How to Analyze Tidal Responses with the SlugTide MATLAB code suite

Stephanie Nale, Earth & Planetary Sciences Dept., University of California, Santa Cruz

Introduction

SlugTide is a suite of MATLAB codes used to analyze tidal responses from water well data. These codes were developed in the University of California, Santa Cruz Seismology Lab to study tidal responses following earthquakes and to analyze permeability changes across fault zones. This document describes the process of calculating and plotting the response of water wells from earth tides using the SlugTide MATLAB code suite. The SlugTide procedure involves preprocessing of raw data from water wells, data import into MATLAB, construction of a data structure for wells in the dataset, processing of synthetic tidal models, and calculation of the amplitude and phase response of the water wells to the tidal model.

Users have two options in processing the dataset:

(a) For quick processing, follow Steps 1-7 and 15 (data must be preprocessed in Step 1)
(b) For users requiring more control with options to adjust the codes to their dataset, use Steps 1-6 and 8-15.

Earthquake Studies

The SlugTide MATLAB suite includes code to analyze tidal response to an earthquake occurring within the timeframe of the data collection. These codes are commented out initially. If the user would like to utilize this code, they must uncomment these parts of code designated by `%%%% EQ`. Scripts that include earthquake code are:

UserParam.m
PlotOriginalWL.m
PlotWellResponse.m

If there is more than one earthquake within the data, the user may either investigate each earthquake one at a time by changing `EQtime` in UserParam.m or process all earthquakes at the same time by adding additional code to correspond with each earthquake occurrence within the dataset

Synthetic Tidal Model

SlugTide uses a synthetic tidal model to calculate the tidal response. We prefer to use the program SPOTL by D.C. Agnew to compute the synthetic tidal model used for tidal analysis in SlugTide, as both solid earth and ocean tides may be calculated. More information on this program can be found at: <http://igppweb.ucsd.edu/~agnew/Spotl/spotlmain.html>.

Sample dataset

A sample dataset of two wells from the Santa Susana Field Laboratory in Simi Valley, CA has been included within this package (Appendix I). The directory "C-1_Pump_Test_Example" includes data files for two wells and the same codes as the SlugTide suite, updated to process these specific wells. Some edits were made to accommodate differences in these wells as well as the occurrence of an earthquake within the dataset. More information on these example wells can be found following the description of the steps to processing well data using the SlugTide MATLAB suite (see pages 21-25).

Comparison to Baytap08

In Appendix II we include results from our comparison of SlugTide to Baytap08, which is commonly used by researchers to perform tidal response studies. In comparing the two programs, we determined that the observed phase offset is due to the difference in tidal models used for each program (see page 26). More information on this program can be found at: <http://igppweb.ucsd.edu/~agnew/Baytap/baytap.html>.

Further reading

The SlugTide codes are based on code originally written by Lian Xue, modified through multiple iterations of tidal response calculations, and have been used in part in previous studies. See the following references:

> Lai, G., Ge, H., Xue, L., Brodsky, E.E., Huang, F., and Wang, W., 2014. Tidal response variation and recovery following the Wenchuan earthquake from water level data of multiple wells in the nearfield, Tectonophysics (2014), http://dx.doi.org/10.1016/j.tecto.2013.08.039.
>
> Xue, L., Li, H.B., Brodsky, E.E., Xu, Z.Q., Mori, J.J., Wang, H., Kano, Y., Si, J.L., Pei, J.L., Zhang, W., Yang, G., Sun, Z.M. and Huang, Y., 2013. Continuous permeability measurements record healing inside the Wenchuan Earthquake Fault Zone, *Science*, **340**, 6140, 1555-1559, DOI: 10.1126/science.1237237.
>
> Elkhoury, J.E., Brodsky, E.E. and Agnew, D., 2006. Seismic waves increase permeability, *Nature*, **441**, 1135-1138, DOI:10.1038/nature04798.

For further reference on tidal analysis using water well data, please see:

> Doan, M.-L., Brodsky, E., Prioul, R., Signer, C., 2006. Tidal analysis of borehole pressure: A tutorial. Schlumberger OilField Services Research. December 20, 2006. <http://www.pmc.ucsc.edu/~seisweb/emily_brodsky/reprints/tidal_tutorial.pdf>.

Contact:

Emily Brodsky
Professor of Earth and Planetary Sciences
University of California, Santa Cruz
brodsky@ucsc.edu

Usage

The SlugTide code is freely available for academic and non-profit use.

Publications using the complete SlugTide suite are forthcoming. Users may reference the (Xue *et al.,* 2013) paper, as SlugTide uses the same method.

SlugTide Process

1. **Create the SlugTide Directory**
   Unzip folder "SlugTide.zip". This will create a directory with all thirteen (13) MATLAB files and the folders "C-1_Pump_Test_Example", "Documentation", and "TideSynth". This will be your working directory and should contain:

   avg_amp_phase.m
   BuildWell.m
   importfile.m
   InitiateWells.m
   LoadTides.m
   MaterWell.m
   nanmean.m
   nanstd.m
   PlotOriginalWL.m
   PlotWellResponse.m
   UserParam.m
   WaterResp_general_interp.m
   water_respon_time.m
   "C-1_Pump_Test_Example"
   "Documentation"
   "TideSynth"

   The functions nanmean.m and nanstd.m are included with this package as they are not standard in every MATLAB installation but are used in the SlugTide codes.

   For quick calculations, users will need to preprocess their data to match the importfile.m format, and then edit UserParam.m, and InitiateWells.m as necessary (Steps 1-7). For more detailed control on the calculations, users may also want to further edit importfile.m, BuildWell.m, PlotOriginalWL.m, WaterResp_general_interp.m, water_respon_time.m, PlotWellResponse.m, and avg_amp_phase.m as needed to better match their dataset (Steps 3 and 8-15).

2. **Prepare the data files**
   Create individual .csv files for your data as columns of numeric data only. If you have an unusual format, the easiest way to get started is to save the data as two columns of text with the first column being days from 00:00:00 0/0/000 (MATLAB datenum convention) and the second being your water level depth. We refer to this as the "Simple" format (Format code S).

   Other formats are also available by following the instructions below.

   If you are starting from an Excel file:

a. In the original Excel file, change the formatting for the date/time columns to serial number. If date and time are in separate columns in original file, merge into one column before converting to serial number.

b. Before saving, make sure that all decimals of data and time values are fully displayed in columns to ensure accuracy of date/time and data values in the .csv file. This will affect the calculated tidal response.

c. Save individual .csv files per each sheet of data needed from original .xls file.

d. Save all .csv files into the same folder as the MATLAB codes.

