1 roundgridfun.m

```matlab
function [val,numpts]=roundgridfun(varargin)
% ROUNDGRIDFUN grids data using a nearest neighbor gridding technique
%   This function is a very simplistic gridding function which effectively
%   bins the data to find the nearest grid node, then performs the input
%   function on all of the points for each node.
%
%   The grid nodes must be evenly spaced in each dimension.
%
%   The function works for ND datasets (tested up to 5D)
%
%   1D: [val,numpts]=roundgridfun(x,y,xg,@fun);           % Vector
%   2D: [val,numpts]=roundgridfun(x,y,z,xg,yg,@fun);      % Surface
%   3D: [val,numpts]=roundgridfun(x,y,z,I,xg,yg,zg,@fun); % Voxels
%
%   The common input functions are @mean, @min, @max, @std, @var
%   The function handle @(x) {x} can be used to return a cell array with
%   all of the data in each grid node bin
%
% Inputs:
%    - x : nx1 : vector of x values
%    - y : nx1 : vector of y values
%    - z : nx1 : vector of z values (optional)
%    - I : nx1 : vector of I values (optional)
%    - xg: mx1 or array : vector or *array of x grid values
%    - yg: mx1 or array : vector or *array of y grid values (optional)
%    - zg: mx1 or array : vector or *array of z grid values (optional)
%    - fun: @function : handle to a function that
%         *array can be a multidimensional array from meshgrid, but must just
%          vary along one dimension
%
% Outputs:
%   - val    : values returned from @fun, nan if no points at grid node
%   - numpts : the number of points at each grid node
```

1.1 Motivation/Concept

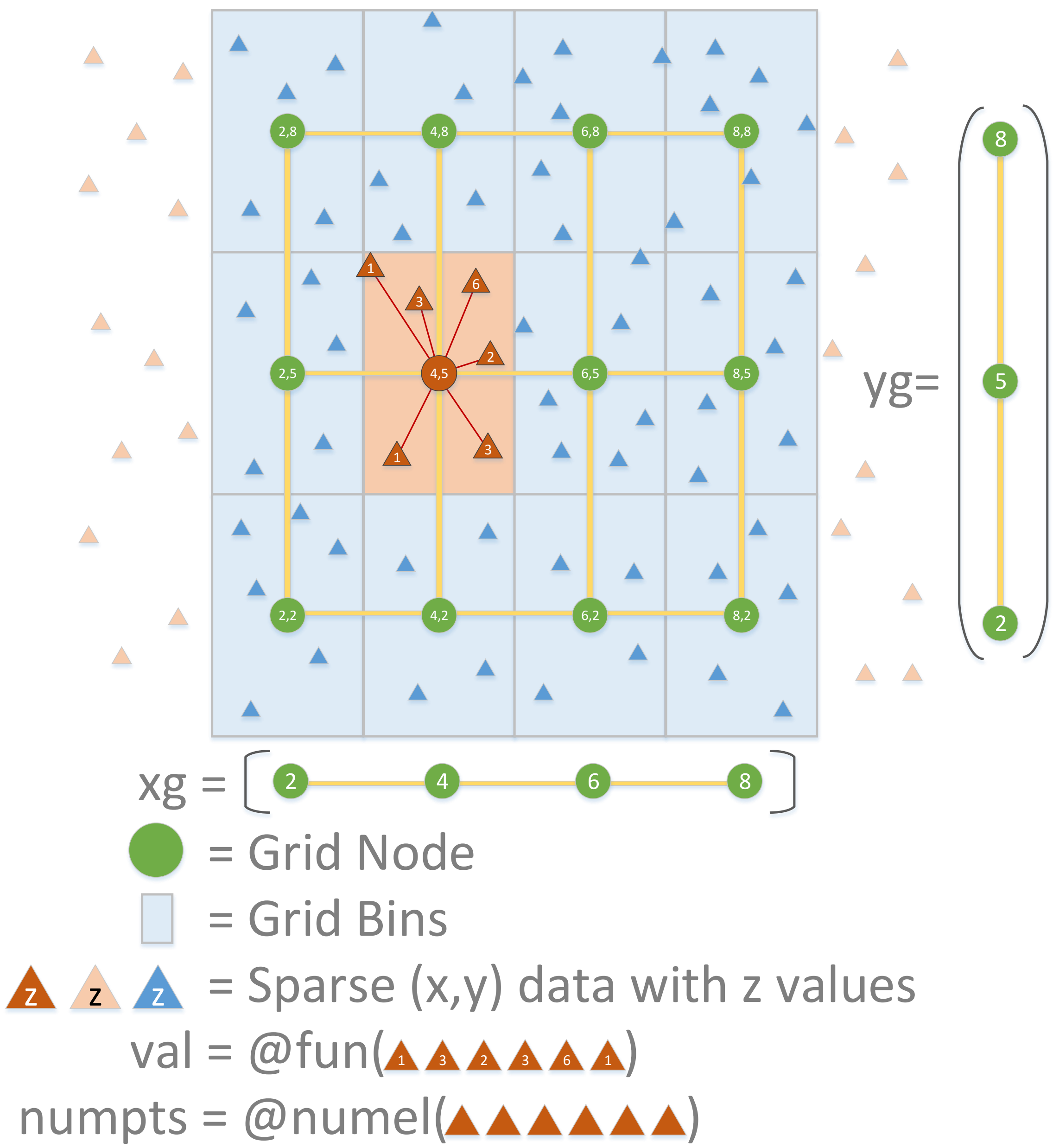
This function was created as a fast way to grid dense data in Matlab using a nearest neighbor binning technique. Often when gridding high density data to a lower density grid, a Delaunay Triangulation does not make sense as your just effectively down-sampling your dataset, and an IDW, while certainly a more elegant and accurate solution, can sometimes be fairly time consuming in matlab. Additionally, *roundgridfun.m* works on N-D datasets and has the ability for alternative function handles to be passed in, which can be used to do more than just calculate the mean of the points in a bin.

