Autumn Quarter 2005
Math. Methods Problem Set 3: Newton's Method and Numerical Integration

November 9, 2005

1 Newton's method

Use Newton's method to find the first 5 positive solutions of $\sin x = 10e^{-x}$.

Solution: See Python script. The answers are 6.30152,9.42397,12.56641,15.70796 and 18.84956

2 Newton's method as a dynamical system

Consider the iterated-map dynamical system defined by applying Newton's method to an attempt to find the roots of $x^2 + 1 = 0$, with real x. Are there fixed points? Are there periodic orbits? Are these stable? What is the attractor basin of the stable solutions, if any? Are there any chaotic orbits?

Solution:

You could spend a lifetime, or at least a PhD thesis, exploring the behavior of this innocuous map. I will only sketch out a little of its basic behavior.

The Newton's method iteration for this problem is

$$x_{n+1} = x_n - \frac{x^2+1}{2x} = \frac{1}{2}x_n - \frac{1}{2x_n} \tag{1}$$

Thus, the iterated map is

$$g(x) = \frac{1}{2}x - \frac{1}{2x} \tag{2}$$

This has no fixed points for real x, because the original function has no zeros on the real axis.

We note a few important properties of this function:

- $g(-x) = -g(x)$
- When $|x|$ is large, $g(x) \approx \frac{1}{2}x$
- $g(x) < 0$ for $0 < x < 1$, and $g(x) > 0$ for $x > 1$.
- $g(x) \to -\infty$ when x approaches zero from the positive side, and $g(x) \to +\infty$ when x approaches zero from the negative side.

the first property means that we only need to look at the behavior of the function and its compositions for positive values, since the behavior for negative x can be inferred from its behavior at positive x. $g(g(x))$ is an even function, $g(g(g(x)))$ is an odd function, and so forth. The second property means that if $|x|$ is initially large, or if an orbit takes $|x|$ to a large value at any point, then the orbit will relax exponentially to smaller values like $|x|/2^n$, until the large $|x|$ form of g is no longer valid. The third property means that values in the interval $[0, 1]$ map to negative values on the next iterate, whereas values $x > 1$ stay positive at the next iterate. Since large values always decay eventually to smaller values (from the second property), the orbits will not be confined to the positive x axis, but instead will switch sides from time to time. Values $x < -1$ stay negative until they fall into the interval $[-1, 0]$, at which point they flip to the positive side.

The last of the listed properties is a real headache, since the occurrence of singularities complicates the problem of finding periodic orbits on the computer. It is also, however, the central feature that organizes the behavior of the orbits of this system. The function has a discontinuity at $x = 0$ so searching for fixed points by looking for sign changes in $g(x) - x$ will give spurious results near this point. A similar remark applies to looking for periodic orbits, which we do by looking for fixed points of compositions of g with itself; points where the compositions become infinite also lead to spurious fixed points if we look for changes in sign. At what points do the higher compositions become singular? To determine this, we look at the points which map to zero after n iterates, since the next iterate will then be infinite. These points are the *pre-images of zero*. For example the point $x = 1$ maps to $x = 0$ after the first iterate, and so $x = 1$ will also yield infinite values

at the second iterate (which would show up on the composition-2 map). We calculate the pre-images by inverting the map in Eq. 2 to solve for $x(g)$. This is done by solving $x^2 - 2gx - 1 = 0$. The solution is

$$x = g \pm \sqrt{g^2 + 1} \tag{3}$$

This is not single valued. The number of pre-images doubles for each iterate, if there are no repeats. See Python script for calculations. Note that the list of singular points of the $(n+1)^{st}$ composition includes all the singular points of the n^{th} composition as well. For example, if $g(g(x_1))$ is infinite then $g(g(g(x_1)))$ is also infinite. Here are some results for the additional singular points that are added for the first few numbers of iterates:

iterate	New singular points
2	1.0
3	0.41421356237309515, 2.4142135623730949
4	0.098491403357164664, 0.3033466836073424, 0.53451113595079169
4(cont'd)	0.82067879082866013, 1.2185035255879766, 1.8708684117893895,
4(cont'd)	3.296558208938321, 10.15317038760886

Only the positive ones are listed, since the property $g(x) = -g(-x)$ assures that the negative ones are just the negatives of the positive ones. The pre-images of zero after n iterates are the singular points of the $n+1$ times composition of the map. Thus, from the table above, the singular points of the two-times composition $g(g(x))$ are 0 and 1, while the singular points of the three-times composition (used in finding the period three orbits) are 0,1,.41... and 2.41....

Now we use `mapExplorer` to look for periodic orbits. To avoid division by zero problems, and also spurious fixed points cause by the singularities, we must search separately within each subinterval bounded by the singular points of the composition. If we use `plotComposition` in the interval $[.1, .9]$ we find that there is an unstable periodic orbit near $x = .6$. Using `findPeriodicOrbit` we find that the more exact value is .5769769769. The script shows the behavior of an orbit started near this point. The orbit initially oscillates between positive and negative values, and as the instability grows, the oscillation breaks down. An examination of the composition for $x > 1$ reveals no periodic orbits there.

