GriddingLib: Gridding-based interpolation in Matlab

F. Sigworth
14 February 2007

1. General idea of gridding

The problem is to provide proper interpolation for converting from one coordinate system to another. For data on Cartesian grids, the proper interpolation function is the sinc function which however has infinite support, and therefore uses many coefficients in an interpolating filter. The idea of gridding is twofold: to use the Kaiser kernel, which has a good tradeoff of minimum extent in the "time" and "frequency" domains; and to first perform a resampling of the data to a finer grid before applying the kernel. The result is interpolation using only a few, say 3-9 points in the kernel, but with results very nearly approximating the sinc interpolation.

2. How to use it.

Suppose we have a 2D problem; we want to rotate an image X to obtain Xr. The gridding-based interpolation, performed by the grotate() function involves

a. Pre-multiply X by a compensation function. This boosts the amplitudes of pixels farthest from the center of the image, to correct for "roll-off" of the frequency response in the pass-band of the kernel filter.

b. Pad X with zeroes, and take its Fourier transform. I use padfactor=1.25 which means that a 64 x 64 image has zero-valued pixels added around the perimeter to make an 80 x 80 image. The resulting FT, which we'll call X', is 80 x 80 and is more finely sampled in frequency space than it would have been without padding of X.

c. Do the resampling in frequency space. We use a Kaiser kernel to interpolate values in the old coordinate system. If we are doing a rotation, the new coordinate system is the same size (80 x 80) and sampled at the same (fine) sampling density as the old one. The result is the rotated Fourier transform Xr'.

d. Do the inverse Fourier transform and remove the padding. The result is an output image Xr the same size as the original, but very precisely rotated.

--Note that the rotation works well only if X is band-limited and space-limited. That is, it and it Fourier transform must have (essentially) zero power in pixels outside a disc of 64 pixel diameter. Otherwise there will be artifacts.

--Note also that the compensation function could be applied just as well at the end as at the beginning, because it is circularly symmetric. However, when converting to and from a higher dimension (i.e. removing or inserting a slice) the compensation has to be done in the higher dimension.

--The whole procedure can be carried out without the pre-compensation, but then the kernel has to be a better approximation to a sinc function. In that case we use padfactor=2 and a different kernel, the convolution of a Kaiser function with sinc. The kernel must be larger to assure both flat passband response and steep cutoff of the effective lowpass filter. We call this 'sinc' mode, as opposed to 'grid' mode.

3. Making projections and backprojections

It is often necessary to make 2D projections from a 3D volume. Here is how you can do it.

```
n=size(m,1);  % m is an n x n x n real volume. n must be a multiple of 4.
ks=3;
comp=gridMakePreComp(n,ks);  % Make the pre-compensation function (a 1D array)
F3=gridMakePaddedFT(m,'grid',comp);  % get the 3D fft in a form for slicing.
    % Once the 3D FT has been made, you can use it repeatedly to make a projection:
P2=gridExtractPlane(F3,angs,ks);  % angs is a 3x1 vector of Euler angles (radians)
img=gridRecoverRealImage(P2);     % get the un-padded projection image
```

Similarly you can make 1D projections from a 2D image

```
ks=5;
comp=gridMakePreComp(n,ks);  % Make the pre-compensation function (a 1D array)
F2=gridMakePaddedFT(m2,'grid',comp);  % get the 3D fft in a form for slicing.
    % Use the 2D FT to make a 1D projection:
P1=gridExtractLine(F2,ang,ks);  % angs is the in-plane angle (radians)
img=gridRecoverRealImage(P1);   % get the un-padded projection line
```

To do a backprojection, you insert planes or lines into a FT domain. Here is back-projecting a 2D image into a 3D volume.

```
% Prepare compensation and a 3D Fourier volume
comp=gridMakePreComp(n,3);  % default is kernelsize=3.
vol1=gridMakeNullFT(n,3);
% Make FT of one image and insert it
fslice=gridMakePaddedFT(img);
vol1=gridInsertPlane(fslice,vol1,angles);
% When all the planes have been insered, transform back to a real-space volume.
revol=gridRecoverRealImage(vol1,comp);
```

4. Data structures and procedures

In addition to the input and output data arrays, gridding makes use of the compensation function and the FT of the padded image.

The compensation function is created by
comp = gridMakePreComp(n, kernelwidth);

Here n is the input dimension (64 in our example), and kernelwidth is an odd number like 3 or 5. The returned value comp is a 1D array--because of separation of variables it can be expanded to any number of dimensions on the fly.

Compensating, padding and FFT are performed by

P=gridMakePaddedFT(X, mode, comp);
The mode argument is a string (takes values 'grid' or 'sinc'). This allows the same routine to be used in either way. The returned value is a structure which contains variables that help with the subsequent processing steps.

P.n is the original dimension (e.g. 64).
P.np is the padded dimension (e.g. 80).
P.np1 is a further-padded dimension, useful to avoid problems at the edges during interpolation. (e.g. 96)
P.PadFT is the Fourier transform itself, X'. It is a 1D, 2D or 3D array (depending on the dimension of X) of size **np1**.

The same structure is used for 1D, 2D or 3D problems; we know the number of dimensions by the dimensionality of P.PadFT, which in turn is obtained from the dimensionality of X.

Often we want an "empty" plane or volume into which we add lines or planes during the reconstruction process. A function to do this is

P=gridMakeNullFT(n, dims, mode);
(….I want to change this to "MakeConstFT" to fill it with a constant instead.)

To compute projections or do reconstruction, we want to determine or increment the values along a line or plane. These are done by

p1=gridExtractLine(P, theta, kernelsize);
--from the 2D structure P, obtain the 1D structure p1.

P2=gridInsertLine(p1, P, theta, kernelsize);
--To the 2D structure P, add the values along a line at angle theta.

p2=gridExtractPlane(P, angles, kernelsize);
--from the 3D structure, obtain the values of the FT along the plane determined by the Euler angles.

P3=gridInsertPlane(p2, P, angles, kernelsize);
--To the 3D structure add values along the plane p2 at the given Euler angles.

Finally, to recover the real-space lines, planes or volumes, use

m=gridRecoverRealImage(P, postcomp);
This does the inverse FT and un-padding. If postcomp is given and is different from 1, it multiplies the result.

Here are two more functions using the gridding principles:
Xr = grotate(X, theta, kernelsize);
Xr = grotate3(X, angles, kernelsize);

2D and 3D rotation.

A note about complex value storage. To save space and time, the values in Fourier space (e.g. P.PadFT) are stored as cosine-and-sine values. That is, CAS(z)=Re(z) + Im(z). This works because the FT of a real function results in an even Re(z) and an odd Im(z), allowing a unique decoding of the values when needed.

4. Low-level functions

z=FromCAS(r);
r=ToCAS(z);
Convert between CAS variables and complex. Works with 1D, 2D or 3D arrays.

Other low-level functions:
kaiser
gridGetAlpha
gridMakeKaiserTable
gridPadFactor