North American
ALMA Science
Center

Automated Clean-boxing Algorithms for CASA

NAASC Memo #106

Amy Kimball

Date: May 20, 2011

ABSTRACT

This memo describes placing of clean regions in a CASA image using algorithms and heuristics developed as part of earlier work at NRAO. This information may be useful in developing the imaging stage of the ALMA pipeline. There are now two such algorithms in CASA: one in the "Autoclean" task and one in the "Boxit" task. The Autoclean task is a wrapper for CASA's "Clean" task: it iteratively finds clean regions and calls Clean to produce a final image. Autoclean includes heuristics to determine when/where to place new clean regions, and also when to stop cleaning. This memo describes the Autoclean heuristics, compares and contrasts the algorithms used in Autoclean and in Boxit, and provides the basic algorithm (as code) used by Autoclean.

1 Autoclean heuristics

This section describes the heuristics used by Autoclean to determine where to place clean regions, what shape to use, how many to place, and when to stop cleaning. It follows the code basically chronologically.

INPUT PARAMETERS:

Parameters measured from current residual image: rms, max_res (maximum residual). User can choose to use absolute value of max_res if desired. Input values for determining clean regions: P (peak), I (island), G (gain), N (number of clean boxes), S (stretch pixels), L (large).

Examples of input values:

P = 6 (factor of rms to determine clean region location)

I = 4 (factor of rms to determine clean region size)

G = 0.2 (factor of maximum residual to determine clean region location)

S = 2 (# of pixels for stretching clean region)

L = 100 (# of pixels in size to determine "large" island of pixels)

The algorithm moves through one plane of image (frequency, polarization) at a time, and thus does boxing in 2-D image planes.

CLEAN REGION LOCATION:

The locations of clean boxes are determined primarily by parameters P and G. A pixel must be brighter than P*rms and G*max in order to get a clean box.

CLEAN REGION SIZE: The size of clean box is determined primarily by I: all surrounding, contiguous pixels which are brighter than I*rms are included in the clean box. User has the option to include diagonal pixels when determining contiguity.

CLEAN REGION LOCATION AGAIN:

A clean box must be larger than 1-pixel wide in each direction, unless it is "significantly bright" (for example, 2.5 * the current rms). Otherwise, that pixel (or 1-pixel wide strip of pixels) is ignored.

NUMBER OF CLEAN REGIONS:

Limit the number of clean boxes to N. However, if there are only n islands with a pixel brighter than P*rms or G*max_res (where n<N), only n clean boxes will be created.

CLEAN REGION SHAPE: Normal clean region can be rectangular or circular. (All that was available in CASA tools at the time the algorithm was originally written). User can choose to always use boxes, always use circles, or to allow code to choose. If code chooses, shape is determined by x-size and y-size of island. If x-size and y-size are within two pixels of each other, then clean region will be a circle. User can also choose that clean "islands" larger than L pixels will have a clean region that outlines the island shape, rather than a box or circle.

CLEAN REGION SIZE AGAIN:

User has option, once "box" or "circle" region is determined, to "stretch" the clean box by S number of pixels in each direction, beyond the size determined by I*rms threshold.

NOTE ON NUMBER OF NEW CLEAN REGIONS:

New clean boxes may not be necessary if the previous round of cleaning did not go deep enough. Therefore, when using the boxing algorithm, include previous clean boxes in the search, and in the calculation of N. For example, if one previous clean box still has a sufficiently (> P*rms or G*max_res) pixel, then only N-1 new clean boxes will be created that iteration.

WHEN TO STOP CLEANING:

If no pixels are brighter than P*rms, then no new clean boxes will be made. However, it is possible that max_res is still brighter than the user's clean threshold. My code treats this situation assuming that the chosen clean threshold was unnecessarily low, and stops cleaning.

Normal stopping: if clean threshold is reached, or a maximum number of total iterations is reached. Also stops cleaning if maximum residual has changed by less than some pre-determined fraction. (This, combined with not clean-boxing isolated bright pixels, needs some additional checking.) Also stops cleaning if maximum residual *increases* by a certain number of times (say, 3 or 4) that is input by the user. Also stops cleaning if maximum residual is less than X*rms where X is input by the user. In an automatic version, X should depend on the number of pixels in the image. For example, in a Gaussian distribution, 0.003168% above 4-sigma. So in a 1000x1000 image, you would expect 23 pixels above 3-sigma and 0—1 above 4-sigma. So 4 X rms would be an appropriate threshold for a 1000^2 image, but maybe not for a much larger or much smaller image.

IDEAS FOR IMPROVEMENTS:

- Instead of a fixed integer, the parameter "I" (for determining "island" or "clean box" size) could instead be a fixed fraction of the peak pixel in clean box.

- Fit an elliptical Gaussian to an island in order to determine orientation of an elliptical or rectangular clean box.