1.2 Math

The input for the algorithm is the raw data, the grid nodes, and the function handle to operate on each computed bin. The algorithm computes which points lie in the bin associated with each grid node, and uses the function handle to compute a value for each bin. Note, to improve processing time, roundgridfun is written with the assumption that the grid spacing along each dimension is constant.

The data is converted from the raw coordinate system to an index based coordinate system based on the grid. The data is then rounded (hence, roundgridfun) to determine the associated grid node that each raw data point should be mapped to. Data that is mapped to a bin that is not associated with a grid node is omitted from the results.

A conceptual diagram of the algorithm for gridding 3D data onto a 2D Grid is shown below:



1.3 Inputs

1.3.1 x

vector of x data points

1.3.2 y

vector of y data points

1.3.3 z

vector of z data points

1.3.4 xg

Either a vector of x grid nodes, or a meshgrid 2d array of x grid nodes

1.3.5 yg

Either a vector of y grid nodes, or a meshgrid 2d array of y grid nodes

1.3.6 fun

Function handle which inputs a vector of x, and outputs a single value
eg. @mean, @min, @std, @max, @var, @(x) x

1.4 Outputs

1.4.1 val

Value from @fun computed for each grid node

1.4.2 numpts

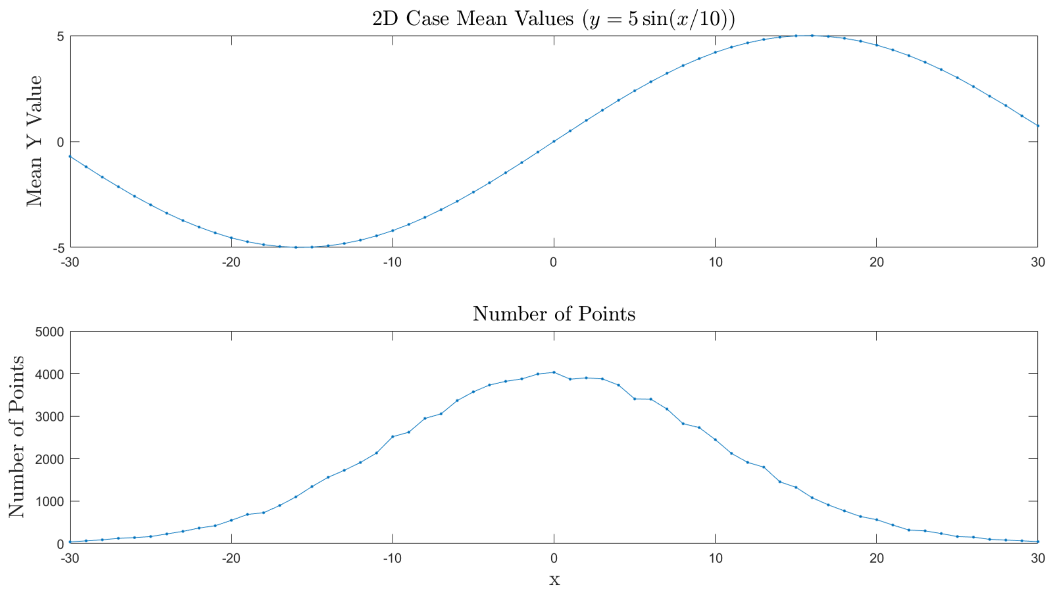
Number of points in each grid node

1.5 Examples *exampleRoundgridfun.m*

2D Case (@mean)

This example demonstrates gridding 2D (x,y) data onto an x grid using the mean values in each grid bin.

