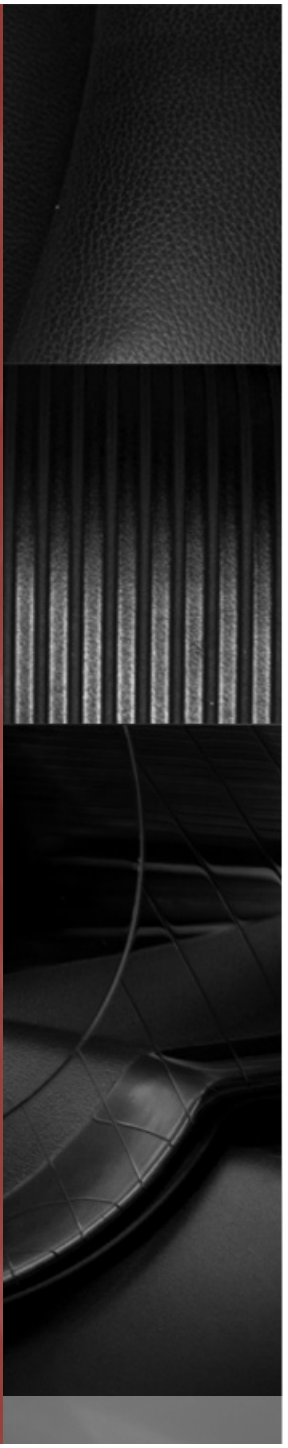
Wphase development at ATWS

Reliable magnitudes for EQs > 8.0



Magnitude Calculator @ ATWS

- Mb – good to <= 6.5.
- Ml – good to <= 6.5 in Australia.
- Msx – rapid solution good to <= 7.5.
- AutoMwp – good to ~ 8.0 (currently disabled).
- Manual Mwp – good to ~ 8.0 (underestimates by ~0.1).
- What about EQs > 8.0?
- CMT – slow, unreliable, very difficult to maintain.
- Wphase may be the future.

Wphase implementation options

- Original Kanamori(?) in C/Fortran
  - License/usage conditions unclear, as is support.
  - May come with similar maintainability issues as with CMT.
  - Only reads inputs from disk in particular format, integration difficulties.
- New implementation by Roberto Benavente at ANU
  - Written entirely in Python, easier to maintain and modify.
  - Takes data as Obspy Trace object – can come from disk, memory or Numpy array.
  - Benefit to ATWS is we can work closely with author.
  - Code still in development to some degree, not mature, still “research level”.
  - And did I mention it is written in Python? Does this rule out evproc etc?

Wphase implementation options (cont)

- Ultimately we decided to pursue Roberto at ANU's Wphase.
- Benefits of collaboration, direct support from author, Python implementation were main factors in decision.
- Also, being written in Python allows greater ease of exploration by scientists (Spiro, Phil Cummins etc) who may not be comfortable with C/Fortran.
- My job to work with Roberto to integrate his code with Antelope somehow.
- Challenge: to attempt some integration with orbevproc, or standalone module.
- Prototype has been written using orbevproc, which may or may not be driver used in final inversion.

Wphase and Evproc. How?

- Evproc (orbevproc/dbevproc) existing network magnitude calculator.
- Not an inversion engine, but maybe can be shoehorned into the job.
- Bigger problem: evproc written in C and embeds Perl interpreter.
- But Wphase is written in Python.
- How can they be integrated together?
- Options
  - Write out waveform database, call Wphase as executable. Ugly, ugly, ugly.
  - Rewrite evproc to embed Python interpreter. Hard, and maybe BRTT already doing that.
  - Is there a third option? YES – next slide.

Perl module Inline::Python

- Perl module that embeds the Python interpreter in Perl.
- Can easily call Python functions from Perl.
- And Python can “call back” into Perl.
- Data structures are automagically marshalled (transformed), including arbitrary and deeply nested data structures (hashes of arrays of hashes).
- OO method calls are transparently converted into correct invocation.
- “Objects” survive being sent from Perl to Python and back again.
- In theory, it might allow Python Wphase to be called from evproc Perl.
- But we would be embedding Python in Perl in C – pushing the limits?

