Quick “how-to” for NWRA automatic ambiguity resolution:

1) Download available at www.cora.nwra.com/AMBIG/
   a) ambig.tgz   code tarball
   b) 01README
   c) Put in accessible place (e.g. A “src” directory)
   d) tar -xvf ambig.tgz
2) Follow Okamoto-san's directions on library path set-up
   a) ssh sinano
   b) source /opt/intel/fce/9.1.043/bin/ifortvars.csh
   c) setenv LD_LIBRARY_PATH /opt/intel/mkl/10.1.1.019/lib/em64t:$LD_LIBRARY_PATH
3) Compile code
   a) local%> make
      I)If need to recompile,
         i) local%> make clean
         ii)local%> make
   b) Resulting compiled code called “ambig”

4) Copy file “par” to your working directory
5) Edit to reflect Hinode data and your file:

```
local%> vi par
```

```
L2 FITS filename    >>   ./sprst_061120_0300.fits
format flags        >>    2     1     0                   irflag  ibflag  iaflag
padding/apodizing   >>    200    10                        npad    nap
smoothing min(B⊥)   >>    4e2                              bthresh
                           1     1     0                   iaunit  ipunit  incflag
                           1     1     20                  iseed   iverb   neq
tfactr: cooling rate >>    1.    2.    0.95                lambda  tfac0   tfactr
Hinode tags         >>     2     3     4    13    33       jfs     jfi     jfa    jsf  jci
```

<u>So, before running have:</u>

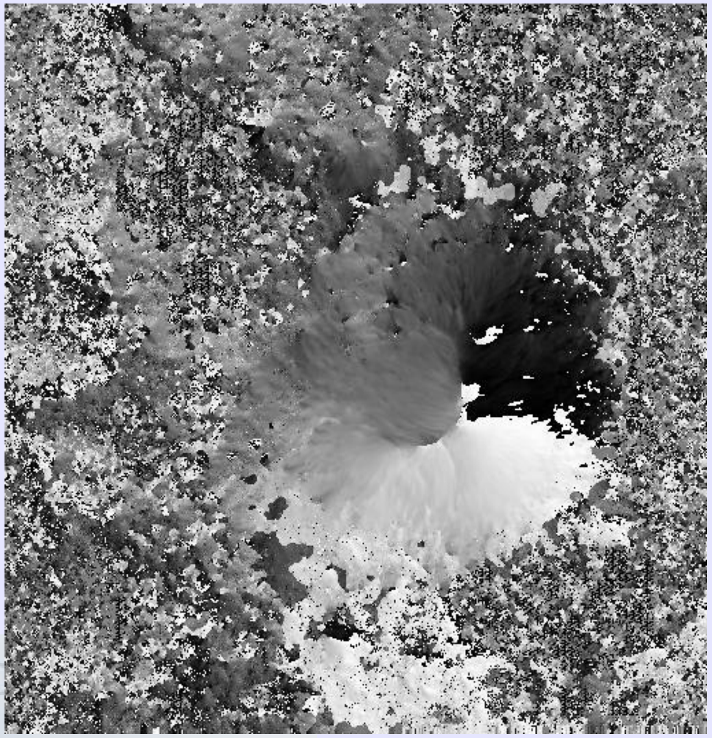
- Executable “ambig” (e.g., ~leka/bin/ambig)
- Working directory containing
  - L2 data files (e.g., sprst_061120_0300.fits)
  - input-parameter file “par” in the same directory

```
local%> ~/bin/ambig
 parameters read
 field read
 transform calculated
 buffering done
 apodizing done
 potential field done
 optimization done
 smoothing done
 results written
 ***END***
```

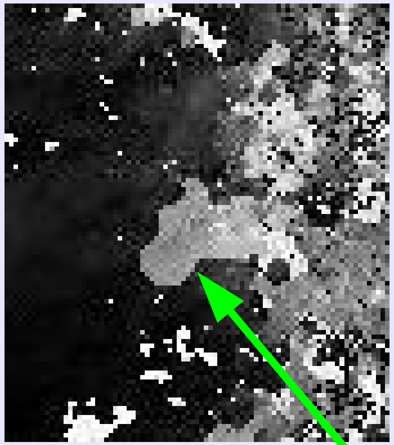
For Hinode FITS files, a new file is written “azimuth.dat”
size is same as original data (nx, ny)

To see (in IDL) for example:

```
IDL> sotsp_read_rst,file,index,data......
IDL> nx=(size(data))(1) & ny=(size(data))(2)
IDL> openr,1,'azimuth.dat'
IDL> azim=fltarr(nx,ny)
IDL> readf,1,azim
IDL> close,1
IDL> tvscl,azim
```

