DoodleVerse: Remaking the Universe in One's Own Image

D. Conathan, Z. Pace, J. Vahala (CS 532; Fall 2015)

November 2015

1 Introduction

Each civilization has their own set of constellations, groups of stars ascribed to a particular piece of cultural memory. The ancient Greeks gave us most of our modern constellations: for example, Orion is a mythical hunter, but the Lakota constellation of the Hand is roughly coincident (and consists of many of the same stars). Could we take an arbitrary constellation, and then find a set of stars that fit it? In this lab, we'll explore how to take a set of about 1000 bright stars, and construct spectacular celestial displays of your own design. Welcome to the DoodleVerse!

Requirements

- Python 2.X (2.7 preferred; 3.X may work, but is not tested)
  - packages: numpy, scipy, astropy (all can be installed with pip)
- The Space-Doodles repo (prepared by the authors), which contains reference code and some utility functions you'll use throughout the lab.

2 Overview

Let's say you draw an outline of a shape (e.g., Fig **??**), and want to find the best way to project it onto the sky. Here are the steps you might follow, when trying to find a best-fit set of stars:

1. Figure out which points define the object (i.e., answer the question, *which points on the outline enable us to recognize the overall outline of the object?*. As it turns out, this is a fairly complicated question—but we'll explore how to solve it with b-splines fit with least-squares.
2. Find the best stars in a catalog to match the particular spatial arrangement of stars from the feature extraction. With infinite computational power, you could, of course, find all possible combinations of all points from the catalog, and compare in all configurations to all possible combinations of all feature points from the object drawing, but that takes an awful lot of time! We're going to be a little smarter about it.
3. With those feature points, set up an optimization problem where the error $|T\ F - s|^2$ is minimized: T is some transformation of feature points F into a subset of stars s ($s \subset S$, and S is the whole list of available stars). This is called the *Procrustes problem.*

Figure 1: Sample drawing (`shape1` in the provided repo), used for future computation and feature matching.

3 Warm-up

3.1 Procrustes problem

Let's consider a set of three points in two dimensions, (`[0.0, 0.0]`, `[1.0, 0.5]`, `[-1.0, 1.0]`), that we're trying to map to another set of points that have the same overall structure, but with a single rotation applied. The second set of points is `([0.0, 0.0], [1.0, 1.0], [0.5, -1.0])`. In this case, it's easy to tell that the rotation is just 90 degrees in a clockwise direction, but how can we find that computationally?

3.1.1 The Brute-force method

Write down the general form for a rotation in two dimensions, and write a few lines of code about how to calculate the actual angle of rotation applied between X_1 and X_2, to within one-hundredth of a degree.

3.1.2 The smarter method

Procrustes analysis finds the optimal rotation, translation, and uniform scaling of points, in order to match other points[1]. Let's explore the form of the solution, by first defining a residual matrix $E = A\,T - B$, formed by applying a transformation T to an initial array of points A, in an attempt to get close to another array of points B. The optimal transformation R between A and B can be stated as $R = argmin_\Omega |\Omega\ A - B|_F^2$. This problem is equivalent to finding the nearest orthogonal matrix to the matrix $M = A^T\ B$. However, one can write M in terms of its SVD and find that $R = V\ U^T$. What condition on the dimensionality of A and B is implicit in this statement? Prove that the best mapping of A to B is provided by the transform $R = V\ U^{T2}$. Now implement the solution you just proved, for the three points given above.

3.2 Many stars

Discuss the implications of trying to fit $N > 3stars$ to 3 feature points. Describe why this problem is not convex, and how you might break it into many convex sub-problems.

4 Main Lab

4.1 Finding feature points

We need to get n feature points from an image. What is a good way of doing this for a simple image like a one-pixel-wide black outline of a diamond on an entirely white background[3] (Build a simple routine that detects the outline of a shape and returns a list or array of pixels that compose that outline).

However, this leaves us with **lots** of feature points. Why do we not want arbitrarily many? There are many ways to cut down on the number of feature points. Describe a couple possibilities, and implement one.

If the shape were wiggly or many-sided, could we exploit the **lack of symmetry** to find the most important/informative points? If you're stuck, think of picking a few points and fitting a closed curve to them (one of many possible ways to do this). Now describe the convexity of such an operation. Don't implement your solution, but feel free to look at ours.

4.2 Reading in the stars data

A catalog of stars is provided in FITS (Flexible Image Transport Standard) format, in the form of a binary table. The astropy package has facilities to read in this table. The following code will read in a FITS table named `data.fits`.

```
import astropy.table as table
data = table.Table.read('data.fits')
```

You can then access columns (features) by, for example, entering `data['a']` to get the value of feature `a` for all entries. You can display all available columns by running `data.colnames`. The spatial features we're interested in are Right Ascension (RA) and Declination (Dec), the equivalent of longitude and latitude in the J2000 celestial coordinate system. Declination is already given in degrees, but Right Ascension is given in Hour Angle, where the celestial sphere is divided into 24 hours. Fortunately, you can just map this to

[1]Procrustes analysis is based on an incidental character in Greek mythology, who made each of his guests fit his bed perfectly by either stretching them or cutting portions of their legs off. Hopefully these activities are somewhat less painful.

[2]If you get stuck, check out Peter Schönemann's 1966 paper on the subject.

[3]?

degrees by multiplying by a constant (it's pretty easy to figure out what that constant is). Otherwise astropy tables work basically the same as standard numpy arrays.

Finally, since the night's sky is periodic (i.e., it wraps around back to 0 hours at 24 hours), the constellations will never cross the so-called 0-hour circle. How could you get around this?

Read in the FITS table of stars in the provided Github repo, convert RA to degrees, and implement a quick way to ensure some constellations are found around the 0-hour circle.

4.3 Implementing the SVD

We now have n feature points, and a set of many stars. If we are trying to map all n features points to a plane, what is the minimum value of n?

Suppose we picked n random feature points and tried to match them with n stars. How can we exploit our knowledge of the constellation's outline to figure out if the stars are worth mapping to our constellation?

Now say that we found a set of n stars that have a compatible geometry to our chosen n feature points. From the Procrustes analysis section, we know how to find the best mapping between them. How would we add the next k best feature points to the constellation?

Pick a small region of the night's sky (say, all points within 5 degrees of a point of your choosing), and use a Procrustes transform to find a best-fit for three feature points found from the `shape1.png` file, using the process above. Did you find an acceptable fit? Discuss one or two ways you could get a better fit?