Perl module Inline::Python (cont)

- Example calling Python from Perl:

```
use Inline Python => <<'EOP';
import sys, os
sys.path.append(os.environ['EVPROCPATH'])
from atws.wphase import  Py_process_channel
EOP
sub process_channel {
   my ($self,$dbtrace,$flush) = @_;
   my ($obj,$disp,$res) = Py_process_channel({%$self},$dbtrace,$flush);
   %$self = (%$obj);
   my @ex = defined $res ? (%$res) : ();
   makereturn($self,$disp,@ex);
}
```

Perl module Inline::Python (cont)

- Example calling Perl from Python:

```
package main; # This ensures all imported values can be seen by Python
use Response;
--------------------------------
Import perl
def response_poles_zeros(path):
   r = perl.get_response_from_file(path)
   groups = r.groups() # Note the OO calls on a Python encapsulated Perl object (r).
   if not isinstance(groups,list):
      groups = [ groups ]
   if groups[0].id() != "PAZ":
      return
```

Perl module Inline::Python (cont)

- Able to call stuff from Perl not available yet in Python bindings.
- This gives added benefit that would not be realized in a stand-alone Python script.
- Hence technique may be more generally useful, reduces pressure to extend Python bindings immediately.
- The whole idea sounds a bit crazy, but appears to work extremely well.
- No performance issues apparent (although could emerge if not careful).
- Inline::Python is old (read stable), widely used and reliable.
- Could be magic ingredient to solve evproc/Wphase integration problem.

Evproc and Wphase

- Roberto’s original Wphase code was tied deeply to his setup, which involved downloading traces from IRIS and reading from disk.
- To aid us, he rewrote the core inversion as a modular function with no other dependencies on his setup.
- As arguments receives station and event metadata, response data (poles and zeroes) and the waveform data as an ObsPy trace.
- Does depend on numpy, Scipy and ObsPy libraries.
- Returns components (beachball), magnitude and misfits
- He did have a dependancy on a certain fortran module for band bass filtering, however we replaced that with Antelope BWBP filter.

Evproc and Wphase (cont)

