

GPR data processing and visualization with GPR-O

Alain Plattner (`plattner@alumni.ethz.ch`)

January 20, 2017

Contents

1	General remarks	2
2	Data structure and data preparation	2
3	Data preprocessing and data loading	4
4	Profile data plotting and processing	5
5	Area plotting	9
6	CMP and WARR	11
7	Topography	17
8	Export to VTK	18

1 General remarks

You can download Octave for free from <https://www.gnu.org/software/octave/>

In the following, whenever I say “run xxxx” I mean in Octave or Matlab, type xxxx and press enter.

Always before you use an Octave program for example by the name xyz, run “help xyz” without the quotation marks to read about what input is required and what the output will be.

For all GPR-O programs to run, your current directory (shown at the top of your Octave program window) must be the GPR-O folder.

In the command window you can recall previously entered commands by pressing the up and down keys and you can let the computer suggest completions of commands with the tab key.

2 Data structure and data preparation

To use GPR-O, switch to the GPR-O folder on your computer. This folder contains a few subfolders and a few m-files. Each m-file is a computer program that was written for Octave (or Matlab; they are for our programs here interchangeable).

If you are using Matlab, to access to all the subprograms you need to run

```
initialize
```

If you are using Octave, run

```
initialize_octave
```

instead. The first time you run this initialization, it will check if all required Octave packages are already installed and if not, will automatically install them. This may take a few minutes. Next time you run it, it will not need to install the packages anymore.

We will use a variety of programs that are stored in the subfolders. Whenever you want to know what a program does and what variables you need to give it, run “help programname”. For example, we have a program that is named “readdata”. To find out how to use it, run

```
help readdata
```

Octave will show you a text saying from where function is stored, what input it requires, and what output it gives.

For the required input, the help page says:

surveyparams A struct with the following fields:

- **nlines** The highest line number
- **lineincr** The distance between the lines (in meters)
- **pnameraw** String containing the directory of the raw files
- **pnametrif** String containing the directory for the processed files

This variable named **surveyparams** will play a key role in GPR-O. For each survey you will need to set it up. This is easiest done by writing your own m-file as described in the following.

Double-click on “Example1.m”. The command window will disappear and a new window named “Editor” will appear.

you will see the first line saying “surveyparams.nlines=24;” and then a percent sign and then some comment. This means that the data set that we will be working with has 25 parallel lines that are numbered 0, 1, 2, ..., 24.

The next line says “surveyparams.lineincr=0.2;”. This means that the distance between the lines is 0.2 m.

The following line says “surveyparams.pnameraw='data/raw/example1/';”. This means that the data that we recorded with the GPR is stored in the folder “data/raw/example1/”. Open that folder on your desktop and you will see that it contains several files that are named “XLINE00.DT1” etc, and “XLINE00.HD” etc. These are example data files that were downloaded from the GPR and we will be working with these as an example. For your own data files you will create a folder besides the “exercise1” etc folders in the data directory and store your GPR files in there.

The last line in “example1.m” says “surveyparams.pnametrif='data/processed/example1/';”. This means that the processed data will be stored in the folder “data/processed/example1/”. If you open, on your desktop, the folder “data/processed”, then you will see that it is empty. We need to create this folder to continue. *If you don't create this folder the program will create an error when you try to pre-process the data. So make a folder “example1” in the directory “processed”.*

For your own data you will need to create your own m-file and your own subfolders in “processed” and “raw”. Use this example as an instruction on how to do this.

3 Data preprocessing and data loading

To get back to the Command Window when the Editor is visible, click on the tab at the bottom of the Octave program window.

Once you have set up the folder structure and the m-file for your data, you can, back in the Command Window, set the “surveyparam” variable by running

```
example1
```

or whatever the name of your m-file is. You can display the surveyparams variable by running

```
surveyparams
```

This should show you all the information you previously put into your m-file. Now run

```
preprawdata(surveyparams)
```

To preprocess your data. If it doesn’t work it is likely that you made one of the following mistakes:

- You did not enter the correct folder for your raw data in your m-file
- you did not enter the correct folder for your processed data in your m-file, or you did not create that folder
- when collecting the GPR data, you did not select “record grid data” and then “X-lines”. You can see this by looking at your raw data folder. If the files say “YLINE00.dt1” etc, or just “LINE00.dt1” etc, then that’s what happened. No worries, in that case you can just run `preprawdata(surveyparams,1)` if the names are “YLINE00.dt1”, or `preprawdata(surveyparams,2)` if the names are “LINE00.dt1”.

After successfully preprocessing the data, you can load it by running

```
data=readdata(surveyparams);
```

The semicolon at the end is necessary to avoid that your screen gets filled up with numbers. If that happens you can press “ctrl-c” to abort and run it again with the semicolon.

