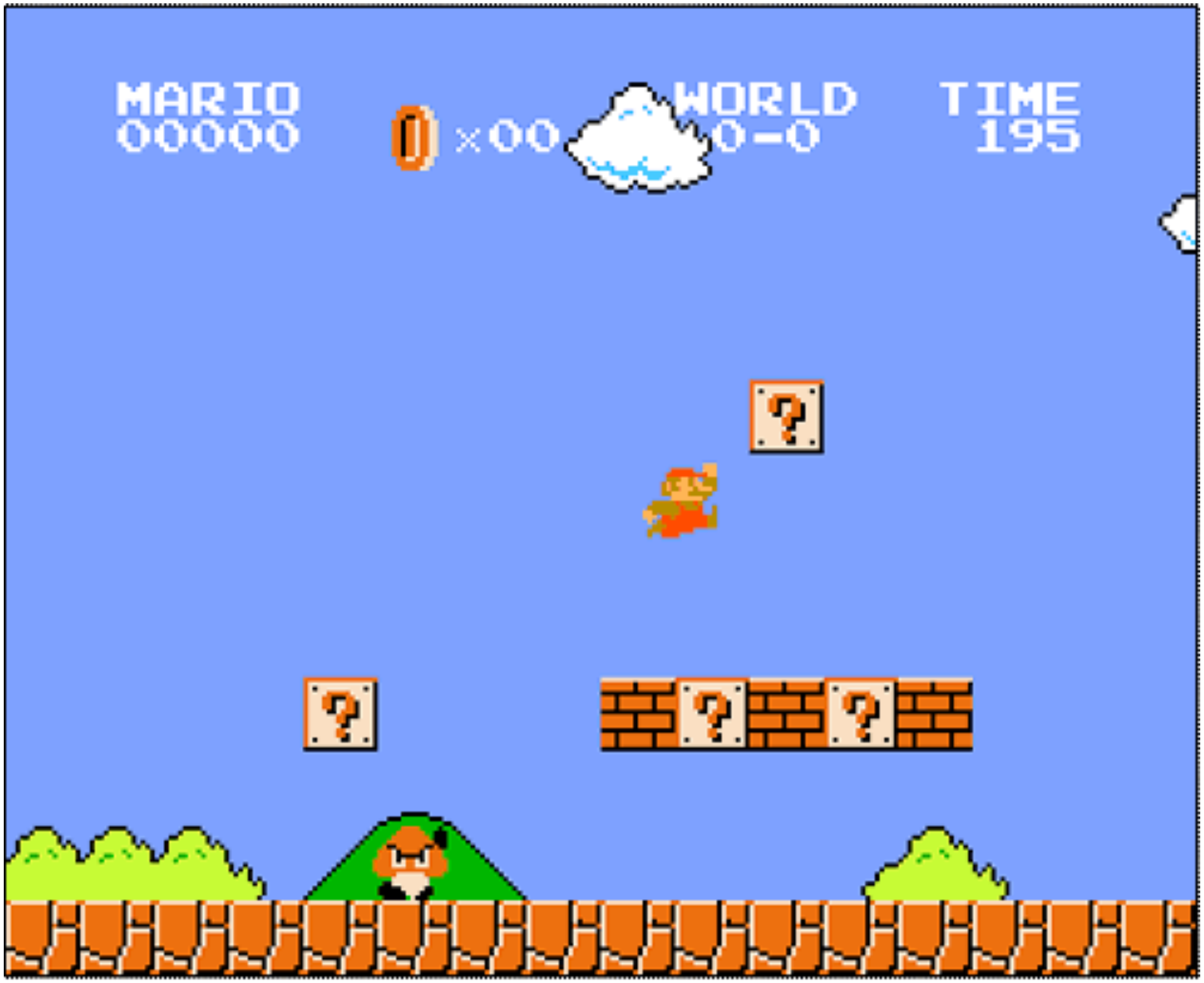
Introductory Exercises

Siwei Li

How this lecture works?



Please feel free to stop me!

Download the files

- cd RSFSRC/book/rsf/school/

- We will work in folder ray & tapprox
  or you could copy the contents into another place

- Under both folders you can find a file

  cheat

  I believe you know what it means!

Download the files

- ray:
  - Basic (everyday) SConstruct for Madagascar
  - Function calling
  - Graphic plotting

- tapprox (time-permitting):
  - Madagascar (typical) C code
  - Fundamental I/O

Ray

- Open SConstruct in your favorite editor

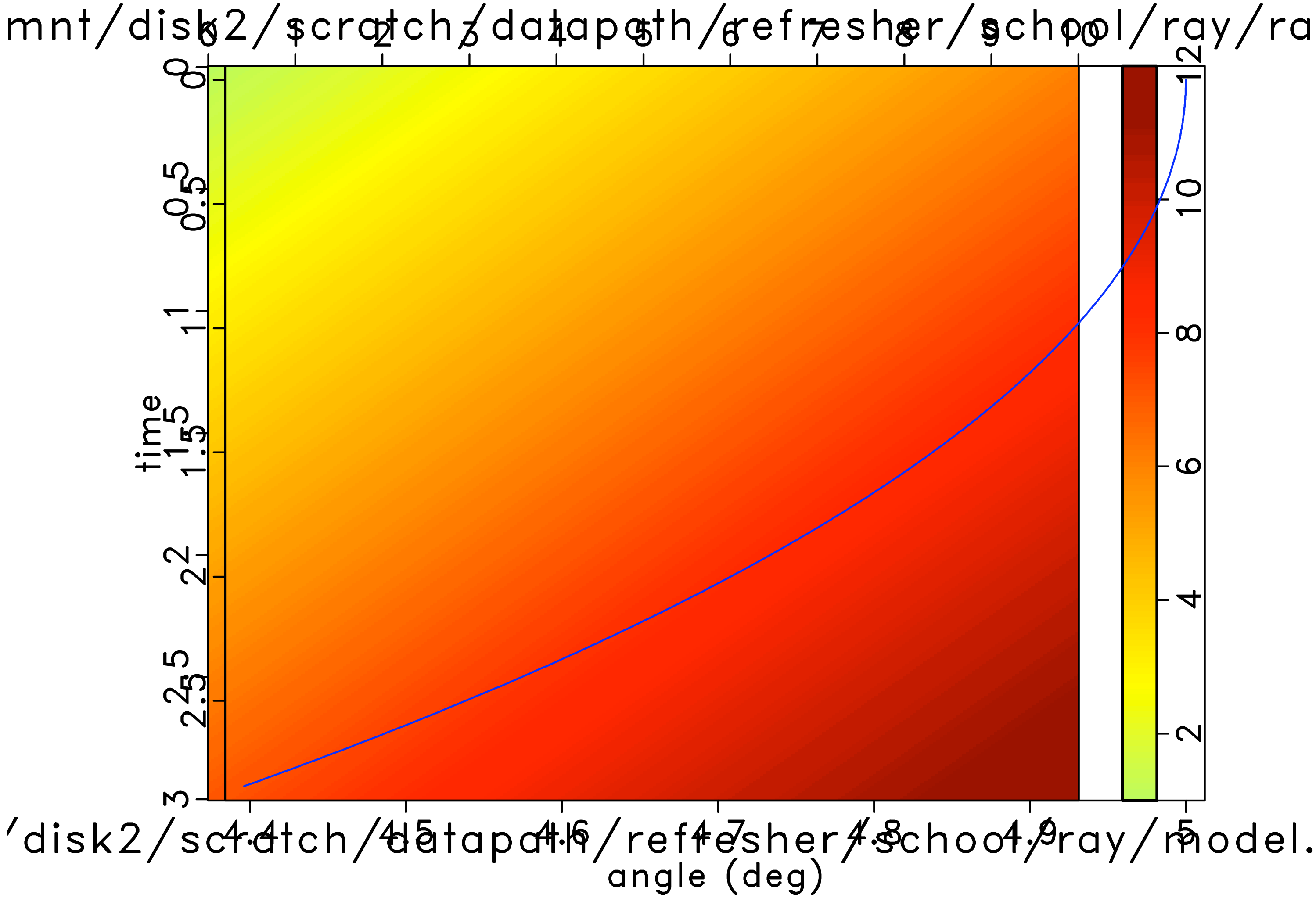
  We will go through it next line-by-line, word-by-word. ☺

