The Justin Guide to Matlab for MATH 241 Part 2.

Table of Contents

M-Files and Publishing

Your second project (and third) will be submitted as a published M-file. This is pretty and far less complicated that it might sound. Basically an M-file is just a text file which contains a series of Matlab commands. You can create and edit one by simply typing

```
edit project.m
```

in the Matlab command window. When you do this an editor will open up and you can edit the file. Inside an M-file there are various ways to make things pretty but all you need to do for this project is make sure that each of your commands is separated by a blank line followed by a line containing %% followed by a blank line. Thus for example:

This will ensure that your output is what you would expect. Basically the %% divide the document into cells and each cell will contain the command and its output, separate from all the rest.

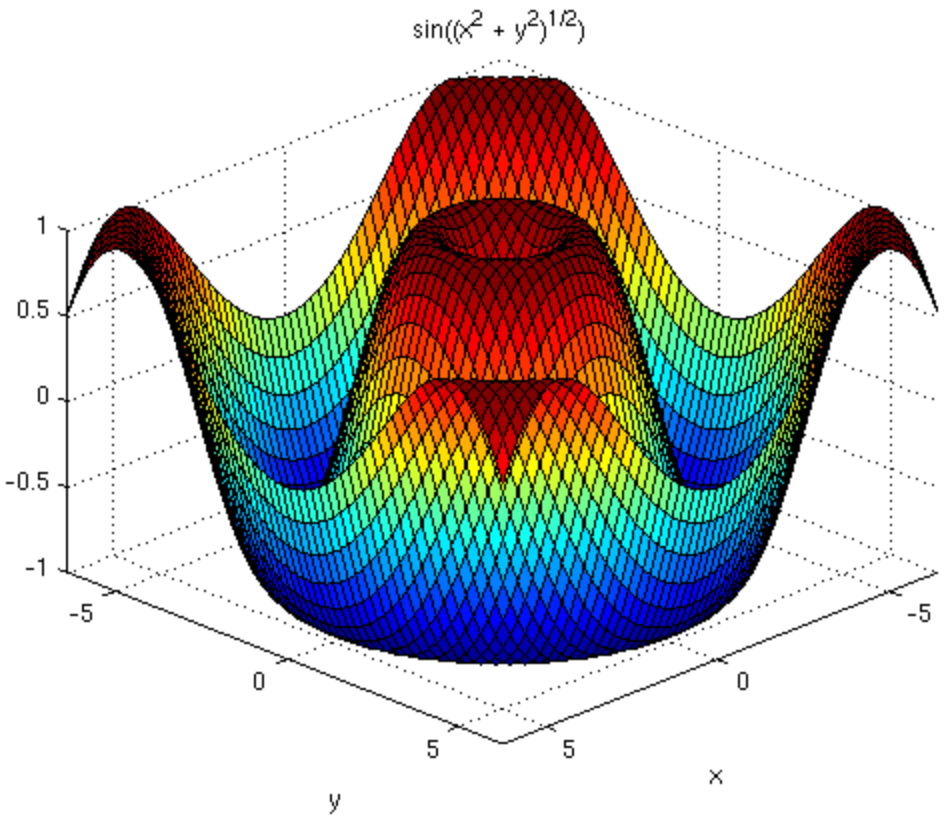
Publishing an M-file basically takes the file, runs the Matlab commands inside it and then creates an HTML document which contains not only the commands but also the output of the commands including pictures. This guide is actually a published M-file. You can publish your M-file by pressing the publish button on the toolbar. This button looks like a swiftly moving letter.

For your project simply publish your M-file and print the resulting HTML document.

