Problem Set 5: Eigenvalues and Fourier Transforms
Computational Physics
Physics 480/680
James Sethna; Due Monday, April 7
Last correction at March 17, 2014, 8:23 pm

Reading
Numerical Recipes chapters 11, 12, and 13, skimming the technical bits
Numerical Recipes section 20.7, skimming the technical bits
Fourier methods appendix, from *Entropy, Order Parameters, and Complexity*, http://pages.physics.cornell.edu/sethna/StatMech/

This homework may be more challenging than most – especially for those who haven't used Fourier methods a lot in their other classes. The FFT is one of the most useful numerical algorithms known; I don't apologize for giving you a thorough introduction in how to use it.

Exercise 5.3 part (e) is optional for those in Physics 4480.

5.1 **Fourier Series, Exponential Sine.** (Computation) ②

In this exercise, we'll examine the Fourier series of the function

$$f(t) = \exp(-6\sin(t)), \tag{1}$$

with period $T = 2\pi$.

Periodic functions with period T can be expanded in a Fourier series,

$$f(t) = \sum_{m=-\infty}^{\infty} \widetilde{f}(\omega_m) \exp(-\mathrm{i}\omega_m t) \tag{2}$$

where

$$\widetilde{f}(\omega_m) = \frac{1}{T}\int_0^T f(t)\exp(\mathrm{i}\omega_m t)\mathrm{d}t \tag{3}$$

and $\omega_m = 2\pi m/T$ with integer m. (Sometimes the Fourier series is written $\widetilde{f}_m$, omitting the explicit frequency dependence.)

The FFT takes a series of N points y_ℓ and returns

$$\widetilde{y}_m^{\mathrm{FFT}} = \sum_{\ell=0}^{N-1} y_\ell \exp(\mathrm{i}2\pi m\ell/N). \tag{4}$$

(Note: Often the FFT is divided by $1/\sqrt{N}$.) Here m is the index of the complex array passed back by the FFT routine, running from $0, \ldots, N-1$. The (discrete) N-term FFT

does a good job of estimating the (continuous) Fourier series if the Fourier coefficients higher than $n = N/2$ are small.

(a) *Take the FFT of the series of points* $y_\ell = f(2\pi\ell/256)$ *for* $\ell = 0 \ldots 255$*, with* $f(t) = \exp(-6\sin(t))$ *from eqn (1). (Don't program eqn 4 yourself! Use the FFT package provided by your software environment.) After how many terms* m_{max} *are the coefficients smaller than machine precision? Make a semi-log plot of the absolute values* $|\widetilde{y}_m^{\rm FFT}|$ *as a function of* m*. Note that many are below the machine precision. Are they converging exponentially (as we expect for analytic functions), or even faster than exponentially? Why are the terms* $m = 255, 254, 253, \ldots$ *large again?*

To convert from the FFT to the Fourier series, one must carefully figure out the appropriate conversion factors. Notice that the FFT doesn't know the period T, so one must convert the sum to an integral. Remember also that the FFT may include a factor of $1/\sqrt{N}$.

Converting FFTs to Fourier series. Write a general routine that converts the FFT into the Fourier series $\widetilde{y}(\omega_m)$ (eqn 2). It should be given both the time spacing Δ and an array $y(t_i)$ for $t_i = 0, \Delta, 2\Delta, \ldots, (N-1)\Delta$. It should return *(i)* the array of frequencies ω properly centered centered at $\omega = 0$, *(ii)* the Fourier series $\widetilde{y}(\omega)$ calculated from the FFT but scaled correctly and centered at zero, and (for convenience) *(iii)* the aliasing frequency $\omega_0 = 2\pi/\Delta$. Again, to do this you'll need to rescale the values $\widetilde{y}_m$ and rearrange and rescale the integers $0 \le m < N$ into the frequencies $-\omega_0/2 < \omega_m \le \omega_0/2$.

