SloppyMonomialsHintsPython

January 19, 2024

Sloppy monomials

(Sethna)

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 fit:

$$\begin{aligned} f(x) &= \sin(2\pi x) \\ f_{\text{Taylor}} &\approx 0.000 + 6.283x + \ \ 0.000x^2 - 41.342x^3 + \ \ 0.000x^4 + 81.605x^5 \\ f_{\text{Chebyshev}} &\approx 0.0066 + 5.652x + \ \ 9.701x^2 - 95.455x^3 + 133.48x^4 - 53.39x^5 \\ f_{\text{Legendre}} &\approx 0.016 + 5.410x + 11.304x^2 - 99.637x^3 + 138.15x^4 - 55.26x^5 \end{aligned}$$

(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].) 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

$$\begin{aligned} &\text{Chebyshev} - \text{Legendre} = \\ &\quad -0.0094 + 0.242x - 1.603x^2 + 4.182x^3 - 4.67x^4 + 1.87x^5 \end{aligned} \tag{1}$$

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

[ ]:
```
# Sometimes gives interactive new windows
# Must show() after plot, figure() before new plot
# %matplotlib

# Adds static figures to notebook: good for printing
%matplotlib inline

# Interactive windows inside notebook: need to stop before new plot
# %matplotlib notebook

from numpy import *
from matplotlib.pyplot import plot, figure
```

(a) Plot $f(x)$, f_{Legendre}, and $f_{\text{Chebyshev}}(x)$ between zero and one on the same graph. Plot $f(x) - f_{\text{Legendre}}$ and $f(x) - f_{\text{Chebyshev}}(x)$ on the same graph with the same range. The first minimizes the squared difference on $[0, 1]$, but it has large errors near the edges. If you were writing a routine to use for calculating $\sin(2\pi x)$ to machine precision in this range, would it be better to use the Legendre or the Chebyshev approximation? Now plot Plot $f_{\text{Chebyshev}}(x) - f_{\text{Legendre}}$ in the range $-1, 2$. Does it indeed get much flatter than you would expect given the coefficients?

[ ]:
```
def f(x):
    return sin(2*pi*x)

def fTaylor(x):
    return 0.000 + 6.283*x + 0.000*x**2 - 41.342*x**3 + 0.000*x**4 + 81.605*x**5

def fChebyshev(x):
    return 0.0066 + 5.652*x + 9.701*x**2 - 95.455*x**3 + 133.48*x**4 - 53.
 ↪39*x**5

def fLegendre(x):
    return 0.016 + 5.410*  + 11.304* **2 - 99.637* **3 + 138.15* **4 - 55.26* **5

x = arange(0,1,0.01)
plot(x,f(x), x,fChebyshev(x), ...)

figure()
plot(x,...)

figure()
xBigRange = arange(-1,2,0.01)
plot(xBigRange,...)
```

Your answer here. Double click to edit. If you were writing a routine to use for calculating $\sin(2\pi x)$ to machine precision in this range, would it be better to use the Legendre or the Chebyshev approximation?

Does it indeed get much flatter than you would expect given the coefficients?

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, the integral of the squared difference between the polynomial and the function:

$$\begin{aligned} C &= (1/2) \int_0^1 (f(x) - y(x))^2 \, dx, \\ y(x) &= \sum_{\alpha=0}^{N-1} \theta_\alpha x^\alpha \end{aligned} \tag{2}$$

How quickly does this cost increase as we move the N parameters θ_α 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 do not cost much-for example, the difference between least-squares and Chebyshev fits given above.

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…

(b) Note that the first derivative of the cost C is zero at the best fit. Analytically (paper and pencil) show that the Hessian second derivative of the cost is

$$\mathcal{H}_{\alpha\beta} = \frac{\partial^2 C}{\partial\theta_\alpha \partial\theta_\beta} = \frac{1}{\alpha+\beta+1}. \tag{3}$$

Your answer here. Double click to edit.

This Hessian is the Hilbert matrix, famous for being ill-conditioned (having a huge range of eigenvalues). Tiny eigenvalues of $\mathcal{H}$ correspond to directions in polynomial space where the fit does not change.

(c) Numerically calculate the eigenvalues of the 6×6 Hessian for fifth-degree polynomial fits. Do they indeed span a large range? How big is the condition number (the ratio of the largest to the smallest eigenvalue)? Are the ratios all approximately equal (a characteristic of sloppy models)?

[ ]:
```
def H(M):
    """ Note: Mth degree polynomials give M+1 x M+1 Hilbert matrices"""
    return array([[... for m in range(M+1)] for n ...])

# Warning: eigh orders smallest to largest eigenvalue.
eigs5,vecs5 = linalg.eigh(H(5))
print("Eigenvalues = ", ...)
print("Condition number = ", ...)
print("Ratios = ", ...)
```

Your answer here. Double click to edit. Do they indeed span a large range? Are the ratios all approximately equal (a characteristic of sloppy models)?

Notice from our Hessian 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 us check this.

(d) Calculate the eigenvector corresponding to the smallest eigenvalue of $\mathcal{H}$, checking to make sure its norm is one (so the coefficients are of order one). Note that the elements of this vector are the coefficients of a polynomial pertubation $\delta f(x)$ that changes the cost the smallest amount for a unit vector . What is that polynomial? Plot the corresponding polynomial in the range $[0, 1]$: does it stay small everywhere in the interval?

[ ]:
```
sloppiest = ... # Is it the first row or the first column?
print("Sloppiest = ", sloppiest)

def sloppyPolynomial(x):
    """If you send a vector x in, sum will sum over all the evaluated␣
 ↪polynomials.
```

```
        axis = 0 will sum only over n, not also over x."""
    return sum([an * x**n for ...],axis=0)

plot(x,...)
```

Especially for larger M, the monomial coefficients of the best fit to a function become sloppy – they can vary over large ranges without damaging the fit, if the other coefficients are allowed to compensate. Only a few combinations of coefficients (those of the largest Hessian eigenvalues) are well determined. 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 θ_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) 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.

One should note that, were we change basis from the coefficients θ_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 in terms of this new basis is the identity matrix. This puzzled us for some time - is the sloppiness intrinsic, or just a sign of a poor choice of variables. Later work, examining the predictions of nonlinear models using information geometry, resolved this question: sloppiness is under rather general conditions expected for the collective predictions of multiparameter nonlinear models.