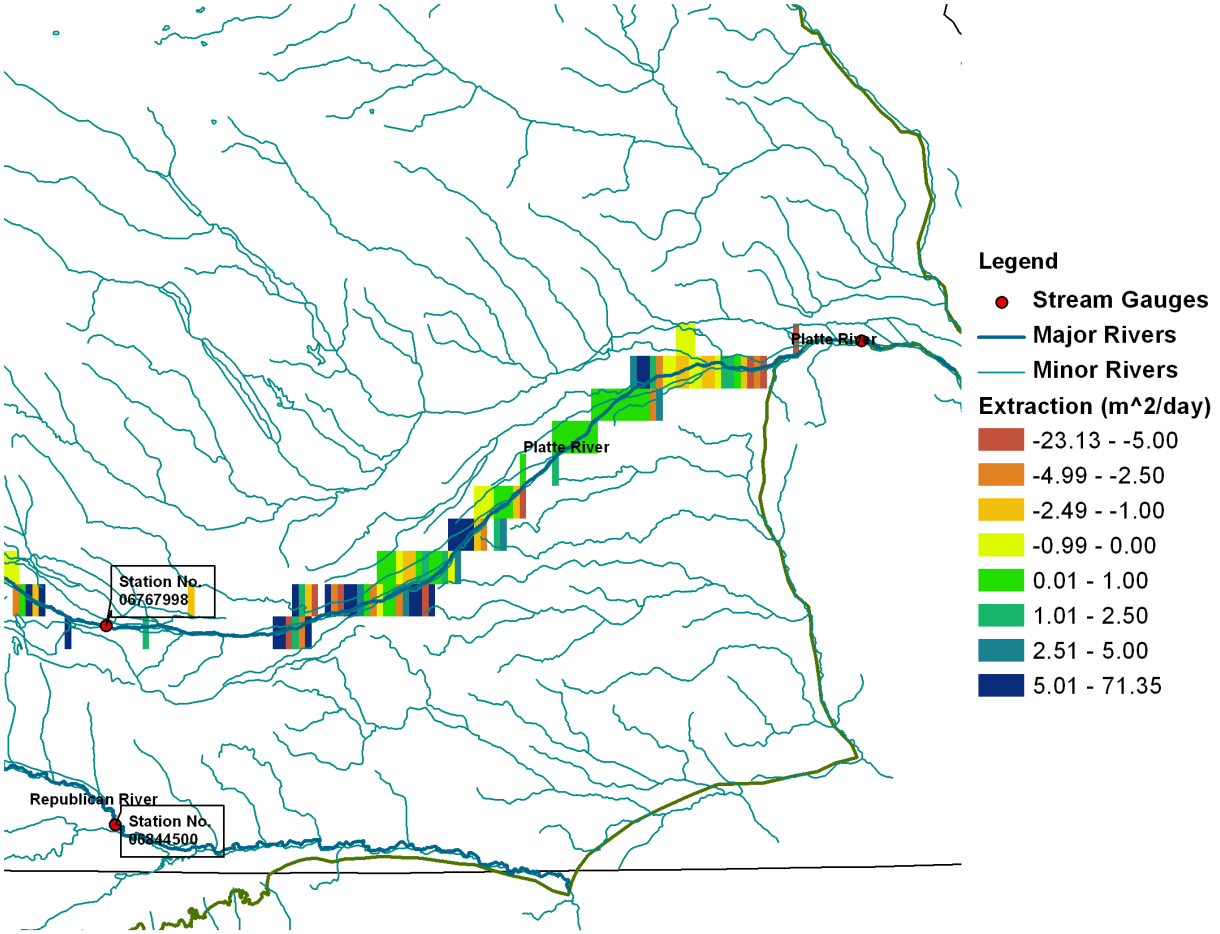

Figure 5.35: *Groundwater-Surface Water Interaction Along the Platte River: Third Segment (L-M results)*

Northern Basin (NE, WY, SD

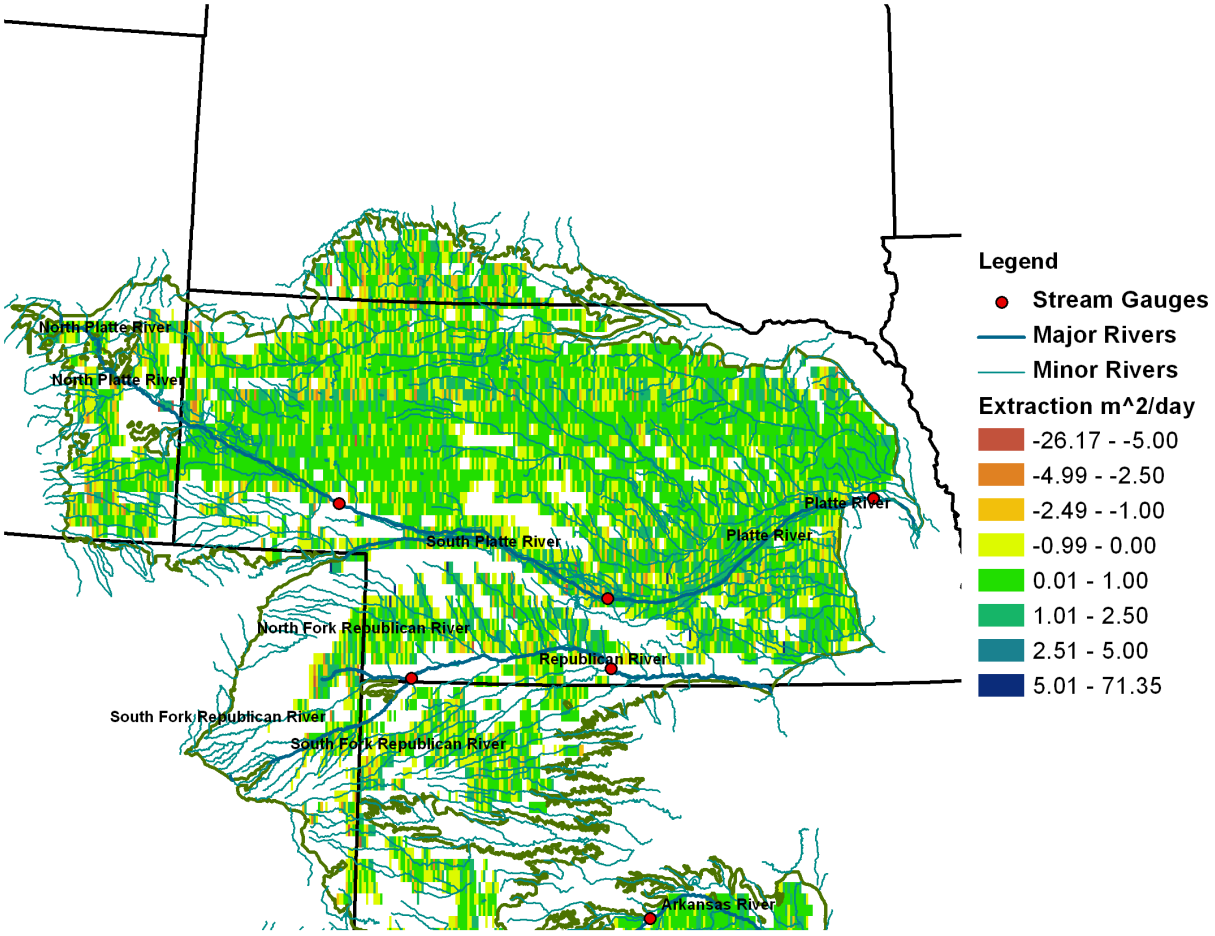

Figure 5.36: *Groundwater-Surface Water Interaction in the Northern Third of the High Plains Aquifer (PSO results)*

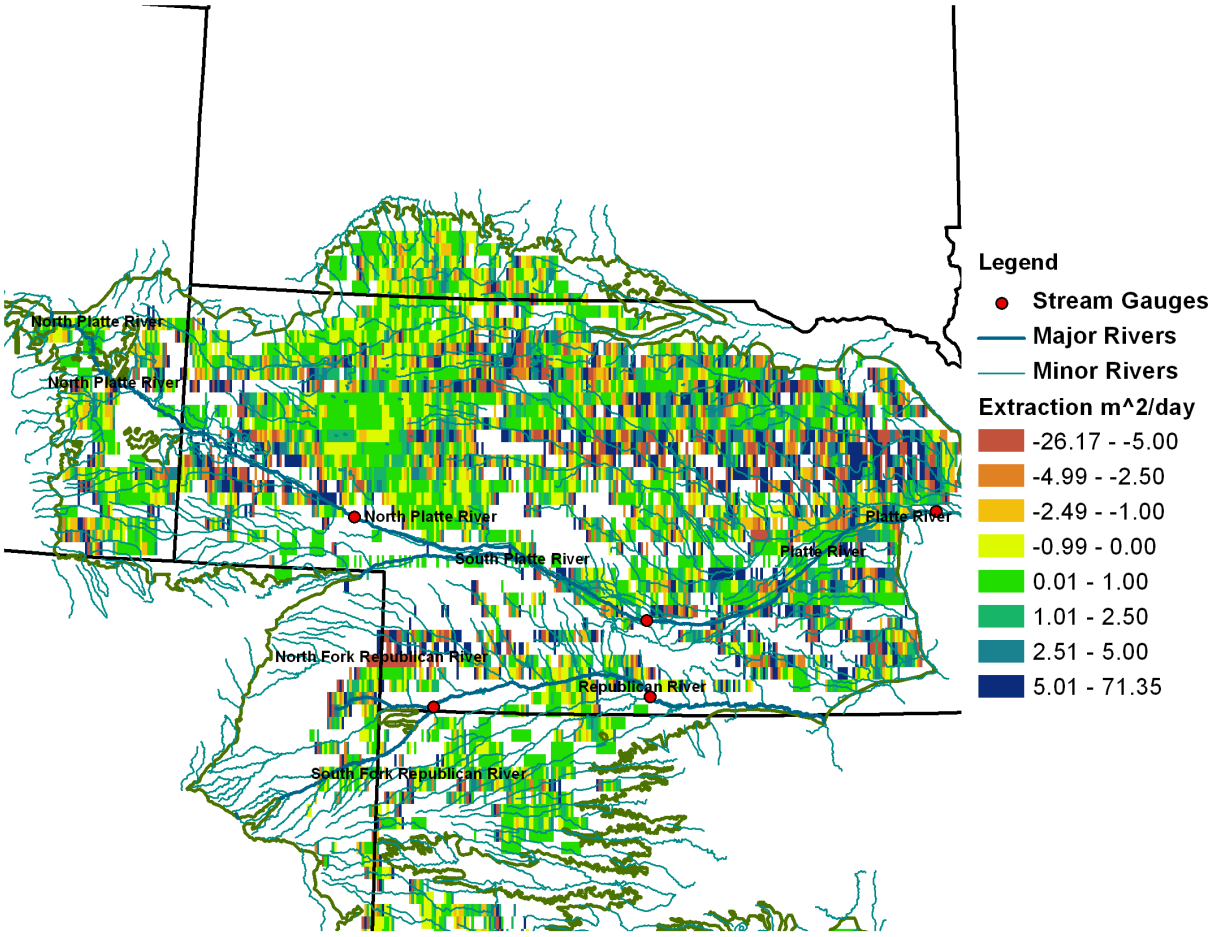

Figure 5.37: *Groundwater-Surface Water Interaction in the Northern Third of the High Plains Aquifer (L-M results)*

Table 5.13: *Regional Groundwater-Surface Water Interaction Summary for NE, WY, & SD*

PSO Results		L-M Results	
No. of Interaction Points	7531	No. of Interaction Points	6288
Cumulative Interaction Rate (cfs)	2429	Cumulative Interaction Rate (cfs)	3720
Average Interaction Rate per 2km Cell (cfs)	0.32	Average Interaction Rate per 2km Cell (cfs)	0.59

Table 5.14: *Regional Error in Simulated Head for NE, WY, & SD*

PSO Results		L-M Results	
Average Model Error (m)	40.5	Average Model Error (m)	-5.7
Error Std. Dev.	90.3	Error Std. Dev.	106.1

Central Basin (KS, CO, OK)

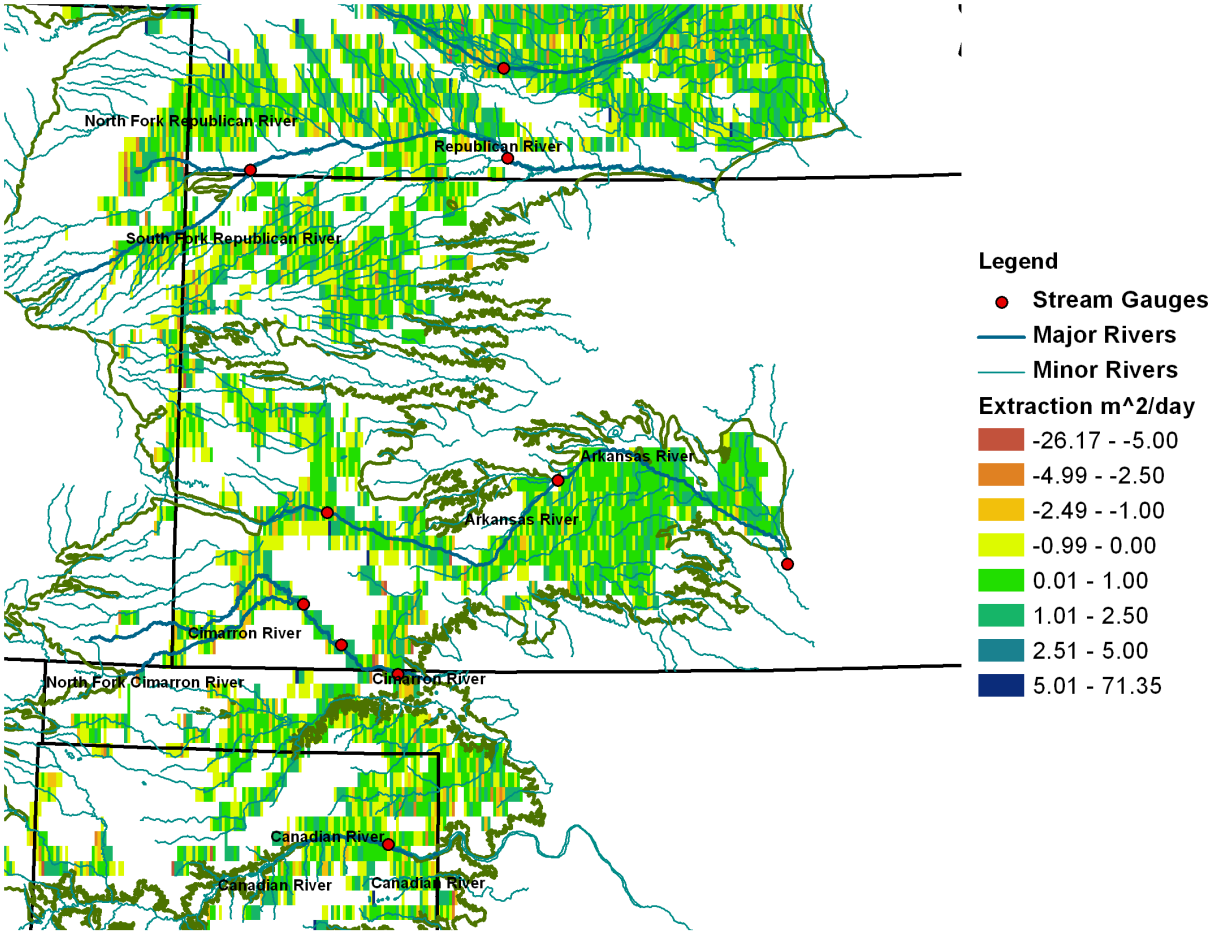

Figure 5.38: *Groundwater-Surface Water Interaction in the Central Third of the High Plains Aquifer (PSO results)*

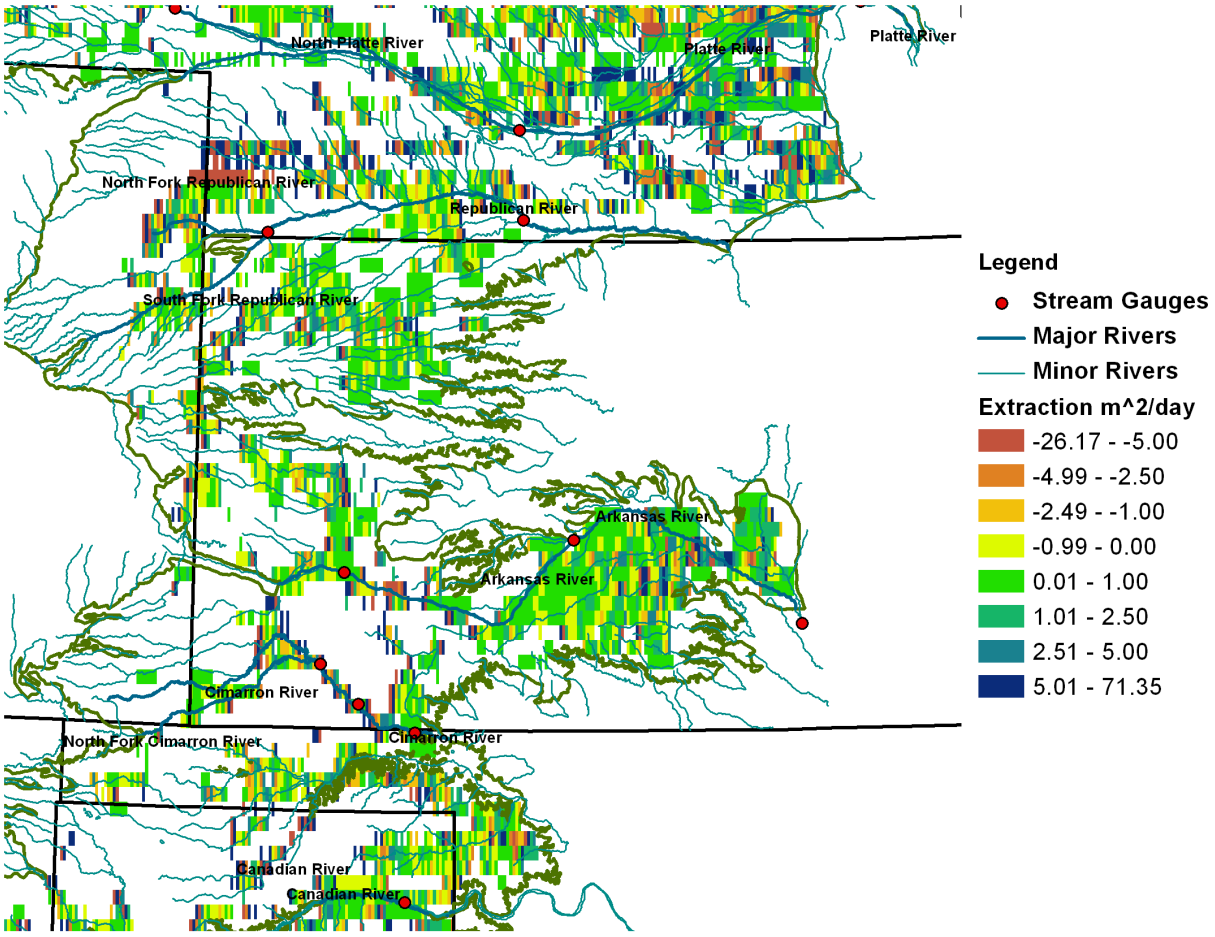

Figure 5.39: *Groundwater-Surface Water Interaction in the Central Third of the High Plains Aquifer (L-M results)*

Table 5.15: *Regional Groundwater-Surface Water Interaction Summary for KS, CO, & OK*

PSO Results	
No. of Interaction Points	2612
Cumulative Interaction Rate (cfs)	685
Average Interaction Rate per 2km Cell (cfs)	0.26

L-M Results	
No. of Interaction Points	2288
Cumulative Interaction Rate (cfs)	780
Average Interaction Rate per 2km Cell (cfs)	0.34

Table 5.16: *Regional Error in Simulated Head for KS, CO, & OK*

PSO Results	
Average Model Error (m)	0.04
Error Std. Dev.	8.6

L-M Results	
Average Model Error (m)	4.83
Error Std. Dev.	18.1

Southern Basin (TX & NM)

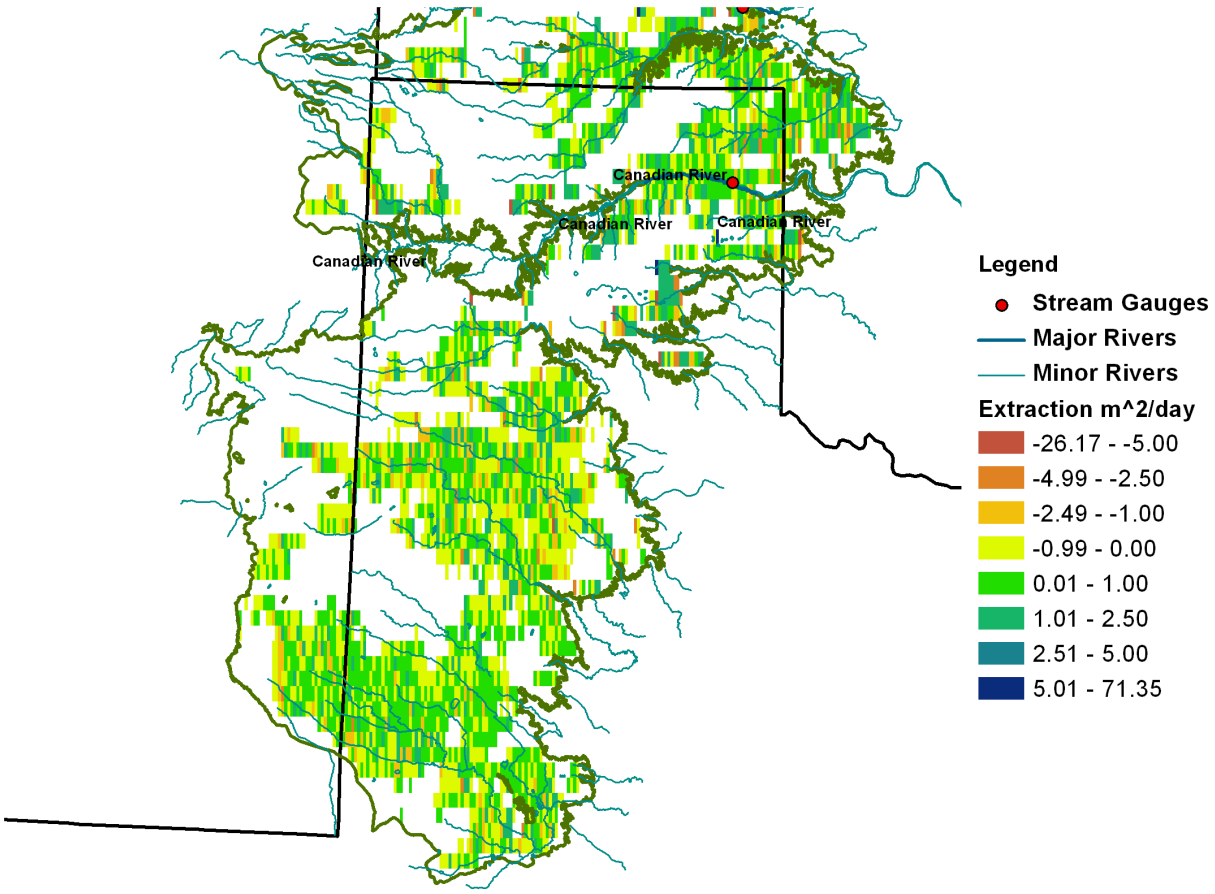

Figure 5.40: *Groundwater-Surface Water Interaction in the Southern Third of the High Plains Aquifer (PSO results)*

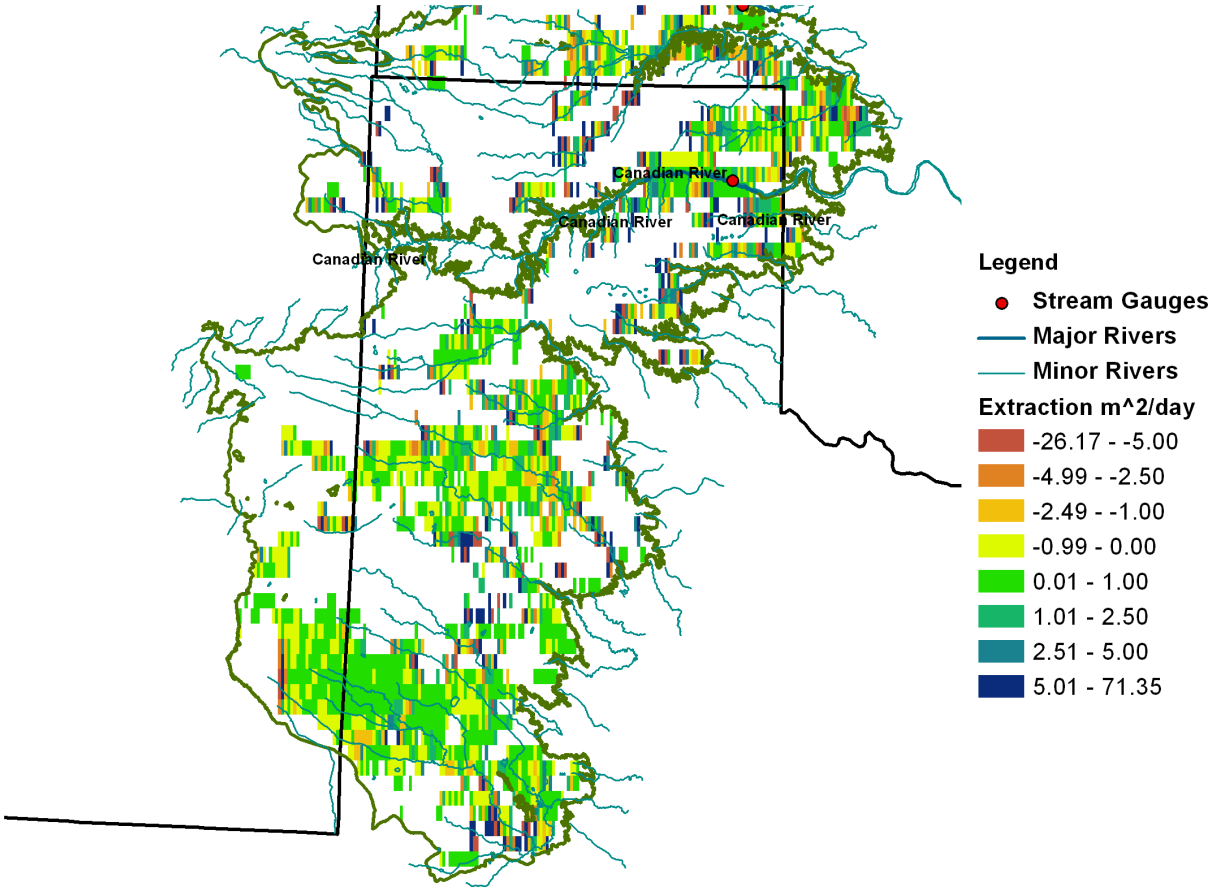

Figure 5.41: *Groundwater-Surface Water Interaction in the Southern Third of the High Plains Aquifer (L-M results)*

Table 5.17: *Regional Groundwater-Surface Water Interaction Summary for TX & NM*

PSO Results	
No. of Interaction Points	2700
Cumulative Interaction Rate (cfs)	245
Average Interaction Rate per 2km Cell (cfs)	0.09

L-M Results	
No. of Interaction Points	2436
Cumulative Interaction Rate (cfs)	258
Average Interaction Rate per 2km Cell (cfs)	0.11

Table 5.18: *Regional Simulated Head for TX & NM*

PSO Results	
Average Model Error (m)	0.79
Error Std. Dev.	12.6

L-M Results	
Average Model Error (m)	1.08
Error Std. Dev.	38.3

5.1.5 Discussion of Groundwater-Surface Water Interaction Results by River

This section discusses the river baseflow results presented in section 5.1.4. Because the flow rates measured at the stream gauges are influenced by surface runoff they are subject to fluctuation throughout the year. The baseflow rate, however, should be fairly constant throughout the year as the water table will not fluctuate enough under predevelopment conditions to greatly alter the aquifer's discharge rate. Based on this, the total baseflow should always be less than or equal to the smallest monthly flow rate that occurs within a river. This can provide a good test for the quality of the model results as baseflow values greater than measured stream gauge flow rates are likely indicators of erroneous results.

Canadian River Discussion

The Canadian River in Texas is a gaining river according to the model results because the amount of water discharged from the aquifer to the river is greater than the amount of water that percolates from the river bed down to the water table. The Levenberg-Marquardt and PSO methods both gave similar results with baseflow calculated from PSO being somewhat less than that found by Levenberg-Marquardt. The Levenberg-Marquardt result of 29cfs contribution to baseflow is actually greater than the average streamflow in the months of July and August indicating that the Levenberg-Marquardt result are likely too large. From the PSO results, the Canadian River appears to be deriving around 77% of its baseflow from groundwater during low flow periods and around 10% during high flow periods.

Cimarron River Discussion

The Cimarron River is a gaining river according to the results obtained. However, Levenberg-Marquardt baseflow results (577cfs) are greater than the total stream flows for all months and PSO baseflow results (124cfs) are larger than the total streamflow for all months except May and June. Here both PSO and Levenberg-Marquardt failed to produce reasonable results. Again, Levenberg-Marquardt is predicting larger baseflows than PSO. Predevelop-

ment water level contours of the Cimarron River are shown in figure fig:CimarronContours. These contours seem to indicate that the river is a gaining river (contours bend upstream) after the two forks combine. But the western half of the river shows no strong signs of being a gaining river. This part of the river may be losing more water to the aquifer than the sloping base model predicted resulting in baseflow values that are too large.

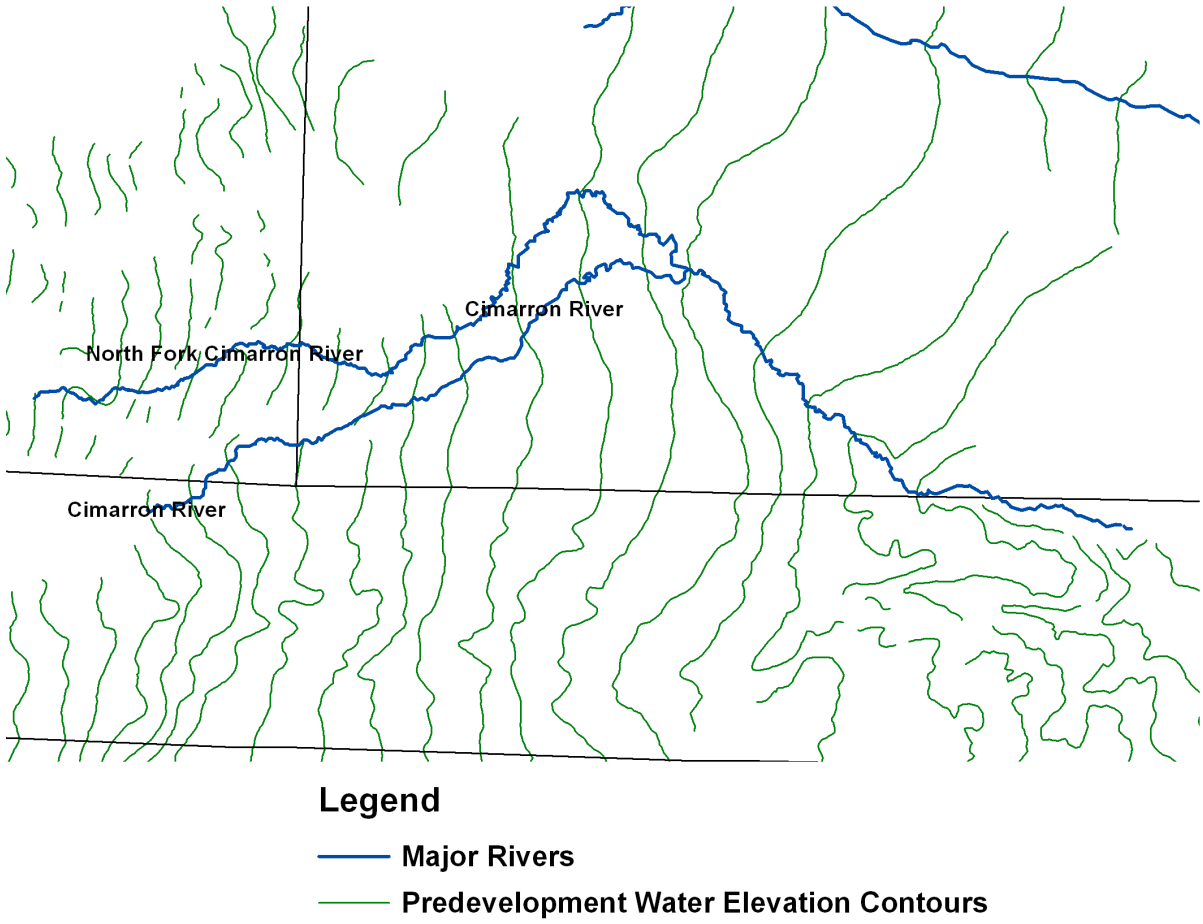

Figure 5.42: *Predevelopment Groundwater Elevation Contours Around the Cimarron River*

Arkansas River Discussion

The Arkansas River is a gaining river according to the model results. The Levenberg-Marquardt and PSO methods both produced very similar results. The PSO method used

a slightly higher number of interaction points and it avoided using large interaction rates (<-5m^2/day or >5m^2/day). From the results, the Arkansas River appears to be deriving approximately 50% of its stream flow from groundwater during low flow periods and approximately 14% during high flow periods.

Republican River Discussion

The Republican River is a gaining river according to the model results. The river appears to obtain around 50% of its flow from groundwater during low flow periods and around 11% during wet periods. The Levenberg-Marquardt and PSO methods both produced similar cumulative baseflow values. The PSO again used more points of interaction than Levenberg-Marquardt but it also avoided the larger interaction rate values which Levenberg-Marquardt did not.