3. **Import the data; importfile.m**

For this step, you can either: (a) preprocess your .csv file into a pre-defined format or (b) edit the importfile function to better match your data.
The predefined formats are listed in section (a) below.

a. Predefined format codes:

Format S:

Well with no ports, no header lines
Column syntax:
Col1: Time in days following MATLAB convention for time 00:00:00 0/0/0
Col2: Water level data (units ft)
Col3: Empty1
Col4: Empty2
Col5: Empty3
Col6: Empty4
Col7: Empty5
Col8: Empty6
Col9: Empty7
Col10: Empty8
Col11: Empty9
Col12: Empty10
Col13: Empty11
Col14: Empty12
Col15: Empty13
Col16: Empty14
Col17: Empty15
Col18: Empty16
Col19: Empty17
Col20: Empty18
Col21: Empty19
Col22: Empty20
Col23: Empty21
Col24 Empty22

Col25: Empty23
Col26: Empty24
Col27: Empty25
Col28: Empty26
Col29: Empty27
Col30: Empty28
Col31: Empty29
Col32: Empty30
Col33: Empty31
Col34: Empty32
Col35: Empty33
Col36: Empty34
Col37: Empty35

Format A:
37 header lines; Well with no ports
column syntax:
Col1: DateTime (units days)
Col2: ETmin (units minutes)
Col3: EThours (units hours)
Col4: ETdays (units days)
Col5: dtw (units feet btc)
Col6: drawdown (feet)
Col7: GWElev (water column elevation, units feet msl)
Col8: Empty1
Col9: Empty2
Col10: Empty3
Col11: Empty4
Col12: Empty5
Col13: Empty6
Col14: Empty7
Col15: Empty8
Col16: Empty9
Col17: Empty10
Col18: Empty11
Col19: Empty12
Col20: Empty13
Col21: Empty14
Col22: Empty15
Col23: Empty16
Col24 Empty17
Col25: Empty18
Col26: Empty19
Col27: Empty20
Col28: Empty21
Col29: Empty22

Col30: Empty23
Col31: Empty24
Col32: Empty25
Col33: Empty26
Col34: Empty27
Col35: Empty28
Col36: Empty29
Col37: Empty30

Format B:

37 header lines; Well with eight (8) ports
column syntax:
Col1: DateTime (units days)
Col2: ETmin (units minutes)
Col3: EThours (units hours)
Col4: ETdays (units days)
Col5: FeetH2OPorta (units feet)
Col6: FeetH2OPortb (units feet)
Col7: FeetH2OPortc (units feet)
Col8: FeetH2OPortd (units feet)
Col9: FeetH2OPorte (units feet)
Col10: FeetH2OPortf (units feet)
Col11: FeetH2OPortg (units feet)
Col12: FeetH2OPorth (units feet)
Col13: InchHg (units inches)
Col14: dtwPorta (units feet btc)
Col15: dtwPortb (units feet btc)
Col16: dtwPortc (units feet btc)
Col17: dtwPortd (units feet btc)
Col18: dtwPorte (units feet btc)
Col19: dtwPortf (units feet btc)
Col20: dtwPortg (units feet btc)
Col21: dtwPorth (units feet btc)
Col22: drawdownPorta (feet)
Col23: drawdownPortb (feet)
Col24: drawdownPortc (feet)
Col25: drawdownPortd (feet)
Col26: drawdownPorte (feet)
Col27: drawdownPortf (feet)
Col28: drawdownPortg (feet)
Col29: drawdownPorth (feet)
Col30: GWElevPorta (units feet msl)
Col31: GWElevPortb (units feet msl)
Col32: GWElevPortc (units feet msl)
Col33: GWElevPortd (units feet msl)
Col34: GWElevPorte (units feet msl)

Col35: GWElevPortf (units feet msl)
Col36: GWElevPortg (units feet msl)
Col37: GWElevPorth (units feet msl)

b. Edit import function to match data being imported

i. Look at files in the dataset to find the maximum number of columns (N) in the original data files (e.g. some wells have 7 columns in data files, others have 37), and change the importfile function:

Example: N = 37 columns

```
function [Column1, Column2, Column3, Column4, Column5,
Column6,  Column7, Column8, Column9, Column10,
Column11, Column12, Column13, Column14, Column15,
Column16, Column17, Column18, Column19, Column20,
Column21, Column22, Column23, Column24, Column25,
Column26, Column27, Column28, Column29, Column30,
Column31, Column32, Column33, Column34, Column35,
Column36, Column37] = importfile(filename, startRow,
endRow, FormatCode)
```

ii. Change formatSpec to import N columns.

Example: N = 37 columns

```
formatSpec =
'%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
%s%s%s%s%s%s%s%s%s%s%[^\n\r]';
```

If data is not originally in .xls format and converted to .csv as described above, the user will need to edit formatSpec to match the data files to import the data correctly. See MATLAB TEXTSCAN documentation for more information.

iii. To run all wells in same import function, even when the individual well files differ in format of data collected, use `switch FormatCode` to import wells with different data files by case (`'S', 'A', 'B',` etc.). Each case in `FormatCode` represents a specific data format as given in the original data files. For wells that have the same data format in the original file, give them the same `FormatCode`. There will be a new `FormatCode` for each variation of the original data files.

