2017-03-01

Cloud-Based Analytic Element Groundwater Modeling

Jacob Baird Fullerton
Brigham Young University

Cloud-Based Analytic Element Groundwater Modeling

Jacob Baird Fullerton

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Norman L. Jones, Chair
Daniel P. Ames
E. James Nelson

Department of Civil and Environmental Engineering

Brigham Young University

ABSTRACT

Cloud-Based Analytic Element Groundwater Modeling

Jacob Baird Fullerton
Department of Civil and Environmental Engineering, BYU
Master of Science

Managing groundwater resources requires computer modeling software, which in turn brings its own costs in education and usage fees. Although many groundwater modeling programs can be obtained at relatively inexpensive rates the hidden costs of software training remain high in general. What has been done is to make both the software and training available online for free, but this method delegates all responsibility to the users for both accessing the software in addition to learning how to use it properly. In this research the accessibility of groundwater models has been improved upon by creating a web app of an open source software called TimML. I divided up the task of building a general purpose groundwater modeling web app into three different apps with increasing sophistication. The first app I developed simulates construction dewatering using the Dupuit assumptions. The second app also simulates construction dewatering and uses the same basic graphical user interface I developed for the first app while implementing TimML as the back-end processor. The culminating app, TimML-Cloud, builds on the user interface created in the first two apps and exposes more of the modeling capacity of TimML. By making TimML a cloud app, people with internet access have greater accessibility to an analytic element groundwater model than before. Because TimML is more accessible than before, people can more readily implement TimML as a decision-making tool in water resource management.

Keywords: groundwater, web app, model, TimML, cloud

ACKNOWLEDGEMENTS

The first acknowledgement goes to my wife who has supported me through the last three years of schooling and made this research possible through financial and emotional support. Next I have to express thanks to Dr. Jones who took a chance in taking me on as his student and provided still more funding for my research in addition to his advice and counsel all along the way. Other recognitions go to the Utah Chapter of the American Public Works Association who awarded me their 2016 Stormwater Scholarship and the half-tuition, two-semester scholarship provided by Russell Berrett. Last, but not least, I express thanks here to my mom and dad who helped me to achieve my educational goals and sacrificed so much for me all of my life, thank you!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Freshwater resources fuel life and the economy. Less than 3% of all water on the earth's surface is freshwater. Of all freshwater only 0.3% is found in surface water bodies while 30% is found in groundwater and more than 68% of freshwater resides in glaciers and icecaps (Mohan et al., 2012). Groundwater dependent ecosystems (GDE's) like the arid Western United States cannot supply their water needs using only surface water. In their paper on Australian GDE's, Murray, Zeppel, Hose, and Eamus (2003) define a GDE as an ecosystem that requires groundwater to maintain its current composition and function. California is a prime example of a GDE, where in 1995 it was estimated that 13 million people used groundwater to supplement their drinking water supply (Howard & Merrifield, 2010). California's population has continued to grow since 1995, increasing freshwater demand for and dependence on groundwater, especially during drought seasons. As another example of a GDE, Spain's agricultural users draw about 90% of their irrigation water from the Western la Mancha aquifer (Esteban & Dinar, 2016). For GDE's to endure prolonged droughts, good management practices must be implemented. Computer models are vital for sustainable groundwater use and informed management decisions.

Computer models provide a mathematical basis for estimating groundwater supply and trends. There are two primary types of groundwater models: the analytic element and the grid-based numerical method. Analytic element models (AEM's) are based on points, lines, and polygons and solve a set of equations that for a set of XY locations satisfy the stresses, boundary

conditions, and aquifer properties defined by the spatial features. Numerical models include the finite difference and finite element methods and require extensive data input and use computational grids consisting of cells or element meshes. Modelers can better simulate where water comes from and where it may go in the future by using groundwater models.

1.1 Web-Based Modeling

A recent trend in water resource modeling is to host models on the cloud via a web interface. Hosted models have the versatility of being usable by virtually any machine with an internet connection and also serve to reduce the need for high-end computing power on the local machine. As an example, online models are being implemented in Germany for the Danube watershed under the Global Change of the Water Cycle initiative (GLOWA) to create a management system using a web-based distributed system (Ludwig et al., 2003). The various models that make up the GLOWA-Danube project are hosted on several computers with data transmission limited only to necessary results and parameters, increasing computer capacity and allowing for greater collaborative involvement and use (Ludwig et al., 2003).

Hosting modeling services decreases the need for local expertise in performing small, repetitive processes. Jones (2012) created a server-based MODFLOW model that allows users to evaluate the impact of new wells. With a more traditional well-permitting review process, a technician would manually modify and update a local version of a given MODFLOW model to determine how the new well(s) would impact water levels and discharge to nearby streams and springs. Online services make repetitive processes like well permitting easier to perform, reducing the need to maintain staff with modeling expertise.

Another example of an online service aimed at lowering the threshold of expertise is the Tethys Platform (Swain, 2015). The Tethys Platform provides a website environment where water resource apps, models, and data resources can be developed and hosted with minimal coding effort. Tethys reduces the code required for developing and hosting web applications by supporting prebuilt tools that function similar to "plug-and-play" devices. To implement the use of plots, maps, geoprocessing tools or other user interface methods, the developer only requires a few lines of code to access any one tool.

A contrasting example to the GLOWA initiative and Tethys Platform is the public domain wellhead protection software supported by the United States Environmental Protection Agency (EPA) called WhAEM 2000. WhAEM 2000 facilitates wellhead protection mapping by helping its users to delineate well capture zones. In order to use WhAEM 2000, the correct Microsoft Windows operating system and patch must be installed on a local computer and then WhAEM 2000 can be downloaded and installed on the local computer. Base maps used by WhAEM 2000 must be downloaded prior to use (Kraemer, Haitjema, & Kelson, 2007) and any other geographical information must be provided by the user. All computations performed by WhAEM 2000 are done on the local machine using its computational resources. WhAEM 2000 has the benefit of being accepted by the EPA and has enough sophistication to perform many general modeling needs. Because WhAEM 2000 is accepted by the EPA, if municipalities in the United States use this software to establish wellhead protection zones the approval process is much easier than if the same study were performed using another method not approved by the EPA. The disadvantages of WhAEM 2000 are: the platform dependency (written only for Microsoft Windows), the manual process of handling data, and the computational capacity being limited to the local machine.

There is a need for more web-based groundwater modeling tools. Although groundwater models exist on the internet that are either inexpensive or free, most modeling software typically is installed and executed on a local computer. With a web app, modelers can access, modify, or build a groundwater model from scratch, using any computer with internet access. An added benefit of a web app is the ability to avoid most issues related to platform or version dependencies. This means that even if a user updates their operating system or switches to a different platform altogether they will still be able to use the web app without needing an update or a patch. Web apps would increase the availability of groundwater modeling software in a way that has never been done before.

Distinctions are sometimes made between the terms "web" and "cloud" by emphasizing the more static nature of a web service as opposed to the scalable resources of a cloud service. As technology continues to push more computational resources and data to "the cloud," the definition of "the cloud" increasingly resembles the word "remote" (Holt, 2016). The terms "web app" or "cloud app" for the purposes of my research represent an application that exists only on the internet and will be used interchangeably. As to whether my cloud app may be better described as a "website" I describe my work as apps due to the fact that each app exists as one or two-page entities. In contrast, a website will typically have several pages with links between each page and distinct URL's to distinguish each page.

1.2 Analytic Element Modeling

There are advantages and disadvantages to using an AEM vs. grid-based numerical models. AEM's have the advantage of having a generally more simplified model input as opposed to numerical methods. In general, AEM's are computationally light and are extremely adept at

solving problems involving contaminant tracing. Numerical methods are more versatile than AEM's. Numerical models can model everything an AEM can model while AEM's may not always be able to model what a numerical method can.

As a general rule, models serve to simplify reality to help make educated guesses about past and future conditions. The concept that "all models are wrong, but some are useful," (Box & Draper, 1987) is inherently dependent on the purpose of the model, the assumptions made, the quality of the data, and the model's capacity to represent the outcome (Bakker, 2013). Whether an AEM or numerical model, a groundwater model is a simplified version of reality. Yet, with a groundwater model hydrogeologists and engineers can evaluate where water is coming from and where it is going with more certainty than they could without a model.

The methods applied by numerical or analytical models are inherently different in approach. Numerical models have the advantage of being able to describe more characteristics of groundwater flow than analytic element models (Rushton, 2005). On the other hand, the analytic element method has the advantage over numerical methods because a grid or element network is not required, making the analytic element method relatively insensitive to scale (Haitjema, 1995). Additionally, the simplified data input for analytic element models more closely resembles real-world scenarios as data is often expensive or unavailable to the modeler. Whether an AEM or a numerical model should be used depends on what data are available and the objective of the modeling study. Many small and simple aquifer systems may be simulated adequately with an AEM while a larger and more complex system might require a numerical model.

TimML is an analytic element groundwater model largely comprised of Python code with some FORTRAN extensions (Bakker, 2017a). TimML has the advantage over WhAEM 2000 in that TimML can simulate multiple aquifers. To model with TimML, a user can access the code

using PyPI (Python Package Index) or < https://github.com/mbakker7/timml > for either a Windows or Mac OS computer (Bakker, 2015). To simulate groundwater flow with TimML the user must use a Python scripting environment. Plotted results of TimML are generated using a supported third-party software including: matplotlib, Surfer, or MatLab.

1.3 Research Objectives

My thesis objective is to create a web app for TimML that facilitates a more broad usage of analytic element groundwater models as a decision-making tool in water resource management. As a test case or prototype for cloud-based groundwater modeling I built a simple well-based dewatering app using the well equations following Dupuit's assumptions. After successfully developing the basic dewatering tool, I built another dewatering app to mimic the first app but replacing Dupuit's well equation with the TimML source code. The culminating project combined the TimML source code with a user interface that exposed part of the modeling capacity of TimML.

2 METHODS

Hosting TimML on the "cloud" with a web interface was accomplished in three phases. The purpose of splitting up the overall development was mostly to clearly identify specific hurdles to overcome and address them incrementally rather than in one step. The intended benefits of splitting up the project were to isolate specific issues and avoid confounding the problems by mixing in more than one challenge at a time. The first phase focused primarily on the rudimentary browser GUI capabilities provided by Tethys, applying the simplest analytical equations intended to simulate construction dewatering. Phase two was designed to expand on the first phase by implementing TimML as the back-end processor instead of the basic equations. The final phase was to construct a new app applying the GUI developed over the first two phases and implementing TimML as the computational engine. The objective of each phase was to develop a new web application, each more sophisticated than the previous.

2.1 Dupuit Dewatering App

The objective of the Dewatering App is to perform basic dewatering simulations. Construction projects commonly employ aquifer dewatering simulations in preparing a site for construction. For instance, if a project requires an excavation depth lower than the existing water table then the water table elevation must be lowered in order to carry out the project. Dewatering a construction site increases the safety of the workers and helps to increase the overall stability of

the soil in the region. Wells and cutoff walls are the most commonly employed means for lowering the water table elevation in a local region. By installing wells in close proximity to the project site the water table will drop close to the wells. However, using wells alone can risk nearby structures if the water table has to be dropped any appreciable amount because of subsidence. Subsidence occurs when water is drawn out of the soil matrix and the soil consolidates. Cutoff walls are used to mitigate the effects of subsidence by isolating the drawdown to the construction site. Common cutoff wall structures are sheet piles with interlocking joints, grout curtains, and slurry trenches. The concept of the cutoff wall is that a section of the aquifer is blocked from the surrounding area and wells are installed just inside of the barrier wall boundary. The primary benefit of cutoff walls is the reduced pumping volume required to draw the water table down to satisfactory levels.