Platte River Discussion

Model results for the Platte River were presented but the reliability of the results is low as the Platte cuts directly through the portion of the model in Nebraska that both the Levenberg-Marquardt and PSO techniques failed to optimize. The Levenberg-Marquardt technique used far fewer points of interaction but the interaction rates are much more variable than the PSO results and they tend to be much larger values. The PSO technique produced a more uniform pattern of recharge and discharge and avoided using the larger values. Much work needs to be done in this region to improve the model before anything definitive can be said about the aquifer's baseflow contribution to the Platte river.

5.1.6 Discussion of Groundwater-Surface Water Interaction Results by Basin

This section discusses the groundwater-surface water results obtained from the sloping base model at the basin scale. The results were separated into a northern basin that includes Nebraska, Wyoming, and South Dakota, a central basin that includes Kansas, Colorado,

and Oklahoma, and a southern basin that includes Texas and New Mexico.

Northern Basin Discussion

The northern basin is comprised of Nebraska, Wyoming, and South Dakota. The northern basin of the High Plains Aquifer is the area that the model had the most problems with. This area has many rivers as well as land areas that have a very shallow groundwater table. The Sand Hills region of Nebraska (figure 2.2) is a region that is highly connected to the aquifer and covers a large portion of the state. There are many ponds and wetland type areas that recharge the aquifer and that the aquifer discharges to. The shallow depth to water can be seen in figure 5.43 especially in the Sand Hill area and the river valleys. The high level of connectivity present in this region is displayed in figures 5.36 & 5.37 by the large number on points of interaction. This basin had the largest number of points of interaction with the optimization techniques using between 6200 and 7500 points of interaction to model this area. This high level of interaction is thought to be the primary reason behind the models inability to find a good solution for this area. Both optimization techniques produced high levels of error making results in this basin (north of the Republican river) highly unreliable.

Central Basin Discussion

The central basin is comprised of Kansas, Colorado, and Oklahoma. This basin is characterized by a water table that is typically fairly deep with the exception of a region in south central Kansas (figure 5.44). The larger depth to water present in this area results in a majority of the groundwater-surface water interaction occurring in river valleys. The PSO technique found the best solution in terms of minimizing errors in the simulated head with an average value of only 0.4m and a standard deviation of 8.6. The aquifer discharges around 700cfs in this basin according to the model results.

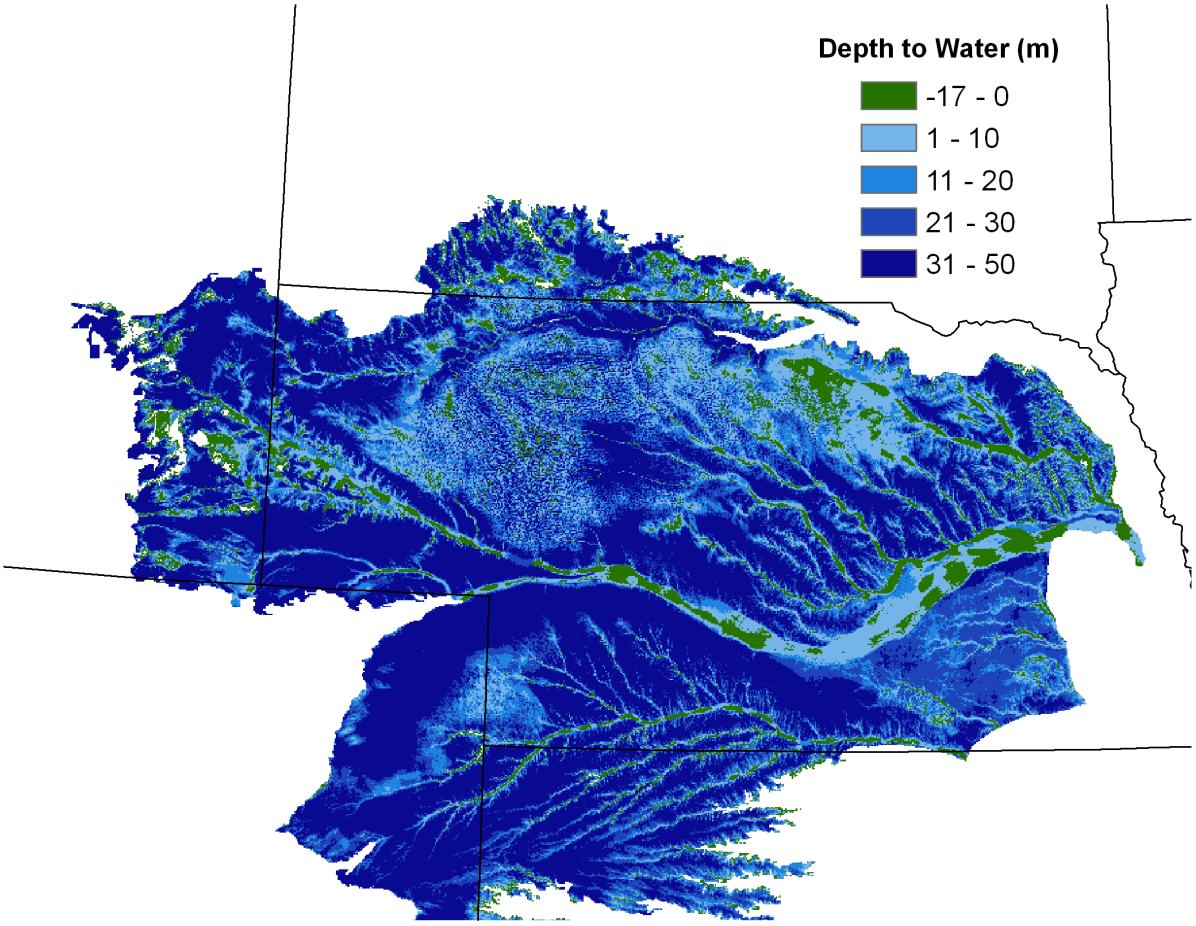

Figure 5.43: *Northern Basin Depth to Water*

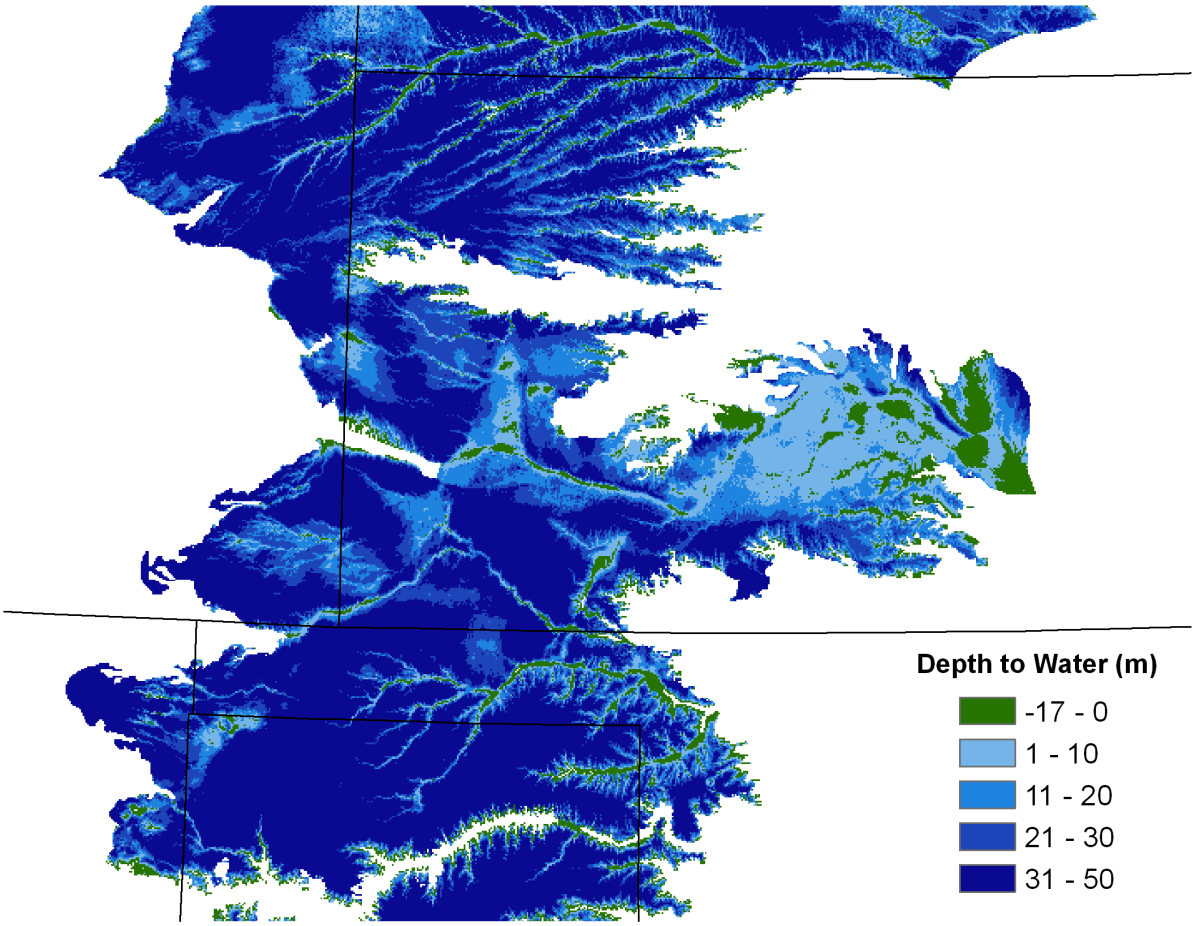

Figure 5.44: *Central Basin Depth to Water*

Southern Basin Discussion

The southern basin is comprised of Texas and New Mexico. The water table in this basin is deep in the northern half but can be shallow in the southern half (figure 5.45). Groundwater-surface water interaction in the north is predominately confined to river valleys whereas in the south the interaction is spread more broadly across the land area. The Levenberg-Marquardt and PSO methods produced similar cumulative interaction results with a value of around 250cfs being discharged by the aquifer in this basin. The average errors in this basin were low as with the central basin but the standard deviations were larger (12.6 for PSO and 38.3 for Levenberg-Marquardt).

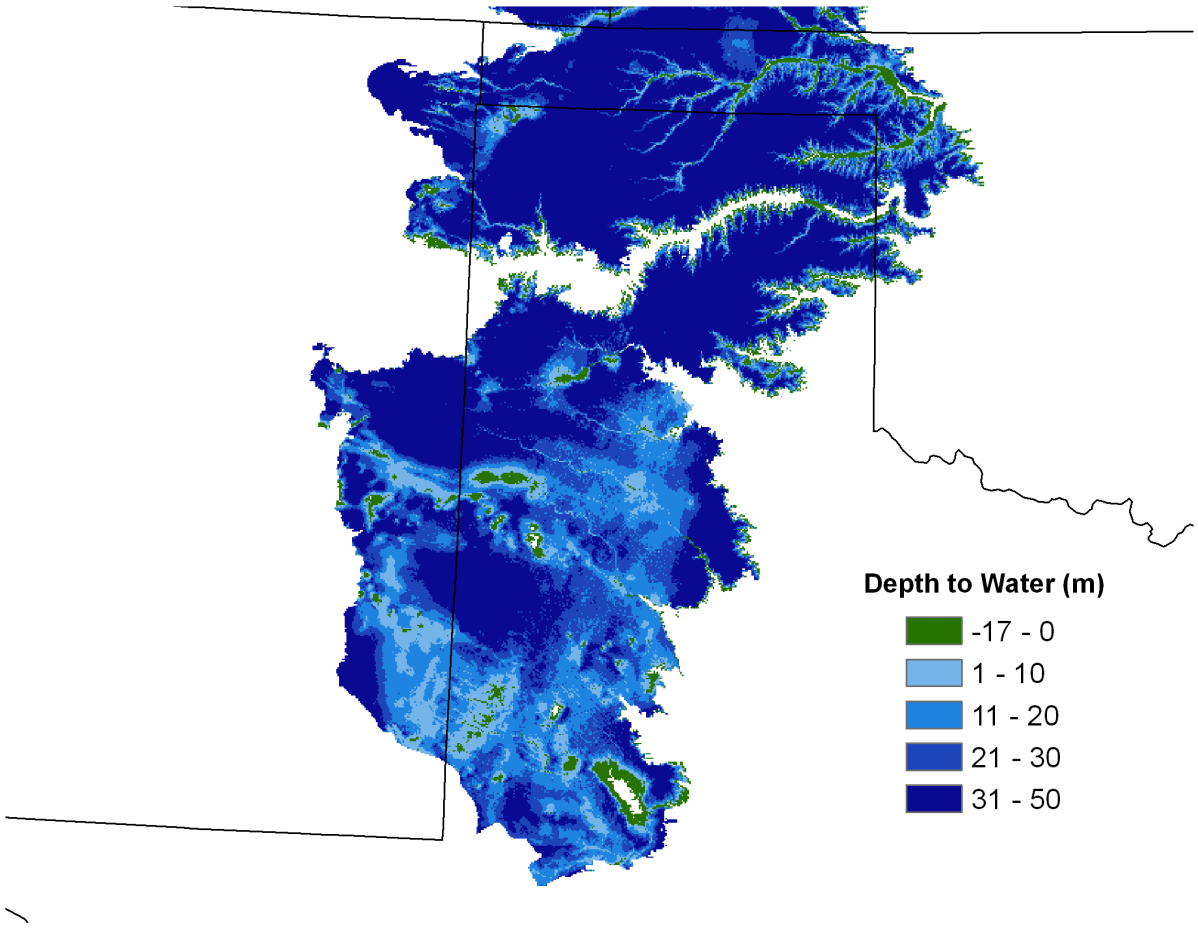

Figure 5.45: *Southern Basin Depth to Water*

Chapter 6

Conclusions

This study addresses the problem of locating and quantifying groundwater-surface water interaction in the High Plains Aquifer. MODFLOW was used to model the aquifer under steady-state conditions using standard datasets for hydraulic conductivity, bedrock elevation and recharge rates (see Appendix C for more information on these data sets). MODFLOW was unable to accurately match predevelopment heads using these inputs. In order to represent groundwater-surface water interaction in the model the values for river conductance were varied but accurate results were still unattainable (figures 3.1-3.4). Based on these results the method of using MODFLOW with a fixed number of rivers and a single conductance needed to be adjusted.

New techniques were employed to model the High Plains Aquifer and simultaneously locate and quantify points of groundwater-surface water interaction. The aquifer was modeled using a sloping base model using the same standard datasets applied in the MODFLOW models. Locations and rates of groundwater-surface water interaction were determined using the Levenberg-Marquardt algorithm and Particle Swarm Optimization (PSO) to optimize the sloping base model by adjusting groundwater-surface water interaction parameters.

Model results are presented in Chapter 5. Baseflow contribution to the Canadian River, Cimarron River, Arkansas River, Republican River, and Platte River are presented. Results for the Canadian River, Arkansas River, and Republican River indicate baseflows that make up approximately 10%, 14%, and 11% respectively of the monthly average stream flows

during the wet periods of the year (Tables 5.4, 5.8, & 5.10). During the dry periods of the year model results for the Canadian River, Arkansas River, and Republican River show baseflows that make up approximately 77%, 50%, and 50% respectively of the monthly average stream flows. Results for the Cimarron River and Platte River are shown in Tables 5.6 & 5.12 but appear to contain large errors, see sections 5.1.5 & 5.1.5 for discussion of errors in those rivers.

Results are also presented for the Northern, Central, and Southern basins. The best results were obtained in the Central and Southern basins were errors in simulated head were low (Tables 5.16 & 5.18). The Central basin had a net discharge rate of approximately 700cfs (Table 5.15). The Southern basin had a net discharge rate of approximately 250cfs (Table 5.17). The Northern basin had large errors in simulated head (Table 5.14).

This study substantiates that some regions have significant groundwater-surface water interaction and the model requires many parameters to match observed heads. Other regions do not have strong groundwater-surface water interaction and can be modeled using only a few groundwater-surface water parameters. Both PSO and Levenberg-Marquardt worked well in regions with few parameters, however, PSO finds a solution with lower objective function. For transects with many parameters, Levenberg-Marquardt converges to a solution that approximates the groundwater surface but leads to larger than expected fluxes of surface water. PSO also converges in these transects but it is unable to complete a comprehensive search of the parameter space.

Findings from this study show that groundwater-surface water interaction plays a large role in the overall water balance in the High Plains Aquifer. In predevelopment conditions most rivers had baseflow and was a net sink from groundwater. Results show that in the Northern basin there is very strong groundwater-surface water interaction with many surface water features. In the Central and Southern basins the groundwater-surface water interactions occur primarily along river valleys as the water table tends to be deep in those basins.

The results of this study provide robust methods for modeling an aquifer with a sloping base while minimizing the model's complexity. The reduced complexity of the stepping base model presented in this study allows for unknown model parameters such as groundwater-surface water interaction to be estimated. Estimation of these parameters was unachievable through the use of other standard modeling techniques which require higher degrees of complexity in input parameters. The ability to estimate highly variable parameters like groundwater-surface water interaction is important for accurate modeling of the High Plains Aquifer as there is little to no published data on these parameters. Parameter results from the sloping base model can be used to inform other models which struggle or are completely unable to accurately model the High Plains Aquifer with the standard published datasets.

Bibliography

Bakker, M., Anderson, E., Olsthoorn, T., and Strack, O. (1999). Regional groundwater modeling of the yucca mountain site using analytic elements. *Journal of Hydrology*, 226(3-4):167–178.

Boussinesq, J. (1904). Recherches theoriques sur l'ecoulement des nappes d'eau infiltrees dans le sol et. sur le debit des sources. *J. Math Pures Appl.*, 5(10):5–78.

Cederstrand, J. R. and Becker, M. F. (1999). Digital map of predevelopment water levels for the high plains aquifer in parts of colorado, kansas, nebraska, new mexico, oklahoma, south dakota, texas, and wyoming. Open-File Report OFR99-264, U. S. Geological Survey, Oklahoma City, OK.

Chapman, T. (2005). Recharge-induced groundwater flow over a plane sloping bed: Solutions for steady and transient flow using physical and numerical models. *Water Resources Research*, 41(7).

Chau, K. W. (2007). A split-step particle swarm optimization algorithm in river stage forecasting. *Journal of hydrology*, 346(3-4):131–135.

Childs, E. (1971). Drainage of groundwater resting on a sloping bed. *Water Resources Research*, 7(5):1256.

Daly, E. and Porporato, A. (2004). A note on groundwater flow along a hillslope. *Water Resources Research*, 40(1).

Darcy, H. (1856). *Les Fontaines Publiques de la Ville De Dijon*. Dalmont; Paris.

Dennehy, K. F. (2000). High plains regional ground-water study: U.s. geological survey fact sheet. Technical Report FS-091-00, U. S. Geological Survey, Reston, Virginia.

Dugan, J. T. and Zelt, R. B. (2000). Simulation and analysis of soil-water conditions in the great plains and adjacent areas, central united states, 1951-80. Technical Report 2427, U.S. Dept. of the Interior, U.S. Geological Survey ; Branch of Information Services,.

Dupuit, J. (1863). *Etudes theoriques et pratiques sur le mouvement des eaux dans les canaux decouverts et a travers les terrains permeables*. Paris:Libr. des Corps Imp. des Ponts et Chaussees et des Mines.

Feinstein, D., Buchwald, C., Dunning, C., and Hunt, R. (2005). Development and application of a screening model for simulating regional ground-water flow in the st. croix river basin, minnesota and wisconsin. U.S. Geological Survey Scientific Investigations Report 5283, U. S. Geological Survey.

Gaur, S. (2011). Analytic elements method and particle swarm optimization based simulation-optimization model for groundwater management. *Journal of hydrology*, 402(3-4):217–227.

Gillespie, J. and Slagle, S. (1972). Natural and artificial groundwater recharge, wet walnut creek, central kansas. Kansas Water Resources Board Bulletin 17, U. S. Geological Survey.

Gutentag, E., Heimes, F., Krothe, N., Luckey, R., and Weeks, J. (1984). Geohydrology of the high plains aquifer in parts of colorado, kansas, nebraska, new mexico, oklahoma, south dakota, texas, and wyoming. Professional Paper 1400-B, U. S. Geological Survey, Reston, Virginia.

Hansen, C. (1991). Estimates of freshwater storage and potential natural recharge for principal aquifers in kansas. Water Resources Investigations Report 87-4230, U. S. Geological Survey, Reston, Virginia.

Hornberger, G. M. (1998). *Elements of Phisical Hydrology*. Johns Hopkins Press.

Hunt, R., Saad, D., and Chapel, D. (2003). Numerical simulation of ground-water flow in la crosse county, wisconsin and into nearby pools of the mississippi river. U.S. Geological Survey Water-Resources Investigations Report 4154, U. S. Geological Survey.

Jordan, P. (1977). Streamflow transmission losses in western kansas. *Journal of the Hydraulics Division*, 103(8):905–919.

JT Dugan, J. P. (1985). Effects of climate, vegetation, and soils on consumptive water use and groundwater recharge to the central midwest regional aquifer system, mid-continent united states. Water Resources Investigations Report 85-4236, U. S. Geological Survey, Reston, Virginia.

Kambhammettu, B. V. N. P. (2010). Estimation of aquifer parameters from residual drawdowns. *Proceedings of the ICE - Water Management*, 163(7):361–365.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proc. IEEE Conf. on Neural Networks*, pages 1942–1948.

McMahon, P. B., Dennehy, K. F., Bruce, B. W., Gurdak, J. J., and Qi, S. L. (2007). Water-quality assessment of the high plains aquifer. Professional Paper 1749, U. S. Geological Survey, Reston, Virginia.

Meij, J. V. D. and Minnema, B. (1999). Modelling of the effect of a sea-level rise and land subsidence on the evolution of the groundwater density in the subsoil of the northern part of the netherlands. *Journal of Hydrology*, 226(3-4):152–166.

Piotrowski, A. (2011). Optimizing neural networks for river flow forecasting - evolutionary computation methods versus the levenberg-marquardt approach. *Journal of hydrology*, 407(1-4):12–27.

Power, B. F. (1993). *Model Calibration Techniques for Use with the Analytic Element Method*.

Raymond, H., Bondoc, M., McGinnis, J., Metropulos, K., Heider, P., Reed, A., and Saines, S. (2006). Using analytic element models to delineate drinking water source protection areas. *Ground Water*, 44(1):16–23.

Sniegocki, R. (1959). Geologic and ground-water reconnaissance of the loup river drainage basin, nebraska, with a section on chemical quality of the water by r.h. langford. U.S. Geological Survey Water-Supply Paper 1493, U. S. Geological Survey.

Sophocleous, M. (2002). Interactions between groundwater and surface water: the state of the science. *Hydrogeology Journal*, 10(1):52–67.

Sophocleous, M. (2005). Groundwater recharge and sustainability in the high plains aquifer in kansas, usa. *Hydrogeology Journal*, 13(2):351–365.

Steward, D. and Jin, W. (2001). Gaining and losing sections of horizontal wells. *Water Resources Research*, 37(11):2677–2685.

Steward, D., Yang, X., and Chacon, S. (2009). Groundwater response to changing water-use practices in sloping aquifers using convolution of transient response functions. *Water Resources Research*, 45(2).

Steward, D. R. (2007). Groundwater response to changing water-use practices in sloping aquifers. *Water Resources Research*, 43(5):W05408.

Strack, O. D. L. (2003). *Groundwater Mechanics*. Upper Saddle River; N.J.: Prentice-Hall.

Szilagyi, J., Zlotnik, V., Gates, J., and Jozsa, J. (2011). Mapping mean annual groundwater recharge in the nebraska sand hills, usa. *Hydrogeology Journal*, 19(8):1503–1513.

Appendix A

Data Preprocessing Methods

The following is a presentation of the procedures and Python scripts used to process the aquifer data sets and get them into a form that is compatible with the sloping base model of the High Plains aquifer presented in this thesis. Python scripts were developed to automate all of the data processing. The scripts were written in Python 2.6 using the arcgisscripting geoprocessing module for ArcGIS 9.3. The scripts are also compatible with ArcGIS 10.

A.1 Introduction to Python Scripts

The flowchart shown in figure A.1 illustrates the order of execution of the python scripts used to create the input data for the sloping base model. Rectangular boxes represent the python scripts used in all preprocessing routines, oval shapes indicate the scripts whose contents may need to be modified by the user to get the desired outputs, diamond shapes represent outputs from the scripts, and the arrows indicate the flow of information. Each script is explained in more detail in the following sections.

A.1.1 MODFLOWFilesGen.py

The script MODFLOWFilesGen.py calls the MODFLOWtxt.py script. The user needs to enter cell sizes for the digital elevation model (DEM) and for the recharge raster. Cell sizes also need to be entered for the rasters that will be created by MODFLOWtxt.py. The user can enter as many output cell sizes as desired, rasters and text files will be saved in seperate

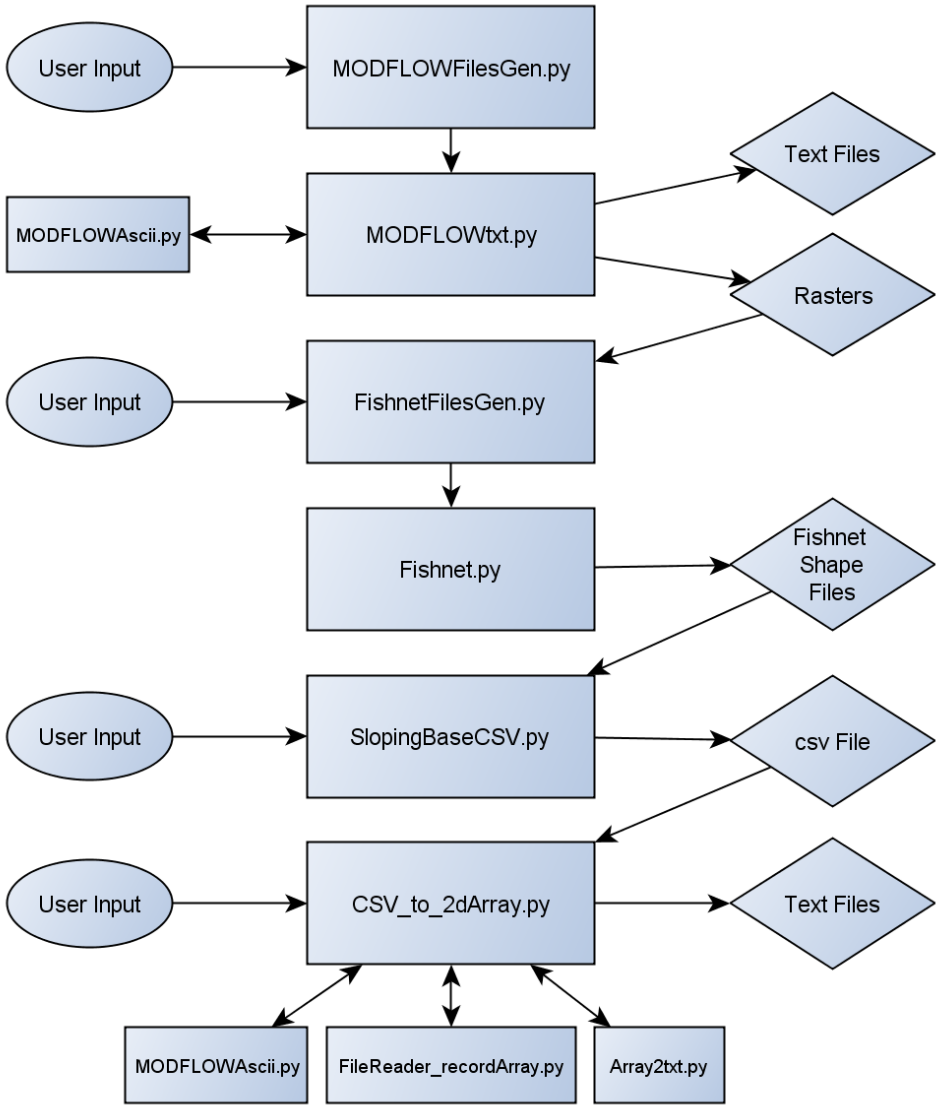

Figure A.1: *Preprocessing Python Scripts Flowchart*

folders for each cell size entered. The user input *dem_minus* is necessary to create an initial water table elevation guess that is used in MODFLOW models, it is not used in the sloping base model. Names of the shapefiles must be entered as well as two folder paths identifying the location of the ArcGIS toolbox folder and the ArcGIS spatial reference folder. The variable *AquiferCoverage* is the shapefile that represents the area to be modeled. The other shapefiles and rasters used must cover all the area in the *AquiferCoverage* shapefile. If,

for example, the *Hyd_K* (hydraulic conductivity) shapefile does not have data for part of the geographic area covered in the *AquiferCoverage* shapefile there will be an error and the script will not execute successfully. All input shapefiles and rasters must have the same projection.

```
#----------------------
MODFLOWFilesGen.py
#----------------------

from MODFLOWtxt import *

#HighPlains Model

#Enter cell sizes and path to working folder and path to output folder
DEM_cellsize = 250.0 #Cellsize of DEM being used for data creation, must be less than or equal to smallest cell size entered below
Recharge_cellsize = 500.0  #cell size of recharge raster
dem_minus = 3 #Value to subtract from DEM to create initial head values, this is only important for a MODFLOW model
cellsizes = [5000, 2500, 1000] #Cellsizes to use for raster creation
workingFolder = "D:/Research/MODFLOW/High_Plains/WorkingFiles/"  #Folder containing primary GIS files
outputFolder = "D:/Research/MODFLOW/High_Plains/Output/"  #Folder to write output to

#Enter the names of your working files
SPYD = "spyd.shp"
Hyd_K = "hyd_k.shp"
Bedrock = "bedrock_elev.shp"
Recharge = "rech_in_yr_ss"
PredevWaterLv = "predevwlv_elev.shp"
HighPlainsRivers = "HighPlainsMajorRiversNHD.shp"
DEM = "dem_1000"
AquiferCoverage = "hp_extents.shp"

#Enter the following folder locations
ToolboxFolder = "C:/Program Files (x86)/ArcGis/Desktop10.0/ArcToolbox/Toolboxes/"
Spatial_Ref_Folder = "C:/Program Files (x86)/ArcGIS/Desktop10.0/Coordinate Systems/Projected Coordinate Systems/UTM/NAD 1983/"

names=[SPYD,Hyd_K,Bedrock,Recharge,PredevWaterLv,HighPlainsRivers,DEM,AquiferCoverage,ToolboxFolder,Spatial_Ref_Folder]
for cellsize in cellsizes:
    outDirec = outputFolder + str(cellsize)
    MODFLOWtxt(cellsize,DEM_cellsize,Recharge_cellsize,dem_minus,workingFolder,outDirec,names)
```

