# Efficient Geophysical Research in Julia

Aaron Stanton, Mauricio D. Sacchi and Nasser Kazemi

EAGE open source workshop (Vienna 2016)

SIGNAL
ANALYSIS &
IMAGING GROUP

# Outline

- Motivation
- Julia
- *Seismic.jl*
- Examples
- Conclusions

# Motivation

- Matlab is a great prototyping language
  - easy to understand, compact code
  - many built in functions for linear algebra and plotting
- But…
  - inefficient for large problems
  - not open source
- Compiled languages are great for implementation
  - efficient for large problems
  - support all forms of parallelism
- But…
  - difficult to follow
  - inefficient to code and debug

# Motivation

- Julia presents a new opportunity with Matlab-like syntax, C-like performance, and is free and open source
- This allows SAIG to make efficient, easy to read prototypes for our sponsors that can be readily implemented in their systems

# Julia

- Released in 2012
- Completely open source with MIT license
- Just-In-Time compiler
  - compiled after execution begins
  - has access to more information than static compilers
- Sophisticated type system
- Multiple dispatch
- Rich library of linear algebra and plotting functions
- Distributed and shared memory parallelism
- Pass-by-reference (pointers)
- Directly call C-libraries with no overhead

# Julia's Motivations

*We want a language that's open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that's homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.*

*(Did we mention it should be as fast as C?)*

# Matlab-like syntax

## Matlab

```
y = zeros(nx1,nx2,nx3,nx4);
for iw = 1 : nw
      y(:) = d(iw,:)
      x = y
      for iter = 1 : Niter
            Y = fftn(y);
            Y = Threshold(Y,p(iter));
            y = ifftn(Y);
            y = a*x + (1 - a*T).*y
      end
      d(iw,:) = y(:)
end
```

## Julia

```
y = zeros(nx1,nx2,nx3,nx4);
for iw = 1 : nw
      y[:] = d[iw,:]
      x = copy(y)
      for iter = 1 : Niter
            Y = fft(y);
            Y = Threshold(Y,p[iter]);
            y = ifft(Y);
            y = a*x + (1 - a*T).*y
      end
      d[iw,:] = y[:]
end
```

# Example: For-Loops

- Multi-level for-loop:

```
a = 0.0
for j1=1:300;
        for j2 =1:300;
                for j3=1:300;
                        for j4=300;
                                a += 1.0;
                        end
                end
        end
end
```

@time include("foo.jl")

1.368727 seconds
(27.01 M allocations:
412.291 MB)

- Matlab: 59.266124 seconds
- Julia: 1.368727 seconds

# Example: For-Loops

- Multi-level for-loop:

```
function foo()
a = 0.0
for j1=1:300;
        for j2 =1:300;
                for j3=1:300;
                        for j4=300;
                                a += 1.0;
                        end
                end
        end
end
end
```

@time foo()

0.026339 seconds
(6.97 k allocations:
326.946 KB)

- Matlab: 59.266124 seconds
- Julia: 0.026339 seconds

# Example: Calling a C library

```
t1 = ccall( (:time, "libc"), Int32, ())
a = 0
for i = 1 : 1e4
      for j = 1 : 1e4
            a += 1
      end
end
t2 = ccall( (:time, "libc"), Int32, ())

println("This program took ",t2 - t1," seconds.")
```