Create each `FormatCode case` (`'S', 'A', 'B',` etc.) by matching Column1 through ColumnN with the appropriate column heading from the original file (for columns without data, name as Empty1, Empty2, etc.). In our example codes, `FormatCode case 'S'` is the simple format, `FormatCode case 'A'` is for wells with 7 columns of data, whereas `FormatCode case 'B'` is for wells with 37 columns of data (the maximum in this dataset, N=37 columns of data).

```
%% Allocate imported array to column variable names
switch FormatCode
        case 'S' % Wellname(s): Well S
```

```
                DateTime = dataArray{:, 1};
                WL = dataArray{:, 2};
                Empty1 = dataArray{:, 3};
                  ...
                Empty35 = dataArray{:, 37};

                Column1 = DateTime;
                Column2 = ETmin;
                Column3 = Empty1;
                  ...
                Column37 = Empty35;

        case 'A' % Wellname(s): Well A
                DateTime = dataArray{:, 1};
                ETmin = dataArray{:, 2};
                EThours = dataArray{:, 3};
                ETdays = dataArray{:, 4};
                dtw = dataArray{:, 5};
                drawdown = dataArray{:, 6};
                GWElev = dataArray{:, 7};
                Empty1 = dataArray{:, 8};
                  ...
                Empty30 = dataArray{:, 37};

                Column1 = DateTime;
                Column2 = ETmin;
                Column3 = EThours;
                Column4 = ETdays;
                Column5 = dtw;
                Column6 = drawdown;
                Column7 = GWElev;
                Column8 = Empty1;
                  ...
                Column37 = Empty30;

        case 'B' % Wellname(s): Well B
                DateTime = dataArray{:, 1};
                ETmin = dataArray{:, 2};
                EThours = dataArray{:, 3};
                ETdays = dataArray{:, 4};
                FeetH2OPorta = dataArray{:, 5};
                FeetH2OPortb = dataArray{:, 6};
                FeetH2OPortc = dataArray{:, 7};
                FeetH2OPortd = dataArray{:, 8};
                FeetH2OPorte = dataArray{:, 9};
                FeetH2OPortf = dataArray{:, 10};
                FeetH2OPortg = dataArray{:, 11};
                FeetH2OPorth = dataArray{:, 12};
                InchHg = dataArray{:, 13};
                dtwPorta = dataArray{:, 14};
                dtwPortb = dataArray{:, 15};
                dtwPortc = dataArray{:, 16};
                dtwPortd = dataArray{:, 17};
                dtwPorte = dataArray{:, 18};
                dtwPortf = dataArray{:, 19};
```

```
dtwPortg = dataArray{:, 20};
dtwPorth = dataArray{:, 21};
drawdownPorta = dataArray{:, 22};
drawdownPortb = dataArray{:, 23};
drawdownPortc = dataArray{:, 24};
drawdownPortd = dataArray{:, 25};
drawdownPorte = dataArray{:, 26};
drawdownPortf = dataArray{:, 27};
drawdownPortg = dataArray{:, 28};
drawdownPorth = dataArray{:, 29};
GWElevPorta = dataArray{:, 30};
GWElevPortb = dataArray{:, 31};
GWElevPortc = dataArray{:, 32};
GWElevPortd = dataArray{:, 33};
GWElevPorte = dataArray{:, 34};
GWElevPortf = dataArray{:, 35};
GWElevPortg = dataArray{:, 36};
GWElevPorth = dataArray{:, 37};

Column1 = DateTime;
Column2 = ETmin;
Column3 = EThours;
Column4 = ETdays;
Column5 = FeetH2OPorta;
Column6 = FeetH2OPortb;
Column7 = FeetH2OPortc;
Column8 = FeetH2OPortd;
Column9 = FeetH2OPorte;
Column10 = FeetH2OPortf;
Column11 = FeetH2OPortg;
Column12 = FeetH2OPorth;
Column13 = InchHg;
Column14 = dtwPorta;
Column15 = dtwPortb;
Column16 = dtwPortc;
Column17 = dtwPortd;
Column18 = dtwPorte;
Column19 = dtwPortf;
Column20 = dtwPortg;
Column21 = dtwPorth;
Column22 = drawdownPorta;
Column23 = drawdownPortb;
Column24 = drawdownPortc;
Column25 = drawdownPortd;
Column26 = drawdownPorte;
Column27 = drawdownPortf;
Column28 = drawdownPortg;
Column29 = drawdownPorth;
Column30 = GWElevPorta;
Column31 = GWElevPortb;
Column32 = GWElevPortc;
Column33 = GWElevPortd;
Column34 = GWElevPorte;
```

```
                            Column35 = GWElevPortf;
                            Column36 = GWElevPortg;
                            Column37 = GWElevPorth;
        end
```

We assign variables in this way so that all wells in the dataset can be imported using the same function with Columns 1 through N, but the original format of the data is saved in the script for documentation and reference (needed in BuildWell.m).

4. **Create the TydeSynth Directory**
    a. Create the synthetic tide files for the wells. Use SPOTL or preferred program. Each well processed will have its own synthetic tide file.

    b. Save the synthetic tide files for all the wells in the dataset in a separate folder "TideSynth" within the directory folder containing the .csv data files and MATLAB codes.

5. **Modify the parameters for the dataset; UserParam.m**
    a. Open UserParam.m and edit the parameters for the test being studied. Parameters from this file are used within later steps in the process to calculate and plot the tidal response. Updating the parameters in this part of the code will negate the necessity to edit some of the individual steps later in the process.

    b. If there are earthquake occurrences in the data, see the note on Earthquakes in the introduction.

6. **Set up the initial structure of wells for the test; InitiateWells.m**
    a. The script sets up the initial structure for the wells, which is input into building the Well structure for the dataset (BuildWell.m). Initially in InitiateWells.m, there are inputs for three wells, one per pre-defined format (`'S'`, `'A'` `and` `'B'`). The user will need to copy and paste the set of inputs by format `'S'`,`'A'`, or `'B'` for the rest of the wells in their dataset using these predetermined formats. Users can change the parameters and add wells as needed within InitiateWells.m for their test. If the user is adding different formats, they will need to update importfile.m and BuildWell.m (steps 3 and 8, respectively) in addition to InitiateWells.m.
        i. Determine how many wells are in the dataset and if there are ports, how many.
            1. For wells with no ports and no headerlines (simple, data only) copy and paste the code for Well S for each well.
            2. For wells with no ports, copy and paste the code for Well A for each well.

3. For wells with multiple ports, copy and paste the code for Well B for each well. This example has eight (8) ports. The user will need to edit each well to match the number of ports with data in that well.

ii. Edit iWell for each well
```
iWell=1;
```

iii. Edit `'filename'` for each well
```
WellInitial(iWell).filename='WellA.csv';
```

iv. Edit `'Wellname'` for each well
```
WellInitial(iWell).Wellname='Well A';
```

v. Edit `'format'` for each well
```
WellInitial(iWell).format='A';
```

vi. Edit `'startrow'` for each well
```
WellInitial(iWell).startrow=55;
```
This is the start of the data to be analyzed from the original data file

vii. Edit `'endrow'` for each well
```
WellInitial(iWell).endrow=inf;
```
This is the end of the data to be analyzed from the original data file

viii. Edit `'tidename'` for each well
```
WellInitial(iWell).tidename='tidename.dat';
```

ix. Edit number of ports for each well.
1. If well has no ports:
```
WellInitial(iWell).nports=1;
```
2. If well has multiple ports (Example: 8):
```
WellInitial(iWell).nports=8;
```

x. Find periods of noise in the data to be removed.
```
WellInitial(iWell).noisestart1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend1='YYYY-MM-DD hh:mm:ss'
```
1. To determine what periods to remove as noise, we recommend inspecting the water level plot (Step 10), which is processed before the noise is removed from the data.
2. This code initially establishes one noise field per well or per port for multi-port wells. The user may add more noise fields as needed, but the user must then also edit BuildWell.m to accommodate these additional noise fields (see step 8.a and 8.b.xiv).

   Example, for a well with no ports and three noise fields:
```
WellInitial(iWell).noisestart1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noisestart2='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend2='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noisestart3='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend3='YYYY-MM-DD hh:mm:ss'
```

Or, for a well with ports, and three (3) noise fields in the first port and two (2) noise fields in the second port:

```
WellInitial(iWell).noisestart_1_1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend_1_1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noisestart_1_2='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend_1_2='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noisestart_1_3='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend_1_3='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noisestart_2_1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend_2_1='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noisestart_2_2='YYYY-MM-DD hh:mm:ss'
WellInitial(iWell).noiseend_2_2='YYYY-MM-DD hh:mm:ss'
% etc.
```

xi. Establish the depth to open interval for the well/ports

```
do=80; % depth to open interval in ft
WellInitial(iWell).dom=(do)*0.3048; % convert to meters
```

xii. Repeat this process for each well in the dataset.

1. Be sure to carefully edit each step described above per well in the dataset, and make adjustments for the number of ports for multi-port wells.