(b) *Now, using your new routine on your data for* $f(t) = \exp(-6\sin(t))$ *from part (a), make a semi-log plot of the estimated Fourier series* $\widetilde{f}(\omega)$ *versus* ω*. (Hint: Check your rescaling of the values using the fact that the integral of* $f(t)$ *is the period* T *times* $\widetilde{f}(\omega = 0)$*. Remember your answer* $\int_0^{2\pi} f(t)\mathrm{d}t = 422.446$ *from problem set two.)*

Aliasing. The Fourier series $\widetilde{f}_m$ runs over all integers m. The fast Fourier transform (FFT) runs only over $0 < m < N$. The contributions from higher Fourier components are folded over into the low frequencies. Since the contributions grow small so quickly, we need to do rather small numbers of points to illustrate the problem.

(c) *Use your routine to plot the Fourier series taking* $N = 4$*, 8, and 16 equally-spaced points* $f(t_i)$ *between zero and* 2π*.* (Suggestion: I drew them with symbols, but connected by dotted lines, to make it easier to visualize. I also used ω_0 to label the curves in the legend.) *Note the peak at* $\omega = 1$*, and how it converges to its true value. The change in this peak when* N *changes from 4 to 8 should be noticable. What peak for* $N = 8$ *got 'eaten' by the* $\omega = 1$ *peak at* $N = 4$*? What peak got 'eaten' going from* $N = 16$ *to* $N = 8$*?* (Hint: Remember how the error in the FFT involves[1] ω_0.)

Windowing. What happens if you get the period wrong? The Fourier series of a periodic function develops 'sidebands' if it is sampled over a time not equal to a multiple of its period.

[1]Don't be confused by the fact that $\omega_0 = N$. $f(t)$ in eqn 1 is a special case whose period is $T = 2\pi$, and so $\omega_0 = 2\pi/\Delta = 2\pi/(T/N) = N$.

(d) *Sample $f(t)$ from eqn (1) at $N = 512$ equally-spaced points spanning $32 + f$ periods of the function (i.e., the time points are separated by $\Delta = (2\pi)(32 + f)/512$). Using your routine, plot the Fourier series (both real and imaginary part) for $f = 0$. Why are the Fourier series components now separated by zeros? Plot the same for $f = 1/3$. What happened to the Fourier series peaks?*

5.2 **Diffusion: Finite Difference vs. Spectral.** (Computation) ③

The one-dimensional diffusion equation

$$\partial u/\partial t = D\partial^2 u/\partial x^2 \tag{5}$$

gives the evolution of a field $u(x, t)$. This field is *conserved*, so $\int u(x, t)\,\mathrm{d}x$ is independent of time. In this exercise, we'll use various techniques to approximate the solution to the diffusion equation with the periodic initial condition

$$u(x, 0) = \exp(-6\sin(x)), \tag{6}$$

with period $L = 2\pi$. We solve this in the interval $[0, 2\pi]$ with periodic boundary conditions, or equivalently in the infinite interval $(-\infty, \infty)$, where we note the solution is also periodic.

Finite Difference Method. Our first approach will be to approximate the continuum differential equation on a spacetime grid, with time increments Δt and spatial increments Δx:

$$(u(x, t + \Delta t) - u(x, t))/\Delta t = D\left(u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x)\right)/(\Delta x)^2. \tag{7}$$

Note that the time step is a forward difference, and the spatial increment is a symmetric second difference. We've derived these before by matching the first terms in the Taylor series for u.