Plotting Surfaces

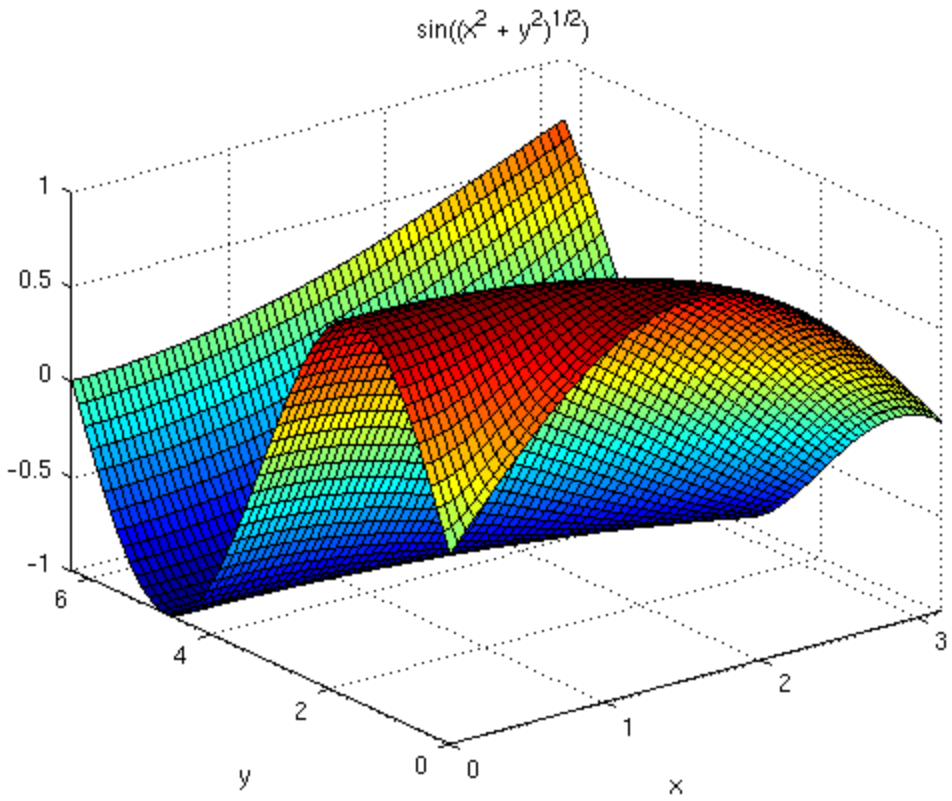
In section 13.1 we learned how to graph functions of two variables. Matlab can do this easily with the `ezsurf` command. Here's an example:

```
syms x y;
ezsurf(sin(sqrt(x^2+y^2)))
view([10 10 10])
```



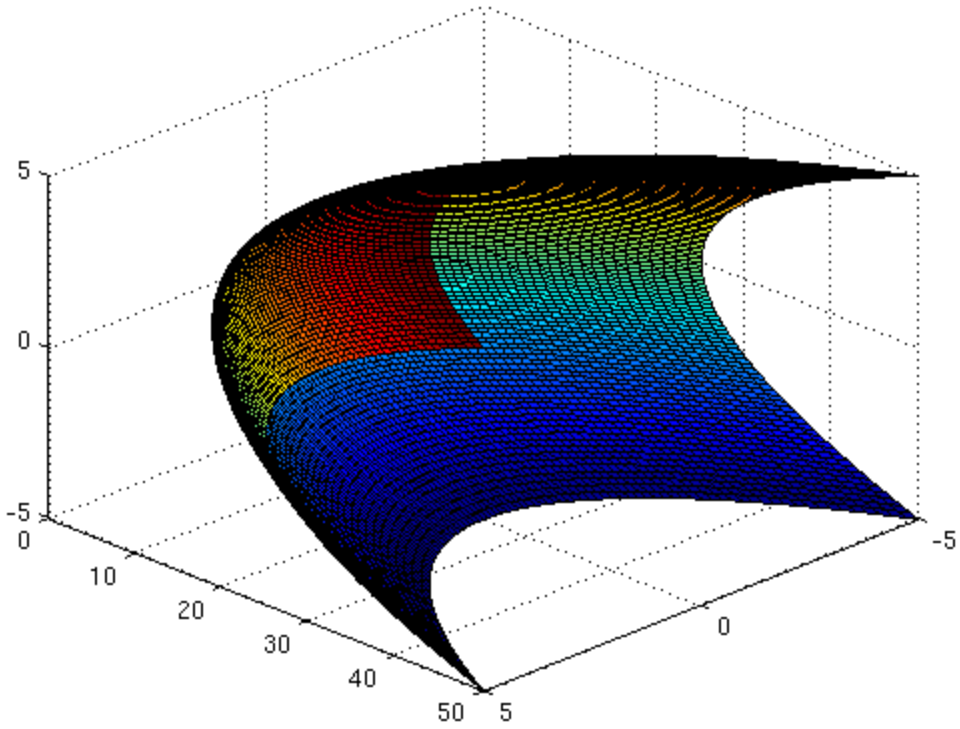
If we wish to give it specific `x` and `y` like `0<=x<=pi` and `0<=y<=2pi` we can do that too:

```
ezsurf(sin(sqrt(x^2+y^2)),[0,pi,0,2*pi])
```