The Dupuit equation analytically simulates the impact of a well on an aquifer (Equation 2-1). The assumptions that help formulate the Dupuit equation are that the exit point of the aquifer coincides with the water level in the well, the hydraulic gradient is constant along a vertical line, the hydraulic gradient is equal to the slope of the free surface, and wells penetrate the full aquifer thickness (Figure 2-1).

$$h = \sqrt{H^2 - \frac{q \ln\left(\frac{R}{r}\right)}{\pi k}} \qquad (2\text{-}1)$$

Where:

- h = Water Table Elevation
- H = Initial Water Table Elevation (prior to pumping)
- q = Pump Rate
- R = Radius of Influence
- r = Distance from Well

- k = Hydraulic Conductivity of Soil

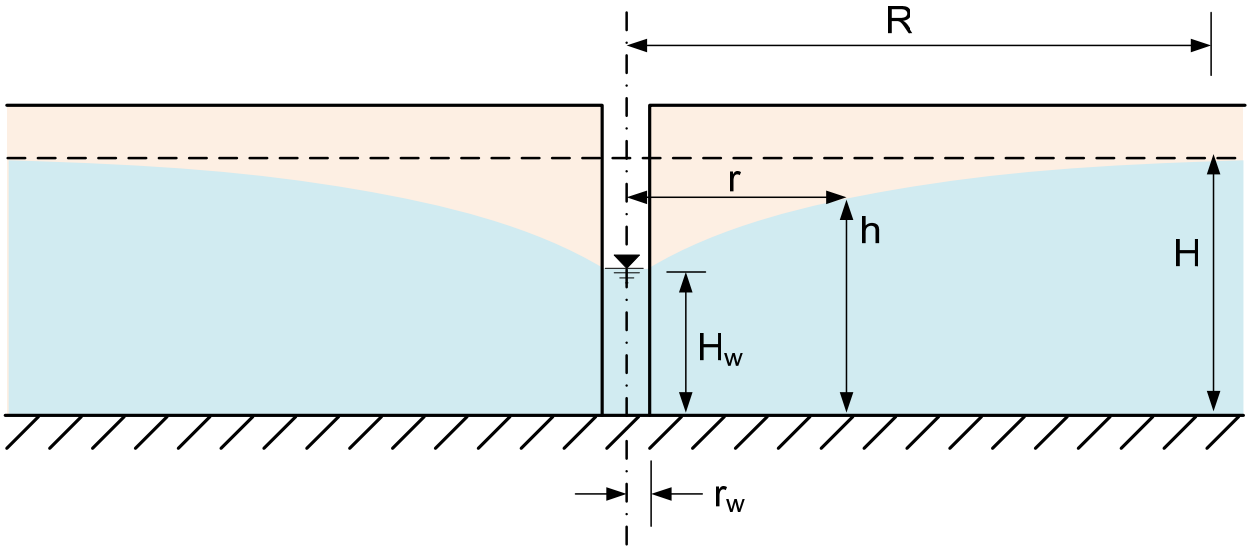


Figure 2-1: Dupuit Equation (Jones, 2017)

To simulate multiple wells using the Dupuit equation, each well solution may be summed together to produce a composite drawdown result. It is possible to sum the individual well solutions by recognizing that flow is linearly proportional to h^2 (Equation 2-2). Because flow is linearly proportional to h^2, superposition can be applied to the term $H^2 - h^2$ (Equation 2-3).

$$H^2 - h^2 = \frac{q}{\pi k} \ln R/r_i \tag{2-2}$$

$$h = \sqrt{H^2 - \frac{1}{\pi k} \sum_{i=1}^{n} q_i \ln(R/r_i)} \tag{2-3}$$

The specific use case for the Dupuit Dewatering App is to model groundwater drawdown using only wells. Because the Dupuit equation cannot simulate complex flow boundaries like a cutoff wall the Dupuit equation can only simulate aquifer drawdown caused by wells. The underlying assumptions for the app are as follows:

1. All groundwater scenarios are described by unconfined flow characteristics.
2. Water is assumed to have an infinite source.
3. Units system is U.S. Customary (feet, seconds).
4. The radius of influence is fixed at 500 feet from the area of interest.
5. Wells follow the Dupuit assumptions with no correction near the well (Equation 2-1).

The basic structure of the web application is a single HTML page consisting of an OpenLayers map (OpenLayers, 2107) with a clickable interface, drawing tools exclusively associated with the map, and a pane on the left-hand side of the page for entering all aquifer and pumping properties (Figure 2-2). To make the page interactive, the Tethys *Map View* gizmo is implemented in conjunction with the *TextInput* and *Button* gizmos provided by Tethys Platform. The input required from the user includes the average hydraulic conductivity, the bedrock elevation, the initial water table elevation, the combined pumping rate of all wells, and a desired drawdown elevation. The bedrock and initial water table elevations are used to calculate the thickness of the aquifer layer. Pumping rates for the individual wells were assigned by dividing the combined pumping rate by the number of wells. Given that Dupuit's equation only simulates the effects of a single well, each well solution was superimposed on the other and added together to provide a composite solution (Equation 2-3). The summed influence of each well simulates the drawdown effect of the wells combined (Figure 2-3).

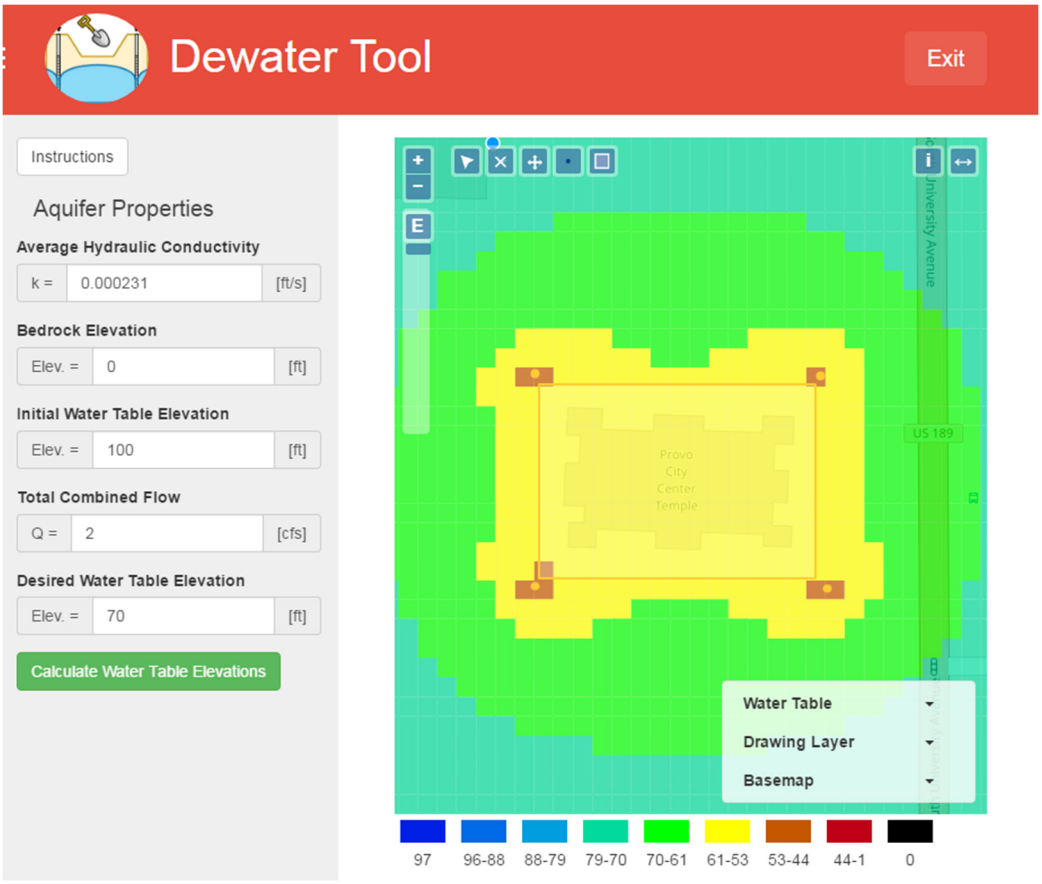


Figure 2-2: Dewatering App User Interface

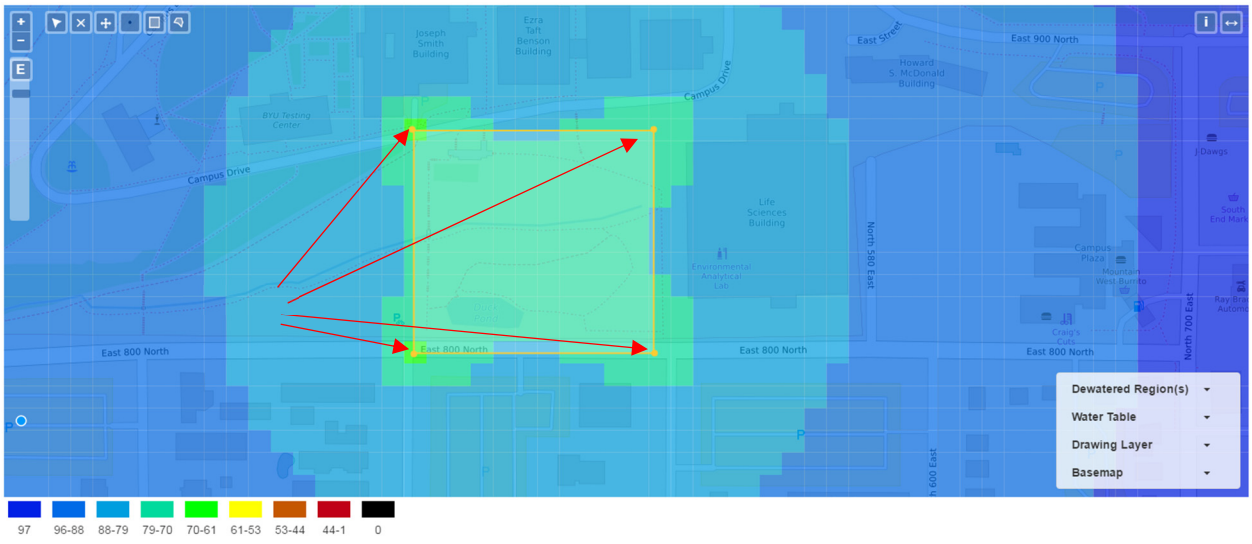


Figure 2-3: Combined Well Solution

All user input was collected by a JavaScript file called ‘main.js’ and subsequently passed to a Python file named ‘controllers.py’. The Dupuit equation was coded as a custom function in the ‘controllers.py’ file which accepts the inputs passed from the ‘map_model.js’ and returns an array of elevations. The elevation calculations are calculated on a grid that divided up the map into 100 cells, using the center of each grid cell as the point for calculating the water table elevation with respect to the datum defined by the user. After completing the grid calculations, the elevations are passed back to the ‘main.js’ file where the cells are treated as rectangles with known coordinates at the corners and plotted on the map for visual feedback. The colors shown in Figure 2-2 were selected from a color ramp based on the desired water table elevation specified by the user. A legend is provided to define which colors represent various ranges of elevations.

2.1.1 Improvements to Tethys Platform

Prior to building the dewatering app, the *Map View* gizmo in Tethys lacked a user-interface feature of allowing the user to delete drawn features. The objective of the dewatering tool is to allow users to design well patterns that achieve a desired drawdown via manual iteration. To enable users to have enough control over the program to perform iterative solutions the ability to delete OpenLayers map objects had to be added to the Map View gizmo. I added a delete tool to the existing class called MVDraw as an additional ‘draw’ control. The delete control was created as a modified version of the ‘edit’ tool already implemented with the addition of a function listener that would delete every object at the location of a click. To prevent the delete tool from deleting features in the solution layers I added an additional screening to only delete drawing objects created by the user.

2.2 TimML Dewatering App

One major limitation of the Dupuit Dewatering App is the inability to simulate flow barriers. In real dewatering scenarios a designer will often employ the use of a flow barrier in addition to wells. The TimML Dewatering App will support both cutoff walls and wells in dewatering simulations by using TimML as its back-end. To develop a version of the dewatering app with TimML as the computational engine, the first hurdle to overcome was compiling and wrapping TimML's FORTRAN libraries into Python "shared-object" libraries. Shared object libraries dynamically link FORTRAN routines with Python scripts used at run time without requiring the FORTRAN files to be recompiled for any successive uses (Meyer, 1988). Current implementations of TimML only work on Windows and Mac OS systems (Bakker, 2015). Given that Tethys currently operates primarily in Linux machines, the FORTRAN files needed to be compiled using a Linux compiler. The coding library F2PY was chosen for wrapping the FORTRAN file into a shared object library accessible to Python readable by Linux machines. Once compiled and wrapped, the FORTRAN shared object files were saved in the same directory as the Python scripts comprising the TimML module.

After successfully extending TimML with the FORTRAN extensions and installing the software dependencies (to include matplotlib, scipy, and numpy) TimML needed to be made accessible to Tethys. The computational advantage of TimML is the fact that TimML does not require input files to run and can be directly accessed as a FORTRAN library via a Python wrapper. Because Tethys is Python-based TimML can be directly integrated with Tethys applications. The ability to directly integrate Tethys web applications with TimML reduces the programming and server overhead by not having to write and receive the input and output files typical with many groundwater modeling software packages. The Python libraries stored by Tethys are kept in a

directory called ‘site-packages’ on the Tethys server. By copying TimML into the ‘site-packages’ directory on the Tethys server I made TimML a cloud-based *library* accessible to applications hosted by the same server.