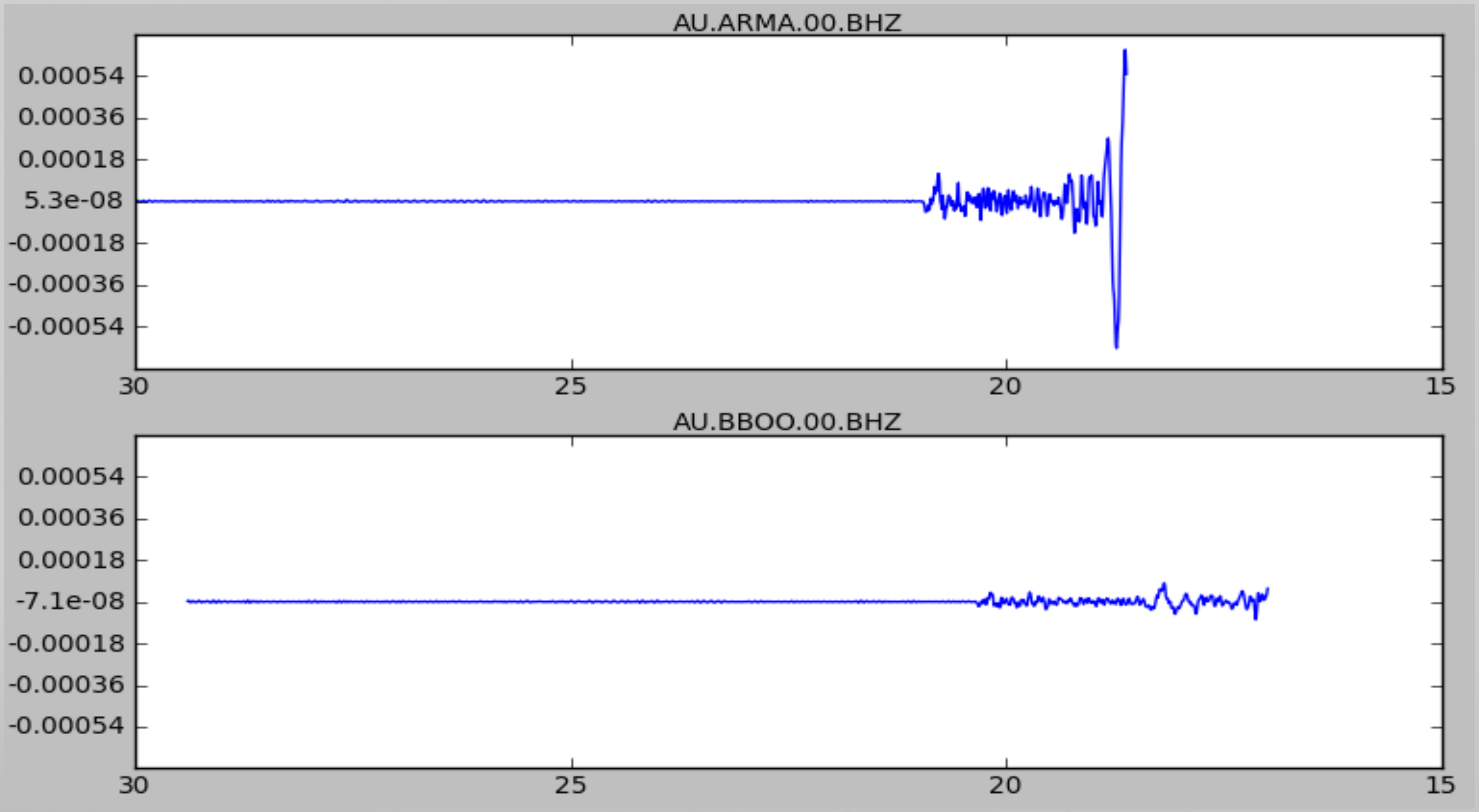
- To call Roberto's module Wphase function, an evproc module was written in Python.
- The usual calls, getwftimes(), process_channel(), process_network() are written in Python.
- Called via stub functions in Perl (which are called directly by evproc in C).
- We now have access to fast libraries like Numpy to process trace data, such as checking for data-gaps or performing interpolation.
- Zero-copy sharing of memory buffers between Numpy and Antelope mean efficient passing around of waveform traces. E.g. using Numpy to check an Antelope trace, or using an Antelope filtering function on an ObsPy Trace object. Not too dangerous if you are careful, and worth the risk.

Running and testing evproc Wphase

- Other inputs include the Greens function used in inversion. Roberto's Wphase is compatible with Kanamori Greens function, and we have a copy (some 17GB) obtained via Phil.
- Also have a test database with around 10 significant events from last few years. Most have Wphase solutions available from the USGS for comparison.
- All testing so far performed using dbevproc. Running under orbevproc should be possible, although very large time window may be an issue.
- Outputs are basically stdout (log) and some matplotlib windows that are displaying during execution.
- No thought yet to bringing results back into evproc or orb2dbt packets.

