A tiger user manual

Rolf Maier

ABSTRACT

A description of the tiger modeling software, and notes on how to prepare input and handle output, has been added to the CREWES website. Tiger is a 3D finite-difference seismic modeling package, capable of producing reflection/transmission surface seismic or VSP data over isotropic or anisotropic targets.

INTRODUCTION

CREWES uses various packages of external software, several of which are described in short help files contained under the For Us section of our website. A full description of any major package is not within the scope of those short notes and requires reading the documentation. While such help may describe tags adequately, it does not substitute for a tutorial style introduction. The current manual is the first in an effort to create such documentation for the larger packages. Tiger itself does have a rather small tutorial since late which became available to us only on November 9th. It is very brief and does not specially emphasize important points, but it will be followed by more.

Tiger is a 3D modeling package from SINTEF, Norway. Tiger takes as input arbitrary isotropic or anisotropic 3D earth models. It can handle any acquisition setting, such as 2D seismic lines, VSP, or even ocean bottom data. The earth model is defined via a few arrays containing in the isotropic case the P-velocity, S-velocity, and density. It accepts Thomsen or Voigt anisotropy parameters to model anisotropic media. Multiple receiver lines, receivers in wells, or randomly placed receivers can be specified in 3D models. The minimum size of a model is the size of the operator plus two; the size is listed under the Options tab. A GUI, written in Java, is available to set up a job which is then run by a Fortran program. This GUI does stability checking of the input parameters.

While the GUI cannot create non-horizontal 3D models, it can read arbitrary models from external files. The same is true for acquisition geometry. The format of these files, seismic unix (SU) for large data sets like velocity fields, a density field, elastic parameters, or wavelets, or ASCII for small files like shot points or receiver lines, is not described in detail except for general information on SU available on the net, but the formats can be gleaned from the output created by an example available from the GUI. These formats are described in the HTML manual already referred to. Sample code to create velocity files in SU or SEG-Y format is also included in the HTML, allowing users to create models quickly.

The SU files created by tiger are virtually free of header information. It is worth creating files with more complete headers both for self-documentation and to enable processing, scaling, and plotting at a later stage.

Any new model, even if it has the same parameters as a previously run model, should be fed to the GUI to allow for stability checks. Changes like a different lowest velocity or a different wavelet can still cause important changes to how tiger runs. Stability checks

are, unfortunately, not a cure-all against failure since the Fortran compute engine depends on some additional conditions the GUI is not (yet) aware of. The result can be runs which the GUI will call successful, possibly hours later, but which leave no output beyond a job log.

The remainder of this report details some of the features of tiger and ways to interact with it, and includes a decimated plot of output based on a model used by Peter Manning in work contained in this year's collection of reports.

Basic workflow

Although the information obtainable in the manner described here is also available from the HTML file, this list gives an idea of how to get to know and use tiger. Tiger will prompt for a directory to work under; make one first. This is always good practice as tiger will overwrite files whenever a change is saved or submitted, as it says. It is important that every time a window is opened, a Submit must be entered if the window was opened the first time or if changes were made, otherwise tiger will not know that we are not only inspecting a window but wish its values to be kept.

First, get the Default Job under the File tab. Pick from the Filemanager which files are to be kept. Second, create a wavelet under the Wavelet tab. Ensure when constructing real models that the wavelet is fine enough to image your smallest structure. Third, go to the Regular geometry tab under Survey and define geometry. Fourth, run Stability test under Options. This goes a long way to ensure that the setup is sensible. Ignore the View tab (it has no close button but may be closed with xkill), and proceed to the simulation tab. Run all tabs below it until the Start tiger tab runs the job. Check on the progress by clicking on the last tab of this group. If nothing happens after several minutes then this may be a bad sign. Do not kill the little progress window as tiger would die with it.

At this point the format of the input files can be inspected. This is really only needed for SU files, the ASCII files may be easily defined from within the GUI unless patterns of receivers or shot movements change.

A look through the directories shows how tiger stores information. Some of this may come as a surprise: velocities and elastic parameters (Thomsen or Voigt) parameters were all contained in a small list when the default job gets started, but tiger is far more flexible than that and hence creates files the size of the model for many physical parameters. As stated before, you can deselect them to shorten the runtime and memory requirements.

Memory requirement exceeds the total size of all input files. There are two reasons for this. First, tiger will pad a model, first to make it as large as the source and receiver arrays should this not be so, and second to implement boundary conditions. The latter depends on what kind of input tiger gets. The more complicated the model is in terms of kinds of information given or expected, the more boundary layers will be added, everywhere from a few to a few dozen. Details are once again available via a sub-tab of the Options tab. Second, there are several copies of the model stored in memory. These are not just the multiple files containing velocities or elastic parameters. As a wave marches through the model, the states of various parameters pertaining to the wave must

be stored, requiring more space, one 3D array for each component. The GUI tells how much space a model requires.

