SloppyExponentialsHintsPython

January 16, 2024

Sloppy exponentials

(Sethna)

The problem of extracting the decay rates from a sum of exponential decays is a famously difficult inverse problem, from the early days of radioactivity to modern simulations of lattice quantum chromodynamics. In a series of exercises, we shall use our information geometry ideas to study the simplest version of this problem: the sum of N exponential decays:

$$y\left(t\right) = \left(1/N\right) \sum_{\alpha=1}^{N} \exp(-\theta_{\alpha} t). \tag{1}$$

We anticipate that it will be challenging to disentangle decay rates θ which are close to one another, unless one has high-precision data over large ranges of time. All the decay curves are smoothly monotonically decreasing, and one could imagine modeling a sum of two decays with a single intermediate decay rate. You shall find in these exercises that this simple model illustrates the behavior we have found widespread in multiparameter models in physics, engineering, biology, and other fields.

In this first exercise, we presume we have perfect experimental data for the decay $d(t)$ at M points t_i equally spread for t between 0 and 10, with separation $\Delta t = 10/M$. We shall be considering how well this data can be represented by other values of the parameters , so our cost is:

$$C(\ ,\ ^{[0]}) = \sum_{i=1}^{M} \left(y\left(t_i\right) - y_{\ [0]}(t_i)\right)^2 / 2\sigma^2 \approx \int_0^{\infty} \left(1/2\right) \left(y\left(t\right) - y_{\ [0]}(t)\right)^2 \, dt.$$

where for convenience (since our data is perfect) we set $\sigma^2 = 1/\Delta t$. We shall use the continuum approximation to evaluate the Hessian at the best fit.

To start, suppose $d(t)$ has two decay rates $^{[0]} = [1, 2]$, so the data $d(t) = (1/2)(\exp(-t) + \exp(-2t))$.

(a) Write a function that returns $y\left(t\right)$, and a function that computes the cost for $\Delta t = 0.01$. Draw a contour plot of C in the square $0.5 < \theta_{\alpha} < 2.5$, with contours at $C = \{2^{-12}, 2^{-11}, \ldots, 2^{0}\}$. Set the number of grid points per side to 40 (so $\Delta\theta = 0.02$) to see the two minima.

[ ]:
```
%matplotlib inline
from numpy import *
from matplotlib.pyplot import plot, scatter, subplots, axes, contour, show
from scipy.optimize import minimize, least_squares
```

```
from mpl_toolkits import mplot3d
```

[ ]:
```
def y(thetas,t):
    """It will run faster if you note that exp(-theta*t) with (t=array([t1,t2,..
 ↪.])
       will give the array of exponents """
    return (...)*sum([... for theta in thetas],axis=0)

# Test your function with a plot

thetas0 = ...
ts = arange(...)
plot(ts, y(thetas0,ts))
```

[ ]:
```
def C(thetas,thetas0,t):
    """Cost function: assumes equally spaced points in time"""
    dt = t[1]-t[0]
    return sum(dt*...)
thetaEval = arange(0.5,2.5,0.02)
levels = 2.**(arange(-14,-4))
costContours = [[C([th1,th2],thetas0,ts) for th1 in thetaEval] for th2 in␣
 ↪thetaEval]
fig, ax = subplots()
costContourPlot = contour(thetaEval, thetaEval, costContours, levels=levels)
ax.set_aspect(1)
```

The diagonal in this plot gives single exponential decays. How well does a single exponential capture the behavior at $^{[0]}$?

(b) Constraining $\theta_1 = \theta_2$, find the point of minimum cost $\theta_{\min}$. Where is the point on the contour plot? Compare the two curves $y_{\theta^{[0]}}(t)$ and $y_{\theta_{\min}}(t)$, and also plot their difference.

[ ]:
```
def toMinimize1exponent(theta):
    return C([...],thetas0,ts)

localThetas = arange(1.3,1.5,0.01)
plot(localThetas,[toMinimize1exponent(theta) for theta in localThetas] )

minTheta = minimize(..., 1.5).x
value = toMinimize1exponent(...)
print("Best single theta =", minTheta, " with cost ", value)
```

Your answer here. Double click to edit. Where is the point on the contour plot?