2.2.1 Modeling with TimML

TimML is an AEM comprised of geometric features called elements. The overarching aquifer properties like the layer elevations, the number of layers in an aquifer, the associated porosities, hydraulic conductivities, etc. are assigned in a ‘model’ object. Each model object in TimML can then have any array of associated elements defined. Available in TimML are elements to represent rivers, ditches, wells, aquifer inhomogeneities, recharge, and sinks. Boundary conditions are specified at a single point known as the constant element, which is analogous with a groundwater observation well. The constant element specifies the aquifer head at a specific point for an assigned layer in the aquifer. Well elements are also specified with points and are assigned a well radius, pumping rate, and contributing layers. TimML differs from the Dupuit equation in that the wells’ radii of influence are calculated based on their relative position with the constant element rather than a fixed value. With regard to the TimML Dewatering App, TimML simulates a leaky flow barrier by using two nested aquifer inhomogeneity elements. The inner inhomogeneity retains the original aquifer characteristics while the outer inhomogeneity has the hydraulic properties of the flow barrier being modeled.

2.2.2 GUI for TimML Dewatering App

I created input fields for the TimML Dewatering App, comprised of the same input fields used by the Dupuit Dewatering App including the average hydraulic conductivity, the bedrock and initial water table elevations, the combined pumping rate, and the desired water table elevation.

The major input difference is the ability to toggle on the use of a slurry trench. I added two more input fields to assign the hydraulic conductivity of the slurry and define the thickness of the barrier wall (Figure 2-4). The fundamental assumptions for the TimML Dewatering App are the following:

1. All dewatering scenarios are for single layer systems.
2. The constant element is located at exactly $\sqrt{500{,}000} \cong 707.1$ meters from the first well placed in the model at an angle of 45° from North. Meters are used in defining the constant element coordinates because the map projection is in meters.
3. Water is considered to have an infinite source.

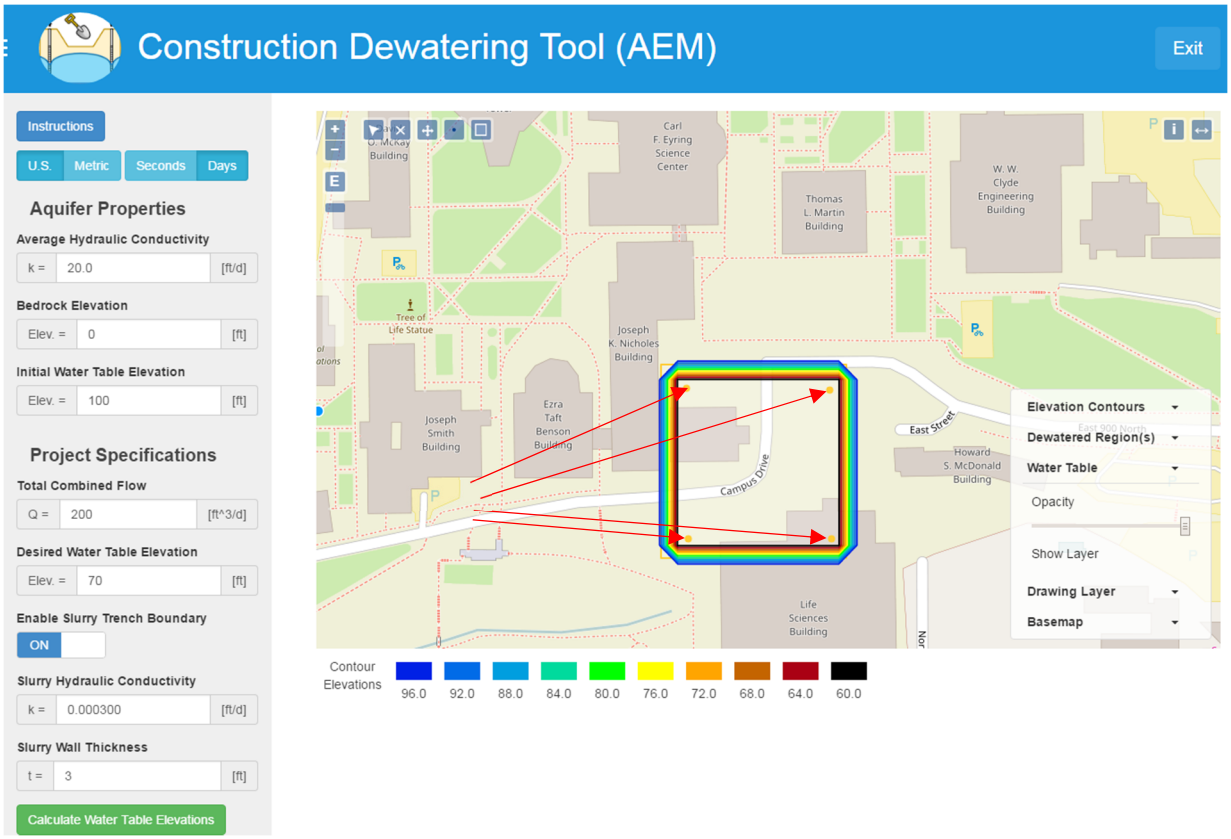


Figure 2-4: Combined Well Solution

The software structure used by the TimML Dewatering App as is similar to that used by the Dupuit Dewatering App. The 'main.js' JavaScript file collects all of the user input fields as variables and passes the information to the 'controller.py' file. Instead of taking the user inputs and assigning values to variables of an equation as with the Dupuit Dewatering App, the TimML Dewatering App assigns characteristics to the respective elements. The bedrock elevation, initial water table elevation, and hydraulic conductivity initialize the model object with layer elevations and the conductivity. The initial water table elevation also initializes the constant element with an observed head value. Each well element is assigned an equal fraction of the combined pumping rate specified by the user. If the slurry trench is enabled in the simulation then the hydraulic conductivity specified by the user is assigned to the inhomogeneity representing the leaky flow barrier.

The "desired drawdown" input value specified by the user was used to generate the colored representation of water table elevations as done with the Dupuit Dewatering App. A new layer indicating areas of acceptable drawdown based on the desired drawdown parameter was added to the TimML Dewatering App. If an area has a water table elevation lower than the specified elevation then the cell is colored with green. If elevations are higher than the desired water table elevation then the cells are colored with red (Figure 2-5).

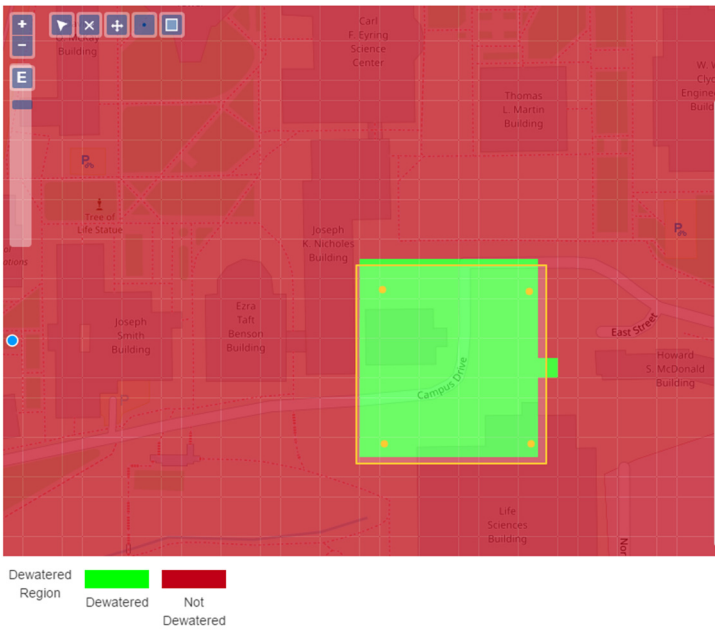


Figure 2-5: Dewatered Regions

2.2.3 Improvements to Tethys Platform

During the development of the first two dewatering apps, I discovered that it was not possible to restrict layers from being edited in the Map View gizmo. In other words, solution layers were exposed to the user by the editing tools, allowing the user to both modify and delete the geometries of solution layers. To allow Tethys to distinguish between editable and non-editable layers, I added a parameter to the MVLayer object in the Map View gizmo. The new parameter consists of adding a Boolean attribute to every layer generated by Tethys called ‘editable.’ If true, the layer is readable by the ‘Modify,’ ‘Delete,’ and ‘Move’ tools. If the editable property is set to false, then the Map View Gizmo will only display the geometries of the layer, regardless of the user control being applied.

2.3 Groundwater Modeling App

The objective of the final rendition of cloud-based groundwater modeling was an app allowing users to develop complete TimML groundwater models. The dewatering apps previously discussed are special cases of groundwater models. Other groundwater modeling needs may include the ability to delineate well capture zones, perform well impact studies or perform preliminary contaminant cleanup investigations. I built a general purpose groundwater modeling app call TimML-Cloud to address the more general groundwater modeling needs. Two significant features were necessary to expand the TimML Dewatering App code base to create a new fully-featured TimML modeling application: 1) a new Tethys feature called a layout that allows users to create, edit, and save spatial and non-spatial data associated with a typical TimML simulation, and 2) a data management system that allows users to create, save, modify, and load custom groundwater models.

2.3.1 Tethys Map Layout

The problem with Tethys prior to developing the *Layout* object was the inability to manage spatial data. The *Map View* gizmo has the built-in ability to annotate a map with features including points, lines, and polygons, but lacks the ability to assign attributes to individual features. *Map View* also lacks the ability save or load features. The Tethys *Layout* addresses the problem of managing spatial datasets by combining three Tethys gizmos as a single entity. In the case of the TimML-Cloud App, a layout was conceived called the "Map Layout". The *Map Layout* consists of the original *Map View* gizmo, an *Attribute Table* gizmo, and a new Tethys gizmo called the *Table of Contents*. The *Map View*, *Attribute Table*, and *Table of Contents* gizmos work together to manage spatial data. The *Map View* manages all feature data, the *Attribute Table* manages all

attribute data associated with all features, and the *Table of Contents* provides user controls over the *Map View* and *Attribute Table* gizmos.

From the users' perspective, the *Map Layout* is a single web page with a map, a table of attributes, and a layer management pane (Figure 2-6). To create and edit features, the user would click on the options button of an associated layer and select the 'edit features' option. When the 'edit features' option is clicked, user controls for that specific layer are turned on in both the *Map View* and *Attribute Table* gizmos. The user controls on the *Map View* allow the user to create, modify, or delete features only for the selected layer. The *Attribute Table* reflects the information for each feature on the layer being edited. The 'save' button preserves all changes made to the layer while the 'cancel' button reverts all changes back to the original state of the layer. When either the 'save' or 'cancel' button is clicked the editing controls are turned off, returning the user to a viewing state.

The *Map View* gizmo operates without modification in the *Map Layout*. The *Attribute Table* gizmo is dynamically linked with the *Map View* gizmo, auto-generating a row to represent each feature and its respective attributes. The *Table of Contents* gizmo manages layer edits by creating methods that define modes for viewing, editing, and saving spatial data. When in the editing mode, the *Attribute Table* and *Map View* gizmos are provided with user inputs and controls for editing a selected layer. When saving edits, the *Table of Contents* gizmo loops through all information on both the *Map View* and *Attribute Table* and stores a copy of the information. While in the viewing mode, the *Table of Contents* gizmo restricts the user from editing either the *Map View* or *Attribute Table* gizmos.

