Everything
you wanted to know about

runupTool

but didn't want to ask.

Version : 3.85
Written : 6/18/2003

The Everything Guide: [runupTool]

runupTool is a matlab dependant application. It is to be executed in a matlab command window, designed around version 6.x. A wide variety of different input arguments are available depending on the applications desired. Below are some typical usages of the command.

[runupTool input]

Minimum required parameters:

---

imname =’782409432.Mon.Oct.17_15_57_12.GMT.1994.ducktape.c2.stack.ras.Z’
instName =’r829’
runupTool(imname,instName)
|
[this.usage]
The above command will save in the directory where **runupTool** was executed from. See also NOTES ON INPUT PARAMETERS, below.

Advanced argument calls:

---

imname =’782409432.Mon.Oct.17_15_57_12.GMT.1994.ducktape.c2.stack.ras.Z’
instName =’r829’
outPath =’/home/ruby/users/dclark/working/output’
runupTool(imname,instName,outPath)
|
[this.usage]
This will place the analysis data in the location that the **outPath** variable points to. It is assumed this directory exists and will not create a new directory for saving data. An error message on the command line will occur if the directory is not present. See also NOTES ON INPUT PARAMETERS, below.

Reviewing Saved Work

---

reRunStack= ’782409432.Mon.Oct.17_15_57_12.GMT.1994.ducktape.c2.r829A.runup.mat’
runupTool(reRunStack)
|
[this.usage]
This used for looking at data that has been already processed from **runupTool**. This will take the output stack that is created and re-insert that data into the **runupTool**, for further checking or analysis. It is expected that this data is in the current directory that **runupTool** was called from. From this usage, a separate instance is created than the data in the file **reRunStack**, and saved as the next letter increment available in the directory. This will not change the data in **reRunStack**. See also NOTES ON INPUT PARAMETERS, below.

NOTES ON INPUT PARAMETERS:

---

imname : The stack has **loadStack** capabilities; the stack is an actual runup stack and was created using proper Argus techniques.

instName : This is the name of the instrument in a time series, which commonly referred to as a transect in time. If you do not know the instruments on a particular stack, you can input a false value for **instName** and **runupTool** will return the proper listing back to you.

outPath : This argument is optional; it is defined simply as where you want to save the analysis data. If no argument is provided, **runupTool** will assume the current directory **‘./’**. The current directory is defined as to where **runupTool** was originally run from on the Command Line.

reRunStack : The saved data created from runupTool on another time can be re-evaluated and saved again. A special note on this usage: a separate instance is created than the data in the file **reRunStack**, and saved as the next letter increment available in the directory.

Digitizing Runup

Background:

Runup, the time-dependent location of the land-sea boundary, is an important signal for a variety of reasons ranging from its diagnostic capabilities in studies of nearshore fluid dynamics to it role in dune erosion and overtopping. Digitization of runup is accomplished in a semi-automated way using the CIL tool runupTool. Input to runupTool is in the form of cross-shore time stacks, created as described in the document makingStacks.html.

The Algorithm:

The basic assumption in automated runup digitization is that the runup edge will be of a lighter intensity than the underlying beach and that this contrast difference can be exploited to find runup edges automatically. Thus, our conceptual signal of interest is the intensity difference between any transect (time) and the intensity of the underlying, dark beach sand. The runup location is defined as the landward-most location where this difference exceeds a threshold.

Figure 1 shows the runupTool window, for reference in the following discussion.

The Background Intensity:

The problem is complicated by the fact that the underlying beach intensity is not known, may have considerable cross-shore structure (brighter and darker areas) and must be found from the data. Simple time-averaging of intensities at each pixel location does not help, since the average intensity in the swash region will be a mix of the darker beach and the intermittent breaking swash (so some of the swash will be darker than this average, so cannot be detected). Instead, we seek an average that is weighted toward the darker underlying beach.

This dark-weighted average background is found by a temporal filtering of the form

$$I_{back}(x,t_i) = \left(N * I_{back}(x,t_{i-1}) + I(x,t_i)\right)/(N+1)\,.$$

This is a running average, where the current background is the average of the current transect and N times the previous background. A large N makes the background change very slowly while a small N yields rapid change.

To find a respectable dark-weighted background, we have the running average respond quickly to the darker intensities yet respond slower to the brighter intensities in time.

The algorithm uses two N values that can be set by sliders and are labeled "lightening" and "darkening". Lightening is the N value for pixel intensities that are brighter than the previous background and is usually a large number such that the background changes only slowly on the time scale of several runup periods (note that N is in indices of rows, not seconds). Darkening is the N value for intensities darker than the background, so should be a small number, commonly less than 10.

Threshold Values

The automated runup pick is the landward-most location where **dI = I(x,t) – I_{back}(x,t)** exceeds a user-selectable threshold. The value selected for the threshold depends on the nature of the swash and on whether the swash is in the uprush or downwash phase. During uprush, the runup front is usually bore-like and has a strong intensity spike. However, during downwash the runup edge is often quite subtle and can often disappear as the downwash simply seeps into the beach sand. To account for these differences, two values of threshold are used, set by sliders labeled uprush and downrush. The uprush value can be large, while a downwash value is often small.

In some cases, the downwash simply disappears and no contrast signal is available to detect. In this event, even a human operator would need to make a subjective choice according to "the principle of least astonishment", and would pick downwash positions that smoothly shifted seaward to the next uprush. RunupTool assists by providing a variable to control the maximum rundown between adjacent samples. This value is set by the slider labeled "rundown".

To place the data in the directory,

/home/ruby/users/dclark/working/runupDoc/tutorial/

Run the commands:

```
imname = '1047418844.Tue.Mar.11_21_40_44.GMT.2003.argus02a.c1.stack.ras.Z';
runupTool(imname, 'r583')
```

Notice that this is the two parameter listing of **runupTool**, meaning that the save file will be located in the directory matlab was currently open to when the save button is pressed.

The program will send out some output:

```
Using the current directory for saving....
This is instance   A   of this data
old ID: 28 new ID: 27
This version 3 stack started at Tue.Mar.11_21:40:43.GMT.2003 (1047418843)
Available Instruments on this stack:
   'r583'
```

Let us discuss a bit on instances of data, referenced to line two of the above output.

An instance is essentially an occurrence of the analysis data such that the outputs of two runs do not conflict when saved to file. This can be from multiple people working with the same stack, or it can be from the same person working with the same stack. It is a safeguard to prevent overwriting of and subsequent loss of information. A quick note for those more advanced users, this program is synchronised. This means that multiple executions of the application are unaware of each other at runtime.

After the brief output to the command screen, a GUI window will soon appear that has several key sections. This includes the sliders, the display choice box, the display window, and the action and movement buttons.

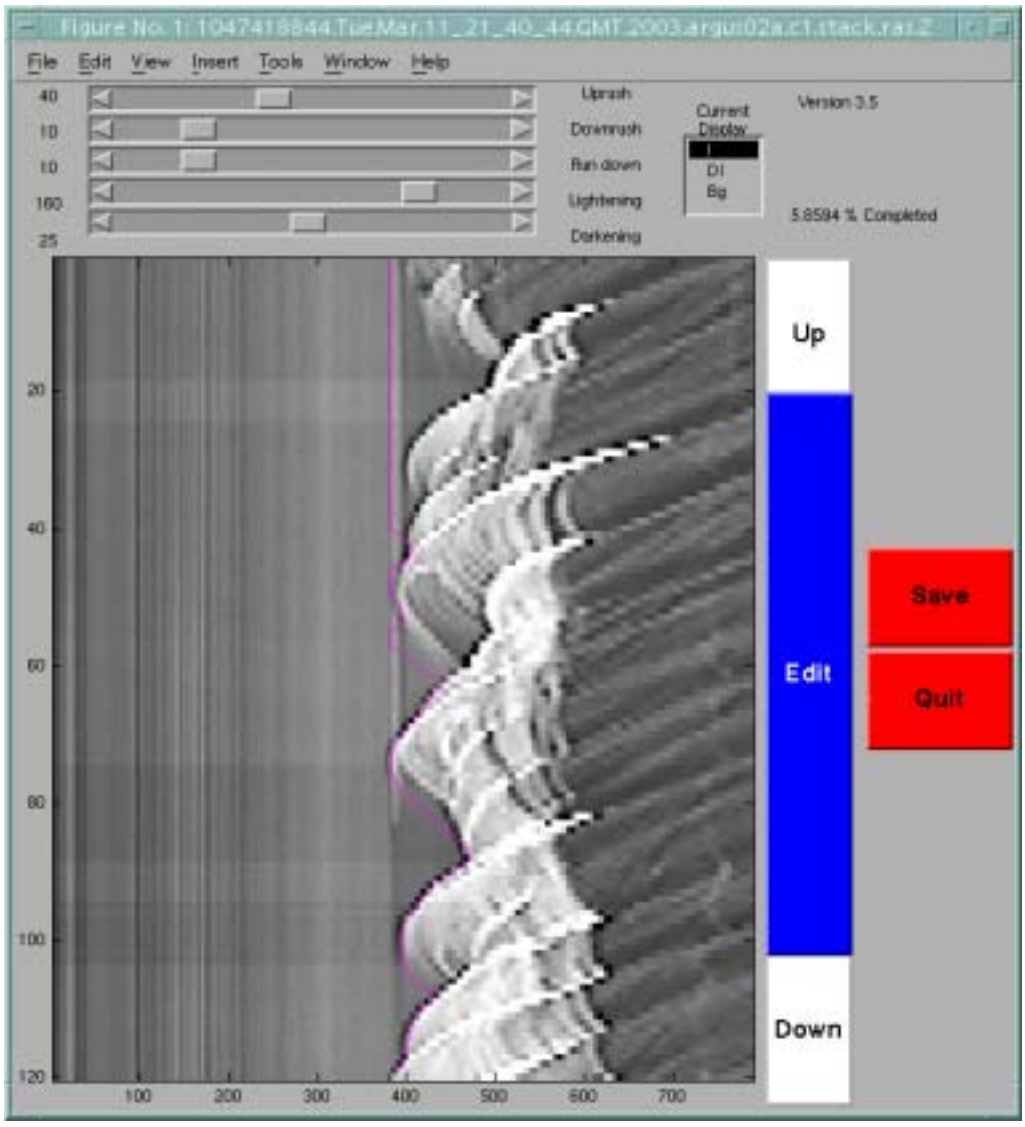


Figure No. 1: 1047418844.TueMar.11_21_40_44GMT.2003.argus02a.c1.stack.rat.2
File Edit View Insert Tools Window Help
40
10
10
160
25
Uprush
Downrush
Run down
Lightening
Darkening
Current Display
DI
Bg
Version 3.5
5.8594 % Completed
Up
Edit
Down
Save
Quit
20
40
60
80
100
120
100
200
300
400
500
600
700