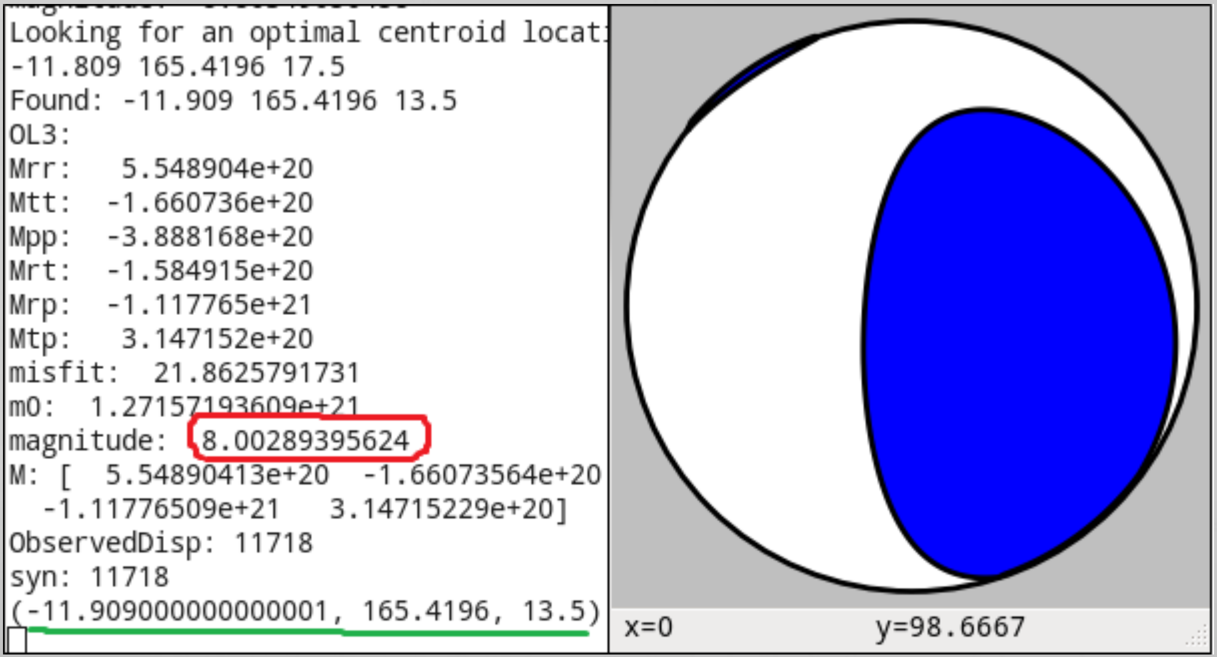
Running and testing evproc Wphase

- Raw waveforms as passed to Wphase. 1500s noise plus 15s*delta signal.

