1 Basics and Notation

All codes here use the C/IDL convention of counting from 0. Ie. if you have N pizels their indices will vary from 0 to $N-1$.

For a two dimensional array what is called the x position (as given by the argument name to a routine) is the dimension for which adjacent elements are adjacent in memory and the y (also given by the argument name to a routine) position is the one in which they are not. If the declared leading (x) dimension is N_{lead} then the memory location of a pixel given by coordinates (x, y) is $x + N_{lead}y$, relative to the first element of the array. Note that the declared leading dimension N_{lead} need not equal the number of elements N_x in the x dimension. However, it should generally be the case that $N_x \leq N_{lead}$.

For many applications you have to call three routines to do the job. The first one will do initialization, the second the actual work and the third cleanup. A structure is passed between the routines to hold the necessary information. Initialization might include such things as doing Fourier transforms of convolution kernels or computing interpolation weigths. Cleanup typically involves freeing memory. Typically you would only call the setup and cleanup routines once. The work routine may be called many times, for example in a loop processing multiple images.

Most, but not all, routines can use OpenMP internally. Therefore you will need to compile them with the appropriate flags. On the other hand you don't have to run them on multi processors, they work just fine on one and will not use more than one unless you set the number of threads in the calling program. There are currently no guarantees that you can call routines from multiple threads simultaneously if the initilization is run from a single thread. FFTW makes it a real pain. Working on it...

2 Spatial Interpolation

These function are in finterpolate.h and finterpolate.c.

The objective here is to evaluate a function given on one set of gridpoints on a different set of grid-points. Unless the function has certain very restrictive properties this is generally not possible to do in a unique way.

Suppose that the input image $image_{in}$ is given on a regular grid with $N_{x,in}$ by $N_{y,in}$ points, that you want to interpolate this to an output image $image_{out}$ with N_x by N_y points and that the coordinates are given by x_{in} and y_{in}, both real arrays of size N_x by N_y.

In that case $image_{out}(i, j) = image_{in}(x_{in}(i, j), y_{in}(i, j))$, where $image_{in}$ is now effectively treated as a function of real variables.

As an example suppose that you have a solar image with a radius R and center (x_0, y_0) and that you

want to interpolate it to a regular array in longitude π (from central meridian) and co-latitude θ. In that case you would set

$$x_{in}(i,j) = x_0 + R\sin(\phi(i))\sin(\theta(j))$$

and

$$y_{in}(i,j) = y_0 + R\sin(\phi(i))\cos(\theta(j))$$

where it has been assumed the the observation distance is infinite and the B angle 0.

2.1 Initialization routines

So far only seperable algorithms have been implemented where first the one and then the other dimension is interpolated. In other words the interpolation kernels are products of a function of x and a function of y.

Note that if all the interpolation does is to shift the image by a constant amount (ie $x_{in}(i,j) = i + \delta_x$ and $y_{in}(i,j) = j + \delta_y$ then interpolation amounts to convolution. From the convolution theorem it then follows that the interpolation error for sinewaves (ie. the difference between the true value and the interpolated ones) is given by the Fourier transform of the kernel minus that of a delta function at (δ_x, δ_y). results are shown in Figure 1. As can be seen polynomial type interpolations generally do better at lower frequencies and the Wiener interpolation better at higher frequencies. In other words polynomial interpolation works best for smooth functions.

I should be noted that only even order interpolations make much sense and that no guarantees are give if called with an odd order.

2.1.1 Linear

The routine init_finterpolate_linear has two arguments:

- pars: The structure to be initialized.
- extrapolate: The distance beyond the edge of the input array to extrapolate.

2.1.2 Cubic convolution

The routine init_finterpolate_cubic_conv has three arguments:

- pars: The structure to be initialized.
- edgemode: 0 to go as far as you can (ie. 1 to N-2) with symmetric kernel (ie. two points on each side of target point). Otherwise go beyond.
- extrapolate: The distance beyond the edge of the input array (ie. 0 to N-1) to extrapolate.