**OUTPUT:** **This program took 6 seconds.**

# Example: Parallelism

```
a = @parallel (+) for i=1:1e6
      randn()
end
```

**addprocs(20); foo();**

**julia -p 20 foo.jl**

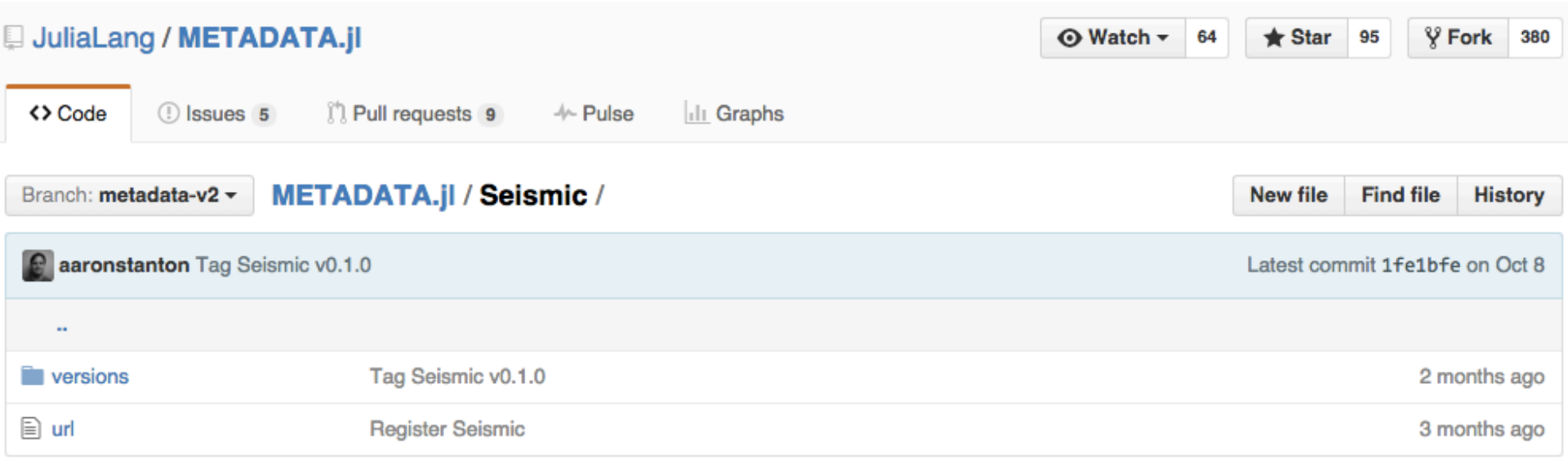
**julia --machinefile nodes.txt foo.jl**

**omplace -nt 12 julia --machinefile nodes.txt foo.jl**

# Seismic.jl

# Installation

Pkg.add(“Seismic”)


JuliaLang / METADATA.jl
Watch 64
Star 95
Fork 380
Code
Issues 5
Pull requests 9
Pulse
Graphs
Branch: metadata-v2
METADATA.jl / Seismic /
New file
Find file
History
aaronstanton Tag Seismic v0.1.0
Latest commit 1fe1bfe on Oct 8
..
versions
Tag Seismic v0.1.0
2 months ago
url
Register Seismic
3 months ago