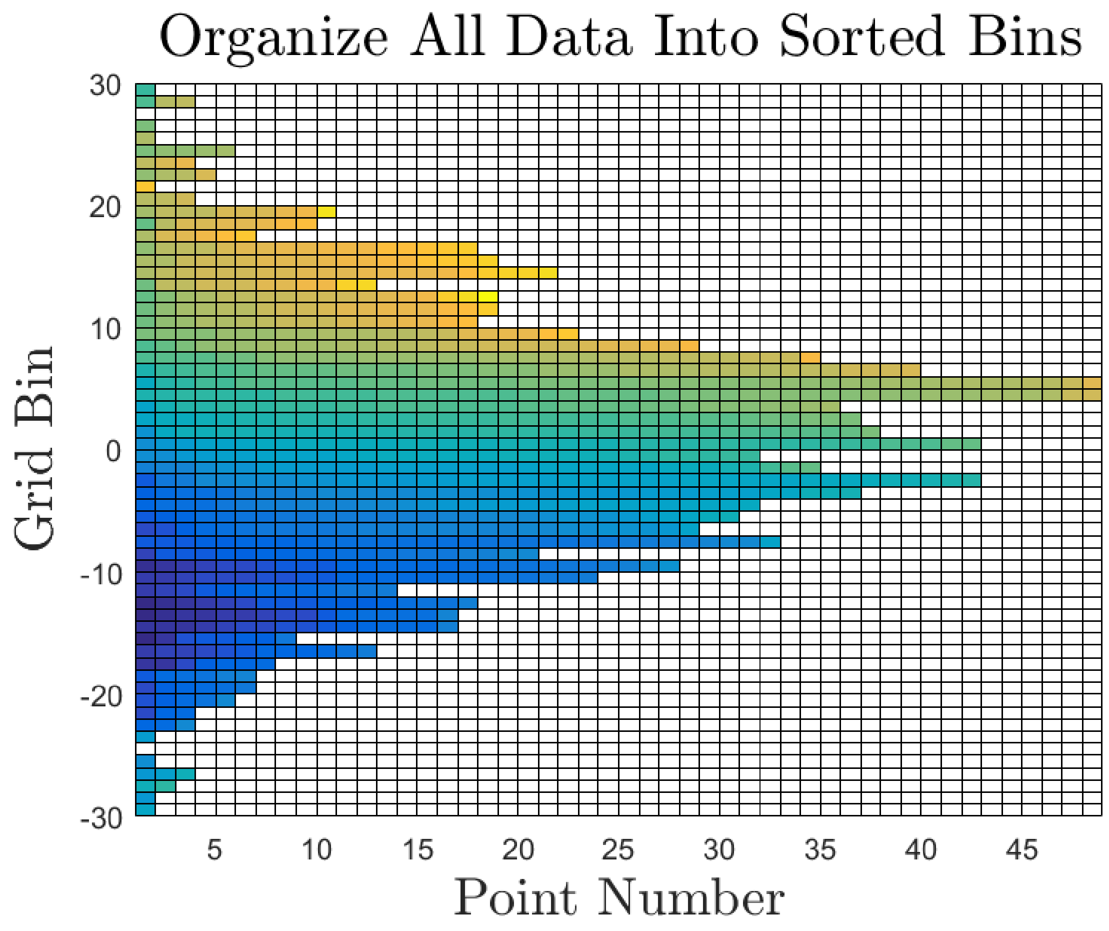
```
%% EXAMPLE 2D
npts1 = 100000;
x = randn(npts1,1)*10;
y = sin(.1*x)*5;
xgi = -30:1:30;

[val,numpts]=roundgridfun(x,y,xgi,@mean);
```



2D Case (cell array)

This example demonstrates how all of the y values in each bin can be returned by returning a cell from the function input. Here, the y values in each bin are sorted, and assembled in a 2D matrix for visualization.