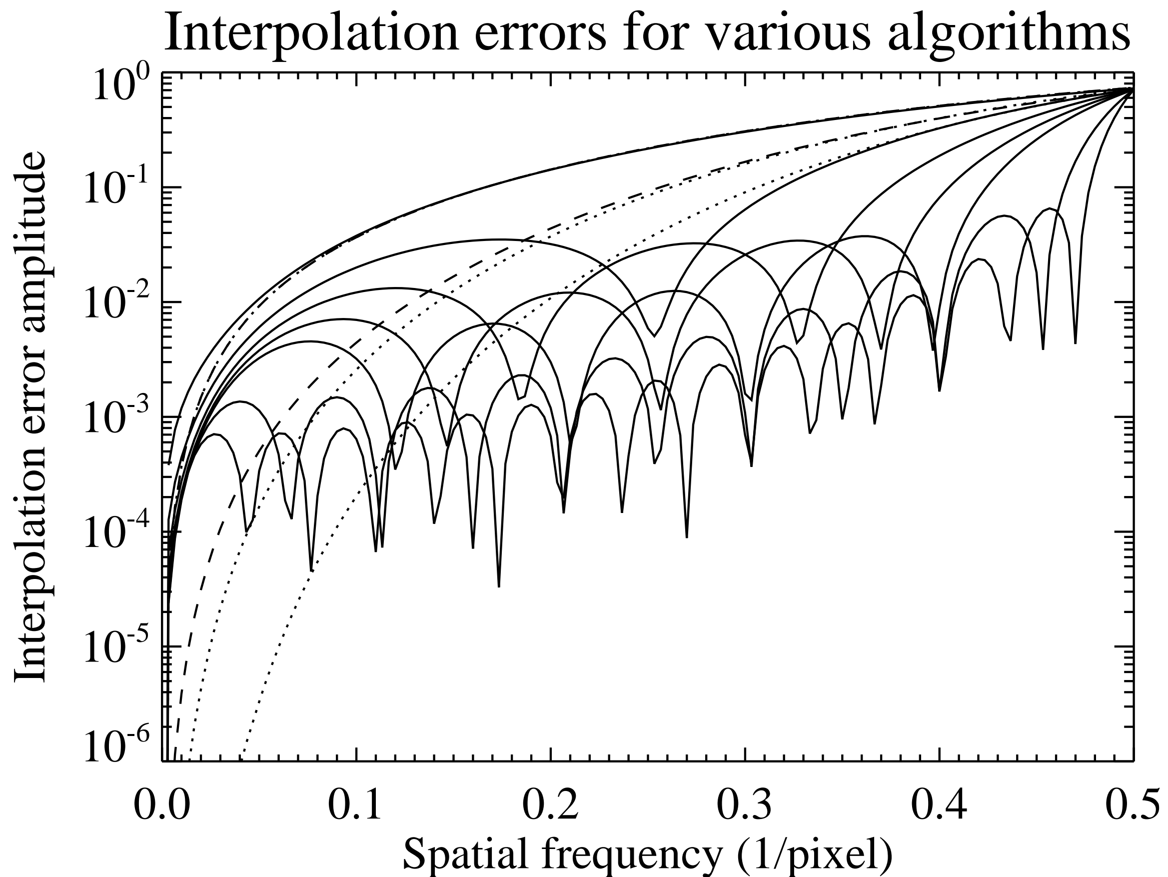

Figure 1: Integration errors as a function of spatial frequency for various algorithms and order. Solid: Wiener for order=2, 4, 6, 8, 10, 20 and 30 and 1 constraint. Dotted: Polynomials with 2,4 and 6 points. Dashed: Linear and cubic. Note that the Wiener order 2 is different from linear since only one constraint was applied. If two constrains are applied (thereby forcing a 1st order polynomial to be preserved) the curves are identical.

2.1.3 Wiener interpolation

In this case a set of coefficients are found such that the expected error is minimized assuming a particular autocorrelation function (or equivalently power spectrum).

The routine init_finterpolate_wiener has eight arguments:

- pars: The structure to be initialized.
- order: Order of the interpolation, as given by number of points to use in interpolation.
- edgemode: 0 to go as far as you can (ie. order/2-1 to N-order/2) with symmetric kernel (ie. order/2 points on each side of target point). Otherwise go beyond.
- extrapolate: The distance beyond the edge of the input array (ie. 0 to N-1) to extrapolate.

- minorder: If edgemode is $\neq 0$ then reduce order to keep kernel symmetric around target point until minorder is reached. Use minorder after that.
- nconst: Number of polynomial constraints to impose. 0 for none, 1 to preserve a constant, 2 to preserve a linear and so forth.
- cortable: Which of the hardcoded tables to use. If 0 use filename pointed to by the followinf argument.
- filenamep: Pointer to the name of the table used. Set to the name if cortable$\neq$0 and filenamep not NULL.