A.1.2 MODFLOWtxt.py

The script MODFLOWtxt.py automates all the shapefile and raster creation that would otherwise be done manually in ArcGIS. It also creates text files formatted for use with the MODFLOW processors PMWIN and mflab. The output folder will contain 3 subfolders called *txt*, *MODFLOW_txt*, and *GIS*. The *txt* folder will contain the unformatted text file outputs from ArcGIS, the *MODFLOW_txt* folder will contain the the same text files but they will be formatted for use with PMWIN and mflab, and the *GIS* folder will contain the output rasters. Within the *GIS* folder there is another subfolder named *Temp* which will contain rasters and shapefiles that were necessary only for creating the final rasters in the *GIS* folder. If inputs for a MODFLOW model are all that is desired then no further steps beyond execution of this script are necessary.

```
#----------------------
MODFLOWtxt.py
#----------------------
import time, sys, string, os, arcgisscripting, shutil
from ModflowASCII import *

def MODFLOWtxt(cellsize,DEM_cellsize,Recharge_cellsize,dem_minus,
    working_folder,outDirec,names):
    start = time.clock()

    gp = arcgisscripting.create(9.3)

    #Turn creation of data on or off
    create_AquiferPolygon = True
    create_DEM = True
    create_SPYD = True
    create_K = True
    create_Bedrock = True
    create_Recharge = True
    create_InitialHead = True
    create_RiverHead = True
    create_RiverBottom = True
    create_Conductance = True
    create_ModelBoundary = True
    create_SlopingBaseBoundary = True
    create_Wells = True

    SPYD = working_folder + names[0]
    Hyd_K = working_folder + names[1]
    Bedrock = working_folder + names[2]
```

```
    Recharge = working_folder + names[3]
    PredevWaterLv = working_folder + names[4]
    HighPlainsRivers = working_folder + names[5]
    DEM = working_folder + names[6]
    AquiferCoverage = working_folder + names[7]
    ToolboxFolder = names[8]
    Spatial_Ref_Folder = names[9]

    #Create folder to hold new model information
    dirname = outDirec

    iteration = 0

    try:
        os.makedirs(dirname)
    except OSError:
        while os.path.exists(dirname):
            iteration += 1
            dirname = outDirec + "_" + str(iteration)
        try:
            os.makedirs(dirname)
        except OSError:
            raise

    os.makedirs(dirname + "/GIS")
    os.makedirs(dirname + "/MODFLOW_txt")
    os.makedirs(dirname + "/GIS/Temp")
    os.makedirs(dirname + "/txt")

    print "Working Directory:",working_folder
    print "Output Directory:", dirname
    print "cell size =",cellsize

    #Create Aquifer Polygon
    #Extents
    desc = gp.Describe(AquiferCoverage)
    extent = desc.Extent
    top = float(extent.ymax)
    bottom = float(extent.ymin)
    left = float(extent.xmin)
    right = float(extent.xmax)

    if top < 0:
        top = int(top) - 1
    else:
        top = int(top) + 1
    if bottom < 0 :
        bottom = int(bottom) - 1
    else:
        bottom = int(bottom) + 1
    if left < 0:
```

```
            left = int(left) - 1
        else:
            left = int(left) + 1
        if right < 0:
            right = int(right) - 1
        else:
            right = int(right) + 1

        #Determine required size of polygon
        width = float(abs(left - right))
        new_width = width
        while new_width/cellsize - int(new_width/cellsize) <> 0:
            new_width += 1.0

        #Adjust left and right extents
        change = (new_width - width)/2
        if left < 0:
            left = abs(left) + int(change)
            left = -left
        else:
            left = left + int(change)

        if right < 0:
            if change - int(change) <> 0:
                right = abs(right) + int(change) + 1
                right = -right
            else:
                right = abs(right) + int(change)
                right = -right
        else:
            if change - int(change) <> 0:
                right = right + int(change) + 1
            else:
                right = right + int(change)


        length = float(abs(top - bottom))
        new_length = length
        while new_length/cellsize - int(new_length/cellsize) <> 0:
            new_length += 1.0

        #Adjust top and bottom extents
        change = (new_length - length)/2
        if top < 0:
            top = abs(top) + int(change)
            top = -top
        else:
            top = top + int(change)

        if bottom < 0:
            if change - int(change) <> 0:
```

```
132             bottom = abs(bottom) - int(change) - 1
                bottom = -bottom
134         else:
                bottom = abs(bottom) - int(change)
136             bottom = -bottom
    else:
138         if change - int(change) <> 0:
                bottom = bottom - int(change) - 1
140         else:
                bottom = bottom - int(change)
142
    left = left - cellsize*10
144 right = right + cellsize*10
    top = top + cellsize*10
146 bottom = bottom - cellsize*10

148 outFolder = dirname + "/GIS"
    gp.workspace = outFolder
150 textFolders = dirname


152
    #Create AquiferPolygon shapefile
154 if create_AquiferPolygon or create_ModelBoundary or create_Conductance
        :
        print 'Creating Aquifer Polygon files'
156     try:
            outFile = "AquiferPolygon.shp"
158
            StartandEndXcoord = left
160         StartandEndYcoord = top

162         PointOneXcoord = right
            PointOneYcoord = top
164
            PointTwoXcoord = right
166         PointTwoYcoord = bottom

168         PointThreeXcoord = left
            PointThreeYcoord = bottom
170         sr = gp.CreateSpatialReference(Spatial_Ref_Folder + "NAD 1983
                UTM Zone 14N.prj", "#", "#", "#", "#", "#")

172         gp.CreateFeatureclass_management(outFolder, outFile, "POLYGON"
                )
            cur = gp.InsertCursor(outFile)
174         row = cur.NewRow()

176         PolygonArray = gp.CreateObject("Array")
            pnt = gp.CreateObject("Point")
178
            pnt.x = StartandEndXcoord
```

```
180             pnt.y = StartandEndYcoord
                PolygonArray.add(pnt)
182
                pnt.x = PointOneXcoord
184             pnt.y = PointOneYcoord
                PolygonArray.add(pnt)
186
                pnt.x = PointTwoXcoord
188             pnt.y = PointTwoYcoord
                PolygonArray.add(pnt)
190
                pnt.x = PointThreeXcoord
192             pnt.y = PointThreeYcoord
                PolygonArray.add(pnt)
194
                pnt.x = StartandEndXcoord
196             pnt.y = StartandEndYcoord
                PolygonArray.add(pnt)
198
                row.shape = PolygonArray
200             cur.InsertRow(row)

202             del row, cur

204         except:
                raise
206
    AquiferPolygon = outFolder + "/AquiferPolygon.shp"
208

210     #Create DEM
        if DEM_cellsize <> cellsize:
212         print 'Resampling DEM to match input cell size'
            try:
214             if DEM_cellsize > cellsize:
                    print "Dem cellsize > model cellsize"
216
                else:
218                 # Local variables...
                    DEM_resample = outFolder + "/dem"
220
                    # Check out any necessary licenses
222                 gp.CheckOutExtension("spatial")

224                 # Load required toolboxes...
                    gp.AddToolbox(ToolboxFolder + "Data Management Tools.tbx")
226
                    # Process: Resample...
228                 gp.Resample_management(DEM, DEM_resample, str(cellsize), "
                        NEAREST")
```

```
230                 DEM = DEM_resample

232         except:
                raise
234
        else:
236         print 'Copying DEM to output folder'
            try:
238             # Local variables...
                DEM_resample = outFolder + "/dem"
240
                gp.CopyRaster_management(DEM, DEM_resample, "", "", "", "NONE"
                    , "NONE", "")
242
            except:
244             raise
                gp.GetMessage(2)
246
        #Create Specific Yield files
248     if create_SPYD:
            print 'Creating Specific Yield files'
250         try:
                # Load required toolboxes...
252             gp.AddToolbox(ToolboxFolder + "Conversion Tools.tbx")

254             #Local Variables
                Raster_SPYD = outFolder + "/SPYD"
256             gp.Extent = str(left) + " " + str(bottom) + " " + str(right) +
                     " " + str(top)
                SPYD_ASCII_file = textFolders + "/txt/spyd.txt"
258
                # Process: Feature to Raster...
260             gp.FeatureToRaster_conversion(SPYD, "AVG_SPYD", Raster_SPYD,
                    str(cellsize))

262             # Process: Raster to ASCII...
                gp.RasterToASCII_conversion(Raster_SPYD, SPYD_ASCII_file)
264
                # Edit text file
266             MODFLOW_ascii(SPYD_ASCII_file, textFolders + "/MODFLOW_txt/
                    SPYD_MODFLOW.txt")

268             del Raster_SPYD,SPYD_ASCII_file

270         except:
                raise
272
        #Create Hydraulic Conductivity files
274     if create_K:
            print 'Creating Hydraulic Conductivity files'
276         try:
```

```
            # Allow overwrite
278         gp.OverwriteOutput = True

280         gp.workspace = outFolder + "/Temp/"

282         # Check out any necessary licenses
            gp.CheckOutExtension("spatial")
284
            # Load required toolboxes...
286         gp.AddToolbox(ToolboxFolder + "Conversion Tools.tbx")
            gp.AddToolbox(ToolboxFolder + "Spatial Analyst Tools.tbx")
288
            #Local Variables
290         gp.Extent = str(left) + " " + str(bottom) + " " + str(right) +
                " " + str(top)
            Raster_K = outFolder + "/Temp/hyd_K"
292         K1_md = outFolder + "\\Temp\\k1_md"
            K2_md = outFolder + "\\Temp\\k2_md"
294         K3_md = outFolder + "\\k_md"
            K_ASCII_file = textFolders + "\\txt\\k.txt"
296
            # Process: Feature to Raster...
298         gp.FeatureToRaster_conversion(Hyd_K, "AVG_K", Raster_K, str(
                cellsize))
            #print 'times'
300         # Process: Times...
            feet_to_meters = .3048
302         gp.Times_sa(Raster_K, feet_to_meters, K1_md)

304         # Process: Single Output Map Algebra...
            gp.SingleOutputMapAlgebra_sa("con(isnull(k1_md),17,k1_md)",
                K2_md, "")
306         # Process: Single Output Map Algebra...
            gp.SingleOutputMapAlgebra_sa("con(k2_md == 0,1,k2_md)", K3_md,
                 "")
308
            # Process: Raster to ASCII...
310         gp.RasterToASCII_conversion(K3_md, K_ASCII_file)

312         # Edit text file
            MODFLOW_ascii(K_ASCII_file, textFolders + "\\MODFLOW_txt\\
                K_MODFLOW.txt")
314
            gp.workspace = outFolder
316
            del Raster_K, K1_md, K2_md, K3_md, K_ASCII_file
318
        except:
320         raise

322 #Create Bedrock files
```

```
    if create_Bedrock or create_ModelBoundary or create_RiverHead or
        create_Conductance:
324      print 'Creating Bedrock files'
         try:
326          # Check out any necessary licenses
             gp.CheckOutExtension("spatial")
328
             # Load required toolboxes...
330          gp.AddToolbox(ToolboxFolder + "Spatial Analyst Tools.tbx")

332          # Local variables...
             bedrock_raster = outFolder + "\\Temp\\bedrock_elev"
334          Output_stream_polyline_features = ""
             Output_remaining_sink_point_features = ""
336          Output_diagnostic_file = ""
             Output_parameter_file = ""
338          bedrock_raster_m = outFolder + "\\bedrock_m"
             Bedrock_ASCII_file = textFolders + "\\txt\\bedrock.txt"
340
             #Create bedrock raster from topo lines
342          gp.TopoToRaster_sa(Bedrock + " ELEV Contour", bedrock_raster,
                 str(cellsize), AquiferCoverage, "20", "", "", "ENFORCE", "
                 CONTOUR", "40", "", "1", "0", "", "",
                 Output_stream_polyline_features,
                 Output_remaining_sink_point_features,
                 Output_diagnostic_file, Output_parameter_file)

344          # Process: Times...
             feet_to_meters = .3048
346          gp.Times_sa(bedrock_raster, feet_to_meters, bedrock_raster_m)

348          # Process: Raster to ASCII...
             gp.RasterToASCII_conversion(bedrock_raster_m,
                 Bedrock_ASCII_file)
350
             # Edit text file
352          MODFLOW_ascii(Bedrock_ASCII_file, textFolders + "\\MODFLOW_txt
                 \\Bedrock_MODFLOW.txt")

354          del bedrock_raster, Output_stream_polyline_features,
                 Output_remaining_sink_point_features,
                 Output_diagnostic_file
             del Output_parameter_file, Bedrock_ASCII_file
356
         except:
358          raise

360  #Create Predevelopment Water Level files
     if create_InitialHead or create_ModelBoundary or create_RiverHead or
        create_Conductance:
362      print 'Creating Initial Head files'
```

```
        try:
364         # Check out any necessary licenses
            gp.CheckOutExtension("spatial")
366
            # Load required toolboxes...
368         gp.AddToolbox(ToolboxFolder + "Spatial Analyst Tools.tbx")

370         # Local variables...
            thickness = outFolder + "\\Temp\\thickness"
372         thickness_reclass = outFolder + "\\thickness_rec"
            pred_wlv_raster = outFolder + "\\Temp\\pred_wlv"
374         Output_stream_polyline_features = ""
            Output_remaining_sink_point_features = ""
376         Output_diagnostic_file = ""
            Output_parameter_file = ""
378         pred_wlv_raster_m = outFolder + "\\pred_wlv_m"
            InitialHead_raster = outFolder + "\\initialhead_m"
380         InitialHead_ASCII_file = textFolders + "\\txt\\initialhead.txt
                "
            PredWlv_ASCII_file = textFolders + "\\txt\\predwlv.txt"
382
            #Create Predeveolopment Water Level Raster for future
                comparison
384         # Process: Topo to Raster...
            gp.TopoToRaster_sa(PredevWaterLv + " ELEV Contour",
                pred_wlv_raster, str(cellsize), AquiferCoverage, "20", "",
                 "", "ENFORCE", "CONTOUR", "60", "", "0.75", "0", "", "",
                Output_stream_polyline_features,
                Output_remaining_sink_point_features,
                Output_diagnostic_file, Output_parameter_file)
386
            # Process: Times...
388         feet_to_meters = .3048
            gp.Times_sa(pred_wlv_raster, feet_to_meters, pred_wlv_raster_m
                )
390
            # Process: Minus...
392         #gp.Minus_sa(InitialHead_raster, bedrock_raster_m, thickness)

394         #Create Initial Head
            #gp.Minus_sa(DEM, str(dem_minus), InitialHead_raster)
396         gp.Minus_sa(pred_wlv_raster_m, bedrock_raster_m, thickness) #
                for use with optimization scripts

398         # Process: Reclassify...
            gp.Reclassify_sa(thickness, "Value", "-10000 5 0;5 10000 1",
                thickness_reclass, "DATA")
400
            # Process: Raster to ASCII...
402         gp.RasterToASCII_conversion(thickness_reclass,
                InitialHead_ASCII_file)
```

```
404             # Process: Raster to ASCII...
                gp.RasterToASCII_conversion(pred_wlv_raster_m,
                    PredWlv_ASCII_file)
406
                # Edit text file
408             MODFLOW_ascii(InitialHead_ASCII_file, textFolders + "\\
                    MODFLOW_txt\\InitialHead_MODFLOW.txt")
                MODFLOW_ascii(PredWlv_ASCII_file, textFolders + "\\MODFLOW_txt
                    \\PredWlv_MODFLOW.txt")
410
                del pred_wlv_raster, Output_stream_polyline_features,
                    Output_remaining_sink_point_features,
                    Output_diagnostic_file
412             del Output_parameter_file, pred_wlv_raster_m,
                    InitialHead_ASCII_file, PredWlv_ASCII_file

414         except:
                raise
416
    #Create Recharge files
418 if create_Recharge:
        if Recharge_cellsize <> cellsize:
420         print 'Resampling Recharge Raster'
            try:
422             if Recharge_cellsize > cellsize:
                    print "Recharge cellsize > model cellsize"
424
                else:
426                 # Local variables...
                    Recharge_clip = outFolder + "\\Temp\\recharge_clip"
428                 Recharge_resample = outFolder + "\\Temp\\recharge_res"
                    Recharge_md = outFolder + "\\recharge_md"
430                 Recharge_ASCII_file = textFolders + "\\txt\\recharge.
                        txt"

432                 # Check out any necessary licenses
                    gp.CheckOutExtension("spatial")
434
                    # Load required toolboxes...
436                 gp.AddToolbox(ToolboxFolder + "Data Management Tools.
                        tbx")
                    gp.AddToolbox(ToolboxFolder + "Spatial Analyst Tools.
                        tbx")
438
                    # Process: Clip (2)...
440                 gp.Clip_management(Recharge, str(left) + " " + str(
                        bottom) + " " + str(right) + " " + str(top),
                        Recharge_clip, AquiferPolygon, "", "
                        ClippingGeometry")
```

```
442                     # Process: Resample...
                        gp.Resample_management(Recharge_clip,
                            Recharge_resample, str(cellsize), "NEAREST")
444
                        # Process: Times...
446                     inchesPERyear_to_metersPERday = 0.0254/365
                        gp.Times_sa(Recharge_resample,
                            inchesPERyear_to_metersPERday, Recharge_md)
448
                        # Process: Raster to ASCII...
450                     gp.RasterToASCII_conversion(Recharge_md,
                            Recharge_ASCII_file)

452                     # Edit text file
                        MODFLOW_ascii(Recharge_ASCII_file, textFolders + "\\
                            MODFLOW_txt\\Recharge_MODFLOW.txt")
454
                        del Recharge_clip,Recharge_resample,Recharge_md,
                            Recharge_ASCII_file
456
                except:
458                 raise

460         else:
                print 'Creating Recharge files'
462             try:
                    Recharge_md = outFolder + "\\recharge_md"
464                 Recharge_ASCII_file = textFolders + "\\txt\\recharge.txt"

466                 # Process: Times...
                    inchesPERyear_to_metersPERday = 0.0254/365
468                 gp.Times_sa(Recharge, inchesPERyear_to_metersPERday,
                        Recharge_md)

470                 # Process: Raster to ASCII...
                    gp.RasterToASCII_conversion(Recharge_md,
                        Recharge_ASCII_file)
472
                    # Edit text file
474                 MODFLOW_ascii(Recharge_ASCII_file, textFolders + "\\
                        MODFLOW_txt\\Recharge_MODFLOW.txt")

476                 del Recharge_md,Recharge_ASCII_file

478             except:
                    raise
480                 gp.GetMessage(2)

482     #Create Boundary Files
        if create_ModelBoundary or create_RiverHead or create_Conductance or
            create_SlopingBaseBoundary:
```

```
484         print 'Creating Model Boundary files'
            try:
486             # Allow overwrite
                gp.OverwriteOutput = True
488
                # Check out any necessary licenses
490             gp.CheckOutExtension("spatial")

492             # Load required toolboxes...
                gp.AddToolbox(ToolboxFolder + "Spatial Analyst Tools.tbx")
494             gp.AddToolbox(ToolboxFolder + "Conversion Tools.tbx")
                gp.AddToolbox(ToolboxFolder + "Data Management Tools.tbx")
496
                # Local variables...
498             gp.Extent = str(left) + " " + str(bottom) + " " + str(right) +
                     " " + str(top)
                AquiferCoverage_raster = outFolder + "\\Temp\\aquifer_ras"
500             Extents_raster = outFolder + "\\Temp\\model_extents"
                Extents_raster_reclassify = outFolder + "\\Temp\\mod_ext_rec"
502             Extents_Polygon = outFolder + "\\Temp\\extents_poly.shp"
                Single_Model_Polygon = outFolder + "\\Temp\\
                   single_boundary_polygon.shp"
504             Single_Model_Polygon_Agg = outFolder + "\\Temp\\
                   single_boundary_polygon_agg.shp"
                Single_Model_Polygon_Agg_Buffer = outFolder + "\\Temp\\
                   single_boundary_polygon_agg_buffer.shp"
506             Spec_Head_Bound_Poly = outFolder + "\\Temp\\spec_head_bound.
                   shp"
                Single_Model_Raster = outFolder + "\\Temp\\bound_raster1"
508             Single_Model_Raster_Agg = outFolder + "\\Temp\\bound_raster2"
                Single_Model_Raster_Agg_Buffer = outFolder + "\\Temp\\
                   bound_raster3"
510             Single_Model_Raster_Reclass = outFolder + "\\Temp\\
                   bound_ras1_rc"
                Single_Model_Raster_Agg_Reclass = outFolder + "\\Temp\\
                   bound_ras2_rc"
512             Single_Model_Raster_Agg_Buffer_Reclass = outFolder + "\\Temp\\
                   bound_ras3_rc"
                Bound_Raster4 = outFolder + "\\Temp\\bound_raster4"
514             Bound_Raster5 = outFolder + "\\Temp\\bound_raster5"
                Bound_Raster6 = outFolder + "\\Temp\\bound_raster6"
516             Boundary_Raster = outFolder + "\\boundary"
                Boundary_Centroids = outFolder + "\\Temp\\boundary_centroids.
                   shp"
518             Bound_X = outFolder + "\\Temp\\bound_x"
                Bound_Y = outFolder + "\\Temp\\bound_y"
520             X_Coords = textFolders + "\\txt\\X_coords.txt"
                Y_Coords = textFolders + "\\txt\\Y_coords.txt"
522             Boundary_ASCII_file = textFolders + "\\txt\\boundary.txt"

524
```

```
            # Process: Feature to Raster...
526         fields = gp.ListFields(AquiferCoverage)
            names = []
528
            for field in fields:
530             names.append(field.Name)

532         if "Raster_val" in names:
                gp.FeatureToRaster_conversion(AquiferCoverage, "Raster_val
                    ", AquiferCoverage_raster, str(cellsize))
534         else:
                gp.AddField(AquiferCoverage, "Raster_val", "short")
536             gp.CalculateField_management (AquiferCoverage, "Raster_val
                    ", "1", "PYTHON_9.3")
                gp.FeatureToRaster_conversion(AquiferCoverage, "Raster_val
                    ", AquiferCoverage_raster, str(cellsize))
538
            # Process: Plus...
540         gp.Plus_sa(AquiferCoverage_raster, thickness_reclass,
                Extents_raster)

542         # Process: Reclassify...
            gp.Reclassify_sa(Extents_raster, "Value", "1 1;2 2;NODATA 1",
                Extents_raster_reclassify, "DATA")
544
            # Process: Extents_raster_reclassify to Polygon...
546         gp.RasterToPolygon_conversion(Extents_raster_reclassify,
                Extents_Polygon, "NO_SIMPLIFY", "VALUE")

548         # Add Area Field
            gp.AddField_management (Extents_Polygon, "Area", "FLOAT", "",
                "", "", "", "NON_NULLABLE", "NON_REQUIRED", "")
550
            # Calculate Area
552         expression = "float(!SHAPE.AREA@SQUAREMILES!)"
            gp.CalculateField_management (Extents_Polygon, "Area",
                expression, "PYTHON_9.3")
554
            # Find, select, and save largest polygon within boundary to
                new shapefile:
556         searchRows = gp.searchcursor(Extents_Polygon)
            searchRow = searchRows.next()
558         Area = []

560         while searchRow:
                if searchRow.GRIDCODE == 2:
562                 Area.append(searchRow.Area)
                searchRow = searchRows.next()
564
            maxArea = max(Area)
566         area_string = "\"Area\" = " + str(maxArea)
```

```
            area_string = "Area > " + str(int(maxArea) - 1) + " AND Area <
                " + str(int(maxArea) + 1)
568
            gp.MakeFeatureLayer(Extents_Polygon,"bound_lyr")
570
            gp.SelectLayerByAttribute_management("bound_lyr", "
                NEW_SELECTION", area_string)
572
            gp.CopyFeatures("bound_lyr", Single_Model_Polygon)
574
            # Process: Single_Model_Polygon Polygons...
576         gp.AggregatePolygons_management(Single_Model_Polygon,
                Single_Model_Polygon_Agg, str(int(cellsize)) + " Unknown",
                 "0 Unknown", "1E+20 SquareMiles", "NON_ORTHOGONAL")

578         # Process: Buffer Single_Model_Polygon...
            gp.Buffer_analysis(Single_Model_Polygon_Agg,
                Single_Model_Polygon_Agg_Buffer, str(-int(cellsize)) + "
                Meters", "FULL", "FLAT", "NONE", "")
580
            # Add Raster_val Field to Single_Model_Polygon
582         gp.AddField_management (Single_Model_Polygon, "Raster_val", "
                SHORT", "", "", "", "", "NON_NULLABLE", "NON_REQUIRED", ""
                )

584         # Add Raster_val Field to Single_Model_Polygon_Agg
            gp.AddField_management (Single_Model_Polygon_Agg, "Raster_val"
                , "SHORT", "", "", "", "", "NON_NULLABLE", "NON_REQUIRED",
                 "")
586
            # Add Raster_val Field to Single_Model_Polygon_Agg_Buffer
588         gp.AddField_management (Single_Model_Polygon_Agg_Buffer, "
                Raster_val", "SHORT", "", "", "", "", "NON_NULLABLE", "
                NON_REQUIRED", "")

590         # Process: Calculate Field for Single_Model_Polygon...
            gp.CalculateField_management(Single_Model_Polygon, "Raster_val
                ", "1", "PYTHON_9.3", "")
592
            # Process: Calculate Field for Single_Model_Polygon_Agg...
594         gp.CalculateField_management(Single_Model_Polygon_Agg, "
                Raster_val", "1", "PYTHON_9.3", "")

596         # Process: Calculate Field for Single_Model_Polygon_Agg_Buffer
                ...
            gp.CalculateField_management(Single_Model_Polygon_Agg_Buffer,
                "Raster_val", "1", "PYTHON_9.3", "")
598
            # Process: PolygonToRaster...
600         gp.PolygonToRaster_conversion(Single_Model_Polygon, "
                Raster_val", Single_Model_Raster, "CELL_CENTER", "NONE",
```

```
                str(cellsize))

602             # Process: Reclassify...
                gp.Reclassify_sa(Single_Model_Raster, "VALUE", "1 1;NODATA 0",
                    Single_Model_Raster_Reclass, "DATA")
604
                # Process: PolygonToRaster...
606             gp.PolygonToRaster_conversion(Single_Model_Polygon_Agg, "
                    Raster_val", Single_Model_Raster_Agg, "CELL_CENTER", "NONE
                    ", str(cellsize))

608             # Process: Reclassify...
                gp.Reclassify_sa(Single_Model_Raster_Agg, "VALUE", "1 1;NODATA
                    0", Single_Model_Raster_Agg_Reclass, "DATA")
610
                # Process: PolygonToRaster...
612             gp.PolygonToRaster_conversion(Single_Model_Polygon_Agg_Buffer,
                    "Raster_val", Single_Model_Raster_Agg_Buffer, "
                    CELL_CENTER", "NONE", str(cellsize))

614             # Process: Reclassify...
                gp.Reclassify_sa(Single_Model_Raster_Agg_Buffer, "VALUE", "1 1
                    ;NODATA 0", Single_Model_Raster_Agg_Buffer_Reclass, "DATA"
                    )
616
                # Add Single_Model_Polygon_Agg to
                    Single_Model_Polygon_Agg_Buffer
618             gp.Plus_sa(Single_Model_Raster_Agg_Reclass,
                    Single_Model_Raster_Agg_Buffer_Reclass, Bound_Raster4)

620             # Process: Reclassify...
                gp.Reclassify_sa(Bound_Raster4, "VALUE", "0 0;1 10;2 2",
                    Bound_Raster5, "DATA")
622
                # Add Single_Model_Polygon to Bound_Raster5
624             gp.Plus_sa(Single_Model_Raster_Reclass, Bound_Raster5,
                    Bound_Raster6)

626             # Process: Reclassify...
                gp.Reclassify_sa(Bound_Raster6, "VALUE", "0 0;1 0;10 -1;11 -1;
                    2 0;3 1;NODATA 0", Boundary_Raster, "DATA")
628
                # Process: Raster to Point...
630             gp.RasterToPoint_conversion(Boundary_Raster,
                    Boundary_Centroids, "VALUE")

632             # Process: Add XY Coordinates...
                gp.AddXY_management(Boundary_Centroids)
634
                # Process: Point to Raster...
636             gp.PointToRaster_conversion(Boundary_Centroids, "POINT_X",
```

```
                Bound_X, "MOST_FREQUENT", "NONE", str(cellsize))

638             # Process: Point to Raster...
                gp.PointToRaster_conversion(Boundary_Centroids, "POINT_Y",
                    Bound_Y, "MOST_FREQUENT", "NONE", str(cellsize))
640
                # Process: Raster to ASCII...
642             gp.RasterToASCII_conversion(Boundary_Raster,
                    Boundary_ASCII_file)

644             # Process: Raster to ASCII...
                gp.RasterToASCII_conversion(Bound_X, X_Coords)
646
                # Process: Raster to ASCII...
648             gp.RasterToASCII_conversion(Bound_Y, Y_Coords)

650             # Edit X_COORDS text file
                MODFLOW_ascii(X_Coords, textFolders + "\\MODFLOW_txt\\X_Coords
                    .txt")
652
                # Edit Y_Coords text file
654             MODFLOW_ascii(Y_Coords, textFolders + "\\MODFLOW_txt\\Y_Coords
                    .txt")

656             del thickness,thickness_reclass,Extents_raster,
                    Extents_raster_reclassify,Extents_Polygon
                del Single_Model_Polygon_Agg,Single_Model_Polygon_Agg_Buffer,
                    Spec_Head_Bound_Poly,Single_Model_Raster
658             del Single_Model_Raster_Agg,Single_Model_Raster_Agg_Buffer,
                    Single_Model_Raster_Reclass
                del Single_Model_Raster_Agg_Reclass,
                    Single_Model_Raster_Agg_Buffer_Reclass,Bound_Raster4,
                    Bound_Raster5
660             del Bound_Raster6,Boundary_Raster, Boundary_Centroids,Bound_X,
                    Bound_Y


662
                #Create Boundary File for Sloping base
664             if create_SlopingBaseBoundary:
                    gp.AddToolbox(ToolboxFolder + "Conversion Tools.tbx")
666                 gp.AddToolbox(ToolboxFolder + "Data Management Tools.tbx")

668                 # Local variables...
                    AquiferCoveragePoly  = outFolder + "\\Temp\\
                        aquifer_coverage_poly.shp"
670                 SlopingBaseExtentsPoly = outFolder + "\\Temp\\
                        sloping_base_poly.shp"
                    SlopingBaseExtentsRaster = outFolder + "\\slp_bse_bound"
672                 SlopingBaseExtents_ASCII_file = textFolders + "\\txt\\
                        slopingbasebound.txt"
                    DEM_ASCII_file = textFolders + "\\txt\\dem.txt"
```

```
674
                    # Process: Raster to Polygon...
676                 gp.RasterToPolygon_conversion(AquiferCoverage_raster,
                        AquiferCoveragePoly, "NO_SIMPLIFY", "VALUE")

678             # Add Area Field
                gp.AddField_management (AquiferCoveragePoly, "Area", "FLOAT",
                    "", "", "", "", "NON_NULLABLE", "NON_REQUIRED", "")
680
                # Calculate Area
682             expression = "float(!SHAPE.AREA@SQUAREMILES!)"
                gp.CalculateField_management (AquiferCoveragePoly, "Area",
                    expression, "PYTHON_9.3")
684
                # Find, select, and save largest polygon within boundary to
                    new shapefile:
686             searchRows = gp.searchcursor(AquiferCoveragePoly)
                searchRow = searchRows.next()
688             Area = []

690             while searchRow:
                    if searchRow.GRIDCODE == 1:
692                     Area.append(searchRow.Area)
                    searchRow = searchRows.next()
694
                maxArea = max(Area)
696             area_string = "\"Area\" = " + str(maxArea)
                area_string = "Area > " + str(int(maxArea) - 1) + " AND Area <
                    " + str(int(maxArea) + 1)
698
                gp.MakeFeatureLayer(AquiferCoveragePoly,"bound_lyr")
700
                gp.SelectLayerByAttribute_management("bound_lyr", "
                    NEW_SELECTION", area_string)
702
                gp.CopyFeatures("bound_lyr", SlopingBaseExtentsPoly)
704
                # Process: PolygonToRaster...
706             gp.PolygonToRaster_conversion(SlopingBaseExtentsPoly, "
                    GRIDCODE", SlopingBaseExtentsRaster, "CELL_CENTER", "NONE"
                    , str(cellsize))

708             #Project Raster
                gp.DefineProjection_management(SlopingBaseExtentsRaster, "
                    PROJCS['NAD_1983_UTM_Zone_14N',GEOGCS['
                    GCS_North_American_1983',DATUM['D_North_American_1983',
                    SPHEROID['GRS_1980',6378137.0,298.257222101]],PRIMEM['
                    Greenwich',0.0],UNIT['Degree',0.0174532925199433]],
                    PROJECTION['Transverse_Mercator'],PARAMETER['False_Easting
                    ',500000.0],PARAMETER['False_Northing',0.0],PARAMETER['
                    Central_Meridian',-99.0],PARAMETER['Scale_Factor',0.9996],
```

```
                PARAMETER['Latitude_Of_Origin',0.0],UNIT['Meter',1.0]]")

            # Process: Raster to ASCII...
            gp.RasterToASCII_conversion(SlopingBaseExtentsRaster,
                SlopingBaseExtents_ASCII_file)
            gp.RasterToASCII_conversion(DEM, DEM_ASCII_file)

            # Edit text file
            MODFLOW_ascii(SlopingBaseExtents_ASCII_file, textFolders + "\\
                MODFLOW_txt\\SlopingBaseBound_MODFLOW.txt")
            MODFLOW_ascii(DEM_ASCII_file, textFolders + "\\MODFLOW_txt\\
                DEM_MODFLOW.txt")

            del AquiferCoveragePoly,SlopingBaseExtentsPoly,
                SlopingBaseExtentsRaster,SlopingBaseExtents_ASCII_file,
                AquiferCoverage_raster

            #Create River Head files
            if create_RiverHead or create_RiverBottom or
                create_ModelBoundary or create_Conductance:
                 print 'Creating River files'
                 try:
                     # Check out any necessary licenses
                     gp.CheckOutExtension("spatial")

                     # Load required toolboxes...
                     gp.AddToolbox(ToolboxFolder + "Spatial Analyst Tools.
                         tbx")
                     gp.AddToolbox(ToolboxFolder + "Conversion Tools.tbx")
                     gp.AddToolbox(ToolboxFolder + "Data Management Tools.
                         tbx")
                     gp.AddToolbox(ToolboxFolder + "Analysis Tools.tbx")

                     # Local variables...
                     gp.Extent = str(left) + " " + str(bottom) + " " + str(
                         right) + " " + str(top)
                     HighPlainsRivers_Clip = outFolder + "\\
                         HighPlainsRivers_Clip.shp"
                     River_Raster = outFolder + "\\Temp\\River_Raster1"
                     River_Raster_Reclass = outFolder + "\\Temp\\
                         River_Raster2"
                     DEM_Resample = outFolder + "\\Temp\\DEM_Resample"
                     RiverHead_Raster = outFolder + "\\river_head"
                     RiverBottom_Raster = outFolder + "\\Temp\\
                         riverbot_temp"
                     RiverBottom_Raster_Reclass = outFolder + "\\
                         river_bottom"
                     RiverHead_ASCII_file = textFolders + "\\txt\\riverhead
                         .txt"
                     RiverBottom_ASCII_file = textFolders + "\\txt\\
                         riverbottom.txt"
```

```
746                     # Process: Clip...
                        gp.Clip_analysis(HighPlainsRivers,
                            Single_Model_Polygon, HighPlainsRivers_Clip, "")
748
                        # Process: Feature to Raster (2)...
750                     fields = gp.ListFields(HighPlainsRivers_Clip)
                        names = []
752
                        for field in fields:
754                         names.append(field.Name)

756                     if "IsRiver" in names:
                            gp.FeatureToRaster_conversion(
                                HighPlainsRivers_Clip, "IsRiver", River_Raster
                                , str(cellsize))
758                     else:
                            gp.AddField(HighPlainsRivers_Clip, "IsRiver", "
                                short")
760                         gp.CalculateField_management (
                                HighPlainsRivers_Clip, "IsRiver", "1", "
                                PYTHON_9.3")
                            gp.FeatureToRaster_conversion(
                                HighPlainsRivers_Clip, "IsRiver", River_Raster
                                , str(cellsize))
762
                        # Process: Reclassify...
764                     gp.Reclassify_sa(River_Raster, "VALUE", "1 1;NODATA 0"
                            , River_Raster_Reclass, "DATA")

766                     # Process: Times...
                        gp.Times_sa(River_Raster_Reclass, DEM,
                            RiverHead_Raster)
768
                        #Process: Minus...
770                     gp.Minus_sa(RiverHead_Raster, "3", RiverBottom_Raster)

772                     # Process: Reclassify...
                        gp.Reclassify_sa(RiverBottom_Raster, "VALUE", "-3 0",
                            RiverBottom_Raster_Reclass, "DATA")
774
                        # Process: Raster to ASCII...
776                     gp.RasterToASCII_conversion(RiverHead_Raster,
                            RiverHead_ASCII_file)

778                     # Process: Raster to ASCII...
                        gp.RasterToASCII_conversion(RiverBottom_Raster_Reclass
                            , RiverBottom_ASCII_file)
780
                        del River_Raster,River_Raster_Reclass,DEM_Resample
782
```

```
                except:
                    raise

            #Create Conductance files
            if create_Conductance or create_ModelBoundary or
                create_RiverHead:
                print 'Creating Conductance files'
                try:
                    # Load required toolboxes...
                    gp.AddToolbox(ToolboxFolder + "Conversion Tools.tbx")

                    #Local Variables
                    gp.Extent = str(left) + " " + str(bottom) + " " + str(
                        right) + " " + str(top)
                    Conductance_Raster = outFolder + "\\conductance"
                    Conductance_ASCII_file = textFolders + "\\txt\\
                        conductance.txt"

                    # Process: Feature to Raster...
                    gp.FeatureToRaster_conversion(HighPlainsRivers_Clip, "
                        Conduc", Conductance_Raster, str(cellsize))

                    # Process: Raster to ASCII...
                    gp.RasterToASCII_conversion(Conductance_Raster,
                        Conductance_ASCII_file)

                except:
                    raise

            #Clean up River Head and Conductance text files
            CleanBoundaryTxt(textFolders + "\\txt\\",textFolders + "\\txt
                \\","boundary.txt")

            # Process: Conductance ASCII to Raster...
            gp.ASCIIToRaster_conversion(Conductance_ASCII_file,
                Conductance_Raster, "INTEGER")

            # Process: RiverHead ASCII to Raster...
            gp.ASCIIToRaster_conversion(RiverHead_ASCII_file,
                RiverHead_Raster, "INTEGER")

            # Process: RiverBottom ASCII to Raster...
            gp.ASCIIToRaster_conversion(RiverBottom_ASCII_file,
                RiverBottom_Raster_Reclass, "INTEGER")

            # Edit Boundary text file
            MODFLOW_ascii(Boundary_ASCII_file, textFolders + "\\
                MODFLOW_txt\\Boundary_MODFLOW.txt")

            # Edit Conductance text file
            MODFLOW_ascii(Conductance_ASCII_file, textFolders + "\\
```

```
                MODFLOW_txt\\Conductance_MODFLOW.txt")

            # Edit RiverHead text file
            MODFLOW_ascii(RiverHead_ASCII_file, textFolders + "\\
                MODFLOW_txt\\RiverHead_MODFLOW.txt")

            # Edit RiverBottom text file
            MODFLOW_ascii(RiverBottom_ASCII_file, textFolders + "\\
                MODFLOW_txt\\RiverBottom_MODFLOW.txt")

            del Conductance_Raster,Conductance_ASCII_file,RiverHead_Raster
                ,RiverHead_ASCII_file,RiverBottom_Raster,
                RiverBottom_ASCII_file,HighPlainsRivers_Clip,
                RiverBottom_Raster_Reclass

        except:
            raise

    #Create Well Files
    if create_Wells:
        print 'Creating Well files'
        try:
            shutil.copy2(textFolders + "\\MODFLOW_txt\\RiverHead_MODFLOW.
                txt", textFolders + "\\MODFLOW_txt\\Wells_MODFLOW.txt")

        except:
            raise
            gp.GetMessage(2)

    finish = time.clock()
    print 'model run time:',finish-start,'seconds'
```

