GeoConvention
Calgary • Canada • March 7-11 2016
OPTIMIZING RESOURCES

Efficient geophysical research in Julia

Aaron Stanton and Mauricio D. Sacchi

Department of Physics, University of Alberta

Summary

A new programming language for scientific computing has undergone rapid development since 2012. The language is named Julia– perhaps a reference to the beautiful fractal patterns of the Julia set. Julia is a high level programming language with an extensive library of mathematical functions that is easy to code and share with others. However, unlike other high level languages it offers C-like performance. It is an open source language with a large community of users and developers with a built-in package manager. The Signal Analysis and Imaging Group (SAIG) has recently released a seismic data processing package named *Seismic.jl* that contains utilities for reading and manipulating seismic data. We believe Julia is a great new language for research and teaching in the geosciences.

Introduction

Geophysical programming typically falls into one of two categories: small scale prototyping for teaching or research using high level languages (e.g. Python, Matlab, or Mathematica) or large scale implementations for production applications or large numerical experiments using statically compiled languages (e.g. C, C++, Fortran, or Java). Julia offers a bridge between these two categories. In Julia we have the ability to quickly create prototypes that can also solve larger problems efficiently. It does this by using a sophisticated type system and multiple dispatch (specified function behaviour for various combinations of argument types). The language design allows Julia's just-in-time (JIT) compiler to execute programs at impressive speeds. Below is a table provided by the creators of Julia that benchmarks the performance of several popular languages for various algorithms.

	Fortran	Julia	Python	R	Matlab	Octave	Mathematica	JavaScript	Go	LuaJIT	Java
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

Table 1 *Benchmark times relative to C (smaller is better, C performance = 1.0). Adapted from http://julialang.org. Please refer to the original for technical specifics.*

Some features of the language that we find particularly useful for geophysical applications are its ability to execute multi-level *for loops* without the performance penalty of other high level languages (eg. Python, and Matlab). The language also allows for C and Python libraries to be called directly with no overhead, and has functionality for large scale parallel computing.

Seismic.jl

The *Seismic.jl* package can be installed from the Julia command line by typing *Pkg.add("Seismic")*. The package contains utilities for reading and writing SEG-Y, SU (Seismic Unix), and RSF (Madagascar) data formats, and uses a simple internal format to store data where headers and data are

kept in separate files with the extensions *.seisd* and *.seish*. By storing data and headers separately we can efficiently compute header statistics and manipulate data based on header values. The type system in Julia allowed us to create a custom Header type that contains many useful indices for 5D processing and imaging.

Seismic.jl contains modules for many conventional data manipulations including windowing, sorting, 5D geometry calculation, 5D binning, as well as wrapper modules for processing groups or patches of data in parallel. The processing functionality of the package is a work in progress, but already includes semblance, NMO, band-pass and FK filtering, FX-deconvolution, Radon demultiple, 5D interpolation, stacking, and 3D shot-profile acoustic and elastic wave equation migration, de-migration and least squares migration.

Examples

To try these examples yourself we ask the interested reader to visit `http://juliabox.org` where a linux virtual machine can be accessed in the browser for no cost. Below we show a simple seismic data interpolation exercise using the Projection Onto Convex Sets (POCS) algorithm (Abma and Kabir, 2006). The example can be reproduced by following `https://goo.gl/gYtWZs`. To create a simple synthetic dataset to test the algorithm we use the *SeisLinearEvents* program and decimate 50% of the traces randomly. Figure 1 shows the decimated input data and its FK-spectrum. The data are reconstructed using 100 iterations of POCS with results shown in figure 2.