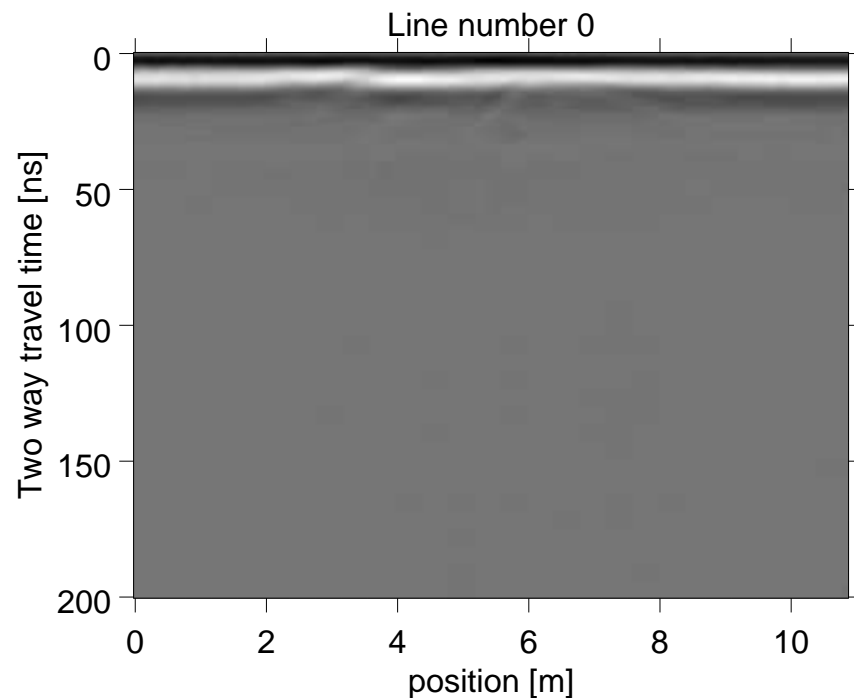
Your data is now loaded and stored in the variable “data”.

4 Profile data plotting and processing

Let's have a look at the data you just loaded. Remember that we have 25 parallel lines that are numbered from 0 to 24. We can plot the first line (number 0) by running

```
plotGPRline(data,0);
```

It should look something like



If you want to show the second line, run `plotGPRline(data,1);`, and for the third line `plotGPRline(data,2);`, etc.

You can always export a figure as pdf by clicking, in the figure window, on “File” and then “Save As”

What this figure shows is: Along the first profile, at every location, the GPR sent out a wave and recorded what came back (as a function of time). This wave is shown as a row of pixels in grayscale going from the top to the bottom. The way we obtain a full 2D image instead of just a single wave signal is that we plot all of these lines next to each other.

In this profile we did not see much else but the black and white horizontal stripes in roughly the first 12 nanoseconds. This is because the wave that excited the transmitter antenna flew directly

into the receiver antenna without going through the ground. In this setup, the antennae were 0.5 m apart and the frequency was 100 MHz. A radar wave moves with the speed of light. In air this is roughly 0.3 m/ns. A wave of this frequency has length 0.3×10^9 m/s divided by 100×10^6 1/s, which is 3 m. For the entire 3 m wave to travel the 0.5 m, it needs to travel 3.5 m. At the speed of light it takes this wave 3.5 m divided by 0.3 m/ns, which is 11.67 ns, exactly the time where we see the black and white stripe.

So we just saw that our image is dominated by the radar wave traveling through air and we don't see much from the waves traveling through ground (which are slower but also more weakened when they travel through the ground). We therefore want to remove these airwaves to see what is in the ground. Because the distance between the antennae is always the same in this profile, the airwave always looks the same at each recorded position. Therefore if we average all recorded positions, we should only get the airwave out and any other signals from the ground should cancel out. We can do this with the function `removeHorizontal`. Run

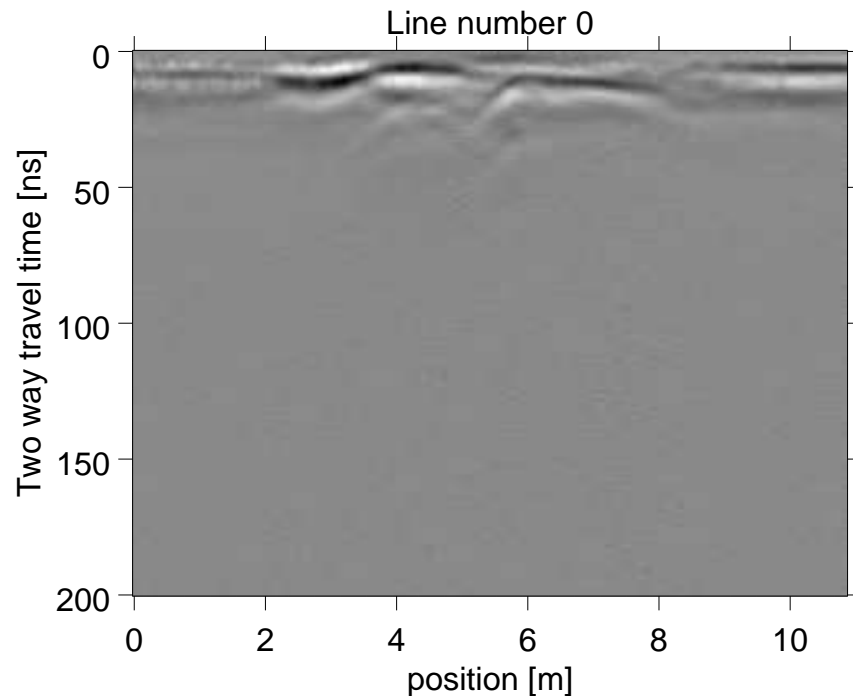
```
dataH=removeHorizontal(data,10000);
```