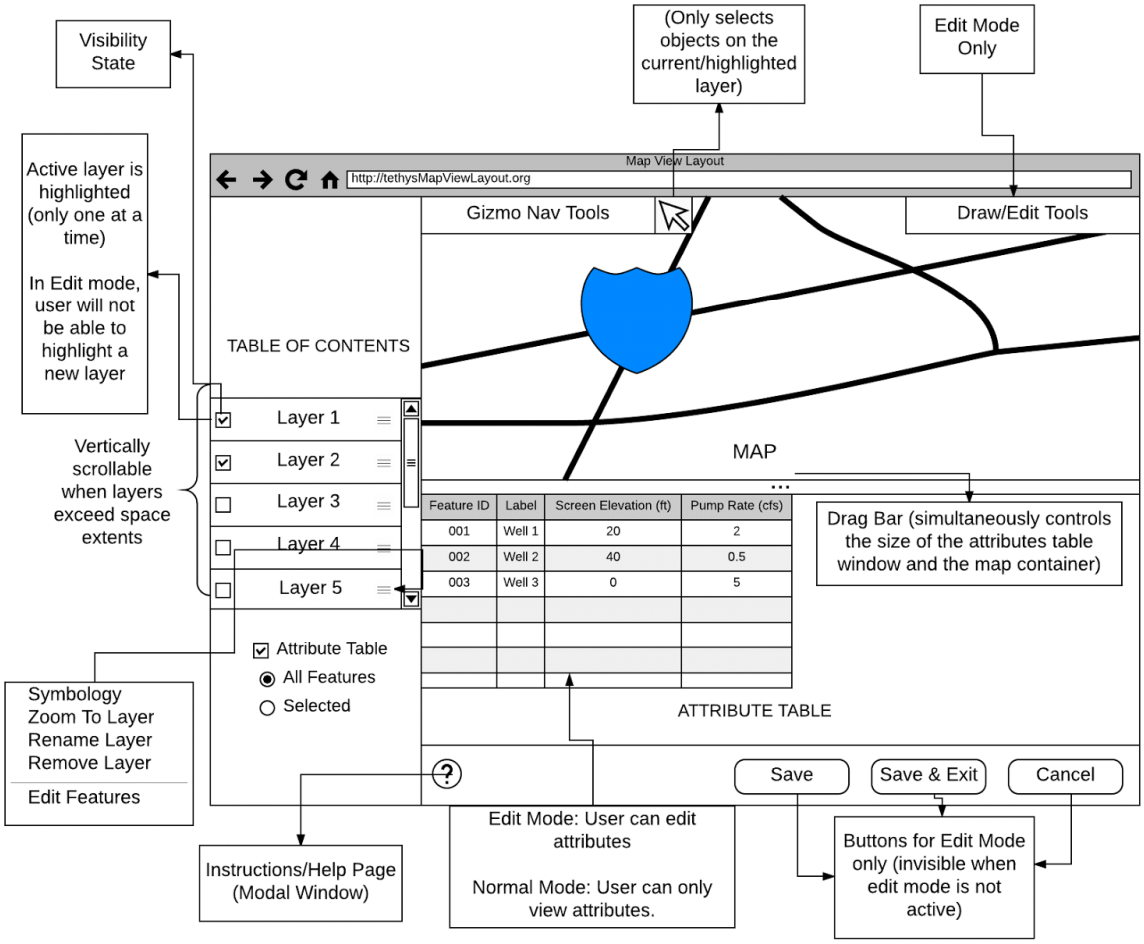


Figure 2-6: Map Layout Concept Design

2.3.2 Table of Contents Gizmo

The *Table of Contents* gizmo was inspired by Crawley (2017) with his work on the Hydroshare GIS App and Crawley's code base was leveraged as a starting point for this application. The abilities to control map layer visibility, map layer ordering, layer renaming, and the context menu were derived from Crawley's widget. The first step to transfer Crawley's *Table of Contents* widget into Tethys as a gizmo was to standardize the behavior of the widget for Tethys applications. The difference between a widget and a gizmo in the context of Tethys is where the entity derives

its functionality. A Tethys gizmo depends on the libraries and classes provided by Tethys while a widget is self-contained and can operate independently of Tethys. The *Table of Contents* needed to be transferred into Tethys as a gizmo for it to work properly in the *Map Layout*. The major consideration for standardizing the *Table of Contents* code was the type of map with which it would interface. Given the common implementation of the OpenLayers maps in Tethys apps, the *Table of Contents* is designed to interface specifically with the OpenLayers map object. Knowing that OpenLayers would be the primary map, I standardized all functions in the context menu to use OpenLayers API methods as opposed to the queries used in Crawley's application. The standard functions I implemented in the gizmo include: isolate layer, edit features, save features, cancel edits, zoom to layer, delete, and rename layer (Figure 2-7). The "isolate layer" method hides all but the selected layer from view while the "zoom to" method scrolls the map to view the extents of the selected layer. The "delete" method allows the user to delete the selected layer from the application. The "rename layer" method handles the user-visible names of each layer in *Map View* gizmo. The "edit features" method enables the user inputs and controls associated with the *Map View* and *Attribute Table* gizmos. To save edits, the "save features" method loops through both the *Map View* and *Attribute Table* features and rows, respectively, to create a saved copy of the information in the temporary session memory of the internet browser. If edits are canceled, the "cancel edits" method retrieves the latest copy of the *Map View* and *Attribute Table* data and restores the app to its original state.

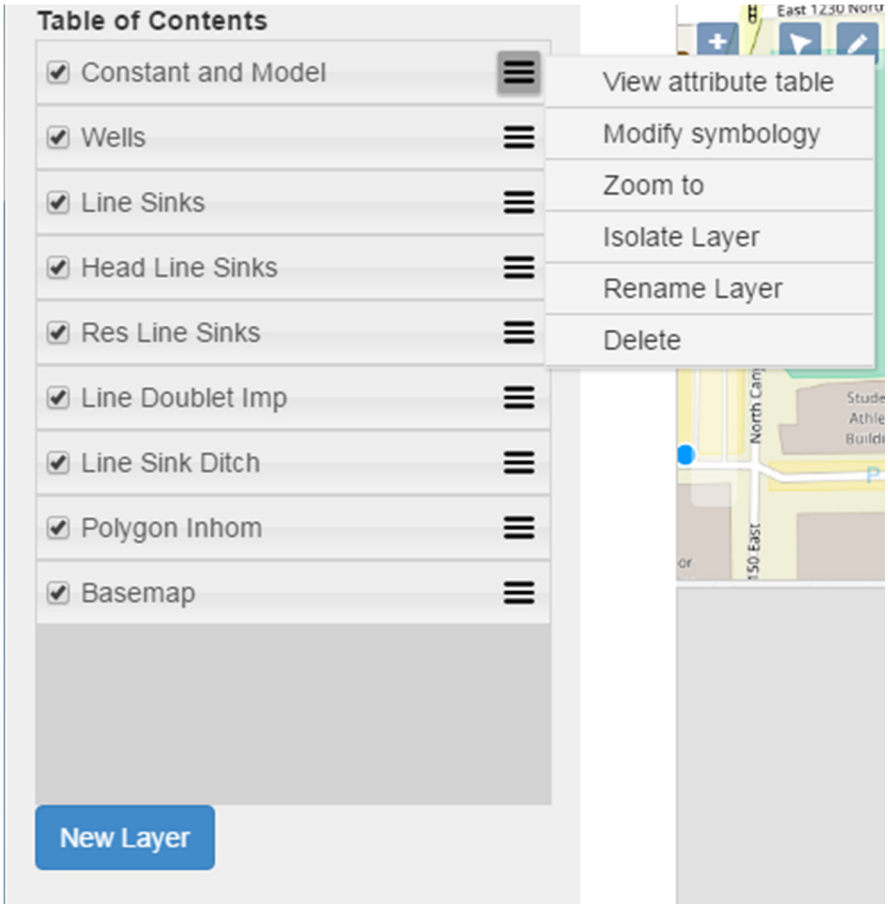


Figure 2-7: Table of Contents Gizmo

The *Table of Contents* manages user edits by creating a data management workflow. The workflow in the *Table of Contents* gizmo consists of the exclusive modes for either viewing or editing data. With the view mode enabled, users do not have access to the drawing or editing tools. With the edit mode turned on, the user has access to the drawing and editing tools pertinent to the layer, but is restricted from viewing information in other layers. Editing controls are also layer specific. If the edit mode is turned on for a polygon layer for instance, the tools specific to polygons are enabled, and the same pattern follows for point layers and line layers. To preserve edits, I made it so that every time the edit mode is entered, all data is pushed to the sessionStorage object

belonging to the internet browser. I also preserved edits in the sessionStorage object whenever the 'save features' method is used.

2.3.3 Attribute Table

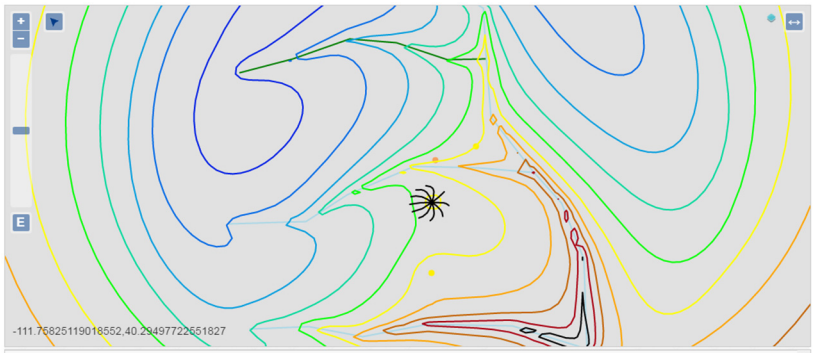
Though the table object was not new to Tethys, development of an *Attribute Table* required special controls to interface with the OpenLayers map object. To tie the attribute table with the map object, a listener was added to each drawing tool so that every time a new feature was added to the layer being edited, an ID would be assigned to the feature and a corresponding row added to the attribute table. The attribute fields are generated from a predefined list provided in the layer defined by the programmer (Figure 2-8). Upon saving the layer using the method defined in the *Table of Contents* gizmo, the features of the layer being edited and the corresponding attribute table rows are pushed through a series of 'for-loops' which set the attributes of each feature equal to the value of the input field within the table row. To uniquely identify each row with its respective feature, a common ID term was assigned to both the feature and the corresponding row.

total_time	layer	end_element	elem_ID	elem_label	end_state
-529026.2958599736	2		well_0	Well_1	flowed out of top of aquifer system
-767393.8601027328	2		well_0	Well_1	flowed out of top of aquifer system
-782267.3910071163	2		well_0	Well_1	flowed out of top of aquifer system
-1308095.4859327876	2		well_0	Well_1	flowed out of top of aquifer system
-787155.3651576274	2		well_0	Well_1	flowed out of top of aquifer system
-671042.5005942744	2		well_0	Well_1	flowed out of top of aquifer system
-804380.6676052096	2		well_0	Well_1	flowed out of top of aquifer system
-583491.2120223094	2		well_0	Well_1	flowed out of top of aquifer system
-540670.4028266761	2		well_0	Well_1	flowed out of top of aquifer system
-574045.622494873	2		well_0	Well_1	flowed out of top of aquifer system

Figure 2-8: Attribute Table Gizmo

2.3.4 Golden Layout

The next step in developing the *Map Layout* was to standardize the view state of the layout. I selected Golden Layouts as the library to organize the *Map View* and *Attribute Table* gizmos for its "plug-and-play" style of code. Golden Layouts (deepstream.io, 2017) is an open source library whose function is to organize webpage content as windows and tabs inside of containers. For the *Map Layout*, the *Map View* and *Attribute Table* were organized into a single window with a split screen. A drag bar splits the window holding the *Map View* and *Attribute Table* gizmos, allowing the user to resize each half of the screen (Figure 2-9). By organizing the attribute table and Map View gizmo with the Golden Layouts container, the essential components of the map and table are accessible with convenient user controls to facilitate a standardized modeling interface. With the standardized modeling interface complete, the groundwater modeling app could then be built on top of Map Layout.



total_time	layer	end_element	elem_ID	elem_label	end_state
-56408.61439344187	0		well_0	Well_1	flowed out of top of aquifer system
-73118.08526338454	0		well_0	Well_1	flowed out of top of aquifer system
-93284.2317670327	0		well_0	Well_1	flowed out of top of aquifer system
-88559.15939447292	0		well_0	Well_1	flowed out of top of aquifer system
-61329.95565823788	0		well_0	Well_1	flowed out of top of aquifer system
-70099.6440307935	0		well_0	Well_1	flowed out of top of aquifer system
-65347.723275019285	0		well_0	Well_1	flowed out of top of aquifer system
-59490.4295010295	0		well_0	Well_1	flowed out of top of aquifer system
-56011.961150741205	0		well_0	Well_1	flowed out of top of aquifer system
-75547.38260543415	0		well_0	Well_1	flowed out of top of aquifer system

Figure 2-9: Golden Layouts Implementation

2.4 Management of Model Instances

Modelers rarely ever build a groundwater model for a single use. Well impact studies for instance will recycle the same groundwater model multiple times as different well locations are tested over an iterative process. A saving mechanism needed to be implemented in TimML-Cloud to service the needs of modeling technicians who will likely want to reuse models constructed on the web application. The fact that TimML is a set of Python functions that run live on the server without any input files necessitates the creation of a custom file that can be translated into a readable format for both the TimML library and the web app interfacing with TimML on the server.