7. **If using the quick method (utilizing UserParam.m and InitiateWells.m, following preprocessing), skip to Step 15: MasterWell.m**
Steps 8 through 14 describe the process of calculating and plotting the tidal response, with instructions on where to edit if needed.
Exception: If the user has made changes to InitiateWells, be sure to make the necessary corresponding changes as described above.

8. **Load Tide data; LoadTides.m**
This function loads the synthetic tides into the data structure within BuildWell.m
This code calls variables established in UserParam.m.
a. Call the time interval of the synthetic tide data.

```
UserParam;
...
t=Well.t0_tide(i)+(0:length(y)-1)'*Tide_dt;
```

9. **Build the Well Structure for the dataset; BuildWell.m**

This part of the code is written as a function that sets up the Well Structure for the dataset through an `'if elseif'` loop using the formats as assigned to the wells in InitiateWells.m.
This code calls variables established in UserParam.m.

a. Establish the variables used in BuildWell.m from the fields saved into the initial well structure in InitiateWells.m
    i. If the user has added any new fields to InitiateWells.m, they will need to establish the new variable here:
    Example:
    Add more noise fields (as in Step 6.e)
    We recommend indicating changes to the code with `%%`

```
     noisestart1=WellInitial.noisestart1; % already
     established
     noiseend1= WellInitial.noiseend1; % already established
     noisestart2= WellInitial.noisestart2;                  %%
     noiseend2= WellInitial.noiseend2;                      %%
     noisestart3= WellInitial.noisestart3;                  %%
     noiseend3= WellInitial.noiseend3;                      %%
    if(nports>1)
     noisestart_1_1= WellInitial.noisestart_1_1;            %%
     noiseend_1_1= WellInitial.noiseend_1_1;                %%
     noisestart_1_2= WellInitial.noisestart_1_2;            %%
     noiseend_1_2= WellInitial.noiseend_1_2;                %%
     noisestart_1_3= WellInitial.noisestart_1_3;            %%
     noiseend_1_3= WellInitial.noiseend_1_3;                %%
     noisestart_2_1= WellInitial.noisestart_2_1;            %%
     noiseend_2_1= WellInitial.noiseend_2_1;                %%
     noisestart_2_2= WellInitial.noisestart_2_2;            %%
     noiseend_2_2= WellInitial.noiseend_2_2;                %%
     % etc.
    end
```

b. Set up the structure for each well in the dataset in BuildWell.m.
    i. If the user has updated importfile.m to create another format, or change one of the initial formats (`'S', 'A' or 'B'`), they will need to update BuildWell.m.

```
        if format=='S'
                % rest of code
        elseif format=='A'
                % rest of code
        elseif format=='B'
                % rest of code
        else ['Error: unknown format']

        end
```

When adding a new format:

1. If the wells in the new format have no ports, copy the code for Format `'A'`, and edit to match the new format case.
2. If the wells in the new format do have ports, copy and paste the code

for Format `'B'`, but edit the number of ports to match the new format.

ii. Edit importfile function for each format:
```
[Column1,...,ColumnN] = importfile(filename, startRow,
endRow, format);
```

iii. Edit date/time column (atime) per format
Example
```
atime=Column1;
```

iv. Edit water column elevation data (adata) per format
Example
```
adata=Column7;
```

1. If well has ports, select columns for adata_1, adata_2, etc. for water column elevation data of each port in the well
```
adata_1=Column30;
adata_2=Column31;
% etc.
```

v. For wells with multiple ports, update rest of code for each well to match the number of ports in the well.

vi. Call the well name from the initial well structure
```
Well.name=Wellname;
```

vii. If the data contains NaN (missing data from original file), remove those points from the dataset
```
indiceNaN = isnan(Well.WL);
Well.WL = Well.WL(indiceNaN~=1);
Well.time = Well.time(indiceNaN~=1);
```

viii. If repeated times in data, remove the repeated times
```
num=find(diff(dtime)==0);
dtime(num)=[];
%Remove the data at repeated times
ddata(num)=[];
```

ix. Call the number of ports from the initial well structure
```
Well.num_ports=nports;
```

x. Input the name of the synthetic tide file for each well.
```
Well.synthfn=strvcat([TideDir tidename]);
```

xi. Input the start date of the synthetic tide files for the test.
```
Well.t0_tide=[datenum(tidedate)]+tcUTC; % UTC
```

xii. Input the time interval dt for the well data.

```
Well(iWell).dt=dt; % units days
```

xiii. If the time interval of the data collection is not constant throughout the data time frame or the user would prefer to process at a different interval, interpolate the data to the appropriate time interval (e.g. dt=10 minutes).

```
xi=Well.tUTC(1):Well.dt:Well.tUTC(end);
yi=interp1(Well.tUTC,ddata_1,xi);
Well.xi=xi; % interpolated time
Well.yi=yi; % interpolated data
```

xiv. Find periods of noise in the data to be removed.

1. If the noise fields are being added in Step 6.e, add the inputs for the additional noise fields here:

```
Well.noise=[datenum('noisestart1')
datenum('noiseend1')
datenum('noisestart2') datenum('noiseend2')       %%
datenum('noisestart3') datenum('noiseend3')];     %%
```

c. Repeat this process for each format in the dataset.

i. For each new format that the user adds to importfile.m (Step 3), the user will also need to update InitiateWells.m (Step 6) and BuildWell.m to reflect these changes.

ii. Be sure to carefully edit each step described above per format in the dataset, and make adjustments for the number of ports for multi-port wells.

10. Plot Original Water Level; PlotOriginalWL.m

Plot the water level per well/port from the original data (not the interpolated data). This code calls variables established in UserParam.m.

a. If there is an earthquake in the data, uncomment all earthquake related code in PlotOriginalWL.m, designated by `%%%% EQ`.

b. Establish time of EQ, if present in the data

```
EQ_t=datenum(EQtime)+cUTC; %%%%UTC %%%% EQ
```

c. Establish time frame to plot

```
xlim([datenum(PlotTime1) datenum(PlotTime2)])
```

d. Plots are saved in both .pdf and .png file formats.

11. Calculate the tidal components; water_respon_time.m

This function performs the Fourier analysis and calculates the tidal components called within WaterResp_general_interp.m.

a. The code calculates the response of the M2, S2, K1 and O1 tidal components.

i. Other tidal components may be added within this function, as needed for the particular tidal study.

