PlottingModelManifoldHintsPython

January 27, 2024

Plotting the model manifold

(Sethna)

In this exercise, we shall use our N parameter model of decaying exponentials explore ways of visualizing the resulting behavior.

```
[ ]: # Sometimes gives interactive graphs in new windows
     # %matplotlib

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

     # Interactive windows inside notebook! Must include plt.figure() between plots
     %matplotlib notebook

     # Better than from numpy import *, but need np.sin(), np.array(), plt.plot(),␣
      ↪etc.
     import numpy as np
     import matplotlib.pyplot as plt

     # When you just need one or two routines, import them directly
     from mpl_toolkits import mplot3d
     from sklearn.decomposition import PCA
     from scipy.stats import reciprocal
```

Remember $y\,(t) = (1/N)\sum_{\alpha=1}^{N}\exp(-\theta_\alpha t)$.

```
[ ]: def y(thetas,t):
         """Remember to not loop over the times: exp(array([1,2,3])) =␣
      ↪[e^1,e^2,e^3]"""
         """Remember to use np.sum and np.exp, with axis=0"""
         return ...
```

A quick look at two styles of making plots: functional (matlab-like) and object oriented.

```
[ ]: thetas0 = np.arange(...);
     ts = np.arange(0,10,0.01);

     # Functional programming: analogous to matlab.
     # Sensible names, but not very flexible
     # Not supported for all applications
     plt.plot(ts, y(thetas0,ts))
     plt.title("Testing y(thetas,t)");
     plt.xlabel("t");
     plt.ylabel(r"$y(\theta_0,t)$");

     # More flexible object-oriented programming.
     # Unfortunate nomenclature: "subplots" needed to do a single plot.
     fig, ax = plt.subplots()
     ax.plot(ts, y(thetas0,ts))
     ax.set_title("Testing y(thetas,t)");
     ax.set_xlabel("t");
     ax.set_ylabel(r"$y(\theta_0,t)$");
```

One way of visualizing the behavior space is to pick two or three quantities of interest, and explore how they vary with one another. This is a projection of the model manifold onto the three coordinate axes of interest.

(a) Taking $N = 2$ different exponents, draw the projection of the the model manifold onto the axes corresponding to $t = \{1/3, 1, 3\}$. (That is, do a 3D parametric plot of $\{y(1/3), y(1), y(3)\}$, varying $0 < \theta_1 < \infty$ and $0 < \theta_2 < \infty$.) You'll want some of the θs to be small and some large. (I had a range $\sim (10^{-2}$-$10^2)$. Identify the two values of θ for the three pointy endpoints. What simpler model (with only one parameter) is associated with the three edges of the manifold?

```
[ ]: ts3 = np.array(...)
     thetaEval = np.logspace(...)
     pointsAr = np.array([[y([th1,th2],ts3) for th1 in thetaEval] for ...]);
     Xar = pointsAr[:,:,0]; Yar = pointsAr[:,:,1]; Zar = pointsAr[:,:,2];

     # Object oriented! Functional interface doesn't have plt.zlim() defined.
     plt.figure()
     ax = plt.axes(projection='3d')
     ax.plot_surface(Xar, Yar, Zar, color='w')
     ax.axes.set_xlim(0,1)
     ax.axes.set_ylim(0,1)
     ax.axes.set_zlim(0,1)
     ax.set_box_aspect((1,1,1))
     ax.set_title("...");
     ax.set_xlabel("...");
     ax.set_ylabel("y(1)");
     ax.set_zlabel("...");

     # When you've rotated to your satisfaction, stop before new plot
```

Your answer here. Double click to edit.

Identify the two values of θ for the three pointy endpoints. What simpler model (with only one parameter) is associated with the three edges of the manifold?

What happens when we take N exponentials? The model manifold will no longer be a surface – it will fill an N-dimensional manifold, and its projection into 3D will also fill a volume. We cannot expect to do this with a grid of points: 10 points in each direction for $N = 7$ θs would be 10^7 curves. Let us choose random vectors to get an idea of the shape.