This creates a new variable “dataH” that contains the old data with horizontal feature removal. We needed to give the program the number of measurement locations it should use to average. By giving it a very high number, say “10000” we just average all locations per profile. The less locations you average, the more aggressively you remove horizontal features. The most extreme is if you say 1, because then you remove everything and end up with no signal at all.

Let's see how the processed data looks like. Run

```
plotGPRline(dataH,0);
```

It should look something like



Much better. We can actually see something. But everything we see is at early two-way travel times and therefore close to the surface. Why could that be?

The thing with radar waves traveling through ground (as opposed through air or space) is that they get weakened (attenuated) quite fast. That means that the deeper down the wave travels, the weaker it is, and therefore we can't really see much when it gets bounced off an interface in the subsurface and comes back to the surface.

But there is a way to make these deeper signals more visible. We can artificially strengthen later travel times by multiplying them with an increasing factor. This is called “gain” and the function `gainData` does just that. In this kind of gain we split up the signal at each position into time windows and make them all equally strong. This means that the time window close to 200 ns will be strongly amplified, while the time window close to 0 ns will not be amplified at all. Here we choose the time window 5 ns. Run

```
dataHG=gainData(dataH,5);
```

This takes a while to process. After it is finished you might want to save your processed data such that you do not need to start all over next time. You can save your data with the command

```
save('exercise1dataHG.mat','dataHG');
```

This created a new file named “exercise1dataHG.mat” in your directory. You can also give it any other name or store any other processed data this way. You can always load this data set now by double clicking on it. Try first deleting everything in your current Octave session by running

```
clear all
```

If you would now try to redo the plots from before you would get an error message. Now run

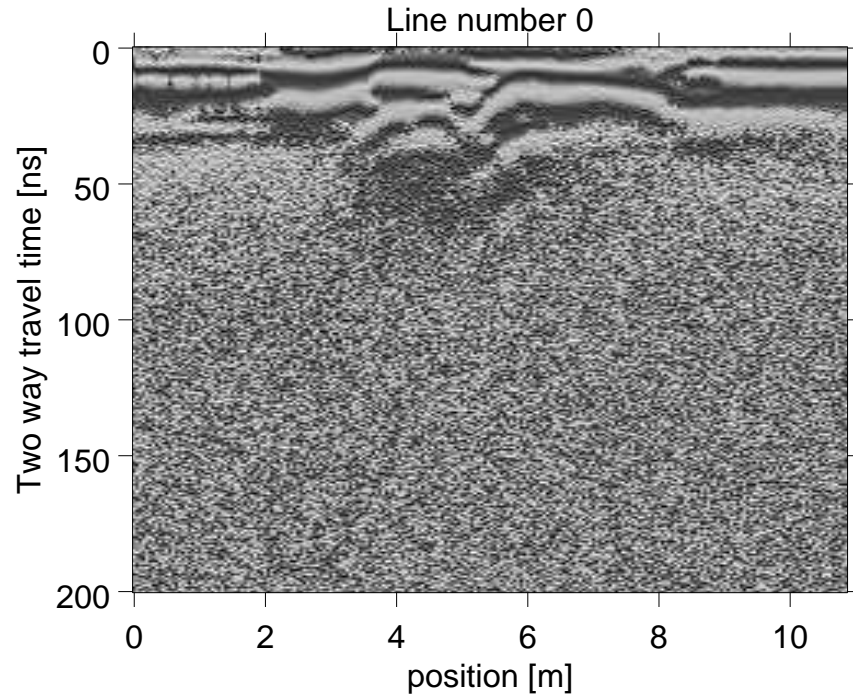
```
load exercise1dataHG.mat
```

to load it again. The loading takes a few seconds.

Let's have a look at how the profile looks like after applying gain. Run

```
plotGPRline(dataHG,0);
```

It should look something like



We can see much much more than before.

You have noticed that whenever you plotted something new, you just overwrote the old plot. You can avoid this by running

```
figure
```

to open a new figure before you plot something new.

Let's try this to see how the profile right next to profile 0 looks like. Open a new figure and then run

```
plotGPRline(dataHG,1);
```

when you compare the two then you will notice that they look very similar. That is a good sign; it means that our data measurements are reproducible and that we didn't just record random noise.