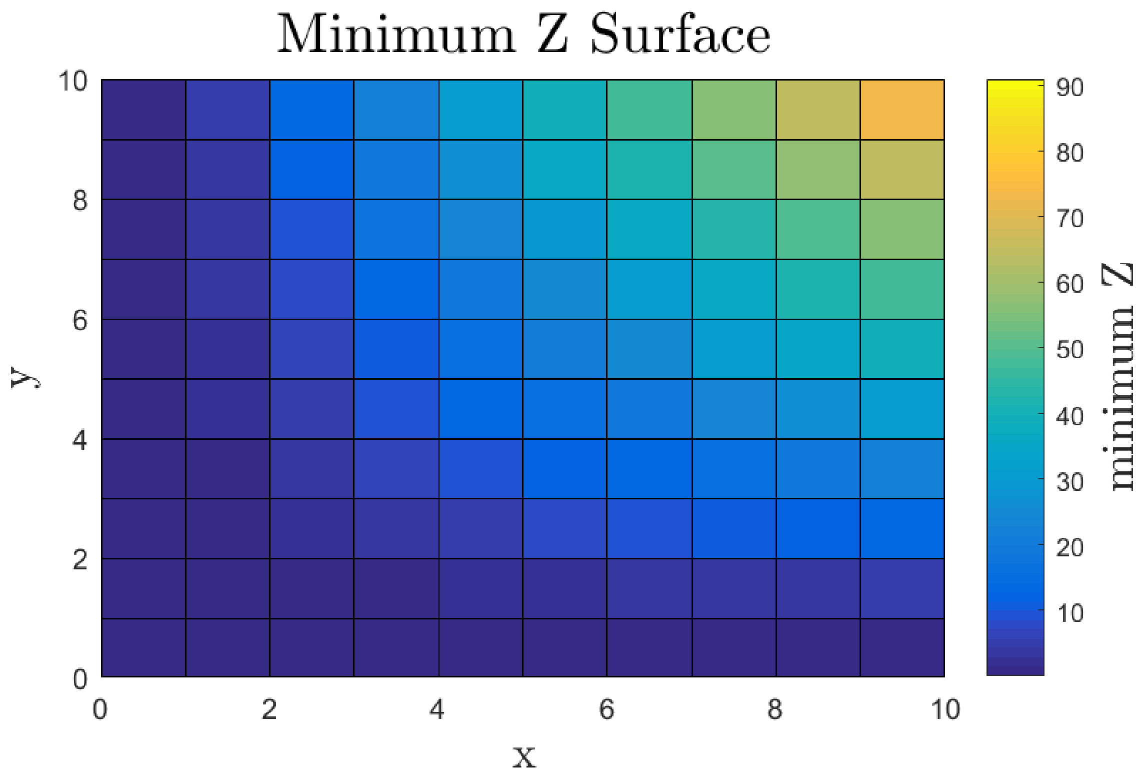
```
%% Example 2D That Returns and Sorts a Cell Array
npts1b = 1000;
x = randn(npts1b,1)*10;
y = sin(.1*x)*5+randn(npts1b,1);
xgi = -30:1:30;

[val,numpts]=roundgridfun(x,y,xgi,@(x) {sort(x)});

%turn all data into a matrix from a
maxpts = max(numpts(:));
A = nan(numel(xgi),maxpts);
for i=1:numel(val)
    A(i,1:numpts(i)) = cell2mat(val(i));
end
```



3D Case (@min)

This examples grids 3D data using the min of all of the data in each bin.