(a) *Solve eqn (7) for $u(x, t+\Delta t)$. Will this approximation conserve u (preserve the sum $\sum u(x+n\Delta x)$), except for rounding errors?* You can interpret the discrete equation (7) as sharing a fraction of $u(x)$ with its two neighbors in each time step. *At what $\Delta t_{\text{overshoot}}$ will that share grow larger than one? How does this overshoot time step change when Δx gets smaller by a factor of two?*

Write a finite-difference routine that implements eqn. (7), with arguments u_0, t_{elapsed}, and $\mathrm{d}t$. It should copy the initial condition $u_0(x)$ (don't overwrite it!), step forward $M = t_{\text{elapsed}}/\Delta t$ time steps, and return the evolved approximation to $u(x, t_{\text{elapsed}})$. (Beware of rounding errors here! Most programming languages will convert the floating point number 31.999... into the integer 32 by default. I used $M = \text{int}(\text{round}(t_{\text{elapsed}}/\Delta t))$.)

(b) *Using $D = 0.8$, $N = 32$, $\Delta x = L/N$, and $\Delta t = \Delta t_{\text{overshoot}}/4$, evolve the initial condition from eqn (6) and plot the result for $t_{\text{elapsed}} = 0.25$, 0.5, 1, 2, 4, and 8. Save the evolved points (both x and u) at $t_{\text{elapsed}} = 8$ for later comparisons. What happens if you use $\Delta t > \Delta t_{\text{overshoot}}$?*

We often would like finer spatial resolution than $N = 32$ points.

(c) *Time how long your routine takes as you repeatedly divide Δx by two (finer interpolations) until it gets tediously slow (a minuite or so). How does the computer time scale with Δx?*

Now let us implement a *spectral method* solution to the diffusion equation.

(d) *Show that the spatial Fourier transform $\widetilde{u_k}(t)$ obeys the diffusion equation*

$$\frac{\partial \widetilde{u}_k}{\partial t} = -Dk^2 \widetilde{u}_k. \tag{8}$$

so

$$\widetilde{u}_k(t) = \widetilde{u}_k(0) \exp(-Dk^2 t). \tag{9}$$

Write a spectral routine implements eqn (9), with arguments u_0 and $t_{\rm elapsed}$. It should compute the FFT of u_0, multiply by $\exp(-Dk^2t)$, and do an inverse FFT (usually called ifft), and return the real part. (Warning: computing the vector of points k^2 is tricky. Remember that the FFT returns values at $k = 2\pi n/L$, but with n growing and then going negative: $n = 0, 1, \ldots, N/2-1, \pm N/2, -N/2+1, \ldots, -2, -1$.) As part of the debugging process, make sure that the inverse FFT has imaginary part near zero.

(e) *Using $D = 0.8$, $N = 4096$, and $\Delta x = L/N$, evolve the initial condition from eqn (6) and plot the result for $t_{\rm elapsed} = 0.25$, 0.5, 1, 2, 4, and 8.* (The answers should roughly agree with those of part (b).) *How long would your finite-difference code have taken to evolve the solution at this resolution? Store the points for $t_{\rm elapsed} = 8$.*

We saw in exercise (1) that the Fourier components of our initial condition rapidly go to zero. There is no reason to solve for $N = 4096$ points: one could solve for many fewer points in Fourier space and then evaluate the Fourier sum as finely as one wishes (speeding things up even further). Let's explore how the solution looks for $N = 32$ points, without writing the routine that interpolates between them.

(f) *Using $D = 0.8$, $N = 32$, and $\Delta x = L/N$, evolve the initial condition from eqn (6) and plot the result for $t_{\rm elapsed} = 8$. Plot these, along with your stored finite-difference results from part (b) and your stored $N = 4096$ spectral points from part e. Zoom in, and estimate the ratio of the errors for the$N = 32$ spectral method compared to that of the $N = 32$ finite-difference method, assuming that the $N = 4096$ solution is exact.*

Solving the diffusion equation for a periodic function here is illustrating much more general and powerful techniques. This basic idea can be generalized to other boundary conditions and to nonlinear equations (see Section 20.7 in NR). Also commonly used are hybrid methods, like operator-splitting methods. These work in nonlinear problems like turbulence when the spatial derivatives terms are linear;[2] you alternate real-space nonlinear term timesteps with Fourier-space gradient term timesteps...