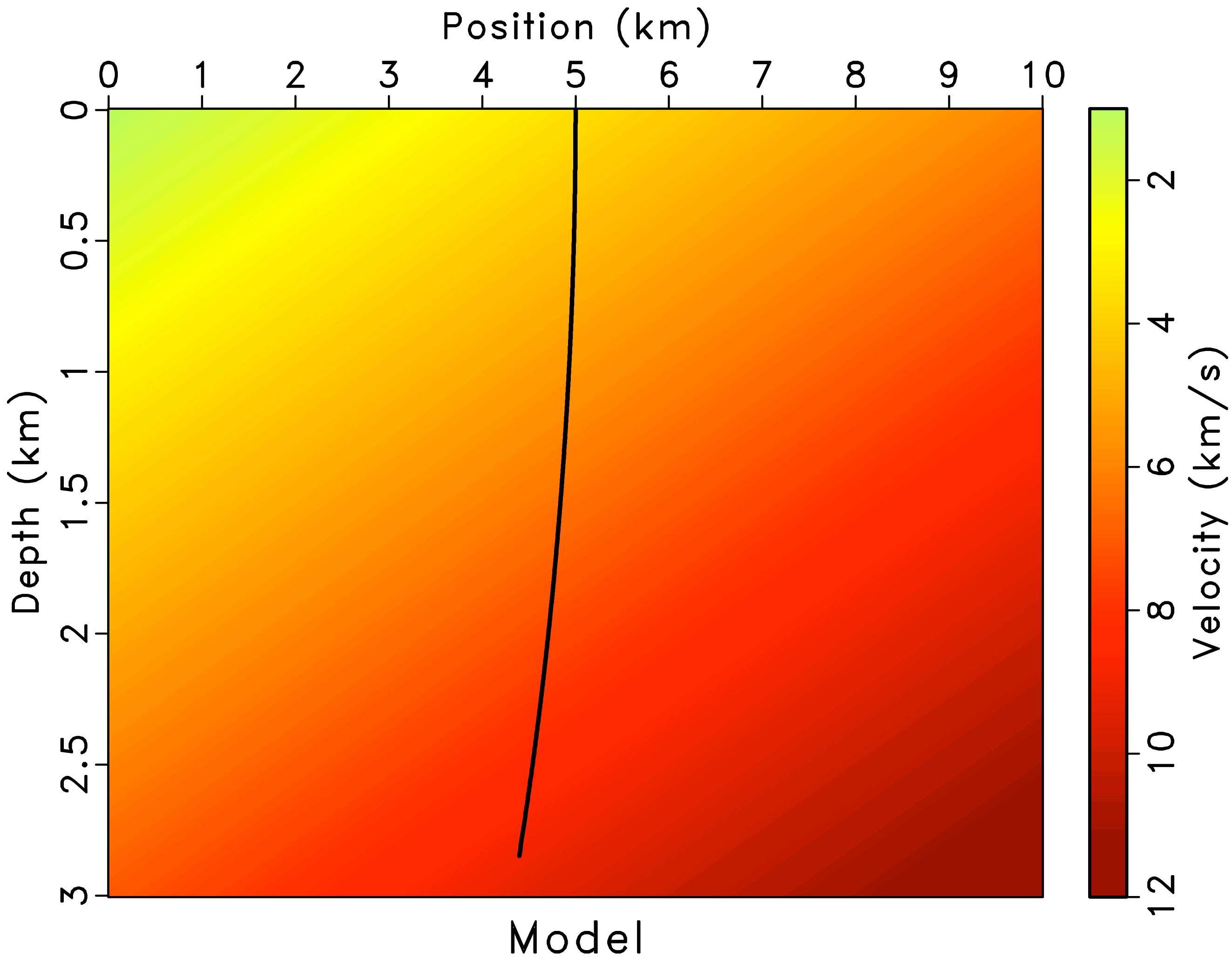
- In terminal, run

  scons view

- You should get something ugly on your screen.

  Don't worry about the terminal print-outs, we will get back to them when it is the right time.


mnt/disk2/scratch/datapath/refresher/school/ray/ra
/disk2/scratch/datapath/refresher/school/ray/model.
time
angle (deg)


Position (km)
0
1
2
3
4
5
6
7
8
9
10
Depth (km)
0
0.5
1
1.5
2
2.5
3
Velocity (km/s)
2
4
6
8
10
12
Model

Task

- The SConstruct that has been provided to you needs some modifications:

  Correct the axis label, scale bar and title of the figure.

  The ray is not overlaid on top of model in right position.

SConstruct

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Flow

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Plot

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Result

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Do-Not-Forget

```
from rsf.proj import *
```

from rsf.proj import *

```
# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

End()

Flow

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Flow(Output, Input, Function)

Flow

- In terminal, run

  scons -c

  scons model.rsf

- Read what is printed in the terminal and compare it with the SConstruct.

  /home/refresher/RSFROOT/bin/sfmath n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2" > model.rsf

- To see function documentation

  sfmath

- To check file header information

  sfin model.rsf

- The RSF file format, as its full name Regularly-Sampled-Format indicates, can be most easily understood as high-dimensional matrix. For each dimension, the axis corresponding to it can be parameterized by

  Axis origin – o#

  Axis sampling – d#

  Axis sample number – n#

- What else do you see in the file header?

Plot

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Plot(Output, Input, Function)

- To produce a visualization of the model

  scons model.vpl

- Then see it on the screen

  sfpen model.vpl

- You will learn more about VPL format in another specific lecture. The function sfpen is basically a tool to bring VPL onto your screen.
- Can you check file header of model.vpl?
- Locate where is the file.
- What we’ve done:

  None -> model.rsf -> model.vpl

- The figure for the ray must be generated by following the logical sequence of files:

  model.rsf -> ray.rsf -> ray.vpl

- Instead of running Flow and Plot one after another, try

  scons ray.vpl

  directly and see what SConstruct has done for you through terminal print-outs.

Result