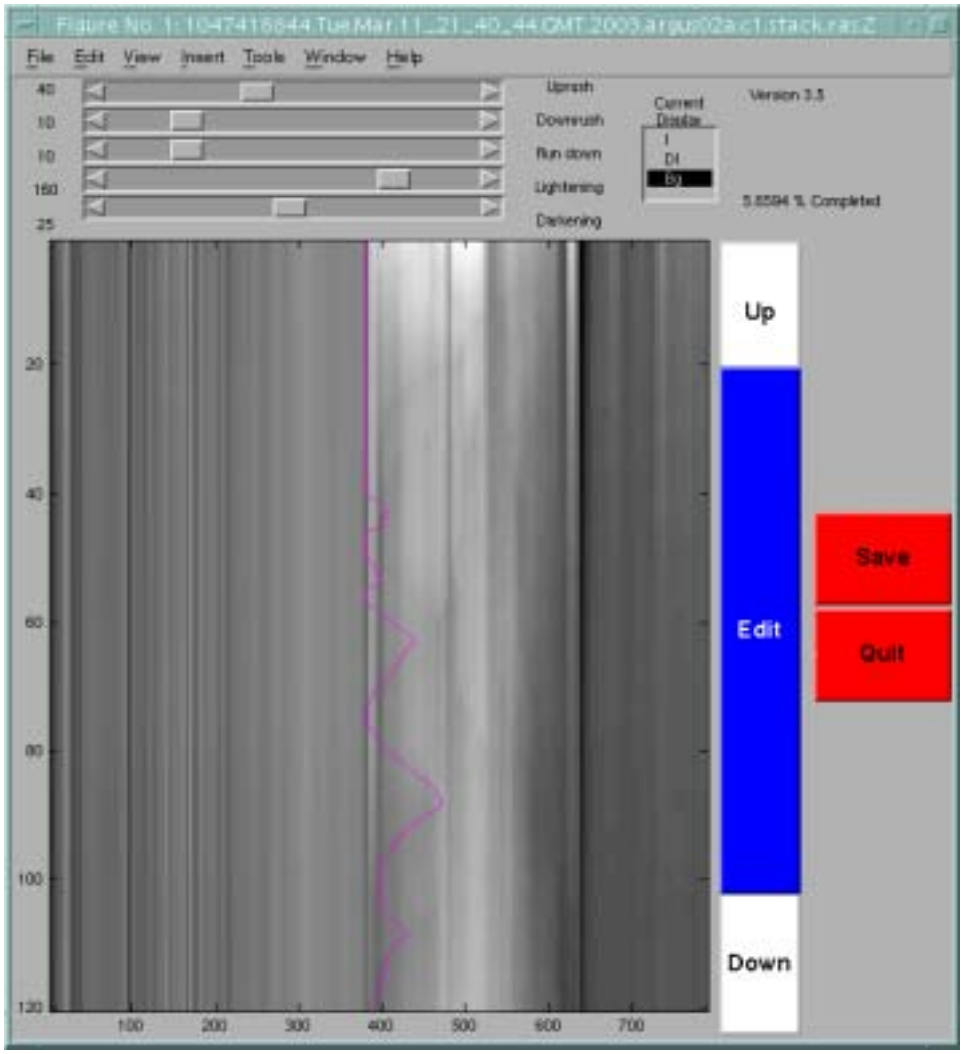
[runupTool tutorial: Sections Defined]

Though the button labelled **'Edit'**, used for manual editing looks like a likely choice of where to begin, this GUI was designed to revolve around a very advanced Automated Edge Detection routine to save time. This routine's parameters interface with the user in form of the sliders above the display window.

The sliders will change the line that is shaped around the runup in below them in the runup display window. Each of the sliders has values to the left, and to the right there are labels for each one. The bottom two sliders deal with background calculations, while the upper two deal with threshold values regarding the intensity of the image, row by row, below in the runup display window. The middle slider, labelled **'maxRundown'** deals with the rate at which the line travels from the left to the right across the display window. The significance of the values on the left for the thresholds is in denominations of pixel intensity jumps, while the bottom two are in units of rows (down in time).

[display modes]

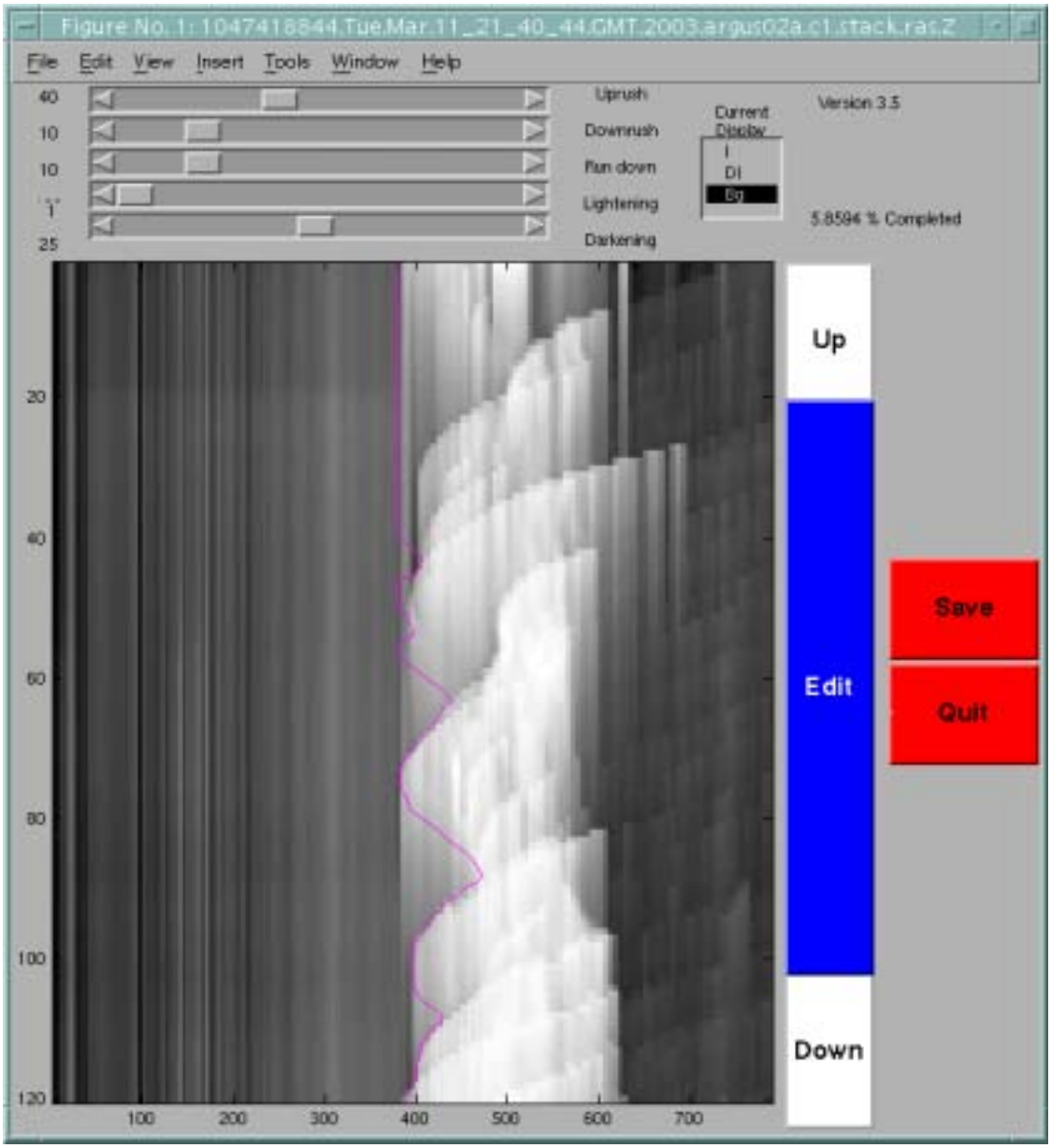
The most critical piece to resolving a good line is to get the proper background. A properly calculated background will make the thresholds respond nicely and predictably. You can see what the background intensity profile is by clicking the option in the box entitled **'bg'**.



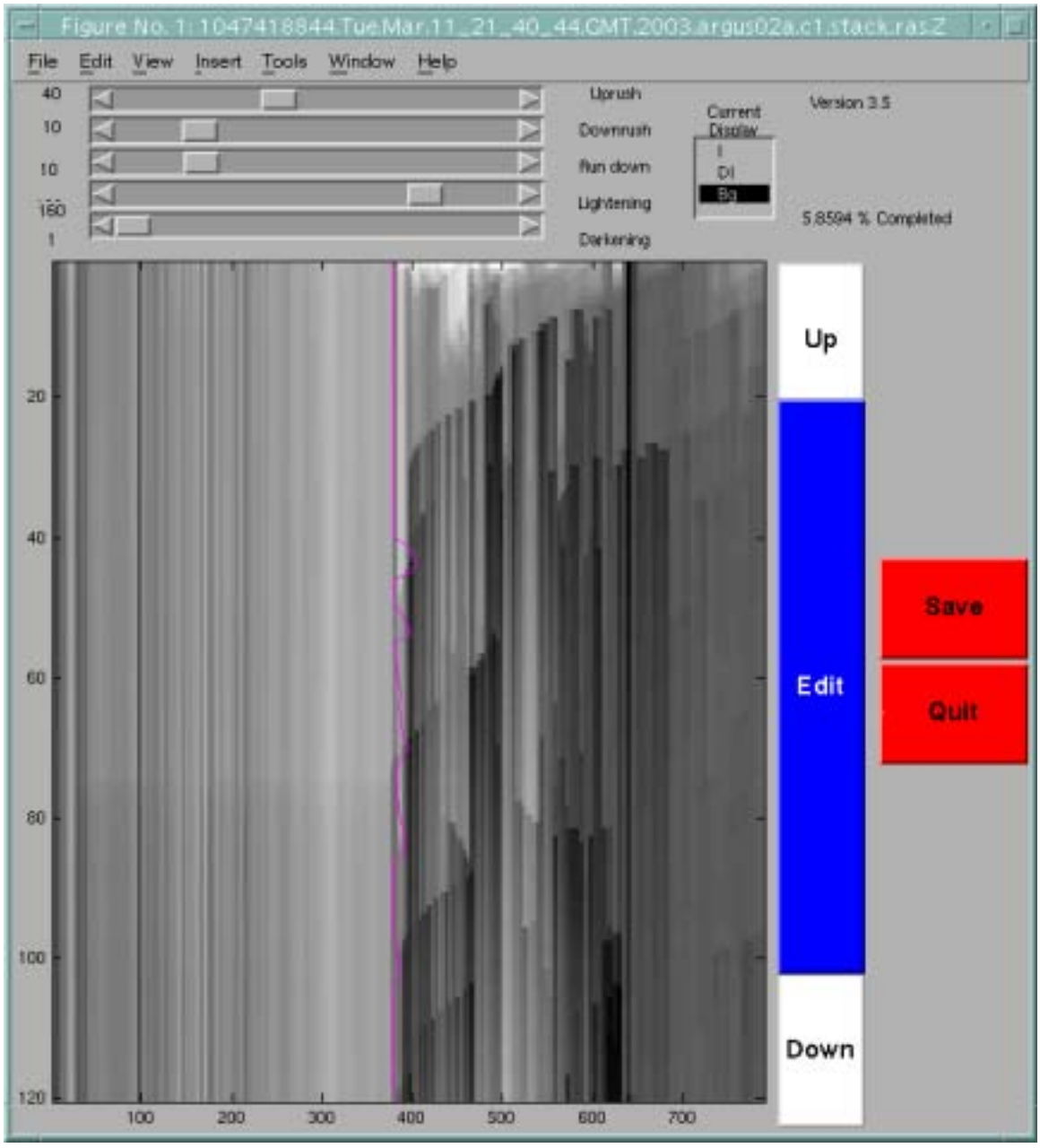
As you can see in the runup display window, there are some bright intensities at the top that will be eventually subtracted from **I** to form **dI**. (dI = I - bg)
The principle theory behind finding a "good" **'bg'**, is to "smooth" the beach detecting the theoretical profile under the waves.
It is still important not to have significant detection of the breaking action subtracted from **I**.

