

Guide to building user defined models in SHaxi

by, Michael S. Thorne and Gunnar Jahnke

Last Updated: July 26, 2007

1. User modules

To incorporate a user defined model in SHaxi we recommend adding your own modules to the code. In this section we show a simple example of how to do this. The example we show will produce a circular shaped velocity anomaly. This may not be the most exciting example, but it is an easy to understand example and demonstrates the process by which more complicated models may be built. The figure to the right displays what the model we will construct looks like.

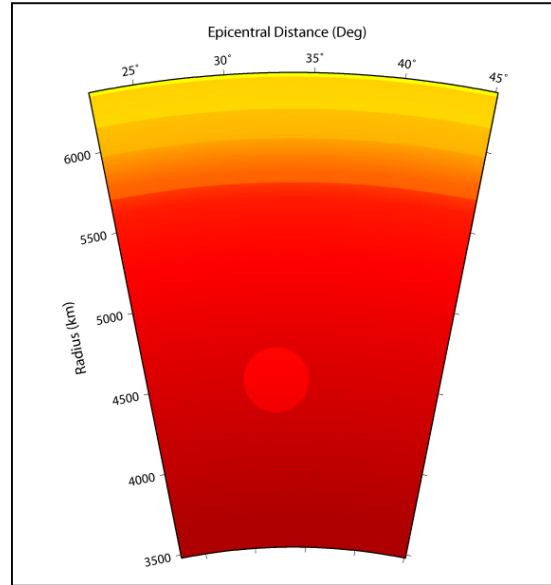


Figure 1. Example of circular velocity anomaly.

My individual user modules typically consist of three main subroutines:

- i) A subroutine setting model parameters.
- ii) A subroutine that specifies *S*-wave velocity and density (and possibly *Q*) at each grid point in SHaxi.
- iii) A subroutine that drives the whole process, to make sure every grid point is handled and that conversion to the correct units for SHaxi are applied.

1.1 Model Parameters

As an example, I will show the above three subroutines to create the circular velocity anomaly. These subroutines are provided in the SHaxi source code in the module named, `mod_circle.f90`.

The first subroutine simply specifies the various parameters about the anomaly I may want to vary. Alternatively, a subroutine could be written to read these parameters in

from a file, but this example is the easiest solution. All this subroutine does is set aside the parameters that I may want to change.

```

=====
!
! SETCIRCLE
!
! Set parameters for circular anomaly
!
=====
SUBROUTINE setcircle(Rcenter,EPcircle,Rcircle,circle_v,circle_rho)
IMPLICIT NONE
REAL, INTENT(OUT) :: Rcenter      !Radius to center of circular anomaly (km)
REAL, INTENT(OUT) :: EPcircle    !Epicentral Distance to center of anomaly (deg)
REAL, INTENT(OUT) :: Rcircle     !Radius of circular anomaly (km)
REAL, INTENT(OUT) :: circle_v    !Vs inside circle (% difference from surrounding mantle)
REAL, INTENT(OUT) :: circle_rho  !Density inside circle (% difference from mantle)

!Set parameters for circle - to be used by makecircle
Rcenter      = 4500.0
EPcircle     = 32.5
Rcircle      = 200.0
circle_v     = -5.0
circle_rho   = -2.0

END SUBROUTINE setcircle
=====

```

The parameters are shown graphically below:

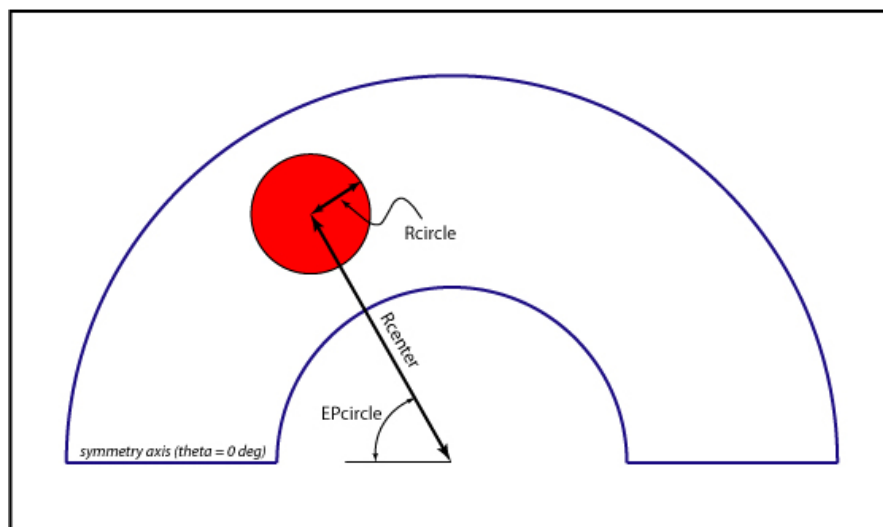


Figure 2. Example parameters for circular anomaly.

1.2 Model driver

The next subroutine drives the process of building the model. This is the subroutine that the main part of the SHaxi code would make a subroutine call to (called in the program fd2_model.f90). The driving subroutine is shown below.

```

=====
!
! CIRCLEDRIVER
!
! Driver routine to produce circular anomaly
!
=====
SUBROUTINE circledriver(rho,mu1,mu2)
USE global, only: nx, nz, r1, theta1, r2, theta2, pi
IMPLICIT NONE
REAL, DIMENSION(nx,nz), INTENT(INOUT) :: rho, mu1, mu2
REAL :: trad, ttheta, Vsin, Rhoin, Vsout, Rhoout
INTEGER :: z, x

CALL sh_prem

DO z=1,nz
  DO x=1,nx

    trad = r1(x,z)/1000.      !trad = radius in km
    ttheta = theta1(x,z)*(180.0/pi) !ttheta = theta in deg
    Vsin = mu1(x,z)          !Vs (km/sec)
    Rhoin = rho(x,z)         !Rho
    CALL makecircle(trad,ttheta,Vsin,Vsout,Rhoin,Rhoout)
    mu1(x,z) = Vsout
    rho(x,z) = Rhoout

    trad = r2(x,z)/1000.0
    ttheta = theta2(x,z)*(180.0/pi)
    Vsin = mu2(x,z)
    CALL makecircle(trad,ttheta,Vsin,Vsout,Rhoin,Rhoout)
    mu2(x,z) = Vsout

  ENDDO
ENDDO
rho=rho*1000.      ! adjust units
mu1=mu1*1000.
mu2=mu2*1000.
mu1=rho*mu1**2    ! convert vs-> mu
mu2=rho*mu2**2    ! ( vs=rho*mu^2 )

END SUBROUTINE circledriver
=====

```

This subroutine can almost be directly copied for each individual use. Before we describe what this subroutine does we should define the global variables that are used and are important for building models.

Important SHaxi Global Variables

Variable	Description
<i>nx</i>	The integer number of grid points (for the current rank) in the theta-direction.
<i>nz</i>	The integer number of grid points (for the current rank) in the radial-direction.
<i>r1</i> & <i>r2</i>	An array of size (<i>nx,nz</i>) containing the radius (units of meters) for each grid point. <i>r1</i> and <i>r2</i> contain the radii for <i>mu1</i> and <i>mu2</i> respectively.
<i>theta1</i> & <i>theta2</i>	An array of size (<i>nx,nz</i>) containing the epicentral distance (from the source – or symmetry axis) (units of radians) to each grid point. <i>theta1</i> and <i>theta2</i> contain the distance to <i>mu1</i> and <i>mu2</i> respectively.
<i>rho</i>	An array of size (<i>nx,nz</i>) containing the density (units kg/m ³) of each grid point.
<i>mu1</i>	An array of size (<i>nx,nz</i>) containing the shear modulus (kg/m·sec) of each grid point (on grid defined by <i>r1</i> , <i>theta1</i>).
<i>mu2</i>	an array of size (<i>nx,nz</i>) containing the shear modulus (kg/m·sec) of each grid point (on grid defined by <i>r2</i> , <i>theta2</i>).

The first thing this subroutine does is make a call to *sh_prem*. This initializes each grid point in the model to values given in the PREM model. A note of possible confusion is that this initial step does not actually store *rho*, *mu1*, and *mu2* in the units described above, but instead temporarily holds *S*-wave velocity (units of km/sec) in *mu1* and *mu2* and holds density (units of g/cm³) in *rho*. This step does not have to be done if your subroutine will fill every grid point on its own.