12. Calculate the tidal response; WaterResp_general_interp.m

This code calls variables established in UserParam.m.
The sign convention for these calculations is extension positive.

a. This code processes the data through a loop for wells with multiple ports. Results for wells with no ports are saved as a double in the Well Structure, whereas results for multiport wells are saved as a cell.

```
for iport=1:Nports,
    if (Nports==1)
        data=Well(iWell).yi;
    else
      data=Well(iWell).yi{iport};
    end
... %(rest of the script)
end
```

b. Only the M2 component is called to calculate the phase and amplitude response in WaterResp_general_interp.m

```
Ph=Respon.pha_M2;
A0=Respon.amp_M2;
T0=Respon.time;
Ph_deg=Ph*180/pi;
```

i. The user may change the component used in calculations as needed. Either:

1. Replace M2 with the wanted frequency. Example:

```
Ph=Respon.pha_S2;
A0=Respon.amp_S2;
T0=Respon.time;
Ph_deg=Ph*180/pi;
```

2. Add an additional tidal component for analysis. This will require adding script to the rest of WaterResp_general_interp.m as well as PlotWellResponse.m and avg_amp_phase.m to plot the additional frequencies.
In WaterResp_general_interp.m:

```
Ph0=Respon.pha_M2;
A0=Respon.amp_M2;
Ph_deg0=Ph0*180/pi;
Ph00=Respon.pha_S2;
A00=Respon.amp_S2;
Ph_deg00=Ph00*180/pi;
T0=Respon.time;

Ph_deg1=[Ph_deg Ph_deg0];
A1=[A A0];
Ph_deg2=[Ph_deg Ph_deg00];
A2=[A A00];
T=[T T0];
```

```
% save the results in the Well structure
Well(iWell).Ph1(:,iport)=Ph_deg1;
Well(iWell).A1(:,iport)=A;
Well(iWell).Ph2(:,iport)=Ph_deg2;
Well(iWell).A2(:,iport)=A2;
Well(iWell).T0(:,iport)=T;
```

In PlotWellResponse.m:

```
Well(iWell).T=Well(iWell).T0(:,i);
Well(iWell).A1=Well(iWell).A1(:,i);
Well(iWell).Ph1=Well(iWell).Ph1(:,i);
ff=find(Well(iWell).Ph1>180);
Well(iWell).Ph1(ff)=Well(iWell).Ph1(ff)-360;
Well(iWell).A2=Well(iWell).A2(:,i);
Well(iWell).Ph2=Well(iWell).Ph2(:,i);
ff=find(Well(iWell).Ph2>180);
Well(iWell).Ph2(ff)=Well(iWell).Ph2(ff)-360;
...etc. Continue on to add/change script to plot
the second amplitude and phase
```

In avg_amp_phase.m:

```
Well(iWell).T=Well(iWell).T0(:,iport);
Well(iWell).A1=Well(iWell).A1(:,iport);
Well(iWell).Ph1=Well(iWell).Ph1(:,iport);
Well(iWell).A2=Well(iWell).A2(:,iport);
Well(iWell).Ph2=Well(iWell).Ph2(:,iport);
...etc. Continue on to add/change script to plot
the second amplitude and phase
```

13. Plot the Phase and Amplitude response; PlotWellResponse.m

This code calls variables established in UserParam.m.

a. If there is an earthquake in the data, uncomment all earthquake related code in PlotWellResponse.m, designated by `%%%% EQ`.

   i. If there is an earthquake:

      1. Input the timing of the earthquake.

         ```
         EQ_t=datenum(EQtime)+cUTC; %%%%UTC %%%% EQ
         ```

      2. Adjust the window on either side of the earthquake. This window should match the window of analysis from WaterResp_general_interp.

         ```
         t_pre=EQ_t-29.5307; %%%% EQ
         t_post=EQ_t+29.5307; %%%% EQ
         ```

b. This code processes wells initially by plotting phase and amplitude response in wells with no ports, and then in wells with multiple ports, plotting ports individually. Next, for wells with multiple ports the code plots the phase and amplitude with all ports together on one plot.

```
if Nports==1
     %Plot response by port
     for i=1:Nports
     ... %(rest of the script)
     end

else

     for i=1:Nports
     %Plot response by port
     ... %(rest of the script)

     %Plot one graph with all ports
     for iport=1:Nports
     ...%(rest of the script)
     end
end
```

c. Establish time frame to plot

```
xlim([datenum(PlotTime3) datenum(PlotTime4)])
```

d. If adding additional frequencies to WaterResp_general_interp.m, make sure to edit PlotWellResponse.m appropriately, as in Step 12.d.i.2.

e. Plots are saved in both .pdf and .png file formats.

14. Calculate and plot the average amplitude and phase for wells with multiple ports over a specified time frame; avg_amp_phase.m

This in an additional code that plots the average amplitude and phase response in wells with depth of the ports for study of vertical control on tidal response.

a. Establish timeframe of response to plot in avg_amp_phase.m

```
[dum ind_data1]=min(abs(Well(iWell).T-datenum(PlotTime5)));
[dum ind_data2]=min(abs(Well(iWell).T-datenum(PlotTime6)));
```

b. If adding additional frequencies to WaterResp_general_interp.m, make sure to edit PlotWellResponse.m appropriately, as in Step 12.d.i.2.

c. Plots are saved in both .pdf and .png file formats.

15. Process the dataset; MasterWell.m

a. Run MasterWell.m on the entire dataset or specific wells, as needed.
This code calls variables established in UserParam.m and InitiateWells.m.

```
UserParam;
InitiateWells; % build the initial well input structure,
inputs used to create Well structure
```

b. To save processing time, BuildWell.m may be run once with the Well Structure saved and commented out in MasterWell.m for subsequent data runs where there is no change to the Well Structure.

```
%%%%%%%%%%%%%% Build Metadata Structure with waterlevel
data fields
for iWell=1:NWells
    [WellArray(iWell)]=BuildWell(WellInitial(iWell));
end
Well=WellArray; % rename the full array to "Well"
save WellStruct Well;
```

Remember to run BuildWell.m and save the Well Structure each time the well parameters input into the initial Well structure are changed (UserParam.m or InitiateWells.m) or if BuildWell.m itself is edited.

Appendix I

Sample wells: C-1 Pump Test from the Santa Susana Field Laboratory, wells WS-14 and HAR-1.
Data from these two wells were collected over the duration of the C-1 pump test on the Santa Susana Field Laboratory in Simi Valley, California, USA. This dataset spans August 2003 through June 2004, and includes the San Simeon earthquake (Mw 6.5) of December 22, 2003. Publications detailing results from these data are forthcoming.

The two wells selected for this example fit the two formats (`‘A’` and `‘B’`) described in Step 3, and we have also created a file to fit the Simple format (`‘S’`). Edits have also been made to the codes accommodate different numbers of ports and the earthquake occurrence. Changes made to the original SlugTide codes in this example have been indicated by `%%` in the same line, following the change. Because this dataset includes an earthquake occurrence, the relevant code has been uncommented for this example, as described in Steps 5, 10 and 13.

For this example, LoadTides.m, water_respon_time.m, avg_amp_phase.m and MasterWell.m are soft-linked from the main SlugTide directory, as these codes remain unchanged.

1. WS-14 is a water supply well that displays a good response to tidal forcing. This well does not have multiple ports, matching the format `'A'` described in Step 3.