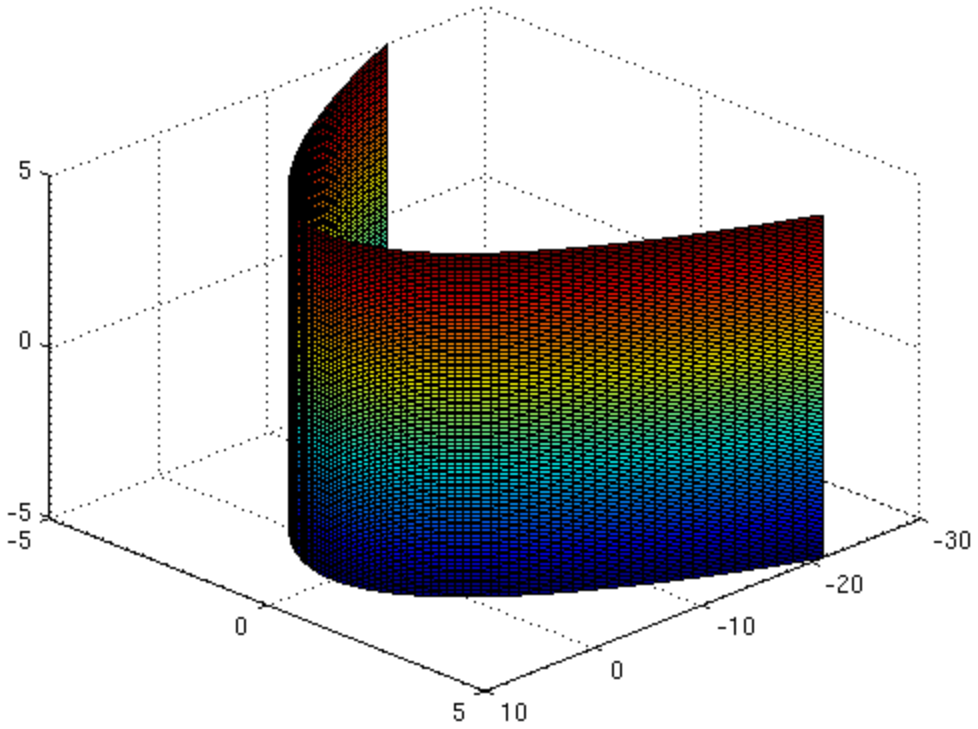
Plotting surfaces which are not functions of `(x,y)` is a bit trickier. For example how about `y=x^2+z^2`? The way it works is basically like this: Since it's of the form `y=...` we first generate a bunch of `(x,z)`, then figure out the `y`, then plot:

```
clear all;
syms x z;
[x,z]=meshgrid(-5:0.1:5,-5:0.1:5);
y=x.^2+z.^2;
surf(x,y,z);
view([10 10 10])
```



I'll take a moment to explain how this works. First the command `[x,z]=...` generates a bunch of 2-D points (`x` and `z` values) with coordinates ranging from `-5` to `5` in steps of `0.1`. We then assign the `y` values. Note that we must use `.^` instead of `^` and similarly we should use `.*` instead of `*` and `./` instead of `/`. The `surf` command itself plots all the 3-D points. I could elaborate further but it's not worth the bother now, it's easier to just plug in the stuff you need. For example here's `x=4-y^2` with `z` anything:

```
clear all;
syms y z;
[y,z]=meshgrid(-5:0.1:5,-5:0.1:5);
x=4-y.^2;
surf(x,y,z);
view([10 10 10])
```



Partial Derivatives

Since we can tell Matlab which variable to take the derivative with respect to, finding partial derivatives is as easy as regular derivatives. Here's an example of a partial derivative with respect to `y`:

```
syms x y;
diff(x^2*y+y^2/x,y)
```

```
ans =

(2*y)/x + x^2
```

Here's an example of a second partial derivative: First finding the derivative with respect to `y` and then `x`:

```
diff(diff(x^2*exp(x*y^2),y),x)
```

```
ans =

2*x^3*y^3*exp(x*y^2) + 6*x^2*y*exp(x*y^2)
```