A.1.3 FishnetFilesGen.py

The script FishnetFilesGen.py calls Fishnet.py which does the actual work of creating the fishnets grids. The user needs to enter a file path where the fishnets will be stored as well as the file path to the folder that contains the rasters that will be used to generate the fishnets. The user needs to add the names of the rasters that will be turned into fishnet grids to the variable *file_list*. The user also needs to enter the names of the resulting fishnet grids in the variable *Poly_Names*. The first entry in *Poly_Names* needs to correspond to the first entry in *file_list* as do the second entries, third entries etc... The width and height of the grid cells to be used in the fishnet also need to be entered.

```
#----------------------
FishnetFilesGen.py
#----------------------

from Fishnet import *
import time
timeS = time.clock()

#Enter Fishnet Model Parameters

#Enter information for Workspace, GIS_Folder, file_list, Poly_Names,
    Cell_Width, and Cell_Height
#"""High Plains Model
Workspace = 'C:/Documents and Settings/andya/My Documents/Python/Py2.6/
    GW_Optimization/Fishnet/Fishnet/HighPlains' #This is where the fishnet
     folder and Geodatabase will be created
GIS_Folder = 'C:/Documents and Settings/andya/My Documents/Python/Py2.6/
    GW_Optimization/New Folder/HighPlainsClean/1000/GIS/'  #Folder
    containing rasters to be used in fishnet

file_list = ['slp_bse_bound','bedrock_m','conductance','k_md','pred_wlv_m'
    ,'recharge_md','river_head','spyd','dem'] #Names of the rasters in the
     workspace to be used in creation of fishnets
Poly_Names = ['Boundary','Bedrock','Conductance','K','PredWlv','Recharge',
    'RiverHead','SPYD','SurfaceElev'] #names of polygons to be created,
    must be in same order as file_array
Cell_Width = 2000  #meters
Cell_Height = 10000  #meters
#"""


raster_list = []
for file in file_list:
    raster_list.append(Workspace + "/GIS/" + file)

file_list = raster_list
GDB_Name = str(Cell_Width) + 'X' + str(Cell_Height)
Fishnet(file_list,Workspace,GDB_Name,Poly_Names,Cell_Width,Cell_Height)

timeF = time.clock()
print "Simulation Time =",(timeF-timeS)/60,"minutes"
```

A.1.4 Fishnet.py

The Fishnet.py script takes the user inputs from FishnetFilesGen.py and uses them to create fishnet at the user specified grid size for each raster given in the FishnetFilesGen.py script and saves the fishnet grids as shapefiles in a geodatabase in the user specified output folder.

```
#----------------------
Fishnet.py
#----------------------

#Import system modules
import sys, string, os, arcgisscripting, math

def Fishnet(file_array,Workspace,GDB_Name,Poly_Names,Cell_Width,
    Cell_Height):
    #Create folder to store files if DNE
    try:
        os.makedirs(Workspace + "/Fishnet")
        fishnet_Workspace = Workspace + "/Fishnet/"

    except OSError:
        fishnet_Workspace = Workspace + "/Fishnet/"

    #Create the Geoprocessor and set overwrite to true
    gp = arcgisscripting.create(9.3)
    gp.CheckOutExtension("Spatial")
    gp.OverwriteOutput = True

    #Process script arguments and derive some variables
    GDB = GDB_Name + ".gdb"
    FDS_Name = "AggregateCells" #Provide a default value if unspecified
    FDS = GDB + "/" + FDS_Name
    Input_Grid = file_array[0]

    #Set the Workspace
    gp.Workspace = fishnet_Workspace

    #Determine spatial reference of Input_Grid for feature data set
    desc = gp.Describe(Input_Grid)
    SpRef = desc.SpatialReference

    #Verify that the Input_Grid is in projected space.
    if SpRef.Type != "Projected":
        print "Input raster data is not a projected data set."
        sys.exit()

    #Create the Geodatabase and empty Feature Data Set.
    print "Creating Geodatabase " + GDB_Name + " with Feature Data Set " +
        FDS_Name + "..."
    gp.CreateFileGDB_management(fishnet_Workspace,GDB_Name)
    gp.CreateFeatureDataset_management(GDB,FDS_Name,SpRef)

    #Specify names for feature classes of Fishnet Lines and Lables.
    FishnetFC = FDS +"/Fishnet"
    LabelsFC = FDS + "/Fishnet_label"

    #Describe Input_Grid and derive extents.
```

```
    desc = gp.Describe(Input_Grid)
    Extent = desc.Extent
    OriginPt = str(Extent.XMin) + " " + str(Extent.YMin)
    AxisPt = str(Extent.XMin) + " " + str(Extent.YMin + 10)

    #Calculate rows and columns needed for fishnet
    Rows = int(math.ceil((Extent.YMax - Extent.YMin)/int(Cell_Height)))
    Cols = int(math.ceil((Extent.XMax - Extent.XMin)/int(Cell_Width)))

    #Create fishnet lines and labels
    print "Creating Fishnet with " + str(Rows) + " rows and " + str(Cols)
       + " columns at Origin Point:" + str(OriginPt) + "..."
    gp.CreateFishnet_management(FishnetFC, OriginPt, AxisPt, Cell_Width,
       Cell_Height, Rows, Cols, "#", "labels", Input_Grid)

    #Make feature layers of the Fishnet and Labels feature classes.
    FishnetLyr = "FishnetLayer"
    LabelsLyr = "LabelsLayer"
    gp.MakeFeatureLayer_management(FishnetFC, FishnetLyr)
    gp.MakeFeatureLayer_management(LabelsFC, LabelsLyr)

    #Add X,Y coordinates to labels
    print "Getting coordinates for cells..."
    gp.AddXY_management(LabelsLyr)

    #Create Polygons feature class from fishnet lines and labels
    iter = 0
    for raster in file_array:
        PolysFC_Name = Poly_Names[iter]
        PolysFC = FDS + "/" + PolysFC_Name
        Input_Grid = raster
        print "Creating " + PolysFC_Name
        gp.FeatureToPolygon_management(FishnetLyr,PolysFC,"#","Attributes"
           ,LabelsLyr)

        #Add ZONE_ID number and set it equal to Object ID
        print "Assigning Zone IDs..."
        gp.AddField_management(PolysFC,"ZONE_ID","long")
        gp.CalculateField_management(PolysFC,"ZONE_ID","[OBJECTID]")

        #Calculate zonal statistics as a table.
        print "Calculating zonal statistics..."
        Stat_Table = GDB + "\ZonalStats"
        gp.ZonalStatisticsAsTable_sa(PolysFC,"ZONE_ID",Input_Grid,
           Stat_Table,"DATA")

        #Join ZonalStats table to PolysFC
        print "Joining tables"
        gp.JoinField_management(PolysFC,"ZONE_ID",Stat_Table,"VALUE")

        iter += 1
```

```
98      #Create Polygons for Head,Depth to Water, and Saturated Thickness
            calculations
        calcs=['Head','ST','DTW','Error']
100     for name in calcs:
            PolysFC = FDS + "/" + name
102         print "Creating " + name
            gp.FeatureToPolygon_management(FishnetLyr,PolysFC,"#","Attributes"
                ,LabelsLyr)
104         gp.AddField_management(PolysFC,str(name),"float")

106
        #Delete variables, releasing memory.
108     del gp, Workspace, GDB_Name, GDB, FDS_Name, FDS, PolysFC_Name, PolysFC
            , Input_Grid, Cell_Width, Cell_Height
        del desc, SpRef, FishnetFC, LabelsFC, Extent, OriginPt, AxisPt, Rows,
            Cols, FishnetLyr, LabelsLyr, Stat_Table
110
        print "Finished!"
```

A.1.5 SlopingBaseCSV.py

The SlopingBaseCSV.py script is used to create a text file that contains all the data represented by the fishnets generated by Fishnet.py. In the script the user enters the dimensions used to create a previous fishnet grid, the file path of the folder containing the geodatabase populated with the fishnets and the file path of a folder to store the output text file in. The names of the data to be included in the output text file need to also be specified. Defaults of *Boundary, xid, yid, X, Y, K, Recharge, Bedrock, PredWlv, SurfaceElev_avg, SurfaceElev_min* are given. *Boundary* is a column of 0's and 1's where zeroes represent cells outside the model extents determined earlier by the *AquiferCoverage* variable defined in MODFLOWFilesGen.py and 1's represent the cells inside the area to be modeled. Coordinate data are represented by *xid, yid, X, and Y* where *xid and yid* are the column and row numbers respectively of the fishnet grid and *X and Y* are the latitude and longitude coordinates. *Bedrock, PredWlv, SurfaceElev_avg, and SurfaceElev_min* are bedrock elevation, predevelopment water table elevation, land surface average elevation for the cell, and land surface minimum elevation for the cell.

Table A.1: *Example of SlopingBaseCSV.py output text file*

Boundary	xid	yid	X	Y	K	Recharge	Bedrock	PredWlv	SurfaceElev_avg	SurfaceElev_min
0	66	13	54668	3631615	17	1.21E-05	1211.3	1213.253418	1150.60376	1118.925171
0	67	13	56668	3631615	17	1.19E-05	1209.14	1210.527344	1155.880005	1123.754761
0	68	13	58668	3631615	17	1.18E-05	1206.93	1208.165649	1162.304199	1123.297241
0	69	13	60668	3631615	17	1.18E-05	1204.4	1206.078125	1169.440308	1127.847534
0	70	13	62668	3631615	17	1.17E-05	1201.19	1203.902466	1176.453979	1130.680054
0	71	13	64668	3631615	16	1.17E-05	1197.02	1201.271362	1187.808472	1139.579956
0	72	13	66668	3631615	15	1.17E-05	1191.76	1198.146606	1198.782471	1145.613403
0	73	13	68668	3631615	14	1.18E-05	1185.42	1194.851929	1216.289551	1158.144897
0	74	13	70668	3631615	13	1.18E-05	1178.31	1192.0354	1225.377686	1170.130493
0	75	13	72668	3631615	12	1.19E-05	1170.92	1189.615356	1227.545776	1197.943359
1	76	13	74668	3631615	11	1.19E-05	1163.96	1186.782227	1220.740479	1206.013428
1	77	13	76668	3631615	11	1.20E-05	1158.5	1183.495605	1213.389771	1199.230469
1	78	13	78668	3631615	11	1.22E-05	1154.17	1180.449951	1206.261597	1185.679077
1	79	13	80668	3631615	11	1.24E-05	1149.21	1177.377563	1199.139771	1180.685059
1	80	13	82668	3631615	11	1.26E-05	1142.67	1173.720215	1191.431885	1170.030273
1	81	13	84668	3631615	11	1.29E-05	1134.83	1169.447388	1185.212036	1170.823853
1	82	13	86667	3631615	18	1.33E-05	1126.46	1165.584106	1180.232666	1166.116577
1	83	13	88668	3631615	23	1.38E-05	1117.95	1162.043701	1174.706543	1164.734009
1	84	13	90668	3631615	23	1.43E-05	1109.7	1157.567383	1168.952881	1157.248901
1	85	13	92668	3631615	23	1.49E-05	1102.61	1152.736328	1162.956787	1147.831055
1	86	13	94668	3631615	23	1.57E-05	1098.27	1146.970093	1156.712769	1144.53125
1	87	13	96668	3631615	23	1.65E-05	1096.55	1140.713501	1148.444336	1136.500488

```
#----------------------
SlopingBaseCSV.py
#----------------------

import arcgisscripting, os, sys, csv

#Enter cell dimensions and output folder name
cellDimensions='2000x10000'  #Fishnet Dimensions width x height
OutputFolder = 'HighPlains'  #Name of Output Folder

#High Plains Model
#Enter path to geodatabase created by Fishnet.py, this is the geodatabase
    that contains the fishnet shapefiles
workspace = "C:/Documents and Settings/andya/My Documents/Python/Py2.6/
    GW_Optimization/Fishnet/Fishnet/" + OutputFolder + "/" +
    cellDimensions + "/" + cellDimensions + ".gdb"

#Enter path for the csv file to be saved to.
csv_file = open("C:/Documents and Settings/andya/My Documents/Python/Py2.6
    /GW_Optimization/Fishnet/Fishnet/" + OutputFolder + "/" +
    cellDimensions + "/" + cellDimensions + ".txt",'wb')



#Data to be included in csv
headers = ['Boundary','xid','yid','X','Y','K','Recharge','Bedrock','
    PredWlv','SurfaceElev_avg','SurfaceElev_min']

gp = arcgisscripting.create(9.3)
gp.Workspace = workspace
datasets = gp.ListDatasets()

for dataset in datasets:
    gp.Workspace = workspace + "/" + dataset + "/"
    fcs = gp.ListFeatureClasses()
    write_xy = True
    X = []
    Y = []
    unique_X = []
    unique_Y = []
    FC_values = {}
    fishnetFC = {}
    skip = ['Fishnet','Fishnet_label','Head','DTW','ST','Error','
        Extraction']
    for featureclass in fcs:
        if featureclass in skip:
            None

        elif featureclass == 'SurfaceElev':
            print 'Reading ' + str(featureclass) + ' Values'
```

```
            searchRows = gp.searchcursor(featureclass)
            searchRow = searchRows.next()
            header1 = str(featureclass) + '_avg'
            header2 = str(featureclass) + '_min'
            list1 = []
            list2 = []

            while searchRow:
                if searchRow.MEAN == None:
                    list1.append(0)
                else:
                    list1.append(searchRow.MEAN)

                if searchRow.MIN == None:
                    list2.append(0)
                else:
                    list2.append(searchRow.MIN)
                searchRow = searchRows.next()

            FC_values[header1] = list1
            FC_values[header2] = list2

        else:
            print 'Reading ' + str(featureclass) + ' Values'
            searchRows = gp.searchcursor(featureclass)
            searchRow = searchRows.next()

            if write_xy == True:
                header1 = 'xid'
                header2 = 'yid'
                header3 = 'X'
                header4 = 'Y'
                header5 = str(featureclass)
                list1 = []
                list2 = []
                list3 = []
                list4 = []
                list5 = []

                while searchRow:
                    list3.append(searchRow.POINT_X)
                    list4.append(searchRow.POINT_Y)
                    X.append(searchRow.POINT_X)
                    Y.append(searchRow.POINT_Y)

                    if searchRow.MEAN == None:
                        list5.append(0)
                    else:
                        list5.append(searchRow.MEAN)

                    searchRow = searchRows.next()
```

```
95
                    FC_values[header3] = list3
97                  FC_values[header4] = list4
                    FC_values[header5] = list5
99
                    for Y_val in Y:
101                     if Y_val not in unique_Y:
                            unique_Y.append(Y_val)
103                 for X_val in X:
                        if X_val not in unique_X:
105                         unique_X.append(X_val)

107                 unique_X.sort()
                    unique_Y.sort()
109
                    for iter in xrange(len(Y)):
111                     list1.append(unique_X.index(X[iter]) + 1)
                        list2.append(unique_Y.index(Y[iter]) + 1)
113
                    FC_values[header1] = list1
115                 FC_values[header2] = list2
                    write_xy = False
117
                else:
119                 header1 = str(featureclass)
                    list1 = []
121
                    while searchRow:
123                     if searchRow.MEAN == None:
                            list1.append(0)
125                     else:
                            list1.append(searchRow.MEAN)
127                     searchRow = searchRows.next()

129                 FC_values[header1] = list1

131 keys = FC_values.keys()
    nvals = len(FC_values[keys[0]])
133 ncols = len(keys)
    output = [ [] for _ in xrange(nvals + 1)]
135

137 # Account for possibility of data not overlapping perfectly at edges
        resulting in zeros in areas within the boundary
    count = 0
139 for bed in FC_values['Bedrock']:
        if bed < 1:
141         FC_values['Boundary'][count] = 0
        count += 1
143
```

```
for header in headers:
    output[0].append(header)

for header in headers:
    temp=[]
    for item in FC_values[header]:
        if header == 'Boundary' or header == 'xid'or header == 'yid'or
           header == 'X'or header == 'Y':
            temp.append(int(item))
        else:
            temp.append(item)
    fishnetFC[header]=temp


i=1
while i <= nvals :
    for header in headers:
        output[i].append(fishnetFC[header][i-1])
    i += 1


print 'Writing values to .txt'
output_writer = csv.writer(csv_file, delimiter = ' ')
for item in output:
    output_writer.writerow(item)

print 'Finished'
```

A.1.6 CSV_to_2dArray.py

The CSV_to_2dArray.py script takes the text file output from SlopingBaseCSV.py and turns each column of that text file into a separate text file properly formatted for use with MODFLOW models and sloping base models. The user needs to specify the location of the folder to store the text files in as well as the path to the text file created previously by SlopingBaseCSV.py.

```
#----------------------
CSV_to_2dArray.py
#----------------------

import numpy
from FileReader_recordArray import *
from Array2txt import *
from MODFLOWAscii import *

#Enter cell dimensions and output folder
```

```
cellDimensions='2000x10000'
cell_width = 2000
OutputFolder = 'HighPlains'

#Enter folder where text file outputs should be saved
outDirec = "C:/Documents and Settings/andya/My Documents/Python/Py2.6/GW_Optimization/Fishnet/Fishnet/" + OutputFolder + "/" + cellDimensions
#Enter path to folder containing fishnet csv file
csv_File="C:/Documents and Settings/andya/My Documents/Python/Py2.6/GW_Optimization/Fishnet/Fishnet/" + OutputFolder + "/" + cellDimensions + "/" + cellDimensions + ".txt"


My_Data_Type=numpy.dtype([("boundary","int"),("xid","int"),("yid","int"),("X","float"),("Y","float"),("K","float"),("Recharge","float"),("bedrock","float"),("predwlv","float"),("surface_elev_avg","float"),("surface_elev_min","float")])
data=read_recordArray(csv_File,My_Data_Type,skip=1,missing='',separator=None)
txtOutput = outDirec + '/txt/'
nrows=max(data['yid'])
ncols=max(data['xid'])

boundary = numpy.ones((nrows,ncols),"float64")*-9999
xid = numpy.ones((nrows,ncols),"float64")*-9999
yid = numpy.ones((nrows,ncols),"float64")*-9999
X = numpy.ones((nrows,ncols),"float64")*-9999
Y = numpy.ones((nrows,ncols),"float64")*-9999
K = numpy.ones((nrows,ncols),"float64")*-9999
R = numpy.ones((nrows,ncols),"float64")*-9999
bedrock = numpy.ones((nrows,ncols),"float64")*-9999
predwlv = numpy.ones((nrows,ncols),"float64")*-9999
surface_elev_avg = numpy.ones((nrows,ncols),"float64")*-9999
surface_elev_min = numpy.ones((nrows,ncols),"float64")*-9999


col = 0
row = nrows - 1
for item in data:
    boundary[row][col] = item[0]
    xid[row][col] = item[1]
    yid[row][col] = item[2]
    X[row][col] = item[3]
    Y[row][col] = item[4]
    K[row][col] = item[5]
    R[row][col] = item[6]
    bedrock[row][col] = item[7]
    predwlv[row][col] = item[8]
    surface_elev_avg[row][col] = item[9]
    surface_elev_min[row][col] = item[10]
```

```
    if col == ncols -1:
        col = 0
        row = row - 1
    else:
        col = col + 1

header=[ncols,nrows,0,0,cell_width,-9999]

Array2txt(boundary, header, 'slopingbasebound' , txtOutput)
Array2txt(xid, header, 'xid' , txtOutput)
Array2txt(yid, header, 'yid' , txtOutput)
Array2txt(X, header, 'X' , txtOutput)
Array2txt(Y, header, 'Y' , txtOutput)
Array2txt(K, header, 'K' , txtOutput)
Array2txt(R, header, 'recharge' , txtOutput)
Array2txt(bedrock, header, 'bedrock' , txtOutput)
Array2txt(predwlv, header, 'predwlv' , txtOutput)
Array2txt(surface_elev_avg, header, 'surface_elev_avg' , txtOutput)
Array2txt(surface_elev_min, header, 'surface_elev_min' , txtOutput)

boundary_txt = txtOutput+ 'slopingbasebound.txt'
xid_txt = txtOutput+ 'xid.txt'
yid_txt = txtOutput+ 'yid.txt'
X_txt = txtOutput + 'X.txt'
Y_txt = txtOutput + 'Y.txt'
K_txt = txtOutput + 'K.txt'
R_txt = txtOutput + 'recharge.txt'
bedrock_txt = txtOutput + 'bedrock.txt'
predwlv_txt = txtOutput + 'predwlv.txt'
surface_elev_avg_txt = txtOutput + 'surface_elev_avg.txt'
surface_elev_min_txt = txtOutput + 'surface_elev_min.txt'

MODFLOW_ascii(boundary_txt, outDirec + "/MODFLOW_txt/
    SlopingBaseBound_MODFLOW.txt")
MODFLOW_ascii(xid_txt, outDirec + "/MODFLOW_txt/xid_MODFLOW.txt")
MODFLOW_ascii(yid_txt, outDirec + "/MODFLOW_txt/yid_MODFLOW.txt")
MODFLOW_ascii(X_txt, outDirec + "/MODFLOW_txt/X_MODFLOW.txt")
MODFLOW_ascii(Y_txt, outDirec + "/MODFLOW_txt/Y_MODFLOW.txt")
MODFLOW_ascii(K_txt, outDirec + "/MODFLOW_txt/K_MODFLOW.txt")
MODFLOW_ascii(R_txt, outDirec + "/MODFLOW_txt/Recharge_MODFLOW.txt")
MODFLOW_ascii(bedrock_txt, outDirec + "/MODFLOW_txt/Bedrock_MODFLOW.txt")
MODFLOW_ascii(predwlv_txt, outDirec + "/MODFLOW_txt/PredWlv_MODFLOW.txt")
MODFLOW_ascii(surface_elev_avg_txt, outDirec + "/MODFLOW_txt/
    Surface_Elev_Avg_MODFLOW.txt")
MODFLOW_ascii(surface_elev_min_txt, outDirec + "/MODFLOW_txt/
    Surface_Elev_Min_MODFLOW.txt")

print "Finished!"
```

A.1.7 Additional Required Scripts

The following scripts contain useful functions that are called multiple times by the other scripts listed in the appendix. These scripts do not require user input.

FileReader_recordArray.py

The FileReader_recordArray script reads in a text file and outputs a numpy record array. In this case it was used to read in the text file produced by SlopingBaseCSV.py so the individual columns can be separated into 2D arrays.

```
#----------------------
FileReader_recordArray.py
#----------------------

def read_recordArray(filename, dtype, skip=1, missing='', separator=',',
    read=0):
    """
        Generalized input routine from http://www.scipy.org/Cookbook/
            InputOutput
        Modified by SM Welch to handle missing data (10 Aug 10)
        Modified by W Kusnierczyk to handle header lines (20 Aug 10)

        Read a file with an arbitrary number of columns.
        The type of data in each column is arbitrary
        It will be cast to the given dtype at runtime

        Arguments:
            filename - name of file to be read (including path if needed)
            dtype - numpy data type specifier
            skip - number of header lines to skip (optional; default 1)
            missing - value to use for missing data (optional; default '')
            separator - character separating data items in a row (optional
                ; default ',')
        Outputs
            a numpy record array of tuples (one per row)

        Requirements
            numpy must be available
    """
    import numpy

    cast = numpy.cast
    data = [[] for dummy in xrange(len(dtype))]
    In_file = open(filename, 'r')
    for dummy in xrange(skip): In_file.readline()
    if read == 0:
```

```
        for line in In_file:
            fields = line.strip().split(separator)
            for i, number in enumerate(fields):
                if number == '': number = missing
                data[i].append(number)
        for i in xrange(len(dtype)):
            data[i] = cast[dtype[i]](data[i])
        return numpy.rec.array(data, dtype=dtype)
    else:
        c1=0
        for line in In_file:
            if c1 < read:
                fields = line.strip().split(separator)
                for i, number in enumerate(fields):
                    if number == '': number = missing
                    data[i].append(number)
                c1 += 1
        for i in xrange(len(dtype)):
            data[i] = cast[dtype[i]](data[i])
        return numpy.rec.array(data, dtype=dtype)
    In_file.close()
```

Array2txt.py

The Array2txt script takes the 2D arrays and writes them to text files.

```
#----------------------
Array2txt.py
#----------------------

import numpy, os

def Array2txt(array, header, propertyName, txtOutput):
    print "Creating " + str(propertyName) + " text file"

    ncols = header[0]
    nrows = header[1]
    xllcorner = header[2]
    yllcorner = header[3]
    cell_width = header[4]
    NODATA_value = header[5]

    filepath = txtOutput + str(propertyName) + ".txt"
    if os.path.exists(filepath):
        append = 0
        while os.path.exists(filepath):
            append += 1
            filepath = txtOutput + str(propertyName) + "_" + str(append) +
                ".txt"
        numpy.savetxt(filepath, array, fmt="%5.10f")
```

```
24
        outfile=open(filepath,'r+')
26      old = outfile.read()
        outfile.seek(0)
28      outfile.write('ncols          ' + str(ncols) + '\n')
        outfile.write('nrows          ' + str(nrows) + '\n')
30      outfile.write('xllcorner      ' + str(xllcorner) + '\n')
        outfile.write('yllcorner      ' + str(yllcorner) + '\n')
32      outfile.write('cellsize       ' + str(cell_width) + '\n')
        outfile.write('NODATA_value   ' + str(NODATA_value) + '.0000' + '\n
            ' + old)
34      outfile.close()

36  else:
        numpy.savetxt(filepath, array, fmt="%5.10f")
38      outfile=open(filepath,'r+')
        old = outfile.read()
40      outfile.seek(0)
        outfile.write('ncols          ' + str(ncols) + '\n')
42      outfile.write('nrows          ' + str(nrows) + '\n')
        outfile.write('xllcorner      ' + str(xllcorner) + '\n')
44      outfile.write('yllcorner      ' + str(yllcorner) + '\n')
        outfile.write('cellsize       ' + str(cell_width) + '\n')
46      outfile.write('NODATA_value   ' + str(NODATA_value) + '.0000' + '\n
            ' + old)
        outfile.close()
48      append = ''
```

MODFLOWAscii.py

The MODFLOWAscii script takes the text file created by the *Raster to Ascii* tool in ArcGIS and formats the header for use with PMWIN and mflab MODFLOW processors.

```
#----------------------
2 MODFLOWAscii.py
#----------------------
4
# ----------------------------------------------------------------
6 # This script processes the ASCII file from ArcGIS to the format required
    by PMWIN and mflab
# The processing procedures include (1) only keep the column and row
    number in the first two lines
8 # (2) delete the next four lines
# (3) replace the no data value -9999 with 0
10 # ----------------------------------------------------------------
# Import system modules
12 import sys, string, os

14 # Get the input file
```

```
def MODFLOW_ascii(InputFile, OutputFile):
    myInFile = open(InputFile, 'r')
    myOutFile = open (OutputFile, 'w')
    for i in range(1,3):
        line = myInFile.readline()
        # only write the column and row number into the new file
        l = line.split()
        myOutFile.write(l[1])
        myOutFile.write('  ')

    myOutFile.write('\n')

    # skip the next four lines
    for i in range(3,7):
        myInFile.readline()

    # copy the rest to the new file
    line = myInFile.readline()
    while (line != ''):
        # replace -9999 with 0
        myOutFile.write(line.replace('-9999', '0'))
        line = myInFile.readline()

    # close the two files
    myInFile.close()
    myOutFile.close()
```

Appendix B

Sloping Base Model and Optimization Python Scripts

This appendix contains the scripts used to run the sloping base model of the High Plains aquifer as well as the scripts used to optimize the model and locate the points of groundwater-surface water interaction. The flowchart in figure B.1 illustrates the basic order of execution of the model scripts as well as the flow of information and the outputs from the scripts.