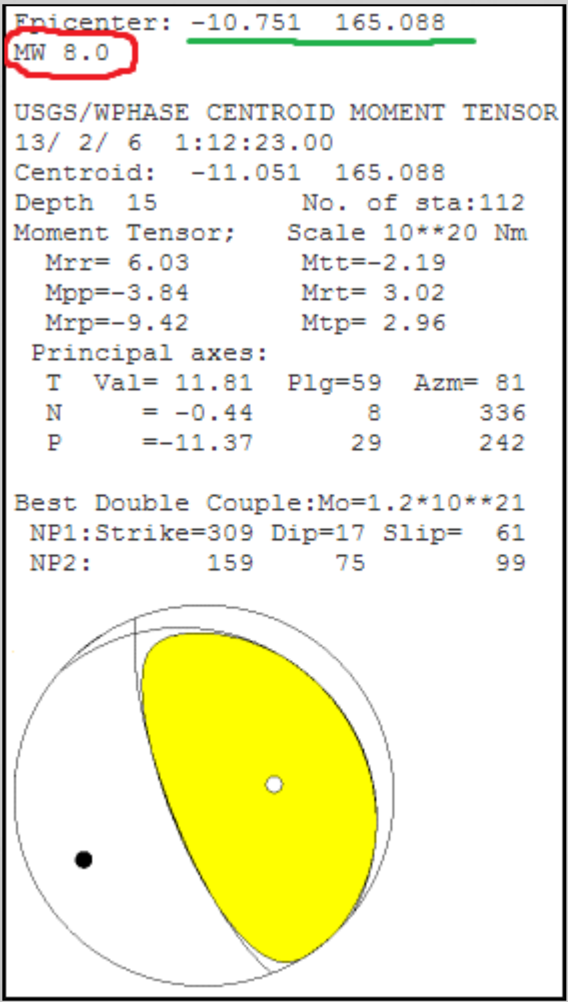


Running and testing evproc Wphase

- Beachball comparison with USGS solution (right).
- Not too bad at all.



```
Looking for an optimal centroid locat
-11.809 165.4196 17.5
Found: -11.909 165.4196 13.5
OL3:
Mrr:   5.548904e+20
Mtt:  -1.660736e+20
Mpp:  -3.888168e+20
Mrt:  -1.584915e+20
Mrp:  -1.117765e+21
Mtp:   3.147152e+20
misfit:  21.8625791731
m0:  1.27157193609e+21
magnitude:  8.00289395624
M: [  5.54890413e+20  -1.66073564e+20
  -1.11776509e+21   3.14715229e+20]
ObservedDisp: 11718
syn: 11718
(-11.909000000000001, 165.4196, 13.5)
```



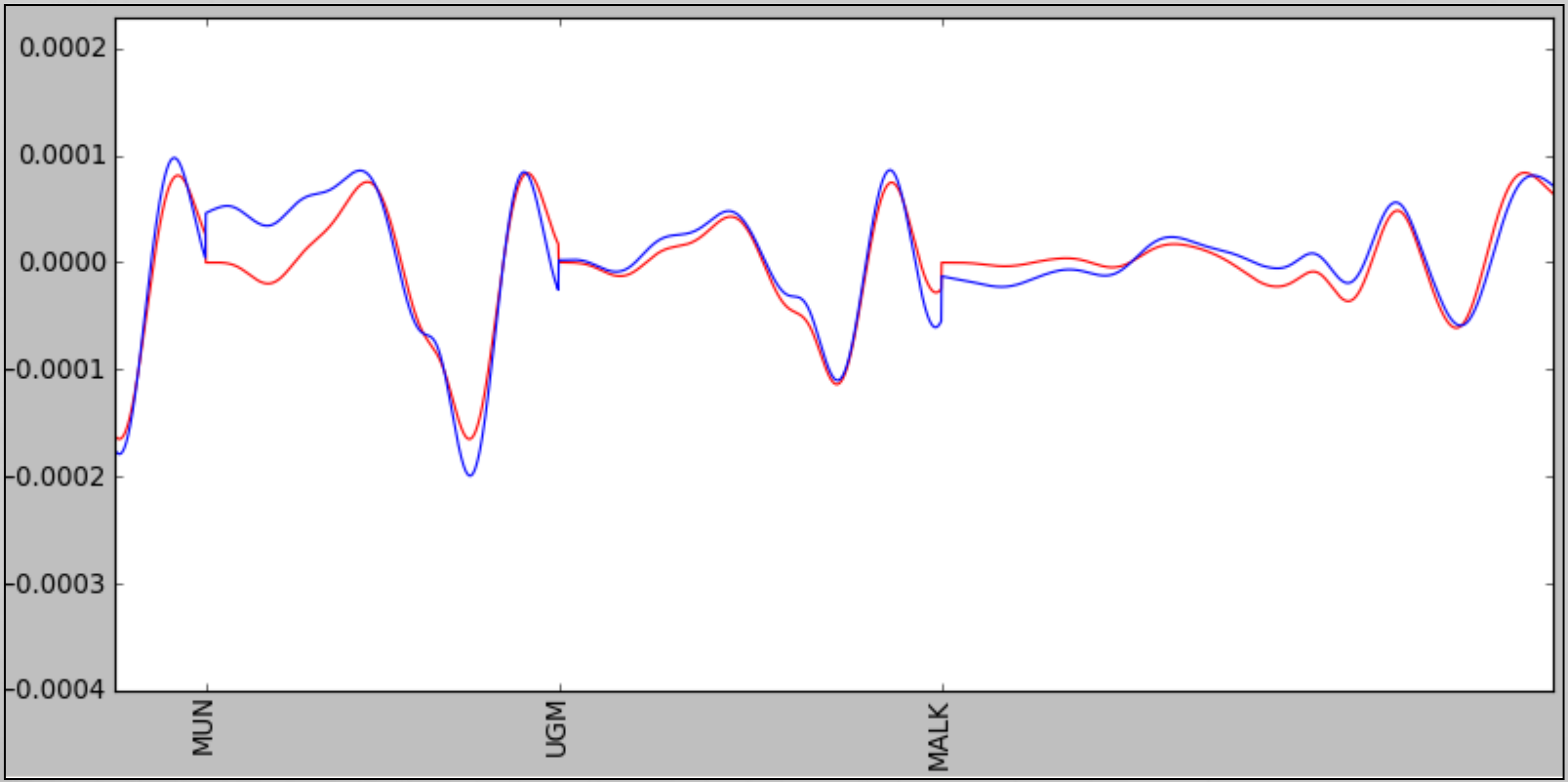
```
Epicenter: -10.751  165.088
MW 8.0

USGS/WPHASE CENTROID MOMENT TENSOR
13/ 2/ 6  1:12:23.00
Centroid:  -11.051  165.088
Depth  15           No. of sta:112
Moment Tensor;     Scale 10**20 Nm
  Mrr= 6.03         Mtt=-2.19
  Mpp=-3.84         Mrt= 3.02
  Mrp=-9.42         Mtp= 2.96
 Principal axes:
  T  Val= 11.81  Plg=59  Azm= 81
  N     = -0.44       8       336
  P     =-11.37      29       242

Best Double Couple:Mo=1.2*10**21
 NP1:Strike=309 Dip=17 Slip=  61
 NP2:        159     75        99
```