[Lightening Slider]

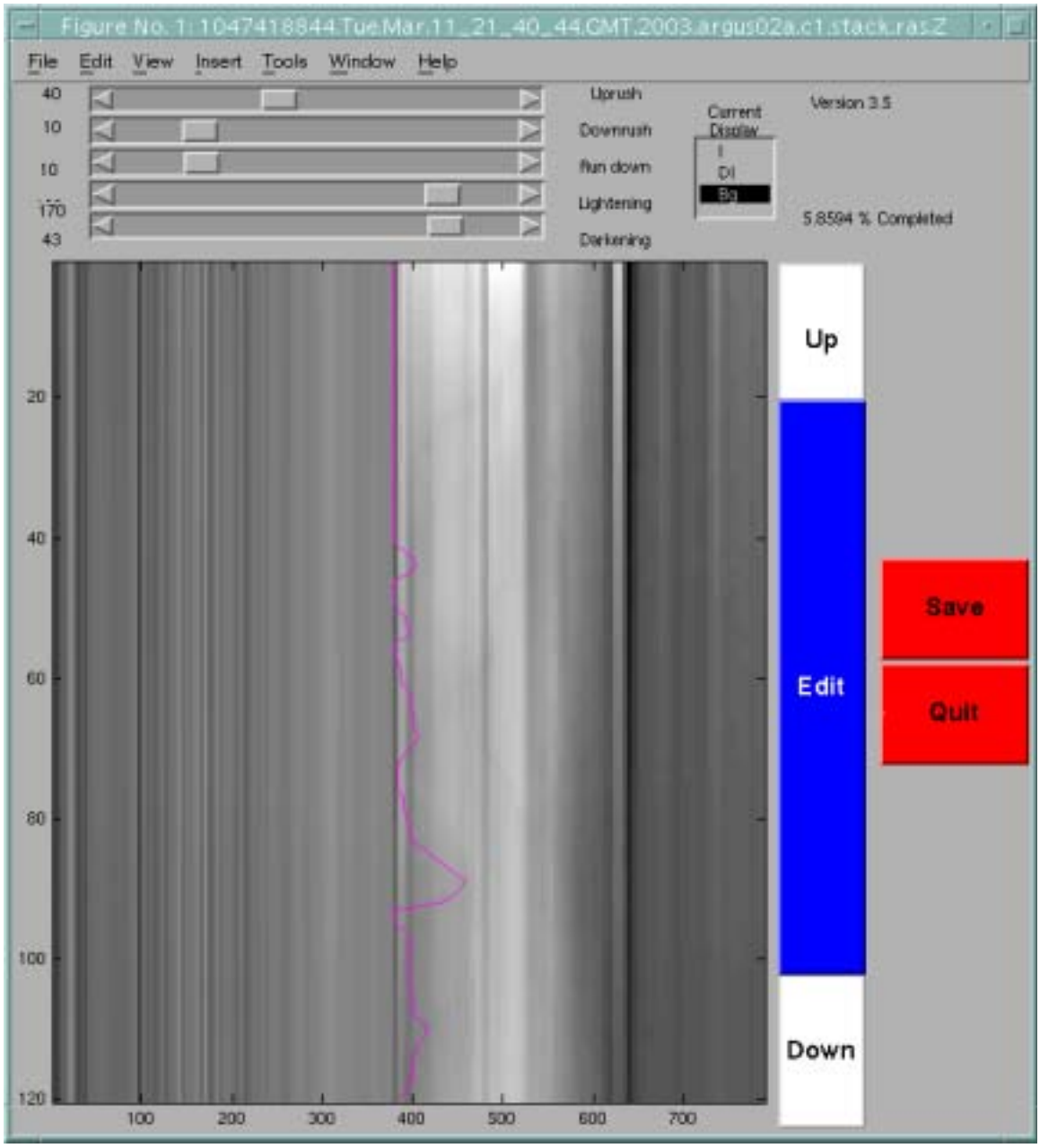
By adjusting the **'Lightening'** slider, one can adjust the amount of light intensities subtracted from **I**.
In the image below, most of the breaking is going to be filtered out. This is a dramatic example of a bad background.



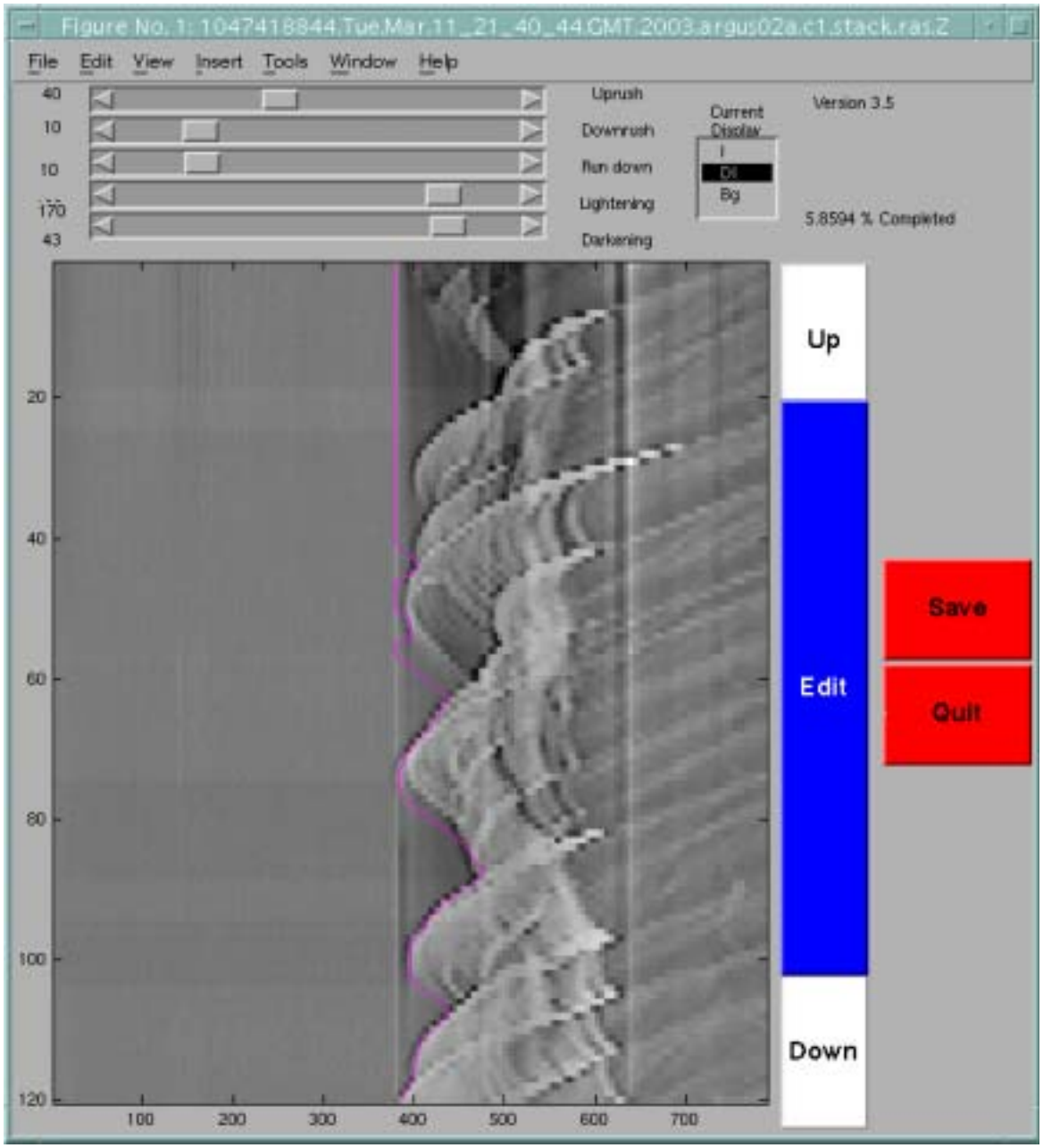
Likewise, by adjusting the filter **'Darkening'**, the background is lowered at a rate that brings out features that will infer with the thresholds.



So what does a good **bg** look like? It is possible to have many different "good" backgrounds that are different.
Because even the same location can have drastically different backgrounds on different runs, it is important to develop a system with the backgrounds you create. The method that works well for me is to make a guess at the background that subtracts the entire beach, with the trade-off of minimal breaking subtraction, and check the results with the thresholds that fit that particular curve.



As you can see, I am filtering to remove a large portion of the image. It is now important to turn the attention to the display mode **DI**.



By utilizing the proper filter techniques of the background that I have selected, a large portion of the beach (left side of the waves) that was once quite noisy is now smooth. The thresholds will run across this area looking for an intensity jump of the value determined by the top two slider values.

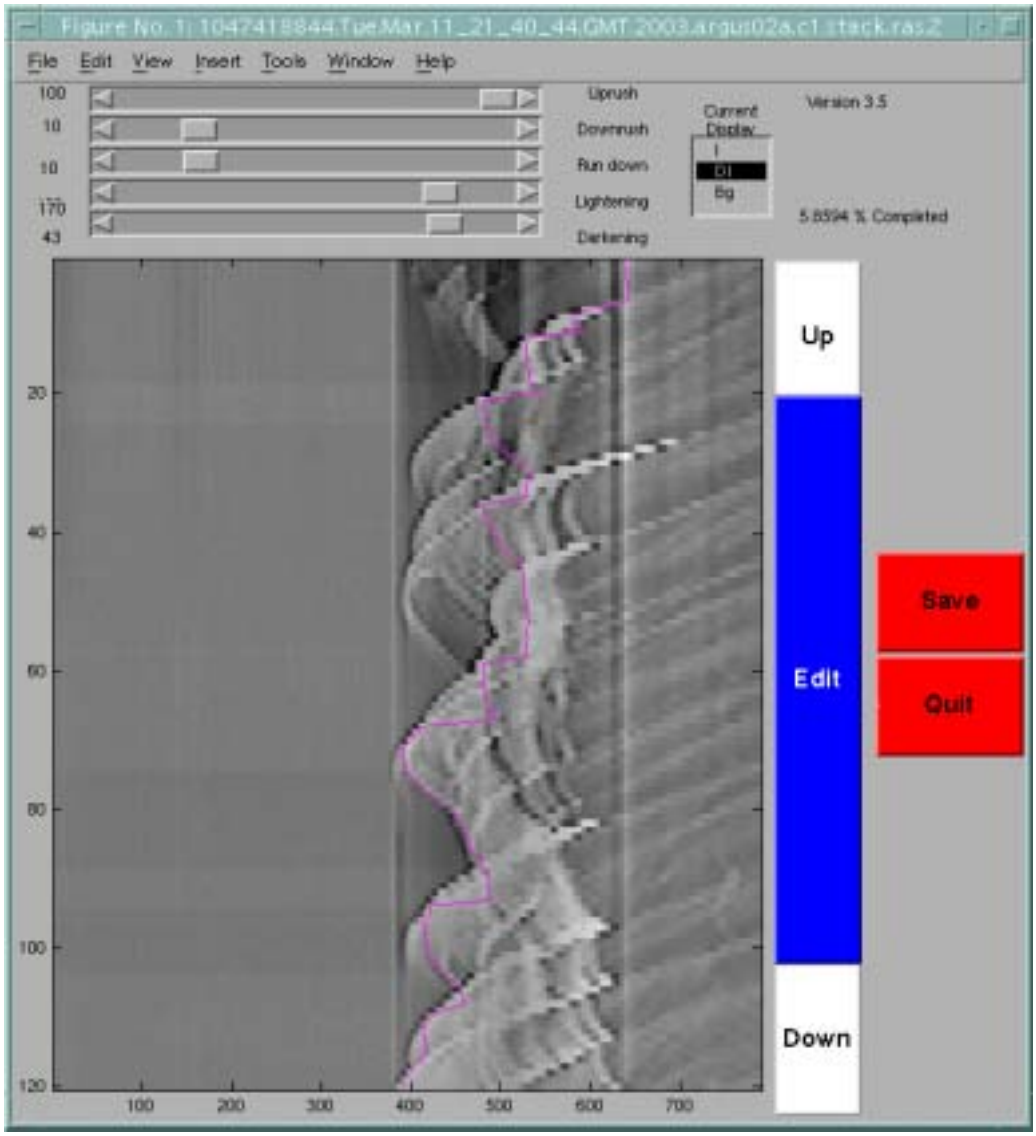
Let us review about waves and the mechanisms of **runupTool** that interpret them.

When a wave breaks, the camera observes a bright intensity of foam on the already dark, wet sand beach. This bright intensity is easily detectable and is the principle signal **runupTool**'s Automated Edge Detection filter "looks" for. As the wave progresses up the beach, (in the figure shown above, right to left) it will leave a light, represented as a white pixel, covering the darkened underlying beach. When **runupTool** encounters this, it is interpreting what is called **'Uprush'**.

