SW4 Notes

SW4 is a seismic wave propagation code developed at Lawrence Livermore National Laboratory. Multiple people within the Department of Earth Sciences at the University of Oregon are using SW4 for various projects. This directory contains information about using SW4. This is meant to complement the official user guide.

If you have tricks or tips that you have learnt, please create a relevant section heading and add them below. You can also upload input files, scripts for analyzing outputs, and other documents that you think may be relevant to other users to this document.

In addition to this documentation, there is a Slack channel dedicated to SW4 hosted through the University of Oregon Earth Sciences Slack account.

Running SW4 on Talapas

SW4 is installed as a module on Talapas, which is the University of Oregon's high performance computing cluster. SW4 can be easily loaded and runs efficiently. Talapas uses SLURM to submit jobs and manage resources. For more information about how to submit jobs on Talapas, see here. Below is an example submission script called `submit.sh` for the input file `example_input.in`. This provides the information about how many compute resources to use (nodes and tasks) before specifying the job to run.

```
#!/bin/bash

#SBATCH --partition=dufek     ### Partition
#SBATCH --account=dufeklab
#SBATCH --job-name=sw4     ### Job Name
#SBATCH --output=sw4.out  ### file in which to store job stdout
#SBATCH --error=sw4.err    ### file in which to store job stderr
#SBATCH --time=4:00:00      ### WallTime
#SBATCH --nodes=2            ### Number of Nodes
#SBATCH --ntasks-per-node=28   ### Number of tasks (MPI processes)

module load sw4

srun sw4 example_input.in
```

The job can be submitted by typing the following command where `$` denotes the command prompt.

```
$ sbatch submit.sh
```

Note that there are now two sw4 modules on Talapas. `sw4` is the standard version of sw4 downloaded directly from LLNL. `sw4-user-src` is a slightly modified version that works much better for user defined sources. More details are provided in the User Source section.

User Defined Sources

Contact: Leighton Watson lwatson2@uoregon.edu

Main takeaway: use module `sw4-user-src` instead of `sw4` if you want to use the discrete time function/user defined source capability in SW4

The files associated with the **User Defined Sources** section are located in the **Input/UserSrc** directory in the google drive folder.

The majority of users of SW4 will use the various built-in source functions. However, SW4 does have the capability to handle user defined sources. The user defined source needs to be supplied as a file with the following format.

Line	Column 1	Column 2	Column 3
1	t_0 (real)	δ_t (real)	N_d (integer)
2	g_1 (real)		
3	g_2 (real)		
$\vdots$	$\vdots$		
$N_d + 1$	g_{N_d} (real)		

t_0 = start time
$delta_t$ = time step
N_d = number of data points
g_i = value of user defined source function at time t_i

A user defined source function can be specified in the input file with the following syntax:

```
source x=1000 y=1000 z=0 fx=0 fy=0 fz=1 f0=1 dfile=src.sw4
```

where `src.sw4` is the source file with the format described above.

The standard version of sw4 does not work very well with user defined sources. We perform one simulation using the built-in source function using the input file `gauss.in`. The details of the source function are:

```
source x=1000 y=1000 z=0 fx=0 fy=0 fz=1 f0=1 type=Gaussian t0=1
freq=5
```

The result is shown in the figure below on the left side. The top plot shows the source used by the solver, which is saved in the output `g1.dat`. This is a four column matrix. The first column is the time vector and the second is the amplitude of the source function (the third and fourth columns are the first and second time derivative of the source function).

In order to test the user-defined source function, we save the source used in the `gauss.in` simulations in the required format for a user-defined source function, which we call `src_g1.sw4`. We then run the solver again using the input file `sw4.in`, which uses the user-defined source:

```
source x=1000 y=1000 z=0 fx=0 fy=0 fz=1 f0=1 dfile=src_g1.sw4
```

The result is shown in the right hand side of the figure below. Ideally, the result should be exactly the same as using the built-in source function (left) as we are performing the same simulation with the same input. This is not what is observed. Instead, the user-defined source solution is non-physical with many oscillations and unrealistic amplitude growth with time.