A last thing to learn in this chapter is how to turn the travel time axis into a depth axis. For that we need to know the speed of radar waves in the ground and this depends on the ground itself. We will learn later how to use Common MidPoint (CMP) and Wide Aperture Reflection and Refraction (WARR) measurements to find out the velocity of our ground but for now we simply use a lookup table, as for example

http://www.epa.gov/esd/cmb/GeophysicsWebsite/pages/reference/methods/Surface_Geophysical_Methods/Electromagnetic_Methods/Ground-Penetrating_Radar.htm

If we scroll down on that page we see that clays have a velocity of around 0.06 m/ns. We now simply use this number in the function `plotGPRline`

```
plotGPRline(dataHG,0,[],0.06);
```

The time axes is now a depth axes in meters. in the previous call we needed to add the “[” and “]” characters because the velocity is the fourth input of the function `plotGPRline`. The third input would be display contrast. If we just want to go with standard display contrast, we can use the placeholder “[]” to say just that. We could increase the contrast by for example running

```
plotGPRline(dataHG,0,3,0.06);
```

5 Area plotting

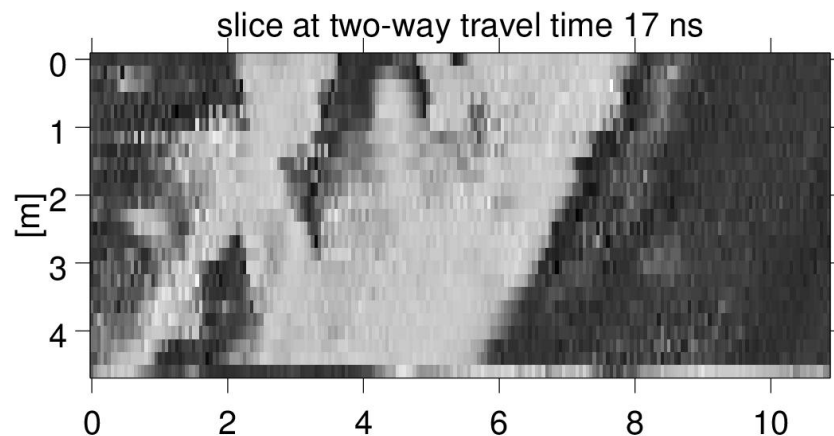
You have now learned to process and visualize profile data. In `example1` we have 24 such profiles. We can visualize them by plotting each of them individually but for geological interpretations it would be good to look at them together simultaneously. We can do that by creating a map view where we display each of these profiles as a strip next to each other and provide the two way travel time (corresponding to depth) for which we want to look at them. Let's say we want to cut through all of the profiles at 25 ns two way travel time and look at it from bird's eye perspective.

First, make sure that you have loaded or preprocessed the data and that you have set the variable `surveyparams` (you probably deleted it earlier when you ran `clear all`. You can reload it by running `example1`).

To make the bird's eye plot of a horizontal cut at 17 ns two way travel time run

```
plotdata2Dgpr(dataHG,surveyparams,17);
```

It should look like this



The x-axis here is the along-profile axis, while the y-axis contains the profile starting points. if your second profile is to the left of your first profile and the third profile is to the left of the second profile, etc, you can plot your bird's eye view with

```
plotdata2Dgpr(dataHG,surveyparams,17,1);
```

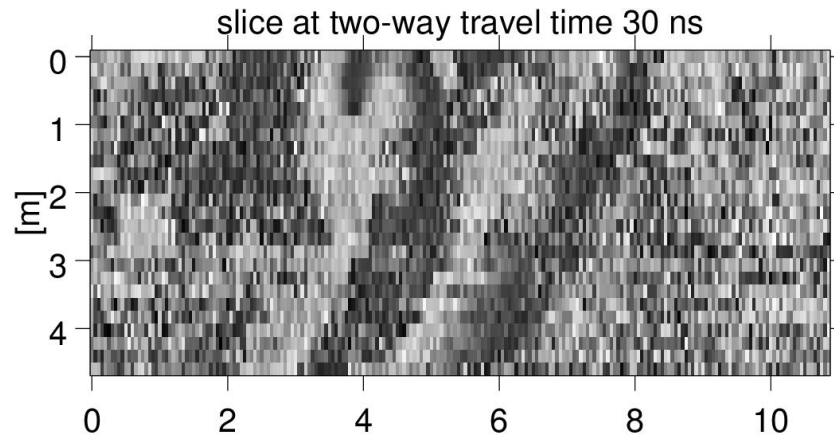
You will see that the entire plot is flipped.

Hint: If you save your figure as pdf, then you'll see that it will be a bit blurry. You can avoid this by saving your figure as jpg. There are ways to get a less blurry pdf by using the `print` function in the Command Window.

There is clearly a lot of structure in this depth slice. Remember that each profile was measured independently. So structure that shows up cross-cutting across several profile lines is some sort of structure. But what it is is the difficult part to find out. Let's first get an estimation for the actual depth of this bird's eye plot. It is at 17 ns two-way travel time, which means the signal travelled $17/2 = 8.5$ ns one way. If we assume a speed of 0.06 m/ns (as we got from the lookup table), then this will give us a depth of $8.5 \text{ ns} * 0.06 \text{ m/ns} = 0.51 \text{ m}$.