Description of the model

The model, implemented on a grid representing a 2 km long and 700 m deep section, with 3 m node spacing, consists of five horizontal layers with five thin low-velocity pockets at the top, and a more substantial channel in the middle of the model at 400 to 500 m depth, shaped like a diamond laying on its side. The layers' velocities range from 2000 to 4500 m/s, with 1400 m/s for the low velocity pockets.

Output

The output, sampled at 1 ms, is produced in SU format. Source and receiver coordinates are defined, together with a few other important values a program might expect, these being the source and receiver elevation, the sample interval and number of samples per trace, and the trace sequential number and gather number. The output data consist of 667 traces at 1000 samples.

Since synthetic data do not have shot gaps, the tops of the traces next to the source point overpower any other data; the gathers need to be scaled or clipped to show anything. The wiggle trace plot at the end shows only every 7^{th} trace in order to stay clear of resolution limits without making the lines so thin that they will be aliased on monitors. The plots shown below all show the distance from the first receiver station in meters on the horizontal axis, and the time in seconds on the vertical.

The first figure shows all traces in gray scale, with a clip (in supsimage) determined by 80 % of the data. The contrasts between events are rather strong, but reflections from the horizontal interfaces, particularly the first one, are visible, as are the reflections from the low velocity pocket boundaries. Also noticeable in this plot are areas of apparent smear due to the low velocity pockets. The wobbly first breaks are showing more clearly in figure 2.

Figure 2 displays only every 7th trace. The effect of the low velocity layers on the first breaks becomes more readily visible, as do reflections which look like they are due to a slanted interface but must be the result of a reflection of ray a hitting first a low/high velocity boundary near the surface, then an interface.

Figure 3 is the only variable area / wiggle plot in this collection. Unfortunately it shows poorly on monitors, but a printed version reveals more clearly than figure 2 two traces at the centre which, after a spike, become straight lines. The reason for this is not presently known. A hardcopy plot also reveals expected numerical jitter towards the bottom.