SeismicJulia / **Seismic.jl**

Unwatch 8 | Star 19 | Fork 13

Code | Issues 0 | Pull requests 0 | Wiki | Pulse | Graphs

Julia tools to read, write and process seismic data

**143** commits | **1** branch | **6** releases | **6** contributors

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

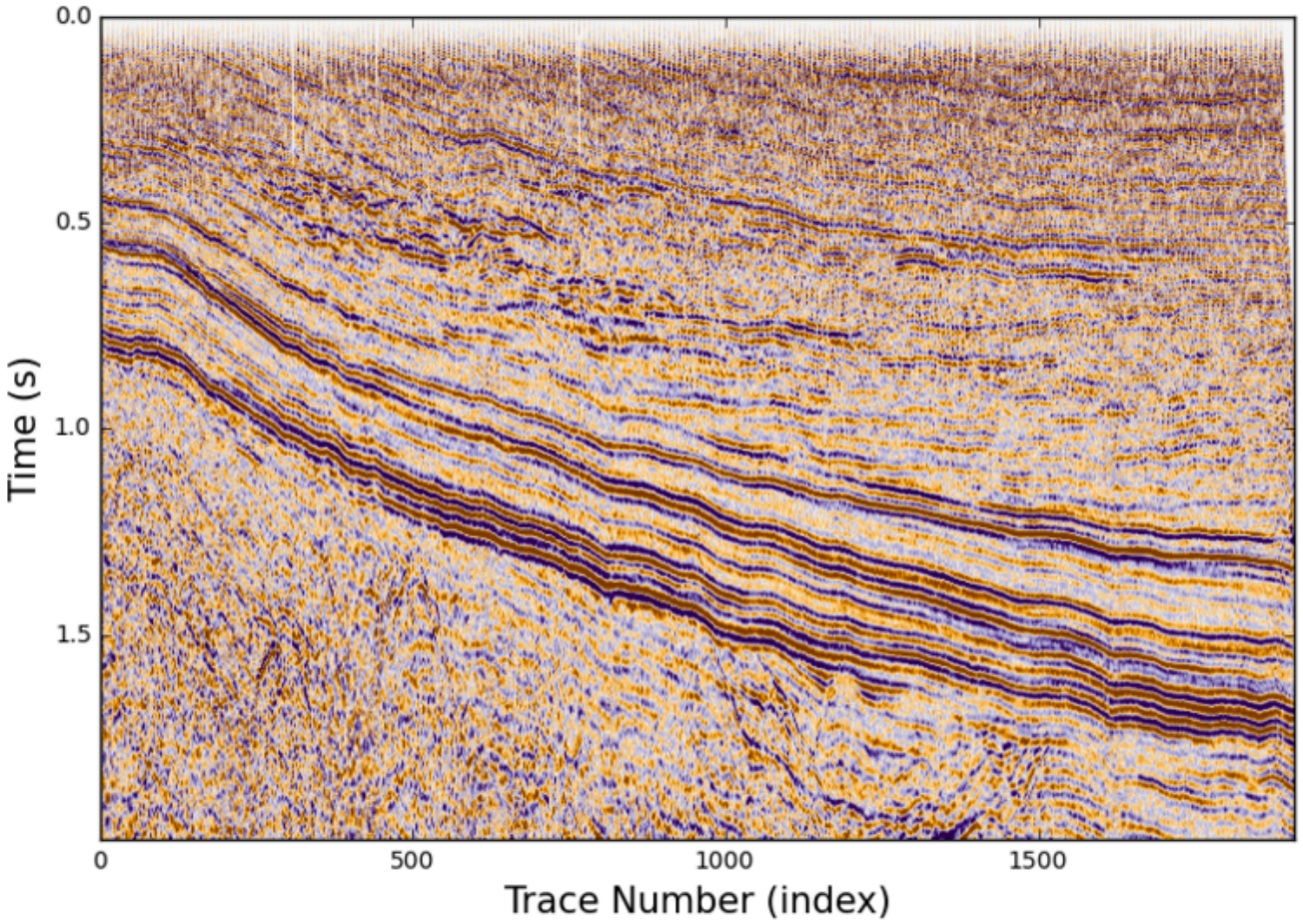
**juanisabbione** Two tests added (#22) ... Latest commit `b4470e6` 16 hours ago

| | | |
|---|---|---|
| docs | tag new version | 3 months ago |
| src | Update wavelets (#21) | 5 days ago |
| test | Two tests added (#22) | 16 hours ago |
| .gitignore | tag new version | 3 months ago |
| .travis.yml | Two tests added (#22) | 16 hours ago |
| LICENSE | tag new version | 3 months ago |
| Modifications.md | tag new version | 3 months ago |
| README.md | tag new version | 3 months ago |
| REQUIRE | tag new version | 3 months ago |
| mkdocs.yml | tag new version | 3 months ago |

# Basic usage

```
using PyPlot,Seismic;

download("http://certmapper.cr.usgs.gov/nersl/NPRA/seismic/
1979/616_79/PROCESSED/616_79_PR.SGY","616_79_PR.SGY");

SegyToSeis("616_79_PR.SGY","616_79_PR.seis");

d,h,e = SeisRead("616_79_PR.seis");

SeisPlot(d[1:500,:],e,cmap="PuOr",wbox=9);
```