```
from rsf.proj import *

# create a model
Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

# plot the model
Plot('model','model','grey color=j scalebar=y')

# do a ray-tracing
Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

# plot the ray
Plot('ray','graph transp=y yreverse=y')

# overlay model and ray
Result('overlay','model ray','Overlay')

End()
```

Result(Output, Input,Option)

Result

- In terminal, run

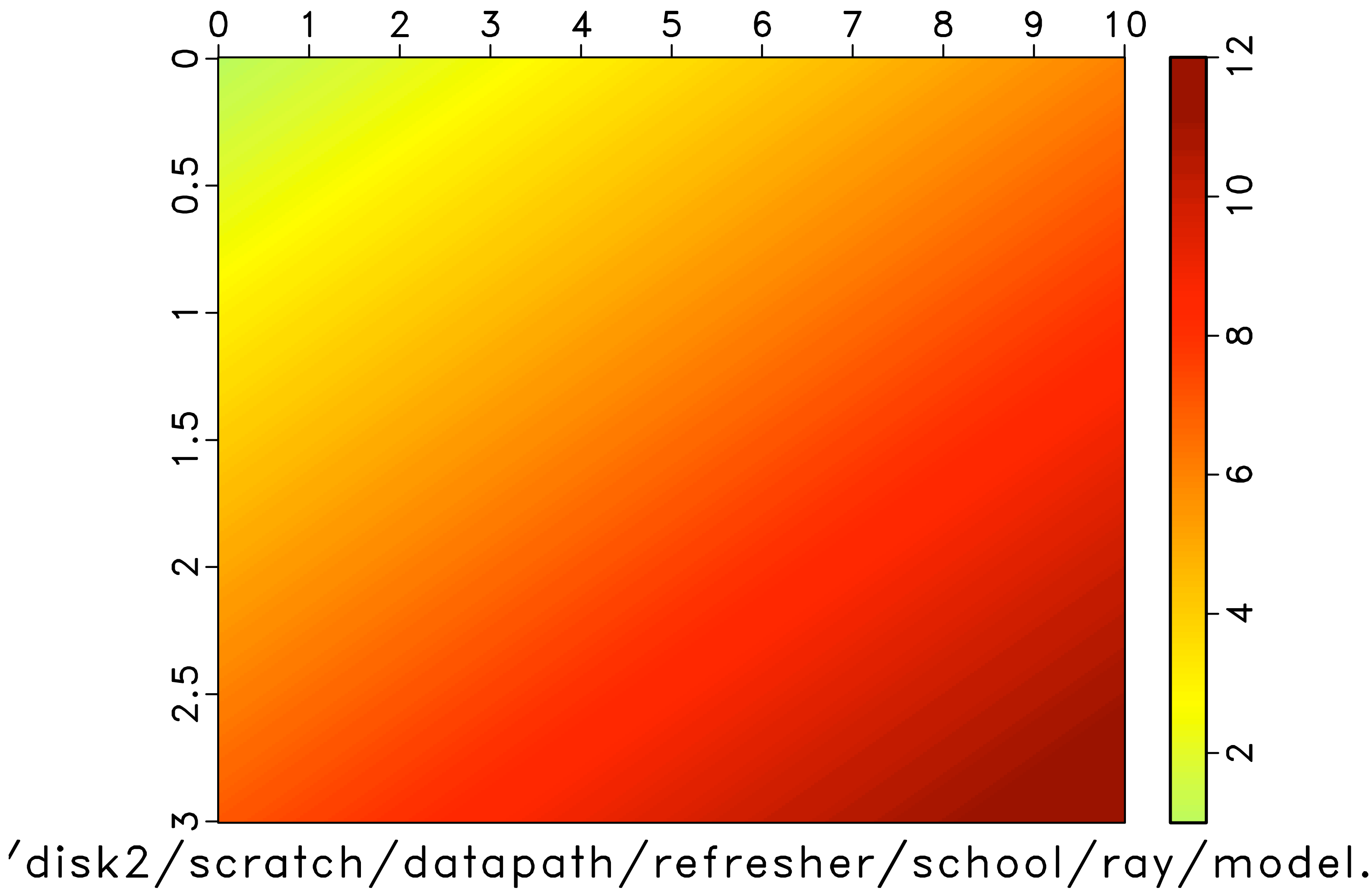
  scons overlay.view

- Find the file at

  Fig/overlay.vpl

- Now you see that both Plot and Result generate VPL files but the ones from Result are put automatically into a separate folder named Fig (why?).

Fix model.vpl

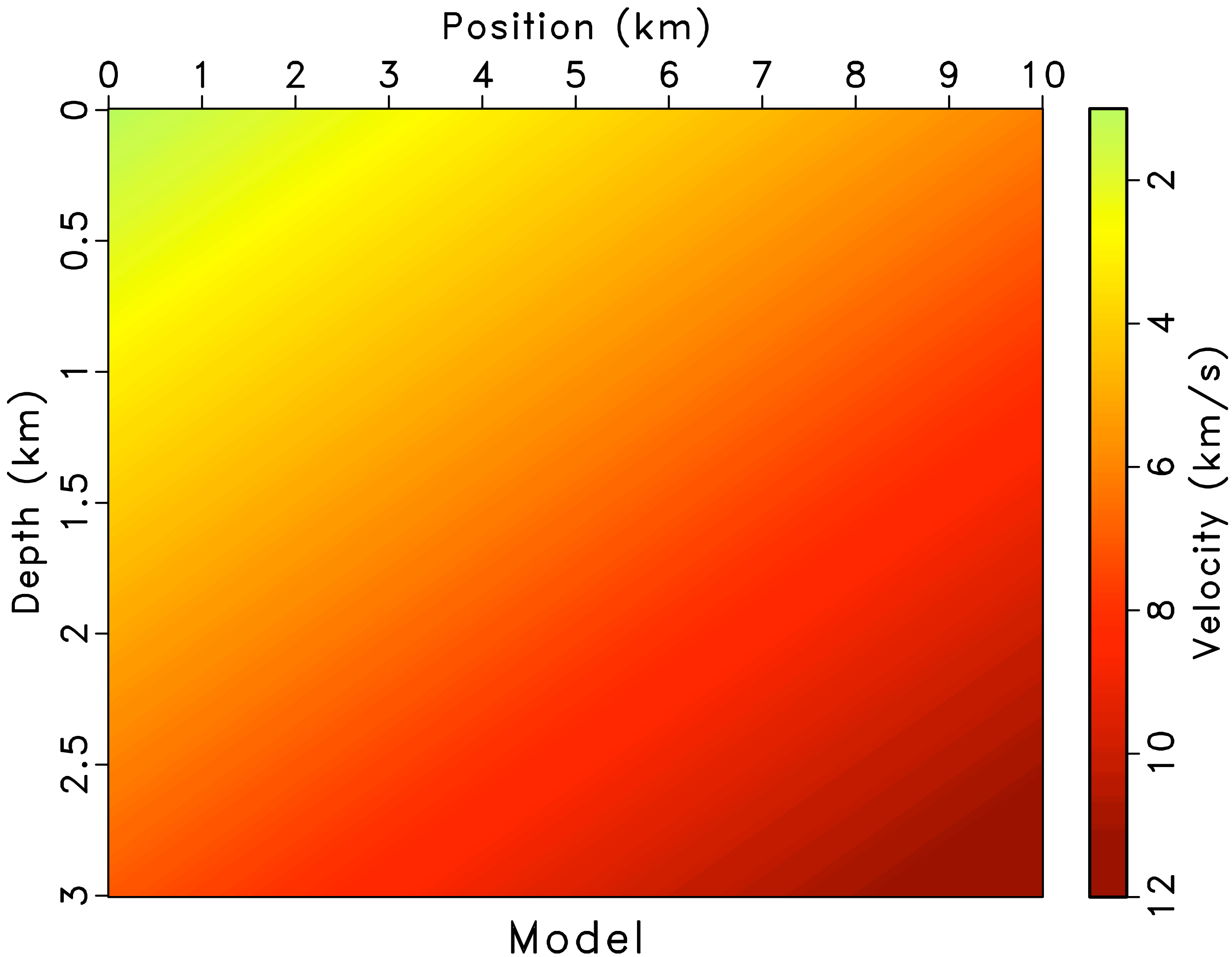


- Plot('model','model','grey color=j scalebar=y')

- Plot('mode',