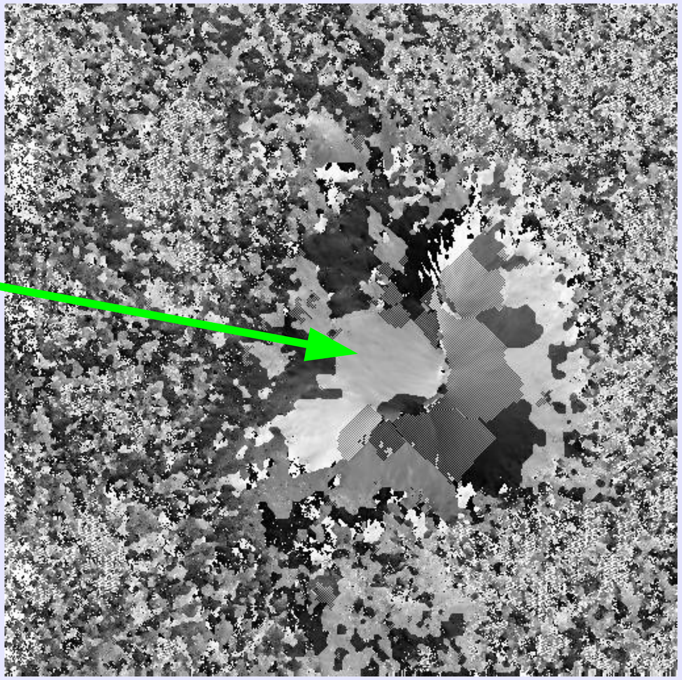


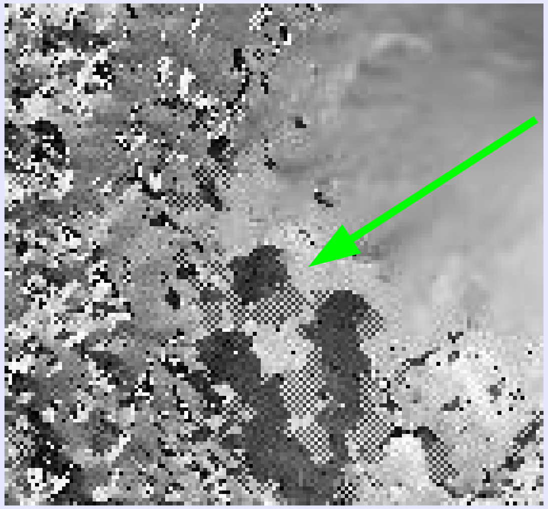
Quick "what can go wrong and how to fix it:"



Lack of smoothness in areas, "line currents": cooling too fast, set "***tfactr***" closer to 1.0 (from 0.95 to >0.99, or 0.995 or 0.999). *Closer to 1 is slower.*

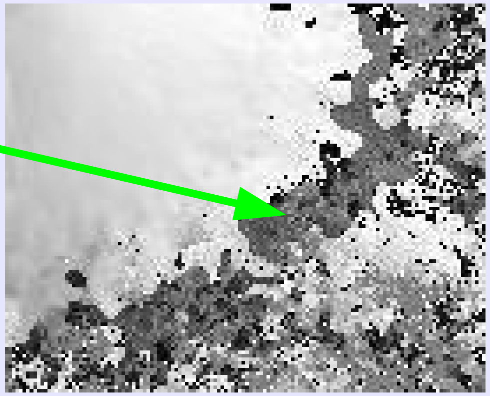
Completely weird looking azimuths: "***iaflag***" is set wrong (sets azimuth=0 at north/west etc)





zig/zag hatching in strong-field areas: smoothing threshold set too low. Increase ***"bthresh"***

Large patches flip in medium-strength field areas: smoothing threshold set too high. Decrease***"bthresh'.***



Quick "how to get the best solution possible:"