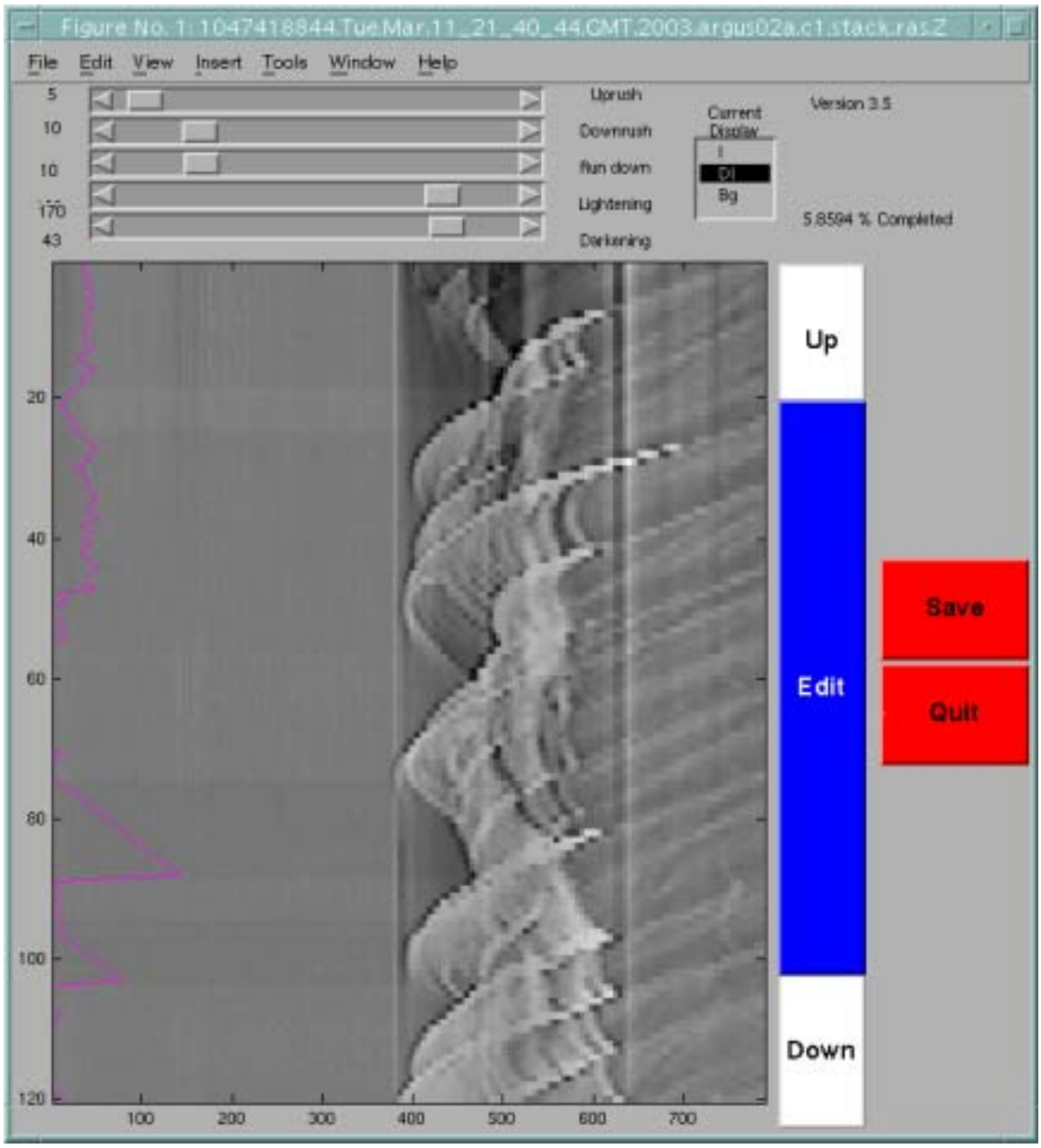
Conversly, when the wave recedes back into the ocean, (left to right movement) **runupTool** is interpreting **'Downrush'**.

For simplicity in discussion, each of these intensity jumps is only considered at one row across the swash zone at a time.

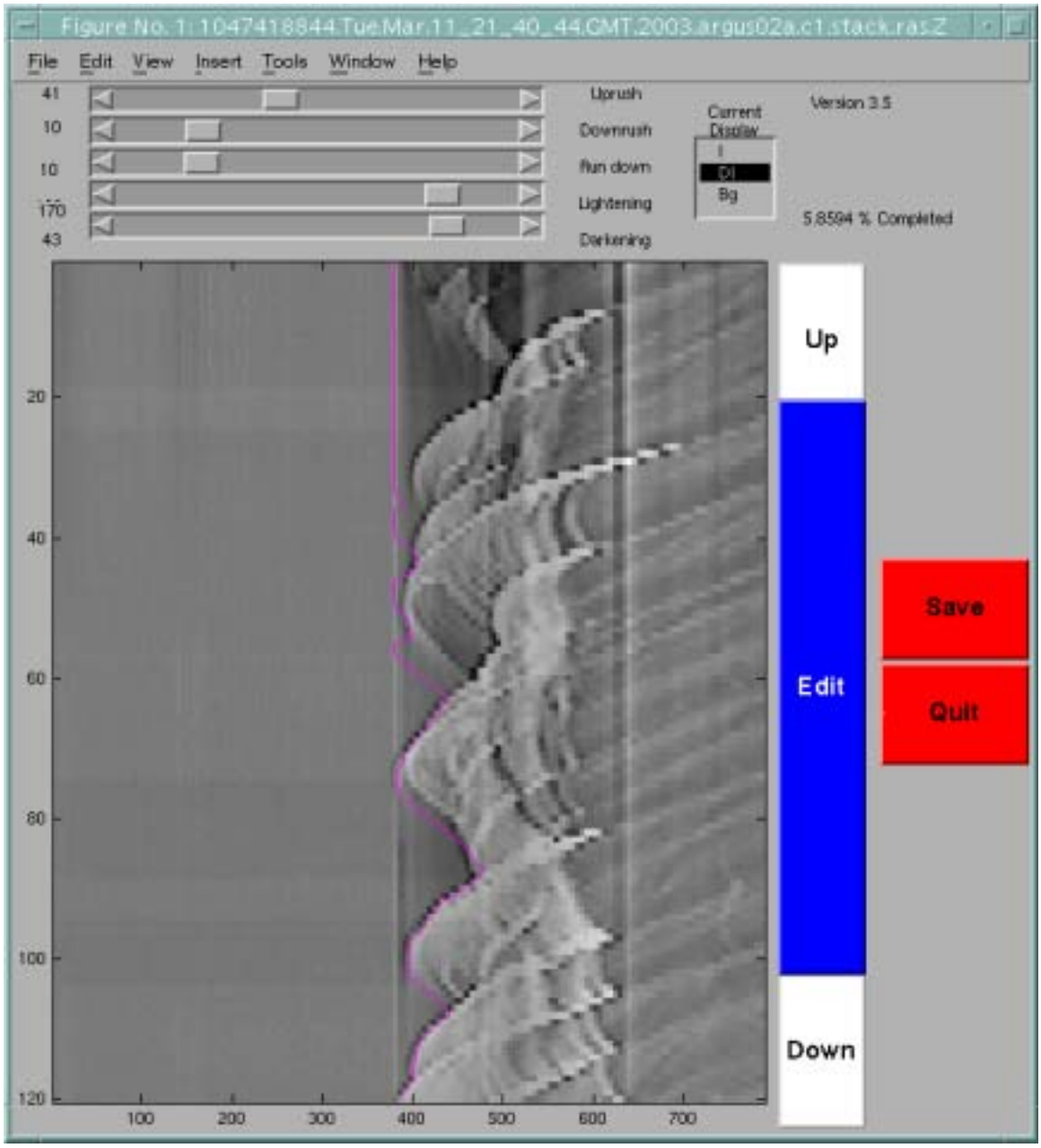
As seen by this image, if the **'Uprush'** filter setting is set too high, the lesser (in intensity...) breaking waves will be "missed". The precision of the leading edge is also lost.



Inversely, setting the **'Uprush'** too low will cause the detected edge to be “stuck” on the beach.

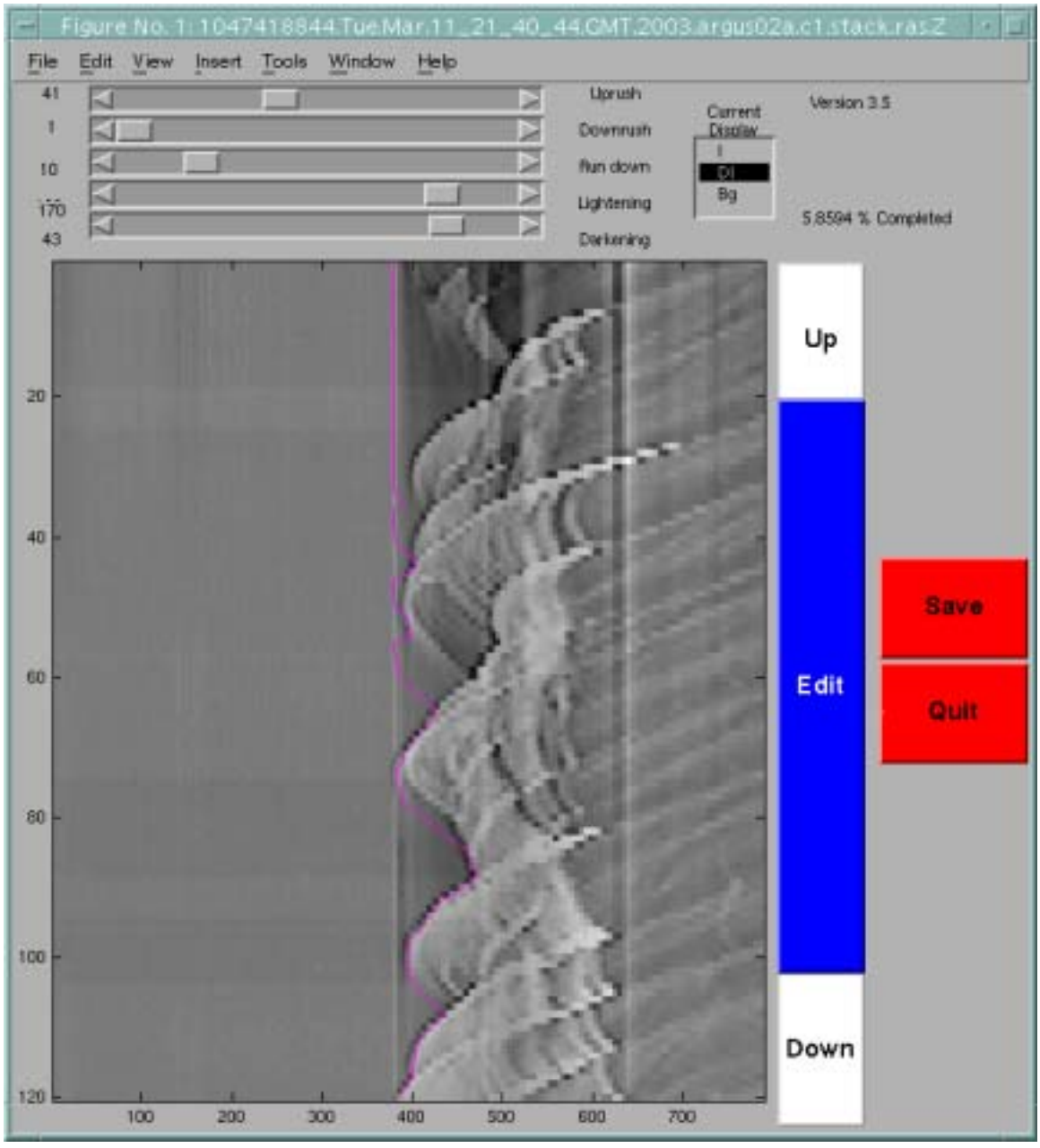


So somewhere in the middle, a trade-off is met. The top of the run as you can see has issues that are going to have to be resolved later. The important part is that the bottom of this screen has a very good setting and will be very useful later screens.