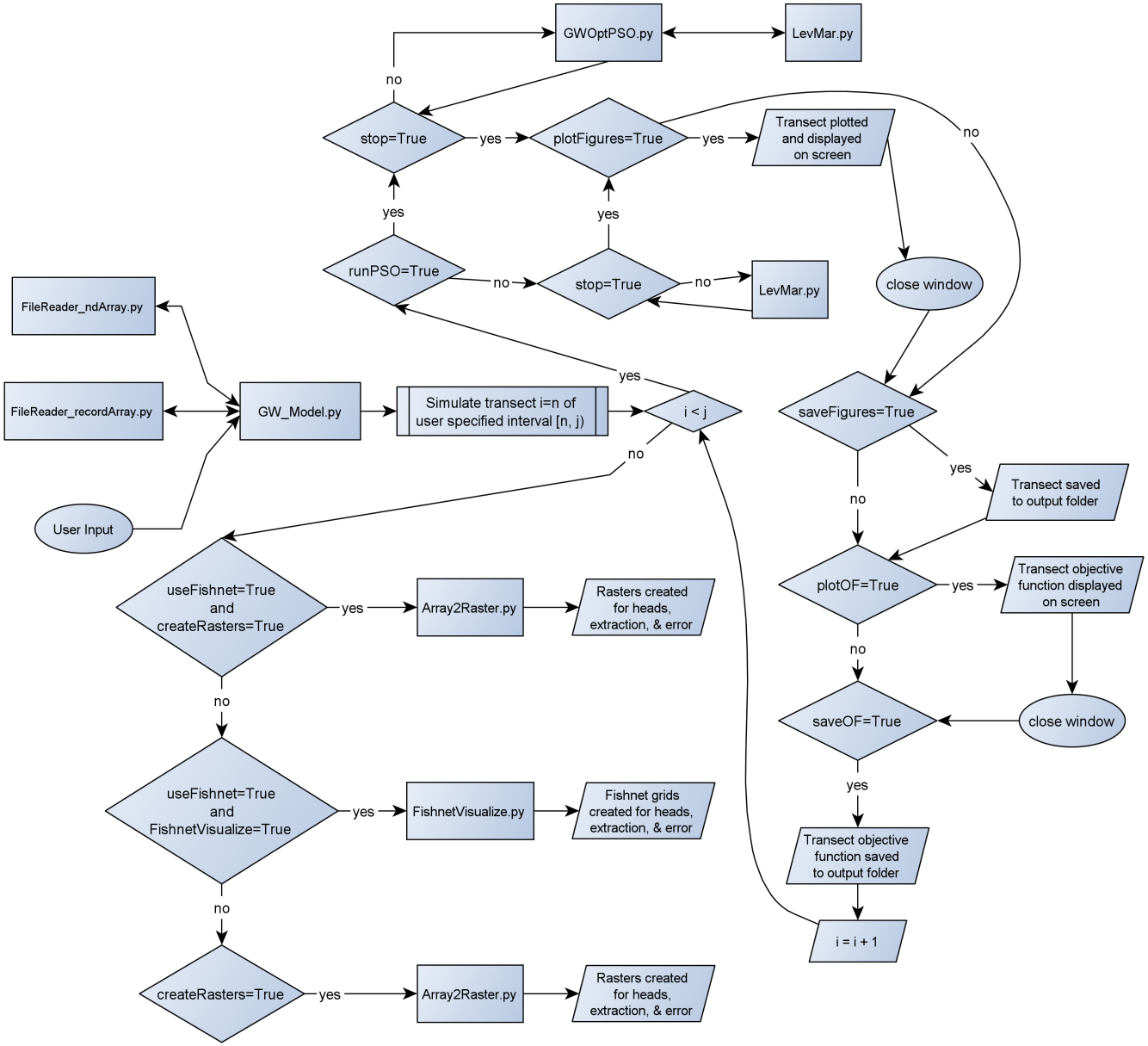

Figure B.1: *Sloping Base Model and Optimization Python Scripts Flowchart*

B.0.8 GW_Model.py

The GW_Model.py script is the primary script used in the sloping base model, all other scripts are called from this script. It does the work of loading the input data, running the sloping base model, running the optimization routine, and creating output text files, plotted figures, rasters, and fishnets of the simulated results. There are several user inputs at the beginning of the script between lines 28 and 52. The usage of each input is described by a comment in the script itself. These inputs define the input and output files and folders that will be used and created. The maximum number of Levenberg-Marquardt iterations is also defined here. If PSO is being used to optimize the model then those settings are between lines 168 and 188. The usage of each of these inputs is described by a comment in the script.

```
#----------------------
GW_Model.py
#----------------------

import os, time
from FileReader_ndArray import *
from FileReader_recordArray import *
from LevMar import *
import numpy
from math import *
import matplotlib.pyplot as plt
from Array2Raster import *
from GWOptPSO import *
from FishnetVisualize import *

start = time.clock()

figsize=[15,10]
params = {'font.size': 20,
          'legend.fontsize': 20,
          'axes.labelsize': 20,
          'xtick.labelsize': 18,
          'ytick.labelsize': 18,
          'lines.markersize': 8,
          'lines.linewidth': 2,
          'figure.figsize': figsize}
plt.rcParams.update(params)

#Enter Model Name, Base Name, output geodatabase name, fishnet folder, &
    Cell Size
```

```
modelName = 'HighPlains'  #The name of the model folder where the input text files are located.
baseName = 'HighPlains'  #This is the name that will be used to name output rasters
outputGeoDatabaseName = 'ResultsFishnet.mdb'  #Name of geodatabase to store fishnet results in
fishnetFolder = 'HighPlains'  #Name of folder containing the output geodatabase
fishnetDim = '2000x10000'  #dimensions of cells in the fishnet grid being used in the model entered as a text string (width X height)
modelCellWidth = 2000  #width of grid cells

#Enter maximum iterations to use for Levenber-Marquardt Optimization.
maxIter = 150
#Enter minimum transect length. Transects with a total number of cells < minTransect length will not be simulated.
minTransectLen = 10

useFishnet = True  #Set to true if using fishnet grid rather than raster grid for model data
FishnetVisualize = False  #Set to true to generate fishnet grids of model results (head, extraction, error)
useMinDEM = True   #Set to true to use the minimum elevation in each fishnet grid cell when searching for potential extraction points based on depth to water. If false, use average elevation of entire cell. Only works with fishnet.

runPSO = True   #If true PSO routine used for optimization. If false Levenberg-Marquardt routine used for optimization.
initilizeE = False  #Set to true to give initial guess for Levenberg-Marquardt. Not used in PSO.
plotFigures = False  #Set to true to display model results of each simulated transect as they finish.
saveFigures = True   #Set to true to automatically save plots
plotOF = False  #Set to true to display the progression of the objective function throughout the optimization iterations.
saveOF = True   #Set to true to automatically save the objective function plot
createRasters = False  #Set to true to create rasters of results (head, extraction, error). Can only be used in model is run on grid with square cells.

if saveFigures:
    figureDirectory = "C:/Documents and Settings/andya/Desktop/Figures/PSO/"

if useFishnet:
    modelDirec = "C:/Documents and Settings/andya/My Documents/Python/Py2.6/GW_Optimization/Fishnet/Fishnet/" + fishnetFolder + "/"
    textFileFolder = modelDirec + fishnetDim + "/MODFLOW_txt/"
else:
```

```
    cwd = os.getcwd()
    textFileFolder = cwd + '\\MODFLOW_texts\\' + modelName + '\\' + str(
        modelCellWidth) + '\\'

#Read data from unformatted .txt to get header info
Dim_Data_Type = numpy.dtype([("dim", "str"),("dimval","float")])
dim = read_recordArray(textFileFolder + 'header.txt', Dim_Data_Type, skip=
    0, missing='', separator=None, read=6)

cols = dim[0]
rows = dim[1]
xllcorner = dim[2]
yllcorner = dim[3]
cellsize = dim[4]
NODATA_value = dim[5]

ncols = int(cols[1])
nrows = int(rows[1])
xllcorner = xllcorner[1]
yllcorner = yllcorner[1]
cellsize = cellsize[1]
NODATA_value = int(NODATA_value[1])

header = [ncols,nrows,xllcorner,yllcorner,cellsize,NODATA_value]


dtype = 'float64'
skip = 1
missing = ''
separator = ' '
read = 0

if useFishnet:
    bedrock = read_ndArray(textFileFolder + 'Bedrock_MODFLOW.txt', dtype,
        skip, missing, separator,nrows,ncols,read)
    bound = read_ndArray(textFileFolder + 'SlopingBaseBound_MODFLOW.txt',
        dtype, skip, missing, separator,nrows,ncols,read)
    if useMinDEM:
        DEMextraction = read_ndArray(textFileFolder + '
            Surface_Elev_Min_MODFLOW.txt', dtype, skip, missing, separator
            ,nrows,ncols,read)
        DEM = read_ndArray(textFileFolder + 'Surface_Elev_Avg_MODFLOW.txt'
            , dtype, skip, missing, separator,nrows,ncols,read)
    else:
        DEM = read_ndArray(textFileFolder + 'Surface_Elev_Avg_MODFLOW.txt'
            , dtype, skip, missing, separator,nrows,ncols,read)
        DEMextraction = DEM
    pwl = read_ndArray(textFileFolder + 'PredWlv_MODFLOW.txt', dtype, skip
        , missing, separator,nrows,ncols,read)
    K = read_ndArray(textFileFolder + 'K_MODFLOW.txt', dtype, skip,
        missing, separator,nrows,ncols,read)
```

```
102     R = read_ndArray(textFileFolder + 'Recharge_MODFLOW.txt', dtype, skip,
            missing, separator,nrows,ncols,read)
    else:
104     bedrock = read_ndArray(textFileFolder + 'Bedrock_MODFLOW.txt', dtype,
            skip, missing, separator,nrows,ncols,read)
        bound = read_ndArray(textFileFolder + 'SlopingBaseBound_MODFLOW.txt',
            dtype, skip, missing, separator,nrows,ncols,read)
106     DEM = read_ndArray(textFileFolder + 'DEM_MODFLOW.txt', dtype, skip,
            missing, separator,nrows,ncols,read)
        pwl = read_ndArray(textFileFolder + 'PredWlv_MODFLOW.txt', dtype, skip
            , missing, separator,nrows,ncols,read)
108     K = read_ndArray(textFileFolder + 'K_MODFLOW.txt', dtype, skip,
            missing, separator,nrows,ncols,read)
        R = read_ndArray(textFileFolder + 'Recharge_MODFLOW.txt', dtype, skip,
            missing, separator,nrows,ncols,read)
110     #SPYD = read_ndArray(textFileFolder + 'SPYD_MODFLOW.txt', dtype, skip,
            missing, separator,rows,cols,read)
        #initialHead = read_ndArray(textFileFolder + 'InitialHead_MODFLOW.txt
            ', dtype, skip, missing, separator,rows,cols,read)
112     #conductance = read_ndArray(textFileFolder + 'Conductance_MODFLOW.txt
            ', dtype, skip, missing, separator,rows,cols,read)

114 delta_E = R[10][10]/1000

116 intervals = [1]
    for runs in xrange(1):
118     for interval in intervals:
            if createRasters:
120             heads = numpy.ones((nrows,ncols),"float64")*-9999
                extraction = numpy.ones((nrows,ncols),"float64")*-9999
122         for row in xrange(20,35):     #use nrows to simulate all transects
                activeCellIndices = indexActiveCells(row,bound)
124             if len(activeCellIndices) > 0:
                    transectNum = 0
126                 for transectIndices in activeCellIndices:
                        lamb = .01
128                     nu = 10
                        if len(transectIndices) > minTransectLen:
130                         stop = False
                            iteration = 1
132                         min_count = 0
                            minChange = 0
134                         minChangeStop = 20
                            smallChangeCount = 0
136                         smallChangeCountStop = 10
                            smallChangeCountValue = 150
138                         change_new = 0
                            objNew = numpy.zeros((1,2),'float64')
140                         objList = [0,0]

142                         transectNum = transectNum + 1
```

```
                    minThickness = 10  #Depth to water
144                 #interval = 5
                    #Mode: DTW
146                 extractionPoints = FindLowPoints(row,
                        transectIndices, cellsize, DEMextraction, pwl,
                         bedrock, K, R, minThickness, interval, mode =
                         "DTW" )
                    #Mode: Interval
148                 #extractionPoints = FindLowPoints(row,
                        transectIndices, cellsize, DEM, pwl, bedrock,
                        K, R, minThickness, interval, mode = "Interval
                        " )
                    if len(extractionPoints[1]) > 0:
150                     initilizeE = False
                        E1 = extractionPoints[1]
152                     extractionPointIndices = extractionPoints[0]
                    else:
154                     extractionPointIndices = extractionPoints[0]
                    E_indices = extractionPointIndices[:]
156                 if len(E_indices) > 0:
                        if initilizeE:
158                         E1 = []
                        print 'Optimizing transect %d of row %d' %(
                            transectNum, row+1)
160                     #print 'transect indices',transectIndices
                        #print 'Extraction points', E_indices
162                     if initilizeE:     #Used in L-M
                            for i in xrange(len(E_indices)):
164                             #E1.append(0)
                                #E1.append(.00001) #cell size = 1000
166                             #E1.append(.0001)   #cell size = 5000
                                E1.append(initialE)
168                     if runPSO:
                            variablePop = False  #Set to true to use a
                                population size that varies based on
                                the number of parameters being
                                optimized
170                         popSize = 300  #number of particles to use
                            dimensions = len(E_indices)
172                         maxIter = 800  #maximum number of
                                iteration to use for PSO
                            xmin = -0.0005  #starting point min (m/d)
174                         xmax = 0.0005  #starting point (m/d)
                                heavily influences final solution; 0.
                                0001 best for 1000m cellsize
                            xmaxEnforce = 2.5   #maximum extraction
                                value, enforced only by penalty
                                function
176                         enforceXMAX = True   #Use if want to limit
                                maximum extraction according to
                                xmaxEnforce value, enforced only by
```

```
    penalty function
xminEnforce = -10.0   #minimum extraction
    value, enforced only by penalty
    function
enforceXMIN = False   #Use if want to
    limit minimum extraction according to
    xminEnforce value, enforced only by
    penalty function
Vmax = 0.0001  #maximum velocity
amax = 1.5 #weights the personal best
bmax = 2.5 #weights the global or local
    best
maxSSE = 1000  #If objective function
    falls below this value then the
    optimization routine ends.
neighborhood = 0  #not used at this point
enforcePenalty = False  #Set to true to
    enforce additional penalty function
    defined in GWOptPSO.py. User needs to
    define this penalty function in the
    GWOptPSO.py script, otherwise leave
    set to False.
LevMarParams=[row,transectIndices,cellsize
    ,E_indices,bedrock,bound,DEM,pwl,K,R]
if variablePop:
    popSize = len(E_indices)*8   #This
        creates a ratio of 8:1 particles
        to parameters. Can be modified by
        user.
    if popSize>400:  #This keeps the
        number of particles from becoming
        too large if the number of
        parameter is quite large. This
        number can be modified by user.
        popSize = 400
PSO_results = PSO(popSize, dimensions,
    maxIter, xmin, xmax, xmaxEnforce,
    xminEnforce,  Vmax, amax, bmax, maxSSE
    , LevMarParams, neighborhood,
    enforceXMAX, enforceXMIN,
    enforcePenalty)
Ebest = PSO_results[0]
OF = PSO_results[1]
OFmin = PSO_results[2]
OFmax = PSO_results[3]
Ebest = Ebest[:]
E1 = Ebest
solnNew = OneD_solve(row,transectIndices,
    cellsize,E1,E_indices,bedrock,bound,
    DEM,pwl,K,R,dtype = 'float64')
headMidNew = solnNew[1]
```

```
                objNew = solnNew[0]
200             print 'SSE:', sum(objNew**2)
                headBest = headMidNew.copy()
202             stop = True
            while not stop:
204             print 'iteration',iteration
                objList[0] = sum(objNew**2)
206             soln1 = OneD_solve(row,transectIndices,
                    cellsize,E1,E_indices,bedrock,bound,
                    DEM,pwl,K,R,dtype = 'float64')
                obj1 = soln1[0]
208             headMid1 = soln1[1]
                headsAdj = numpy.zeros((len(headMid1),len(
                    E1)),'float64')
210             for i in xrange(len(E1)):
                    E_adj = E1[i] + delta_E
212                 E2 = E1[:]
                    E2[i] = E_adj
214                 soln2 = OneD_solve(row,transectIndices
                        ,cellsize,E2,E_indices,bedrock,
                        bound,DEM,pwl,K,R,dtype = 'float64
                        ')
                    obj2 = soln2[0]
216                 headMid2 = soln2[1]
                    if i == 0:
218                     headsAdj = numpy.zeros((len(
                            headMid2[0]),len(E1)),'float64
                            ')
                    headsAdj[:,i:i+1] = headMid2.T.copy()
220
                jac = DD_Jacobian(headMid1, headsAdj,
                    delta_E, E_indices,dtype='float64')
222             #jac = jac*100
                identityDim = jac.shape[1]
224             try:
                    E_new = numpy.reshape(E1,(len(E1),1))
                        - numpy.linalg.solve((numpy.dot(
                        jac.T,jac) + lamb*numpy.eye(
                        identityDim)),(numpy.dot(jac.T,
                        obj1)))
226             except:
                    None
228             E_new = numpy.reshape(E_new,(1,len(E1))).
                    tolist()[0]
                E_old = E1[:]
230             E1 = E_new[:]
                solnNew = OneD_solve(row,transectIndices,
                    cellsize,E1,E_indices,bedrock,bound,
                    DEM,pwl,K,R,dtype = 'float64')
232             objNew = solnNew[0]
                headMidNew = solnNew[1]
```

```
234                 objList[1] = sum(objNew**2)
                    print 'E',E1
236                 print 'Sum objFunc', objList[1]

238                 if iteration == 1:
                        objBest = objList[1].copy()
240                     headBest = headMidNew.copy()
                        Ebest = E1[:]
242                 else:
                        if objList[1] < objBest:
244                         objBest = objList[1].copy()
                            Ebest = E1[:]
246                         headBest = headMidNew.copy()

248                 if objList[1] < 10000:
                        stop=True
250
                    smallChange = sum(abs(objList[0])-abs(
                        objList[1]))
252                 if abs(smallChange) <
                        smallChangeCountValue:
                        smallChangeCount = smallChangeCount +
                            1
254                     if smallChangeCount ==
                            smallChangeCountStop:
                            stop = True
256                         print 'smallChangeCount = %d' %(
                                smallChangeCountStop)
                    else:
258                     smallChangeCount = 0

260                 if sum(objNew**2)[0] > sum(obj1**2)[0]:
                        lamb = lamb * nu
262                     print'lamb',lamb
                    else:
264                     lamb = lamb / nu
                        print'lamb',lamb
266
                    iteration = iteration + 1
268                 if iteration > maxIter:
                        stop = True
270                     stop_criteria = 'iteration > maxIter'
                        print 'iteration > maxIter'
272
                    change = sum(abs(objList[0])-abs(objList[1
                        ]))
274                 if abs(change) < 500:
                        minChange = minChange + 1
276                     if minChange == minChangeStop:
                            stop = True
278                         print 'minChange == 20'
```

```
                else:
280                     minChange = 0

282                 change_old = change_new
                    change_new = change
284
                    if abs(change_new - change_old) < 0.1:
286                     min_count = min_count + 1
                        if min_count == 50:
288                         print 'min_count = 50'
                            stop = True
290
            if plotFigures or saveFigures:
292             xaxis = [x*modelCellWidth for x in xrange(
                    len(headMidNew[0]))]
                E_orig = [0 for i in xrange(len(E_indices)
                    )]
294             solnOrig = OneD_solve(row,transectIndices,
                    cellsize,E_orig,E_indices,bedrock,
                    bound,DEM,pwl,K,R,dtype = 'float64')
                objOrig = solnOrig[0]
296             headMidOrig = solnOrig[1]
                pwlP = solnOrig[2]
298             bedrockP = solnOrig[3]
                DEMP = solnOrig[4]
300             E_plotX = [numpy.NaN for _ in xrange(len(
                    headMidNew[0]))]
                E_plotY = [x for x in xrange(len(
                    headMidNew[0]))]
302             i=0
                SI=i
304             Ebest_index = 0
                for index in transectIndices:
306                 if index in E_indices:
                        if Ebest[Ebest_index]*cellsize==0:
308                         E_plotY[i]=DEMP[0,i]
                            SI = i
310                         i = i+1
                            Ebest_index = Ebest_index + 1
312                     else:
                            E_plotX[i]=i*modelCellWidth
314                         E_plotY[i]=DEMP[0,i]
                            SI = i
316                         i = i+1
                            Ebest_index = Ebest_index + 1
318                 else:
                        E_plotY[i]=DEMP[0,SI]
320                     i=i+1

322             fig, ax = plt.subplots(1)
                plt.title('PSO_' + 'Row' + str(row+1) + '
```

```
                    _T' + str(transectNum))
                ax.plot(xaxis,headMidOrig[0],'b',xaxis,
                    DEMP[0],'#663000',xaxis,pwlP[0],'rx',
                    xaxis,bedrockP[0],'k',xaxis,headBest[0
                    ],'g',E_plotX,E_plotY,'d',markersize=5
                    ,markerfacecolor='blue')
                ax.legend(('No Optimization', 'DEM', 'pwl'
                    ,'bedrock', 'Optimized Soln'))
                ax.set_xlabel('X coordinate (m)')
                ax.set_ylabel('Elevation (m)')
                ymin = min(min(bedrockP)) - 20
                if max(max(headMidOrig)) > max(max(DEMP)):
                    ymax = max(max(headMidOrig)) + 150
                else:
                    ymax = max(max(DEMP)) + 150
                ax.set_ylim(ymin,ymax)

                if saveFigures:
                    plt.savefig(figureDirectory + 'PSO_' +
                        'Row' + str(row+1) + '_T' + str(
                        transectNum) + '.png')
                if plotFigures:
                    plt.show()
                plt.close()

            if plotOF or saveOF:
                #plot O.F.
                plotAllOF = False #If false only plot min
                    and max values
                objPlotX = []
                objPlotY = []
                shape = OF.shape
                rowsOF = shape[0]
                colsOF = shape[1]
                for OFrow in xrange(rowsOF):
                    objPlotY.append(OF[OFrow][:].tolist())
                    objPlotX.append([x for x in xrange(
                        colsOF)])

                fig, ax = plt.subplots(1)
                plt.title('PSO_OF_' + 'Row' + str(row+1) +
                    '_T' + str(transectNum))
                if plotAllOF:
                    for OFrow in xrange(rowsOF):
                        ax.plot(objPlotX[OFrow],objPlotY[
                            OFrow],'b')
                    ax.set_xlabel('iteration')
                    ax.set_ylabel('Objective Function
                        Value')
                else:
                    objPlotX = [_ for _ in xrange(len(
```

```
                                        OFmin))]
362                                     ax.plot(objPlotX,OFmin,'b',objPlotX,
                                            OFmax,'r')
                                        ax.set_xlabel('iteration')
364                                     ax.set_ylabel('Objective Function
                                            Value')
                                        ax.legend(('Global Best', 'Global
                                            Worst'))
366
368                                     if saveOF:
                                            plt.savefig(figureDirectory + 'PSO_OF_
                                                ' + 'Row' + str(row+1) + '_T' +
                                                str(transectNum) + '.png')
370                                     if plotOF:
                                            plt.show()
372                                     plt.close()

374
                                    if createRasters:
376                                     i=0
                                        for index in transectIndices:
378                                         heads[row,index] = headBest[0,i]
                                            i=i+1
380                                     c=0
                                        for index in extractionPointIndices:
382                                         extraction[row,index] = Ebest[c]
                                            c=c+1
384
                                else:
386                                     print 'Transect %d in row %d has no extraction
                                            points' %(transectNum,row+1)
                                        E1 = [0]
388                                     E_indices = [0]
                                        solnNoOpt = OneD_solve(row,transectIndices,
                                            cellsize,E1,E_indices,bedrock,bound,DEM,
                                            pwl,K,R,dtype = 'float64')
390                                     objNoOpt = solnNoOpt[0]
                                        headMidNoOpt = solnNoOpt[1]
392                                     headBest = headMidNoOpt.copy()
                                        if createRasters:
394                                         i=0
                                            for index in transectIndices:
396                                             heads[row,index] = headBest[0,i]
                                                i=i+1
398 #                                       c=0
    #                                       for index in extractionPointIndices:
400 #                                           extraction[row,index] = Ebest[c]
    #                                           c=c+1
402                             else:
                                    transectNum = transectNum + 1
```

```
404                         print 'Transect %d of row %d length < %d' %(
                                transectNum,row+1,minTransectLen)
                    else:
406                     print 'row %d has no active cells' %(row+1)
                if useFishnet and createRasters and not FishnetVisualize:
408                 outputDirectory = 'C:/Documents and Settings/andya/My
                        Documents/Python/Py2.6/GW_Optimization/output/' +
                        outputGeoDatabaseName + '/'
                    txtOutput = 'C:/Documents and Settings/andya/My Documents/
                        Python/Py2.6/GW_Optimization/output/' + modelName + "/"
410                 Array2Raster(heads, header, baseName + "head" ,
                        outputDirectory, txtOutput)
                    Array2Raster(extraction, header, baseName + "Exm" ,
                        outputDirectory, txtOutput)
412                 extractionm2 = extraction*cellsize
                    extractionm2 = (extractionm2 <= -9999.0).choose(extractionm2,-
                        9999.0)
414                 Array2Raster(extractionm2, header, baseName + "Exm2" ,
                        outputDirectory, txtOutput)
                    headsErr = (heads <= -9999.0).choose(heads,1000000.0)
416                 error = pwl - headsErr
                    error = (error <= -100000.0).choose(error,-9999.0)
418                 Array2Raster(error, header, baseName + "Err" , outputDirectory
                        , txtOutput)
                    print "Rasters saved in", outputDirectory
420
                elif useFishnet and FishnetVisualize:
422                 extractionm2 = extraction*cellsize
                    extractionm2 = (extractionm2 <= -9999.0).choose(extractionm2,-
                        9999.0)
424                 headsErr = (heads <= -9999.0).choose(heads,1000000.0)
                    error = pwl - headsErr
426                 error = (error <= -100000.0).choose(error,-9999.0)
                    visualizeFishnet(heads,extractionm2,error,fishnetDim,
                        modelDirec)
428                 print "Fishnet Saved in ", modelDirec

430             elif createRasters:
                    outputDirectory = cwd + "\\output\\" + outputGeoDatabaseName +
                         "\\"
432                 txtOutput = cwd + "\\output\\" + modelName + "\\"
                    Array2Raster(heads, header, baseName + "head" ,
                        outputDirectory, txtOutput)
434                 Array2Raster(extraction, header, baseName + "Exm" ,
                        outputDirectory, txtOutput)
                    extractionm2 = extraction*cellsize
436                 extractionm2 = (extractionm2 <= -9999.0).choose(extractionm2,-
                        9999.0)
                    Array2Raster(extractionm2, header, baseName + "Exm2" ,
                        outputDirectory, txtOutput)
438                 headsErr = (heads <= -9999.0).choose(heads,1000000.0)
```

```
                error = pwl - headsErr
440             error = (error <= -100000.0).choose(error,-9999.0)
                Array2Raster(error, header, baseName + "Err" , outputDirectory
                    , txtOutput)
442             print "Rasters saved in", outputDirectory

444 print "Finished!"
    finish = time.clock()
446 runtime = (finish - start)/60/60
    print 'Runtime: ',runtime,' hours'
```

B.0.9 LevMar.py

The LevMar.py script contains several function used by the sloping base model and the Levenberg-Marquardt optimization routine. The *indexActiveCells* function determines the indices of cells in a transect to be included in the sloping base model. The *FindLow-Points* function is used to identify the points of extraction (parameters) to be optimized by Levenberg-Marquardt or PSO. There are three methods of identifying the extraction points: *DTW* (depth to water), *random*, and *Interval*. *DTW* looks for cells that have a depth to water less than or equal to a value given by the user and selects these cells as the extraction points. *Interval* selects cells at a user defined interval as extraction points. For example, if the user entered a value of 2 for the interval, then the second cell in a transect and every second cell after that would be used as extraction points. The *random* setting selects points at random. The *OneD_Solve* is the sloping base routine, it solves simulates the groundwater heads and flows for a given one dimensional transect. *DD_Jacobian* calculates the Jacobian matrix for use with the Levenberg-Marquardt routine using a simple divided difference method to calculate the derivatives.

```
1 #----------------------
  LevMar.py
3 #----------------------

5 import numpy, random
  from math import *
7
  def indexActiveCells(row,boundaryArray):
9     initValue = boundaryArray[row,0]
      if initValue == 1:
```

```
11          indexList = [[]]
            one2zero = True
13      else:
            indexList = []
15          one2zero = False
        change = False
17      segmentCount = 0
        for index,value in enumerate(boundaryArray[row]):
19          if one2zero:
                if value == 0 and not change:
21                  change = True
                if change and value == 1:
23                  indexList.append([])
                    segmentCount = segmentCount + 1
25                  change = False
                if value == 1:
27                  indexList[segmentCount].append(index)
            else:
29              if not change and value == 1:
                    change = True
31                  indexList.append([])
                if change and value == 0:
33                  change = False
                    segmentCount = segmentCount + 1
35              if value == 1:
                    indexList[segmentCount].append(index)
37      return indexList

39
    def ObservationInterval(numCells,usagePercent):
41      minLen = 5
        if numCells <= minLen:
43          minObservations = numCells
        else:
45          minObservations = minLen
        numObservations = ceil(usagePercent/100 * numCells)
47      if numObservations < minObservations:
            numObservations = minObservations
49      interval = numCells/numObservations
        observationIndices = []
51      for i in xrange(numObservations):
            if i == 0:
53              observationIndices.append(int(floor(interval/2)))
                newInterval = interval*1.5
55          else:
                observationIndices.append(int(floor(newInterval)))
57              newInterval += interval
        return observationIndices
59

61  def FindLowPoints(row, transectIndices, cellWidth, DEMArray, pwlArray,
```

```
    bedrockArray, KArray, RArray, minThickness = 5, interval = 1, mode="
    DTW"):
     E = []
63   if mode == "DTW":
         lowPointIndices = [index for index in transectIndices if DEMArray[
            row,index] - pwlArray[row,index] < minThickness]
65
     if mode == "random":
67       numExtPts = random.choice(xrange(1,8))
         indexes = transectIndices[:]
69       lowPointIndices=[]
         for i in xrange(numExtPts):
71           index = random.choice(xrange(len(indexes)))
             lowPointIndices.append(indexes[index])
73           indexes.pop(index)
         lowPointIndices.sort()
75
     if mode == "matchQ":
77       firstIndex = transectIndices[0]
         lastIndex = transectIndices[-1]
79       DEM = DEMArray[row,firstIndex:lastIndex+1]
         DEM = numpy.reshape(DEM,(1,len(transectIndices)))
81       bedrock = bedrockArray[row,firstIndex:lastIndex+1]
         bedrock = numpy.reshape(bedrock,(1,len(transectIndices)))
83       pwl = pwlArray[row,firstIndex:lastIndex+1]
         pwl = numpy.reshape(pwl,(1,len(transectIndices)))
85       K = KArray[row,firstIndex:lastIndex+1]
         K= numpy.reshape(K,(1,len(transectIndices)))
87       R = RArray[row,firstIndex:lastIndex+1]
         R = numpy.reshape(R,(1,len(transectIndices)))
89
         H = [pwl[0,i]-bedrock[0,i] for i in xrange(len(transectIndices))]
91       Q_noE = [R[0,0]*cellWidth]
         slope = [abs(DEM[0,0]-DEM[0,1])/len(transectIndices)/cellWidth]
93       for i in xrange(1,len(transectIndices)):
             Q_noE.append(Q_noE[i-1]+R[0,i]*cellWidth)
95           slope.append(abs(DEM[0,i]-DEM[0,i-1])/len(transectIndices)/
                cellWidth)
         slope = [abs(DEM[0,0]-DEM[0,-1])/len(transectIndices)/cellWidth
            for i in xrange(len(transectIndices))]
97       Q_reality = [H[i]*K[0,i]*slope[i] for i in xrange(len(
            transectIndices))]
         for i in xrange(len(transectIndices)):
99           if i <> 0:
                 E.append(Q_noE[i] - sum(E) - Q_reality[i])
101          else:
                 E.append(Q_noE[i] - Q_reality[i])
103      lowPointIndices = transectIndices[:]

105  if mode == "Interval":
         lowPointIndices = []
```

```
107         for i in xrange(min(transectIndices),max(transectIndices),interval
                ):
                lowPointIndices.append(i)
109
        return lowPointIndices,E
111

113 def OneD_solve(row,transectIndices,cellWidth,E,E_indices,bedrockArray,
        boundaryArray,DEMArray,pwlArray,KArray,RArray,dtype = 'float64'):
        firstIndex = transectIndices[0]
115     lastIndex = transectIndices[-1]
        bedrock = bedrockArray[row,firstIndex:lastIndex+1]
117     bedrock = numpy.reshape(bedrock,(1,len(transectIndices)))
        bound = boundaryArray[row,firstIndex:lastIndex+1]
119     bound = numpy.reshape(bound,(1,len(transectIndices)))
        DEM = DEMArray[row,firstIndex:lastIndex+1]
121     DEM = numpy.reshape(DEM,(1,len(transectIndices)))
        pwl = pwlArray[row,firstIndex:lastIndex+1]
123     pwl = numpy.reshape(pwl,(1,len(transectIndices)))
        K = KArray[row,firstIndex:lastIndex+1]
125     K= numpy.reshape(K,(1,len(transectIndices)))
        R = RArray[row,firstIndex:lastIndex+1]
127     R = numpy.reshape(R,(1,len(transectIndices)))

129     totalCells = len(transectIndices)

131     WArray = RArray.copy()
        for i in xrange(len(E_indices)):
133         WArray[row,E_indices[i]] = RArray[row,E_indices[i]] - E[i]
        W = WArray[row,firstIndex:lastIndex+1]
135     W = numpy.reshape(W,(1,len(transectIndices)))

137     heads = numpy.zeros((1,totalCells+1),dtype)
        discharge = numpy.zeros((1,totalCells+1),dtype)
139     potentials = numpy.zeros((1,totalCells*2),dtype)

141     N = totalCells - 1   #index of last value in segment
        head0 = pwl[0,N]
143     Phi0 = 0.5*K[0,N]*(head0 - bedrock[0,N])**2
        Q0 = sum(W[0])*cellWidth
145     heads[0,N+1] = head0
        discharge[0,N+1] = Q0
147     potentials[0,N*2+1] = Phi0

149     for i in xrange(totalCells):
            Phi = -(W[0,N]/2)*(-cellWidth)**2 - Q0*(-cellWidth) + Phi0
151         potentials[0,N*2] = Phi
            if Phi > 0:
153             head = bedrock[0,N] + sqrt((2*Phi)/K[0,N])
            else:
155             head = bedrock[0,N] + 1
```

```
157         N = N - 1

159         if N >= 0:
                Phi0 = 0.5*K[0,N]*(head - bedrock[0,N])**2
161             potentials[0,N*2+1] = Phi0
                Q0 = sum(W[0,0:N+1])*cellWidth
163         else:
                Q0 = 0
165         heads[0,N+1] = head
            discharge[0,N+1] = Q0
167
        phiMid = numpy.zeros((1,totalCells),dtype)
169     headMid = numpy.zeros((1,totalCells),dtype)

171     c = 0
        for i in xrange(totalCells):
173         phiMid[0,i] = (potentials[0,c] + potentials[0,c+1])/2 + W[0,i]*(-
                cellWidth)**2/8
            if phiMid[0,i] > 0:
175             headMid[0,i] = bedrock[0,i] + sqrt(2*phiMid[0,i]/K[0,i])
            else:
177             headMid[0,i] = bedrock[0,i] + 1
                #headMid[0,i] = bedrock[0,i] - sqrt(2*-phiMid[0,i]/K[0,i])
179         c = c + 2

181     objfunc = headMid - pwl
        objfunc = objfunc.T
183     return objfunc,headMid, pwl, bedrock, DEM

185
    def DD_Jacobian(headMid1, headsAdj, deltaE, E_indices,dtype='float64'):
187     jac = numpy.zeros((len(headMid1[0]),len(E_indices)),dtype)
        headsNoAdj = numpy.zeros_like(jac)
189
        for i in xrange(len(E_indices)):
191         headsNoAdj[:,i:i+1] = headMid1.T.copy()

193     jac = headsAdj - headsNoAdj
        jac = jac/deltaE
195
        return jac
```