Using a similar procedure in the script, we find the positive "seeds" of a number of unstable period-3 orbits. These are at $x = .228, .797, 2.07$

As an example, we also find an unstable period-4 orbit starting from $x = .105$

Since all the periodic orbits are unstable, they have no attractor basin

Is the map chaotic? We look for the three symptoms of chaos. First we check for sensitive dependence on initial conditions in the script. This is verified. The lyapunov exponent is approximately .698. Next we look for a dense set of unstable periodic orbits. This is tricky for this map, since the singularities prevent us from using the `orbitDiagram` function to make a plot of periodic orbits in the straightforward way. From the experimentation above, it looks like there is an unstable periodic orbit between each pair of preimages of zero, and that these get dense near the unit interval as the number of iterates is made large. This is conjectural and ought to be verified more systematically. The final check for chaos is to see if the set of points visited by the orbit is "ergodic" in some subregion of the domain, i.e. whether there is some collection of subintervals for which the orbit eventually comes arbitrarily near any point. You can examine this aspect best by making a histogram of the points in a long orbit, which we do in the script. The orbit certainly seems to come near every point in the unit interval, as opposed to being confined to a finite set of points as a periodic orbit would be.

Certainly, we haven't proved that this system is chaotic. However, it certainly barks like a dog.

An interesting characteristic of this map is that for "most" initial conditions (i.e. for initial conditions other than the pre-images of zero) the orbit doesn't run away to infinity. Nonetheless, the orbit is not bounded. The longer you iterate, the higher is the maximum value you encounter. Once the orbit is flung out to large values, it then relaxes exponentially back towards the unit interval, where it rattles around some more until it is flung out again.

3 Numerical evaluation of definite integrals

Study the convergence of the trapezoidal rule approximation of

$$\int_a^b \sqrt{x} dx \tag{4}$$

to the exact value for $n = 2, 4, 8, 16, ...4096$. Save your results in a list for plotting and other future uses. Is it quadratic in $1/n$ as expected? Now look at how rapidly the result converges if you use Romberg extrapolation (helped

by the polynomial interpolation/extrapolation routine `polint`) for the same sequence of n.

First do the above for $a = 1, b = 10$. Then try again for $a = 0$ and compare results. Does Romberg extrapolation improve your convergence as much in this case? What is the reason for the difference between the two cases?

Solution: The exact integral is $\frac{2}{3}(b^{3/2} - a^{3/2})$. For the answer to the rest of this problem, see the accompanying Python script.

Finally,just for fun, use the trapezoidal rule with Romberg extrapolation to evaluate the integral

$$\int_0^\pi e^{-\cos x} dx \tag{5}$$

If you begin with a single interval ($n = 1$), how many refinement steps do you need to achieve 6 decimal place accuracy?

Solution: The answer is 3.97746326051. You get 6 decimal place accuracy with only 5 refinement steps (i.e. a modest 2^5, or 32 subintervals). See the Python script for details of the calculation. This uses the improved version of Romberg extrapolation, but the "obvious" version is not much worse.

What's a little disappointing, and very surprising, is that the ordinary trapezoidal rule with 32 subintervals gives essentially the same accuracy as the Romberg method, for this function. `dumbTrap(f,[0.,math.pi],32)` gives 3.977463260506422, which is the same as the Romberg result to 6 decimal places. This is much better accuracy than we should expect from the trapezoidal rule, since $dx = .098$ and $dx^2 = .0096$, so we ought to have an error on the order of 1%. I am baffled and dumbfounded by the unexpected performance of the trapezoidal rule for this function. Any ideas about what is going on here would be very welcome. Can we say anything useful about the class of functions for which the trapezoidal rule works this well? Students are encouraged to experiment with other smooth functions, to see what they can come up with.

4 Programming Project: A better trapezoidal rule integrator

The simplest form of the trapezoidal rule integrator, `dumbTrap(f,interval,n)` is wasteful if you've already computed the result for `n = N` and then want

to recompute it for `n = 2N`. The recomputation computes the sum over all the points, whereas half the work was already done when you computed the result for `n = N`.

Develop a class called `betterTrap`, which has `dumbTrap` as one of its methods, and has the function being integrated, the interval of integration, and the current values of `n` and `dumbTrap(f,interval,n)` as members. Include a method `refine(...)`, which computes the trapezoidal rule approximation for `2n` in terms of the previous value, without throwing away the part you've already computed. The `refine(...)` method should update the value of `n` and of the estimate.

Note: The `__init__` method should take the initial value of `n`, the interval one is integrating over, and the function `f` as its argument, and compute the initial value of the trapezoidal rule approximation.

Solution: This class is defined in the accompanying Python script. In that script, we actually go a bit further and implement a class which does quadrature by Romberg extrapolation.