MIT OpenCourseWare
http://ocw.mit.edu

Introduction to MATLAB
IAP 2008

Introduction to Matlab—Ideas for IAP Project

January 2008

This is not a well-defined list of projects; they are mostly meant as starting points and ideas. You can take any of these projects and make of it as much or as little as you feel like. However, you came to learn a new skill, and practice makes perfect!

1 Queens on a chess board

The first part was already a homework problem: Find a set-up of 8 queens on a chess-board so that no queen threatens another.

Now we can extend this a little:

- How does your solution "scale"? That is, if I ask you to solve n queens on a $n \times n$ board, will it be possible to do with your current solution or will you need to write a new one? (consider writing one for which this is very easy). How long does your program take as a function of n? Is it linear $t \propto n$? Polinomial, $t \propto n^\alpha$ (if so, what is α?), or exponential, $t = e^{an}$ (if so, what is a?)? These are important questions to ask when you are coding a program that needs to work on "large" data. Can you think of ways to make your code faster? Use less memory?

- Find how many solutions (arrangements of queens that is) are there? You will probably have to change you code to find this out. Can you also count without including rotations and mirror symmetries?

- How many placements of n queens on a $m \times m$ board are there? Can you find a nice formula? how about a few of the first examples? How about when $m = n$?

- What about other pieces? Is it really that easy for knights as was discussed on the newsgroup (that 32 is the largest possible number) or is there a way to put 33? More? How many (truly different) ways are there to put 32 knights on the board?

2 Make your own Autostereogram

Ever seen these 3D posters that work when you stare at them for a while? Did you ever care to know how they work or wanted to make one yourself? Here's your chance!

First, here is the theory behind autostereograms (very non-scientific). Our mind figures out the depth of objects in the world by comparing the angles at which our eyes see the same object through our eyes. We can fool this mechanism by creating a pattern that is periodic in the horizontal direction, and then looking at it as if it is closer or further away than it actually is. By staring or looking cross-eyed at the pattern, we make each eye focus on a different copy of the repetitive pattern. However, since the patterns is repetitive the brain thinks that it is looking at object that is located at a different depth. To create an illusion of a 3D object we deviate from the repetitive pattern in the following precise fashion: Each dot in a copy is moved horizonally with respect to the matching dot (in the repetitive pattern). The amount in which the dot is moved is proportional to the desired depth of the illusion at the corresponding point.

In terms of coding you will need to figure out how to:

- Create a random pattern of dots (with about 50% fill) in a narrow vertical rectangle

- Create a vertical strip of the autostereogram, copy the previous dot pattern and move the dots in proportion to the desired depth at that point. Remember that you need to keep coping the previous strip and not the original one

- Plot your points to create the stereogram. It is useful to plot two additional large points below your autostereogram to indicate the spacing of repetition. This will help when trying to view the autostereogram

Your input could be a matrix, a picture, or a mathematical formula. This doesn't mean that your program should accept all of these as input, but rather that you consider the possibilities and think about converting the input into a map, $(x, y) \rightarrow z$.

Try your program first on a simple input that you know more or less what the answer should look like, for example a simple square that is lifted from the background. Then try it on other simple geometric objects, like pyramids and spheres.

Have a 3D picture in mind and create it.

Print the result and post it on your "fridge".

Extra things to think about if you want:

- If your height function has a large jump, you will find that a big area with no points is created and this will hurt the quality of the result. to avoid this you can try to locate big empty spaces at each repetition and fill it up with random points. These extra points should be copied and moved with the others.

- Big jumps in the height function can also cause points to "overlap" creating dense "blobs" of points. This, too can hurt the quality of the autostereogram. Removing points that got overlapped (not the points doing the overlapping) is another way to imporove your autostereogram.

3 Spline your name

Write your name[1] in cursive on graph paper (you'll need to write it rather large). For each letter in your name select 4–6 points in the letter that best "define" the letter (letters with extra strokes, t, i, j and x, require extra care). Write down, in order, the x and y coordinates of each of the points.

Using a *spline* interpolation, reconstruct your name from the measured coordinates. Create a plot of the reconstruction. Try to do as much as possible on your own. You'll need to do the following (each can be done with some internal matlab command):

- Find the interpolating spline ploynomials

- Evaluate the polynomials for other points

- Plot the resulting (x, y) curve

Remember that such a curve is rarely a function $y(x)$ but rather each of the coordinates is a function of a third "hidden" parameter, usually marked t. It is with respect to this paramter that $x(t)$ and $y(t)$ are found, but then $y(t)$ should be plotted versus $x(t)$. The more of this you do "on your own" the better. If you don't know the equations for the spline polynomials, you can probably find it online.

4 Random motion (or drunken walk)

Compute and plot the paths of a set of many (about 1000) random walkers which are confined by a pair of barriers at $+B$ and $-B$ (we assume that they all start at $x = 0$

A random walk is computed by repeatedly performing the calculation

$$x_{j+1} = x_j + s \tag{1}$$

where s is a number drawn from the standard normal distribution (randn in MATLAB). For example, *one walker* taking N steps could be handled by the code fragment

```
x(1) = 0;
for j = 1:N
    x(j+1) = x(j) + randn(1,1);
end
```

There are three ways in which the barriers can "act":

a. Reflecting - In this case, when the new position is outside the walls, the walker is "bounced" back by the amount that it exceeded the barrier. That is, when $x_{j+1} > B$, $x_{j+1} = B - |B - x_{j+1}|$ when $x_{j+1} < (-B)$, $x_{j+1} = (-B) + |(-B) - x_{j+1}|$. If you plot the paths, you should not see any positions that are beyond $|B|$ units from the origin.

---

[1]If you have a name for which the following is annoying or for any other reason, you may use a different name or a drawing you make with one long smooth stroke

b. Absorbing - In this case, if a walker hits or exceeds the wall positions, it is absorbed and the walk ends. For this case, it can be of interest to determine the mean lifetime of a walker (i.e., the mean and distribution of the number of steps the "average" walker will take before being absorbed).

c. Partially absorbing - This case is a combination of the previous two cases. When a walker encounters a wall, "a coin is flipped" to see if the walker relfects or is absorbed. Assuming a probability $p, (0 < p < 1)$ for reflection, the pseudo-code fragment that follows uses the MATLAB uniform random-number generator to make the reflect/absorb decision:

```
if rand < p
    reflect
else
    absorb
end
```

(of course, you will have to supply the code that does the reflection or absorbtion)

What do you do with all the walks that you generate? Compute statistics, of course. Answering questions like:

- What is the average position of the walkers as a function of time?

- What is the standard deviation of the position of the walkers as a function of time?

- Does the absorbing or reflecting character influence these summaries?

- For the absorbing/partial-reflection case, a plot of the number of surviving walkers as a function of step number/time is a very interesting thing. It is useful, informative and interesting, particularly if graphically displayed. What about the "density" of walkers? Can you plot it?

5   Chaos: $x_n + 1 = rx(x - 1)$

You already found this simple formula in a homework assigment. Now's the time to take this homework assigment to the next level.

- Find how to make the plot have small dots without plotting each data-point with a single `plot` command.

- For each r the iterates converge on a finite set of values, x. Don't plot more points than you need. Make sure you iterate enough for convergance (the original value of 1000 was only a rule of thumb).

- Can you allow the user to "zoom in" on your plot? once asked to see a region smaller than $(0, 4) \times (0, 1)$ you should probably increase the "density" of your r measurements, and confine the plotting of the points so that only the requested $x's$ are plotted.

- Can you find out how to redraw the figure whenever the zoom tool is used?