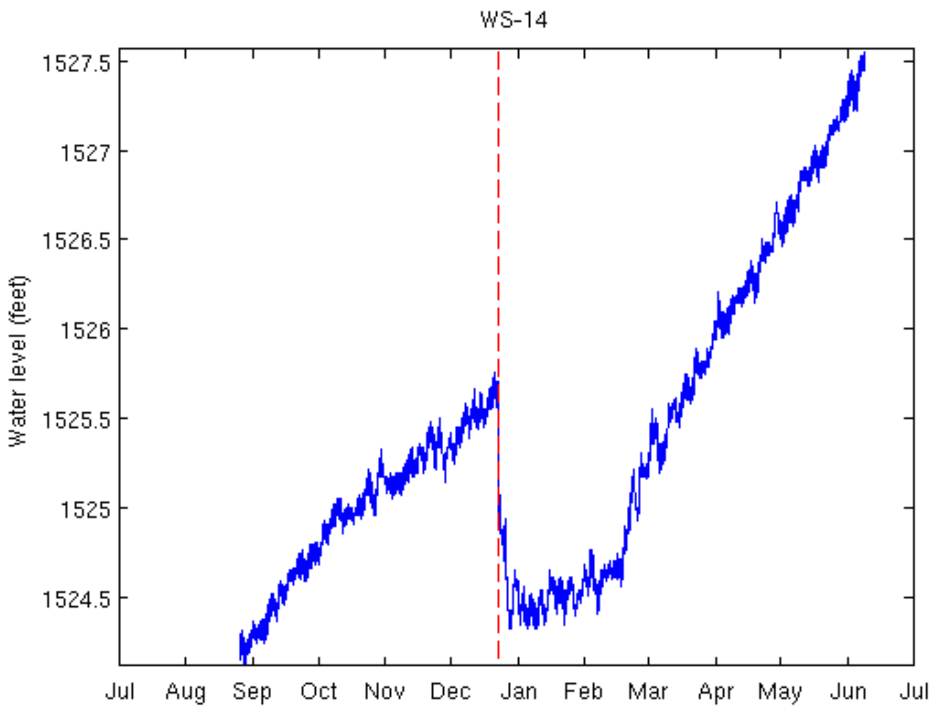


Figure 1. Water level plot for well WS-14. This well showed a dramatic drop in water level coincident with the occurrence of the San Simeon earthquake on December 22, 2003 (red dashed vertical line), with water level recovery over the following months.

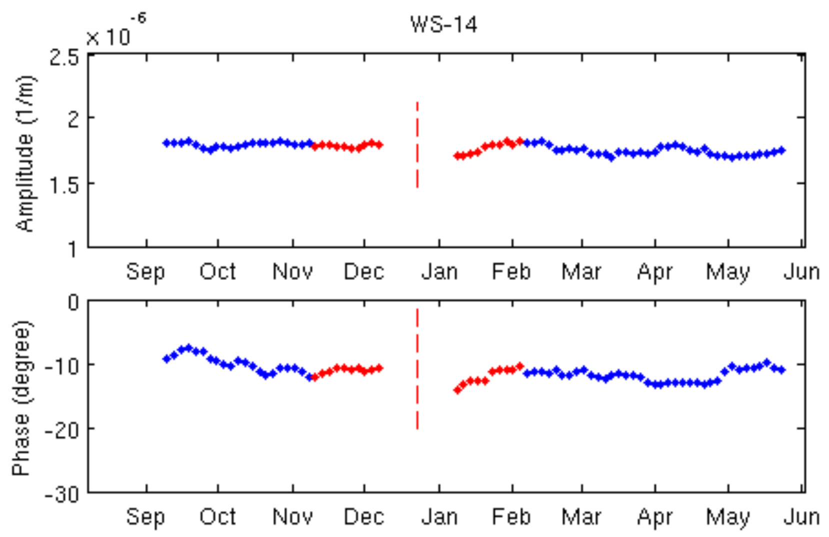


Figure 2. Amplitude and phase of the tidal response for well WS-14 (blue dots). The San Simeon earthquake is indicated by the red dashed vertical line, with the window surrounding that event indicated by red dots for visual inspection of the affect of the earthquake on the phase response.

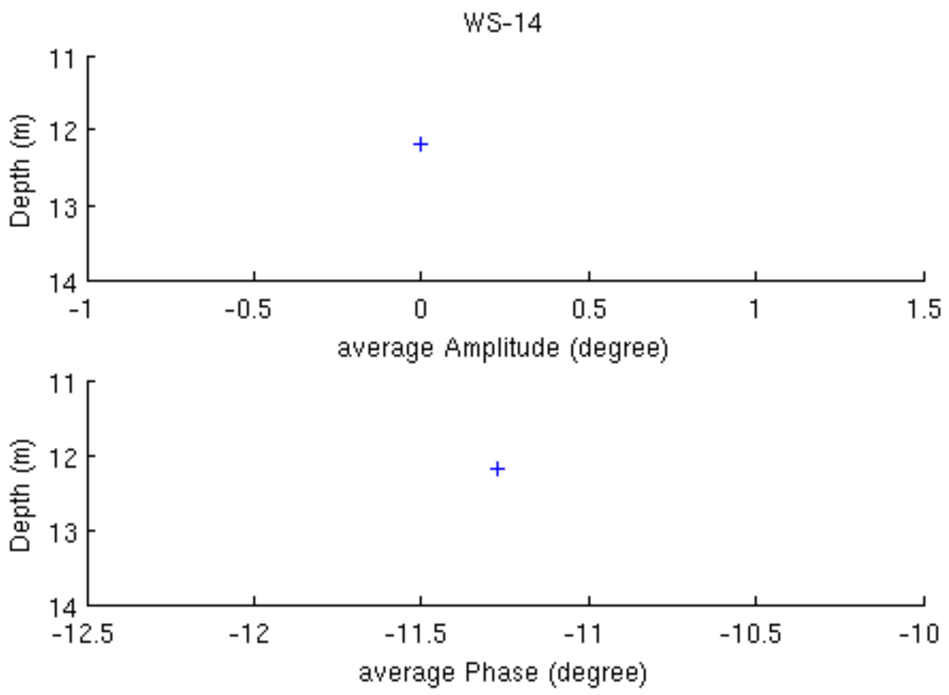


Figure 3. Average amplitude and phase. Since WS-14 does not have multiple ports, this figure shows the average amplitude and phase of the tidal response for the well over the entire timeframe.

Because we used data from the same well (WS-14), formats `'S'` and `'A'` should produce the same results.

2. HAR-16 has multiple ports, making format `'B'` as described in Step 3 the appropriate format for processing. Because one of the ports was dry and one port had a bad transducer during the data collection period, some edits were made to Initiate Wells (Step 6) and BuildWell (Step 9) to accommodate a well with only six ports of data. Examining the tidal response of each port may give more information on the vertical controls of the phase and amplitude response of the well.

