PH 120 Project # 1: Smorgasbord

Due: Friday, January 9, 2004

0. Warm-up. Try out some simple calculations in Mathematica (e.g. start with something like $3+4$). Remember that you have to press Shift-Enter to evaluate a cell.

   Ordinarily, text you type into a Mathematica notebook cell is interpreted as input to be evaluated. But you can also insert text for comments and explanations. What you do is click the little bracket on the right-hand side corresponding to the cell you would like to be text, and then under the `Format` menu, choose `Style` and select `Text`. (You should see the little bracket change shape slightly.) Then this cell will not be evaluated and can contain any (formatted) text you want. Use this feature to put your name and a title on your notebook for this assignment. Be sure to use this feature liberally throughout the course to insert comments and explanations!

1. Some calculus. Consider the function $f(x) = e^{-x^2}$. Use Mathematica to:

   (a) Create a definition `f[x_]` for this function.

   (b) Plot $f(x)$ for a reasonable range of x (include both positive and negative values), labeling the axes.

   (c) Define a new function `fderiv[x_]` to be the derivative of $f(x)$ (have Mathematica compute the derivative, don't do it yourself).

   (d) Plot the derivative of f as a function of x, labeling the axes.

   Hint: there are a couple of ways to do this, but if you find you are getting messages like

   ```
   General::ivar:  "-3.32153 is not a valid variable."
   ```

   what is probably happening is the following: What `Plot` does it to repeatedly substitute numerical values for `x` and then evaluate the result numerically. The problem is that if this substitution takes place *before* you take the derivative, then it ends up trying to take the derivative with respect to a numerical value like `-3.2153` rather than the variable `x`. The solution is to make sure the derivative gets evaluated *before* `Plot` can get its hands on it, which you can do by using `=` instead of `:=` in the definition of the function. (This distinction is probably the single most confusing aspect of Mathematica programming.)

(e) Find $\int_{-\infty}^{\infty} f(x)\,dx$.

(f) Define a new function `fint[x_]` to be an indefinite integral (antiderivative) $\int f(x)\,dx$. (The result will probably not be a function you are familiar with; that's OK.) Plot this result as a function of x, labeling the axes.

2. Euler's constant. In this problem we would like to compare

$$\sum_{i=1}^{n} \frac{1}{i} \qquad \text{and} \qquad \int_1^n \frac{1}{x}\,dx$$

as functions of n.

(a) Start by plotting `1/x` and `1/Floor[x]` on the same graph. Explain what this has to do with the comparison above. Which one is bigger?

Hint: Think about areas.

(b) Define `diff[n_]` to be the difference between the sum and the integral as a function of the upper limit n. Plot this function and see what happens as $n \to \infty$. The result is a fundamental mathematical constant (like π and e) called Euler's constant. In Mathematica, it is called `EulerGamma`. Check your result by using `N` to evaluate Euler's constant.

(c) Repeat this same analysis, but now compare

$$\sum_{i=1}^{n} \frac{1}{x+1} \qquad \text{and} \qquad \int_1^n \frac{1}{x}\,dx$$

Find how your result in this case is related to EulerGamma, and explain this result graphically.

3. Solving polynomial equations. Suppose we have a mass m_1 at $x = -a$, another mass m_2 at $x = a$, and a third mass m sits on the x-axis in between. Recall that the potential energy of the third mass is given as a function of its position x by $v(x) = -Gm\left(\frac{m_1}{a+x} + \frac{m_2}{a-x}\right)$. (If we replace the masses with charges, this could also be a model of the energy of an electron in a diatomic molecule.)

(a) Use `Solve` to find the extrema (possible minima or maximia) of `v[x]`. Note that what `Solve` returns is a list of replacement rules, corresponding to each possible solution. One of these will be spurious (because it is outside the region where our formula is valid), so use `Part` (which is the same as double angle brackets) to select the solution that is between $-a$ and $+a$. (It might help to use `Simplify` here.)

(b) Use the substitution operator `/.` to substitute the x value of the extremum into the second derivative $v''(x)$ to determine whether the extremum is a minimum or maximum (it might help to apply `FullSimplify`). Also find the value of $v(x)$ at the extremum.

4. In this problem you will study a simple example of a *fractal.* Suppose you have an equation like $\cos x = x$ that you would like to solve numerically. Mathematica's `FindRoot` function uses *Newton's method* to search for an approximate solution given a starting point.

   What we would like to do is to take a simple equation, which we know how to solve, but has more than one solution. Our goal is to describe which solution Newton's method finds as a function of the starting point. The equation we will consider is
$$z^3 = 1$$
   where we allow the variable z to be complex.