Running and testing evproc Wphase

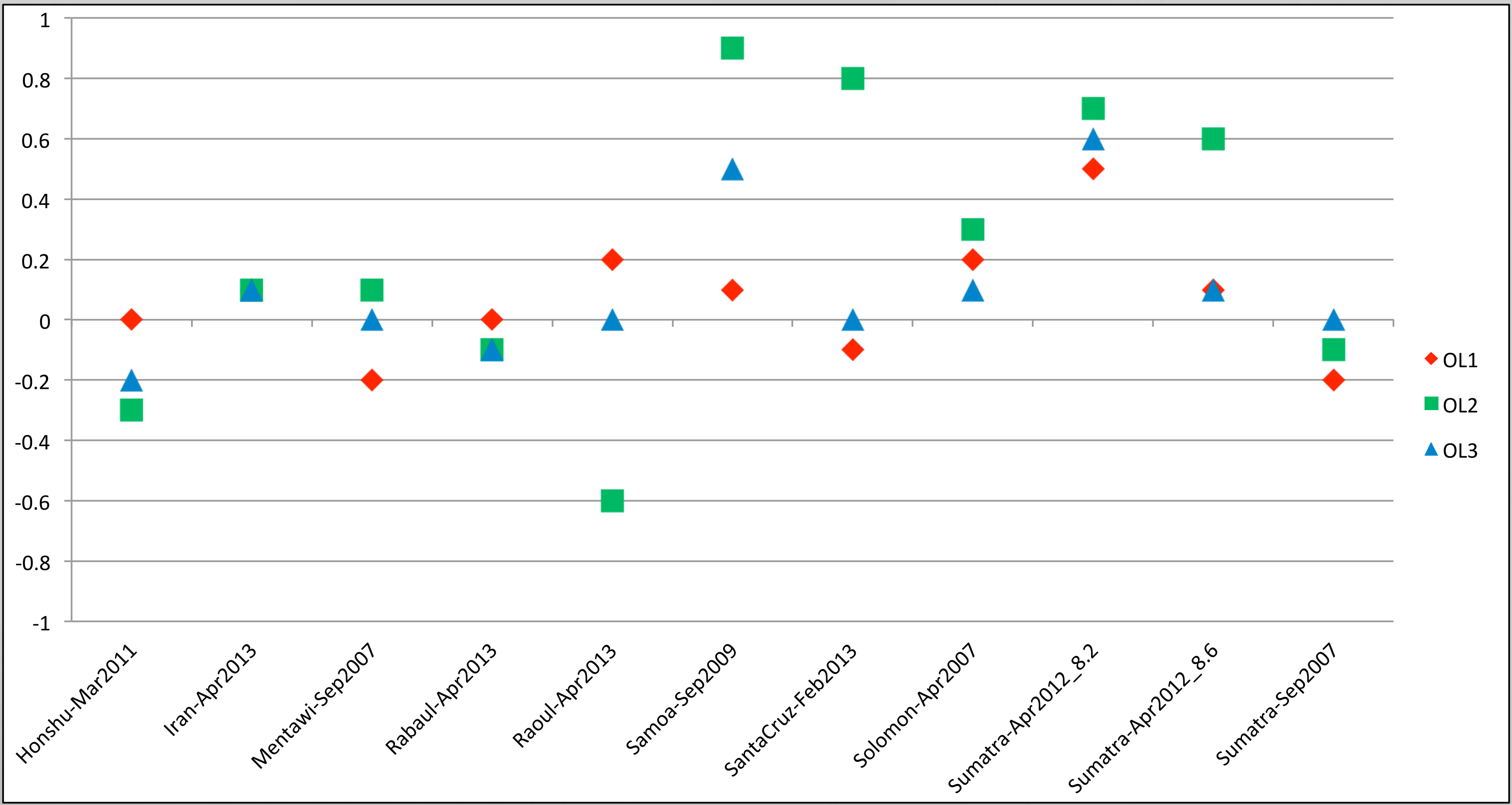
- Some examples of misfits between observations (blue) and synthetics (red).



