# Lab #2b -- 2D Function Evaluation and Plotting in Matlab

## Overview

This lab will focus on various techniques for evaluating and plotting of 2-D functions -- that is, working with functions that have two independent variables, such as f(x,y) where f depends on both the x and y variables. Since the quantity of interest is a function of two variables, it should be stored as a 2-D array (with proper initialization, if needed) within your Matlab code. As discussed in the class lectures, there are three ways to evaluate functions of this type (using scalar, vector, and matrix arithmetic) and two general plotting techniques that emphasize either a quantitative or qualitative view of the functional dependence. This lab will highlight all these evaluation and plotting options with a single instructor-led example, and then you will be given time to try two other problems on your own.

By the end of this lab you should have sufficient background to tackle all the tasks requested as part of HW2b, and be quite comfortable with a variety of techniques for the evaluation and visualization of 2-D functions with Matlab…

## Part 1 -- Instructor-Led Example

Consider the simple 2-D function f(x,y) given below:

$$f(x,y) = xye^{-(x^2+y^2)} \qquad \text{for } -3 \le x \le 3 \quad \text{and} \quad -2 \le y \le 2$$

Our goal will be to evaluate and plot this function in a variety of ways.

Let's first focus on how to evaluate this 2-D function in Matlab. As you know from our discussions in class, there are three ways we could do this -- using **scalar**, **vector**, or **matrix arithmetic**. Since Matlab has been optimized to work with arrays and matrices, the calculational efficiency is greatest for the matrix approach (usually), then the vector method, and finally the slowest method is the scalar approach. However, knowing how to use all three methods is important, since it is not always obvious or even possible to take advantage of the computational efficiency of the matrix (and sometimes vector) calculations within Matlab -- but the scalar approach always works!!!

**Scalar Evaluation:** If you visualize the continuous x and y independent variables in the above expression as discrete quantities, say $x_i$ and $y_j$, then the continuous function f(x,y) also becomes discrete and can be written as $f_{ij} = f(x_i,y_j)$, or

$$f_{ij} = x_i y_j e^{-(x_i^2+y_j^2)}$$

In this form, it is easy to visualize the evaluation of this function for all $x_i$ and $y_j$ within two nested ***for … end*** loops (one loop for each discrete independent variable), as follows:

```
      y = [-1.5 -1 0 1 1.5];  Ny = length(y);  % few specific values of y (snapshots)
      Nx = 201;  x = linspace(-3,3,Nx)';       % define x values (col vector)
%
%   Case 1  --  scalar arithmetic
      f1 = zeros(Nx,Ny);                    % initialize storage space for f(x,y)
      for j = 1:Ny
        for i = 1:Nx
          f1(i,j) = x(i)*y(j)*exp(-(x(i)^2 + y(j)^2)); % do needed calcs (no dots)
        end
      end
```

Notice that there are no dots (i.e., no element by element arithmetic) in this code since every calculation is with scalars. Note also that every instance of x uses x(i) since i is the index of the looping structure that goes from i = 1 to i = Nx in increments of one (i is simply a counter that gets incremented by 1 on each pass through the inner loop). A similar statement can be made for the j variable which is indexed from 1 to Ny (i.e. the number of values in the y vector) in the outer loop, again with increments of unity. Finally, the discrete function value, $f_{ij}$, is saved within the nested loops for each individual i and j value and, since f1(i,j) is computed within the loop, we use the ***zeros*** command before the looping structures to pre-allocate space for this 2-D array.

**Vector Evaluation:** Now, for the vector approach, we simply replace the inner loop with element by element computations for the functional dependence on the x vector, being sure to continue to use y(j) wherever y occurs in the equation -- because we are still doing discrete scalar arithmetic with this variable.

```
%
%   Case 2  --  vector arithmetic
      f2 = zeros(Nx,Ny);                      % initialize storage space for f(x,y)
      for j = 1:Ny
        f2(:,j) = x*y(j).*exp(-(x.*x + y(j)^2));  % do needed vector calcs  (need dots here)
      end
```

Note now that only one loop is needed, that proper dot placement is essential to do vector arithmetic with the x vector, and that the f2(:,j) notation implies that **all** the values of the $j^{th}$ column of the f array are computed at once and placed in the appropriate column of the 2-D array. Also note that initialization is needed because the f2 variable is still computed inside a looping structure (thus we still need to define "how big" the array will be prior to starting the loop).

**Note #1:** Make sure you understand the above coding, since many of our subsequent examples in this course will use this **vector approach**.

**Note #2:** Does everyone see why you would get a "inner matrix dimensions must agree" error if the dot location was given as shown below in **red** instead of the correct location in **black** in the above code? `f2(:,j) = x.*y(j)*exp(-(x.*x + y(j)^2));` Clearly the proper dot placement is absolutely essential!!!