B.0.10 GWOptPSO.py

The GWOptPSO.py script executes the PSO optimization. The inputs to this function are described by the comments in the GW_Model.py script. Within the GWOptPSO.py script are three penalty functions which can be modified here by the user if desired. The three

penalty functions are contained within three if statements. The penalty function beginning on line 48 defines the penalty for an extraction value greater than the value set by the user in GW_Model.py. The penalty function beginning on line 57 defines the penalty for an extraction value less than the minimum value set by the user in GW_Model.py. The penalty function beginning on line 64 is a space where the user can write their own penalty function if desired. More than one penalty function can be defined here.

```
#----------------------
GWOptPSO.py
#----------------------

import numpy
from LevMar import *
import time

def PSO(popSize, dimensions, maxIter, xmin, xmax, xmaxEnforce, xminEnforce
    , Vmax, amax, bmax, maxSSE, LevMarParams, neighborhood=0, enforceXMAX
    = True, enforceXMIN = True, enforcePenalty = False):
    timeS = time.clock()
    totalRows = LevMarParams[0]
    transectIndices = LevMarParams[1]
    cellWidth = LevMarParams[2]
    E_indices = LevMarParams[3]
    bedrockArray = LevMarParams[4]
    boundaryArray = LevMarParams[5]
    DEMArray = LevMarParams[6]
    pwlArray = LevMarParams[7]
    KArray = LevMarParams[8]
    RArray = LevMarParams[9]

    pBest = numpy.zeros((dimensions,popSize),"float64")
    pLocal = numpy.zeros((dimensions,popSize),"float64")
    pCurrentFitness = numpy.zeros((dimensions,popSize),"float64")
    pGlobal = numpy.zeros((dimensions,1),"float64")
    distances = numpy.zeros((popSize-1,popSize),"float64")
    offsetMatrix = numpy.ones((popSize,popSize-1),"float64")
    for i in xrange(1,popSize-1):
        offsetMatrix[i,0:i] = 0
    fitnessBest = []
    fitnessBestGlobal = []
    OFmin = []
    OFmax = []
    x = numpy.random.rand(dimensions,popSize)*(xmax-xmin) + xmin
    v = numpy.random.rand(dimensions,popSize)*(Vmax)
    OF = numpy.zeros((popSize,maxIter+1),"float64")

    stop = False
```

```
    iteration = 0

    while not stop:
        for j in xrange(popSize):
          if iteration==0:
              E = x[:,j].tolist()
              objFunc = OneD_solve(totalRows,transectIndices,cellWidth,E,
                  E_indices,bedrockArray,boundaryArray,DEMArray,pwlArray,
                  KArray,RArray,dtype = 'float64')
              fitness = sum(objFunc[0]**2)
              OF[j][iteration] = fitness[0]
              if enforceXMAX:
                  #if max(E)*cellWidth > xmaxEnforce:
                      #fitness = fitness + 40000
                  if max(E)*cellWidth > xmaxEnforce:
                      penalty = 0
                      for value in E:
                          if value*cellWidth > xmaxEnforce:
                              penalty = penalty + 30000
                      fitness = fitness + penalty
              if enforceXMIN:
                  if min(E)*cellWidth < xminEnforce:
                      penalty = 0
                      for value in E:
                          if value*cellWidth < xminEnforce:
                              penalty = penalty + 30000
                      fitness = fitness + penalty
              if enforcePenalty:
                  count = 0
                  penalty = 0
                  for extraction in E*cellWidth:
                      if extraction < 0:
                          count = count + 1
                  #for extraction in E*cellWidth:
                  #    if extraction <> 0:
                  #        count = count + 1
                  penalty = 7500*count
                  fitness = fitness + penalty
              fitnessBest.append(fitness)
              pBest[:,j] = x[:,j]
              if j>=popSize-1:
                  fitnessBestGlobal = min(fitnessBest)
                  fitnessBestGlobalIndex = fitnessBest.index(
                      fitnessBestGlobal)
                  pGlobal[:,0] = x[:,fitnessBestGlobalIndex]
          else:
              E = x[:,j].tolist()
              objFunc = OneD_solve(totalRows,transectIndices,cellWidth,E,
                  E_indices,bedrockArray,boundaryArray,DEMArray,pwlArray,
                  KArray,RArray,dtype = 'float64')
              fitness = sum(objFunc[0]**2)
```

```
                OF[j][iteration] = fitness[0]
86              if enforceXMAX:
                    #if max(E)*cellWidth > xmaxEnforce:
88                      #fitness = fitness + 40000
                    if max(E)*cellWidth > xmaxEnforce:
90                      penalty = 0
                        for value in E:
92                          if value*cellWidth > xmaxEnforce:
                                penalty = penalty + 30000
94                      fitness = fitness + penalty
                if enforceXMIN:
96                  if min(E)*cellWidth < xminEnforce:
                        penalty = 0
98                      for value in E:
                            if value*cellWidth < xminEnforce:
100                             penalty = penalty + 30000
                        fitness = fitness + penalty
102             if enforcePenalty:
                    count = 0
104                 penalty = 0
                    for extraction in E*cellWidth:
106                     if extraction < 0:
                            count = count + 1
108                 #for extraction in E*cellWidth:
                    #    if extraction <> 0:
110                 #        count = count + 1
                    penalty = 7500*count
112                 fitness = fitness + penalty

114             if fitnessBest[j] > fitness:
                    fitnessBest[j] = fitness
116                 pBest[:,j] = x[:,j]
                if j >= popSize-1 and min(fitnessBest) < fitnessBestGlobal:
118                 fitnessBestGlobal = min(fitnessBest)
                    fitnessBestGlobalIndex = fitnessBest.index(
                       fitnessBestGlobal)
120                 pGlobal[:,0] = x[:,fitnessBestGlobalIndex]

122         OFmin.append(fitnessBestGlobal[0])
            OFmax.append(max(fitnessBest)[0])
124
            if iteration == maxIter:
126             print 'Maximum Iteration Reached'
                print 'best fit:', fitnessBestGlobal
128             stop = True
            if fitnessBestGlobal < maxSSE:
130             print '\nFinished'
                print 'iteration:', iteration
132             print 'best fit:', fitnessBestGlobal
                stop = True
134         else:
```

```
            print 'iteration:', iteration
            print 'best fit:', fitnessBestGlobal
            a = numpy.random.rand(dimensions,popSize)*(amax)
            b = numpy.random.rand(dimensions,popSize)*(bmax)
            v = v + a*(pBest - x) + b*(pGlobal - x)
            v = (v > Vmax).choose(v,Vmax)
            x = x + v
            #x = (x < xmin).choose(x,xmin)
            #if enforceXMAX:
                #x = (x*cellWidth > xmaxEnforce).choose(x,xmaxEnforce/(
                    cellWidth))
            iteration = iteration + 1

    Ebest = pGlobal.tolist()
    for i in xrange(len(Ebest)):
        Ebest[i]=Ebest[i][0]
    print 'Extraction m^2/day:',[E * cellWidth for E in Ebest]
    timeF = time.clock()
    print 'run time=',timeF-timeS
    return Ebest,OF,OFmin,OFmax
```

B.0.11 Additional Required Scripts

The following scripts contain useful functions that are called multiple times by the other scripts listed in the appendix. These scripts do not require user input.

FileReader_ndArray.py

The FileReader_ndArray.py script reads in a text file and outputs a numpy n-dimensional array. In this case it was used to read in the text files produced by CSV_to_2dArray.py if using the fishnet grid. If rasters are being used for input data, this script is used to read in the text files produced by MODFLOW_txt.py.

```
#----------------------
FileReader_ndArray.py
#----------------------

def read_ndArray(filename, dtype, skip=1, missing='', separator=',',rows=1
    ,cols=1,read=0):
    """
        Generalized input routine from http://www.scipy.org/Cookbook/
            InputOutput
        Modified by SM Welch to handle missing data (10 Aug 10)
        Modified by W Kusnierczyk to handle header lines (20 Aug 10)
```

```
        Read a file with an arbitrary number of columns.
        The type of data in each column is arbitrary
        It will be cast to the given dtype at runtime

        Arguments:
            filename - name of file to be read (including path if needed)
            dtype - numpy data type specifier
            skip - number of header lines to skip (optional; default 1)
            missing - value to use for missing data (optional; default '')
            separator - character separating data items in a row (optional
               ; default ',')
            read - number of lines to read (0 means read all)

        Outputs
            a numpy N-dimensional array with single given datatype (numpy.
               float32, numpy.float64, etc...)

        Requirements
            numpy must be available
    """
    import numpy

    cast = numpy.cast
    data = numpy.zeros((rows,cols),dtype)
    In_file = open(filename, 'r')
    for dummy in xrange(skip): In_file.readline()
    if read == 0:
        rowCount=0
        for line in In_file:
            fields = line.strip().split(separator)
            for colCount, number in enumerate(fields):
                if number == '': number = missing
                data[rowCount][colCount] = number
            rowCount = rowCount + 1
        return data
    else:
        c1=0
        for line in In_file:
            if c1 < read:
                fields = line.strip().split(separator)
                for i, number in enumerate(fields):
                    if number == '': number = missing
                    data[i].append(number)
                c1 += 1
        for i in xrange(len(dtype)):
            data[i] = cast[dtype[i]](data[i])
        return numpy.rec.array(data, dtype=dtype)
    In_file.close()
```

Array2Raster.py

The Array2Raster.py script is used to create rasters from 2D arrays in Python.

```
#----------------------
Array2Raster.py
#----------------------

import numpy, os, arcpy

def Array2Raster(array, header, rasterName, outputDirectory, txtOutput):
    print "Creating " + str(rasterName) + " Raster"

    ncols = header[0]
    nrows = header[1]
    xllcorner = header[2]
    yllcorner = header[3]
    cell_width = header[4]
    NODATA_value = header[5]

    filepath = txtOutput + "//txt//" + str(rasterName) + str(int(
        cell_width)) + ".asc"
    if os.path.exists(filepath):
        append = 0
        while os.path.exists(filepath):
            append += 1
            filepath = txtOutput + "//txt//" + str(rasterName) + str(int(
                cell_width)) + "_" + str(append) + ".asc"
        numpy.savetxt(filepath, array, fmt="%5.10f")

        outfile=open(filepath,'r+')
        old = outfile.read()
        outfile.seek(0)
        outfile.write('ncols         ' + str(ncols) + '\n')
        outfile.write('nrows         ' + str(nrows) + '\n')
        outfile.write('xllcorner     ' + str(xllcorner) + '\n')
        outfile.write('yllcorner     ' + str(yllcorner) + '\n')
        outfile.write('cellsize      ' + str(cell_width) + '\n')
        outfile.write('NODATA_value  ' + str(NODATA_value) + '.0000' + '\n
            ' + old)
        outfile.close()

    else:
        numpy.savetxt(filepath, array, fmt="%5.10f")
        outfile=open(filepath,'r+')
        old = outfile.read()
        outfile.seek(0)
        outfile.write('ncols         ' + str(ncols) + '\n')
        outfile.write('nrows         ' + str(nrows) + '\n')
        outfile.write('xllcorner     ' + str(xllcorner) + '\n')
        outfile.write('yllcorner     ' + str(yllcorner) + '\n')
```

```
            outfile.write('cellsize      ' + str(cell_width) + '\n')
46          outfile.write('NODATA_value  ' + str(NODATA_value) + '.0000' + '\n
                ' + old)
            outfile.close()
48          append = ''

50      inAscii = filepath
        outRaster = outputDirectory + rasterName + str(append)
52      if os.path.exists(outRaster):
            while os.path.exists(outRaster):
54              append += 1
                outRaster = outputDirectory + rasterName + str(append)
56      rasterType = "FLOAT"
        arcpy.ASCIIToRaster_conversion(inAscii, outRaster, rasterType)
```

FishnetVisualize.py

The FishnetVisualize.py script is used to create fishnet grids for head, error, and extraction values calculated by the GW_Model.py script. The grids are saved as shapefiles in a user defined output geodatabase.

```
1  #----------------------
   FishnetVisualize.py
3  #----------------------

5  import sys, time, numpy, os, fileinput, arcgisscripting
   from FileReader_recordArray import *
7  gp = arcgisscripting.create(9.3)

9  def visualizeFishnet(heads,E,error,fishnetDim,modelDirec):
       #High Plains Fishnet Directory
11     model_directory = modelDirec + fishnetDim
       #Fishnet csv file path
13     fishnet_filepath = model_directory + "/" + fishnetDim + ".txt"

15

       #Read in Fishnet csv
17     print 'Reading in Fishnet csv file'
       Fishnet_Data_Type = numpy.dtype([("boundary","int32"),("xid","int32")
           ,("yid","int32"),
19                                      ("X","int32"),("Y","int32"),("K","
                                           float32"),
                                        ("bedrock","float32"),("pwl","float32
                                           "),
21                                      ("rech","float32"),("avg_lse","
                                           float32"),
                                        ("min_lse","float32"),])
```

```
    fishnet = read_recordArray(fishnet_filepath, Fishnet_Data_Type, skip=1
        , missing='', separator=None)

    length=fishnet.shape
    length=length[0]
    xid=fishnet['xid']
    yid=fishnet['yid']
    ncols=max(xid)
    nrows=max(yid)
    array_points=(ncols)*2

    #Add heads to fishnet
    print "Rearranging Heads"
    Head_mid = numpy.zeros((1,nrows*ncols),'float32')
    row=nrows-1
    col=0
    for i in xrange(nrows*ncols):
        Head_mid[0][i]=heads[row][col]
        if col==ncols-1:
            col=0
            row=row-1
        else:
            col=col+1

    print 'Writing head values to Head feature class'
    headFC = model_directory + "/" + fishnetDim + ".gdb/AggregateCells/
        Head"
    FCrows = gp.UpdateCursor(headFC)
    FCrow = FCrows.Next()
    iter = 0
    while FCrow:
        if Head_mid[0][iter] == -9999:
            FCrow = FCrows.Next()
            iter += 1
        else:
            value = float(Head_mid[0][iter])
            FCrow.SetValue('Head',value)
            FCrows.UpdateRow(FCrow)
            FCrow = FCrows.Next()
            iter += 1
    del FCrow, FCrows

    #Add errors to fishnet
    print "Rearranging Errors"
    model_err = numpy.zeros((1,nrows*ncols),'float32')
    row=nrows-1
    col=0
    for i in xrange(nrows*ncols):
        model_err[0][i]=error[row][col]
        if col==ncols-1:
            col=0
```

```
            row=row-1
        else:
            col=col+1

print 'Writing Error values to Error feature class'
errFC = model_directory + "/" + fishnetDim + ".gdb/AggregateCells/
    Error"
FCrows = gp.UpdateCursor(errFC)
FCrow = FCrows.Next()
iter = 0
while FCrow:
    if model_err[0][iter] == -9999:
        FCrow = FCrows.Next()
        iter += 1
    else:
        value = float(model_err[0][iter])
        FCrow.SetValue('Error',value)
        FCrows.UpdateRow(FCrow)
        FCrow = FCrows.Next()
        iter += 1
del FCrow, FCrows

#Add extraction values to fishnet
print "Rearranging extractions"
extraction = numpy.zeros((1,nrows*ncols),'float32')
row=nrows-1
col=0
for i in xrange(nrows*ncols):
    extraction[0][i]=E[row][col]
    if col==ncols-1:
        col=0
        row=row-1
    else:
        col=col+1

print 'Writing extraction values to Extraction feature class'
extractionFC = model_directory + "/" + fishnetDim + ".gdb/
    AggregateCells/Extraction"
FCrows = gp.UpdateCursor(extractionFC)
FCrow = FCrows.Next()
iter = 0
while FCrow:
    if extraction[0][iter] == -9999:
        FCrow = FCrows.Next()
        iter += 1
    else:
        value = float(extraction[0][iter])
        FCrow.SetValue('Extraction',value)
        FCrows.UpdateRow(FCrow)
        FCrow = FCrows.Next()
        iter += 1
```

```
121     del FCrow, FCrows
```

Appendix C

Input Data

This appendix describes the six input data sets used for the sloping base model.

C.1 Bedrock Elevation

The bedrock contours (figure C.1) used to produce the bedrock surface raster were obtained from the Water Resources NSDI Node website (http://water.usgs.gov/lookup/getgislist). The shapefile was produced by the U.S. Geological Survey in 1999 and it contains contours of bedrock elevations given in feet. This is the *bedrock_elev.shp* shapefile used to define the *Bedrock* variable in the MODFLOWFilesGen.py script. A bedrock surface raster is generated by the MODFLOWtxt.py script by using the *Topo to Raster* tool in ArcGIS.

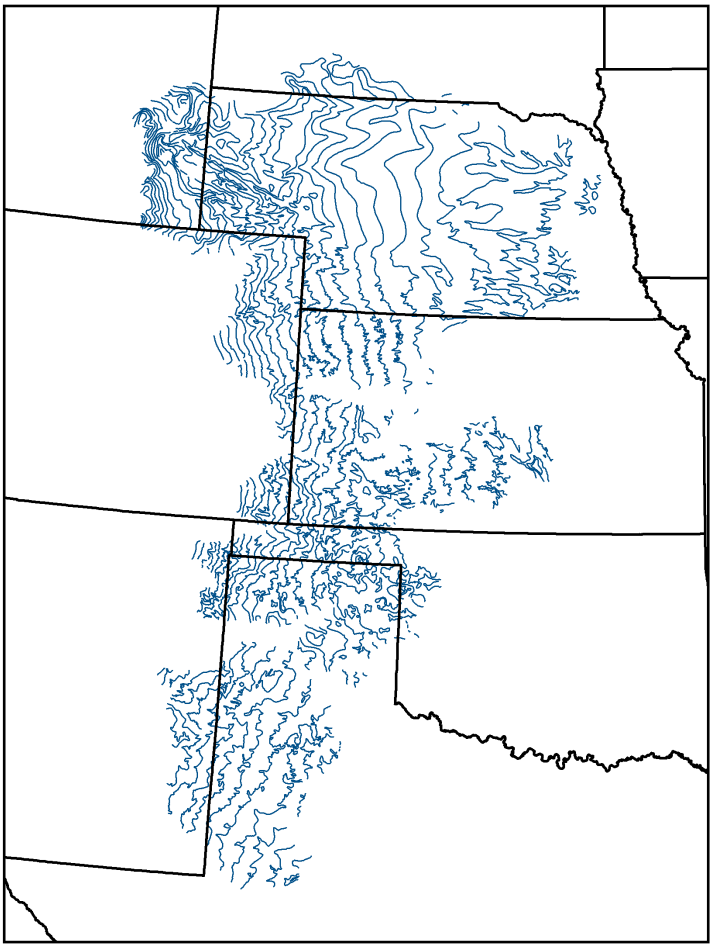

Figure C.1: *High Plains Bedrock Contours Input Shapefile*

C.2 Land Surface Elevation

The land surface elevation raster is a digital elevation model (DEM). Smaller DEM sections of the overall raster were downloaded from the USGS Seamless Data Warehouse (seamless.usgs.gov) at the one arc second resolution which is approximately a 30 meter cell size. These small DEMs were combined into a single raster shown in figure C.2 using the *Mosaic* tool in ArcGIS. This 30 meter DEM was then aggregated up to a 1000 meter cell size. This is the *dem1000* raster used to define the *DEM* variable in the MODFLOWFilesGen.py script.

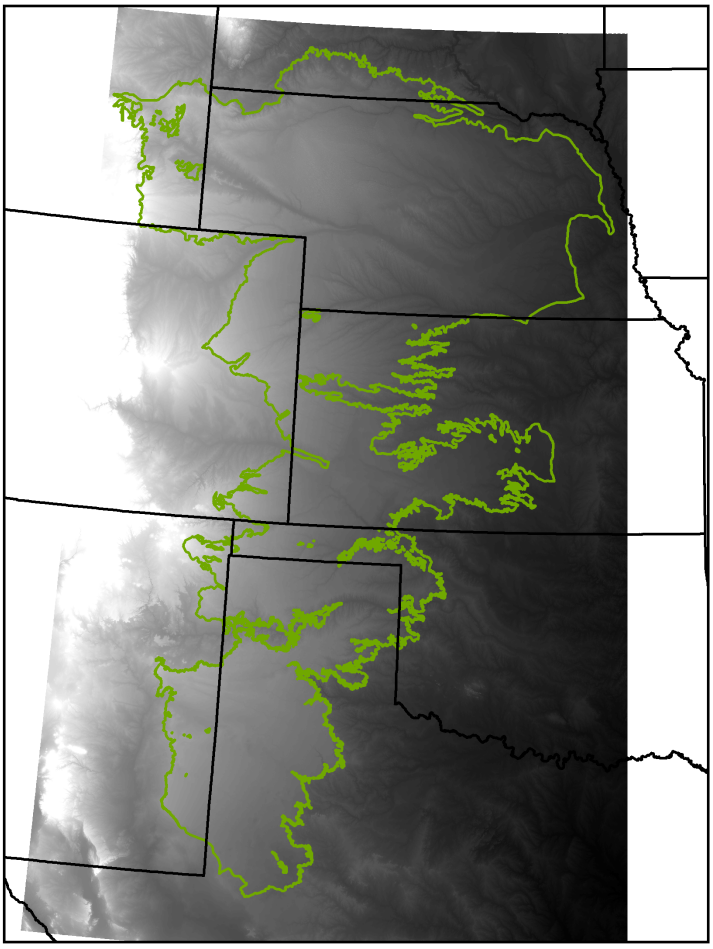

Figure C.2: *High Plains DEM Input Raster*

C.3 Predevelopment Groundwater Table Elevation

The predevelopment water level contours (figure C.3) used to produce the predevelopment groundwater level surface raster were obtained from the Water Resources NSDI Node website (http://water.usgs.gov/lookup/getgislist). The shapefile was produced by the U.S. Geological Survey in 1999 and it contains contours of predevelopment groundwater elevations given in feet. This is the *predevwlv_elev.shp* shapefile used to define the *PredevWaterLv* variable in the MODFLOWFilesGen.py script. A predevelopment groundwater level surface raster is generated by the MODFLOWtxt.py script by using the *Topo to Raster* tool in ArcGIS.

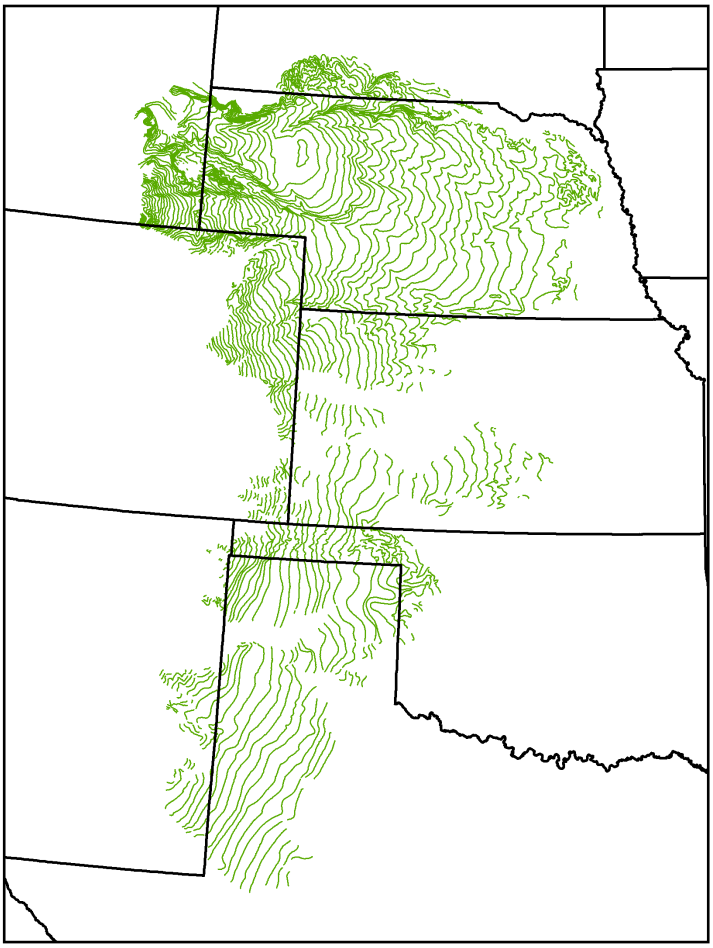

Figure C.3: *High Plains Predevelopment Contours Input Shapefile*

C.4 Hydraulic Conductivity

The hydraulic conductivity shapefile (figure C.4) was obtained from the Water Resources NSDI Node website (http://water.usgs.gov/lookup/getgislist). The shapefile was produced by the U.S. Geological Survey in 1999 and it contains polygons representing areas of different hydraulic conductivity. This is the *hyd_k.shp* shapefile used to define the *Hyd_K* variable in the MODFLOWFilesGen.py script. In the attribute table, each polygon has a minimum and maximum hydraulic conductivity value. The average value for each polygon was then calculated in ArcGIS and added to the attribute table. The average hydraulic conductivity value is the value used in the sloping base model.

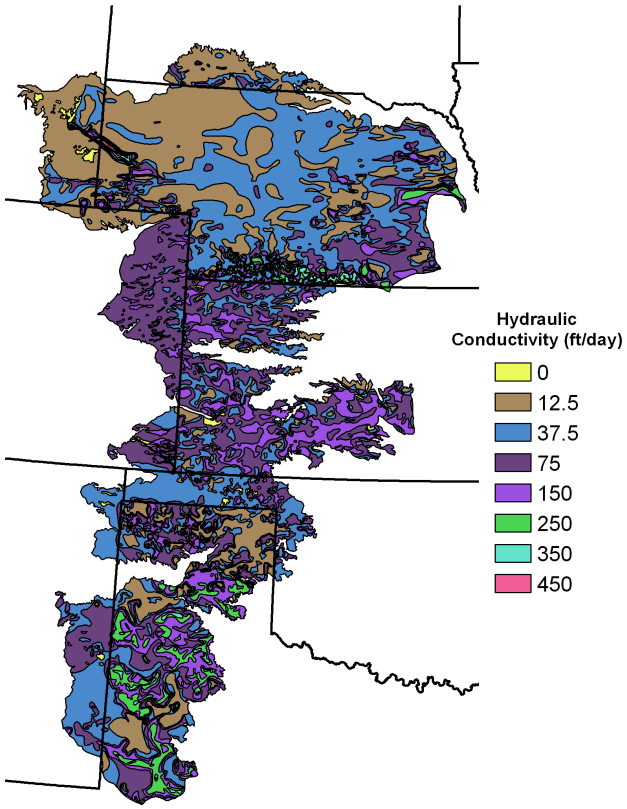

Figure C.4: *High Plains Hydraulic Conductivity Input Shapefile*

C.5 Recharge Rate

Recharge rates for the High Plains aquifer were obtained from an image on page 49 of a report produced by Dugan et. al. 2000. The image of recharge contours was copied and georeferenced in ArcGIS. The contour lines were then traced in ArcGIS and the shapefile shown in figure C.5 was produced which represents the recharge in inches per year. This shapefile was then interpolated in ArcGIS to produce the recharge raster shown in figure C.6. This recharge raster is the *rech_in_year* raster used to define the *Recharge* variable in MODFLOWFilesGen.py script.

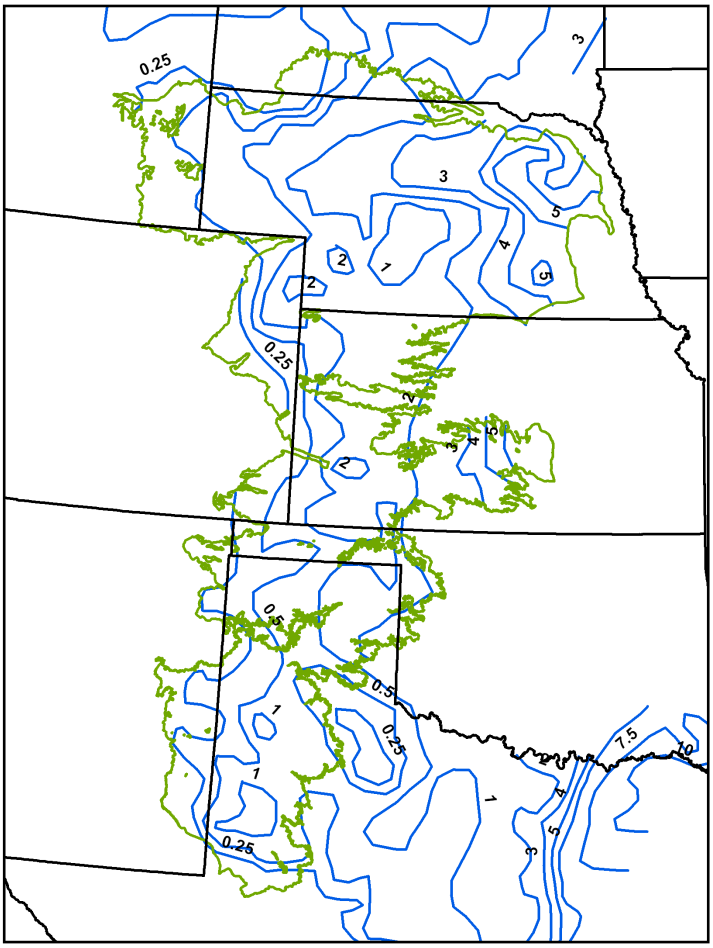

Figure C.5: *High Plains Recharge Contours Shapefile*

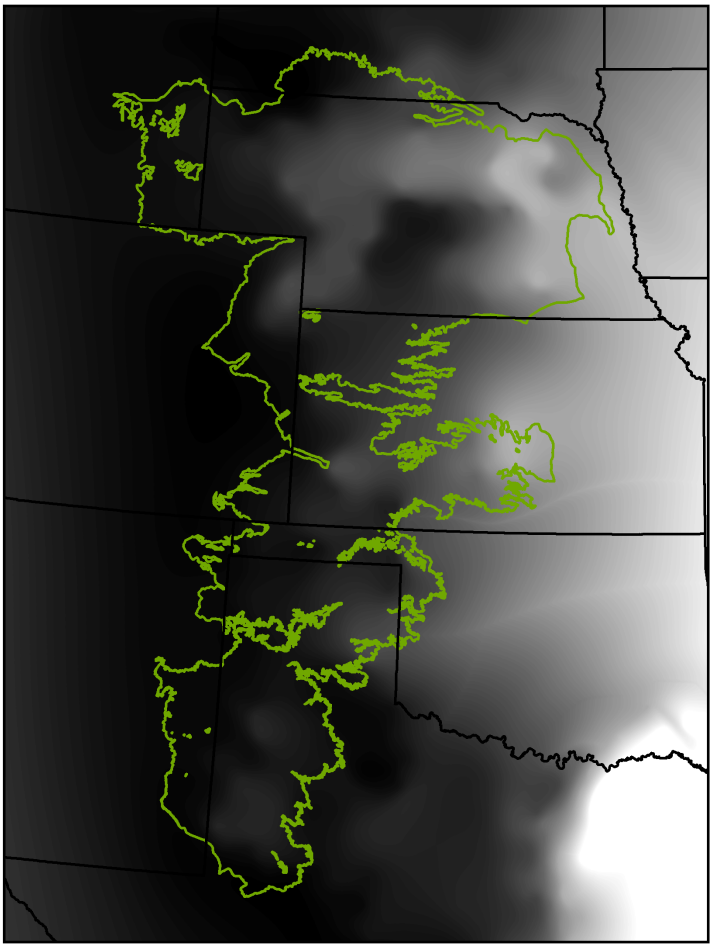

Figure C.6: *High Plains Recharge Raster Input Shapefile*

Appendix D

USGS Stream Gauge Data and Baseflow Results

This appendix contains the tables used to determine monthly average stream flows for the major rivers of the High Plains aquifer. It also contains results of baseflow analysis not presented in section 5.1.4.