If we look at greater two-way travel times, corresponding to deeper slices, then we see that the structure is less and less well visible. A slice at 30 ns gives us

It should look like this



We can still see some structure but there is much more noise in the image.

The depth is $1/2 * 30 \text{ ns} * 0.06 \text{ m/ns} = 0.9 \text{ m}$. Considering that this is a 100 MHz system it is a bit disappointing that we can only see that far. This has to do with signal attenuation (weakening) because of the type of soil. The soil where these data were collected is clay-rich and electrically conductive. From general GPR theory (for example EPA's GPR site) we know that attenuation is proportional to electrical conductivity. Therefore, in general, conductive soils are bad for GPR depth investigations and electrically resistive soils (sand, rock, etc) are good.

6 CMP and WARR

To turn travel time into depth we have so far always used a value from a lookup table. This is based on the assumption that the velocity does not change with depth and that we made the right guess for the soil in the lookup table. Of course there are field-based ways of obtaining the subsurface velocity. Two of the classical very similar methods are Common MidPoint (CMP) measurements and Wide Aperture Reflection and Refraction (WARR). In both cases we assume that there is a subsurface reflector that is (at least where we do our measurements) horizontal.

Here we will not go into detail about the theory of CMP and WARR measurements but will talk about how you can use GPR-O to analyze and visualize CMP and WARR data. We will work with the data in the folder “example2”, which is WARR data. GPR-O has dedicated programs for WARR and CMP. Here we will only work with the WARR programs. The CMP programs work the same, you simply need to switch WARR with CMP such as for example switch `plotWARRhyperbola` to `plotCMPhyperbola`.

Make sure that you use the WARR programs for WARR data and the CMP programs for CMP data, otherwise you will get wrong velocity estimations.

As before, we first need to prepare the `surveyparams` variable and the data directories. When you open “example2.m” you see that the raw data directory is “data/raw/example2/”. In that directory you will only see a single line “XLINE00”.

It is generally a good procedure to not mix CMP/WARR lines with for example GPR profiles. The reason is that when GPR-O reads the data it adjusts every profile such that they all have the same length (needed for the bird’s eye slices). So if you have a 5 m CMP data set and a 20 m profile in the same folder, the profile will get truncated at 5 m. So keep them in separate folders and treat them as individual surveys.

The processed directory in the m-file “example2.m” says “data/processed/example2/”. If we go to the folder “data/processed/” we see that this folder does not yet exist. So we need to make it.

Then, go back to the Command Window (using the tab at the bottom) and run

```
example2
```

and as before, prepare the raw data using

```
preprawdata(surveyparams)
```

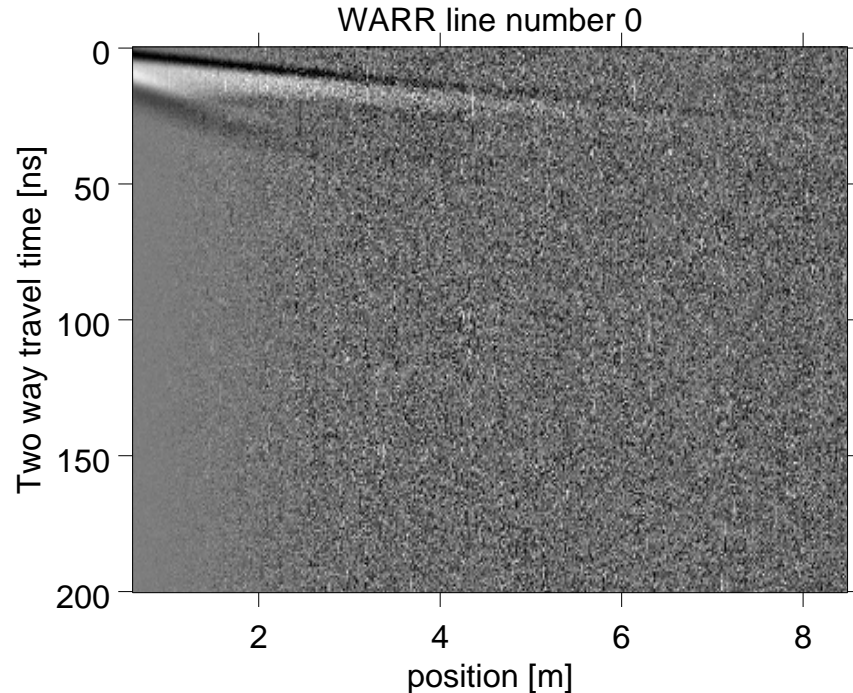
The data is now prepared and stored. You can now load the data with

```
data=readdata(surveyparams);
```

Now we need the right plotting program for this WARR data set to visualize it:

```
plotWARR(data)
```

The result should be



What we see is, as for the GPR profiles, position versus two way travel time. But here, position does not mean location of the GPR system but distance between the transmitter and receiver antenna. As you can see, the larger the antenna distance, the greater the noise. Why would that be?