Note that polynomial interpolation may be implemented by setting nconst equal to order. The stability of the algorithm used for this (unlike the one in interp2.pro) is not great, but probably adequate.

Maybe should explain method in detail and talk about regularization.

Note that there is a routine init_finterpolate_wiener_old with onl the first six arguments for backwards compatibility.

2.2 Main interpolation routine

The routine doing the work is finterpolate. Arguments are the following

- pars: The parameter structure initialized earlier.
- image_in: The input data array.
- xin and yin: Coordinates to interpolate to.
- image_out: Output array.
- nxin, nyin and nleadin: Size of input array and leading dimension.
- nx, ny and nlead: Size of output array and leading dimension.
- fillval: Value to put in output array when outside of the valid interpolation area.

2.3 Cleanup

All this routine (free_finterpolate) does is to free any memory allocated by the initialization routines.

3 Spatial Filtering and Binning

These routines are in fresize.h and fresize.c.

Purpose is to provide various types of filtering and subsampling of images.

In several case both brute force and FFT based algorithms have been implemented. Which one is the faset depends on circumstances. Also note that a single bad value (even if not NaN) will spread to the whole image if the FFT versions are used.

The FFT based versions are not separately described below. They have an added "_fft" added to the name and two extra parameters added to the argument list. The extra parameters give the input size of the images (ie. nxin and nyin) and are required due to the implementation of the FFTW library.

3.1 Initialization routines

Though not explicitly noted below all kernels are normalized, ie. a constant is preserved.

3.1.1 Simple subsampling and binning

The routines init_fresize_sample and init_fresize_bin perform simple sampling and binning. Unlike the rest of the routines the binning does not operate with a centered kernel.

Note that binning and sampling can also be done directly without initialization, work and cleanup routines by calling the routines fsample and fbin, whose arguments are:

- image_in and image_out: Input and output images.
- nxin, nyin and nleadin: Size and leading dimension of input image.
- nxout, nyout and nleadout: Size and leading dimension of output image.
- nsub: Amount of subsampling or binning.
- xout and yout: Offset to apply in input image before sampling or binning.
- fillval: Value to use outside of valid area.

The sampling and binning functions with zero offsets are equivalent to the IDL function rebin with and without the sample flag, respectively.

3.1.2 Boxcar convolution

This routine convolves the input image with a centered boxcar. Parameters are:

- pars: The structure to be initialized.
- hwidts: Half width of the boxcar. Total width is 2hwidth+1.
- nsub: Distance between sampled points.

3.1.3 Gaussian convolution

This routine convolves the input image with a truncated Gaussian centered on the target pixel. Parameters are:

- pars: The structure to be initialized.
- sigma: Width of Gaussian. Shape is $\exp(-(d/sigma)^2/2)$, where d is the distance from the target point.
- hwidth: Half width of the kernel. Total width is 2hwidth+1.
- nsub: Distance between sampled points.

3.1.4 Sinc convolution

This routine convolves the input image with an apodized sinc function centered on the target pixel. Note that the convolution is done separately in x and y. The convolution kernel is a product of a function of x and the same function of y.

Examples of the responses are shown in Figure 2.

Parameters are:

- pars: The structure to be initialized.
- wsinc: Width of sinc function. Shape is $sinc(d/wsinc)Ap(d)$, where d is the distance from the target point and Ap is the apodization function (see below). You may want to set wsinc=nsub.
- hwidth: Half width of the hernel. Total width is 2hwidth+1.
- iap and nap: Type and width of apodization. $Ap(d) = 0$ for $d > (nap \times wsinc)$. Types are:
  - iap=0: $Ap(d) = 1$.
  - iap=1: Parabolic apodization $Ap(d) = 1 - (d/(nap \times wsinc))^2$.
  - iap=2: Sinc apodization $Ap(d) = sinc(d/(nap \times wsinc))$.
  - otherwise: $Ap(d) = 1$. This is not to be relied on!
- nsub: Distance between sampled points.

  Some results are shown in Figure 3. As can be seen simple truncation results in a lot of ringing but a sharper dropoff near the cutoff frequency, while a sinc apodization leads to a lot less ringing but a slower falloff. Parabolic apodization is in between.