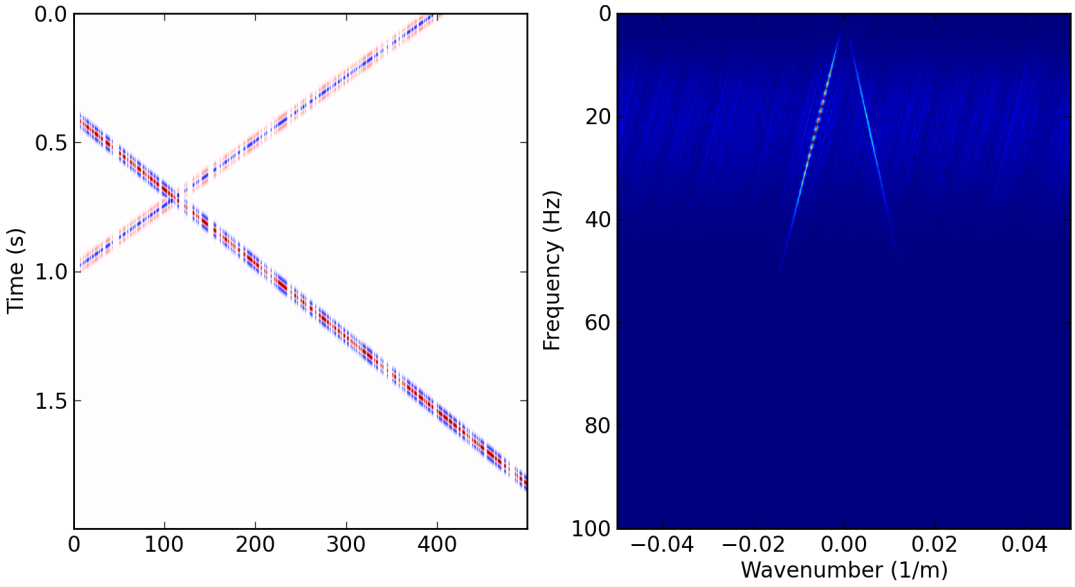

Figure 1 *decimated seismic data used as input to the SeisPOCS program. The data have 50% missing traces.*

As a second example we compute reflector dip for a velocity model using Plane Wave Destruction (PWD) (Claerbout, 1992). In figure 3 the velocity model was used as an input to the program SeisPWD, with computed reflector normal directions plotted with arrows throughout the model. This example can be reproduced by following `https://goo.gl/zu40wg`.

Conclusion

Julia offers the unique ability to write simple prototypes for geophysical research and teaching that can also scale to solve large problems efficiently. We have introduced a set of utilities for writing and

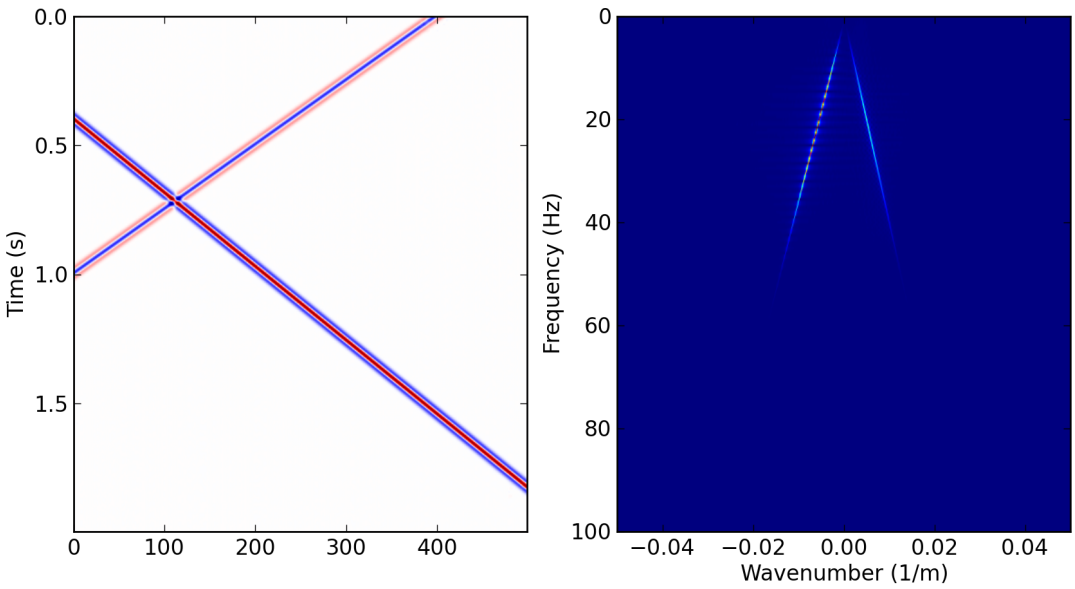

Figure 2 *Seismic data interpolated using the SeisPOCS program.*

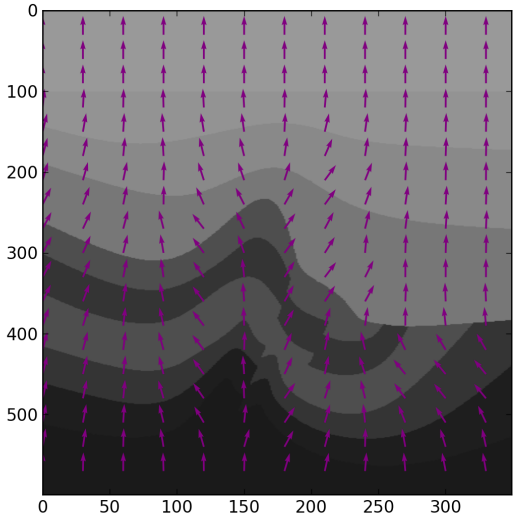

Figure 3 *A velocity model used to demonstrate the program SeisPWD. Computed normal vectors are plotted as purple arrows.*

manipulating seismic data in the Julia language.

Acknowledgements

We thank the creators of Julia and the sponsors of Signal Analysis and Imaging Group (SAIG) at the University of Alberta.

References

Abma, R. and N. Kabir, 2006, 3d interpolation of irregular data with a pocs algorithm: Geophysics, **71**, E91–E97.
Claerbout, J. F., 1992, Earth soundings analysis: Processing versus inversion, volume **6**: Blackwell Scientific Publications Cambridge, Massachusetts, USA.