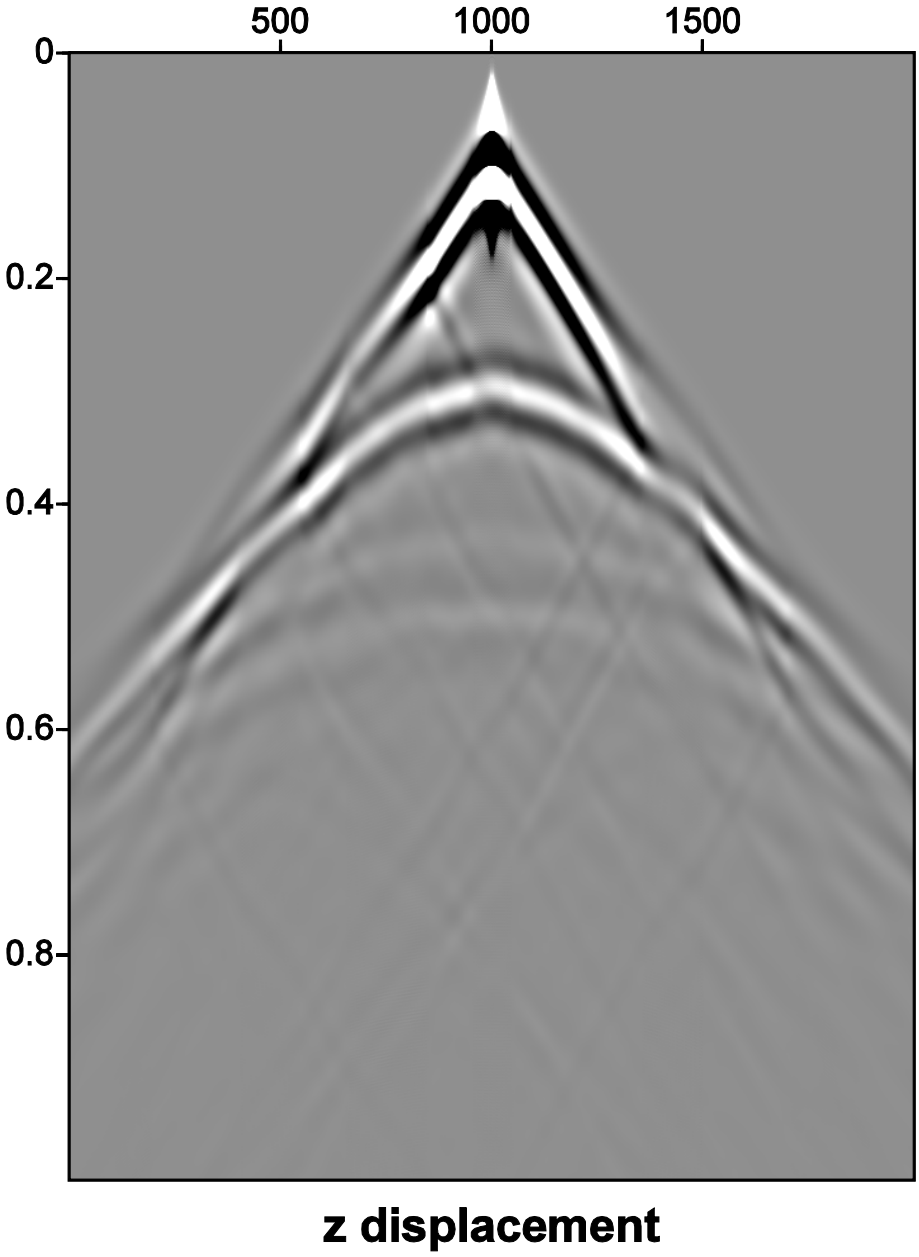

FIG. 1. Plot of all traces of one shot, showing alternating clear and grayed out first breaks due to low velocity pockets, and straight lines due to the vertical boundaries of the low velocity pockets.