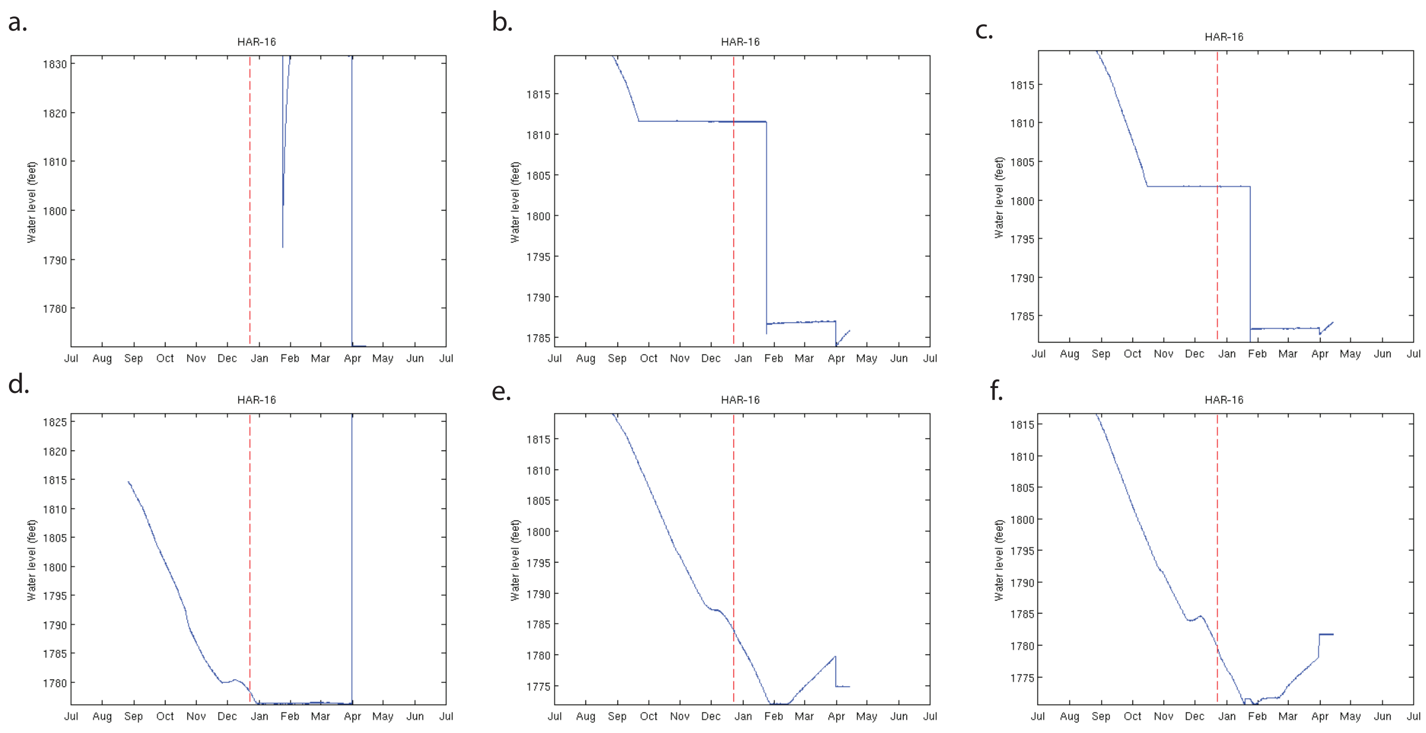
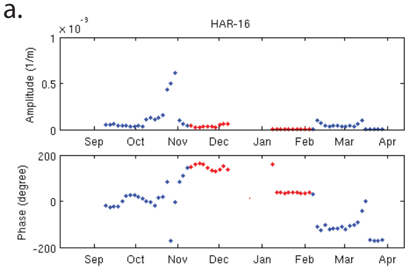
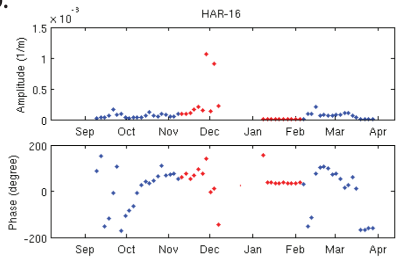


Figure 4. Water level for each port in HAR-16. Ports 1, 2, 3, 4, 5, 6 in the script correspond to a, b, c, d, e, f respectively. The red dashed vertical line indicates the San Simeon earthquake.

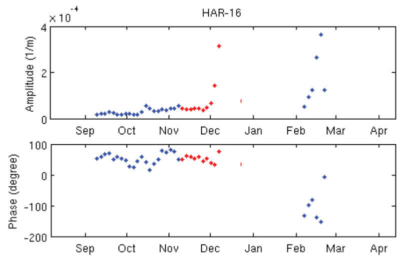
a.



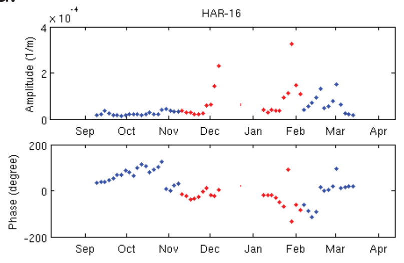
b.



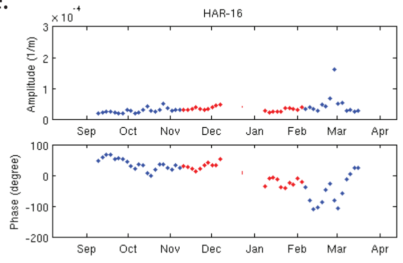
c.



d.



e.



f.

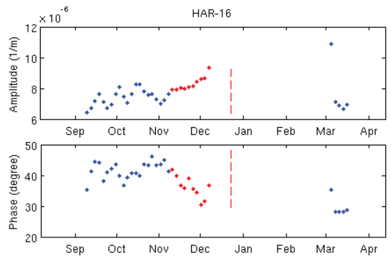


Figure 5. Amplitude and phase of the tidal response for each port in well HAR-16 (blue dots). Ports 1, 2, 3, 4, 5, 6 in the script correspond to a, b, c, d, e, f respectively. The window surrounding the San Simeon earthquake is indicated by red dots for visual inspection of the affect on the phase response.