[Downrush Slider]

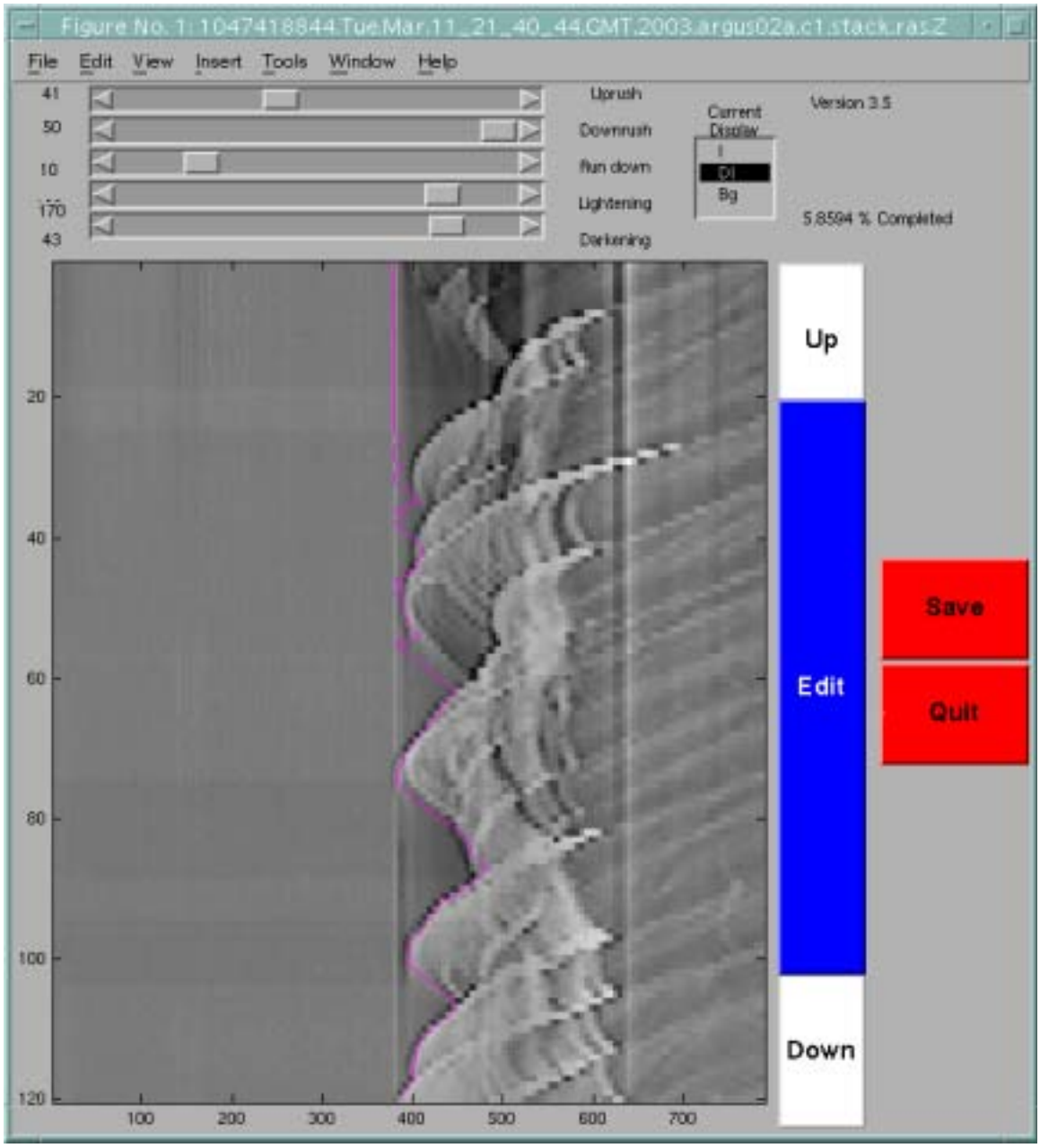
Downrush varies from person to person, and instance to instance. It is a very difficult parameter to both measure and interpret. Interestingly, even insitu have a difficult time measuring this parameter of runup. Moving on however to the **'downrush'** in a similar fashion, the extreme low setting looks like this.



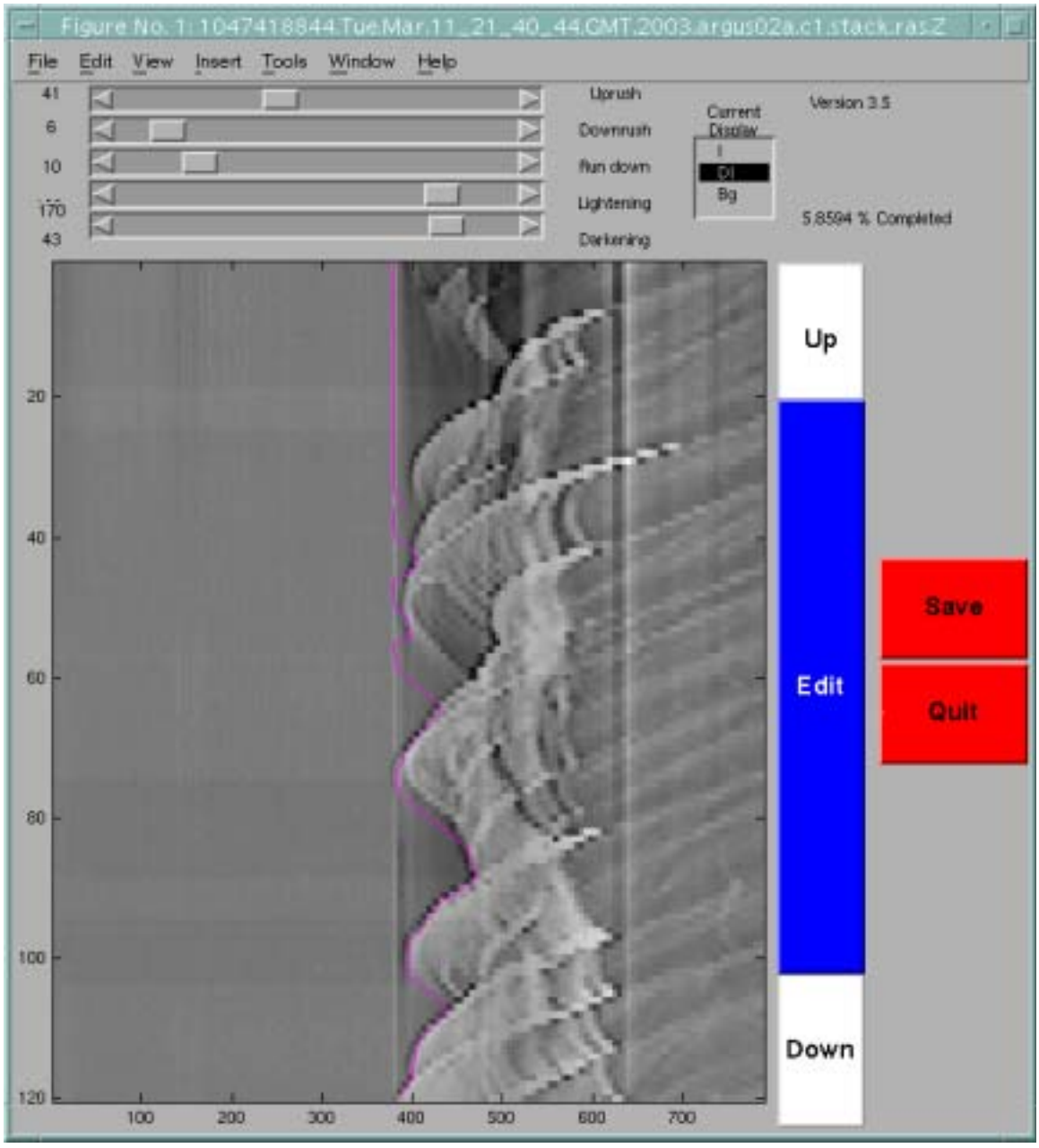
As you can see, at this setting, the filter will accept nearly any jump in pixel intensity, which is not usually the correct setting. Although it is not a direct co-relation always, conceptually the lower settings will cause the wave to progress back into the ocean slower.

As you may have already guessed, setting the **'downrush'** too high causes exact opposite.

If no value breaks this threshold on a given transect, then the runup will progress seaward at a maximum rate (left to right only) in which **'Run down'** is defined by.

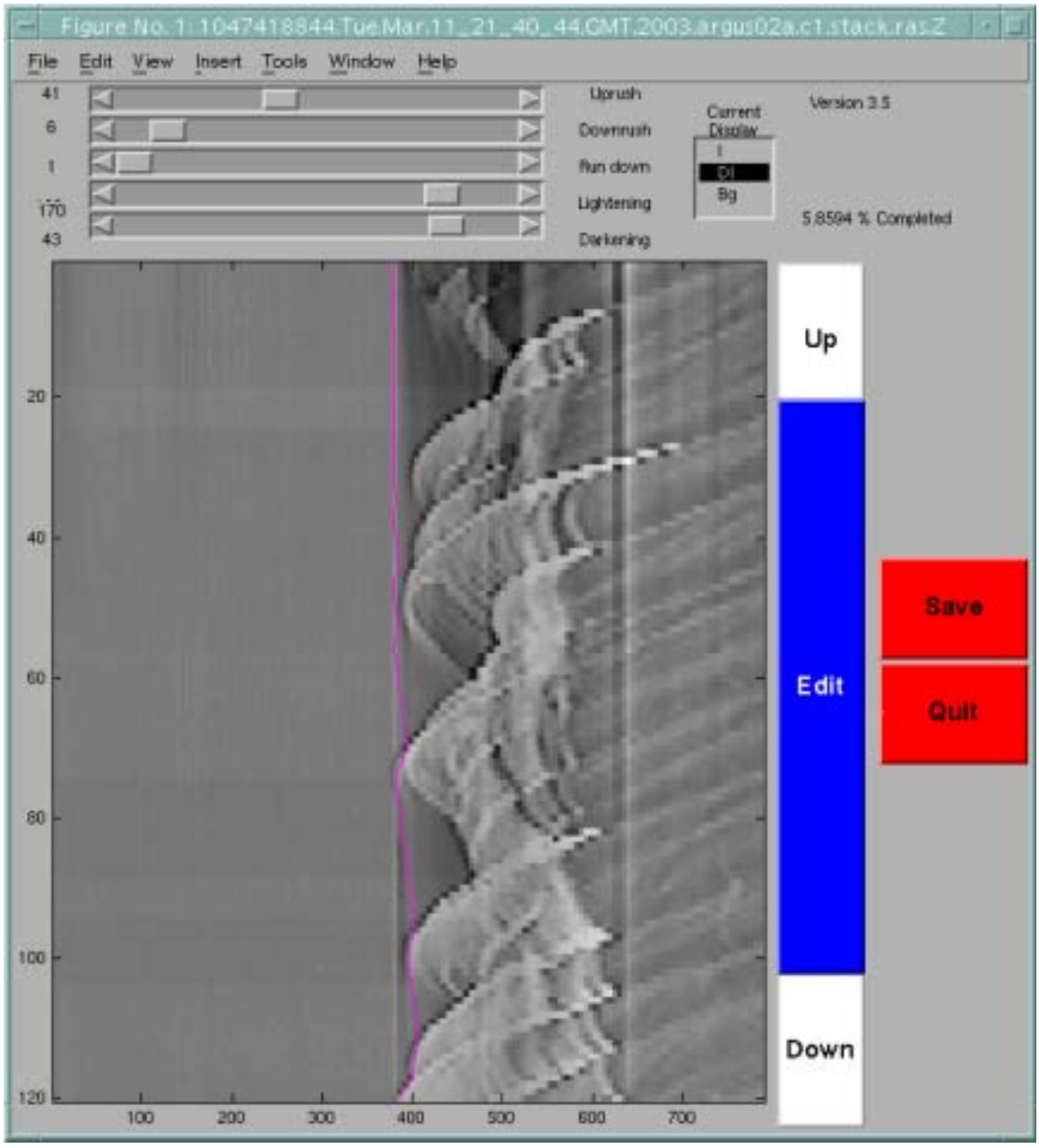


Adjust this slider until you are satisfied with the results. The true definition of **'downrush'** is the landward most edge where visible water is moving. This is not always a visually detectable signal, and we will cover manual editing by visual interpolation later on in this tutorial. As you can see by the image below, not all of this detected runup is correct, however as time progresses, the computer detected edge is closer to correct.