```
'''
grey color=j scalebar=y
label1=Depth unit1=km label2=Position unit2=km
barlabel=Velocity barunit=km/s barreverse=y
title=Model
''')
```

Better?



Position (km)
0
1
2
3
4
5
6
7
8
9
10
Depth (km)
0
0.5
1
1.5
2
2.5
3
Velocity (km/s)
2
4
6
8
10
12
Model

Exercise 1

- You will have more options for plotting, in terminal type

  sfdoc stdplot

- Try following options:

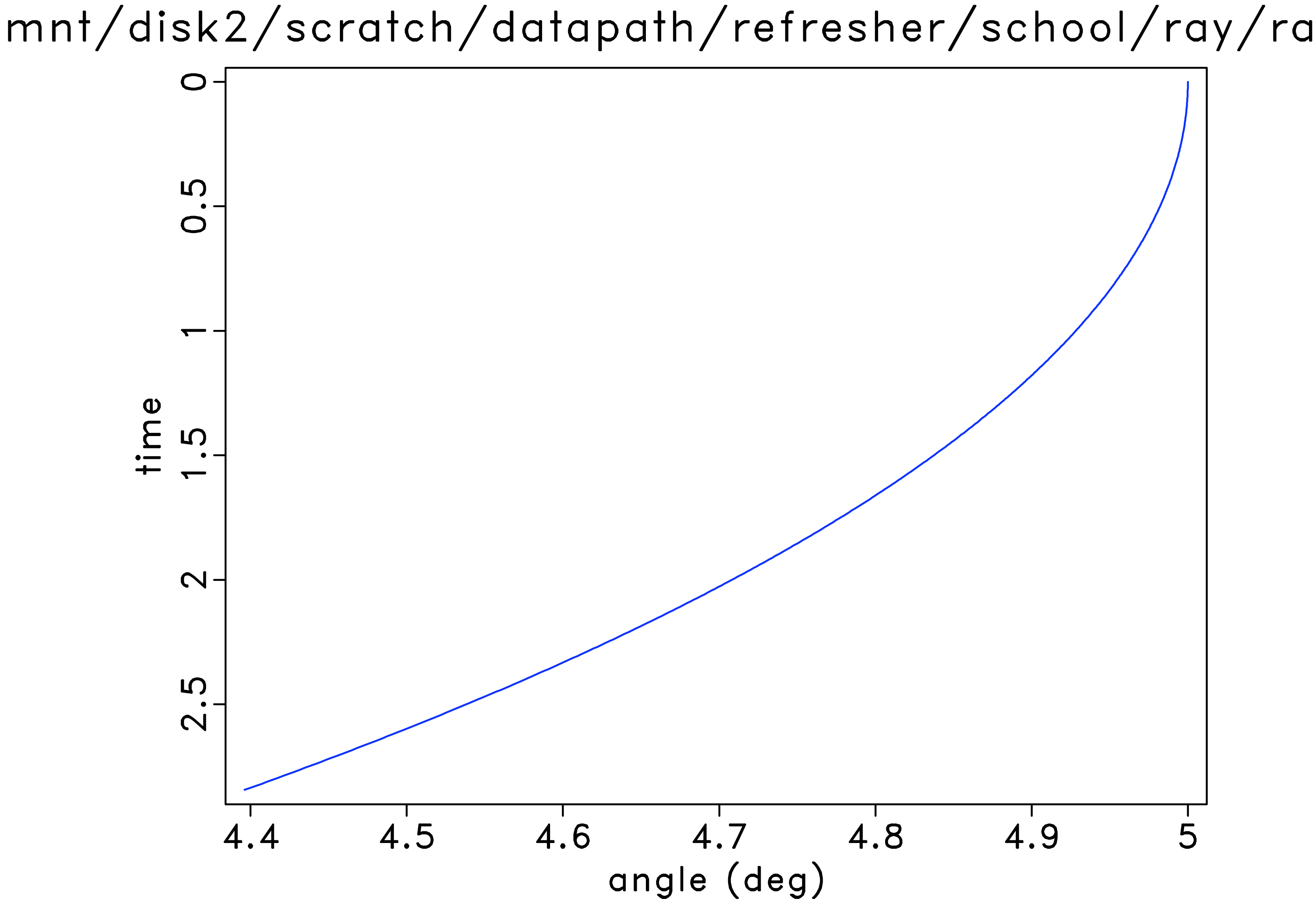
  allpos=y wanttitle=n minval=1 maxval=15

- Again, to see the figure

  scons model.vpl

  sfpen model.vpl

Fix ray.vpl



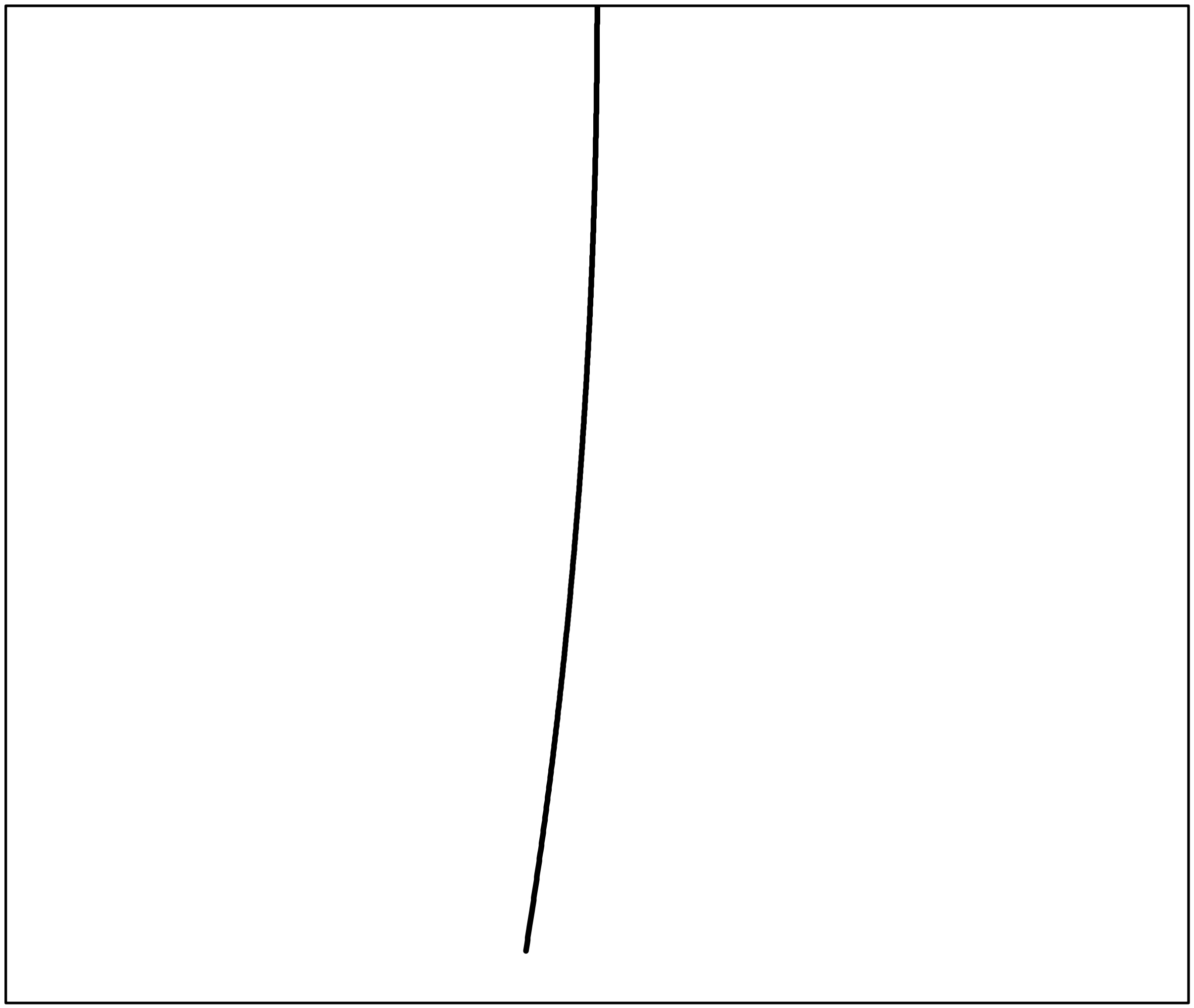
mnt/disk2/scratch/datapath/refresher/school/ray/ra
0
0.5
1
1.5
2
2.5
time
4.4
4.5
4.6
4.7
4.8
4.9
5
angle (deg)

- Plot('ray','graph transp=y yreverse=y')

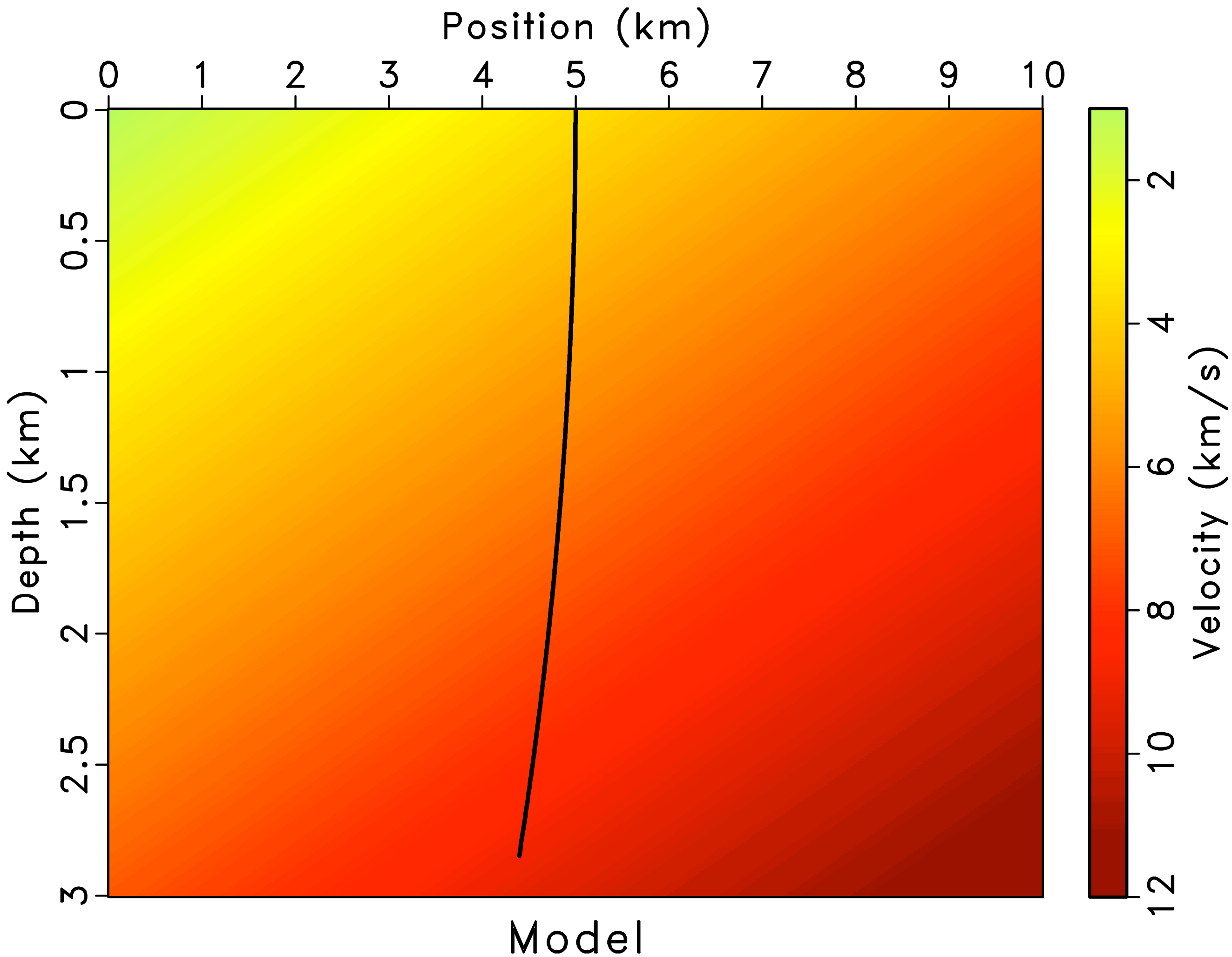
- Plot('ray',