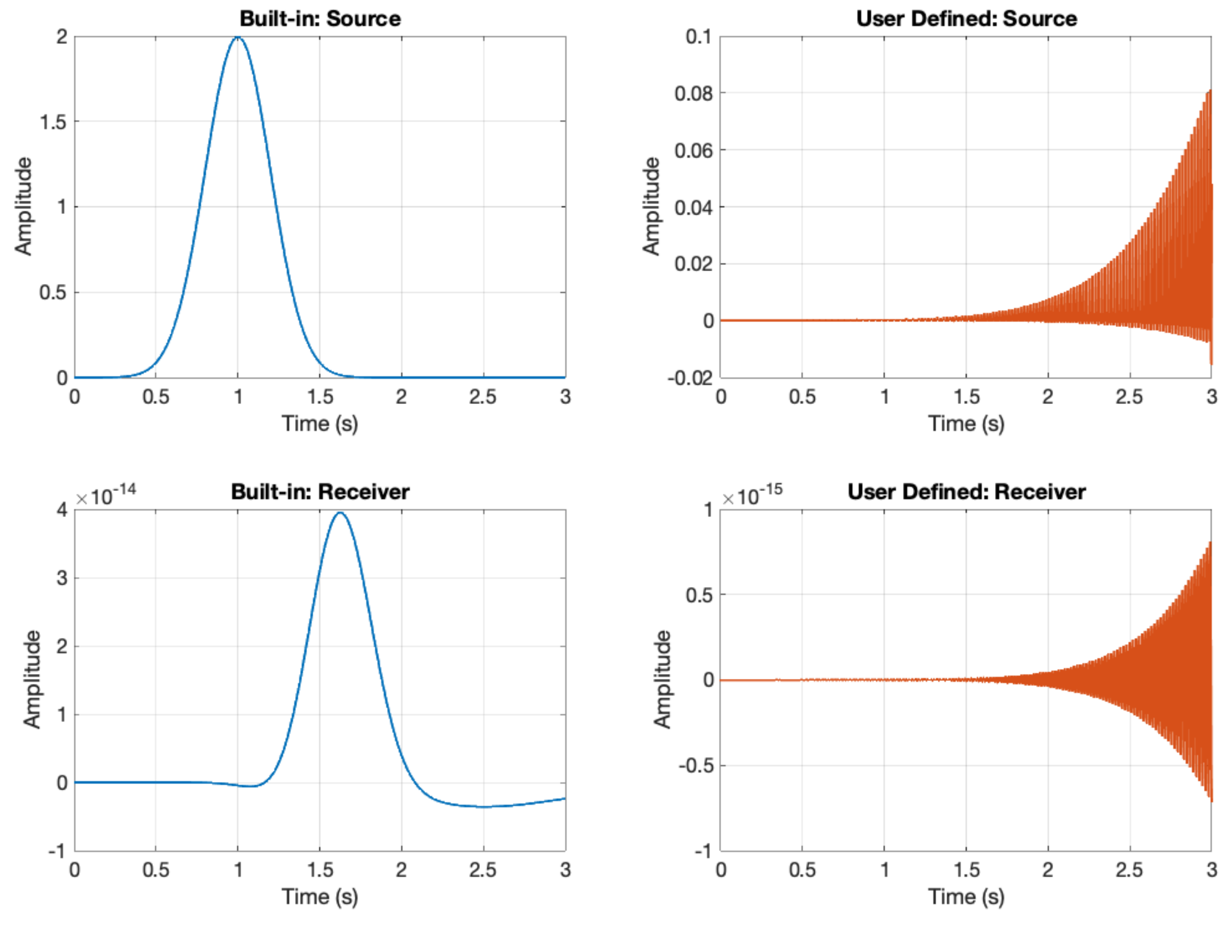


Figure: *comparison of source and receiver for SW4 simulation using the built-in source function (left, blue:* `gauss.in`*) and the user-defined source (right, red:* `sw4.in`*) using module* `sw4`*. The user-defined source simulation is non-physical. It has significant oscillations and grows in amplitude with time.*

I found the solution to this issue in this youtube video by Dr Po Chen from the University of Wyoming. SW4 interpolates the user-defined source function. Apparently, what is being plotted in the top right of the above figure are the coefficients of the spline interpolation. This can be fixed by commenting out lines 1432 and 1433 in source.c and then recompiling the code. The Talapas support staff have helped with this and created a new module called sw4-user-src.