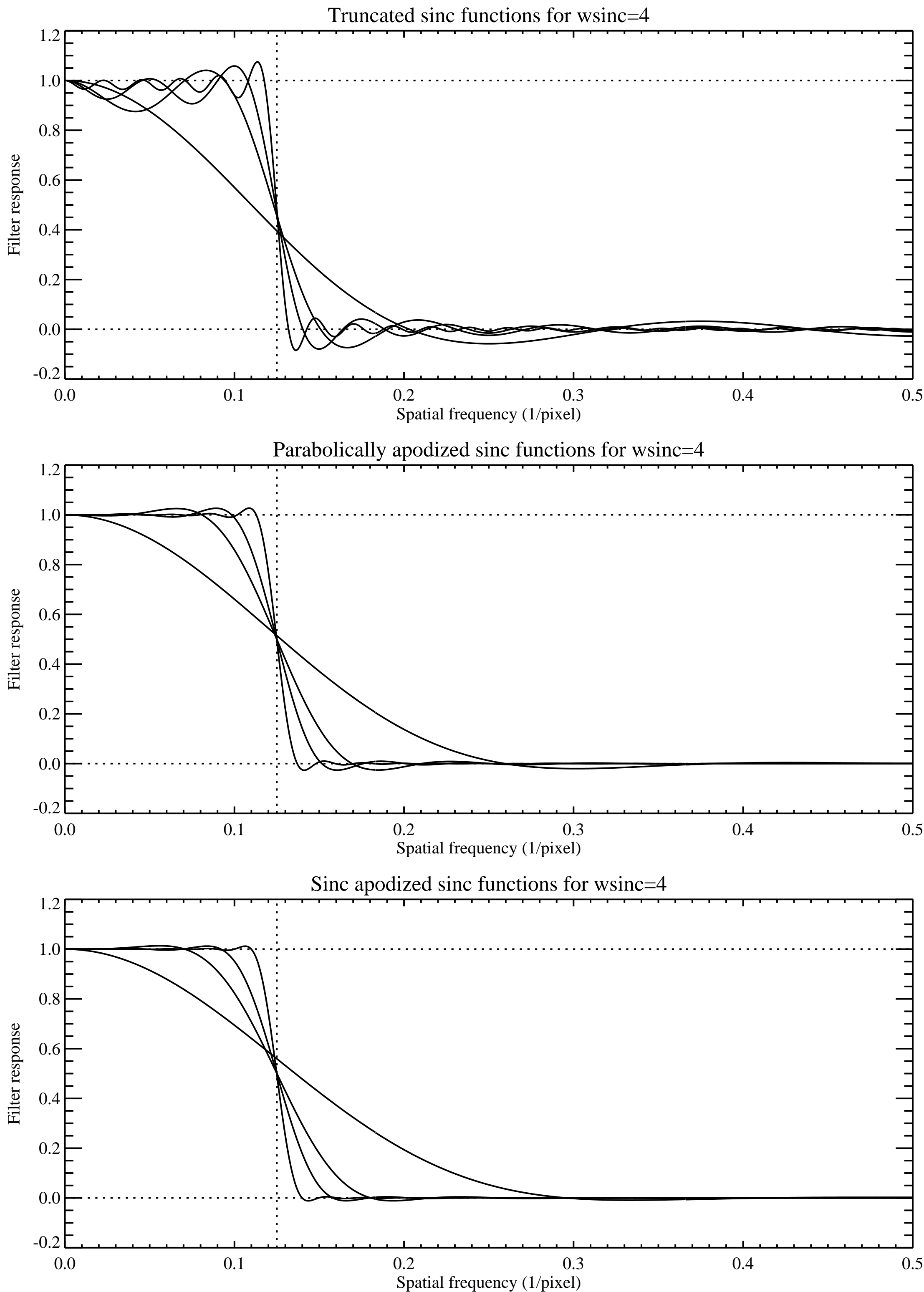

Figure 2: Filter responses for various sinc filters. wsinc=4 in all cases. Title gives type of apodization. Different lines show nap=1, 3, 5 and 11. Vertical dotted line shows nominal cutoff frequency (0.5/wsinc). Note that the imaginary component of the response is 0 (ie. there is no phase error) since the kernels are symmetric.