The answer is: the farther away the antennae are, the more the signal gets attenuated. To compensate for that effect and to still be able to see the signal when the antennae are 8 m apart, we amplify the measured traces for larger antennae distances. But with that, we also amplify the noise.

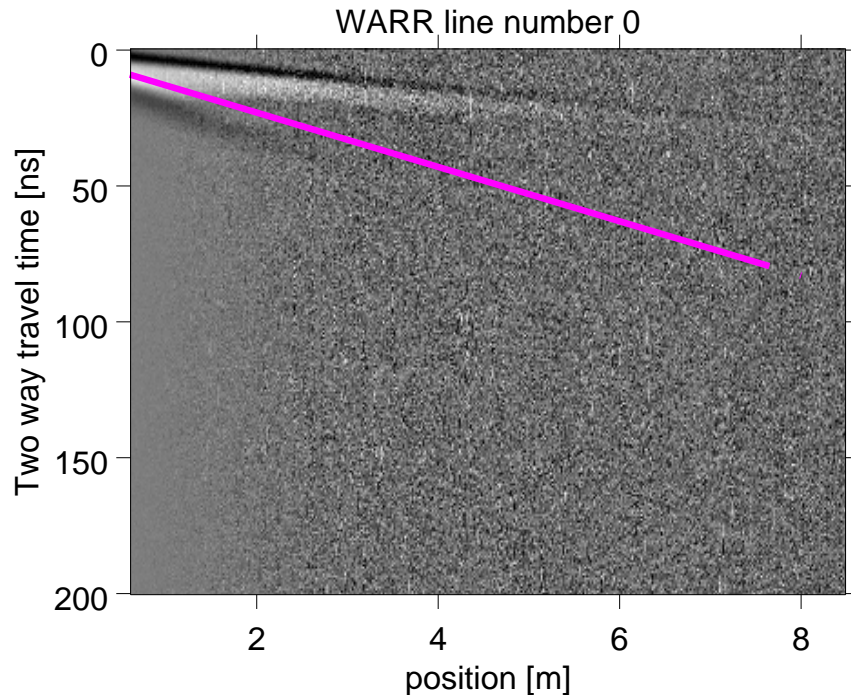
The following part requires you to have a basic understanding of CMP/WARR velocity determination. If you are confused by what we do next, please read it up in your textbook.

How can we obtain subsurface velocity from this WARR data set. We could guess a velocity and whether the wave is being refracted or reflected and then plot the corresponding line/hyperbola over the curve to see if we are right.

Let's assume for now that we can see a refracted wave with velocity 0.1 m/ns that starts (for antenna distance 0 m, which we can't see because we can't physically put the antennae on top of each other) at two-way travel time 3 ns. We can plot how this wave would look like on top of this plot with

```
plotWARRrefraction(3,0.1,8);
```

The last number, 8, was to say how far out we want to draw the refracted wave line.



It seems as if our guess was not so good. But it is generally very hard to tell. If only we had a tool that could help us with this. In fact, we have one. We could just ask our computer to try out many different such lines and see which ones “fit” a line on the WARR plot. But what does it mean to fit a line on the WARR plot? One measure of how well a drawn line fits a line on the WARR plot is to sum up all the pixels along the line. If all pixels have the same positive or negative value, then their sum will be large. On the other hand, if our line crosses lines of positive and negative values, or just runs through noise, then the summands will cancel out and the sum will be small. Let’s do this. The function for it is `plotWARRrefSemblance`.

Before we continue, let’s open a new figure to make sure we do not half-way overwrite the old one with what’s to come. Run

```
figure
```

We need to give this program the WARR data, a range for the different starting two-way travel times, a range for the velocities that it should test, and the maximum antenna separation we want to consider.

