

University of Cincinnati  
Department of Electrical & Computer Engineering and Computer Science

**20 ENFD 112 – Fundamentals of Programming**

LABORATORY 4: MATRIX DATA, VISUALIZATION

Spring 2008

## 1. Objective

The objective is to practice the use of MATLAB matrices on a common engineering task and to visualize the result.

## 2. Background

Some number of wells are sunk in various positions and at various depths. Sensors are sent down the wells to their respective depths. These sensors read concentration levels of a specified substance. All sensors report concentrations simultaneously at intervals which could range anywhere from 1 day to 1 month. These data are used to track the flow of the substance over time. This is best done through some graphical visualization of the data.

Unfortunately, there are typically too few wells to give an interesting visualization without some processing. Typically, the processing amounts to gridding a volume enclosing the wells and estimating the concentration at each grid point from the data returned by all the wells. This is called *interpolation*.

There are several ways to do interpolation, each with its own advantages and disadvantages. Obviously, the interpolated value at a point should be influenced more by closer wells than distant wells. But there are several ways to implement this idea, all starting with a general expression for estimating the concentration at an arbitrary point  $u$  in three dimensional space. We develop such an expression now.

Let  $Z(u)$  be the measured concentration at point  $u$  (there is a sensor here) and  $Z^*(u)$  denote an estimated concentration at point  $u$  (there does not have to be a sensor here). The simplest estimate is the weighted sum of measured concentrations. This is written

$$Z^*(u) = \sum_{i=1}^n \lambda_i \cdot Z(u_i)$$

where  $n$  is the number of sensors,  $u_i$  is the position of the sensor in the  $i^{\text{th}}$  well,  $Z(u_i)$  is the concentration reported by that sensor, and  $\lambda_i$  is a function of the distance between  $u$  and  $u_i$ , among other things. The following presents two ways to compute  $\lambda_i$ . We use simple Euclidean distances throughout. In other words, the distance between two points  $u_1 = \langle x_1, y_1, z_1 \rangle$  and  $u_2 = \langle x_2, y_2, z_2 \rangle$  in 3-space is

$$\text{dist}(u_1, u_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

### 2.1 Polygonal Interpolation

Let  $u_x$  be such that  $\text{dist}(u, u_x) = \min_{i=1}^n \text{dist}(u, u_i)$ . To find  $Z^*(u)$  use

$$\lambda_i = \begin{cases} 1 & \text{if } i = x \\ 0 & \text{otherwise} \end{cases}$$

which means use the concentration at the nearest measured point  $u_x$  for the estimated concentration at  $u$ . Obviously, we can do better.

## 2.2 Inverse Distance Squared Interpolation

For each measured point compute

$$\mathcal{W}_i = \frac{1}{c + \text{dist}(u, u_i)^2}$$

where  $c$  is a small constant needed to prevent overflow if  $u$  happens to co-locate with a measured point. Then

$$\lambda_i = \frac{\mathcal{W}_i}{\sum_{i=1}^n \mathcal{W}_i}$$

This has the advantage of producing a correct estimate at all the measured points. Unfortunately, the error at other points, particularly those points that are far from any wells, can be great. There are other interpolation techniques that address this concern but they are beyond the scope of this course.

## 3. Problem

Write a MATLAB m-file called `interp.m` which reads a file of data points, asks the user for interpolation information, interpolates on a three dimensional grid, and displays an isometric view of the grid values. The format of the file is as follows: The first line contains the number of data points. Each remaining line is a data point. Each data point has four values:  $x$ ,  $y$ ,  $z$  coordinates and a magnitude at that coordinate, in that order from left to right. An example is the following:

```
3
12.2    1.34    16.34    29.1
10.1    2.10    15.34    22.34
9.21    4.34    12.45    19.23
```

The program `interp` will prompt the user for a filename (the name of the file that contains the data). The prompt looks like this:

```
Data file:
```

to which the user responds with a file name, for example like this:

```
Data file: plume.dat
```

The program then asks the user for an interpolation type as follows (showing the user has chosen Inverse Distance Squared):

```
Type of interpolation (1=Polygon, 2=Inv Dis Squ): 2
```

A volume must next be specified by the user. For the purposes of this exercise (to keep things simple) the volume will be a perfect cube (length, width, height are all the same) and the coordinates of one corner of the cube (typically the one closest to the origin) will have the same value. Thus, the user needs to be prompted only for a minimum coordinate value, a maximum coordinate value, and the number of grid points in each dimension. This should look as follows:

```
Range minimum: 0
Range maximum: 20
No. grid elements: 40
The type is Inverse Distance Squared
```