Finding Directional Derivatives and the Gradient

Matlab does have a `gradient` command but it gives numerical approximations, not what we want. Instead we want a specific manifestation of the Matlab `jacobian` command. Specifically for a function of two variables we do the following. What this really does is take a bunch of derivatives with respect to the letters given and product the output we desire:

```
jacobian(x^2*y^3,[x y])
```

```
ans =

[ 2*x*y^3, 3*x^2*y^2]
```

Here's the same gradient with values plugged in for `x` and `y`. Note the use of the familiar `subs` command but with `{x,y}` to substitute for both values:

```
subs(jacobian(x^2*y^3,[x y]),{x,y},{-1,3})
```

```
ans =

[ -54, 27]
```

Here's one with three variables:

```
syms z;
jacobian(x*exp(x*y)/z,[x y z])
```

```
ans =

[ exp(x*y)/z + (x*y*exp(x*y))/z, (x^2*exp(x*y))/z, -(x*exp(x*y))/z^2]
```

We learned in class that the directional derivative is the dot product of the unit vector with the gradient. Consider the following compound command. Really think about what it all does:

```
a=[1 2];
dot((a/norm(a)),subs(jacobian(x*y^2-1/y,[x y]),{x,y},{-2,3}))
```

```
ans =

-(133*5^(1/2))/45
```

You (hopefully!) guessed it! This is the directional derivative of `f(x,y)=xy^2-1/y` in the direction of `a=1i+2j` at the point `(-2,3)`. We make the vector `a` a unit vector and dot it with the gradient.

Finding Critical Points - Preliminaries

We know how to solve a single equation equal to zero in Matlab:

```
solve(x^2+2*x)
```

```
ans =

  0
 -2
```

We can solve a system of equations by simply stuffing the two equations into a vector. Since the output is a vector we need to assign it to make it legible. If that confuses you don't worry, just do it. Here is how to simultaneously solve `xy-3=0` and `x-y-2=0`:

```
[xsoln,ysoln]=solve([x*y-3 x-y-2])
```

```
xsoln =

  3
 -1


ysoln =

  1
 -3
```

Be aware that the first vector `xsoln` is giving you the two possible `x` values. These pair up with the two possible `y` values. Thus in friendlier terms the two solutions are `(3,1)` and `(-1,-3)`.

Also note that the `solve` command returns the solution in alphabetical order, meaning it returns `x` and then `y`. This is why we assign the solution to `[xsoln,ysoln]=`.

Finding Critical Points

Now then when we look for critical points we're setting both partial derivatives equal to `0`. This is the same as setting the gradient equal to the `0` vector. Here's example 4 from page 878 in the book:

```
f=x^2-2*x*y+1/3*y^3-3*y;
[xsoln,ysoln]=solve(jacobian(f,[x y]))
```

```
xsoln =

  3
 -1


ysoln =
```

```
     3
    -1
```

Thus the critical points are `(3,3)` and `(-1,-1)`. Wow, that was easy!

Solving Lagrange Multiplier Problems

When we solve a problem using Lagrange multipliers what we're doing is solving `grad f = L*grad g` along with the constraint. The first of these is the same as solving `grad f - L*grad g=0` and the constraint can be rewritten to equal `0` too. The tricky thing is solving them all at once. This is how it's done for finding the extreme values for `f(x,y)=3x^2+2y^2-4y+1` subject to `x^2+y^2=16`. Well first here we'll just solve the equations:

```
clear all;
syms x y L;
f=3*x^2+2*y^2-4*y+1;
g=x^2+y^2-16;
firstpart=jacobian(f,[x y])-L*jacobian(g,[x y]);
[Lsoln,xsoln,ysoln]=solve(firstpart,g)
```

```
Lsoln =

 3/2
 5/2
   3
   3


xsoln =

         0
         0
  2*3^(1/2)
 -2*3^(1/2)


ysoln =

  4
 -4
 -2
 -2
```

Again note that the values group together. For example `L=3/2` corresponds to the point `(0,4)` and so on through all four. Also again note the order `[Lsoln,xsoln,ysoln]` because `solve` returns the solutions to `L`, `x` and `y` in alphabetical order.

Even better we can plug `xsoln` and `ysoln` into `f` at the end:

```
subs(f,{x,y},{xsoln,ysoln})
```

```
ans =

    17
    49
    53
    53
```

and simply pick off the largest and smallest. In this case the maximum is `53` occuring at both `(2*3^(1/2),-2)` and `(-2*3^(1/2),-2)` and the minimum is `17` occuring at `(0,4)`.

The end.

Published with MATLAB® R2013b