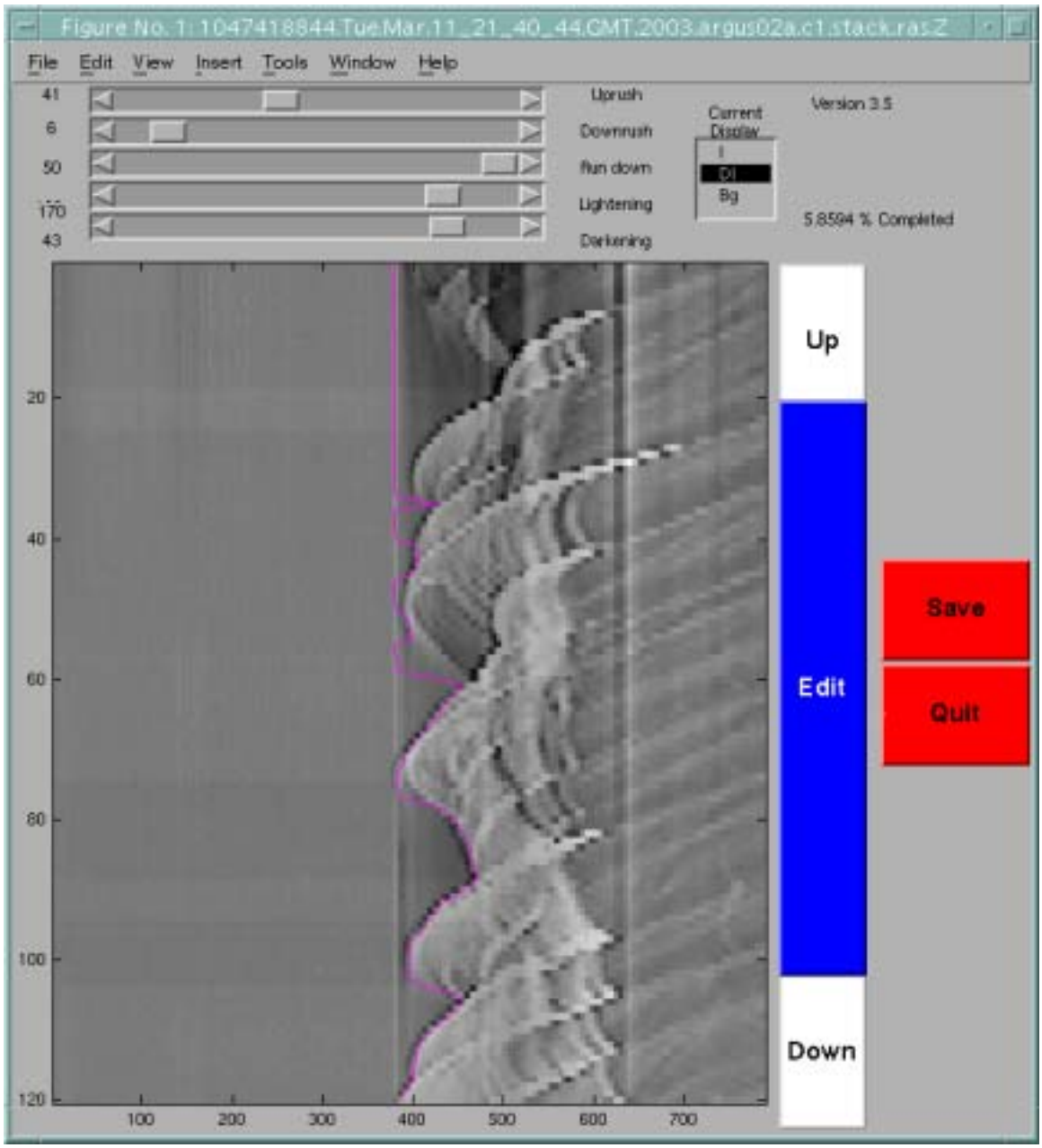


[max Rundown Slider]

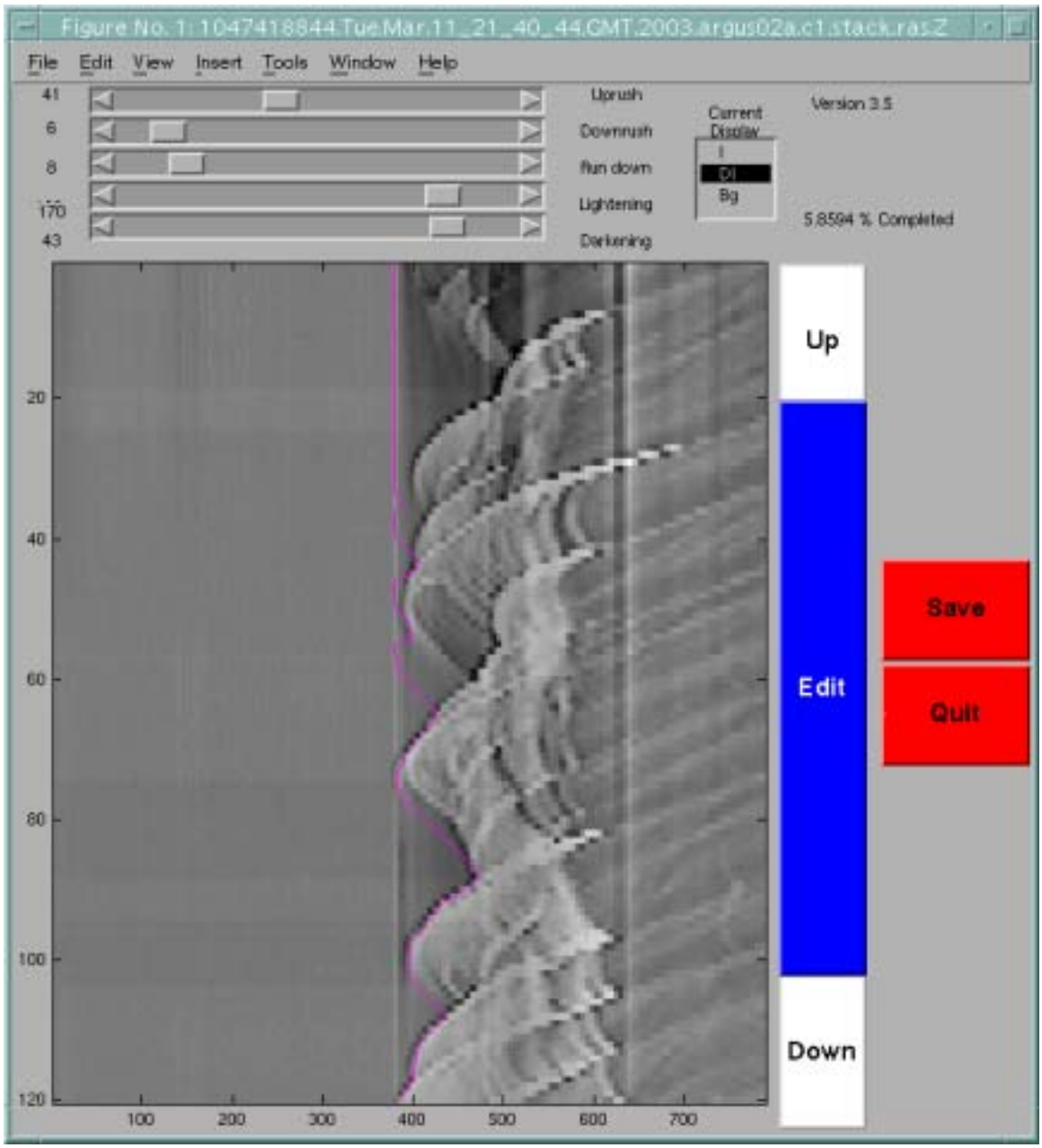
As discussed above, it is important also to address the difference in **'max Rundown'**. The difference is rather dramatic. Here is **'downrush'** to low:



Conversely, too **'max Rundown'** too high:



Finally, the **'max Rundown'** set:

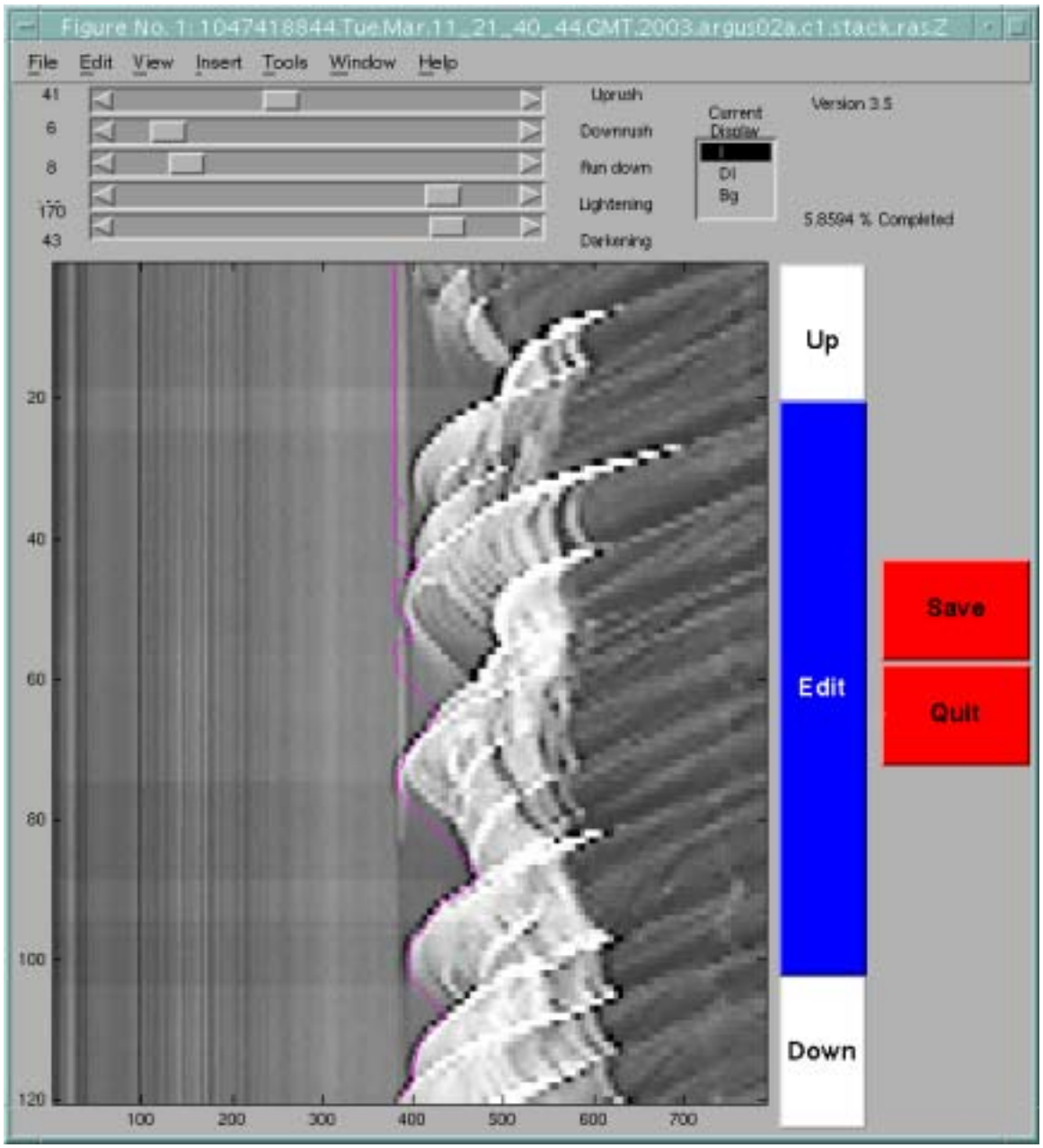


If the desired results are not achieved through the sliders, try a change in the background and then continue through this conceptual procedure again. Though this may seem a little tedious and foreign now, soon it will become second nature.

[Manual editing: the 'Edit' button]

Clearly, the points at the top of the display window are incorrect. To re-emphasize, the number of points yet to solve is greater than the points at the very top.