Next we loop through all grid points in the model making a call to the subroutine *makecircle*. The *makecircle* subroutine takes as input the radius (in km) and theta (deg) position, and the currently assigned *S*-wave velocity and density value of the current grid point. And it spits out a modified *S*-wave velocity and Density to the variables *Vsout* and *Rhoout*. Because of the staggered grid this must be done twice once for the SHaxi

variable *mu1* and again for *mu2*. Inside the DO loops *mu1* and *mu2* take on the values of S-wave velocity (units of km/sec) as given by the *sh_prem* subroutine. However, *mu1* and *mu2* are assigned for the shear moduli, and are thus converted to such at the end of the subroutine.

As noted, this driver subroutine can be directly copied into other user specified routines, with just a replacement of the *makecircle* subroutine.

```

=====
!
! CIRCLEDIVER
!
! Driver routine to produce circular anomaly
!
=====
SUBROUTINE circledriver(rho,mu1,mu2)
USE global, only: nx, nz, r1, theta1, r2, theta2, pi
IMPLICIT NONE
REAL, DIMENSION(nx,nz), INTENT(INOUT) :: rho, mu1, mu2
REAL :: trad, ttheta, Vsin, Rhoin, Vsout, Rhoout
INTEGER :: z, x

CALL sh_prem

DO z=1,nz
DO x=1,nx

    trad = r1(x,z)/1000.      !trad = radius in km
    ttheta = theta1(x,z)*(180.0/pi) !ttheta = theta in deg
    Vsin = mu1(x,z)          !Vs (km/sec)
    Rhoin = rho(x,z)         !Rho
    CALL makecircle(trad,ttheta,Vsin,Vsout,Rhoin,Rhoout)
    mu1(x,z) = Vsout
    rho(x,z) = Rhoout

    trad = r2(x,z)/1000.0
    ttheta = theta2(x,z)*(180.0/pi)
    Vsin = mu2(x,z)
    CALL makecircle(trad,ttheta,Vsin,Vsout,Rhoin,Rhoout)
    mu2(x,z) = Vsout

ENDDO
ENDDO
rho=rho*1000.      ! adjust units
mu1=mu1*1000.
mu2=mu2*1000.
mu1=rho*mu1**2    ! convert vs-> mu
mu2=rho*mu2**2    ! ( vs=rho*mu^2 )

END SUBROUTINE circledriver
=====

```

1.3 Specifying the velocity anomaly

Creating the velocity anomaly is done simply. In the above driving routine we looped through all grid points. Here all we do is test each grid point in the model and see if they are within the specified radius (Rcircle). If they are we change the velocity and density, if not we leave the velocity and density alone. The following subroutine accomplishes this:

```

=====
! MAKECIRCLE
! Subroutine that creates the circular anomaly
!   r      = radius of grid point (km) EXPECTS UNITS == KM!
!   theta  = theta of grid point (deg) EXPECTS UNITS == DEG!
!   Vsin   = Current Vs of grid point (km/sec)
!   Vsout  = Output velocity of grid point (km/sec)
!   Rhoin  = Current density of grid point (g/cm^3)
!   Rhoout = Output density of grid point
=====
SUBROUTINE makecircle(r,theta,Vsin,Vsout,Rhoin,Rhoout)
IMPLICIT NONE
REAL, INTENT(IN) :: r, theta, Vsin, Rhoin
REAL, INTENT(OUT) :: Vsout, Rhoout
REAL, PARAMETER :: dtr=.01745329252222
REAL :: x, y, xc, yc, d
REAL :: Rcenter, EPcircle, Rcircle, circle_v, circle_rho

CALL setcircle(Rcenter,EPcircle,Rcircle,circle_v,circle_rho)

!Convert r,theta position to x,y
x = r*sin(theta*dtr)
y = r*cos(theta*dtr)

!Find center of circle
xc = Rcenter*sin(EPcircle*dtr)
yc = Rcenter*cos(EPcircle*dtr)

!Calculate distance between current position and center of circle
d = sqrt(((xc-x)**2 + (yc-y)**2))
IF (d <= Rcircle) THEN
  Vsout = (Vsin)*(circle_v/100.0) + Vsin
  Rhoout = (Rhoin)*(circle_rho/100.0) + Rhoin
ELSE
  Vsout = Vsin
  Rhoout = Rhoin
ENDIF

END SUBROUTINE makecircle
=====