Because TimML-Cloud directly processes features and their associated attributes, the model can be entirely preserved by saving a text file describing the feature objects and attributes. Saving a model instance starts a routine that first collects all of the features and attributes into a single JavaScript object and then passes the object as text to a Python controller function. Feature geometry and data attributes are stored in the JavaScript object as GeoJSON objects. A GeoJSON object is a standardized protocol for preserving spatial data which organizes a feature data by type, geometry, coordinates, and feature properties (Butler et al., 2016). The Python function then writes all of the text into a file and assigns a unique file ID and appends the model name to the end of the ID after an underscore character. The unique file ID allows the user to generate multiple instances of models with the same name without overwriting a previously saved file.

User models are stored in the Tethys *User Workspace* which is a folder structure that exists within the app directory. *User Workspaces* allow users to view and edit files created with a specific account which is directly associated with the folder storing the files.

I provided controls to create copies of existing models and to delete a selected model instance. The ‘save’ and ‘save-as’ controls can be accessed from within the *Map Layout* interface (Figure 2-10). The ‘duplicate’ and ‘delete’ methods for the model instances were provided on the landing page of TimML-Cloud (Figure 2-11). Using the ‘save’ method overwrites the existing file that is being edited while the ‘save-as’ method allows the user to create a new file with a new model instance name. The ‘delete’ method passes the name of the selected file to the Python controller which then deletes the file with the matching name. The ‘duplicate’ method passes the name of the selected file to the Python controller which duplicates the existing model instance file with a new unique ID while preserving the name. The landing page list is refreshed to show the latest changes after either deleting or duplicating a model instance. The save date is displayed in a column next to the file name on the same row to help distinguish models from one another.

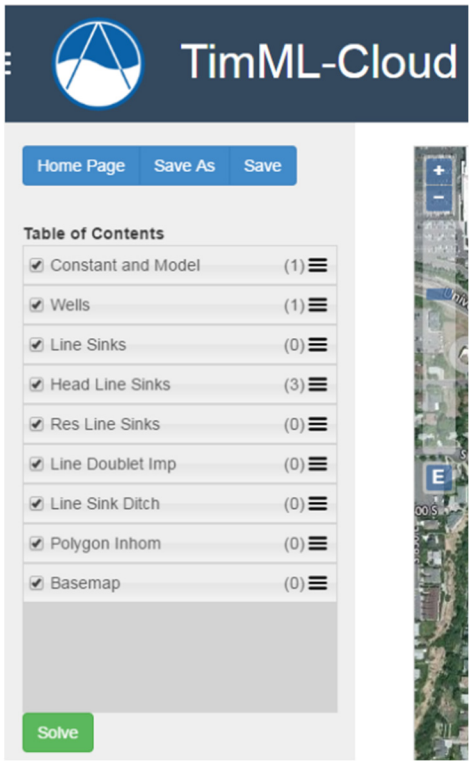


Figure 2-10: Model Management Controls

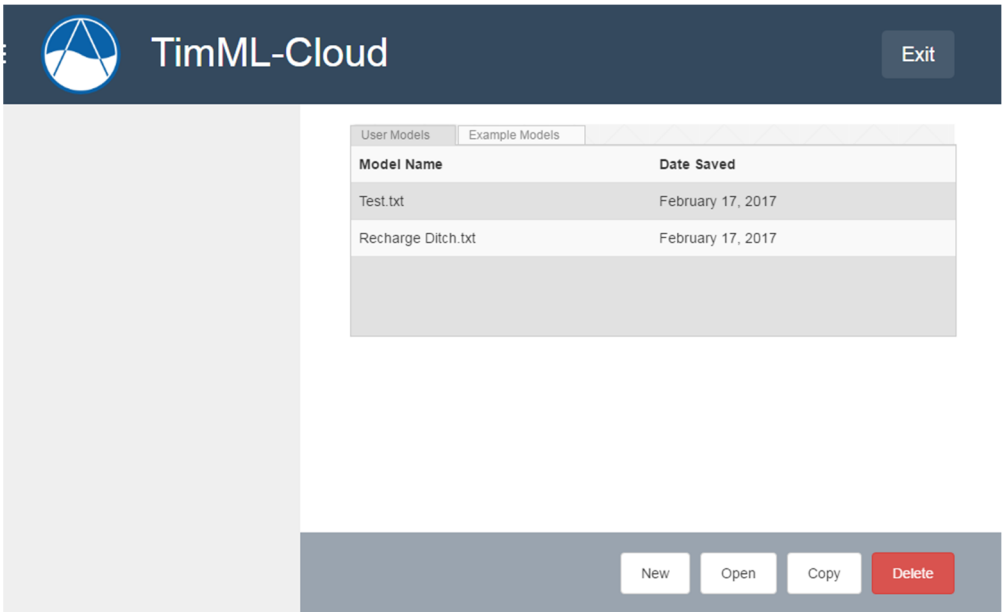


Figure 2-11: Model Instance Management

2.4.1 Particle Tracking

For TimML-Cloud to serve as a general purpose groundwater modeling service a user control for particle tracking had to be created. Particle tracking is especially important for delineating well capture zones for wells. From well capture zones a modeler can indicate wellhead protection zones and evaluate what sources contribute to a given well. TimML provides its own method for particle tracking, giving methods for both plotting and attribute feedback. The attribute feedback generated by TimML includes the aquifer layer of a particle at any given time, the total time traveled by the particle, and the reason for termination (e.g. if the particle encountered a well or exited the window boundary). A challenge associated with implementing the default TimML particle tracking in TimML-Cloud is the over-generalized nature of the particle tracking methods. Providing full-featured particle tracking would require much more effort and time than is feasible with this first implementation of TimML-Cloud. To simplify the effort, I selected well capture zones as the focus of the particle tracking utility in the app.

A well capture zone is the area of influence associated with a well. The concept of the well capture zone is that a tracer or contaminant introduced to the aquifer within the boundaries of the capture zone would be "captured" by the well. Although TimML already provided a method for calculating well capture zones, the method lacked the capacity to plot the results independently of matplotlib, Surfer, or MatLab. To enable plotting I built custom methods by modifying Dr. Bakker's existing capture zone tracing methods to retrieve feature coordinates for trace lines. Another modification that I added to the custom capture zone methods was the ability to retrieve the attributes for each trace line generated by the well capture zone.

After programming the modified capture zone method, the only remaining step was to create GeoJSON objects using the geometry coordinates and assign elevations to each tracer. Once the GeoJSON objects are created, they are passed back to the 'map_model.js' and are colored based on their respective layer in the aquifer. At this point, the TimML-Cloud app is capable of performing custom models and delineating well capture zones.

3 RESULTS

I applied use cases to each web app to verify that the apps were functioning properly. For the TimML apps, the use cases were repeated using TimML software in its native scripting environment for comparison. The objective of repeating the simulations in both environments is to make sure that no unexpected behaviors have been introduced by the web apps. Four scenarios were selected: four equally spaced wells, a construction dewatering scenario with a slurry trench and four wells, and two example problems provided in the TimML documentation.

3.1 Four Equally Spaced Wells

I used the equally spaced well scenario to test the Dupuit Dewatering App. I compared the Dupuit Dewatering App directly with a manual solution of the Dupuit equation using superposition. For this scenario, four wells are spaced in a square formation, each well separated from the nearest two wells by approximately 50 meters (Figure 3-1). The aquifer properties consist of a single, saturated soil layer that is 100 feet thick with a hydraulic conductivity of 20 feet per day. The pumping rate for each well is 0.25 cfs, summing to a combined 1 cfs. The radius of influence is 500 feet from each well. For comparison, I checked heads at 7 different cells in strategic locations: each cell containing a well, one cell in the center of the square formed by the wells, and one to the left and right of the wells at about 80 feet (Figure 3-2).

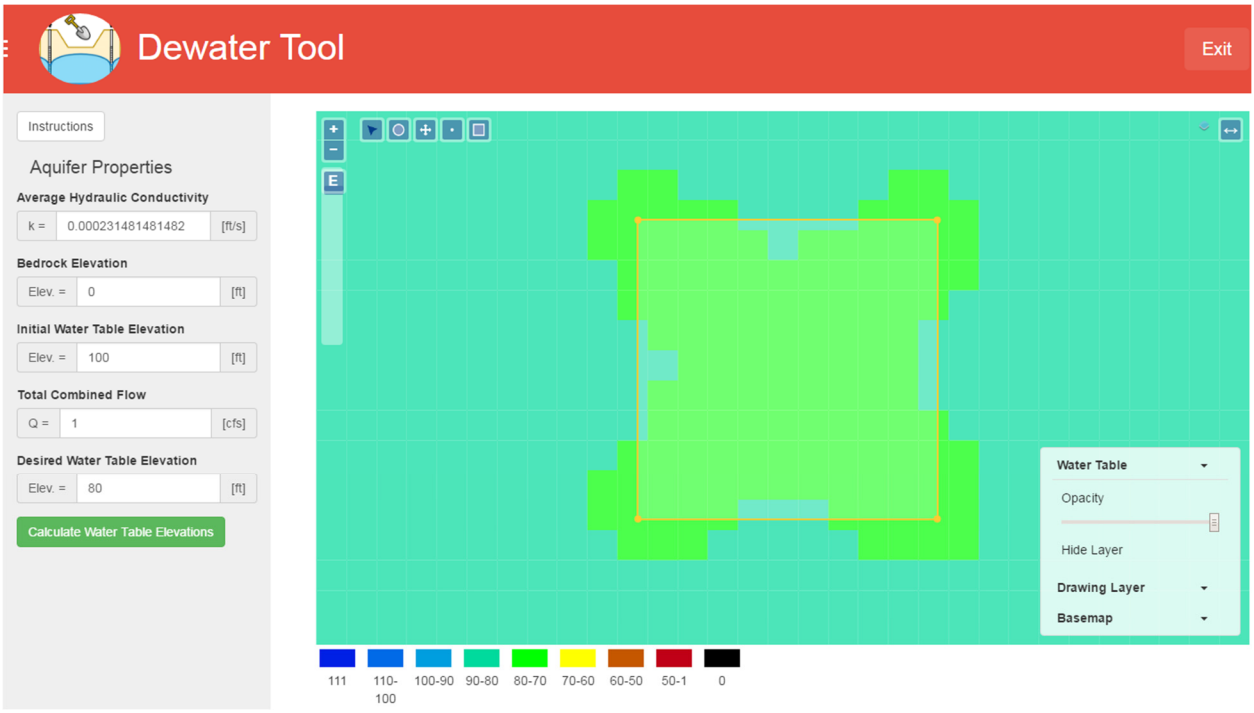


Figure 3-1: Scenario Setup

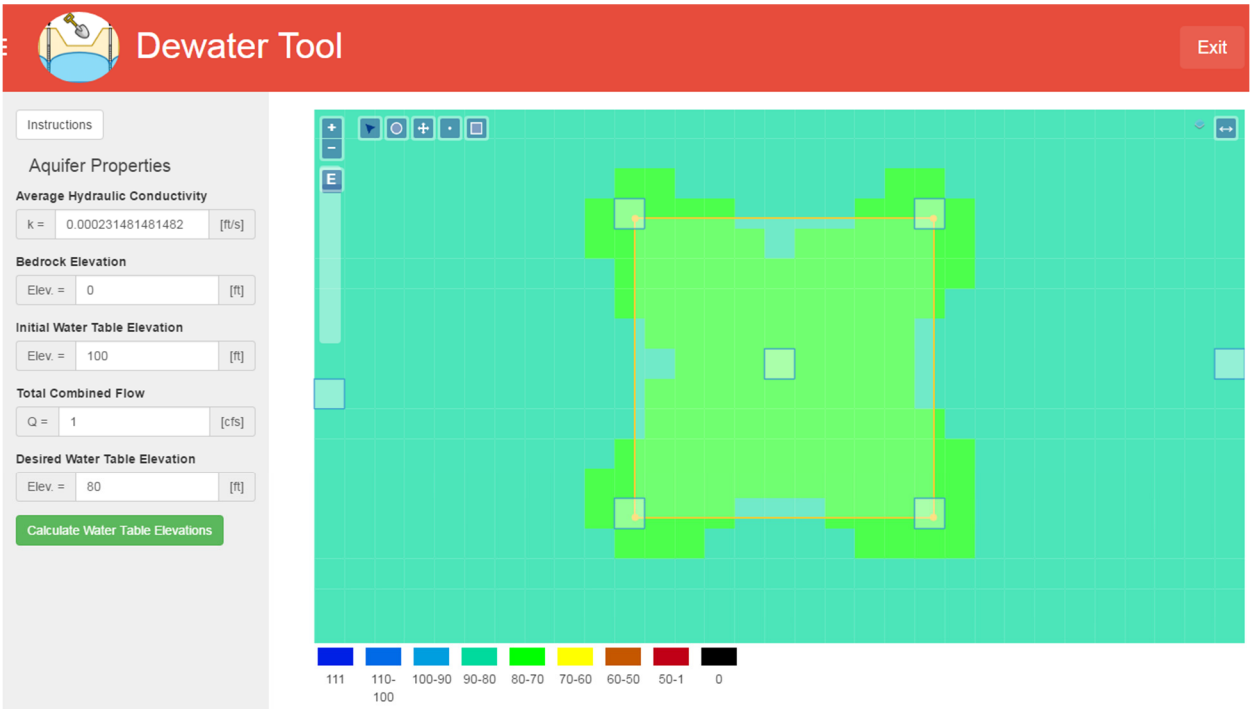


Figure 3-2: Comparison Cells

After solving the model using both the Dupuit Dewatering App and the Dupuit equation I compared the heads at the 7 cells. The heads at the 7 cells showed no variance between the Dupuit Dewatering App and the Dupuit equation.