[ ]:
```
plot(ts, y(thetas0,ts))
plot(ts, y(...,ts))
```

[ ]:
```
plot(ts, y(thetas0,ts)-y([...],ts))
```

One can see from the contour plot that measuring the two rate constants separately would be a challenge. This is because the two exponentials have similar shapes, so increasing one decay rate and decreasing the other can almost perfectly compensate for one another.

This clearly is not a deep truth for two exponentials. But the effect is hugely magnified when we have many parameters. We can see this by computing the eigenvalues of the cost Hessian.

(c) Analytically calculate the Jacobian $J_{t\alpha} = \partial y\,(t)/\partial t_\alpha$ in the continuum approximation. Using the Jacobian, show that the Hessian for the cost evaluated at the best fit is

$$\mathcal{H}_{\alpha\beta} = \left.\frac{\partial^2 C(\ ,\ _0)}{\partial\theta_\alpha \partial\theta_\beta}\right|_{[0]} = \frac{2}{N^2}\frac{1}{(\theta_\alpha+\theta_\beta)^3}.$$

Your answer here. (Double click to edit.)

(d) Using your answer from part (c), write a routine to calculate the entire array $H(\ )$. Check it by examining the eigenvectors and eigenvalues for the $N = 2$ case of part~(b). What do you predict the ratio \$R = \$ (long axis/short axis) to be, in terms of the two eigenvalues λ_{stiffer} and $\lambda_{\text{sloppier}}$? Are the directions roughly in line with the eigenvectors?

[ ]:
```
def H(thetas):
    N = len(thetas)
    return [[... for thetaBeta in thetas] for thetaAlpha in thetas]

print(H(thetas0))

vals, vecs = linalg.eigh(H(thetas0))

print("Eigenvalues of N=2, \n", ...)
print("Ratio of long to short axis =", ..., "roughly correct / wrong")
print("Slope of stiff (short) axis = ", ...)
print("Slope of sloppy (long) axis = ", ...)
print("This agrees/disagrees with slope at", thetas0, "in contour plot")
```

(e) For a sum of seven exponentials, with $^{[0]} = [1, 2, 3, \ldots, 7]$, construct the Hessian, and find its eigenvalues. Are they sloppy (roughly equally spaced in log)? By roughly what factor does each successive eigenvalue shrink?

[ ]:
```
thetas7 = arange(1,8)
Hess7 = H(...)
vals7 = linalg.eigvals(...)
print("Eigenvalues of N=7, \n", ...)
print("Eig ratios = ", ...)
print("These ratios are roughly the same / going to one / getting bigger, \n␣
  ↪making the system sloppy / not sloppy / sloppier than expected")
print("The ratio of the largest to smallest eigenvalue is", ...)
```

This sloppiness makes it strikingly difficult to extract the parameter values from the data.

(f) Argue that the number of measurements n_{measure} needed to estimate a parameter scales inversely with its variance ($n_{\text{measure}} \sim 1/\sigma^2$). Given that the eigenvalues of the Hessian give the

variance along the various eigendirections, by what factor $n_{\text{sloppy}}/n_{\text{stiff}}$ is it harder to measure the parameters along the sloppy directions, for your sum of seven exponentials?

Your answer here. (Double click to edit.)

Measurement errors scale with repeated measurements as $1/\sqrt{n}$, so to reduce a measurment error of σ to $O(1)$ takes $1/\sqrt{n} = \sigma$ so $n \sim 1/\sigma^2$ measurements, which is also the eigenvalue along the eigendirection. So $n_{\text{measure}} = 882,593,000$ measurements are needed to determine the sloppiest parameter combination as well as the stiffest parameter is estimated with a single measurement.

(g) Given that the diagonal elements of the inverse cost Hessian, $(\mathcal{H}^{-1})_{\alpha\alpha}$ are proportional to the variance in parameter α for one sampling of the Gaussian given by the cost, what are the variances in the seven parameters $\theta_\alpha^{[0]}$?

[ ]:
```
Hess7inv = linalg.inv(...)
sigmas = [... for n in range(len(Hess7inv))]
print("Parameter uncertainties = ", sigmas)
```