D.1 Monthly Stream Flow Data

D.1.1 Canadian River

Table D.1: *USGS Gauging Station 07228000*
USGS 07228000 Canadian River near Canadian, TX

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1964										2.25	71.8	94.6
1965	55.4	51.4	83.7	7.72	115.7	1054	2.9	89.6	27.8	42.3	23.1	48.7
1966	36.9	85.9	25.3	15.4	1.39	0.339	58.9	26.6	88.4	1.43	4.97	22.4
1967	59.9	40.1	23	36.2	219	102.6	166.6	26.3	22	4.78	13.8	44.8
1968	82.5	63.4	60.8	4	140.1	381.5	36.7	27.5	3.07	425.6	34.9	45.5
1969	85.1	116	163	35.7	549.9	224.4	12.2	43.5	127.6	31.4	42.7	76.6
1970	50.9	55.5	83.2	81.9	6.65	2.99	0.019	0.082	266.4	11.8	24.7	47.3
1971	54.8	97.7	59.4	20.9	3.46	216.9	97.4	36.3	158	277.4	848	490
1972	292	146	43.3	18.8	51.6	23.8	13.5	52.6	19.6	3.53	32	32.4
1973	36.8	78.1	473	555	68.8	8.4	18.6	2.29	1.19	2.08	11.1	29.1
1974	38.3	59.7	327	42.5	17.1	16.7	0.32	11.8	5.51	25.5	39.3	33.9
1975	83.7	115	69.8	64.7	101	60.9	80	47	0.729	0.349	24.7	39.7
1976	54.6	48.7	74.2	80.2	77.1	15	0.206	0.261	12.5	15.2	29.2	33.1
1977	31	62.3	38.2	114	1022	331.5	2.13	89.4	37.7	13.4	33.6	34.4
1978	32.8	113	68.2	45.7	527.9	674.9	5.88	28.2	71.8	12.3	31.5	46.9
1979	109	74.5	199	52	366.6	199	51	8.33	9.15	3.89	52.7	45.4
1980	76.9	104	89.6	137	273.1	27.9	0.851	0.019	0.02	0.783	9.66	27.9
Mean of monthly discharge	73.8	82	118	82	221.4	208.8	34.2	30.61	53.23	51.41	78.1	70.1

D.1.2 Cimarron River

Table D.2: *USGS Gauging Station 07156500*
USGS 07156500 Cimarron River Near Satanta, KS

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1942										53	0	3.77
1943	5.1	1.04	0	0	0	0.8	1.52	25	0	0	0	0
1944	0	0	0	113	230.5	34.5	3.32	0	0	0	0	0
1945	0	0	0	0	0	12.8	0	50.4	0	0	0	0
1946	0	0	0	0	77.1	17.2	0	11.3	0			
Mean of monthly Discharge	1.3	0.26	0	28	77	16	1.2	22	0	13	0	0.94

Table D.3: *USGS Gauging Station 07156800*
USGS 07156800 Cimarron River Near Liberal, KS

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1895										16.9	17.1	17.4
1896	17.9	18	18	19.9	17.9	17	17.4	16.5	16			
1938				28.7	67.9	541.8	301.2	147.6	491.3	75.2	27.4	29.6
1939	30.7	25.2	33.2	26	111.2	46.5	49.7	43.5	6	12.2	18.7	21.8
1940	23.3	37.9	20.2	20.9	52.7	50.2	8.19	19.6	9.5	58.4	15.5	21.4
1941	23.7	31.4	24.2	23	1,014	430.4	370.1	65	1,533	131.5	63	35.7
1942	39.7	35.4	32.5	2,291	143.1	531.6	70.2	57.4	167.5			
Mean of monthly discharge	27	30	26	402	234	270	136	58	371	59	28	25

Table D.4: *USGS Gauging Station 07157000*
USGS 07157000 Cimarron River near Mocane, OK

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1942										154.6	56.7	75.9
1943	73.6	53.7	53.1	51.5	57.6	26.9	25.7	37.2	15.1	34.8	49.5	53.1
1944	93.2	66.2	65.4	170	338.1	75.8	41.7	21.7	33.2	56.7	52.5	62
1945	85.8	61.5	59.8	51.8	39.2	120.6	39.3	62.6	26.6	55.6	47.9	41.9
1946	74.8	79.9	59.9	62.2	139.1	45.2	14.6	66.1	52.9	368.5	170	82.2
1947	88.4	86.6	114	126	89	82.8	87.9	36.7	22	42.9	72.7	92.6
1948	56.6	105	78.5	55	58.6	152.4	53.9	197.5	171.1	58.1	125	82
1949	84.4	147	92.2	99.9	184.5	620.8	127.8	76.1	93.4	105.1	84.7	95.6
1950	88	88.5	71.1	65.5	57.5	92.5	327.3	466.5	202.1	159.5	81.3	72.4
1951	82	133	77.1	76.5	1040	317	152	141.8	73.8	64.2	88	87.9
1952	105	78.7	84.3	91.5	73.8	35.4	28.5	62.9	49.7	61.3	76.6	127
1953	79.5	76.5	70.7	63.4	54.5	31.3	108.5	273.6	69.7	62.9	71.4	76.1
1954	98.8	100	64.6	72.7	53.9	33.8	84.9	207.1	35.4	85.5	59.7	69.7
1955	90.6	98	74.9	253	1101	102.4	57.9	87.9	56.6	58	60.9	85.3
1956	61.7	87.2	79.3	54.4	67.1	37.4	88.6	88	20.2	38.1	68.4	66.4
1957	60.5	80	104	84.8	328.3	175.4	116.5	222.5	189.7	71.6	68.7	78.8
1958	72.9	62.5	74.9	106	64.1	352.8	592.5	128.2	69.8	54.8	76.9	64.7
1959	73.8	73.6	83	70.5	94.6	38.7	42.1	37.7	26.6	49.2	60.8	90.8
1960	85.9	100	97.9	62.1	62.9	72.7	41.9	21	40.6	58.8	53.3	54.9
1961	63	55.5	95.6	64.3	67	68.1	97.4	109.7	67.5	108.4	149	43
1962	59.5	64.6	68.5	73.6	58.9	125.6	51.5	31.5	105.9	72.6	76.3	73.4
1963	31.8	75.3	66.5	58.1	69.9	101.5	79.8	47.4	133.8	65.2	57.9	53
1964	85	69.2	65.5	66.8	74	82.7	22.5	23.2	31.5	43.2	79.5	86.4
1965	67.1	61.6	63.1	81.3	122.8	795	187.2	379.6	153.5			
Mean of monthly discharge	77	83	77	85	187	156	107	123	76	84	78	75

D.1.3 Arkansas River

Table D.5: *USGS Gauging Station 07139000*
USGS 07139000 Arkansas River At Garden City, KS

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1922										1.34	7.15	7.44
1923	7.5	14.5	14.2	15.1	360.3	1197	767.2	3949	1611	2751	895	673
1924	680	850	903	1278	224.5	93.4	1.32	0	0.667	3.32	17.6	91.5
1925	89.5	131	18.9	15	12	4.81	914.9	514.7	3	0.158	22.4	25.9
1926	101	35.7	7.77	12	2.9	1.72	0	0	0	0.161	5.03	8.61
1927	10.9	13	14.6	8.87	0.871	2.27	1144	2099	25.3	8.35	19.1	23
1928	106	46	32.8	35.9	849.4	2439	339.7	27.8	0.133	11.1	63.7	310
1929	164	97.4	148	15.3	20.6	2.43	0	1326	6.2	12.5	142	211
1930	90.3	116	34.9	11.2	60.8	10.3	0	15.6	3.9	640.4	148	369
1931	236	26.9	90.2	324	88.4	35.8	0.387	3.39	0.667	0.323	0.33	6.87
1932	23.5	41.3	14.4	8.8	4.74	96.3	0	0	0	1	2	2
1933	2	4.32	3.87	0.9	320.4	4.13	0	253.9	226.5	0.871	13.3	10.1
1934	39.3	46.8	61.4	2.67	1.16	0	0	0	24.5	0.71	4	3.81
1935	2	2	0	0	455.1	293.5	431.4	121.7	25.3	0.806	2.67	20.8
1936	3.48	18.4	1.45	0.17	1048	684.8	235.8	1230	20.1	2.13	10.9	8.32
1937	4.97	67.1	22.8	1.17	0	239.2	0	0	273.2	0	3	4.29
1938	1.16	1.46	1.81	7.13	10.1	84.6	136.3	0	298.6	1.94	8.87	64.8
1939	78.9	64.5	130	7.7	0.226	0.2	0	0	0	0	3.87	3.84
1940	2.87	2.83	2.71	1.13	0.323	1.43	2.9	0.258	0	2.61	7.5	8.35
1941	6.74	4.32	1.68	3.8	259.6	761.3	397.8	37.8	63.5	660.4	1023	504
1942	655	706	479	5556	4693	4082	763.8	698.5	504.3	509.6	652	485
1943	757	143	276	29.9	14.2	44.9	1.52	0.29	2.2	0.839	7.47	10.3
1944	31.3	20.9	29.5	120	613.8	1722	479.8	86.3	88.7	96.4	35.1	136
1945	183	238	127	44.6	6.74	13.9	47.4	41.7	6.97	72	74.3	68.2
1946	162	164	69.7	7.57	7.97	18.5	4.16	1.74	9.83	163.4	349	198
1947	198	123	273	198	602.9	1200	1696	42.9	8.87	14.6	80.1	86.8
1948	150	175	186	67.4	47.9	101.2	145.3	132.8	66.9	41	142	181
1949	124	255	180	86.4	101.2	1915	218	220.6	273.8	186.5	238	212
1950	215	169	71.2	35.7	25.8	72.9	157.4	567.8	354.9	211.6	195	139
Mean of monthly discharge	147	128	114	282	351.2	540.1	281.6	406.1	139.2	186	144	134

Table D.6: *USGS Gauging Station 07140500*
USGS 07140500 Arkansas River At Larned, KS

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1922							32	3.02	0	0	0	0
1923	0	4.06	13.9	5.21	331.3	1628	753.8	3045	1802	2609	1197	679
1924	750	1100	1055	1326	384.7	60.8	5.16	3.58	0	0	0	0
1925	0	183	37.9	30.8	69	0.033	404.5	653.9	36	19.6	83.1	40.4
1926	216	152	81.9	132	53.6	0	0	0	0.467	0	0	0
1927	0	58.6	49.7	356	44.7	33.5	462.9	2771	208.6	50	45.7	32.8
1928	83.1	86.3	90	102	799.4	2313	462.4	45.9	0.9	0	60.3	260
1929	132	70.7	218	59.9	67.1	9.13	1.97	1515	181.3	84.1	237	376
1930	99.5	234	68.3	45.6	108.2	36	0.645	0	4	549.6	313	490
1931	331	203	287	627	228.5	59.9	4.87	0	0	0	0	12.2
1932	25.5	77.1	52.8	49.4	24.4	173.7	264.9	8.94	0.367	0	5.9	10.3
1933	41.3	38.6	36.3	58.3	276.8	23	8.65	47.1	386.1	33.5	31.5	50.1
1934	48.7	50.2	74.9	46.3	15.9	10.8	0	0	15.1	0	0	2.81
1935	23.7	27.1	20.5	13.2	873.8	526	186.7	47.7	84.9	5.81	16.4	27.7
1936	26	9.45	37.5	19	360	978.9	30.1	1123	49.8	24.8	32.3	40
1937	9.94	68.5	85.3	45.4	19.3	197.7	61.1	2.84	167.3	9.39	7.23	9.16
1938	20.5	16.3	39.5	61.1	128.1	128	119.3	11.3	159.7	20.3	21.6	65.6
1939	119	72.1	162	62.5	18.3	3.93	71.3	17.3	0	0	0	0
1940	0	35.8	68.5	55.2	288.7	391.7	32.1	30.8	3.33			
Mean of monthly discharge	107	138	138	172	227	365	153	491	163	189	114	116

Table D.7: *USGS Gauging Station 07144300*
USGS 07144300 Arkansas River At Wichita, KS

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1934										97.9	72.8	55.6
1935	45.5	64.7	63.2	58.1	3,037	3440	1591	233.8	221	172.9	152	132
1936	120.6	69.4	132	85	489.2	1275	115.5	636	105.5	377.9	138	74.2
1937	59	672	266	168	193.7	571.5	701	176.6	418.6	92	69.4	72.6
1938	61.1	68.9	156	181	1342	2384	551.2	718.1	521.2	99.2	354	136
1939	142.4	94.7	286	285	132.3	1060	816.7	955.2	186.8	85.5	81.1	94
1940	46.8	142	223	257	1147	869.3	467.5	110.3	330.2	86.4	134	134
1941	184.8	290	288	224	344.1	2458	1640	392.8	684.2	2119	1582	1088
1942	736.5	958	898	2894	6228	4894	2560	1573	2025	2451	1076	1078
1943	1277	1132	689	652	545.7	775.9	907.9	342.6	438.1	428.4	273	200
1944	407.7	376	1757	6891	4876	2882	1765	1594	1058	1113	558	1582
1945	742.5	846	1682	4115	1402	979.9	1053	513.1	1322	1071	425	348
1946	681.6	544	635	382	269.4	272.5	138.6	51.9	417.6	1567	1081	694
1947	508.8	460	926	2647	1748	3459	2367	496.8	167.8	135	159	348
1948	236.5	776	2514	703	431.7	2879	7626	2820	929.1	417.7	597	816
1949	1824	5278	1875	1163	3896	6568	1670	824.1	806.9	1007	631	636
1950	561.3	670	577	436	384.9	652.7	1667	9202	3136	2423	814	746
1951	680.7	781	834	991	9215	8851	12080	2405	3573	1678	1331	985
1952	1156	979	1397	1912	1468	721.5	262.6	231.5	106.8	108.7	203	333
1953	416.8	380	492	324	355.7	153	379	843	136.7	92.3	178	304
1954	208.5	340	300	232	508.1	423.8	59	29.6	30.4	27.1	60.5	88
1955	130	150	175	284	563.4	1254	384.9	87.2	608.3	1417	210	197
1956	193.5	263	238	203	156.6	118.7	117.5	14.2	7.9	10.2	30.7	23.4
1957	18.8	53.7	240	846	5326	4309	4195	622.9	1613	820.5	617	494
1958	451.4	451	2322	1934	2464	1195	6939	4057	2503	992.1	610	626
1959	572.7	801	842	807	2320	684.6	1046	366	772.1	2457	692	590
1960	708.3	1270	4409	1800	2383	2133	705.5	1773	998.4	549.2	439	443
Mean of monthly discharge	468.2	689	931	1172	1970	2126	1993	1195	889.1	811	465	456

D.1.4 Republican River

Table D.8: *USGS Gauging Station 06824500*
USGS 06824500 Republican River At Benkelman, NE.

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1947	105	118	189	197	136.3	124.5	51.5	6.4	21.5	42.6	82.1	99.7
1948	83.2	151	145	89.1	165.3	381.4	90.9	43.3	25.1	53.9	104	102
1949	76	164	180	221	201.7	235.2	47.3	50.5	78.4	110.5	115	85.6
1950	108	136	121	111	104	42.6	51.9	248.8	122	73.6	108	126
1951	93.9	108	106	125	256.8	266.3	78.8	120.1	375.9	109.7	132	90.4
1952	123	159	166	166	155.8	32.2	9.02	3.41	12.3	40.6	63	78.9
1953	128	111	145	120	90.9	42.8	5.72	3.26	1.7	40	87	96.2
1954	72.3	116	116	64	75.5	23.5	0.632	19.5	12.3	54.8	94	85.4
1955	84.4	84.3	103	76.3	329.4	277.6	23	11.1	9.94	43.4	64.8	74.5
1956	79.4	116	109	73.8	42.4	21.6	48	203.4	22.7	41.5	108	115
1957	74.5	113	150	176	261.3	181	130.8	26.7	26.2	71.2	96.3	102
1958	108	101	173	171	166	93.4	94.7	126.2	63.2	65.5	99	129
1959	107	144	154	162	113.2	35.7	11.5	3.99	12.4	78.4	85.6	91.4
1960	78.9	120	538	136	114.5	68.2	49.6	0.813	10.9	51.8	96.8	108
1961	110	133	163	121	113.6	47.1	20.2	63.5	46.1	69.1	112	95.5
1962	98.3	150	152	98.1	77.6	269.5	477.4	115.6	66.6	89.8	113	108
1963	83.1	149	134	77.4	36	19.9	1.4	26.7	102.1	66.9	94.4	76.7
1964	112	140	114	143	52.2	90.7	8.46	0.442	6.89	29.3	69.5	78.8
1965	95.6	97.9	96.4	73.7	29.3	139.4	256.5	150.3	121.9	116.3	115	108
1966	112	127	132	125	41.2	84.8	76.3	90.5	82.5	79.9	99.5	109
1967	127	116	111	69.4	93.1	138.6	63.4	9.74	28.9	53.2	93.4	84.9
Mean of monthly discharge	98.1	126	157	124	126.5	124.6	76.05	63.06	59.49	65.81	96.7	97.4

Table D.9: *USGS Gauging Station 06844500*
USGS 06844500 Republican River near Orleans, Nebr.

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1947										82.1	266	282
1948	384	535	913	385	292	2732	785.5	282.1	104	107.9	292	367
1949	222	772	924	915	1154	1270	281.1	228.3	259.6	235.1	358	239
1950	215	618	476	401	693.9	193.9	580.7	962.2	251.4	161.1	204	228
1951	209	359	382	386	1528	1685	1233	552.2	2026	332	399	284
1952	311	689	763	775	733.1	144.9	180.6	85.7	13.4	35	123	190
1953	392	422	447	423	440.4	180.8	99.9	29	0.283	0.519	85.6	101
1954	93.9	242	205	148	269.8	80.2	12.1	106.9	0.47	27.3	73.4	97.8
1955	75.8	120	276	169	68.8	507.9	41.8	3.51	2.9	8.58	62.7	56.7
1956	128	147	197	160	54.8	233.5	129.6	48.8	0.227	3.03	83.8	98.4
1957	73.2	170	189	336	1518	1934	488.3	162.8	200.6	202.5	297	217
1958	248	510	709	793	740.6	463.2	413.9	194.2	187.2	166.9	281	378
1959	324	514	629	700	555.1	263.2	86.7	54.1	59.8	136.6	175	210
1960	126	505	1720	814	1124	1241	264.3	105.5	64.3	118.1	157	177
1961	153	224	261	268	690.7	605.5	96.4	132.9	66.6	111.3	146	104
1962	114	222	332	321	347.2	1818	1602	1396	334.8	314	267	257
1963	276	630	574	353	142.6	373.8	36.9	57.8	222.8	133.1	152	101
1964	137	196	331	480	233.6	298.3	129.2	257.1	65.9	82.4	110	93.1
1965	143	148	238	237	613.7	680.9	761.2	172	456	840.2	519	438
1966	331	709	609	523	235.5	270.3	217.6	368.5	283.3	201.1	257	193
1967	223	334	247	183	206.9	1680	1069	200.9	162.6	158	222	221
Mean of monthly discharge	209	403	521	438	582.1	832.8	425.5	270	238.1	164.6	216	206

D.1.5 Platte River

Table D.10: *USGS Gauging Station 06686500*
USGS 06686500 North Platte River At Oshkosh, NE

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1930										3289	3090	2523
1931	2466	2199	1896	1648	1,185	705	464	606.9	803	1474	1441	1479
1932	1228	1381	1445	1,538	1050	1118	1157	1020	1004	2037	1957	1565
1933	1924	1634	1796	1616	3603	1106	931.4	980.7	2451	2285	2326	1837
1934	1758	1506	1505	1230	463.5	391.9	48.5	20.2	174.3	482.9	500	1187
1935	1471	1165	890	998	1414	2676	303.4	280.1	314	807.1	1272	1239
1936	1116	1034	1512	1094	414	729.2	283.9	279.1	239.9	960.9	1571	1270
1937	886	1328	1441	895	365	1038	823.9	315.1	682	1554	1542	1522
1938	1365	1655	1416	1621	1895	1005	920.8	653.1	1872	1580	1762	1531
1939	1538	1220	1722	1389	450.2	866.2	326.9	270	403.8	1113	1231	1220
1940	1098	1615	1254	1073	545.7	417.1	222.5	81	175.6	759.4	792	1022
1941	959	986	999	1036	841.8	1072	406.2	378.7	690.6	1456	1280	1181
1942	1176	1354	1204	1408	5207	1678	666.4	630.1	1115	1802	1781	1658
1943	1618	1627	1464	1985	1121	991.4	510.5	385.5	841.1	1499	1707	1554
1944	1311	1299	1419	1362	1619	1290	1091	731.5	960.3	1646	1672	1393
1945	1267	1375	1386	1385	1824	2806	903	1443	1442	2050	1697	1275
1946	1402	1467	1628	1165	1157	913.8	553	360.6	1439	1908	1761	1507
1947	1421	1354	1423	1205	856.8	3397	2360	673.9	1129	1836	1896	1817
1948	1402	1784	2040	1497	872.7	1444	1115	910.3	1332	1836	1864	1817
1949	1085	1785	1745	1348	1259	1810	760.7	762.2	1464	1951	1730	1534
1950	1104	1411	1449	1134	822.7	571.6	833.2	928.7	1769	1946	1706	1675
1951	1420	1559	1251	1161	804.5	1597	1338	830.8	2452	2227	1810	1428
1952	1424	1464	1600	1603	1922	3013	764.2	641.7	1191	2064	1781	1759
1953	1700	1469	1413	1253	828.7	990.9	523.4	1296	846.4	1529	1528	1303
1954	1198	1144	1210	998	752.8	411.2	160.3	340.2	374.2	916.8	1125	1056
1955	878	922	1117	1020	754.9	848.7	552.1	215.8	534.2	1104	1143	967
1956	949	875	945	782	437.5	258.4	763.8	148.5	294.4	795.5	1228	1073
1957	808	956	990	1075	1760	1389	622.6	756.7	1149	1484	1394	1227
1958	992	1076	1270	1256	942.3	1538	1398	451.1	992.3	1657	1359	1277
1959	1166	1194	1199	1165	1038	506.2	329.3	198.2	813.2	1735	1317	1183
1960	1098	1033	1202	996	704.2	274.5	79	182.1	394			
Mean of monthly discharge	1310	1360	1390	1260	1230	1230	707	559	978	1590	1580	1440

Table D.11: *USGS Gauging Station 06767998*
USGS 06767998 Platte River Near Overton, NE

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1968										793.3	810	762
1969	841	864	1127	692	742	1011	1278	217.7	568	771.1	965	1133
1970	1127	1431	1431	1,633	1245	1972	641.7	134.5	440.8	1041	915	1288
1971	1442	1430	1604	1733	6655	8473	998.4	244.9	639.7	1355	1459	1653
1972	1857	1934	2439	1364	1626	377.8	332.7	582.9	645.8	1319	1440	1122
1973	1305	1880	1632	1938	9081	9326	970.8	1234	4039	5036	3070	2566
1974	2647	3477	5178	5450	1471	562.8	225.6	364.4	675.1	890.6	888	1033
1975	1112	1150	1165	1231	664.1	1483	414.4	464	842.3	842.4	1060	1199
1976	1469	1558	1441	1478	747.6	385.6	300.1	199.6	786.9			
Mean of monthly discharge	1470	1720	2000	1940	2780	2950	645	430	1080	1510	1330	1340

Table D.12: *USGS Gauging Station 06796000*
USGS 06796000 Platte River at North Bend, NE

YEAR	Monthly mean in cfs											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949				7496	6413	9740	4795	1957	3196	3051	3829	2673
1950	2081	4471	6700	4728	5,131	3665	7387	3603	2,937	4298	3942	4149
1951	3956	5186	6716	6,015	7444	8866	5612	4764	5836	4855	4919	3128
1952	3836	8317	8459	7690	5604	3541	2195	1972	1617	2558	3439	3078
1953	3692	6561	6645	5284	6382	3344	1861	1332	1259	1668	3541	3329
1954	2248	5238	4401	3647	3956	3384	907.9	1555	1601	2471	2836	2763
1955	2325	2972	5151	3330	1952	3889	1310	441.6	936.2	1826	1938	1413
1956	2390	2822	4687	3193	2292	2046	1111	1083	1246	1626	2049	1896
1957	1206	3430	3685	4178	6345	8270	2534	1307	2697	3125	3822	3005
1958	2734	4206	6067	7327	4658	4207	6533	2039	1895	2300	3008	2690
1959	1947	2899	7272	5494	5494	2439	2156	1989	1669	3385	3881	3449
1960	1907	4676	10250	8026	6694	6464	2748	1641	1635	2312	3099	2552
1961	2933	4273	4172	3518	5014	5042	1275	1539	1853	3251	4185	2766
1962	2848	4968	10470	5196	4475	9778	6016	3476	2640	3838	3736	3119
1963	2403	5279	7492	3824	2831	3505	760.6	1329	2633	2282	3246	1997
1964	2677	3917	4809	5560	4404	4916	2186	1742	2913	2693	2627	1935
1965	2697	3371	6029	6260	7058	6510	6059	1687	7207	6252	5175	4893
1966	2960	6807	6522	5344	3194	3337	1235	5873	2532	2710	3246	2356
1967	2852	3429	4312	2881	2471	19110	4953	1743	1588	2906	3848	3103
1968	2758	4500	4962	4061	2993	3682	1691	1528	2256	4552	3902	2412
1969	2956	3709	10900	5475	4284	4263	4590	1318	2467	4095	3917	3483
Mean of monthly discharge	2670	4552	6485	5168	4719	5714	3234	2091	2505	3145	3533	2866

D.2 Additional Baseflow Results

D.2.1 Cimarron River

Table D.13: *Simulated Groundwater-Surface Water Interaction along Cimarron River up to Gauging Station 07156500*

PSO Results	
No. of Interaction Points	96
Cumulative Contribution to Baseflow (cfs)	52.8
Average Interaction Rate (cfs)	0.53

L-M Results	
No. of Interaction Points	80
Cumulative Contribution to Baseflow (cfs)	321
Average Interaction Rate (cfs)	4.0

Table D.14: *Simulated Groundwater-Surface Water Interaction along Cimarron River up to Gauging Station 07156500 (Mean Discharge from 1942-1946)*

	Cimarron River Baseflow (up to station 07156500)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	1.3	0.26	0	28	77	16	1.2	22	0	13	0	0.94
% Derived from Groundwater Discharge (PSO)	4059%	20293%		188%	69%	330%	4397%	240%		406%		5613%
% Derived from Groundwater Discharge (L-M)	24729%	123644%		1148%	417%	2009%	26790%	1461%		2473%		34199%

Table D.15: *Simulated Groundwater-Surface Water Interaction along Cimarron River up to Gauging Station 07156800*

PSO Results	
No. of Interaction Points	112
Cumulative Contribution to Baseflow (cfs)	124
Average Interaction Rate (cfs)	1.1

L-M Results	
No. of Interaction Points	94
Cumulative Contribution to Baseflow (cfs)	390
Average Interaction Rate (cfs)	4.1

Table D.16: *Simulated Groundwater-Surface Water Interaction along Cimarron River up to Gauging Station 07156800 (Mean Discharge from 1895-1942)*

Cimarron River Baseflow (up to station 07156800)												
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	27.0	30.0	26.0	402.0	234.0	270.0	136.0	58.0	371.0	59.0	28.0	25.0
% Derived from Groundwater Discharge (PSO)	461%	414%	478%	31%	53%	46%	91%	214%	34%	211%	444%	497%
% Derived from Groundwater Discharge (L-M)	1444%	1299%	1499%	97%	167%	144%	287%	672%	105%	661%	1392%	1559%

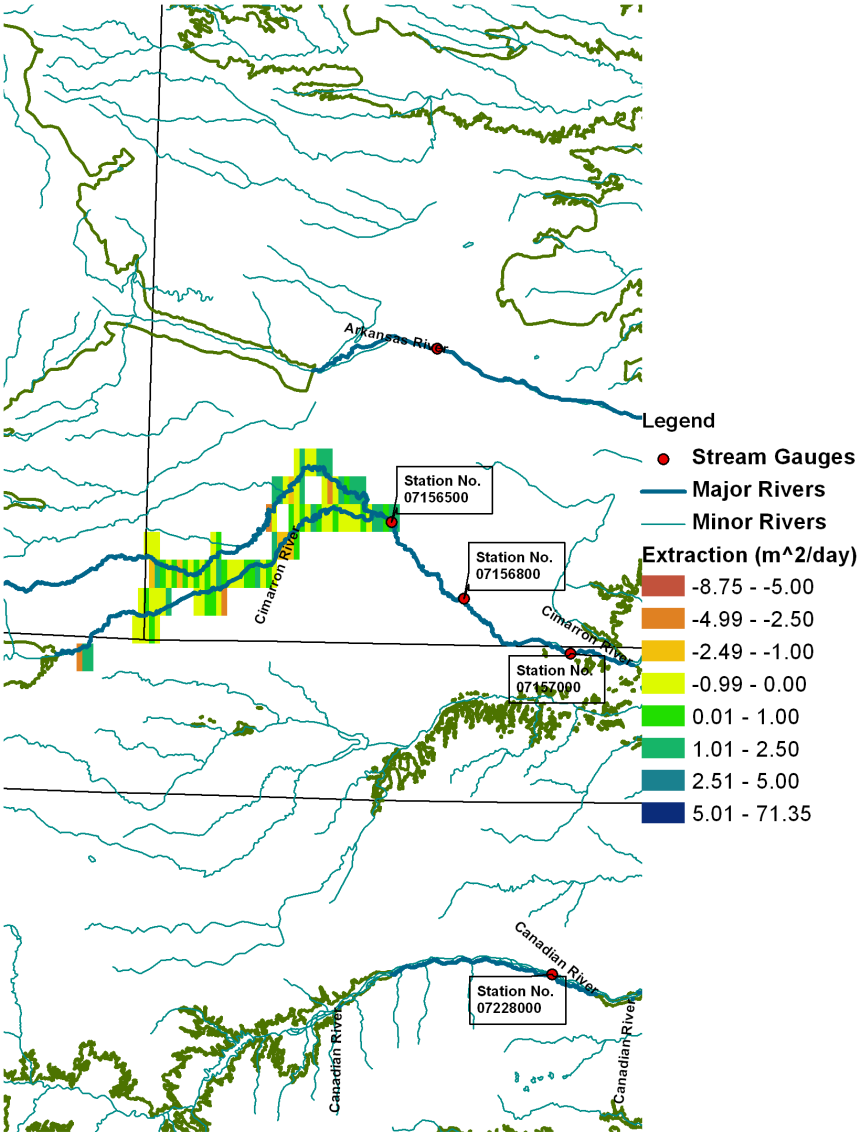

Figure D.1: *Groundwater-Surface Water Interaction Along the Cimarron River (PSO results, Gauging Station 07156500)*

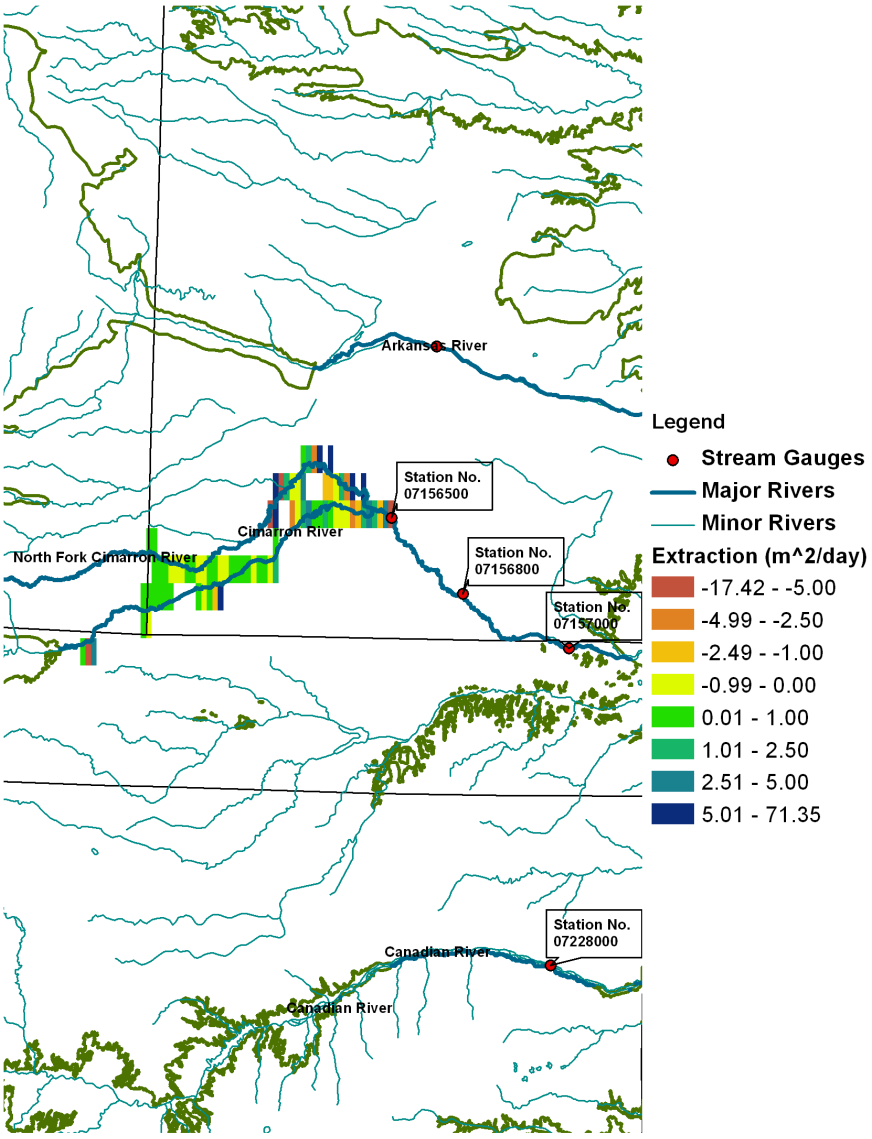

Figure D.2: *Groundwater-Surface Water Interaction Along the Cimarron River (L-M results, Gauging Station 07156500)*

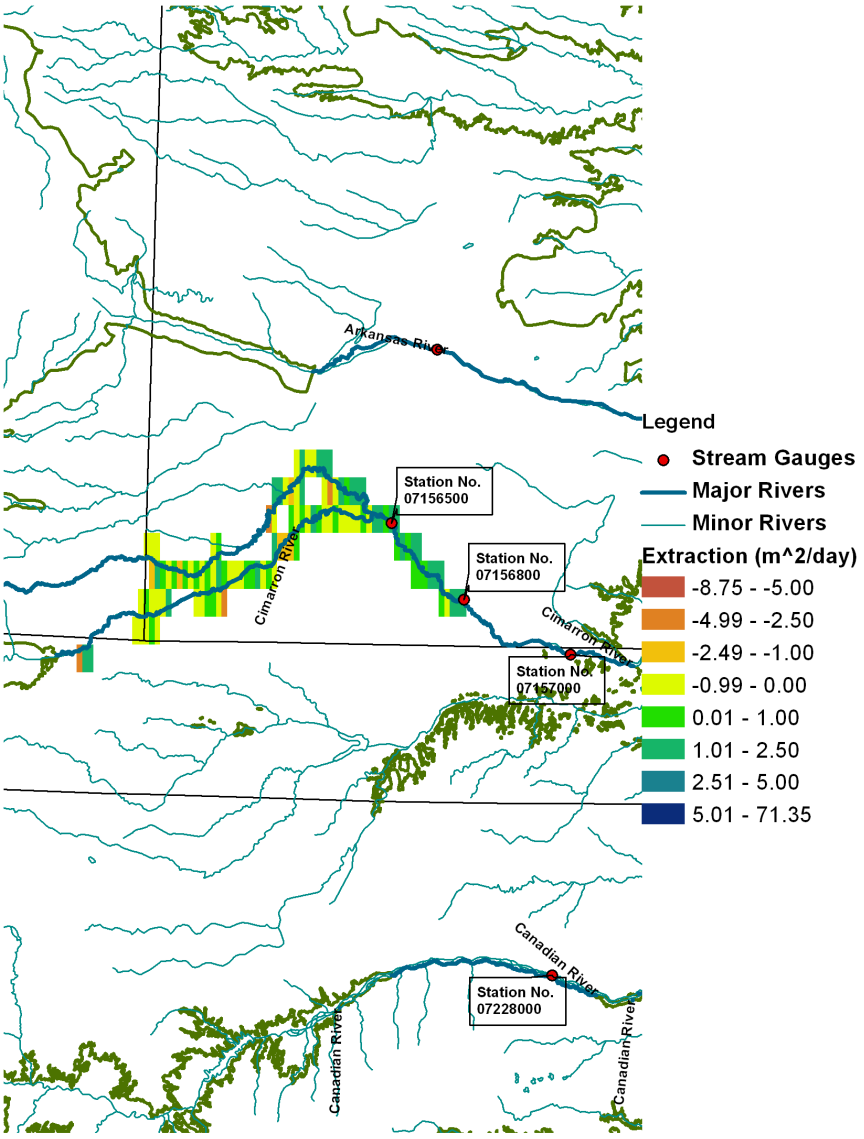

Figure D.3: *Groundwater-Surface Water Interaction Along the Cimarron River (PSO results, Gauging Station 07156800)*