```
'''
graph transp=y yreverse=y
min1=0 max1=3 min2=0 max2=10
wantaxis=n wanttitle=n scalebar=y
plotcol=7 plotfat=3
''')
```

Better?



Now the final figure



Exercise 2

- Result may take several other options (of course, the overlay is exact what we want here). Try replacing Overlay in Result by these options:

  OverUnderIso

  SideBySide

- Run

  scons view

  to see what changes are made.

RSF header

- Flow('model',None,'math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0 output="1+2*x1+0.5*x2"')

- Flow('model',None,
```
'''
math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0
output="1+2*x1+0.5*x2 |
put label1=Depth unit1=km label2=Position unit2=km
''')
```

- Now see header of model.rsf again.

Add a Gaussian amomaly

$$G(z,x) = amp \cdot \exp\left[-\frac{(z-z_0)^2 + (x-x_0)^2}{rad \cdot rad}\right]$$

- Flow('model',None,

```
'''
math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0
output="1+2*x1+0.5*x2 |
math output="intput+%g*exp(-((x1-1.5)*(x1-1.5)+(x2-5)*(x2-5))/(%g*%g))" |
put label1=Depth unit1=km label2=Position unit2=km
''' % (amp,rad,rad))
```

- We use Python variables. This allows us to change parameters easier and opens possibility of defining Python functions that call Flow, Plot and Result.

- Pay attention to the way variable ‘substitution’ is done

  %g – for float

  %d – for integer

  %s – for string

- The number of variables and their sequence must match!

Define Python variable

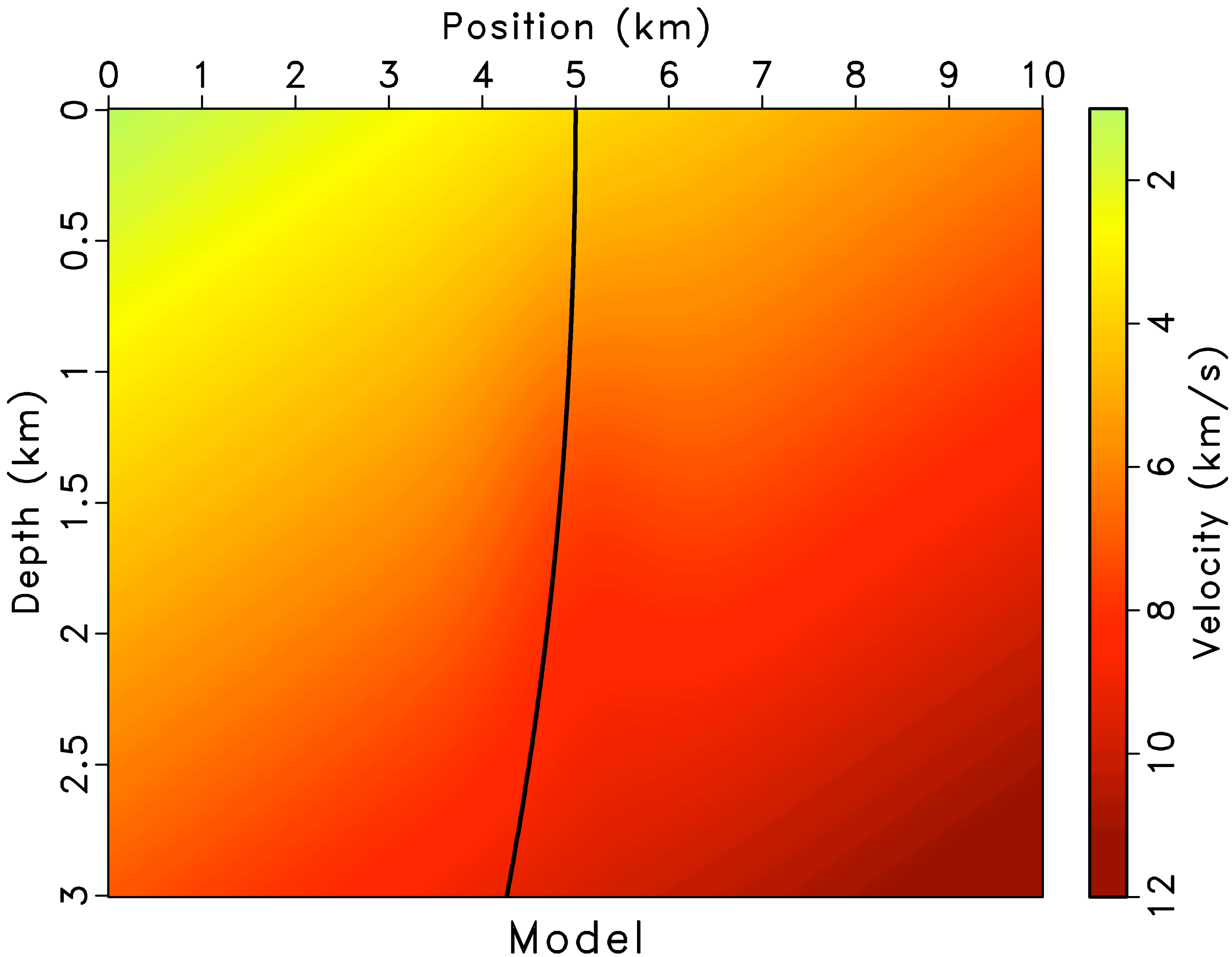
- # create a model

  amp = 1 # amplitude of Gaussian anomaly

  rad = 1 # radius of Gaussian anomaly

```
Flow('model',None,
     '''
     math n1=301 d1=0.01 o1=0 n2=1001 d2=0.01 o2=0
     output="1+2*x1+0.5*x2 |
     math output="intput+%g*exp(-((x1-1.5)*(x1-1.5)+(x2-
     5)*(x2-5))/(%g*%g))" |
     put label1=Depth unit1=km label2=Position unit2=km
     ''' % (amp,rad,rad))
```

Anything changed?



Exercise 3

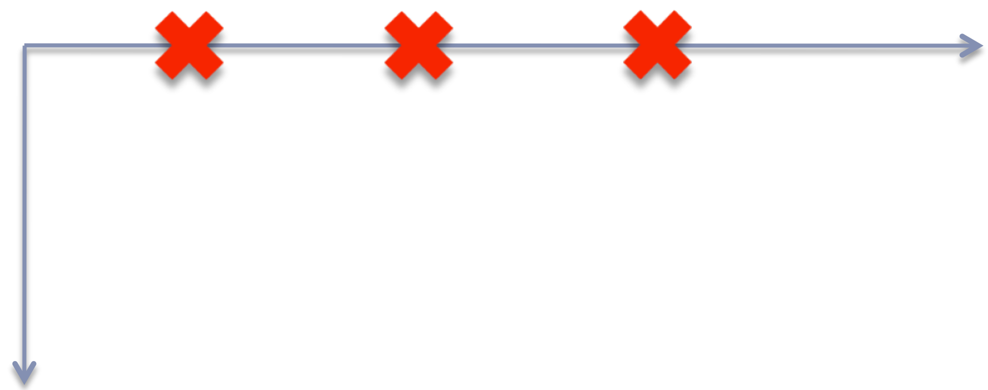
- How do you shoot from one shot location multiple rays (i.e. with different angles)?
- Read function documentation of sfrays2 by typing in the terminal