We run the exact same simulation as before with the user-defined source but now use

```
module load sw4-user-src
```

The result is shown in the figure below. There is very good agreement between the user-defined source and the solution using the built-in source function. This is the expected result as both simulations should be using the exact same input parameters.

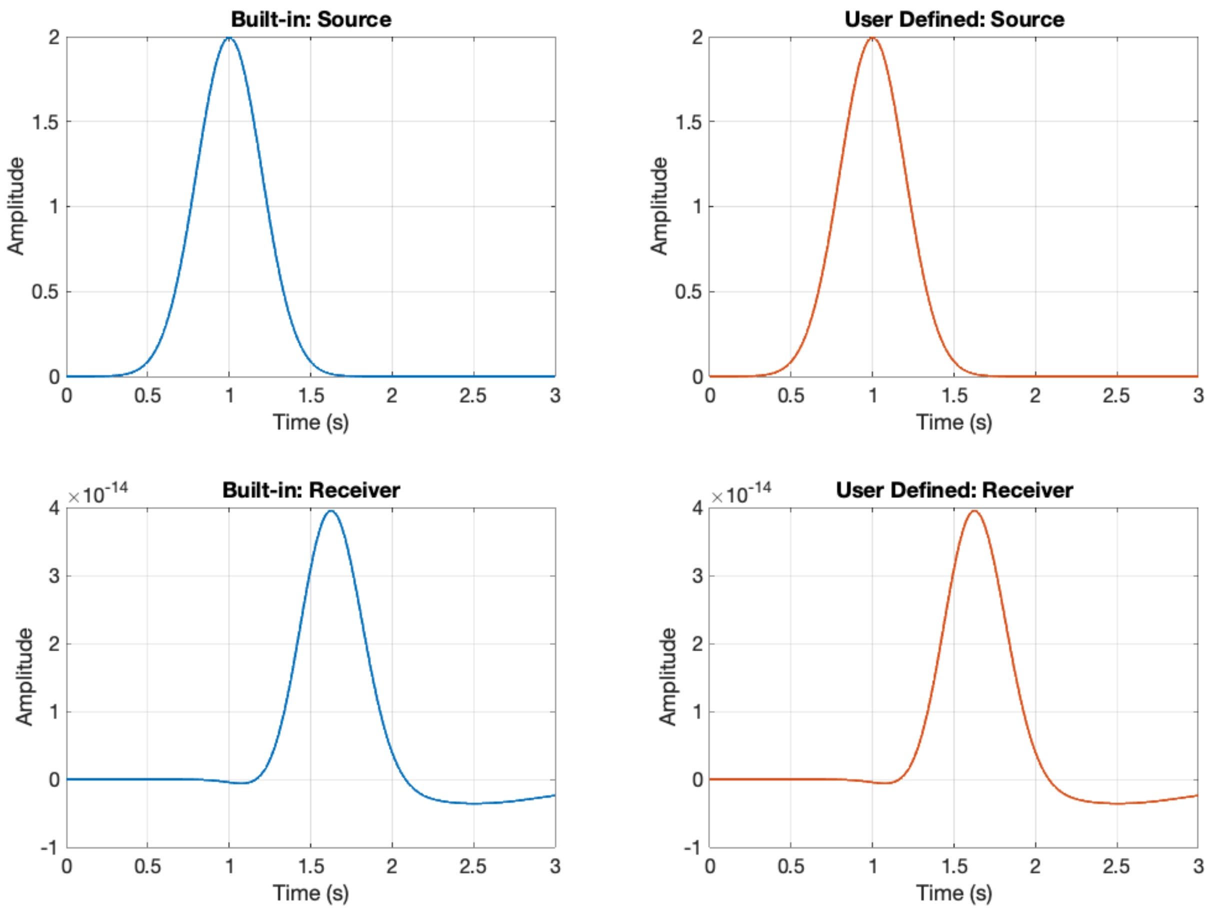


Figure: *comparison of source and receiver for SW4 simulation using the built-in source function (left, blue:* `gauss.in`*) and the user-defined source (right, red:* `sw4_user_src.in`*) using module* `sw4-user-src`*. The two solutions are now in good agreement.*

Topography

Contact: Leighton Watson lwatson2@uoregon.edu

There are multiple ways to define the shape of the top surface of the computational domain. Here, I show how to define topography using a cartesian grid of elevation values.