3.2 Construction Dewatering w/ Slurry Trench

I used the use case of a single slurry wall and 4 wells to validate the TimML Dewatering App. The aquifer properties for this case study are as follows: hydraulic conductivity of 20 ft/day, a bedrock elevation of 0 feet, and the initial water table elevation is initialized at 87 feet. The slurry trench hydraulic conductivity is 0.0009 ft/day and the trench width is 9.84 feet. The slurry trench wall forms a rectangle around the wells, the x-direction having a length of 240 feet and the y-direction having a length of 210 feet. The wells are set inside the corners of the slurry trench so as to be approximately 35 feet from either wall (Figure 3-1). The pumping rate for each well is 90 cubic feet per day.

Applying the case study scenario to both the web application and TimML gave almost identical results. Water table elevations were checked within the slurry trench wall on each program and just outside of the slurry trench. The overall error differences was 4% and 3% (Table 3-2) for the interior and exterior head measurements, respectively. The differences in values are attributed to the precision in reading values off of the Dewatering App. If queries were constructed to extract exact values at the same locations in both the web application and TimML there would likely be no discrepancy between the readings.

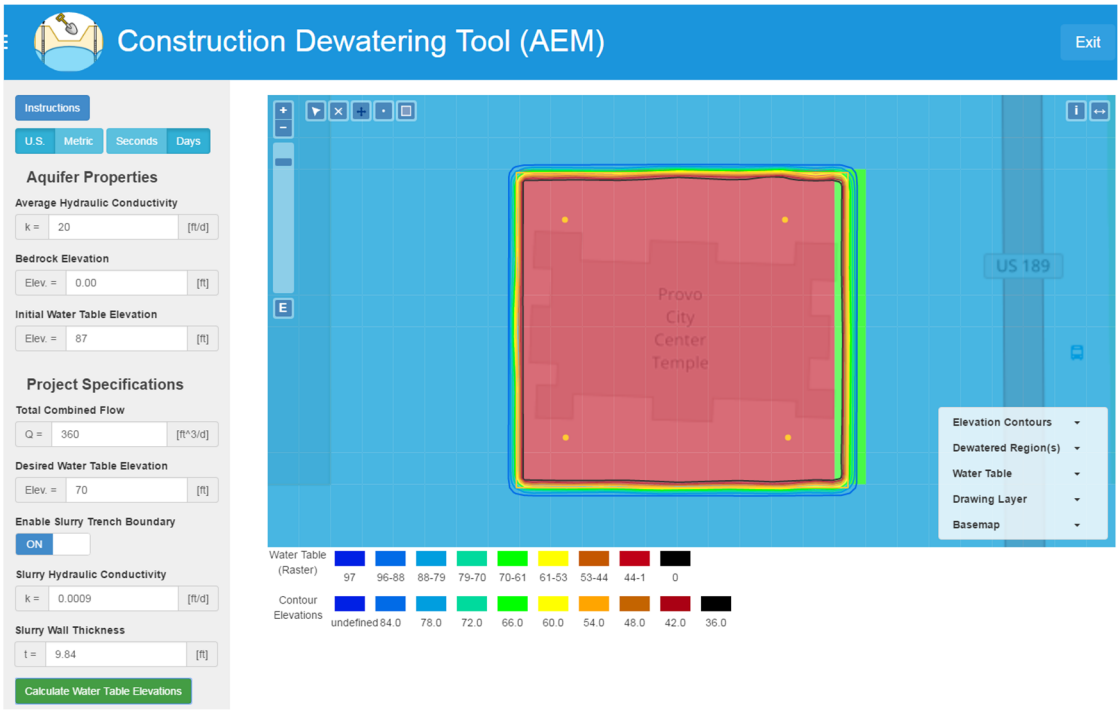


Figure 3-3: Dewatering App (AEM)

Table 3-1: TimML Dewatering App vs TimML

	Inside Barrier	Outside Barrier
Dewatering App	36	84
Native TimML	34.7	86.96
Difference (ft)	1.3	2.96
Percent Error (%)	4%	3%

3.3 TimML-Cloud Use Cases

I used two example cases provided with TimML to validate the TimML-Cloud App's performance (Bakker, 2017a). The first example was used to evaluate the performance of TimML-Cloud using a single layer aquifer. The second example verifies TimML-Cloud's simulation of

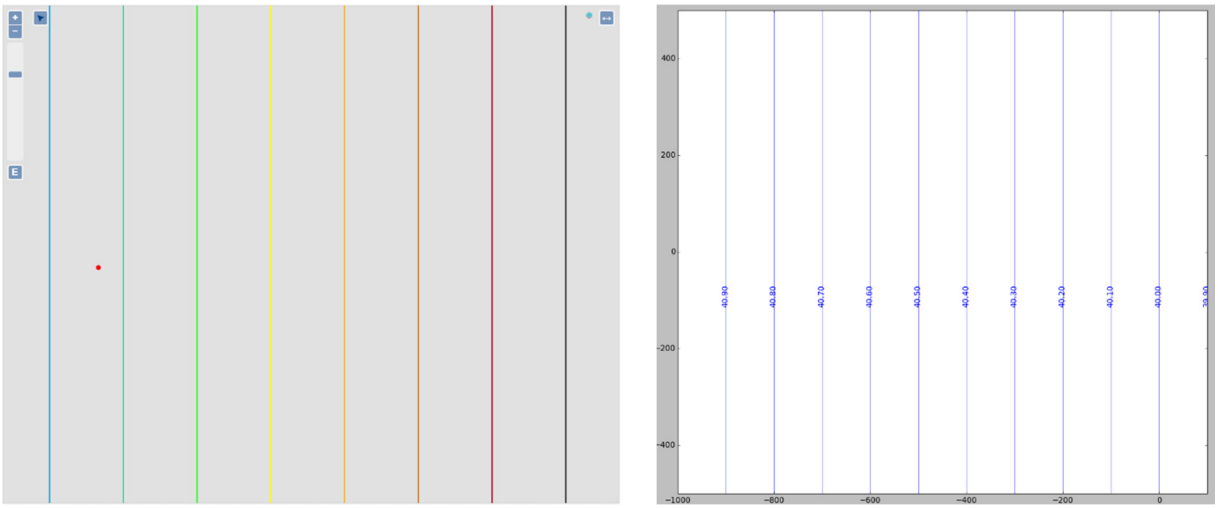
multi-aquifer systems. I compared the web application solutions directly with the solutions generated by TimML in the scripting environment.

The first case study called "Single Layer Flow" focuses on an aquifer with only one layer in ambient flow condition. The example adds elements to the aquifer first with a well, then with a river that is discretized into 20 sections. Particle tracking is also evaluated in the single layer case with tracers spaced evenly from North to South upstream of the well tracing forward in time. Another particle tracking exercise is performed at the well tracing backward in time. The second case study is called "A System with Wells, Rivers, and Recharge." In this second example problem TimML is used to evaluate a multi-layer aquifer system comprising three layers. Three wells are evaluated where two of the wells are screened in either the first or third layer while the third well is screened in both the second and third layers. Additionally, the multi-layer aquifer example includes a regional recharge element and a system of rivers.

The particle tracking and river discretization exercises were modified in Bakker's original example problems. Given the modification to TimML-Cloud which restricts all particle tracking abilities to just well capture zone delineation, the particle tracking exercises in the example studies have been modified to delineate capture zones using ten particles with a large maximum time value. Drawing many small increments of a river via the GUI is at best cumbersome if not unrealistic, so the river was approximated with a single feature instead of twenty as performed in the first example. Additionally, I modified all plots from the example displays to only plot contours for the topmost layer because TimML-Cloud only supports contouring for the topmost aquifer.

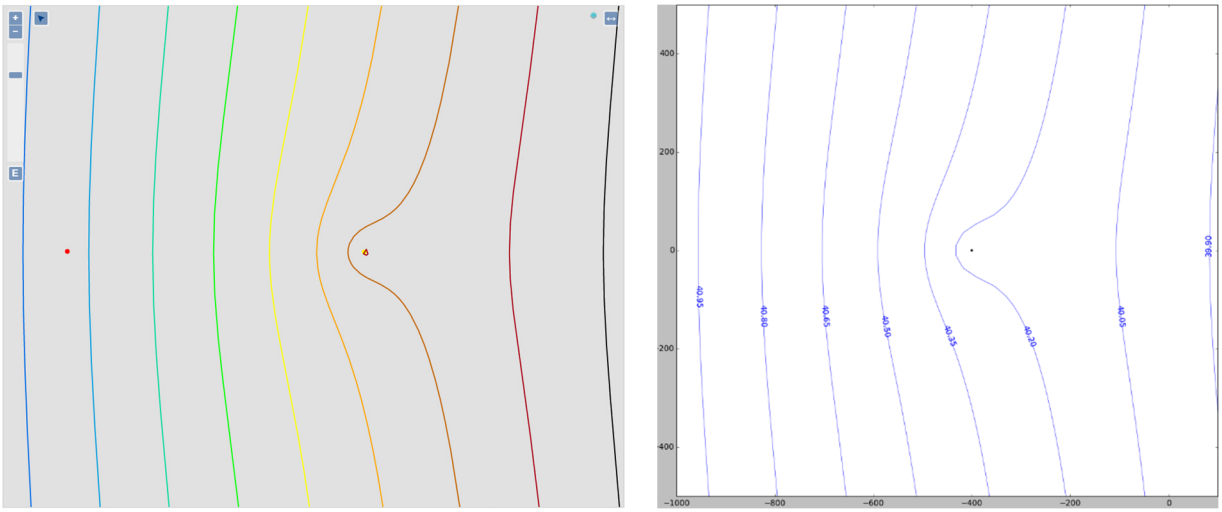
3.3.1 Single Layer Flow

I used the 'Single Layer Flow' example for comparing TimML-Cloud and TimML (Bakker, 2017b). After completing the single layer flow example in TimML and in TimML-Cloud, I compared the two solutions graphically to check for anomalies. Specific areas of concern were contour line spacing and the particle tracking paths. The first comparison was the ambient flow simulated by both applications showed no variations between the TimML-Cloud and TimML (Figure 3-3). Adding a single well to the ambient flow also showed no differences between the two applications (Figure 3-4). No appreciable differences were noted between the models after adding the single element for the river (Figure 3-5). The particle tracking examples also yielded identical solutions between TimML-Cloud and TimML (Figure 3-6 and Figure 3-7).