```
sfrays2
```

- And modify the following Flow:

  Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

Add rays and shots



- Flow('yshot',None,'math n1=3 d1=2.5 o1=2.5 output=x1')
  Flow('zshot','yshot','math output=0.')
  Flow('shots','zshot yshot','cat axis=2 ${SOURCES[1]} | transp')
- Check result with sfin and sfattr

- The function sfcat is used to concatenate RSF files. Function sftransp will transpose the dimensions of RSF file. See documentation by

  sfcat / sftransp

- The expression ${SOURCES[1]} will refer to the second file in input file list, i.e. yshot.rsf in this case.

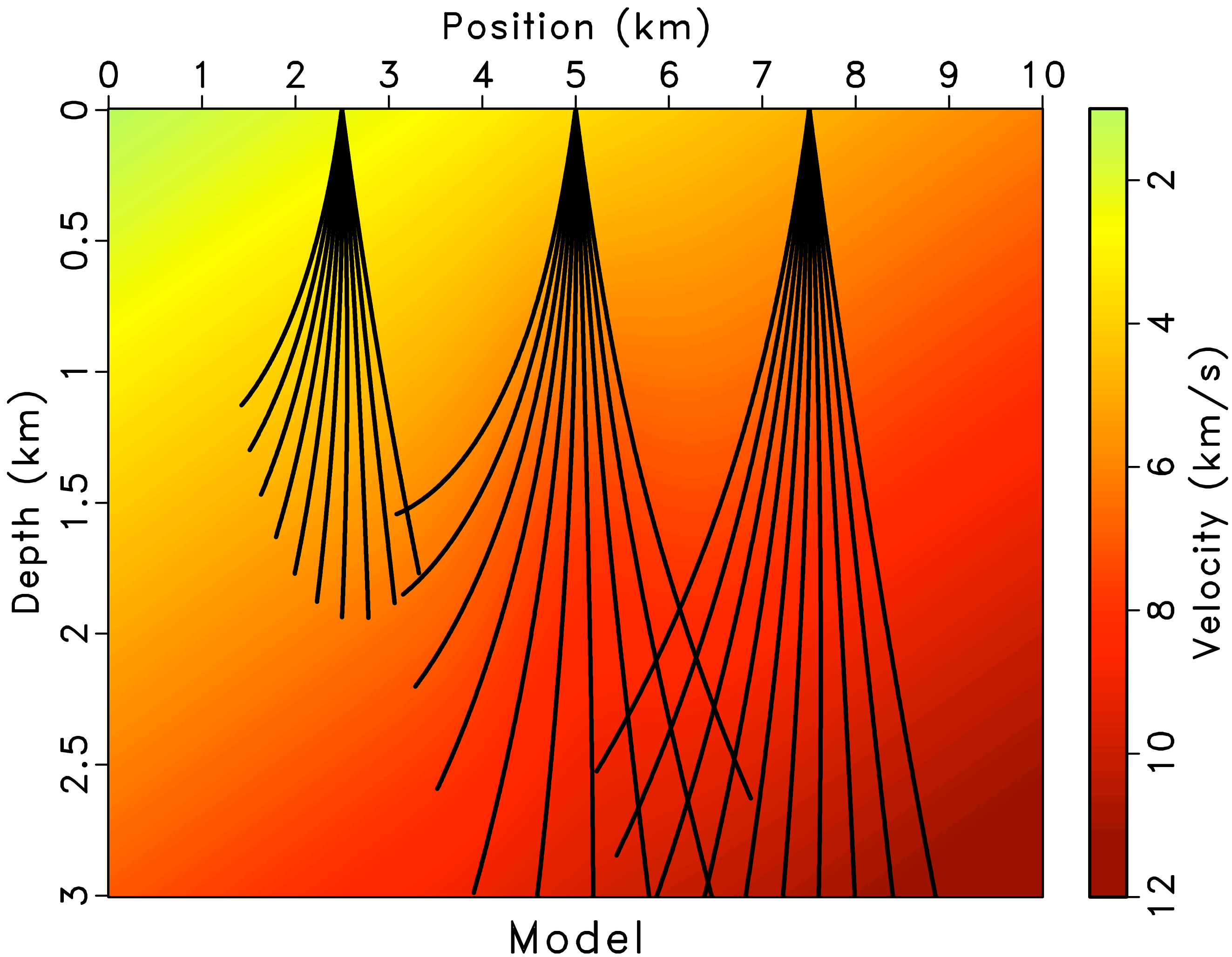
- ${SOURCES[…]} is in general the way of handling multiple input files in a Flow calling of Madagascar functions.

- Flow('ray','model','rays2 yshot=5 nt=500 dt=0.001 a0=180 nr=1')

- Flow('ray','model shots','rays2 nt=500 dt=0.001 a0=160 amax=200 nr=10 shotfile=${SOURCES[1]}')

- See sfrays2 for information regarding the function

As expected?



Position (km)
0
1
2
3
4
5
6
7
8
9
10
Depth (km)
0
0.5
1
1.5
2
2.5
3
Velocity (km/s)
2
4
6
8
10
12
Model

Separate shots

- Check dimensions of ray.rsf

  sfin ray.rsf

- For each shot

```
Plot('ray?', 'ray',
      '''
      window n3=1 f3=? |
      ...
      ''' % ?)
```

- The function sfwindow is to be used for selecting specific shot out of ray.rsf.
- It assumes a C fashion of numbering: the first element is at sampling #0, the second at #1 and so on…
- We could for sure write three different Flow for the shots, but this could be both tedious and create a hidden versatility trouble. So what can we do for better?

- # plot the ray (overlay model)
- for n in range (3):

```
    Plot('ray'+str(n),'ray',
        '''
        window n3=1 f3=%d |
        graph transp=y yreverse=y scalebar=y plotcol=7
        plotfat=3 wantaxis=n wanttitle=n min1=0 max1=3
        min2=0 max2=10
        ''' % n)