Topography is specified in the input file using the following syntax:

```
topography input=cartesian file=elev_file.topo zmax=200
```

`Cartesian` specifies that the topography file is provided in a cartesian grid. The file name is specified by `file` and should be in the same directory as the input file. A curvilinear grid is created between the surface specified in `elev_file.topo` and `zmax` (remembering that z is positive downwards). Suppose the topographic surface is given by *e(x,y)* where *e* is positive upwards, then according to the user guide choosing *zmax* = $-e_{grid} < e_{min} - 3(e_{max} - e_{min})$ should work well. See the user guide for more details.

The elevation file should be an ASCII text file with the following format. The first line of the file holds the number of data points in each direction.

$$Nx \quad Ny$$

The next lines should have three columns where the first column is the x coordinate, the second column is the y coordinate, and the third column is the elevation (positive upwards).

$$\begin{array}{ccc}
x_1 & y_1 & e_{1,1} \\
x_2 & y_1 & e_{2,1} \\
\vdots & \vdots & \vdots \\
x_{Nx} & y_1 & e_{Nx,1} \\
\vdots & \vdots & \vdots \\
x_1 & y_{Ny} & e_{1,Ny} \\
x_2 & y_{Ny} & e_{2,Ny} \\
\vdots & \vdots & \vdots \\
x_{Nx} & y_{Ny} & e_{Nx,Ny}
\end{array}$$

To visualize the outputs, use the image command in the input file to obtain a cross-section through the domain:

```
image mode=uz y=1000 cycleInterval=100 file=UZ
```

The simulation results can be read into matlab with the command `readimage` which takes two inputs. The first is the file name and the second is the patch number. The patch number for the cartesian grid is 1 and 2 for the curvilinear grid.

```
% load cartesian grid
[im,x,y,z,plane,t,timestring] = readimage(picturename,1);

% load curvilinear grid
[im,x,y,z,plane,t,timestring] = readimage(picturename,2);
```