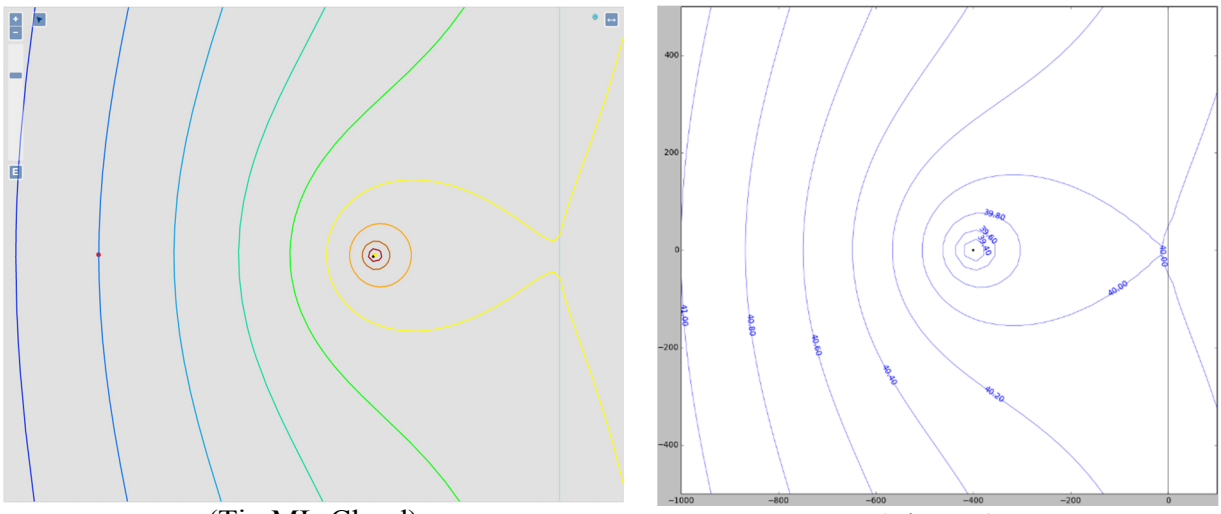
(TimML-Cloud) (TimML)

Figure 3-4: Ambient Flow Comparison



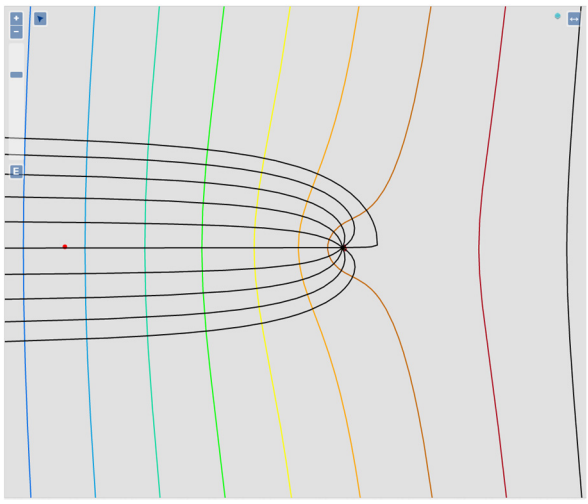
(TimML-Cloud) (TimML)

Figure 3-5: Single Well Comparison

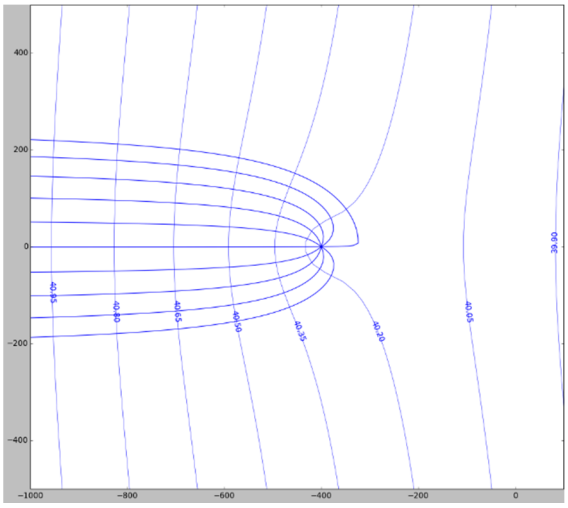


(TimML-Cloud) (TimML)

Figure 3-6: Well and River Comparison

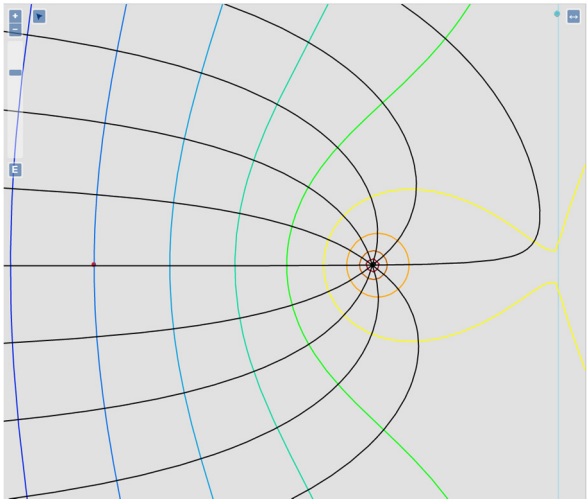


(TimML-Cloud)

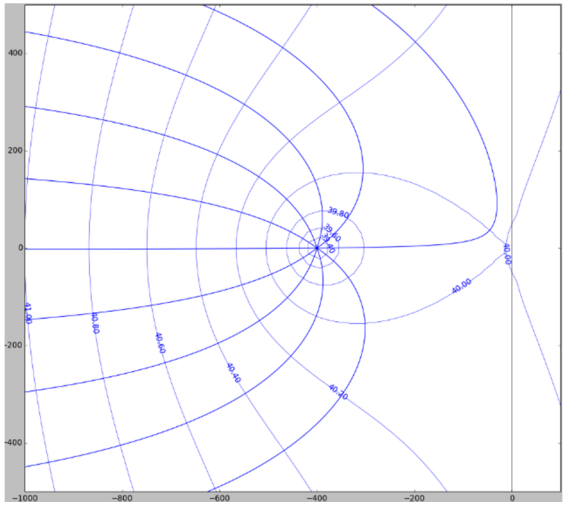


(TimML)

Figure 3-7: Single Well Capture Zone



(TimML-Cloud)



(TimML)

Figure 3-8: Single Well Capture Zone

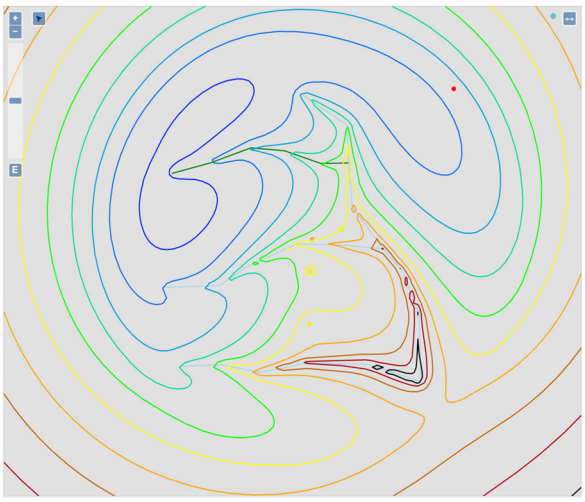
3.3.2 A System with Wells, Rivers, and Recharge

To verify the multi-aquifer modeling capabilities of TimML-Cloud, the final example consisted of three wells, a river system defined by stages and resistances, and a fixed area recharge element (Bakker, 2017c). Three layers comprise the aquifer with varied porosities, hydraulic

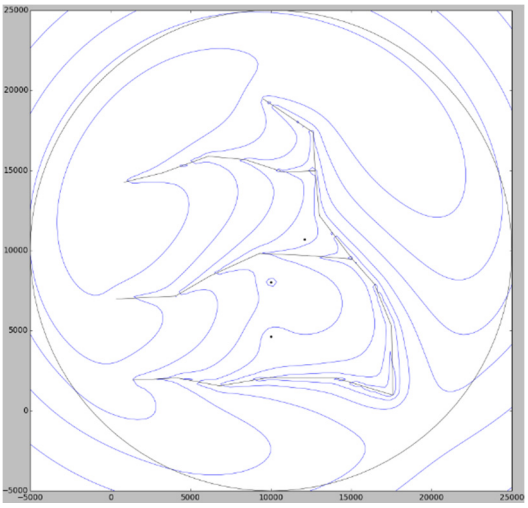
conductivities, and thicknesses. Using only the topmost aquifer layer for visual comparison, no significant variations were found in the solutions generated by both the web application and the scripting environment using TimML (Figure 3-8).

I performed particle tracking in the three different layers using TimML-Cloud and TimML. The two programs are expected to vary in the plotting of results when crossing layers within the aquifer as TimML-Cloud cannot currently plot trace results crossing layer boundaries. The first comparison made was with particles placed in the third well in the lowest aquifer layer. Within the same starting layer TimML-Cloud and TimML yield identical results, but when crossing layer boundaries the plotted results vary. Looking carefully at Figure 3-9 shows that when traces cross aquifer layers the lines do not plot in TimML-Cloud while they continue in a different color in TimML. The same plotting variation between TimML-Cloud and TimML is observed with the capture zones created by the two programs for the second well (Figure 3-10). When particles do not cross layers, no variations are found between TimML-Cloud and TimML solutions (Figure 3-11).

One new capability exposed in the TimML-Cloud web application is the ability to retrieve the full capture zone detail including time of travel for each particle and reason for termination (Figure 3-12). I drew no attribute comparisons between TimML-Cloud and TimML given that TimML does not natively have the methods to extract particle tracking attribute data when using the well capture zone methods.

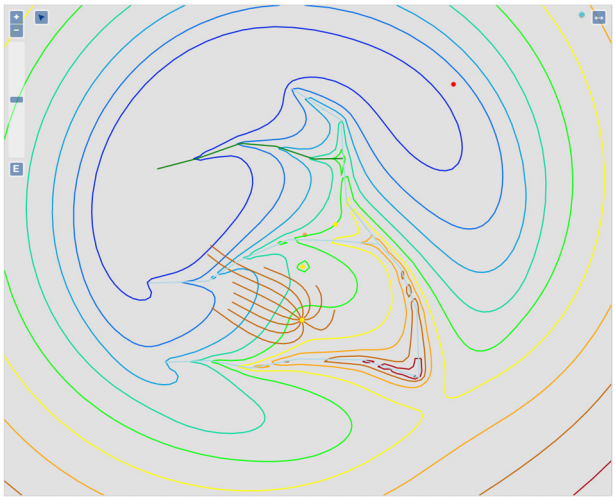


(TimML-Cloud)

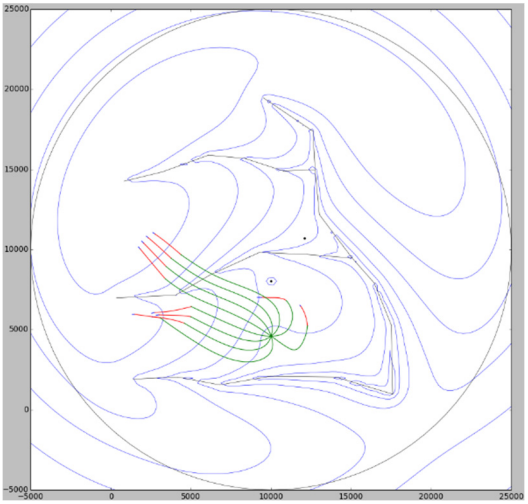


(TimML)

Figure 3-9: Multi-Aquifer Solution

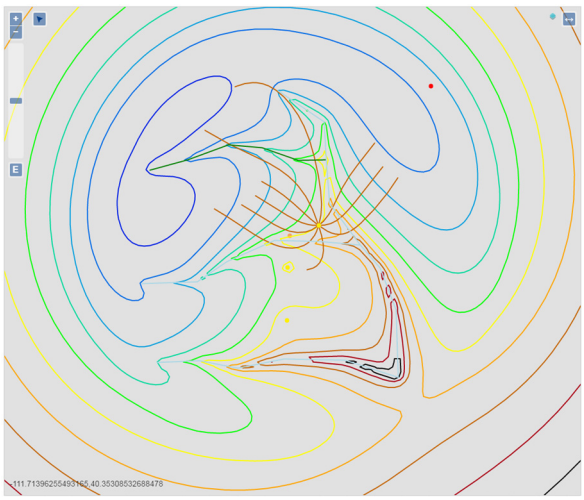


(TimML-Cloud)

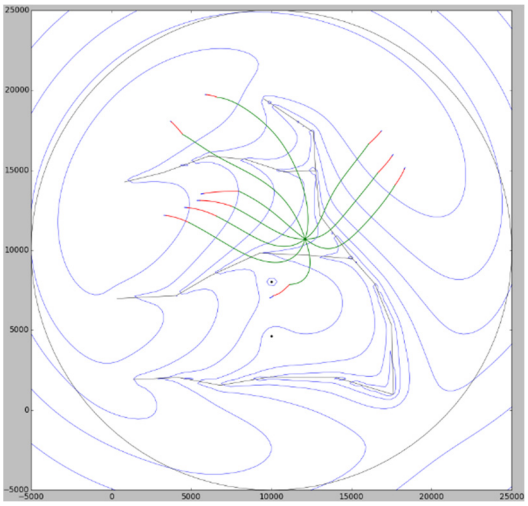


(TimML)