    Plot('overlay'+str(n),['model','ray'+str(n)],'Overlay')
```

- The clause

  for n in range (3):

  is from Python. If it is the first time you see it, a good way to understand is that it is equal to (in C):

  for (n=0; n < 3; n++) {}

- Another thing (a bit crazy): Python use TAB line-up to identify loops, and there is no {}!

- Same with str(n), it is also a Python thing.

- Also notice how I separate the input in Flow from within one " to multiple "s.

  ['model','ray'+str(n)]

- What about multiple outputs?
  What is the counterpart of ${SOURCES[…]}?

- There are many more Python functions and modulus that can be fully incorporated into a Madagascar SConstruct script.

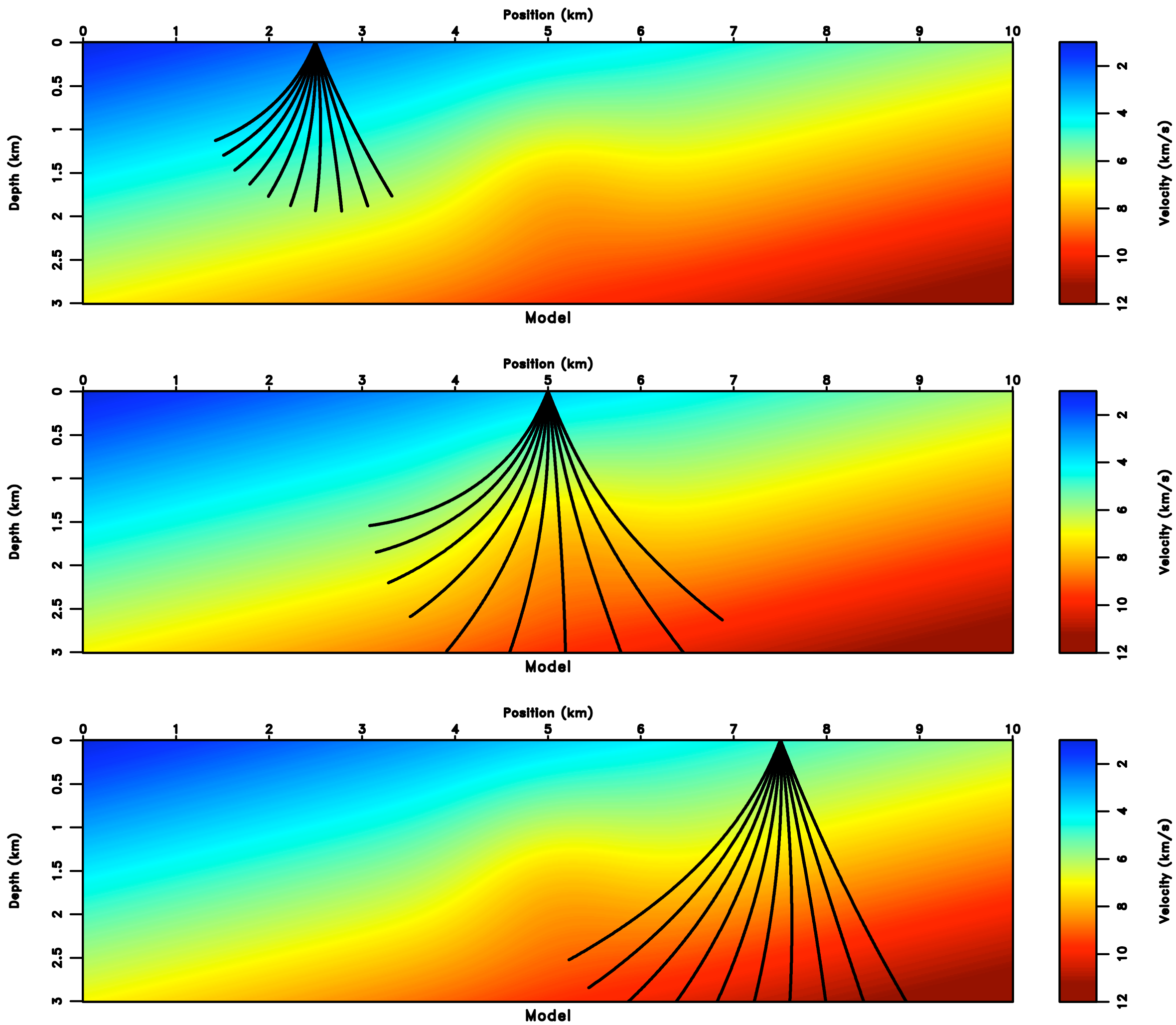
```
def Draw(rsffile,extra=''):
    Plot(rsffile,'grey color=j scalebar=y barreverse=y allpos=y
   %s' % extra)
```

- # plot the model

  Draw('model','title=Model barlabel=Velocity barunit=km/s')

- Result('figure','overlay0 overlay1 overlay2','OverUnderAniso')

Final figure



Exercise 4

- Modify your SConstruct:

The shot (number and location) will be defined through Python variables.

Use different color for different shots.

Define a Python function that takes shot variables and do the ray tracing, similar to what I show you in Draw.

Tapprox

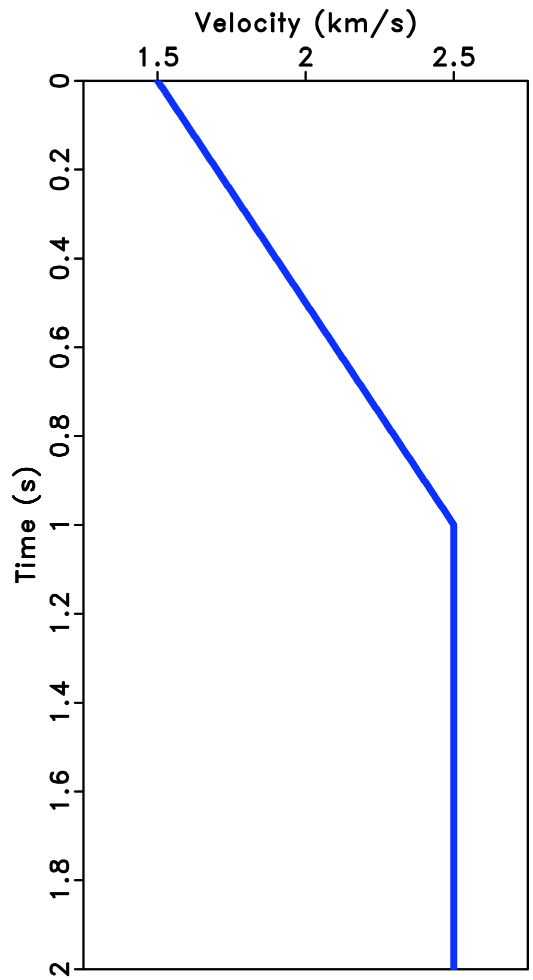
- Open Sconstruct and traveltime.c in your favorite editor

  We will deal with the C code in this exercise.

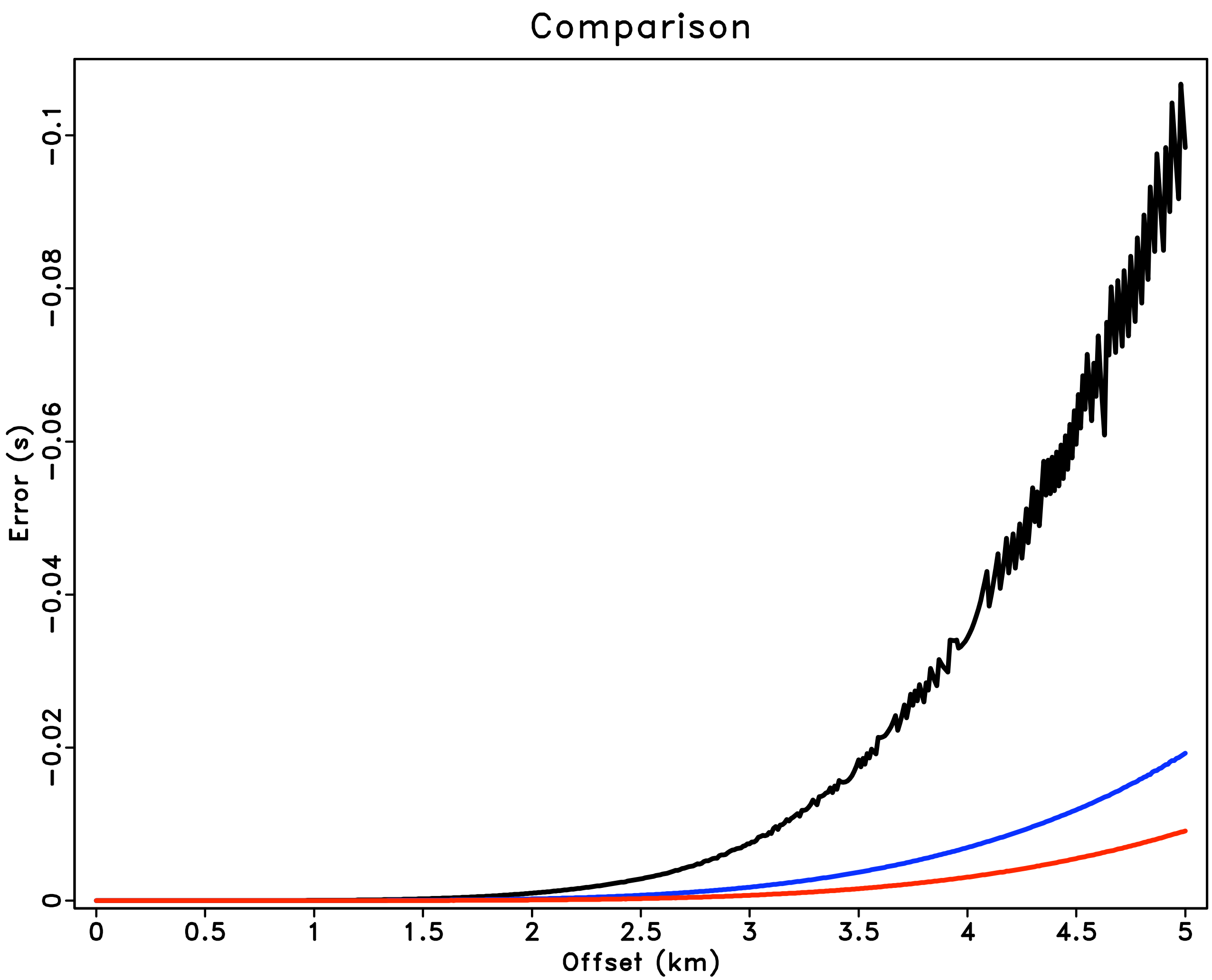
- In terminal, run

  scons view

- You should get two pictures of curves on your screen.

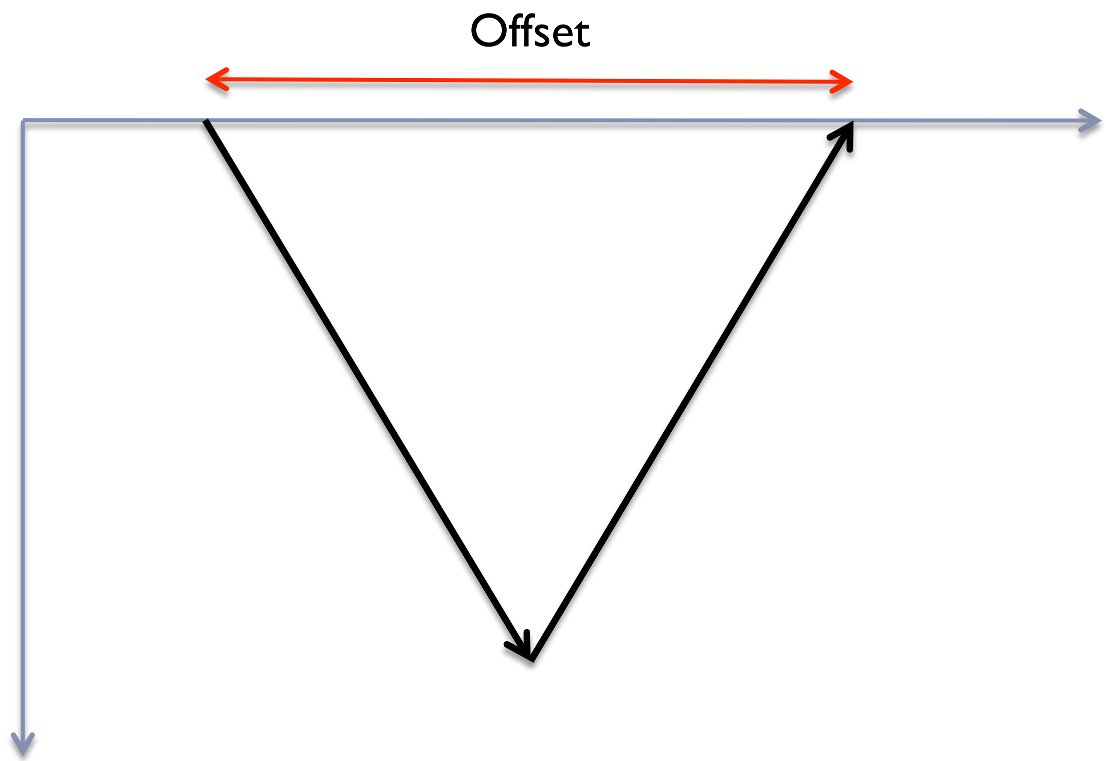


Interval Velocity


Comparison
Error (s)
0
-0.02
-0.04
-0.06
-0.08
-0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Offset (km)

What are the curves about?

- In a perfectly layered medium, i.e. v = v(z):



Task

- Modify the source code traveltime.c (and SConstruct) so that it also outputs a temporary variable used during traveltime computation.

- Since this is probably the first time you are exposed to a Madagascar program, it must be somehow unfamiliar (it does not fit your impression on C). Thus we will spend more time surfing around the most apparent features.

- The task itself (as you will see) is in fact very simple. But dealing with it should help you understand the I/O of Madagascar.

SConstruct