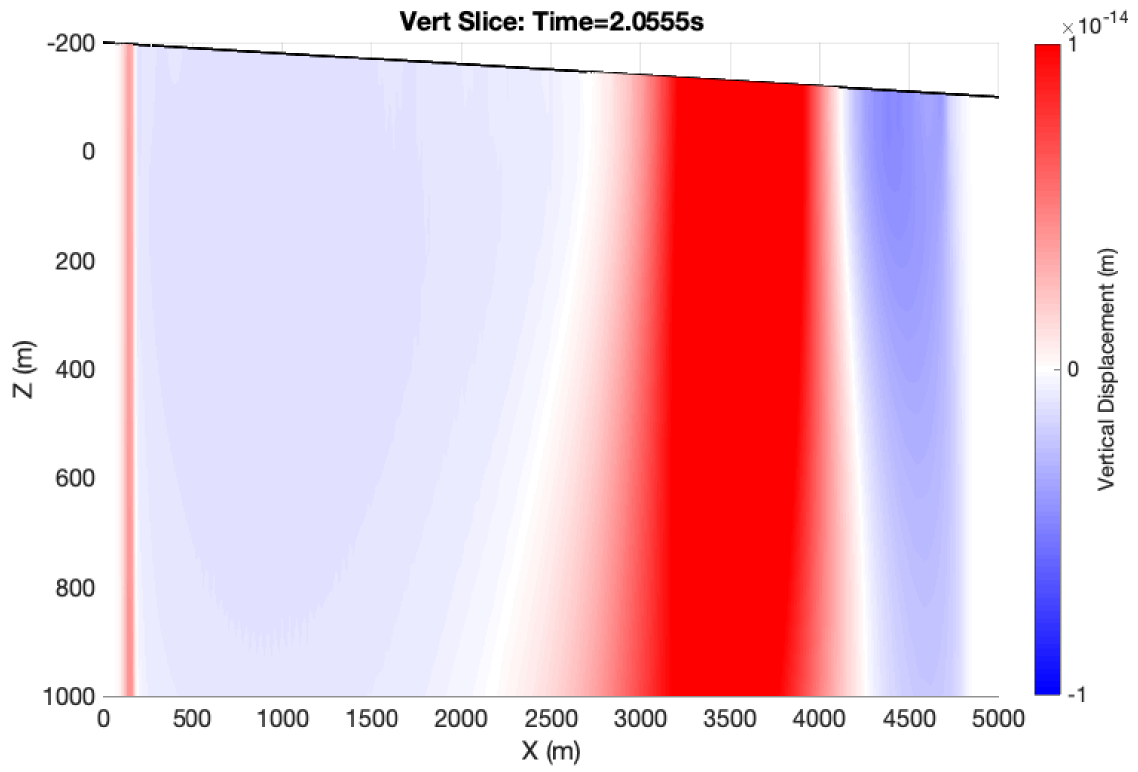


Figure: *Vertical cross-section for a simulation with topography. The surface is shown by the black line.*

Material Properties

Contact: Leighton Watson lwatson2@uoregon.edu

There are multiple ways to specify material properties for SW4. The simplest way is through the `block` command in the input file. I also provide information about using `pfile`, which allows for 3D heterogeneous material properties. See the user guide for other methods.

Block

The block command is the simplest way to specify material properties. It does not require any additional files and can be entirely specified in the input file. The following command sets the P and S velocities and density throughout the entire computational domain:

```
block vp=3000 vs=2000 rho=2000
```

Layer cake models can be easily produced by adding multiple blocks and specifying the depth extent:

```
block vp=4000 vs=2500 rho=2000
block vp=6000 vs=3500 rho=2700 z1=15000
block vp=8000 vs=4500 rho=3300 z1=35000 z2=100000
```

If the horizontal extent is not specified, then the block covers the entire horizontal extent of the computational domain. Smaller blocks can be specified by adding horizontal coordinates:

```
block vp=3000 vs=2000 rho=1000 \\
      x1=4000 x2=8000 y1=3000 y2=7000 z1=10000 z2=70000
```

The grad commands `(vpgrad, vsgrad, rhograd)` can be used to specify profiles that linearly vary with depth.

Pfile

The pfile command can be used to assign material properties based on depth profiles. This allows for heterogeneous material properties. The pfile contains the values of the material properties (P-velocity, S-velocity, and density) as a function of depths at points on a regular lattice covering the horizontal extent of the computational domain.

The number of grid points in the depth direction needs to be the same for all profiles, but the grid spacing does not have to be uniform and can be different for each profile. The lattice of the pfile does not need to have any relation to the computational mesh of SW4 and is often much coarser. The material properties in the computational mesh are assigned values using smoothing/interpolation.

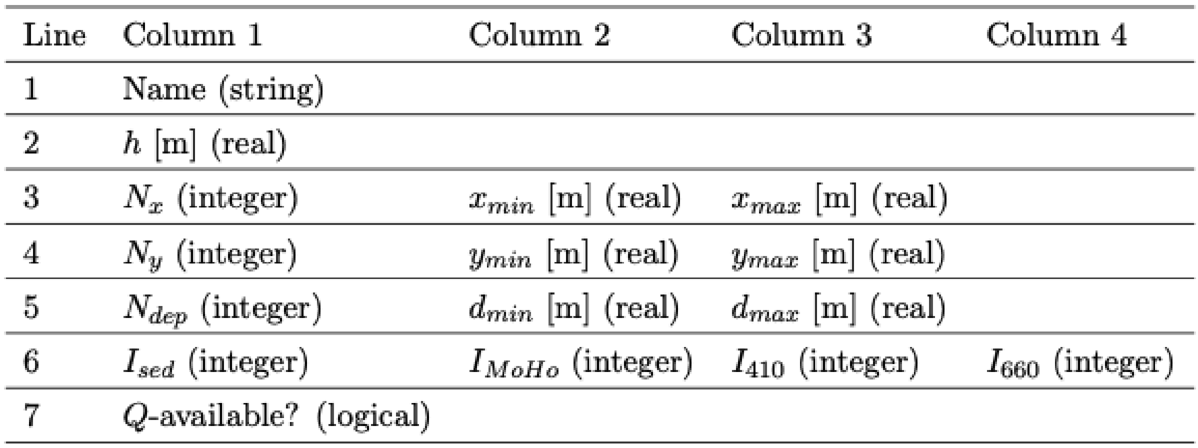
To use the pfile specification of the material properties, add the following line to your input file:

```
pfile filename=material.ppmod vsmin=1000 vpmin=1732 smoothingsize=4
\\ style=cartesian
```

The pfile with the material properties is specified by the text file `material.ppmod`. The minimum P-velocity and S-velocity are specified by `vsmin` and `vpmin. Smoothingsize` sets the number of points used in the horizontal smoothing of the material properties.