[2]They are also useful for problems where the the term with the highest number of spatial derivatives (which determines the maximum time step) is linear.

5.3 **Sloppy Monomials.**[3] (Eigenvalues, Fitting) ③

We have seen that the same function $f(x)$ can be approximated in many ways. Indeed, the same function can be fit in the same interval by the same type of function in several different ways! For example, in the interval $[0, 1]$, the function $\sin(2\pi x)$ can be approximated (badly) by a fifth-order Taylor expansion, a Chebyshev polynomial, or a least-squares (Legendre[4]) fit:

$$\begin{aligned} f(x) &= \sin(2\pi x) \\ &\approx 0.000 + 6.283x + \ 0.000x^2 - 41.342x^3 \\ &\quad + \ 0.000x^4 + 81.605x^5 \quad \text{Taylor} \\ &\approx 0.007 + 5.652x + \ 9.701x^2 - 95.455x^3 \\ &\quad + 133.48x^4 - 53.39x^5 \quad \text{Chebyshev} \\ &\approx 0.016 + 5.410x + 11.304x^2 - 99.637x^3 \\ &\quad + 138.15x^4 - 55.26x^5 \quad \text{Legendre} \end{aligned}$$

It is not a surprise that the best fit polynomial differs from the Taylor expansion, since the latter is not a good approximation. But it is a surprise that the last two polynomials are so different. The maximum error for Legendre is less than 0.02, and for Chebyshev is less than 0.01, even though the two polynomials differ by

$$\begin{aligned} \text{Chebyshev} - \text{Legendre} = & \\ & - 0.009 + 0.242x - 1.603x^2 \\ & + 4.182x^3 - 4.67x^4 + 1.87x^5 \end{aligned} \tag{10}$$

a polynomial with coefficients two hundred times larger than the maximum difference!

This flexibility in the coefficients of the polynomial expansion is remarkable. We can study it by considering the dependence of the quality of the fit on the parameters. Least-squares (Legendre) fits minimize a cost C^{Leg}, the integral of the squared difference between the polynomial and the function:

$$C^{\text{Leg}} = (1/2) \int_0^1 (f(x) - \sum_{m=0}^{M} a_m x^m)^2 \, \mathrm{d}x. \tag{11}$$

How quickly does this cost increase as we move the parameters a_m away from their best-fit values? Varying any one monomial coefficient will of course make the fit bad. But apparently certain coordinated changes of coefficients don't cost much – for example, the difference between least-squares and Chebyshev fits given in eqn (10).

[3] Thanks to Joshua Waterfall, whose research is described here.

[4] The orthogonal polynomials used for least-squares fits on [-1,1] are the Legendre polynomials, assuming continuous data points. Were we using orthogonal polynomials for this exercise, we would need to shift them for use in [0,1].

How should we explore the dependence in arbitrary directions in parameter space? We can use the eigenvalues of the Hessian to see how sensitive the fit is to moves along the various eigenvectors...

(a) *Note that the first derivative of the cost C^{Leg} is zero at the best fit. Show that the Hessian second derivative of the cost is*

$$H^{\text{Leg}}_{mn} = \frac{\partial^2 C^{\text{Leg}}}{\partial a_m \partial a_n} = \frac{1}{m+n+1}, \tag{12}$$

$$H^{\text{Leg}} = \begin{pmatrix} 1 & 1/2 & 1/3 & \dots & 1/M \\ 1/2 & 1/3 & \dots & & \\ \dots & & & & \\ 1/(M-1) & 1/M & \dots & \dots & 1/(2M-1) \end{pmatrix} \tag{13}$$

This Hessian is the Hilbert matrix,[5] famous for being ill-conditioned (having a huge range of eigenvalues). Tiny eigenvalues of H^{Leg} correspond to directions in polynomial space where the fit doesn't change.