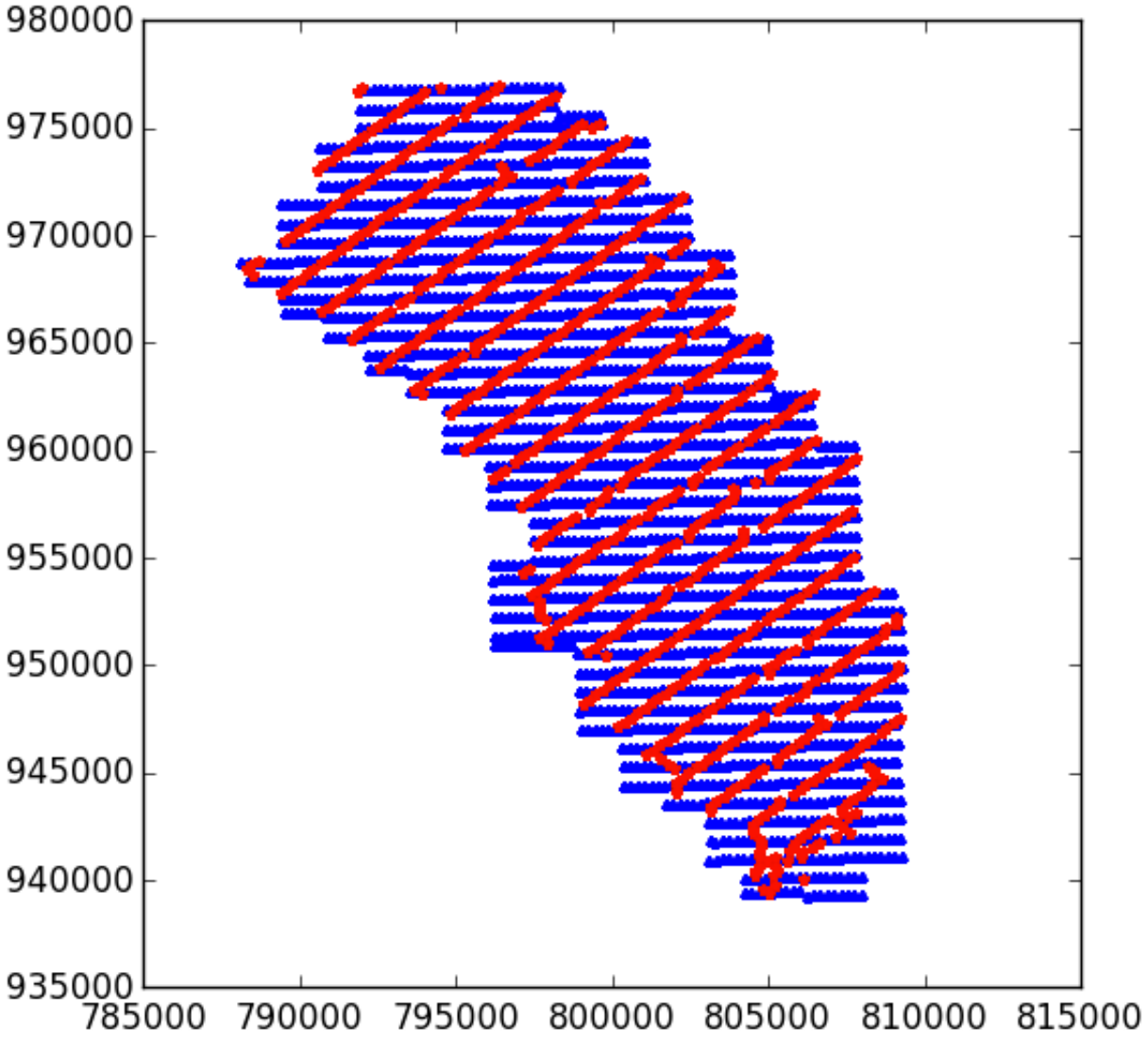
# Basic usage



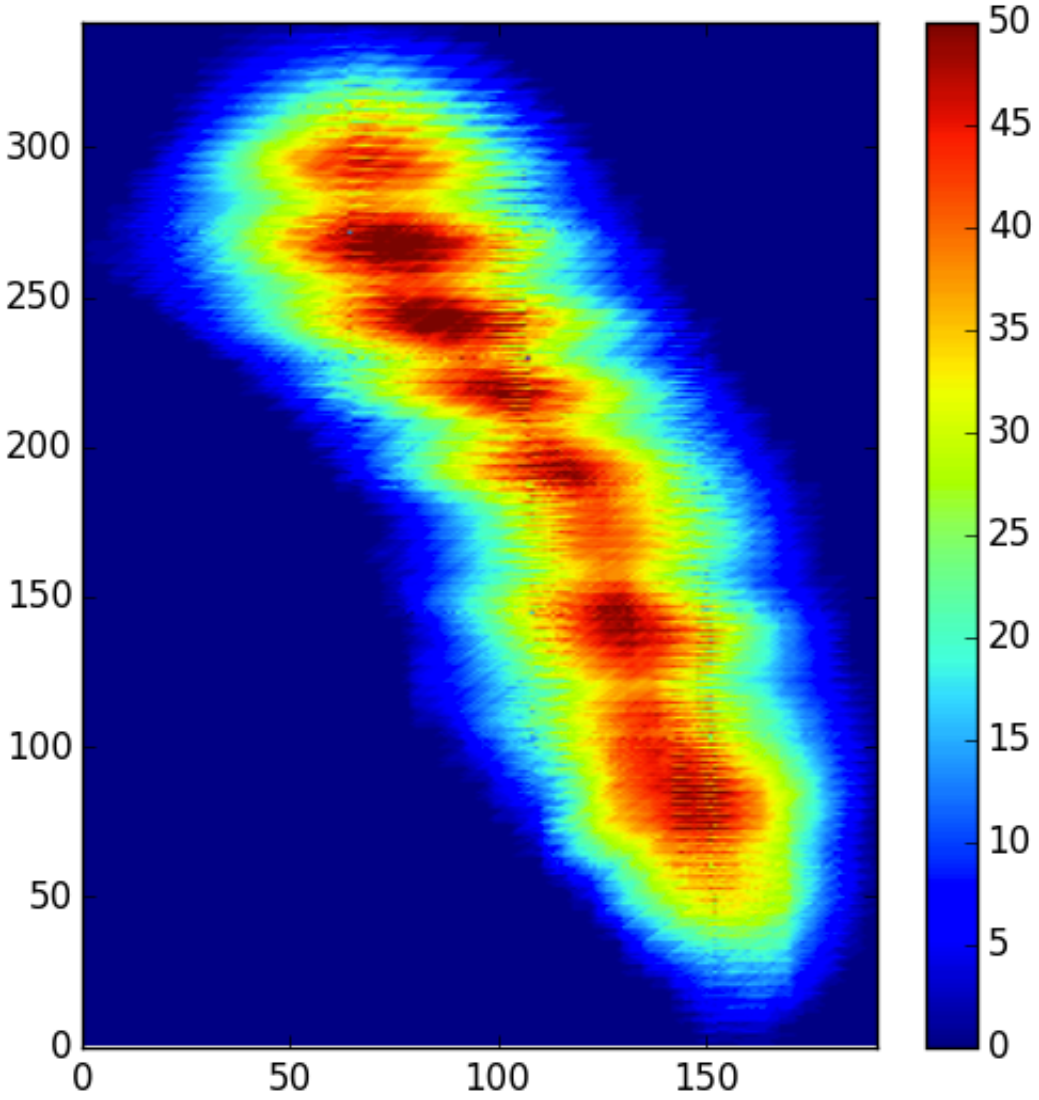
# Basic usage

```
using PyPlot,Seismic;

download("http://s3.amazonaws.com/teapot/
npr3_gathers.sgy","npr3_gathers.sgy")

SegyToSeis("npr3_gathers.sgy","npr3_gathers.seis");

SeisGeometry("npr3_gathers.seis",ang=90,
                    omx=788937,omy=938845,
                    dmx=110,dmy=110,oh=0,
                    oaz=0,dh=420,daz=45)

h = SeisReadHeaders("npr3_gathers.seis")

im1 = SeisPlotCoordinates(h,style="sxsygxgy")
im2 = SeisPlotCoordinates(h,style="fold",cmap="jet",vmin=0,vmax=50,
                                  aspect="auto",xlabel="imx",ylabel="imy")
```

# Basic usage


980000
975000
970000
965000
960000
955000
950000
945000
940000
935000
785000
790000
795000
800000
805000
810000
815000

# Basic usage



# Data format

- The .seis file format is a clone of madagascar and SEPLIB's format (.rsf)
- a .seis file is just a text file with information and file paths to the data and the headers
- Environmental variable DATAPATH can be set to store your data and headers in a particular directory, but the default is the current directory
- $N_{traces}$ x $N_{samples}$ of Binary IEEE floats

# Header format

- For each trace the header is stored as type *Header*
- $N_{traces}$ x 31 Binary IEEE floats/integers
- To see all fields type “names(Header)”

# Package Contents

- Utilities
- Processing
- Imaging
- Solvers

# Utilities

- Conversion to/from SEGY, SU, and RSF formats
- Reading and writing of internal format
- Windowing
- Geometry
- Header statistics
- Sorting
- Binning
- Patching/UnPatching
- Processing on groups of traces

# Processing

- Semblance
- NMO
- Mute
- Stack
- Bandpass and FK fan filtering
- Radon transform
- 5D interpolation
- Structural Smoothing
- FX deconvolution
- Plane wave destruction for dip estimation
- …

# Imaging

- Pre-stack time migration
- Wave equation migration, modeling, and least squares migration