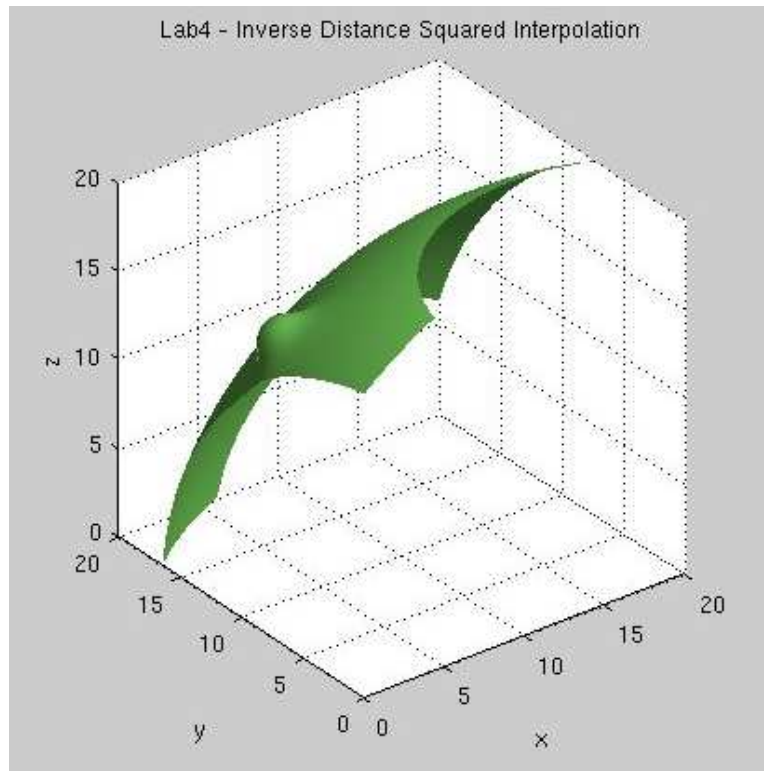


Figure 1: Sample interpolated output.

where the user has responded with the numbers 0, 20, and 40, and the program has displayed a string indicating the choice of Inverse Distance Squared interpolation. The program then displays an isometric plot of the interpolated data, for example, as shown in Figure 1. Observe the title, and labels on the axes.

## 4. Help

You may need some help in writing one or more segments of the program so we have organized the help into sections relating to particular functional portions of the program.

### 4.1 File and console input

The following line reads a string from the console and `filename` takes the value of the read string:

```
filename = input('Data file: ', 's');
```

where `Data file:` is the prompt that is displayed to the console. The following line opens a file:

```
fid = fopen(filename, 'r');
```

You will have to store file data in an array. You can create an array using the following:

```
dpts = zeros(n,4);
```

where `n` is the number of data points in the input file. Observe that `dpts` is a two dimensional array with `n` rows and 4 columns. The following line may be used to read a single entry in the file to an element of `dpts`:

```
dpts(i,j) = fscanf(fid, '%f', 1);
```

where the 1 in `fscanf` means read one file entry and the 'f' is a format specifier that means treat the entry as a real number.

## 4.2 Establish a gridded volume

The program asks for the minimum range, maximum range, and number of grid points using something similar to the following:

```
rn = input('Range minimum: ');
rx = input('Range maximum: ');
n  = input('No. grid elements: ');
```

From this information grid vectors in the x, y, and z directions may be obtained in the usual way as follows:

```
x = linspace(rn,rx,n);
y = linspace(rn,rx,n);
z = linspace(rn,rx,n);
```

The grid matrix (three dimensions) of interpolated values can be created using the following:

```
v = zeros(n,n,n);
```

The elements of this matrix are to be computed before `isosurface` is applied to get the visualization requested.

## 4.3 Visualization

To display an isometric surface from the gridded data you can use `isosurface`, for example as follows:

```
isosurface(x,y,z,v);
```

There are a lot of variations possible when using `isosurface` that allow you to choose color, threshold, and so on. Check MATLAB help for more information.

Setting values for `v` is, unfortunately, most easily accomplished using three nested `for` loops, one for each dimension, instead of `meshgrid`, and a fourth inner loop which sums the contributions to the `v` value of each of the `n` data points. That is, the structure of the code is something like this where vector `Z` contains the values of all the data points and `lambda` is a vector that will hold contributions  $\lambda_i$  as described in Sections 2.1 and 2.2:

```
lambda = zeros(1,n);
for i=1:n
    for j=1:n
        for k=1:n
            some initialization of variables may be required here
            for m=1:n
                lambda(m) = contribution of the mth data point to v(i,j,k)
            end
            v(i,j,k) = lambda*Z';
        end
    end
end
```

Computing the contribution of the  $m$ th data point should require no more than two lines of code that depends on  $x(i)$ ,  $y(j)$ ,  $z(k)$ ,  $dpts(m,1)$ ,  $dpts(m,2)$ , and  $dpts(m,3)$ . You can save on the loop for  $m=1:n$  by using  $dpts(:,1)$  and so on. You can save on the other for loops by using `meshgrid`, but for 3 dimensions you may find it hard to do. If you figure it out, though, you will receive 25 stars.

You can set labels and title using, for example:

```
grid;  
xlabel('x');  
title('This is a title');
```

## 5. Submission

Submit `interp.m`, your solution to the problem of Section 3, on or before April 27 using blackboard. See the course webpage at

<http://gauss.ececs.uc.edu/Courses/HTML/E112.html>

for instructions.