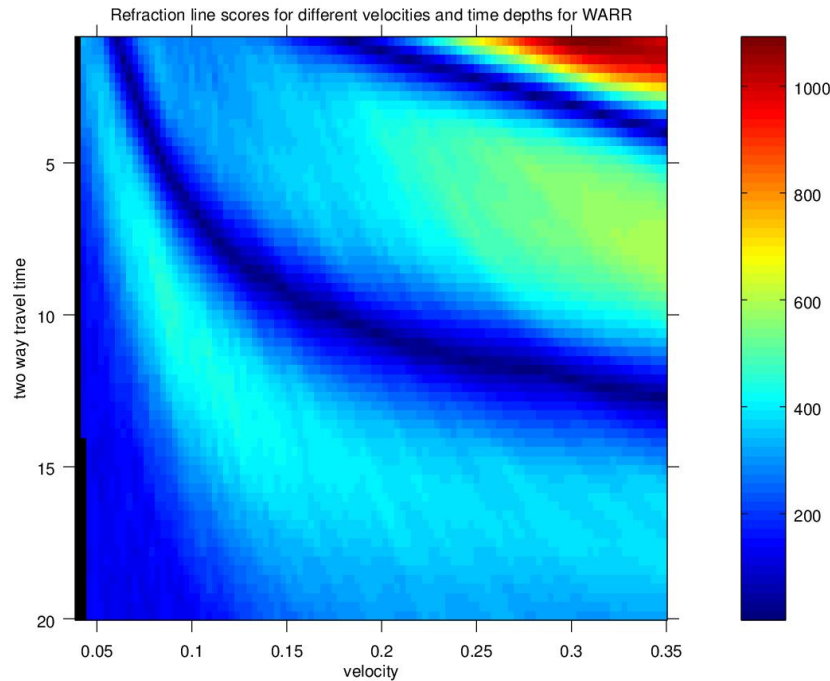
Let’s first see how Octave wants us to provide ranges. If we want to have a list of numbers, between a first number, say 2.3, and a last number, say 3.1, and we want all the numbers in between with step, say 0.1, then we need to write

```
2.3:0.1:3.1
```

So let's say we want to calculate the how well all the lines with starting two way travel times between 0 and 20, step size 0.5, fit when they have velocities between 0.04 and 0.32 with step size 0.005 and we want to use maximum antenna separation 8 m. Run

```
plotWARRrefSemblance(data,1:0.3:20,0.04:0.003:0.35,8);
```

This calculation might take a while. To speed it up you can make the step size a bit larger but that will also make the resulting image coarser.



The x-axis of this plot shows the different velocities that your computer tested and the y-axis shows the different two-way travel times at zero antenna distance. Dark-blue means low sum values, so that means that for these velocity/two-way travel time choices, the refracted wave lines do not fit any lines in the data. Light blue to green to red values means that for these choices of velocity/two-way travel times, the refracted wave lines match a line in the data.

We see three smeared out blobs of good matching. The reason for the smearing out is that the refracted wave lines in the data are relatively wide. So even if the chosen refracted wave line is a bit off, it still lies within the broad actual refracted wave line. Let's look at the centers of the blobs. When you hover your mouse over the figure you can read the two-way travel time and velocity values of the cursor location at the bottom of the figure. The center of the blob at the top right seems to be for travel time roughly 1 ns and velocity 0.3 m/ns. The second blob seems to be very broad. It's center bay be at around time 6.5 ns and velocity 0.3 m/ns. Finally, the center of the third blob seems to be at around travel time 9 ns and velocity 0.085.

Let's see what these blobs could represent. To do that, let's first redo the figure with the WARR data:

```
figure
```

```
plotWARR(data);
```

That's just the WARR data that we already saw. Let's put the refraction curve corresponding to the first blob on top of this:

```
plotWARRrefraction(1,0.3,8);
```

It perfectly overlaps with the first black line in the WARR data. Now the second one:

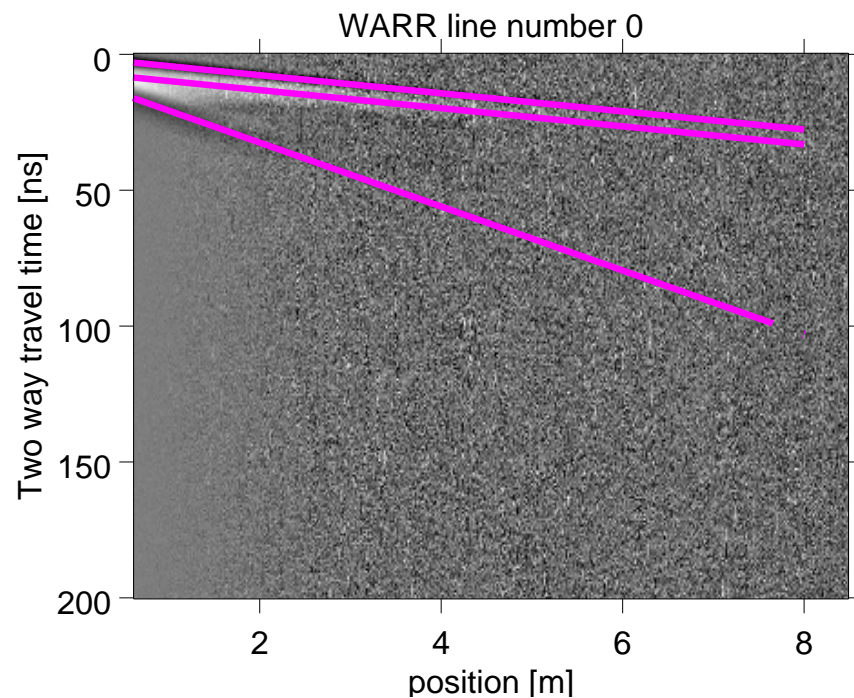
```
plotWARRrefraction(6.5,0.3,8);
```

This one perfectly overlaps with the white line parallel to the black line. Now the third blob:

```
plotWARRrefraction(9,0.085,8);
```

This one follows the black line that was at an angle.