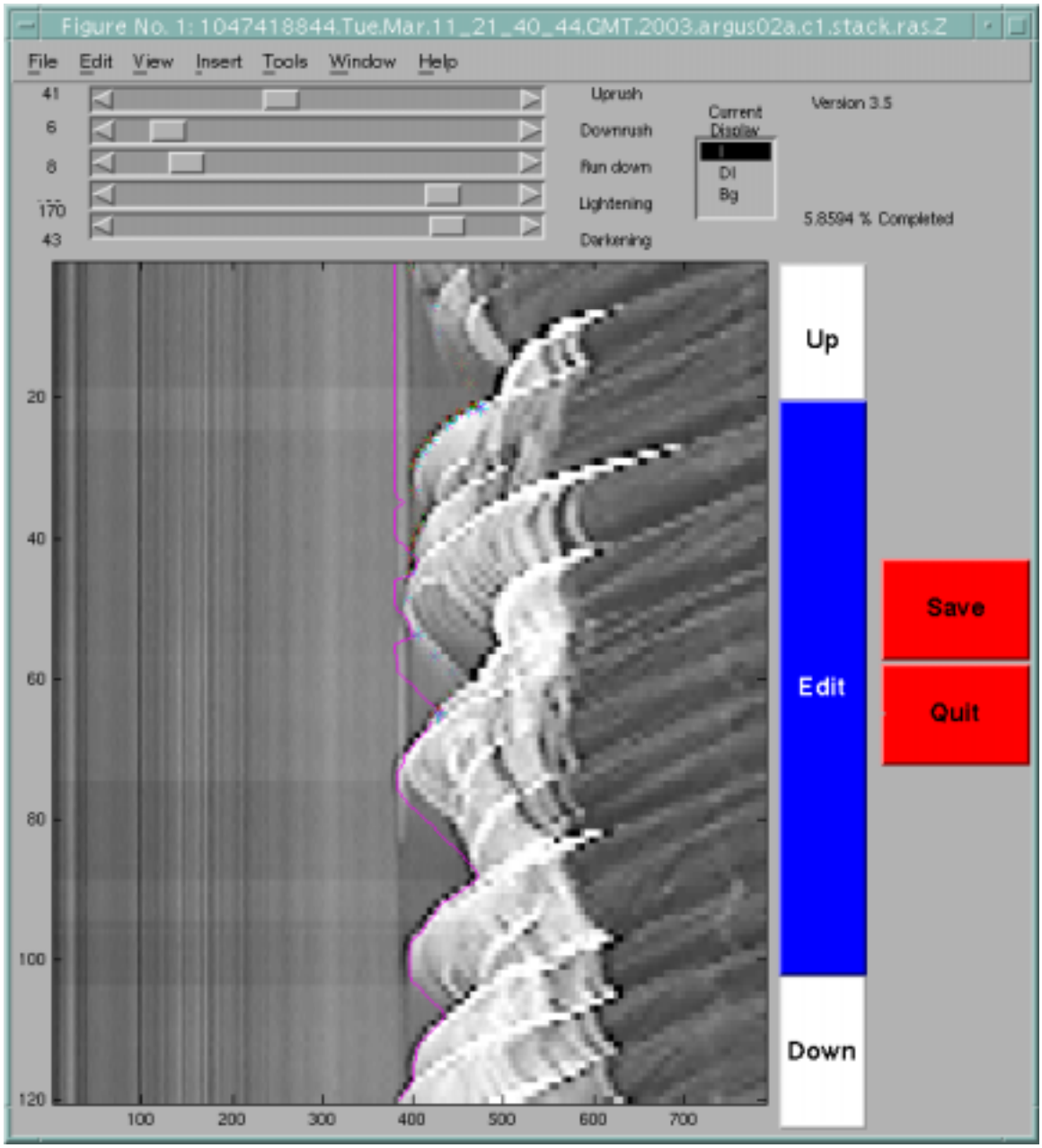
It is important to solve correctly for the **Ri** (detected edge) before proceeding on in the stack however. Let us use the big blue **edit** button to edit those points manually. I like to switch the display mode to **I** so the user can see all the intensities of data.



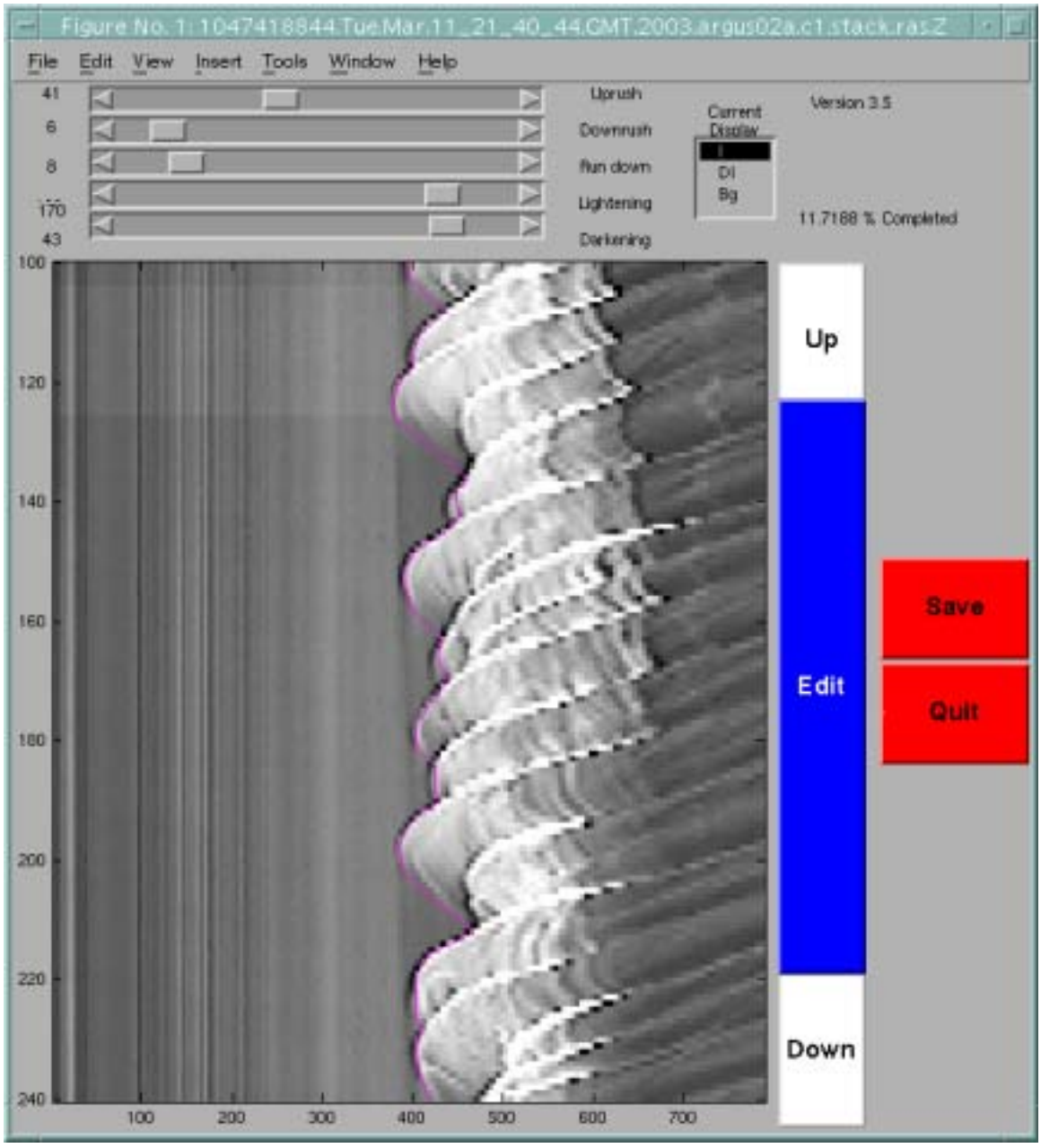
Interpreting this data visually by human eye, it is now clear why the Automated Edge Detection filter was "stuck" there is a vertical linear artefact that happened to pass directly in front of the runup collection transect. Such artefacts sometimes take the form of birds or people.

To remove this from the detect edge; click the big blue button labelled **'Edit'**.

You are now in **'Edit'** mode. Notice that your mouse cursor has changed to an X. wherever you click, you will select a point. Your task now is to interpret the runup visually, creating points where you think the runup is detected. When you are finished doing this, clicking a final right click to close the points will end **'Edit'** mode and subsequently draw a linearly interpolated line between each of the points. You may do this as many times as necessary: in very small or very large increments as desired. Here are my points before closing the line.



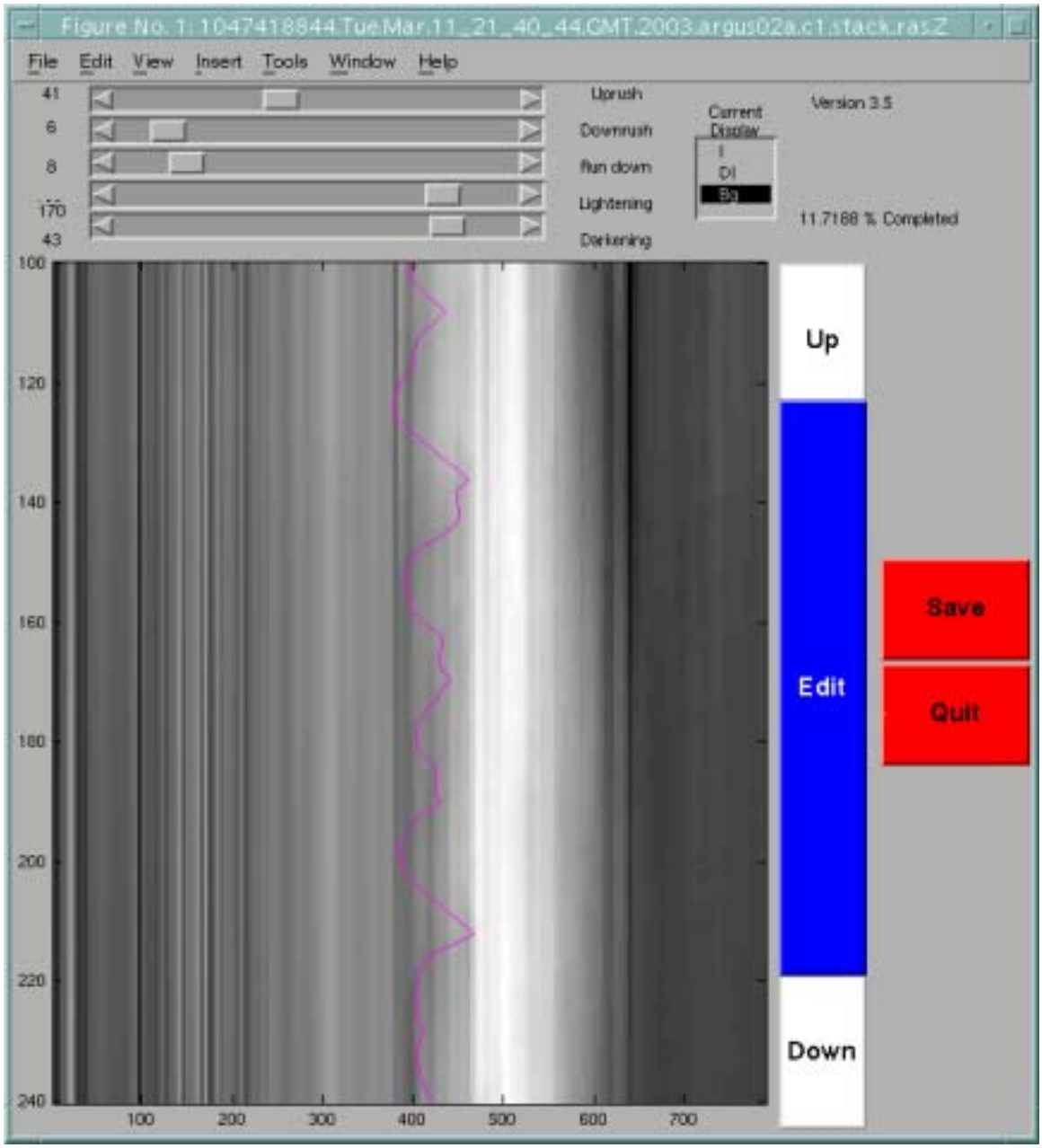
When this screen looks satisfactory, press the down button to view data later in time. Pressing this button progresses on to screen two.