# Solvers

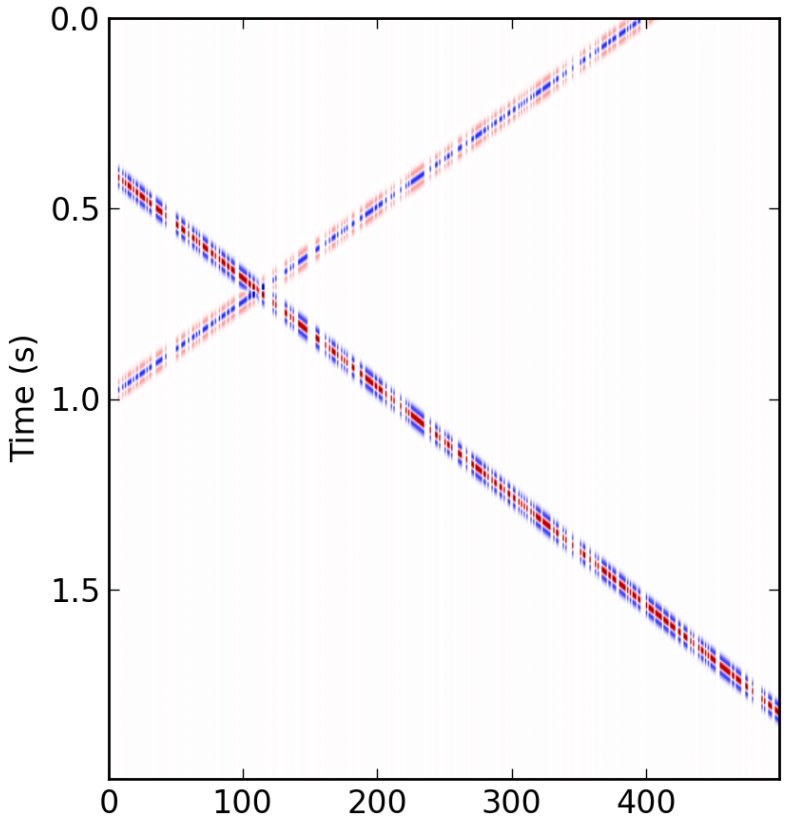
- Dot product test
- Memory or disk based preconditioned conjugate gradients
- Iteratively Reweighted Least Squares (IRLS)
- Fast Iterative Soft Thresholding Algorithm (FISTA)