3.1.5 Gaussian convolution with circular truncation

This routine convolves the input image with a truncated Gaussian centered on the target pixel. Parameters are:

- pars: The structure to be initialized.
- sigma: Width of Gaussian. Shape is $\exp(-(d/sigma)^2/2)$, where d is the distance from the target point.
- rmax: Truncation radius. You will likely want $rmax \leq hwidth$.
- hwidth: Half width of the kernel. Total width is 2hwidth+1.
- nsub: Distance between sampled points.

3.1.6 Convolution with apodized Airy function

This routine convolves the input image with an apodized Airy function centered on the target pixel. With no apodization this would result in a perfect cutoff i k-space. Parameters are:

- pars: The structure to be initialized.
- cdown: The ratio of the pixel nyquist frequency to the cutoff frequency of the Airy function.
- hwidth: Half width of the kernel. Total width is 2hwidth+1. Routine will set appropriate value if $hwidth < 0$.
- iap and nap: Type and width of apodization. If Z_{nap} is the position of the nap'th zero in the Airy function then $Ap(d) = 0$ for $d > Z_{nap}$ and the types are:
  - iap=0: $Ap(d) = 1$.
  - iap=1: Parabolic apodization $Ap(d) = 1 - (d/Z_{nap})^2$.
  - iap=2: Sinc apodization $Ap(d) = sinc(d/Z_{nap})$.
  - iap=3: Airy apodization $Ap(d) = Airy(d/d_0)$, with d_0 chosen such that the first zero is at Z_{nap}.
  - otherwise: $Ap(d) = 1$. This is not to be relied on!
- nsub: Distance between sampled points.