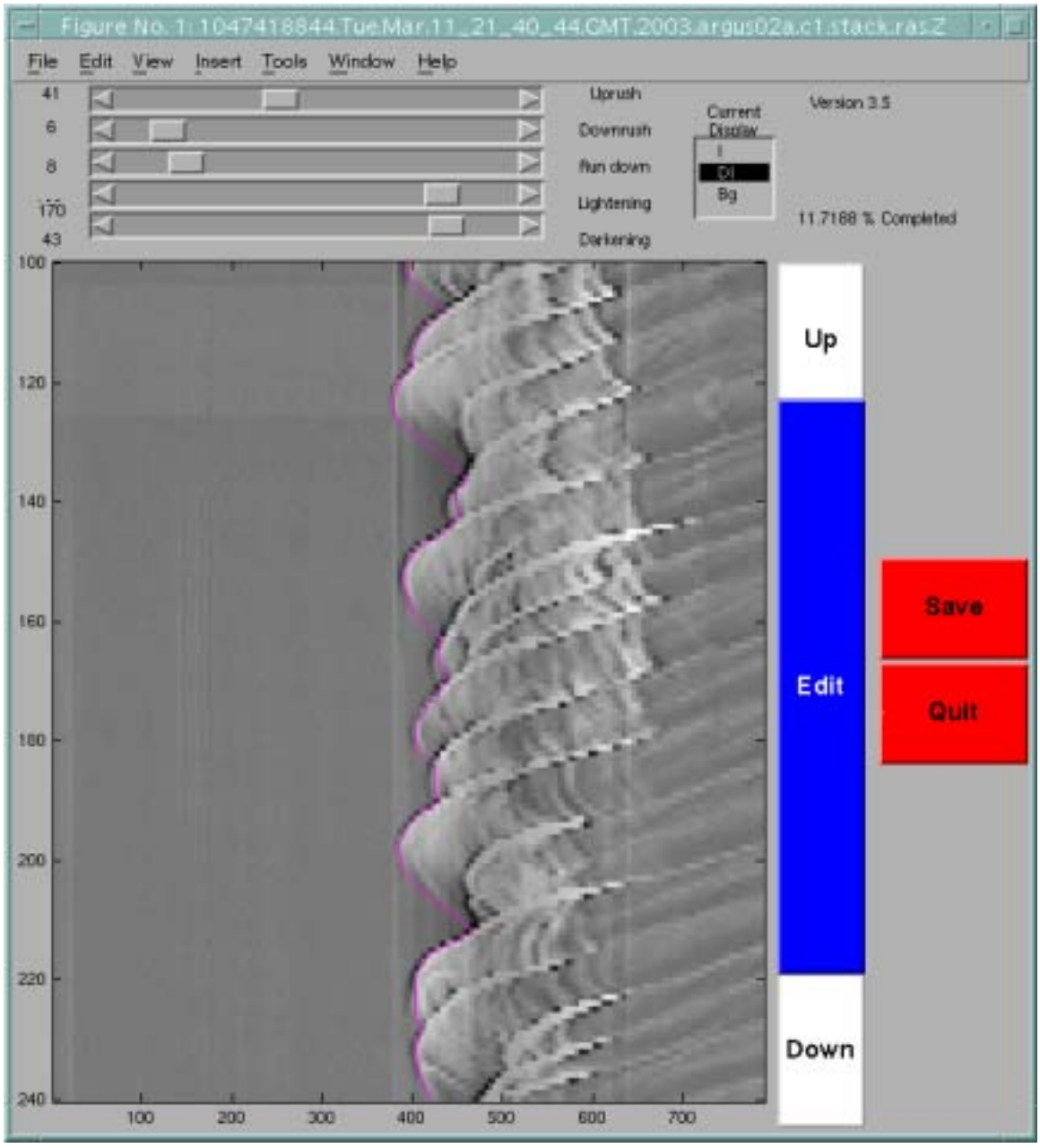
[The next screen and beyond]

As you can now see, the background has now had time to subtract most of the artefacts that remain in constant in time across the beach. The variability of the wave intensity is clear now. Let us quickly reflect on how the runup looks in each of the views.

The Background of screen two:

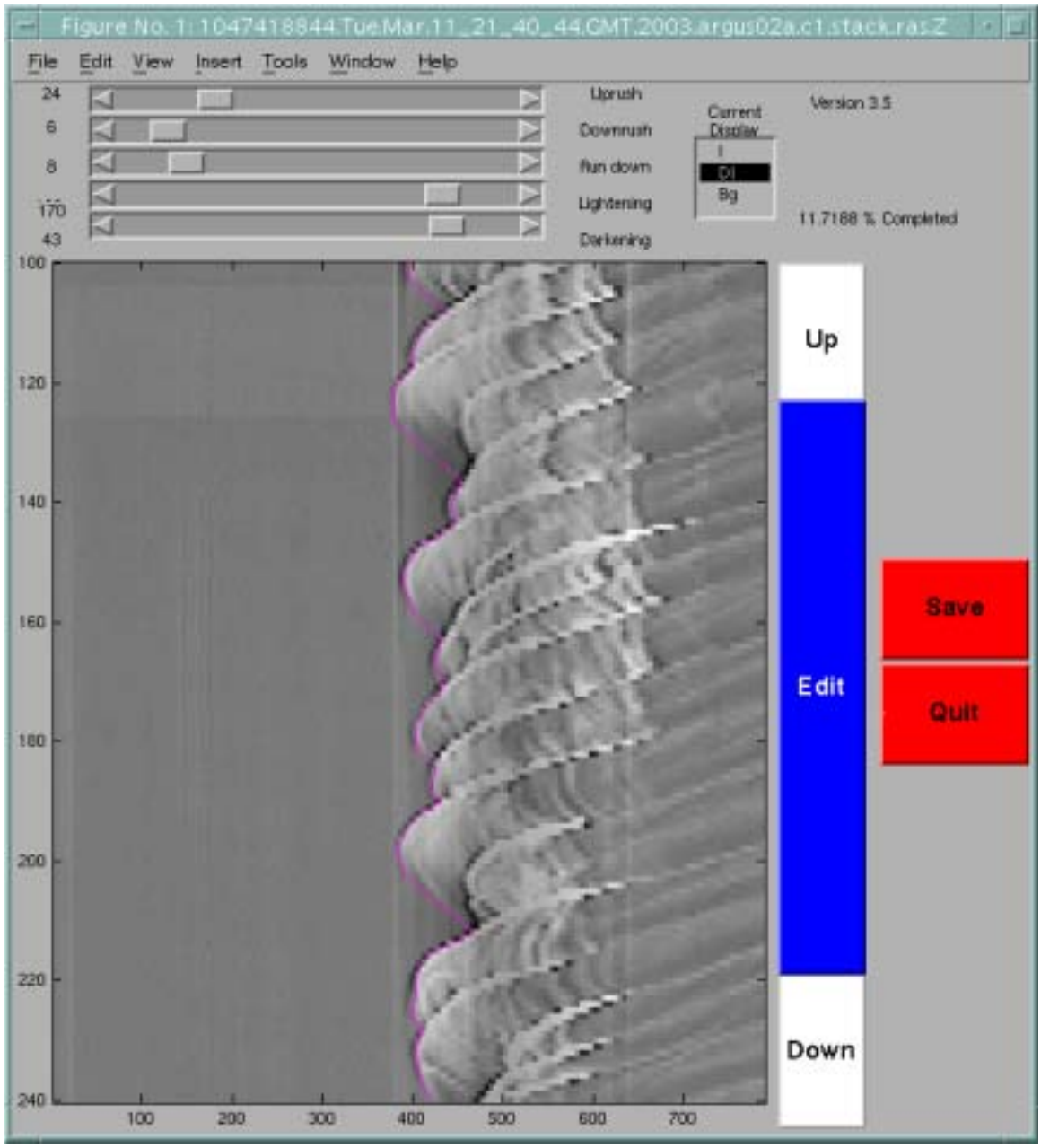


The DI of screen two:

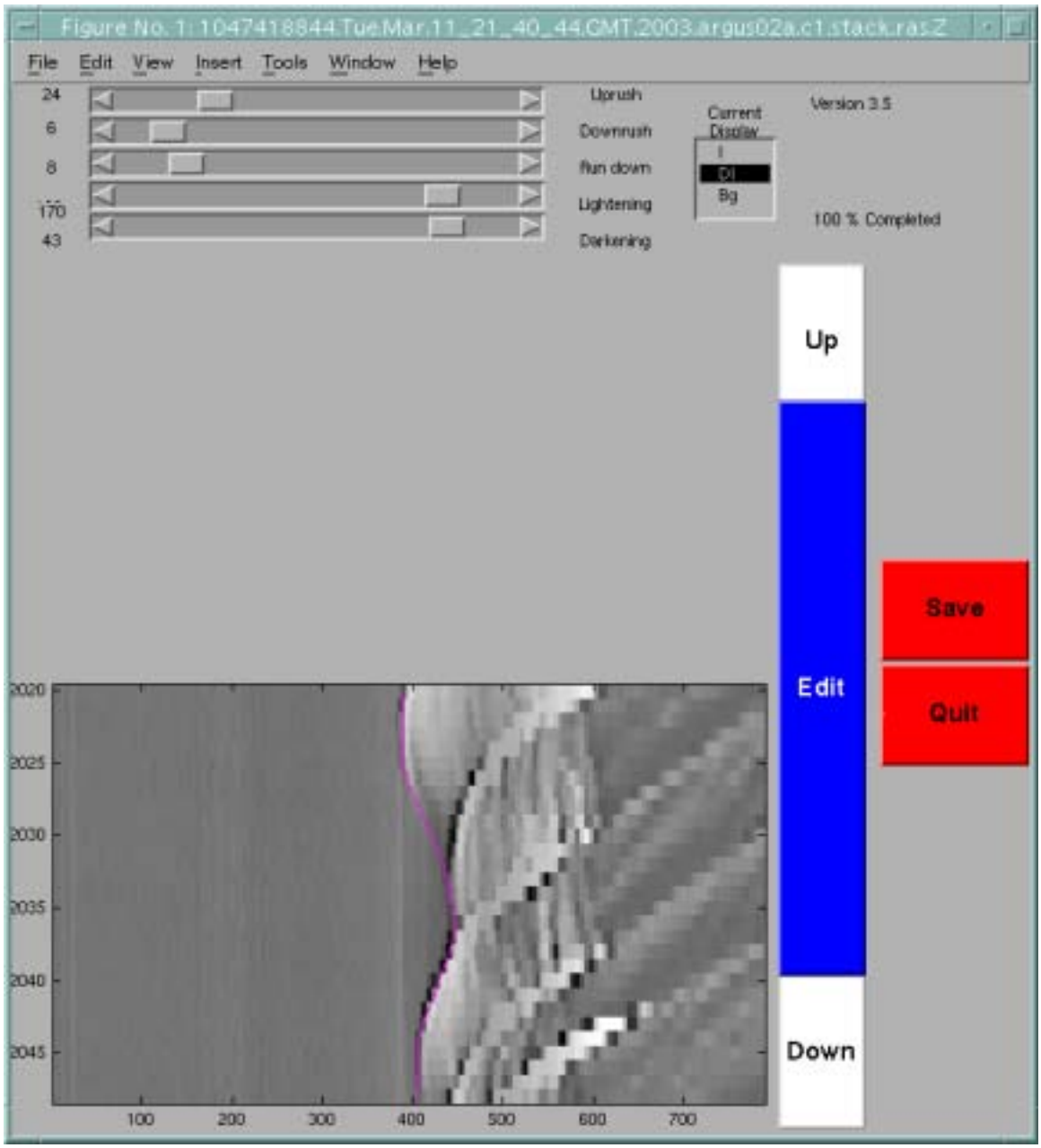


If you recall from the previous screen, the bottom 20 pixels are repeated in this screen, but are not editable. This feature is to maintain the Ri that is solved for in the next screen is a fluid continuation of the previous Ri solutions.

Upon careful inspection, one will notice that the tip of each of the waves is slightly overshot. To correct for this, pull back the **'Uprush'** slider until the trade-off is met between finding the edge and getting Ri "stuck" on the beach.



This process of editing continues until the end of the stack is reached. When this happens, and you are finished, press the **'Save'** button.



In the command window, this information will appear:

save file was ./1047418844.Tue.Mar.11_21_40_44.GMT.2003.argus02a.r583A.mat

runup =

```
        Ri: [2048x1 double]
    params: [1x1 struct]
     epoch: [2048x1 double]
       xyz: [2048x3 double]
        UV: [2048x2 double]
```

Notice how each of the matrices indices in rows match. If there have been any issues with processing the data, these numbers will not match.

Pressing the **'Quit'** button will return the user to the command window and close the figure. Good Luck using **runupTool**!