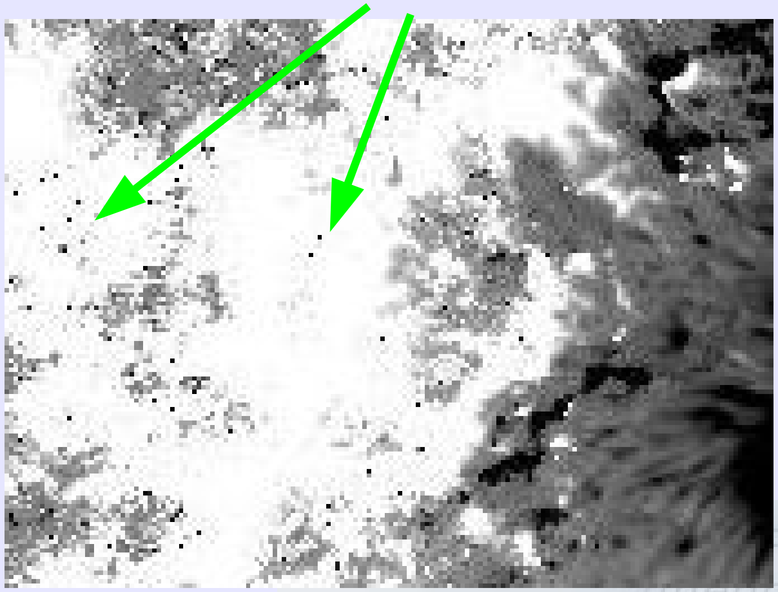
1) Slow cooling time down: set $0.995 <$ ***tfactr*** < 0.999
2) Run multiple times with different random-number seeds,
3) if there are areas of instability, use the most common answer:
   a) change ***iseed*** (integer ≥ 1)
   b) re-name azimuth.dat each time
   c) compare resulting azimuth.dat files for changes.
   d) Where pixels disagree between runs, the resolution is uncertain

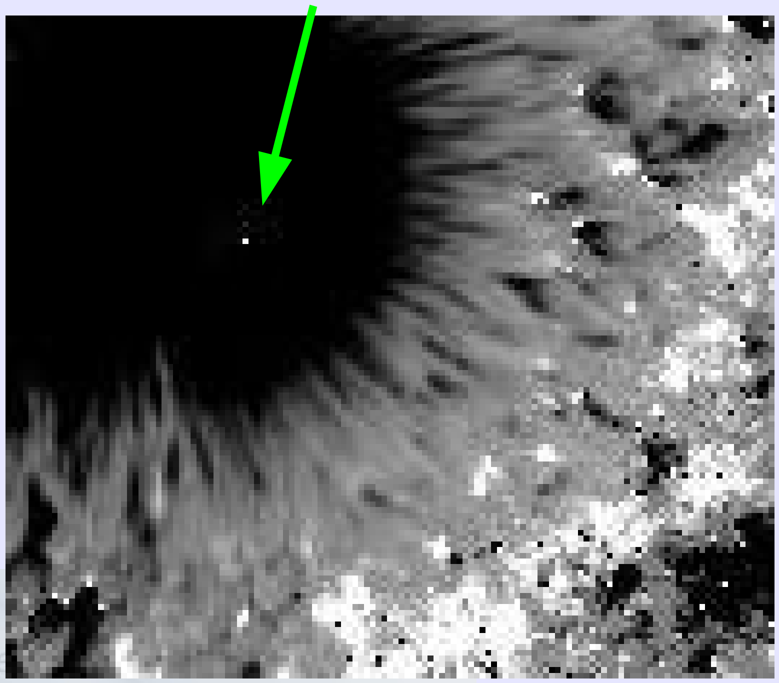
"Bad stuff in →bad stuff out":

If inversion is not good, ambiguity resolution won't be, either.

Know your data!

e.g., sprst_070502_1400.fits: one point has $B_\parallel > 0$ in the middle of $B_\parallel < 0$ spot, similar effects in some areas in plage





A few notes:

You do not need to re-compile the code after changing the "par" file.

I find it good to save parameter files (or work in a different directory) for each file that I am working on. That way it does not get over-written. Example:

```
%> emacs par
%> ambig
```

(use IDL to figure out that I like this solution)

```
%>  cp par par.i_like_this.2009.03.02
```

bthresh is the flux-equivalent transverse-field threshold below which to do nearest-neighbor smoothing,$f|B|\sin(\gamma)$. 400 is a good default value, but change this if you are working in quiet sun (for example).

npad should easily be 100-200, it is the size of "padding with zeros". Computer memory limits this.

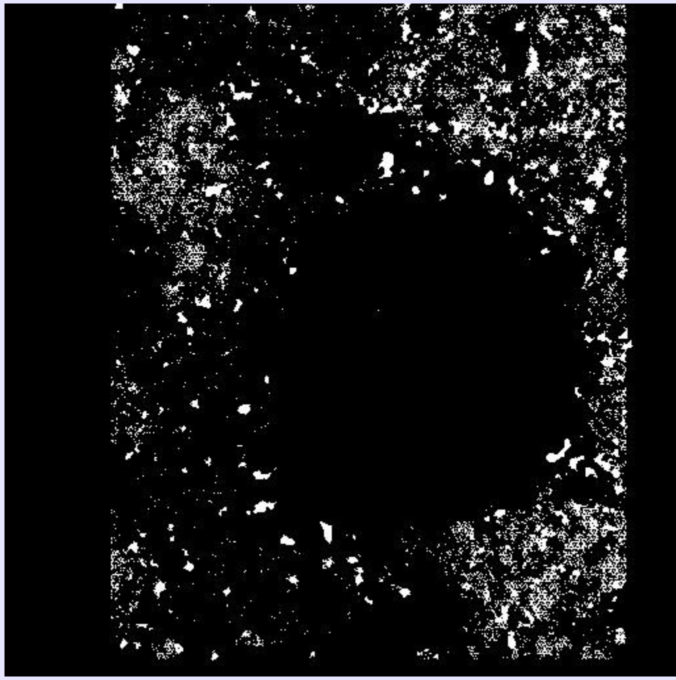
nap should generally (generally! not a rule) be something like the size of "real" structure in your image near the edges. if granulation is 3--7 pixels across, 10 is a good number (for example).