(b) *Calculate the eigenvalues of the* 6×6 *Hessian for fifth-degree polynomial fits. Do they span a large range? How big is the condition number?*

(c) *Calculate the eigenvalues of larger Hilbert matrices. At what size do your eigenvalues seem contaminated by rounding errors? Plot them, in order of decreasing size, on a semi-log plot to illustrate these rounding errors.*

Notice from Eqn 12 that the dependence of the polynomial fit on the monomial coefficients is *independent of the function* $f(x)$ being fitted. We can thus vividly illustrate the sloppiness of polynomial fits by considering fits to the *zero* function $f(x) \equiv 0$. A polynomial given by an eigenvector of the Hilbert matrix with small eigenvalue must stay close to zero everywhere in the range $[0, 1]$. Let's check this.

(d) *Calculate the eigenvector corresponding to the smallest eigenvalue of the* 6×6 *matrix* H^{Leg} *with* $M = 5$*, checking to make sure its norm is one (so the coefficients are of order one). Plot the corresponding fifth-degree polynomial in the range* $[0, 1]$*: does it stay small everywhere in the interval? Plot it in a larger range* $[-1, 2]$ *to contrast its behavior inside and outside the fit interval.*

This turns out to be a fundamental property that is shared with many other multiparameter fitting problems. Many different terms are used to describe this property. The fits are called *ill-conditioned*: the parameters a_n are not well constrained by the data. The *inverse problem* is challenging: one cannot practically extract the parameters from the behavior of the model. Or, as our group describes it, the fit is *sloppy*: only a few directions in parameter space (eigenvectors corresponding to the largest eigenvalues)

[5] Note that our matrix starts with $m = n = 0$, the convention in C, C++, and Python for arrays and matrices, but different from the traditional convention in mathematics where arrays and vectors start at one rather than zero. Thus the usual definition of the Hilbert matrix would be $H_{mn} = 1/(m+n-1)$.

are constrained by the data, and there is a huge space of models (polynomials) varying along sloppy directions that all serve well in describing the data.

At root, the problem with polynomial fits is that all monomials x^n have similar shapes on $[0,1]$: they all start flat near zero and bend upward. Thus they can be traded for one another; the coefficient of x^4 can be lowered without changing the fit if the coefficients of x^3 and x^5 are suitably adjusted to compensate. Indeed, if we change basis from the coefficients a_n of the monomials x^n to the coefficients ℓ_n of the orthogonal (shifted Legendre) polynomials, the situation completely changes. The Legendre polynomials are designed to be different in shape (orthogonal), and hence cannot be traded for one another. Their coefficients ℓ_n are thus well determined by the data, and indeed the Hessian for the cost C^{Leg} in terms of this new basis is the identity matrix.

Numerical Recipes states a couple of times that using equally-spaced points gives ill-conditioned fits. Will the Chebyshev fits, which emphasize the end-points of the interval, give less sloppy coefficients? The Chebyshev polynomial for a function f (in the limit where many terms are kept) minimizes a different cost: the squared difference weighted by the extra factor $1/\sqrt{x(1-x)}$:

$$C^{\text{Cheb}} = \int_0^1 \frac{(f(x) - \sum_{m=0}^{M} c_m x^m)^2}{\sqrt{x(1-x)}}\,\mathrm{d}x. \tag{14}$$

One can show that the Hessian giving the dependence of C^{Cheb} on c_m is

$$H_{mn}^{\text{Cheb}} = \frac{\partial^2 C^{\text{Cheb}}}{\partial c_m \partial c_n} = \frac{2^{1-2(m+n)}\pi(2(m+n)-1)!}{(m+n-1)!(m+n)!}. \tag{15}$$

with $H_{00}^{\text{Cheb}} = \pi$ (doing the integral explicitly, or taking the limit $m, n \to 0$).

(e) *Calculate the eigenvalues of H^{Cheb} for fifth-degree polynomial fits.*

So, sloppiness is not peculiar to least-squares fits; Chebyshev polynomial coefficients are sloppy too.