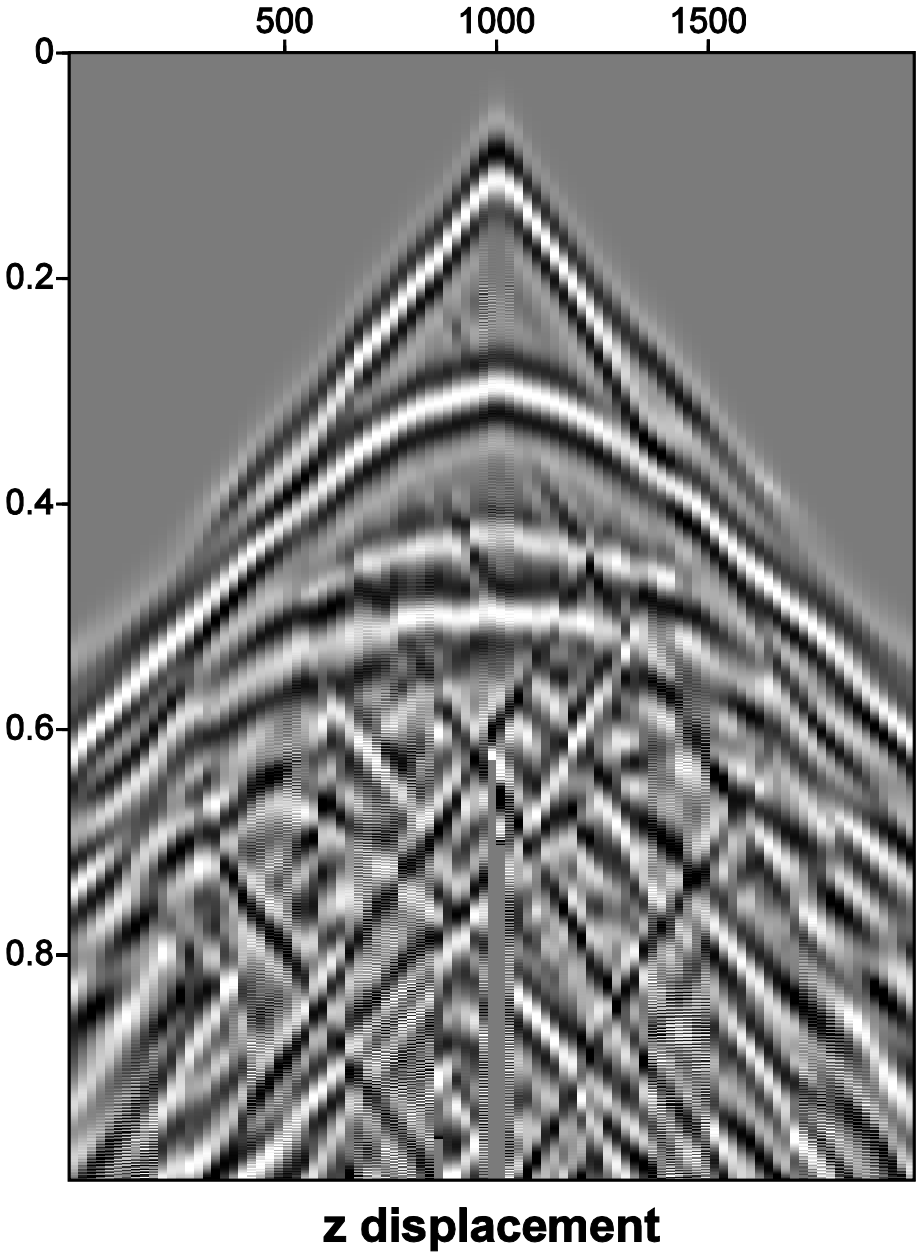

FIG 2. The same data displaying only every 7th trace. The effect of the low velocity layers on the first breaks becomes more readily visible, as do reflections which look as if due to a slanted interface but must be the result of a reflection of ray a hitting first a low/high velocity boundary near the surface, then an interface.

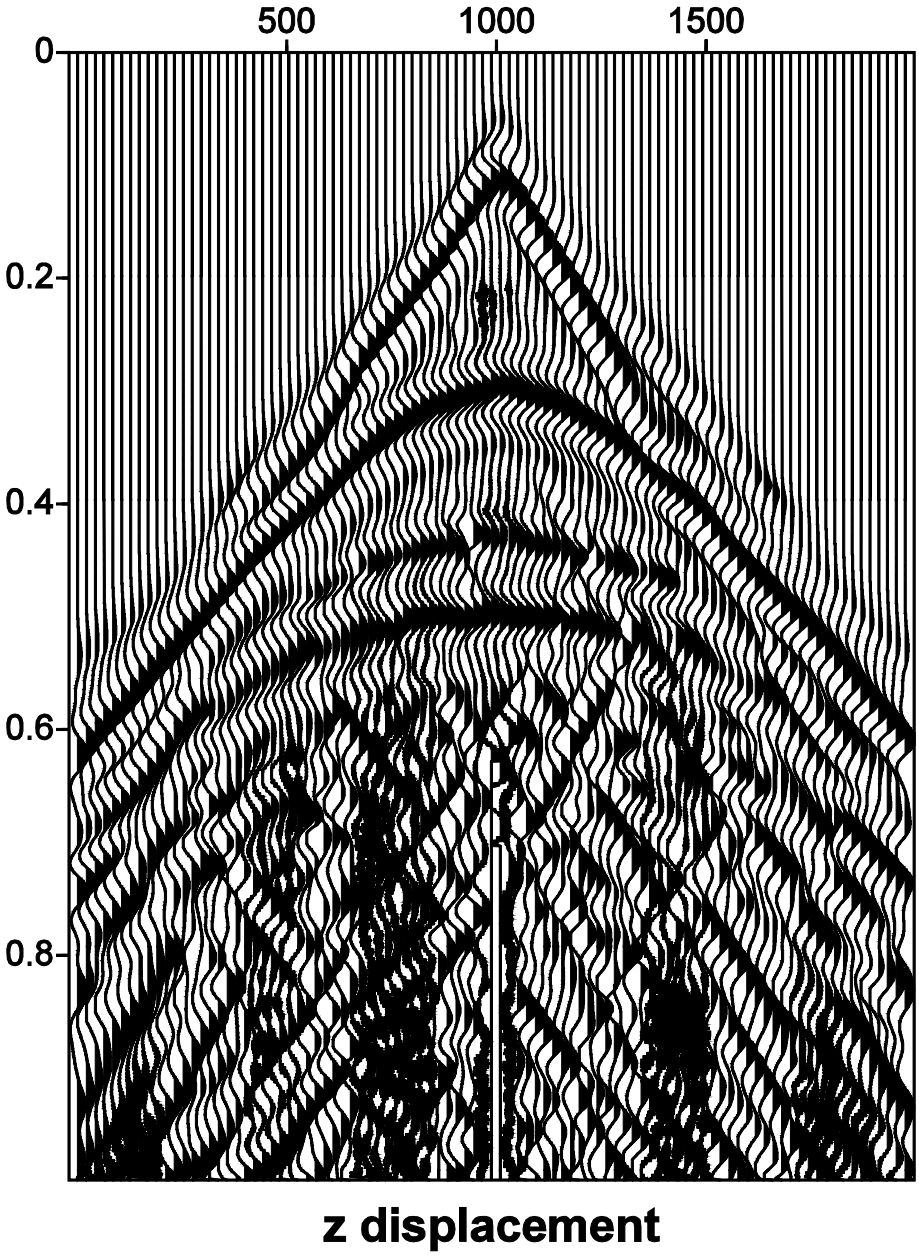

FIG 3. This variable area / wiggle plot shows poorly on monitors, but a printed version reveals two traces at the centre which, after a spike, become straight lines. A hardcopy plot also reveals expected numerical jitter towards the bottom.