`style=cartesian` tells the solver that the horizontal grid is in cartesian coordinates (rather than latitude and longitude. To use geographic coordinates, remove this last command).

The pfile is an ASCII text file with a header and body that specifies the material properties in a given format. The header consists of 7 lines with the following format:



Line	Column 1	Column 2	Column 3	Column 4
1	Name (string)			
2	h [m] (real)			
3	N_x (integer)	x_{min} [m] (real)	x_{max} [m] (real)	
4	N_y (integer)	y_{min} [m] (real)	y_{max} [m] (real)	
5	N_{dep} (integer)	d_{min} [m] (real)	d_{max} [m] (real)	
6	I_{sed} (integer)	I_{MoHo} (integer)	I_{410} (integer)	I_{660} (integer)
7	Q-available? (logical)			

Line 1: name of the material model
Line 2: horizontal spacing of the depth profiles
Line 3: number of depth profiles in the x direction, as well as minimum and maximum coordinates values in x direction
Line 4: number of depth profiles in the y direction, as well as minimum and maximum coordinates values in y direction
Line 5: number of depth values in each depth profile, as well as the minimum and maximum depth
Line 6: supplies optional information about material discontinuities in each depth profile. This is more applicable for large scale earthquake simulations where the MoHo or the 410 km or 660 km discontinuities in the mantle are relevant. Set these to -99 if you want to ignore them.
Line 7: indicates if attenuation (quality factors) are included in the pfile. Set to .TRUE. or .FALSE.

The first seven lines of my pfile for testing material properties looks like:

```
Pfile_test
40
126 0.000000 5000.000000
51 0.000000 2000.000000
51 0.000000 2000.000000
-99 -99 -99 -99
.FALSE.
```

After the header, the body of the text provides the material properties in a series of *Nx* by *Ny* depth profiles with the x coordinate varying fastest (i.e., depth profile at *Nx1 Ny1* followed by *Nx2 Ny1*, *Nx3 Ny1*,...).

The first line of each depth profile holds the x-coordinate and y-coordinate and the number of depth values, which must equal *Ndep* from the header. The subsequent Ndep lines have the following format:

Index (int)	depth [m]	C_p [m/s]	C_s [m/s]	ρ [kg/m^3]	Q_P	Q_S

The first column is the index (increasing from 1 to Ndep), the second column is the depth, and the next three columns are the P-velocity, S-velocity, and density. The last two columns are the quality factors and can be neglected if the Q-availability flag on line 7 of the header is `.FALSE.` An extract from the example pfile used here is shown below:

```
0.000000 0.000000 51
1 0.000000 2000.000000 1000.000000 1000.000000
2 40.000000 2040.000000 1030.000000 1040.000000
3 80.000000 2080.000000 1060.000000 1080.000000
4 120.000000 2120.000000 1090.000000 1120.000000
5 160.000000 2160.000000 1120.000000 1160.000000
6 200.000000 2200.000000 1150.000000 1200.000000
...
49 1920.000000 3920.000000 2440.000000 2920.000000
50 1960.000000 3960.000000 2470.000000 2960.000000
51 2000.000000 4000.000000 2500.000000 3000.000000
40.000000 0.000000 51
1 0.000000 2000.000000 1000.000000 1000.000000
2 40.000000 2040.000000 1030.000000 1040.000000
3 80.000000 2080.000000 1060.000000 1080.000000
```

Generating Pfile

I have included two Matlab scripts in the *Material Properties* directory that are useful for generating and formatting material properties:

- `generateMaterialProps.m` creates the 3D material properties with velocities and densities that increase with depth as well as a cube of lower velocity/density
- `writeMaterialProps.m` loads these material properties and saves them into an ASCII text file with the appropriate formatting for a pfile for SW4.