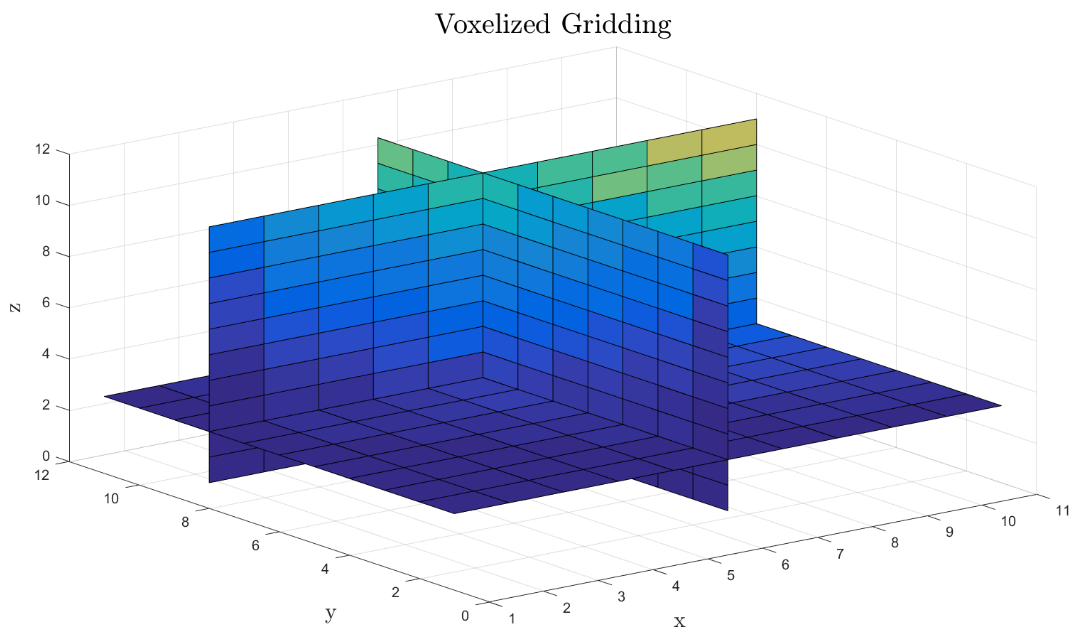
```
%% EXAMPLE 3D
npts2 = 10000;
x = rand(npts2,1)*10;
y = rand(npts2,1)*10;
z = x.*y;
xgi = 0:1:10; ygi = 0:1:10;

val=roundgridfun(x,y,z,xgi,ygi,@min); % Minimum Z Surface
```



4D Case (@mean)

This example demonstrates the creation of rectangular prism voxels.

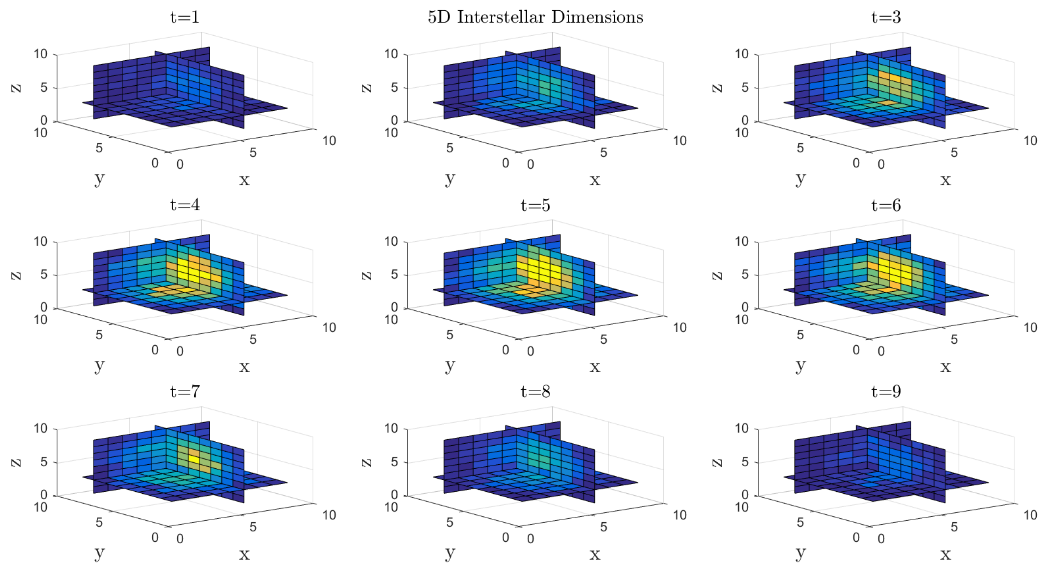
```matlab
%% EXAMPLE 4D *uses meshgrid array
x = rand(100000,1)*10;
y = rand(100000,1)*10;
z = rand(100000,1)*10;
I=x.^2.*y.*z;
xgi = 0:1:10;
ygi = 0:1:10;
zgi = 0:1:10;
[xg,yg,zg]=meshgrid(xgi,ygi,zgi);
val=roundgridfun(x,y,z,I,xg,yg,zg,@std); % Mean Intensity voxels
```



5D Case (@std)

This example demonstrates how higher order data can also be gridded into higher dimensions.

```
%% Example 5D
npts4 = 100000;
xn = randn(npts4,5)*2+5;
xni = 1:1:9;
[xg1,xg2,xg3,xg4]=ndgrid(xni,xni,xni,xni);
[val,numpts]=roundgridfun(xn(:,1),xn(:,2),xn(:,3),xn(:,4),xn(:,5),...
    xg1,xg2,xg3,xg4,@std);%interstellar dimensions
```


5D Interstellar Dimensions
t=1
t=3
t=4
t=5
t=6
t=7
t=8
t=9
x
y
z