Figure 3-10: Well 3 Capture Zone

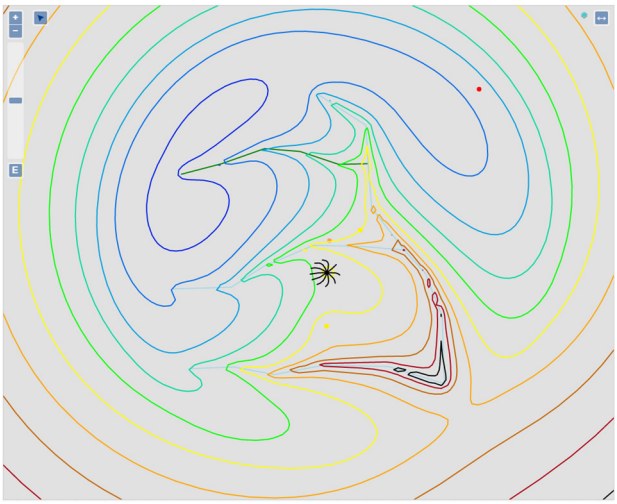


(TimML-Cloud)

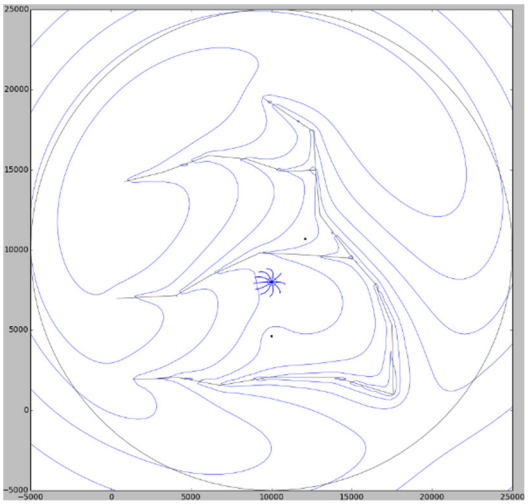


(TimML)

Figure 3-11: Well 2 Capture Zone

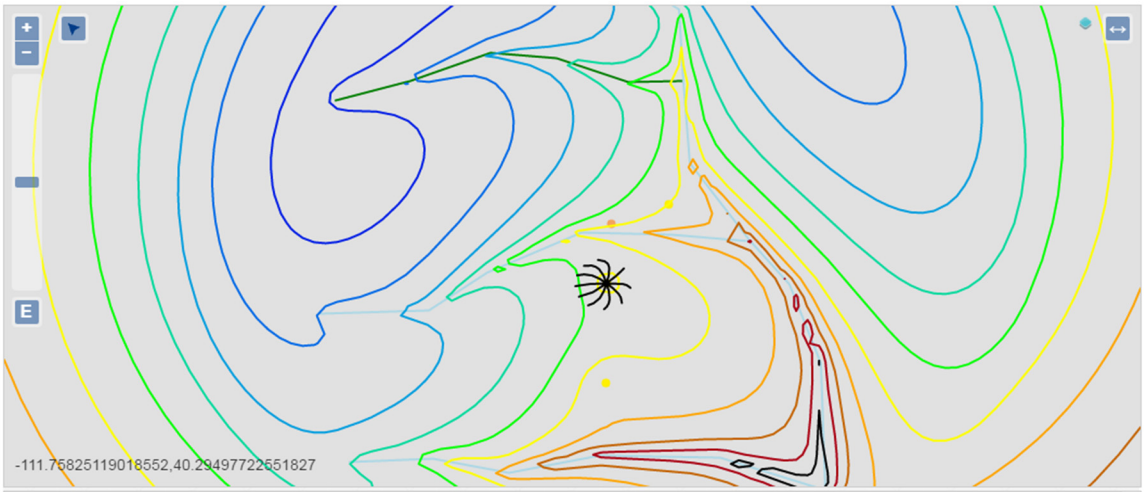


(TimML-Cloud)



(TimML)

Figure 3-12: Well 1 Capture Zone



total_time	layer	end_element	elem_ID	elem_label	end_state
-56408.61439344187	0		well_0	Well_1	flowed out of top of aquifer system
-73118.08526338454	0		well_0	Well_1	flowed out of top of aquifer system
-93284.2317670327	0		well_0	Well_1	flowed out of top of aquifer system
-88559.15939447292	0		well_0	Well_1	flowed out of top of aquifer system
-61329.95565823788	0		well_0	Well_1	flowed out of top of aquifer system
-70099.6440307935	0		well_0	Well_1	flowed out of top of aquifer system
-65347.723275019285	0		well_0	Well_1	flowed out of top of aquifer system
-59490.4295010295	0		well_0	Well_1	flowed out of top of aquifer system
-56011.961150741205	0		well_0	Well_1	flowed out of top of aquifer system
-75547.38260543415	0		well_0	Well_1	flowed out of top of aquifer system

Figure 3-13: Well 1 Capture Zone

4 CONCLUSIONS

For my thesis research, I created three web apps that perform various groundwater modeling tasks. The Dupuit Dewatering App focused primarily on the influence of wells in construction dewatering scenarios. The TimML Dewatering App allowed users to visualize the influence of wells and a cutoff wall, also used for construction dewatering simulations. TimML-Cloud was the culminating project that enabled users to perform more general groundwater modeling tasks. With each web app I was able to improve the user interface of Tethys platform gizmos and expose groundwater modeling capabilities. I improved on the user interface by creating a tool for deleting spatial features and I added a control that prevented users from modifying solution layers. My culminating contribution to the user interface provided by the Tethys platform was my work in creating the *Map Layout* object and the new *Table of Contents* gizmo. The ways I exposed groundwater modeling capabilities was by incorporating the Dupuit equation for calculating drawdown in the Dupuit Dewatering App and by integrating TimML for both the TimML Dewatering and TimML-Cloud Apps.

Hosting TimML as a cloud-based service has lowered the barrier to utilizing groundwater models in water resource management. Having TimML function on the internet as a cloud-based service reduces the educational barriers to learning how to use TimML. The particular benefit of using TimML over most other software packages with regards to typical educational requirements is its simplified data input. TimML requires input that most hydrogeologists either have or can

reasonably estimate and allows technicians to perform preliminary studies with relative ease compared with other programs like MODFLOW. The first step that made it possible to host TimML on the internet was wrapping TimML's FORTRAN extensions into a readable format.

The process of wrapping the TimML repository of code into a Linux readable format was made possible through the use of a third-party library called F2PY. Using F2PY to wrap the FORTRAN extensions into shared objects allowed TimML to be read and deployed on Linux machines. Once packaged in a readable format, TimML was uploaded as a custom library for Tethys applications to access. The final step that enabled users to access TimML as a cloud-based service was the creation of a web-based application.

Three web apps were created in sequence, each one building off the preceding one. To support each successive web application, the Tethys platform was expanded with improvements and modifications to existing gizmos and a new gizmo was created. A new *Map Layout* object was created and added to the Tethys platform. This enabled a standardized implementation of prearranged gizmos with the ability to customize specific features within each gizmo.

Two example problems were used to verify the accuracy of solutions generated using TimML-Cloud. Comparing the computed flow patterns from the first example involving a single aquifer layer showed no variation between TimML and TimML-Cloud. Particle tracking exercises also yielded identical results in the first example study. The second example involving a multi-layer aquifer system was used to further validate TimML-Cloud's accuracy. The computed steady state aquifer conditions were identical when comparing TimML-Cloud with TimML. With no significant solution differences between TimML and TimML-Cloud, the web application appears to have the full modeling accuracy of TimML.

Having TimML-Cloud operate on the internet as a cloud-based service enables automatic data integration. Groundwater models depend heavily on data, but incorporating data in groundwater models requires time and effort on the part of the modeling technician. The web interface of TimML-Cloud lends itself to the idea of automatically integrating other cloud-based databases a groundwater model. As an example, every stream network in the United States of America is delineated in a cloud database managed by the United States Geologic Survey (USGS). As shown by Souffront (2017) in his work with the National Water Center, stream segments can be added to TimML-Cloud's interface that automatically populate every available stream network with default stage values that can be modified and used by the user. Additionally, water table elevations for regions can be predetermined from existing well logs, borehole data, and satellite data. Automatically integrating cloud-based datasets would both increase the model's inherent value and decrease the time required to integrate the data.

REFERENCES

Bakker, M. (2013). Are All Models Wrong? Absolutely Not. *Groundwater, 51*(3), 313-313. doi:10.1111/gwat.12037

Bakker, M. (2015, August 20, 2015). A Multiaquifer Analytic Element Model. Retrieved from https://github.com/mbakker7/timml/blob/master/docs/timml.pdf

Bakker, M. (2017a). TimML. Retrieved from https://github.com/mbakker7/timml

Bakker, M. (2017b). TimML Notebook 0: Single Layer Flow. Retrieved from https://github.com/mbakker7/timml/blob/master/notebooks/timml_notebook0.ipynb

Bakker, M. (2017c). TimML Notebook 2: A system with wells, rivers, and recharge. Retrieved from https://github.com/mbakker7/timml/blob/master/notebooks/timml_notebook0.ipynb

Box, G. E., & Draper, N. R. (1987). *Empirical model-building and response surfaces* (Vol. 424): Wiley New York.

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., & Schaub, T. (2016). *GeoJSON*. Retrieved from https://datatracker.ietf.org/wg/geojson/charter/

Crawley, S. (2017). HydroShare GIS. Retrieved from https://github.com/hydroshare/tethysapp-hydroshare_gis

deepstream.io. (2017). GoldenLayout. Retrieved from https://www.golden-layout.com/

Esteban, E., & Dinar, A. (2016). THE ROLE OF GROUNDWATER-DEPENDENT ECOSYSTEMS IN GROUNDWATER MANAGEMENT. *Natural Resource Modeling, 29*(1), 98-129. doi:10.1111/nrm.12082

Haitjema, H. M. (1995). Chapter 5 - Analytic Element Modeling *Analytic Element Modeling of Groundwater Flow* (pp. 203-305). San Diego: Academic Press.

Holt, A. H. (2016). What is the difference between a cloud and web based application? Retrieved from https://www.quora.com/What-is-the-difference-between-a-cloud-and-web-based-application

Howard, J., & Merrifield, M. (2010). Mapping groundwater dependent ecosystems in California. *PLOS ONE, 5*(6), e11249.

Jones, D. J. (2012). *A Server-Based Tool for Automating MODFLOW Simulations for Well permitting Decision Support.* (Master of Science Thesis), Brigham Young University, Brigham Young University. Retrieved from http://scholarsarchive.byu.edu/etd/3333 (3333)

Jones, N. L. (2017). Seepgage and Slope Stability Analysis. Retrieved from http://ce544.groups.et.byu.net/syllabus/

Kraemer, S. R., Haitjema, H. M., & Kelson, V. A. (2007). Working with WhAEM2000 Capture Zone Delineation for a City Wellfield in a Valley Fill Glacial Outwash Aquifer Supporting Wellhead Protection. *EPA*.

Ludwig, R., Mauser, W., Niemeyer, S., Colgan, A., Stolz, R., Escher-Vetter, H., . . . Hennicker, R. (2003). Web-based modelling of energy, water and matter fluxes to support decision making in mesoscale catchments—the integrative perspective of GLOWA-Danube. *Physics and Chemistry of the Earth, Parts A/B/C, 28*(14–15), 621-634. doi:http://dx.doi.org/10.1016/S1474-7065(03)00108-6

Meyer, B. (1988). *Object-oriented software construction* (Vol. 2): Prentice hall New York.
Mohan, L., Posner, A. J., Postel, S., Treiber, T. G., Covitt, B. A., Hinojosa, T. T., . . . Miller-Rushing, A. (2012, November 12, 2012). Earth's Freshwater. *National Geographic Education.* Retrieved from http://nationalgeographic.org/media/earths-fresh-water/
Murray, B. B. R., Zeppel, M. J. B., Hose, G. C., & Eamus, D. (2003). Groundwater-dependent ecosystems in Australia: It's more than just water for rivers. *Ecological Management & Restoration, 4*(2), 110-113. doi:10.1046/j.1442-8903.2003.00144.x
OpenLayers. (2107). OpenLayers. Retrieved from https://openlayers.org/
Rushton, K. R. (2005). Background to Groundwater Flow *Groundwater Hydrology* (pp. 9-59): John Wiley & Sons, Ltd.
Souffront, M. (2017, December 7, 2016). National Water Model Forecast Viewer. Retrieved from https://github.com/msouff/nwm-forecast-viewer
Swain, N. R. (2015). *Tethys Platform: A Development and Hosting Platform for Water Resources Web Apps.* (Doctor of Philosophy Dissertation), Brigham Young University, Brigham Young University. Retrieved from http://scholarsarchive.byu.edu/etd/5832 (5832)