- Change value of P to depend on location. For example, if a local peak pixel is the same distance from a higher peak as the first sidelobe is from the beam center, then require a stricter value of P to reduce the chance of cleaning sidelobes.

2 Autoclean and Boxit: a tale of two algorithms

Both algorithms find potential "islands" in an image above a given threshold. Amy Kimball wrote the algorithm in Autoclean; Dave Mehringer wrote the algorithm in Boxit.

Autoclean uses a numpy rec.array to keep track of the position, flux, and (current) mask value of each pixel. It's a multi-dimensional array, where sub-arrays can be indexed by *name*. The xyMask rec.array contains four NxM (image-sized) arrays, named "x", "y", "value", and "mask". This seemed a good way to be able to index the position, flux density, and current mask for any particular pixel.

Autoclean produces an ascii file with a list of pixels in each island. The four columns in the output file are island#, x-pixel position, y-pixel position, and flux of pixel. The user can also input the name of an output mask image; Autoclean will create a mask image of the individual pixels. It will overwrite any previous image with this file name. The result is similar to simply doing an image threshold, except that the Npeak parameter limits the total number of islands. Note that if Npeak is large enough and island_threshold = peak_threshold, then the maskimage will be equivalent to just doing an image threshold with ia.getregion(mask=<pixels above threshold>).

The two pieces of code use similar algorithms to identify pixels above the threshold, with two crucial differences:

Part A)

Boxit was written to literally "box" the islands, as in locate a rectangular region around each island. Boxit therefore keeps track of the coordinates of the rectangle corners, but not the individual pixels. On the other hand, Autoclean does not try to force a particular region shape on the user. Instead, it keeps track of the individual pixels in each island, and produces an ASCII file with the pixel identifications, allowing the user to later place a clean region of any shape.

Part B)

Autoclean starts with the brightest local maximum in the image, and works its way down from there. It allows the user to limit the total number of islands find using the Npeak input parameter. Boxit, on the other hand, works with the given threshold and finds *every* island above that threshold. Thus, Boxit doesn't care about the order (brightest to faintest) when identifying islands.

How Boxit does Part A

Boxit uses a recursive subroutine (find_nearby_island_pixels) to search the neighbors around each above-threshold pixel, in order to identify nearby above-threshold pixels. Each time a new pixel is identified, the module is (recursively)

called. When a new bright pixel is outside the current box corner coordinates, the coordinates are stretched in order to increase the size of the box. I'm not sure whether this type of module will still work if one wants to keep a list of individual pixels, as Autoclean does.

How Autoclean does Part A

Autoclean iterates over a list of pixels of interest, starting with the brightest (unmasked) pixel in the image. As it finds neighboring pixels that are above the island_threshold, it adds them to the list. It continues iterating over the list of pixels (even while it adds to that list; an interesting feature of Python allows this technique).

How Boxit does Part B

Boxit uses numpy.unravel_index(mask.argmax()) to identify pixels that are above the threshold. Becase argmax() is being used on the mask, which has only True/False values, the "pos" (position index) variable indexes the first "True" (above-threshold) pixel in the image, which is not (probably ever) the brightest pixel in the image. Thus, Boxit does not start with the brightest source in the image, and can't currently "rank" the sources from brightest to faintest.

How Autoclean does Part B

Runs numpy.where to find current non-islanded above-threshold pixels. Identifies next peak as the max of those pixels. Finds the position of that peak. There *might* be an easier way to do this using either CASA tools or Python features, but I couldn't figure it out. As it stands, I don't know of a way to run ia.statistics() on an image while supplying a list of True/False values as the mask. It's not even clear to me that a mask image can be given as input to ia.statistics via the mask parameter. If this is changed, then it will be easy to get the peak flux/peak position using ia.statistics and the current value of the xyMask['mask'] array. Alternatively, if the ['value'] array in xyMask were zeroed when a pixel was added to the island (and its xyMask['mask'] value set to False), then one could simply use numpy.unravel_index(argmax()), similar to what Boxit does.