After adding the last line your figure should look like this



So it looks like we have two refracted waves that travel with velocity 0.3 m/ns and one refracted wave that travels with velocity 0.085 m/ns. The speed of radar waves in air is roughly 0.3 m/s. From this we know immediately that the first two waves traveled through air and not through the ground. The separation between them, about 5 ns, corresponds to the distance $0.3 \text{ m/ns} * 5 \text{ ns} = 1.5 \text{ m}$. This is exactly half the length of the wave in air (3 m). Therefore the “second” wave is simply the second lobe of the airwave traveling directly from the source antenna to the receiver without going through the ground.

The last wave has a velocity of 0.085 m/ns. This is completely within the normal range we would expect for this type of soil and confirmed that we were a bit off with our guess from the table. This wave is called direct wave. It moves just below the surface as a refracted wave on the air/surface interface.

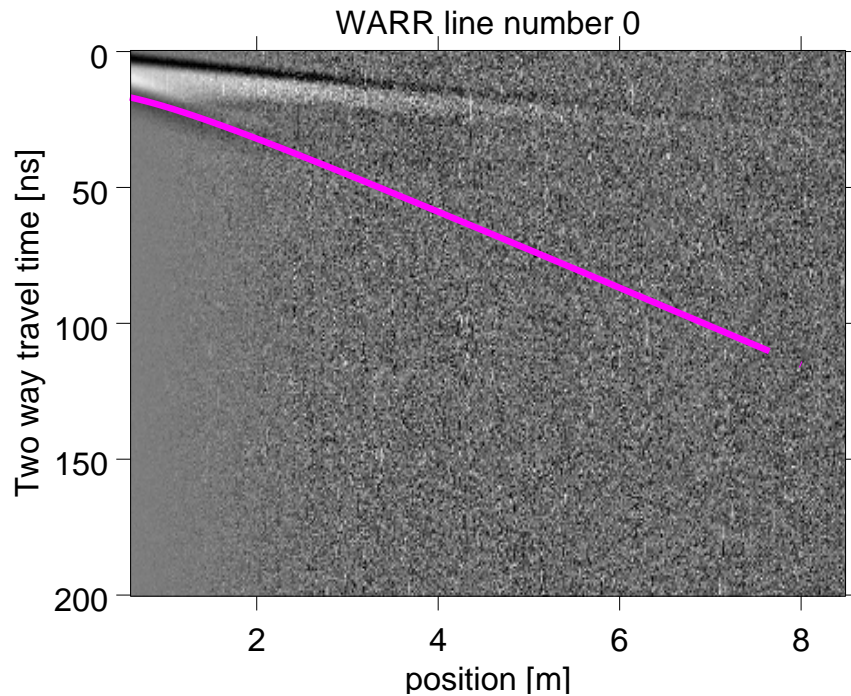
The question is now: How did we know that these are refracted and not reflected waves? For reflected waves we would have needed to fit hyperbolas and use `plotWARRhyperbola` and `plotWARRhypSemblance`.

The answer is: We need to look at the subtle shape of the lines that we see in the WARR data plot. If these lines bend, then they are likely to be hyperbolas. If they are straight, then these are either air- or refracted waves. Another option would be to see what happens when we use `plotWARRhypSemblance`. The first two blobs would be centered at travel time 1.6 ns and velocity 0.28 m/ns, and time 10 ns and velocity 0.25 m/ns. Both of these velocities are outside of what we would expect for any type of soil and they would be too slow for air. The last blob would be centered at velocity 0.07 m/ns and time 14.5 ns, leading to a slightly lower velocity estimation of the soil. To figure out which of the two velocities: 0.07 or 0.085 we should trust more, we can look at the hyperbola drawn over the WARR data and compare it to the refraction line drawn over the WARR data and see which one we think fits slightly better:

```
figure
```

```
plotWARR(data);
```

```
plotWARRhyperbola(14.5,0.07,8);
```

It is hard to tell but I think I see that close to the left edge of the graph, the magenta line curves a bit more than the thick line in the WARR data. One approach in such a case of doubt is to give a depth range of an object based on the two different velocities 0.07 m/ns and 0.085 m/ns.

At this point, in a geophysical investigation, we would go back to the the GPR profiles in Section 4 and replace the velocity there with the velocities that we just obtained.

7 Topography

GPR-O can account for topography. Use the function

```
makeElev
```

to prepare an interpolated topography matrix from elevation values on a grid or along a line (the grid has to have the same spacing on each line). Then use the function

```
elevCorrect
```

to shift each trace by the (interpolated) corresponding two-way travel time.

I will add a more detailed description at a later point. Keep your software and this documentation updated by regularly running, in a command prompt/terminal in the folder GPR-O

```
git pull origin master
```

8 Export to VTK

The function

`gpro2vtk` exports GPR-O data to a `.vtk` file, which can be read by programs like ParaView etc.