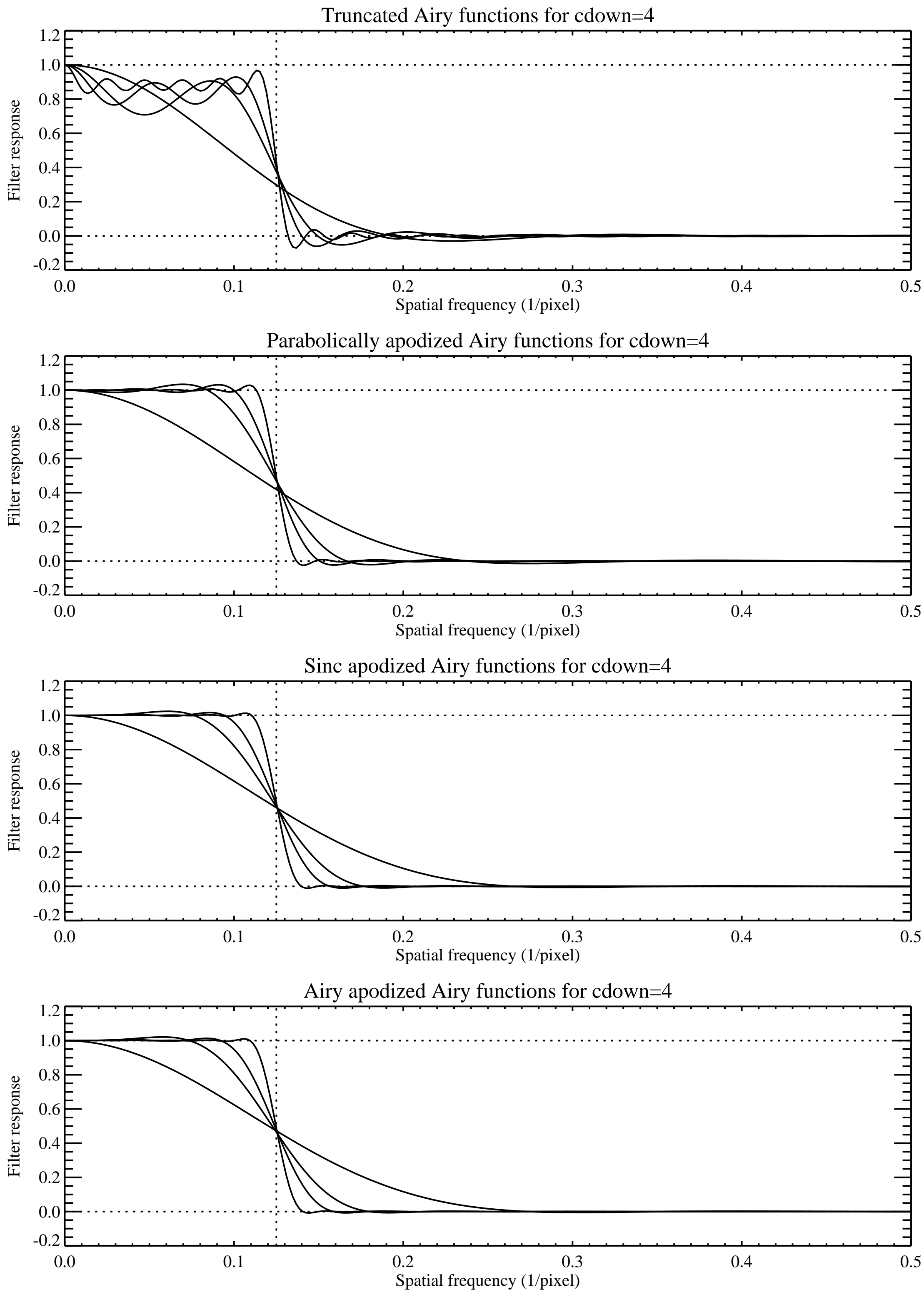

Figure 3: Filter responses for various Airy filters. cdown=4 in all cases. Title gives type of apodization. Different lines show nap=1, 3, 5 and 11. Vertical dotted line shows nominal cutoff frequency (0.5/cdown). Note that the imaginary component of the response is 0 (ie. there is no phase error) since the kernels are symmetric.

3.2 Main sampling routine

The main routine fresize has 12 parameters:

- pars: Structure passed from initialization routine.
- image_in: Input image
- image_out: Output image
- nxin, nyin, nleadin: Size and leading dimension of input image
- nxout, nyout, nleadout: Size and leading dimension of output image
- xoff,yoff: Image offset. Values are added to indices for the input image.
- fillval: Value to use outside of valid area.

3.3 Cleanup

The routine free_fresize frees any memory associated with the structure passed to it.

4 Spatial Gapfilling

These routines are in gapfill.h and gapfill.c.

The gapfilling algorithm is essentially the same as used for the Wiener interpolation in the previous section.

There are three routines:

4.1 Initialization

The initialization routine is init_fill. Parameters are the following:

- method: Currently use 11. This is in need of some improvement.
- pnoise: Assumed photon noise. $1/\sqrt{N}$, when N is the exposure level in e^{-}.
- order: Interpolation order (number of points in each direction). Note that unlike the interpolation this number is generally odd.
- targetx and targety. Set to (order-1)/2.
- pars: Structure to be initialized.

- filenamep: Pointer to the name of the table used. Set to the name if method≠0 and filenamep not NULL.

This routine should only be called once from a single thread.

4.1.1 Main routine

The main gapfilling routine is fgap_fill. Parameters are:

- pars: Structure passed from initialization routine.
- im: Image to be processed.
- nx,ny,nlead: Size and leading dimension of image.
- mask: Byte array of the same size as image with a mask giving the missing pixels. Values are:
  - 0: Pixel is valid.
  - 1: Pixel is missing and should be filled.
  - 2: Pixel is missing and should not be filled (eg. crop area).
- cnorm: Array of same size as image. Gives the white noise error magnification
- ierror: Array of same size as image. Gives the estimated interpolation error.

This routine parallizes using OpenMP internally. Should also work with calls from different external threads, but this is untested.

5 Temporal Interpolation

This code interpolates filtergrams in time. The method is based on a Wiener interpolation where an averaged single pixel spectrum from a MDI filtergram is assumed. The algorithm will attempt to to interpolate even if there are missing pixels.

A single routine (tinterpolate) does the work. Arguments are:

- nsample: Number of images.
- tsample: Times for each of the images.
- tint: Time to interpolate to.
- nconst: Number of polynomial constraints to apply. 0=none, 1=constant preserved, 2 linear preserved, etc.

- images: Array of pointers to the input images.
- masks: Array of pointers to the masks. 0 for a pixel means good, 1 means missing.
- image_out: Interpolated image.
- nx,ny,nlead: Size and leading dimension of arrays.
- method: Method to use. Currently use 1.
- filenamep: Pointer to the name of the table used. Set to the name if method$\neq$0 and filenamep not NULL.

6 Temporal Averaging

To be done.