(b) Select a large number of random vectors in the space of parameters $0 < \theta_\alpha < \infty$. Starting with $N = 2$, reproduce the model manifold you found in part (a). You'll want some of the θs to be small and some large to get to the edges: choose values with probability $\rho(\theta) \propto 1/\theta$ in the same range you used in part (a). Try $N = 7$. Rotate the 3D plot to see how "thick" the model manifold is.

```
[ ]: plt.figure()
     M=10000 # or other large number
     # Random number generation with distribution rho(x)
     # reciprocal(a,b) gives rho(x) ~ 1/x between limits a, b. reciprocal.rvs␣
      ↪generates random elements from rho.
     th2s = reciprocal.rvs(...,size=(M,2));
     pointsR2 = np.array([y(th2,ts3) for th2 in th2s])
     Xr2 = pointsR2[:,0]; Yr2 = pointsR2[:,1]; Zr2 = pointsR2[:,2];

     ax = plt.axes(projection='3d')
     ax.scatter(Xr2, Yr2, Zr2, s=2)
     ax.axes.set_xlim(...)
     ...
     ax.set_box_aspect(...)
     ax.set_title("...")
     ...;

     plt.figure()
     # reciprocal has rho(x) ~ 1/x between limits. reciprocal.rvs generates random␣
      ↪elements from rho.
     th7s = reciprocal.rvs(...,size=(M,7));
     pointsR7 = np.array(...)
     Xr7 = ...

     ax = plt.axes(projection='3d')
     ax.scatter(..., s=0.1)
     ...
```

You should find a thin sheet with what appears to be pointy-tipped scallops along one "edge". For $N = 1$, there were three points on the manifold, including the two at the ends of the edge.

(c) How many points do you observe for $N = 7$? From part (a), argue that the three cuspy points for $N = 2$ correspond to values where $y(t)$ is a constant except perhaps at $t = 0$. Is the same (likely) true for $N = 7$?

Your answer here. Double click to edit.

The cusps in the model manifold are simpler models with no adjustable parameters! We shall find in general that the edges, corners, and hyper-edges of the model manifold form emergent, simpler models. In our exercise “Model boundaries and noise”, we shall use noise to sample the edges of the model manifold.

Using three predictions is not an exhaustive study for a complex model. Can we create a 3D view of the entire behavior? In our stock-price hypertetrahedron figure, we used principal component analysis to rotate our 5000 dimensional stock price information so that the most important few directions could be separated out and viewed. Let us apply principal component analysis to our data. There are packaged routines you can use for this.

(d) Test your implementation of PCA. Generate random trajectories $y(t)$ for pairs of θs as in part (a), but now for 20 timepoints $y(t)$ evenly spaced with t between zero and ten. Find the first three principal components from these trajectories. You should get a similar manifold (perhaps flipped), except rotated so that the longest axis is along the first component and the narrowest axis is along the third component.

[ ]:
```
plt.figure()
ts = np.arange(0,10,0.5)
pointsR2 = np.array([y(th2,ts) for th2 in th2s])
pca2 = PCA()
pca2Components = pca2.fit_transform(pointsR2)
(Xpca,Ypca,Zpca) = pca2Components.transpose()[0:3]


ax = plt.axes(projection='3d')
ax.scatter(...)

ax.set_title(...)
ax.set_xlabel("...")
...

ax.set_xlim(-1,3)
ax.set_ylim(-1,1)
ax.set_zlim(-1,1)
ax.view_init(90,-90)
ax.set_box_aspect((2,1,1))

# Rotate to see third component
```

(e) Now generate a random set of trajectories with $N = 7$ for $t \in (0, 10)$, and plot the first three principal components. Do they appear to be thinning by roughly a constant factor for each new component? Plot the next three components. Does the manifold continue to get thinner?

[ ]:
```
plt.figure()
ts = np.arange(0,10,0.5)
pointsR7 = np.array([... for th7 in th7s])
```

```
pca7 = ...
pca7Components = ...
(Xpca,Ypca,Zpca) = ...

ax = plt.axes(projection='3d')
ax.scatter(...)

ax.set_title(...)
...

plt.figure()

(Xpca,Ypca,Zpca) = pca7Components.transpose()[3:6]
...
```

The surface swept out by $y(t)$ in the space of trajectories is the model manifold. You have found that it forms a hyperribbon - a geometrical object that is longer than it is wide, wider than it is thick, and so on for as many perpendicular directions as there are parameters. In practice, most multiparameter models share this behavior, and for NLLS models with certain smoothness conditions this hyperribbon behavior can be rigorously proven. And, just as the edges and corners of our exponential decay model correspond to models with fewer exponentials, so too the hyperedges of the hyperribbons for models in these other fields give rapidly converging approximate models for systems with complex microscopic laws.