More about Wphase outputs

- Produces 3 levels of results, OL1, OL2 and OL3.
- OL1 not an inversion – fits the maximum Wphase amplitude against a curve of known events (Phil can perhaps explain this more).
- OL2 is a single point inversion.
- OL3 is a multi-stage grid search inversion starting with hypocentre.
- OL3 also produces refined hypocentre resulting from grid search.
- OL2 and OL3 produce focal mechanism components (beachball)
- OL1 just produces a magnitude value.
- OL1 runs very quickly, OL2 10-20s, OL3 30s to minutes depending on number of stations selected for inversion.

Comparison of results with USGS



Evaluation of results

- Accuracy of OL1 a surprise, often outperforming inversion.
- Some results heavily affected by bad response information.
- Many stations discarded due to poor (or wrong) response, resulting in large azimuthal coverage gaps.
- OL2 tends to overestimate, sometimes blowing out completely.
- While OL3 tends to come back down to sensible or accurate number.
- More work needed to understand how the results are being calculated and find improvements.
- First need to fix response information! Then re-evaluate results.

Where to from here

- Still at prototype stage, although core Wphase module work well.
- Needs hardening and robustness before production operation.
- Would like to reduce or remove dependence on ObsPy and SciPy.
- However still need to figure out deconvolution in Antelope.
- Also runtime environment – evproc or custom driver?
- Considering running OL1 in evproc for an early estimate of >8.0 events.
- And running inversion is custom driver only for prefor after a period of time.
- Custom driver can use all stations, not just arrivals, and capture output.
- Integration into decision support system, manual execution, and more.

Closing comments

- Overall looking very promising, having a reliable and fast calculator for >8.0 events crucial for Tsunami Warning.
- Even if evproc not used to run Wphase, the work done in using Python within evproc will be extremely useful – future network magnitude calculators will be developed or redeveloped in Python using this method.
- Raises issue of using 3rd party libraries like ObsPy/Scipy to perform functions also in Antelope. Consistency of results vs effort in replacing functions with Antelope equivalents, both in terms of development and verifying results.
- Questions?