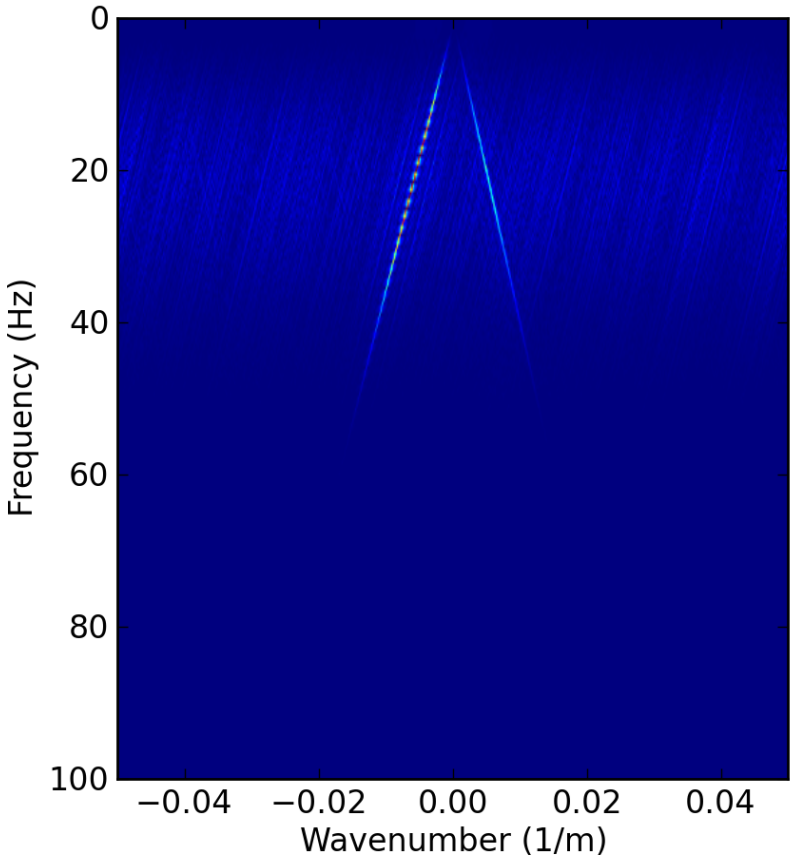
# Examples

# Calling POCS Interpolation

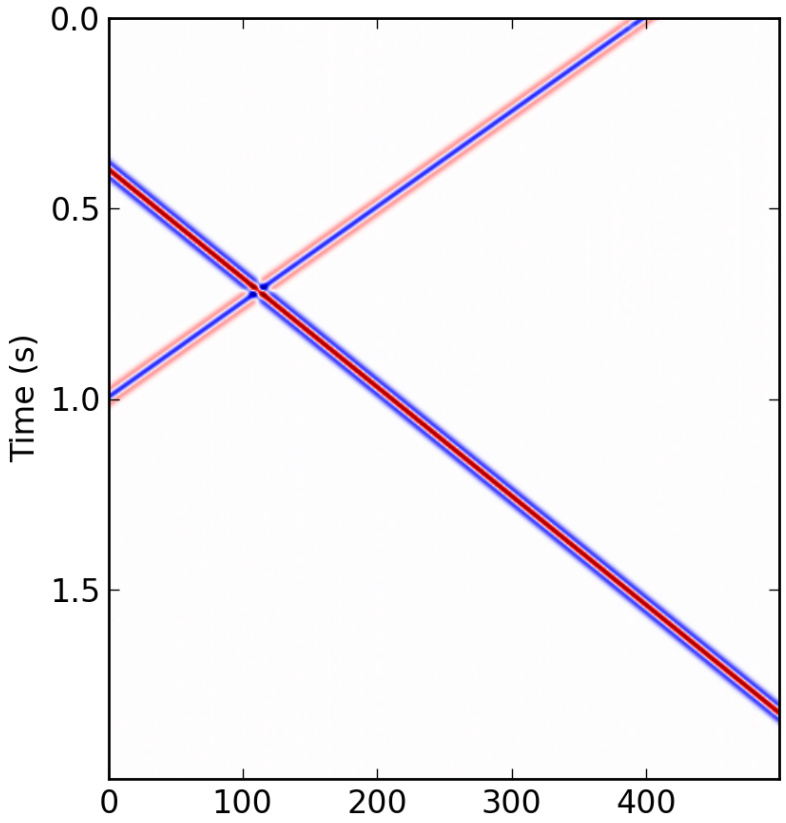
```
using Seismic
dpocs = Seismic.pocs(ddec,Niter=100,fmax=60,dt=0.004);
```