lambda changes the weighting between Jz and div(B): $E = \lambda|Jz|+\mathrm{div}(B)$

"unencumbered science"

To easily compare azimuth results, directly subtract the azimuth images. They will only differ by either 0, or +- 180 degrees.

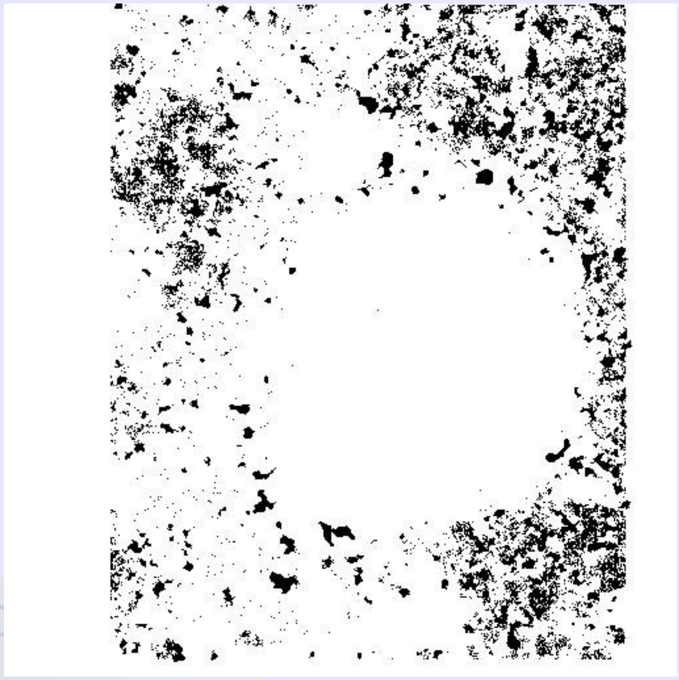
```
IDL> tvscl,abs(azim1 - azim2)
```



Agree is black, *disagreement* is white.
The outside black is where the smoothing happened and the optimization did not (all weak field). The "speckle" of agreement and disagreement are primarily noise (but strong Btrans, meaning bad fits).

To understand which areas are simply uncertain for ambiguity resolution, compare results from different random number seeds (***iseed***).
If an area is not consistent between solutions, then you can treat those pixels as uncertain in your analysis.
Generally this will only be the case in low-transverse-field areas.
As an example: I have 3 results from 3 number seeds, azim1, azim2,azim3:

```
IDL> ok=where(abs(azim1-azim3) lt 1 AND $
IDL  abs(azim2-azim3) lt 1 AND abs(azim1-azim2) lt 1)
IDL> mask=bytarr(512,512)
IDL> mask(ok)=1
IDL> tvscl,mask
```



mask (at right) now shows white (good) and black (bad) areas, black areas are those with inconsistent results (use with care).
This is most robust with 10—20 random number seeds (the 3 here is for demonstration)

Quick “how to judge between different azimuth solutions”:

The following quantities are sensitive to evaluating
how well the NWRA code worked (and maybe AZAM, too),
if you need help in addition to “visual” checks (p4).
(very small bits in IDL language):

- `mean(-1*cos(shift(azim,1,0) - azim)*!dtor)*Btrans`
  - `Btrans = B*sin(incl*!dtor)*(1-straylight))`
  - Lower is better
  - Sensitive to the “patchwork” areas which should be smoothed by the nearest-neighbor.
- `total(abs(Jz(where(Btrans is weak))),`
- `skew(abs(Jz(where(Btrans is weak)))`
  - Lower is better
  - Will need to calculate vertical current
  - Sensitive to line-currents
- `smooth(sobel(gradmag(Bx)),5)*(where(Btrans is weak))`
  - Lower is better
  - Sensitive to bad-propagation of nearest-neighbor smoothing
  - Bx is *heliographic* “Bx” (By works fine, too)
  - Gradmag is the magnitude of the horizontal gradients