```

II. Incorporating the user modules in SHaxi

To incorporate an individual user module into SHaxi, it must be added to the program that specifies the models. This program is `fd2_model.f90`. In the above example the subroutines were added in a module named *circle*. So, a *USE circle* statement must be added to `fd2_model.f90` as shown here:

```
!=====!  
! FD_MODEL  
!=====!  
subroutine fd_model  
  USE global  
  USE mod_cart2D, only: mpi_rank,mpi_xrank  
  USE attenuation  
  USE ncoutput  
  USE circle  
  implicit none  
  
  ---- the rest of the code ----
```

Next we need to add a call to the driver subroutine and define a model number (SHaxi variable *model_type*) to the module. This can be done inside `fd2_model.f90` as shown:

```
IF (model_type==1) THEN           ! Homogeneous Model  
  
--- several lines of code ---  
  
ELSEIF (model_type == 48) THEN    ! circular anomaly  
  
  IF (attenuate == 1) THEN        ! initialize Quality factors  
    CALL Qprem(Q1,1)  
    CALL Qprem(Q2,2)  
  ENDIF  
  CALL circledriver(rho,mu1,mu2)  
  
ELSEIF (model_type == 49) THEN    ! yet another model  
  
--- more code ---  
  
ENDIF
```

III. The Makefile

In addition, the user module should be added to the makefile. Here two things must be done. First, it needs to be added to the OBJECTS variable:

```
OBJECTS= mod_sacoutput.o mod_global.o mod_circle.o mod_cart2D.o ...
```

Second, we add the lines to compile the module file itself:

```
mod_circle.o : mod_circle.f90 mod_global.f90
    $(F90) $(OPTIONS) -c mod_circle.f90
```

In this case, we add the mod_global.f90 dependency since we used several of the global variables in our module.

V. Model visualization

How do we know our subroutine's actually produced the model we expected in SHaxi? For the purpose of checking models we supply a subroutine for writing out the models in netCDF format. A model file is written for each process rank. These files have a naming convention *_model.nc.

Each netCDF file contains the parameters, ρ , $Vs1$, $Vs2$, and Q . Where $Vs1$ and $Vs2$ are the shear wave velocities calculated from the shear moduli and density. The quickest way to check the model file is to use a freeware program such as ncview (http://meteora.ucsd.edu/~pierce/ncview_home_page.html). A Ncview prompter used with a SHaxi model file is shown below. Each of the above variables can be selected in the Ncview prompter and one can quickly check if the model looks correct (although Ncview only displays the files in a Cartesian grid).