3 Autoclean algorithm

The Autoclean task in CASA iteratively finds clean regions in an image (using the algorithm described in Section 2 of this memo) and cleans the image by calling the CASA Clean task, and decides to stop the iterations using the heuristics described in Section 1 of this memo. As of the CASA 3.2 release, the Autoclean task does not work, because the Clean task has been modified while Autoclean has not been kept up to date. I extracted the basic clean-boxing algorithm from the Autoclean task which can work on an image on its own, without doing any of the cleaning iterations. The python code “get_islands” for this algorithm is provided below.

```
import numpy

# Starting with peak in image, find islands: contiguous pixels above threshold.
# Tests peak: bright enough to cleanbox?  Then adds chosen region shape to mask.
# Continues with next peak pixel until Npeak peaks have been found.
def get_islands(imagename='', maskimage='', outputfile='', Npeak=3,
                island_threshold=0, peak_threshold=0, diag=False):

    # Npeak: maximum number of [new] boxes
    # island_threshold: flux threshold for island edges
    # peak_threshold: put boxes around pixels with flux density above this value
    # diag: count diagonal connections when identifying islands

    writemask = bool(maskimage)  # checking to see whether there is a mask image to modify

    if(writemask):
        # create/overwrite new output mask image
        ia_tool = ia.newimagefromimage(infile=imagename,
                                       outfile=maskimage,
                                       overwrite=True)
        ia_tool.set(pixels=0.0)
        ia_tool.close()
        ia_tool.done()

    # types (as lists) for the recarrays that will store pixel and island info.
    pix_dtype = [('x','i4'),('y','i4'),('value','f8'),('mask','bool')]

    # Find all pixels above the threshold; make temporary mask: tmp_mask
    ia.open(imagename)
    # escape characters in image name that would confuse the lattice expression processor
    escaped_imagename = imagename
    for escapeme in ['-', '+', '*', '/', '_']:
        escaped_imagename = re.sub("[" + escapeme + "]", "\\" + escapeme, escaped_imagename)
    mask_command = escaped_imagename+'>'+str(island_threshold)
    origmask = ia.getregion(mask=mask_command,getmask=True).squeeze()

    # pixel values
    pixelValues = ia.getregion().squeeze()
    ia.close()
    # store pixel positions and mask values in a numpy recarray
    grid = numpy.indices(origmask.shape)
    xyMask = numpy.rec.fromarrays([grid[0], grid[1], pixelValues, origmask], dtype=pix_dtype)
    nx, ny = origmask.shape
```

```
outf = open(outputfile+'pixels.dat', 'w')

# keep going until we've found Npeak islands
# or there are no more pixels above the island_threshold
# or the peak is less than the peak_threshold
Nregions = 0
Nkept = 0
while Nregions < Npeak:

    if not(xyMask['mask'].max()):
        # no more pixels above island threshold: we're done
        break

    # find the next peak and its location
    good = numpy.where(xyMask['mask'])
    peak = pixelValues[good].max()
    peakind = pixelValues[good] == peak
    pos = (good[0][peakind][0], good[1][peakind][0])

    if peak < peak_threshold:
        # if peak is below peak threshold for clean boxing: we're done
        break

    # since we've already checked this pixel, set its mask to False
    listPix = [xyMask[pos]]
    xyMask[pos]['mask'] = False

    # find all above-threshold contiguous pixels of this island
    for pixel in listPix:
        thisX = pixel['x']
        thisY = pixel['y']
        # search the pixels surrounding the pixel of interest
        xLook1 = max(0,thisX-1)      # in case we're at the image edge...
        xLook2 = min(thisX+2,nx-1)   #            |
        yLook1 = max(0,thisY-1)      #            |
        yLook2 = min(thisY+2,ny-1)   #            V
        # add new above-threshold pixels to the list for this island
        if(diag):
            contig_pix = xyMask[xLook1:xLook2, yLook1:yLook2]
            listPix += [pix for pix in contig_pix.ravel() if(pix['mask'])]
            # as we've now added these pixels, set their mask to False
            # (contig_pix is reference to xyMask)
            contig_pix['mask'] = False
        else:
            # only look at the four pixels that share an edge with pixel-of-interest
            contig_pix = []
```

```
                contig_pix += [xyMask[thisX, yLook1]]
                contig_pix += [xyMask[thisX, yLook2-1]]
                contig_pix += [xyMask[xLook1, thisY]]
                contig_pix += [xyMask[xLook2-1, thisY]]
                listPix += [pix for pix in contig_pix if(pix['mask'])]
                # since we've already added these pixels, set their mask to 0
                xyMask[thisX, yLook1]['mask'] = False
                xyMask[thisX, yLook2-1]['mask'] = False
                xyMask[xLook1, thisY]['mask'] = False
                xyMask[xLook2-1, thisY]['mask'] = False

        # found all pixels in this island; get bounding box
        islandPix = numpy.rec.fromrecords(listPix, dtype=pix_dtype)
        xmin = islandPix['x'].min()
        xmax = islandPix['x'].max()
        ymin = islandPix['y'].min()
        ymax = islandPix['y'].max()

        Nregions += 1  # This island should be in a clean region.

        # This is a new island for clean boxing.  Prepare to mask!
        Nkept += 1

        # Write this island to the output file
        for outpixel in listPix:
            outstring = "%s %s %s %s" % (Nkept, outpixel['x'],
                                         outpixel['y'], outpixel['value'])
            outf.write(outstring+'\n')

        # Put these pixels into the image mask
        if(writemask):
            mask_island(maskimage, islandPix)

    outf.close()
    return Nkept


def mask_island(maskimage='', pixelList=None):
    ia.open(maskimage)
    mask = ia.getregion()
    mask[pixelList['x'], pixelList['y']] = 1
    ia.putchunk(mask)
    ia.close()
    return
```