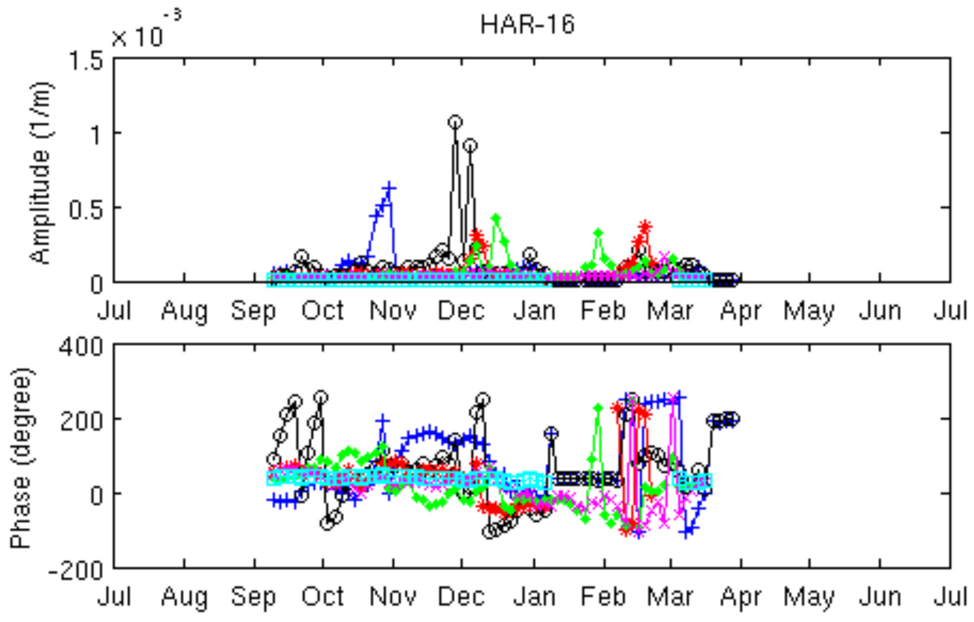


Figure 6. Amplitude and phase of the tidal response for all six ports in HAR-16. Aside from inspecting the tidal response of each well separately, this graph allows the user more ease in comparing response of each port in the well.

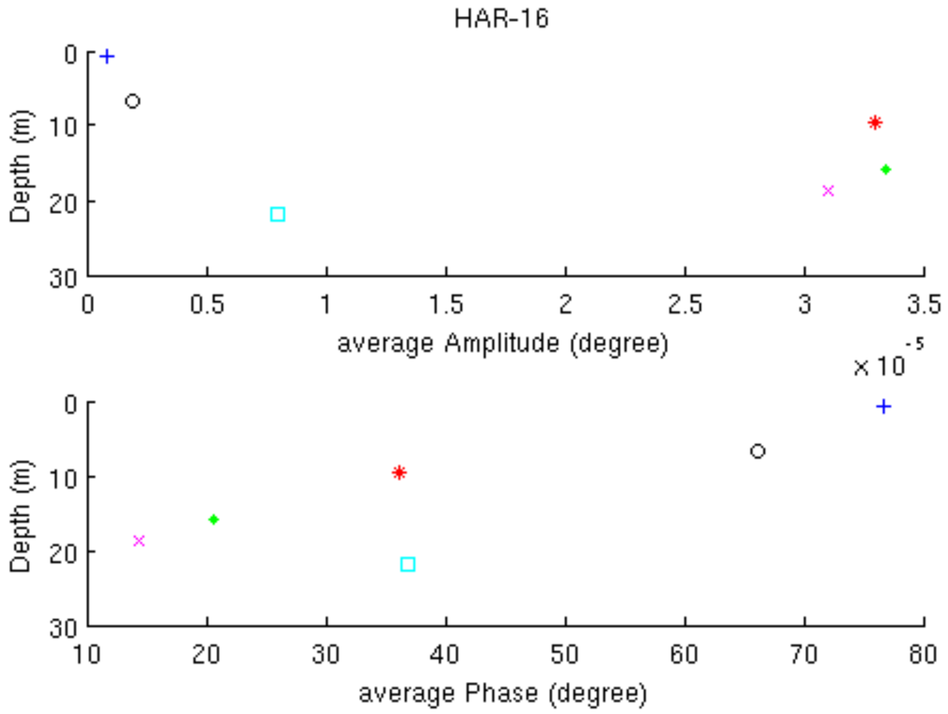


Figure 7. Average amplitude and phase per port in HAR-16 by depth of the six ports recording data.

Appendix II

SlugTide vs. Baytap08 Comparison

Baytap08: A Program for Analyzing Tidal Data is the program currently used by researchers for tidal response studies. When comparing our results from SlugTide to Baytap08, a phase difference was seen that could not be explained by a clock error.

Using the data from Baytap08's PatBay example, we compared the two programs to find differences in how they were processing the data that may lead to the observed phase shift. Upon investigation of the tidal models used in the two programs, we found that Baytap08 does not compute the ocean tides and its tidal constituents are based on the Cartwight table, whereas we compute solid earth with ocean loading (whole tides) in SPOTL to use in our SlugTide calculations.

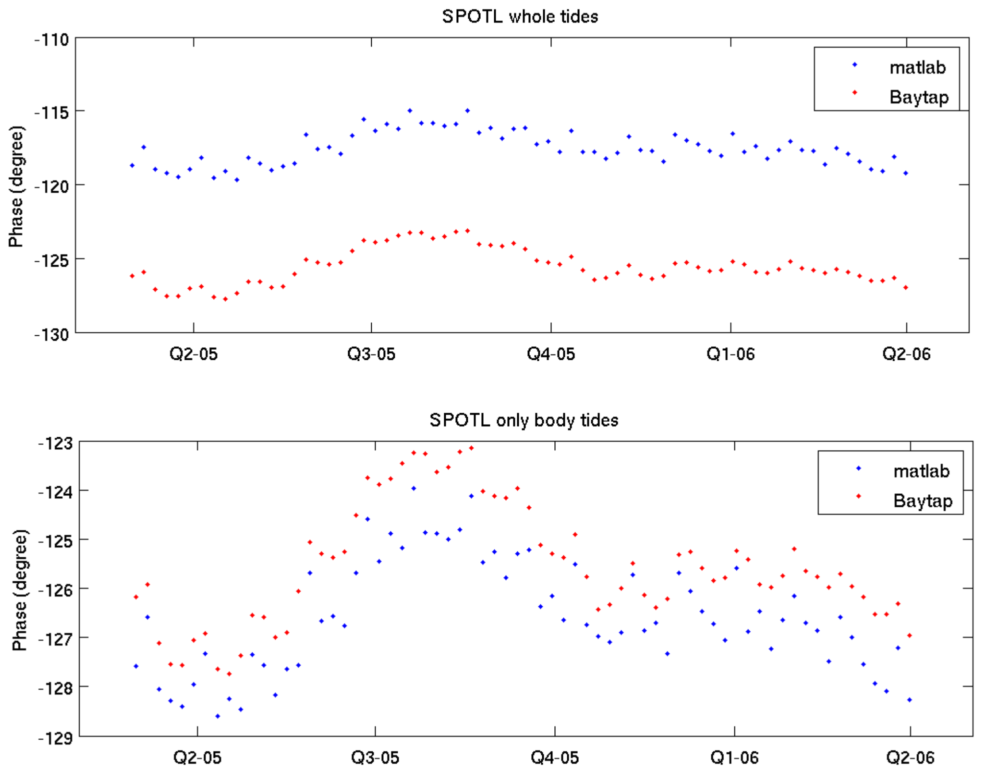


Figure 8. Upper panel: SlugTide result (blue) using whole tides (solid earth + ocean loading) from SPOTL is shifted in phase from Baytap08 result (red). Lower panel: SlugTide result excluding ocean tides is very close to the Baytap08 result.

From these results, we believe that Baytap08 may be more ideal for situations where only solid earth loading is important (no ocean loading) and when the barometric effect is important, whereas SlugTide is more ideal for data where ocean loading is important.