   (a) First, use Mathematica's `Solve` function to find all three solutions to this equation. As practice for what you will do next, define a function `sol[j_]` that takes in an integer `j` from 1 to 3 and gives back the `j`th solution to this equation. Note that doing so takes two steps: you have to select out the `j`th solution and then apply the rule to `z` to get the result. So `sol[1]` should return `1`, *not* `z -> 1`.

   (b) Use `FindRoot`, picking any starting point you want (other than one of the roots!), and see that it finds one of the answers returned by `Solve`.

   (c) Define a function `startToRoot[z_]` whose input is a complex number representing the starting value for `FindRoot`, and whose output is the solution that `FindRoot` converges to from this starting value. Evaluate your function for a few inputs to verify that it always returns one of the solutions you found in part (a). Note that `FindRoot` returns a single replacement rule, like {`z -> 1.`} (which is similar to `Solve`, except that `Solve` returned a list of such rules). So you will need to create a function `startToRoot[z_]` that calls `FindRoot` to get a replacement rule, and then applies this rule to substitue the answer into `z`.

   (d) Use Mathematica's `DensityPlot` function to generate a picture of which solution `FindRoot` converges to as a function of the starting point $x + iy$. What we would like to see is a color-coded map where the color at a point (x, y) corresponds to which of the three solutions `FindRoot` returns if you tell it to start at $z = x + iy$. The function you are plotting should take on one of three different *real* values, each corresponding to one of the three different solutions that `startToRoot` can produce. But these solutions are *complex*, so you will need to transform them into three distinct real values.

(Will taking the magnitude of the complex number work? How about taking the argument?)

There are several options you should use for `DensityPlot`:

- Set `ColorFunction -> Hue` because this plot will look a lot cooler in color. Because of a bug in Version 5, in that version you also need to set `ColorFunctionScaling -> False`.
- Use the `PlotPoints` option to increase the number of points Mathematica uses to generate the plot — the default is much too small to see the patterns.
- Set `Mesh -> False` because if you use a lot of plot points, Mathematica will idiotically cover the entire plot with gridlines, so that you can't see the plot itself.

5. The *Fiboncacci sequence* is determined by the following rule: The first term is $F_1 = 1$. The second term is $F_2 = 1$. Subsequent terms are give by the sum of the previous two terms: $F_j = F_{j-1} + F_{j-2}$ for $j > 2$.

   (a) Create a Mathematica function `fib[j_]` that returns the j^{th} term in the Fibonacci sequence. Of course, you are not allowed to use Mathematica's built-in Fibonacci function! (Although you may use it to check your answer.)

   Hint: you can either use Mathematica's `If` statement or just use different definitions for the special cases and the general case.

   Hint: throughout this problem, you will use recursion — a function calling itself repeatedly with a different argument. This approach is rather inefficient (which is easily remedied), but don't worry about that here.

   (b) Use `Table` to create a list of the first 20 or so Fibonacci numbers, and then use `ListPlot` to plot them as a function of j. At large j, you should find the ratio F_{j+1}/F_j of each term to the previous term approaches a finite limit ϕ. Use the techniques you have learned to find ϕ numerically.

   (c) Because of what you found in the previous problem, at large N we can approximate $F_{N+1} = F_N\phi$ (and thus $F_{N+2} = F_{N+1}\phi$). Use this fact to find a formula for ϕ and compare it to your numerical result. ϕ is called the *Golden Ratio.*

   (d) Show numerically that $F_n = \frac{1}{\sqrt{5}}\left(\phi^n - (1-\phi)^n\right)$. Extend your function `fib[j]` such that if j is a positive integer it still uses the recursive definition you implemented already, but for any other value of j it uses this formula.

   (e) We can define a family of Fibonacci sequences by changing the values of F_1 and F_2 to be something different from 1 (but the rule for the rest of the terms is the same). Use Mathematica's `Function` feature to define a function `makefib[f1_,f2_]` that, for given values of F_1 and F_2, returns a *function* to compute the Fibonacci sequence with these initial values. In other words, `makefib[f1,f2][j]` should be the `j`th term in the Fibonacci sequence whose first two terms are `f1` and `f2`. This is tricky! (You only need to implement the ordinary definition, not the extension to nonintegers.)

Hint: Here you probably want to use an `If` statement rather than separate definitions for the special cases.

(f) Pick a values for F_1 and F_2 and use your function to plot this Fibonacci sequence. Find the ratio of successive terms at large N.

(g) Because of its use of recursion, this approach is rather inefficient. Here is an easy way to speed it up. Go back to your original definition of `fib[j]` and replace it by the following:

```
fib[j_] := (fib[j] = ....)
```

where you put your previous definition in where I've written "`...`". You should find that this change makes `fib[j]` run much faster. Explain how it works.

Hint: You might find the output of evaluating `?fib` helpful. To see what `?` does, see Section 1.3.9 of the Mathematica Book (available within the help system).