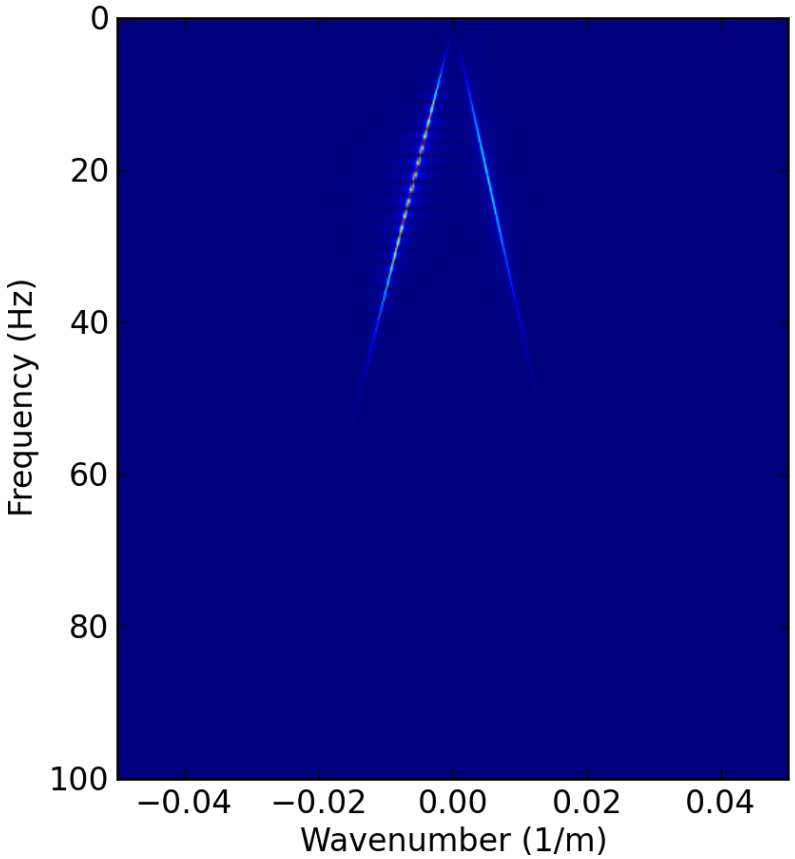
# Interpolation input



# Interpolation output

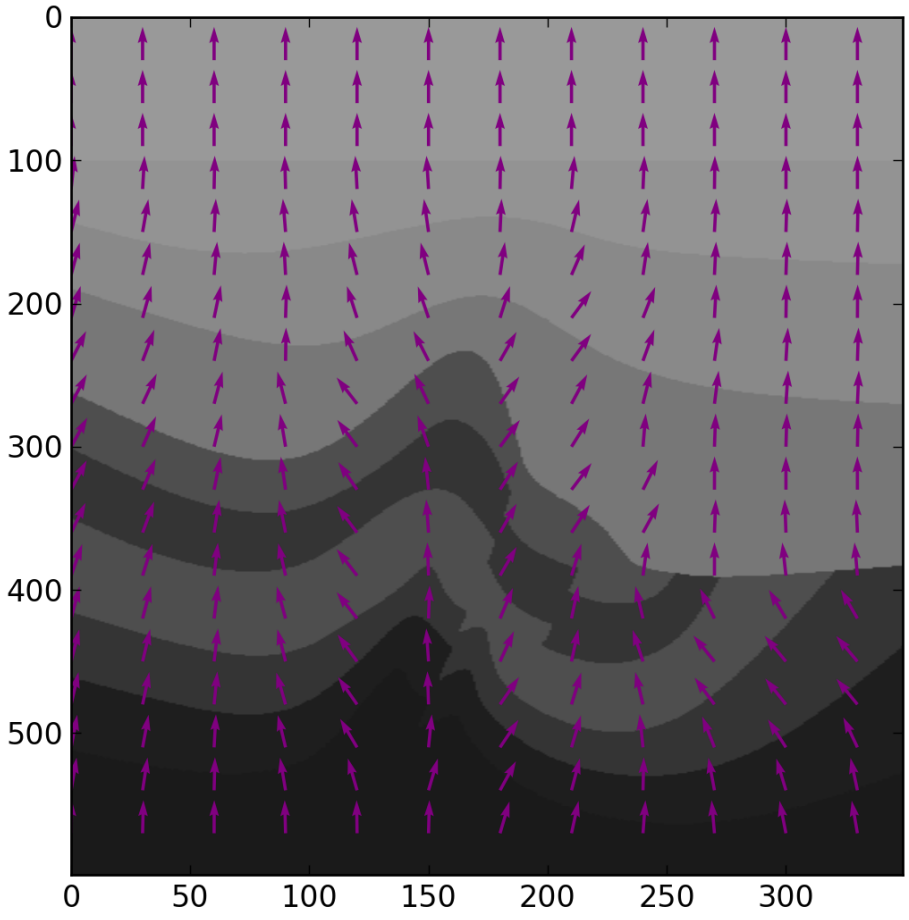


# Calling Plane Wave Destruction

```
using Seismic
download("http://ualberta.ca/~kstanton/files/
kevin_0off_velocity.su","kevin_0off_velocity.su");
SegyToSeis("kevin_0off_velocity.su","kevins_model",{"format"=>"su"});
d,h,e = SeisRead("kevins_model");
coh,pp,res = SeisPWD(d,w1=30,w2=20,dx=8,dz=4);
```

# Estimated reflector normals



# Conclusions

- Julia can be used for fast prototyping as well as large scale production
- Julia programs are easy to follow for implementation in other languages
- We developed a package in Julia that is part of the ecosystem of Julia packages: Pkg.add("Seismic")
  - Utilities
  - Processing
  - Imaging
  - Inversion
- Additional closed-source packages are available to SAIG sponsors

# References

www.julialang.org

www.seismic.physics.ualberta.ca

# Acknowledgements


- Thanks to the creators of Julia and to Sam Kaplan for introducing us to Julia

- Thanks to the sponsors of the Signal Analysis and Imaging Group for their generous support

# Contact us:

kstanton@ualberta.ca

msacchi@ualberta.ca

kazemino@ualberta.ca