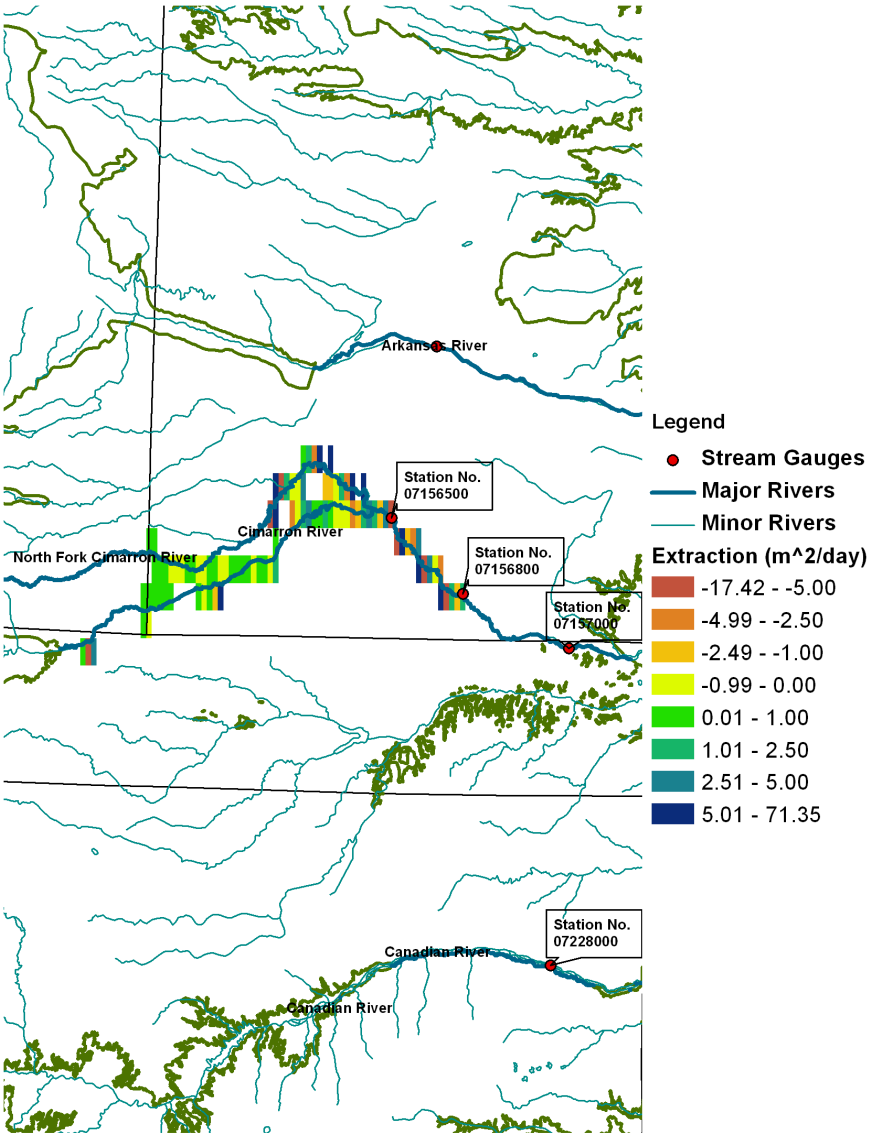

Figure D.4: *Groundwater-Surface Water Interaction Along the Cimarron River (L-M results, Gauging Station 07156800)*

D.2.2 Arkansas River

Table D.17: *Simulated Groundwater-Surface Water Interaction along Arkansas River up to Gauging Station 07139000*

PSO Results		L-M Results	
No. of Interaction Points	31	No. of Interaction Points	27
Cumulative Contribution to Baseflow (cfs)	29.0	Cumulative Contribution to Baseflow (cfs)	45.4
Average Interaction Rate (cfs)	0.94	Average Interaction Rate (cfs)	1.68

Table D.18: *Simulated Groundwater-Surface Water Interaction along Arkansas River up to Gauging Station 07139000 (Mean Discharge from 1922-1950)*

Arkansas River Baseflow (up to station 07139000)

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	142	114	101	296	380	518	217	226	104	122	143	125
% Derived from Groundwater Discharge (PSO)	20%	26%	29%	10%	8%	6%	13%	13%	28%	24%	20%	23%
% Derived from Groundwater Discharge (L-M)	32%	40%	45%	15%	12%	9%	21%	20%	44%	37%	32%	36%

Table D.19: *Simulated Groundwater-Surface Water Interaction along Arkansas River up to Gauging Station 07140500*

PSO Results	
No. of Interaction Points	119
Cumulative Contribution to Baseflow (cfs)	117
Average Interaction Rate (cfs)	0.98

L-M Results	
No. of Interaction Points	101
Cumulative Contribution to Baseflow (cfs)	133
Average Interaction Rate (cfs)	1.31

Table D.20: *Simulated Groundwater-Surface Water Interaction along Arkansas River up to Gauging Station 07140500 (Mean Discharge from 1922-1940)*

	Arkansas River Baseflow (up to station 07140500)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	107	138	138	172	227	365	153	491	163	189	114	116
% Derived from Groundwater Discharge (PSO)	109%	85%	85%	68%	52%	32%	76%	24%	72%	62%	103%	101%
% Derived from Groundwater Discharge (L-M)	125%	97%	97%	78%	59%	37%	87%	27%	82%	71%	117%	115%

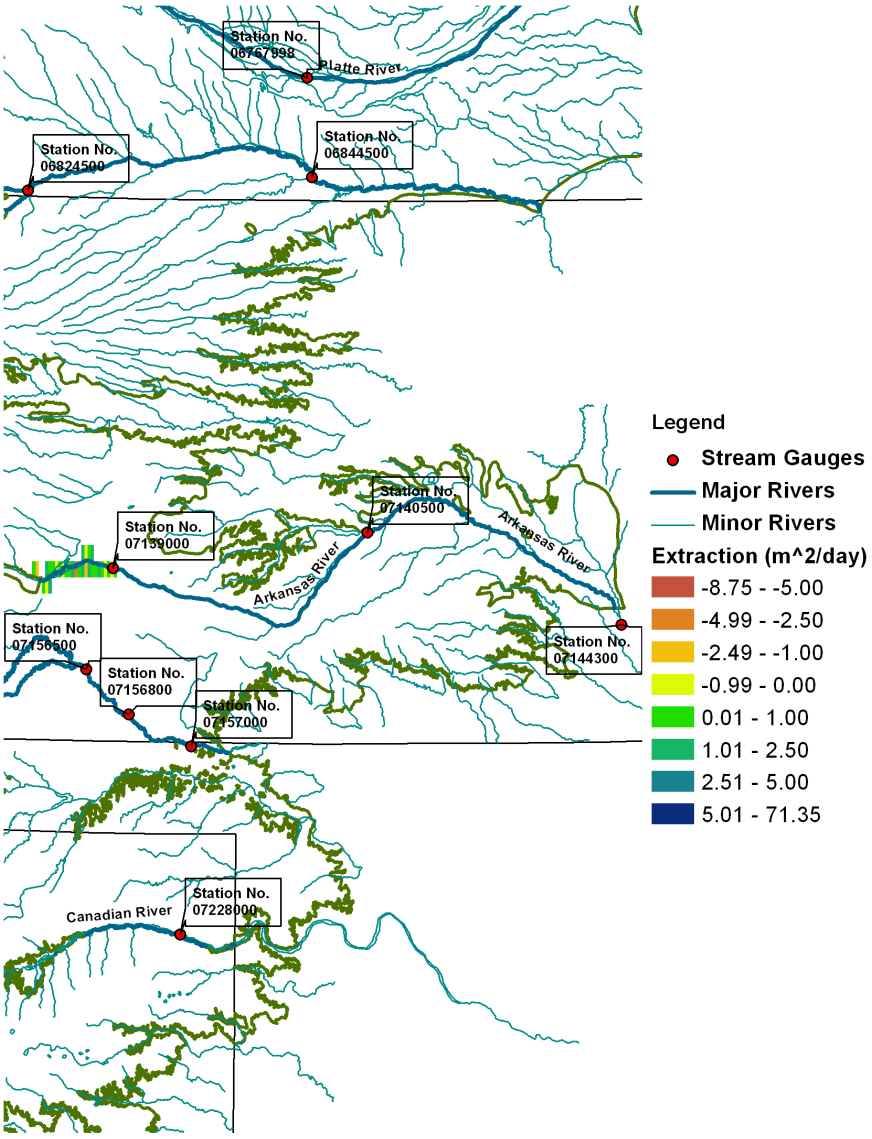

Figure D.5: *Groundwater-Surface Water Interaction Along the Arkansas River (PSO results, Gauging Station 07139000)*

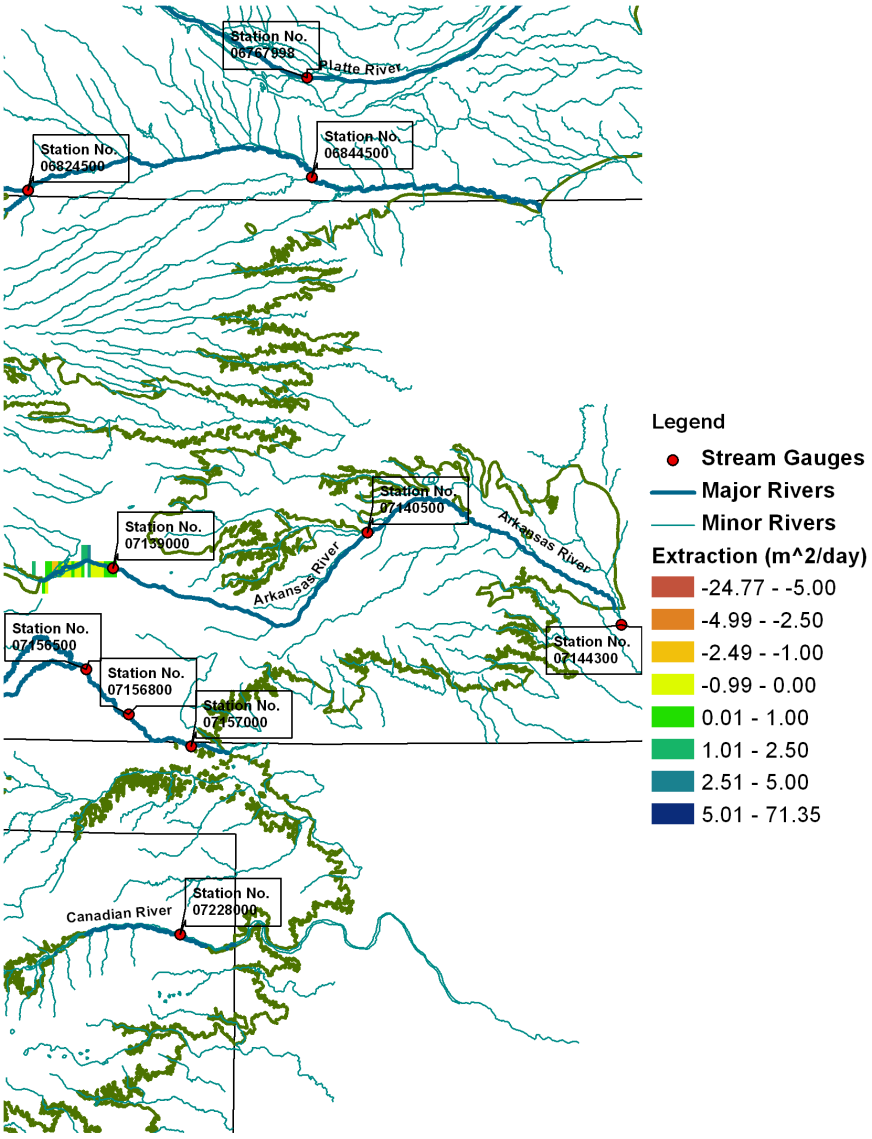

Figure D.6: *Groundwater-Surface Water Interaction Along the Arkansas River (L-M results, Gauging Station 07139000)*

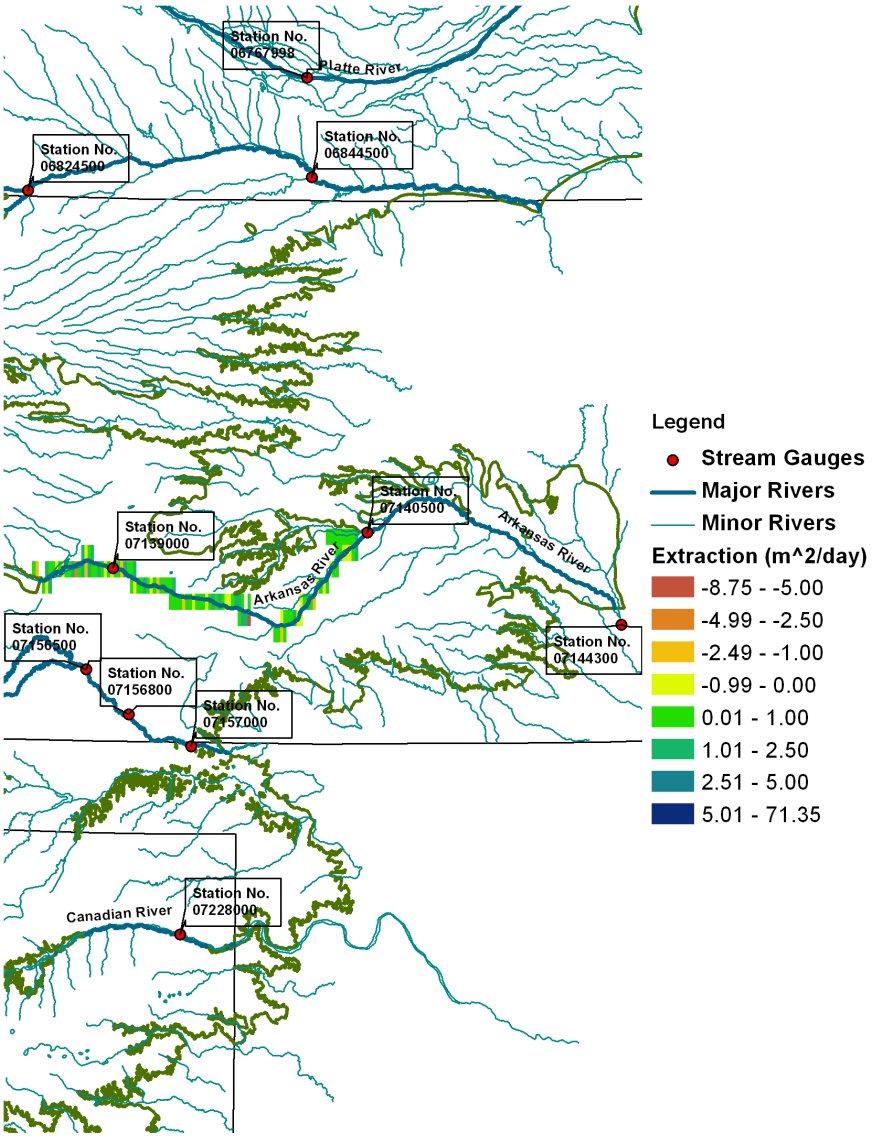

Figure D.7: *Groundwater-Surface Water Interaction Along the Arkansas River (PSO results, Gauging Station 07140500)*

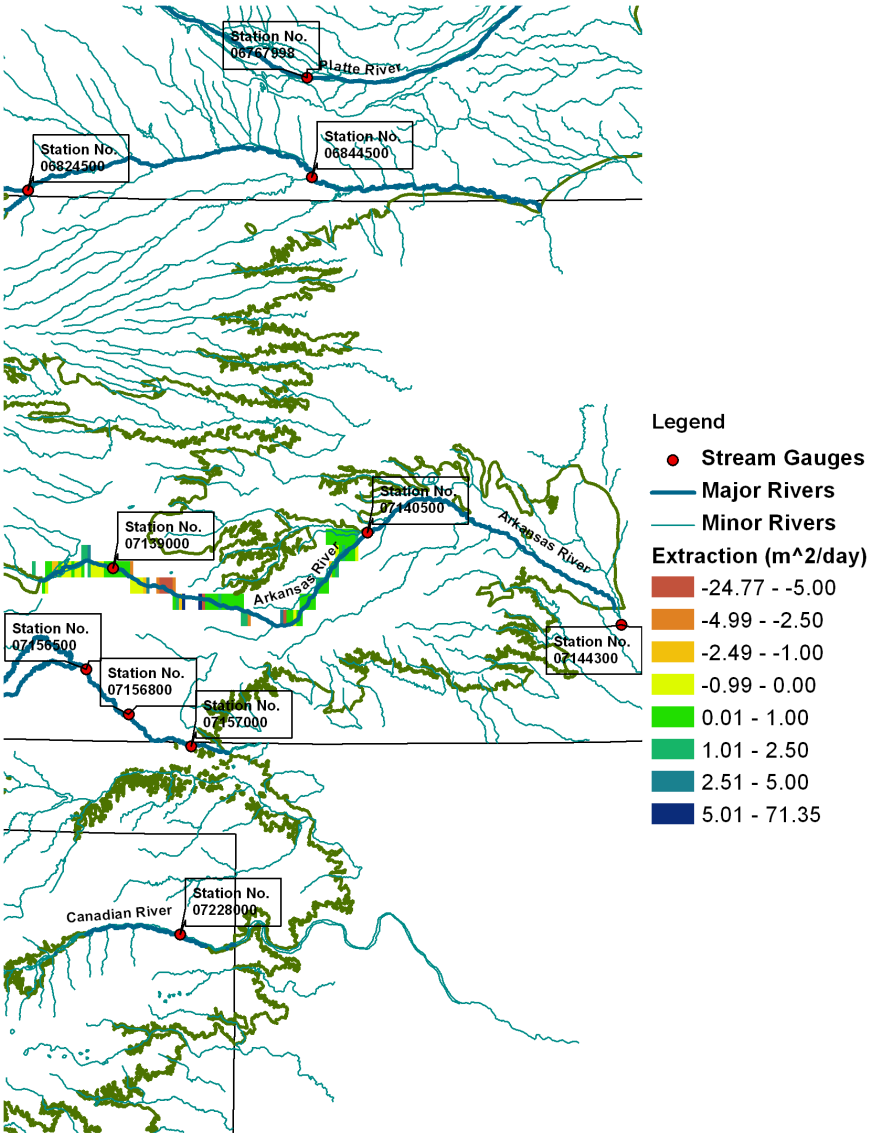

Figure D.8: *Groundwater-Surface Water Interaction Along the Arkansas River (L-M results, Gauging Station 07140500)*

D.2.3 Republican River

Table D.21: *Simulated Groundwater-Surface Water Interaction along Republican River up to Gauging Station 06824500*

PSO Results	
No. of Interaction Points	55
Cumulative Contribution to Baseflow (cfs)	47.0
Average Interaction Rate (cfs)	0.86

L-M Results	
No. of Interaction Points	38
Cumulative Contribution to Baseflow (cfs)	49.1
Average Interaction Rate (cfs)	1.31

Table D.22: *Simulated Groundwater-Surface Water Interaction along Republican River up to Gauging Station 06824500 (Mean Discharge from 1947-1967)*

	Republican River Baseflow (up to station 06824500)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	98	121	159	113	110	104	84	57	42	64	95	97
% Derived from Groundwater Discharge (PSO)	48%	39%	30%	42%	43%	45%	56%	82%	112%	74%	49%	48%
% Derived from Groundwater Discharge (L-M)	50%	40%	31%	43%	45%	47%	58%	86%	117%	77%	52%	50%

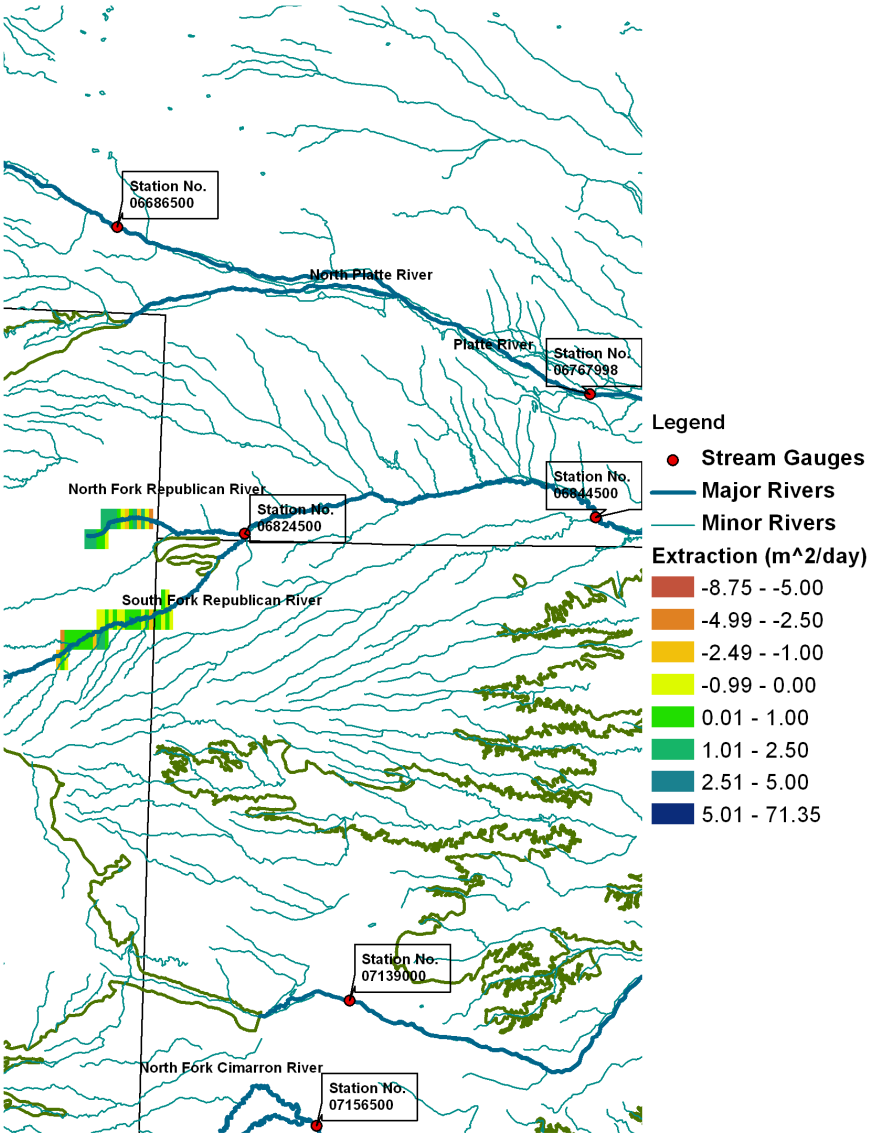

Figure D.9: *Groundwater-Surface Water Interaction Along the Republican River (PSO results, Gauging Station 06824500)*

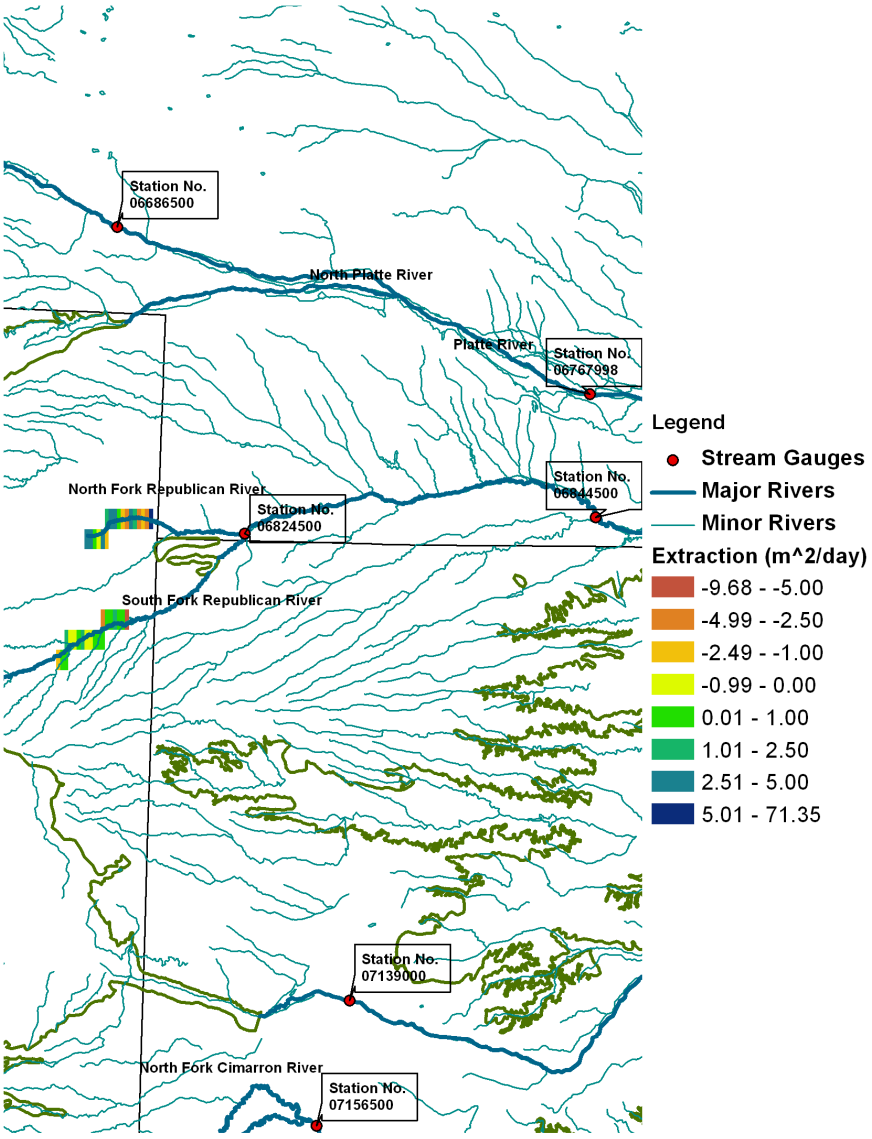

Figure D.10: *Groundwater-Surface Water Interaction Along the Republican River (L-M results, Gauging Station 06824500)*

D.2.4 Platte River

Table D.23: *Simulated Groundwater-Surface Water Interaction along Platte River up to Gauging Station 06686500*

PSO Results	
No. of Interaction Points	115
Cumulative Contribution to Baseflow (cfs)	157
Average Interaction Rate (cfs)	1.35

L-M Results	
No. of Interaction Points	55
Cumulative Contribution to Baseflow (cfs)	484
Average Interaction Rate (cfs)	9.0

Table D.24: *Simulated Groundwater-Surface Water Interaction along Platte River up to Gauging Station 06686500 (Mean Discharge from 1930-1960)*

	Platte River Baseflow (up to station 06686500)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	1310	1360	1390	1260	1230	1230	707	559	978	1590	1580	1440
% Derived from Groundwater Discharge (PSO)	12%	12%	11%	12%	13%	13%	22%	28%	16%	10%	10%	11%
% Derived from Groundwater Discharge (L-M)	37%	36%	35%	38%	39%	39%	68%	87%	50%	30%	31%	34%

Table D.25: *Simulated Groundwater-Surface Water Interaction along Platte River up to Gauging Station 06767998*

PSO Results	
No. of Interaction Points	284
Cumulative Contribution to Baseflow (cfs)	342
Average Interaction Rate (cfs)	1.2

L-M Results	
No. of Interaction Points	114
Cumulative Contribution to Baseflow (cfs)	769
Average Interaction Rate (cfs)	6.5

Table D.26: *Simulated Groundwater-Surface Water Interaction along Platte River up to Gauging Station 06767998 (Mean Discharge from 1968-1976)*

	Platte River Baseflow (up to station 06767998)											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Mean monthly discharge (cfs)	1470	1720	2000	1940	2780	2950	645	430	1080	1510	1330	1340
% Derived from Groundwater Discharge (PSO)	23%	20%	17%	18%	12%	12%	53%	80%	32%	23%	26%	26%
% Derived from Groundwater Discharge (L-M)	52%	45%	38%	40%	28%	26%	119%	179%	71%	51%	58%	57%

Appendix E

Validation of Optimization Methods

This appendix presents results from the application of the Levenberg-Marquardt and PSO optimization techniques to an idealized transect. This was done to verify that the techniques, as they were applied in this study, are capable of locating the optimal parameter set. As shown, these tests show the ability of each method to accurately find predefined extraction sets in an ideal setting.

E.1 Transect Set Up

A test transect was set up with known values of groundwater-surface water interaction represented by the parameter E. The test transect has a base that steadily declines from 500m to 442m over a transect length of 60km. There are 30 cells in the transect with widths of 2km. Hydraulic conductivity for each cell was set to 30m/yr which was a typical value for the High Plains Aquifer. Recharge for each cell was set to 0.000114m/year which was the average value for recharge in the High Plains Aquifer. The set of known groundwater-surface water interaction parameters used are shown below:

E = [-0.0015, 0.0025, 0.002, 0.0015, -0.003] (where these values are removed from specific cells in the tests shown in the following sections)

The locations of the groundwater-surface water interaction parameters were the 6th, 7th, 20th, 21st, and 22nd cells of the transect when counting from left to right from 1 to 30 in

the transect. The groundwater elevations with and without the groundwater-surface water interaction parameters included in the solution are shown in Figure E.1.

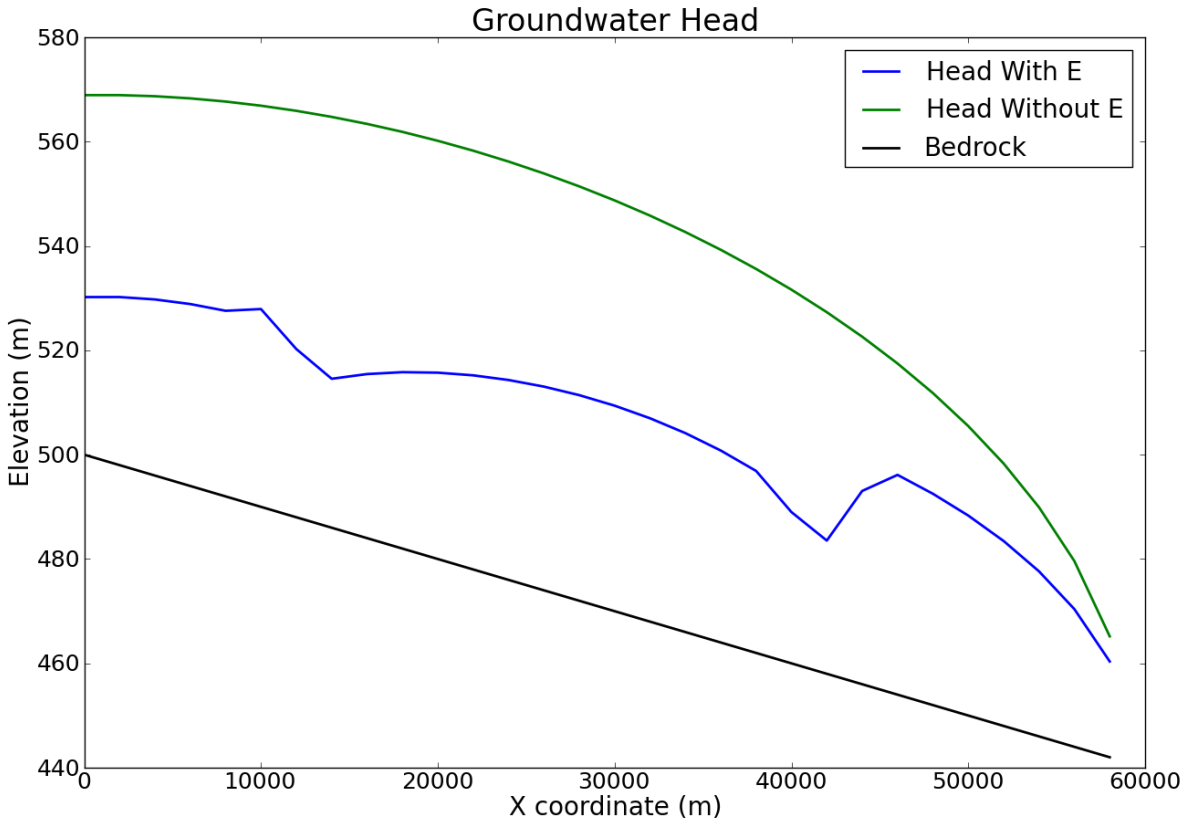

Figure E.1: *Heads with Known Interaction Parameters Included and Excluded*

E.2 Optimization Test Results

E.2.1 Levenberg-Marquardt

Different initial guesses where used to test the Levenberg-Marquardt algorithm. The initial guesses where chosen at random from a given range for each trial. The range was determined by taking the smallest and largest parameters of the known values of E and multiplying them by 1.1 to increase the maximum value and decrease the minimum value

by 10%. The resulting range was $[-.0033, 0.00275]$. The results are shown in Tables E.1-E.2. The Levenberg-Marquardt algorithm was able to find good results with the cumulative percent error being no more than 8.5% for the different trials shown. This confirms that the Levenberg-Marquardt algorithm as it was written for this study is able to minimize the objective function and obtain good parameter results.

Table E.1: *Levenberg-Marquardt Test: Optimized Parameter Values*

Optimized E1	Optimized E2	Optimized E3	Optimized E4	Optimized E5	Objective Function Value
-0.00149	0.00253	0.00198	0.00149	-0.00295	458.3
-0.00152	0.00251	0.00198	0.00151	-0.00293	616.7
-0.00153	0.00250	0.00207	0.00151	-0.00302	649.0
-0.00150	0.00253	0.00193	0.00151	-0.00307	1000.2
-0.00155	0.00252	0.00205	0.00148	-0.00303	936.7

Table E.2: *Levenberg-Marquardt Test: Optimized Parameter % Error*

%Error E1	%Error E2	%Error E3	%Error E4	%Error E5	Cumulative Error
0.39%	1.10%	0.78%	0.82%	1.77%	4.86%
1.27%	0.25%	1.10%	0.78%	2.38%	5.78%
2.06%	0.12%	3.69%	0.55%	0.56%	6.98%
0.19%	1.39%	3.43%	0.42%	2.31%	7.74%
3.39%	0.77%	2.32%	1.20%	0.90%	8.58%

E.2.2 PSO

The initial position of the particles in the PSO test were selected randomly from the range $[-.0033, 0.00275]$. As the optimization procedure was carried out, any parameter with a value larger or smaller than the bounds of this range was penalized by adding 10,000 to the objective function. The results are shown in Tables E.3-E.4. The PSO routine was able to find good solutions with the largest cumulative error being 12.1%. This confirms that the PSO optimization routine used in this study is able to minimize the objective function and obtain good parameter results.

Table E.3: *PSO Test: Optimized Parameter Values*

Optimized E1	Optimized E2	Optimized E3	Optimized E4	Optimized E5	Objective Function Value
-0.00158	0.00247	0.00193	0.00150	-0.00301	1002.1
-0.00151	0.00240	0.00201	0.00143	-0.00304	1085.9
-0.00157	0.00249	0.00200	0.00145	-0.00309	1152.7
-0.00151	0.00243	0.00191	0.00148	-0.00309	1234.3
-0.00153	0.00241	0.00193	0.00147	-0.00303	1462.4

Table E.4: *PSO Test: Optimized Parameter Errors*

%Error E1	%Error E2	%Error E3	%Error E4	%Error E5	Cumulative Error
5.28%	1.29%	3.30%	0.08%	0.41%	10.36%
0.62%	3.98%	0.30%	4.42%	1.49%	10.82%
4.88%	0.26%	0.21%	3.59%	2.94%	11.89%
0.48%	2.66%	4.57%	1.29%	2.90%	11.90%
2.00%	3.62%	3.59%	1.78%	1.10%	12.07%