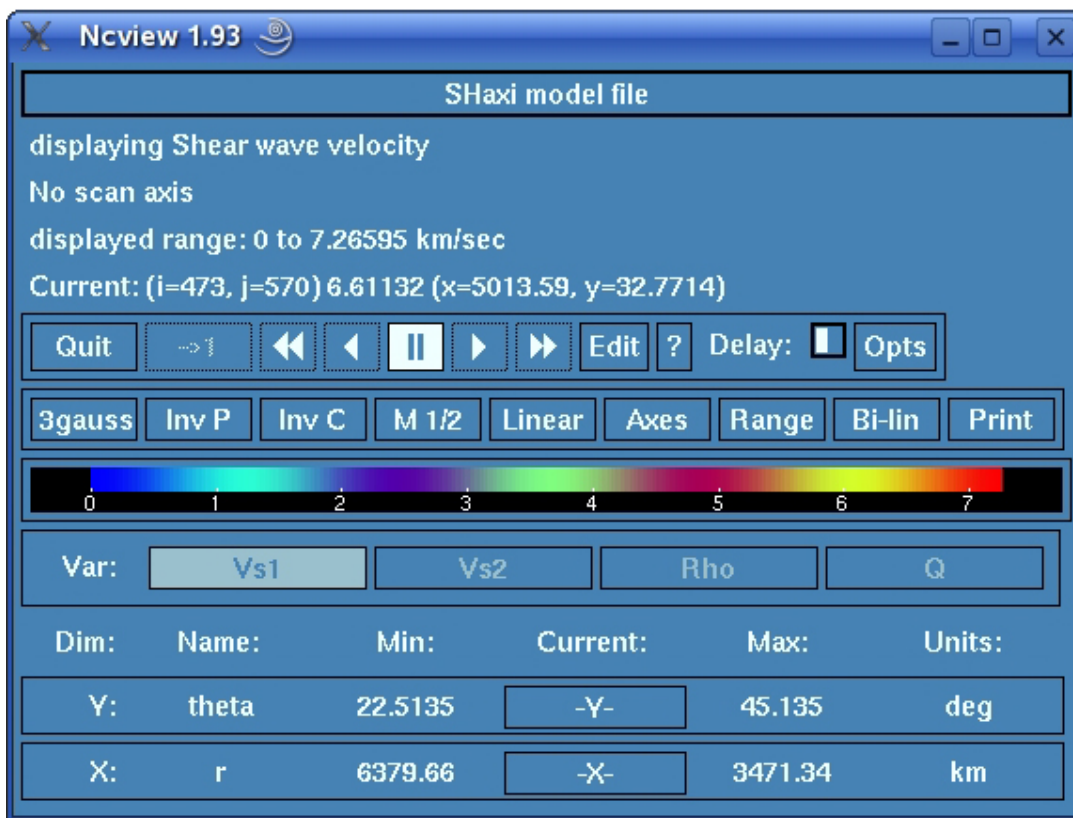


Figure 3. The Ncview prompter.

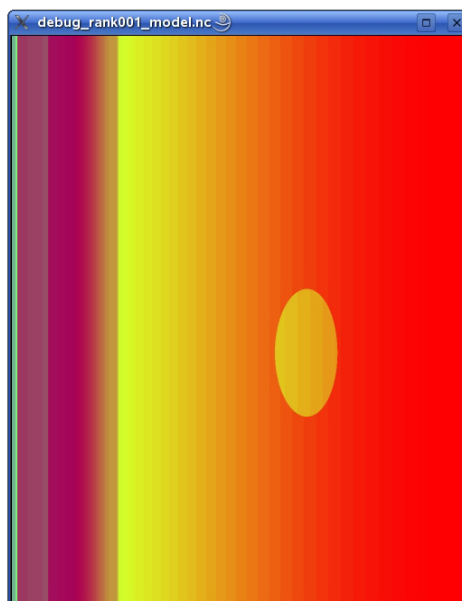
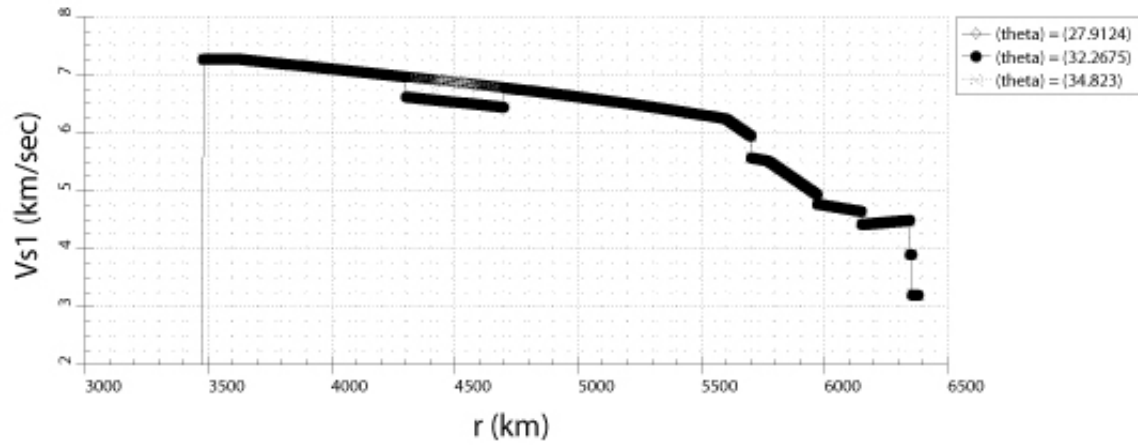


Figure 4. Ncview image of Vs1.

A good advantage to using Nview is that clicking inside the model space produces a profile. So, for example clicking in a few different places in the image displayed in Figure 4 produces an *S*-wave velocity profile that can be inspected, zoomed into, printed, etc.:



Shear wave velocity from SHaxi model file