```
from rsf.proj import *

# compile program
# v-of-z model
# reflector file

# compute traveltime
for case in 'ae':
    Flow('time_'+case,['vofz',exe],
        '''
        ./${SOURCES[1]} nr=3 r=125,250,375
        nh=501 dh=0.01 h0=0 type=%c
        ''' % case)

# compute error
# plot

End()
```

traveltime.c

traveltime.c

- Madagascar

  #include <rsf.h>

- Look around for variables and functions that start with

  sf_

- For example:

  sf_file, sf_putint, sf_floatread, sf_floatwrite, sf_intalloc…

traveltime.c

1
- Define variables, I/O preparation

2
- Allocate memory, read input, do the calculation

3
- Output, clean up

Read parameter from file header

```
/* initialize */
sf_init(argc,argv);

/* input and output */
vel = sf_input("in");

/* time axis from input */
if (!sf_histint(vel,"n1",&nt)) sf_error("No n1=");
if (!sf_histfloat(vel,"d1",&dt)) sf_error("No d1=");
```

```
Flow('time_'+case,['vofz',exe],
''' ./${SOURCES[1]} nr=3
r=125,250,375 nh=501
dh=0.01 h0=0 type=%c''' %
case)
```

Read parameter from command line

```
/* initialize */
sf_init(argc,argv);

/* offset axis from command line */
if (!sf_getint("nh",&nh)) nh=1;
/* number of offsets */
if (!sf_getfloat("dh",&dh)) dh=0.01;
/* offset sampling */
if (!sf_getfloat("h0",&h0)) h0=0.0;
/* first offset */
```

```
Flow('time_'+case,['vofz',exe],
''' ./${SOURCES[1]} nr=3
r=125,250,375 nh=501
dh=0.01 h0=0 type=%c''' %
case)
```

Change traveltime.c

- ```
  /* reflector axis from command line */
  if (!sf_getint("nr",&nr)) nr=1;
  /* number of reflectors */
  r = sf_intalloc(nr);
  if (!sf_getints("r",r,nr)) sf_error("Need r=");
  ```

- ```
  sf_file ref;
  ref = sf_input("ref");
  if (!sf_histint(ref,"n1",&nr)) sf_error("No nr=");

  r = sf_intalloc(nr);
  sf_intread(r,nr,ref);
  ```

Change SConstruct

- Flow('time_'+case,['vofz',exe],

```
'''
./${SOURCES[1]} nr=3 r=125,250,375
nh=501 dh=0.01 h0=0 type=%c
''' % case)
```

- Flow('time_'+case,['vofz',exe,refl],

```
'''
./${SOURCES[1]} ref=${SOURCES[2]}
nh=501 dh=0.01 h0=0 type=%c
''' % case)
```

Add extra output for variable p

```
……
case 'a': /* accelerated RMS approximation */
    ……
    p = -1/(4*t2)+va/(4*t2*v2*v2);
    ……
    break;

case 'e': /* exact */
    for (iter=0; iter < niter; iter++) {
            ……
            p = p-(hp-h)/ghp;
            ……
    }
    ……

    break;
……
```

Change traveltime.c

- sf_file ref, par;

  par = sf_output("par");

  sf_putint(par,"n1",nr);
  sf_putfloat(par,"o1",1.);
  sf_putfloat(par,"d1",1.);
  sf_putstring(par,"label1","Reflector");
  sf_putstring(par,"unit1","number");

- sf_floatwrite(&p,1,par);

Change SConstruct

- ```
  Flow('time_'+case,['vofz',exe,refl],
           '''
           ./${SOURCES[1]} ref=${SOURCES[2]}
           nh=501 dh=0.01 h0=0 type=%c
           ''' % case)
  ```

- ```
  Flow(['time_'+case, 'par_'+case],['vofz',exe,refl],
           '''
           ./${SOURCES[1]} ref=${SOURCES[2]}
           nh=501 dh=0.01 h0=0 type=%c par=${TARGETS[1]}
           ''' % case)
  ```

Congratulations!