**Matrix Evaluation:** If the goal is a qualitative visualization of the data, then a fine grid should be used for both independent variables (so that we don't have the "jaggies" in either direction"). In this case (i.e., when the Nx and Ny values are both relatively large), it is often most efficient to do full 2-D array calculations wherever possible. As we have already seen in class, Matlab's ***meshgrid*** command is used to convert the two independent vectors (x and y in this example) into two 2-D arrays, X and Y, of exactly the same size that define the coordinates of Ny by Nx grid points over the x,y domain of interest (see ***help meshgrid***, if needed). Now, with the Ny by Nx arrays of x,y points available (i.e., X and Y in this example), one can easily do element by element arithmetic with no looping needed at all. The code here is extremely simple, as follows

```
      Ny = 51;  y = linspace(-2,2,Ny);   % define y values   (fine grid for 3-D qualitative view)
      Nx = 101;  x = linspace(-3,3,Nx)'; % define x values
%   Case 3  --  array/matrix arithmetic
      [X,Y] = meshgrid(x,y);
      f3 = X.*Y.*exp(-(X.*X + Y.^2));
```

but it is important that the user understand what is really happening here (again, refer to the **help** capability in Matlab and/or to your classroom notes on this subject, as needed).

**Note:** The lab instructor will briefly review the **scalar**, **vector**, and **matrix** approaches for evaluating a 2-D function in Matlab using the above example -- but make sure you ask questions if you don't fully understand what is happening in each of these cases!!!

**Visualization:** As noted in class, there are two general goals -- **quantitative** or **qualitative** -- for visualization of a particular function. For example, if you want to extract numerical data from a plot of a 2-D function (i.e., a **quantitative view**), then a 2-D snapshot view that presents a "family" of curves is usually the best option (or possibly a contour map as shown in the actual Matlab **fxy1_lab2b.m** code). In this case, one might plot $f(x,y_i)$ for several specific values of $y_i$. This approach gives a continuous view of f(x) for some reasonable number of discrete y values (usually 3-6, but this depends on the particular application). With a well-labeled plot, this approach can be very effective in presenting the functional dependence of the function f on both independent variables in a quantitative fashion.

In contrast, if the goal is simply to present the general functional behavior with little or no interest in obtaining detailed quantitative information from the plots (i.e., a **qualitative view**), then a 3-D view may be the best way to go. Matlab's has several presentation formats here with many user options, and the views can indeed be quite impressive and effective in displaying the desired generic functional dependence. Some of the available options were demonstrated in class with the **maxwell_2b** code, and a few similar views will again be demonstrated here for this particular example (see the **fxy1_lab2b.m** code). However, the only way to get a good feel for what is possible is to try some of the many possible combinations yourself -- so don't be timid and have a little fun…

**Part 2 -- Now it is YOUR turn to write some Matlab Scripts of your own**

In the time remaining in this lab, you should get started with evaluating and plotting the following two functions using Matlab. Make sure that you store the function information in 2-D arrays, that the requested plots look good, and that they are as informative as possible. Follow any special instructions that are given. Please complete these tasks outside of class if you run out of time during the lab period.

a. The Gaussian or normal distribution is often used to describe the probability of a particular event occurring. This distribution function can be written as

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left[\frac{(x-\mu)^2}{2\sigma^2}\right]}$$

where $\mu$ is the mean value, $\sigma$ is the standard deviation of the distribution, and x is the actual value of the quantity of interest -- that is, p(x)dx gives the probability that the quantity of interest will have a value that falls within a small interval dx about x. If p(x) is plotted against x, a "bell-shaped" curve should result. The mean of the distribution, $\mu$, is the point about which the bell-shaped curve is centered, and the standard deviation, $\sigma$, is a measure of the spread of the probability distribution about the mean.

With this background, evaluate and plot the normal distribution for the following three (3) cases:

**Case 1:** $\mu = 0, \quad \sigma = 1$ **Case 2:** $\mu = 2, \quad \sigma = 1$ **Case 3:** $\mu = 0, \quad \sigma = 2$

where, for all cases, $-8 < x < 8$. For this exercise, all three curves should be plotted on the same axis -- with appropriate labeling, of course. Also, be sure to briefly discuss your results -- i.e. are the curves as expected? Explain…

**Note:** For this situation, we can view the probability distribution, p(x,n), as a function of two variables, x and n, where x is the real independent variable and n is simply an index to track the case number of interest. Since this lab exercise clearly focuses on the evaluation and plotting of functions of two variables, be sure that you write your Matlab code accordingly and store the resultant probabilities in a 2-D array, p(x,n).

b. Consider the following function of two variables,

$$f(x,y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2$$

over the domain given by $-5 \le x \le 5$ and $-5 \le y \le 5$

The goal here is to find the minima of this function using only graphical techniques (i.e. by visual inspection of a variety of different plot types). As you know, Matlab has several built-in routines for generating a variety of different visualizations of the same function. As part of your study of the function given here, you might try a few options (see the ***meshgrid***, ***surf***, and ***contour***/***clabel*** commands, for example) -- and don't hesitate to be a little creative here. In the end, however, be sure to present one or more plots that clearly show the approximate locations of any minima of f(x,y) in the given portion of the x-y plane. Overall, in your opinion, which is the best presentation format for this problem? Explain…