Draft version May 1, 2020
Typeset using LaTeX **twocolumn** style in AASTeX63

Using Astrosiesmology and Genetic Algotithms to Measure Stellar Properties

Peter Craig

1. INTRODUCTION

Astrosiesmology is a useful way to measure the properties of a star. The basic idea is that stars can exhibit siesmic activity, through what are essentially sound waves propogating throughout the star. These waves can travel throughout the entire object, and if observed can provide information not just about general stellar properties like mass and radius, but also internal and core properties. This is one of the only ways to actually probe the interior of a star to find out what is going on on the inside.

The way to observe such fluctuations is simply by taking a light curve. As sound waves inside the stellar atmosphere reach the surface of the star, they cause small changes in the radius of the star. As the radius of the star changes, the luminosity also oscillates, since the luminosity is dependent on the size of the star. So by observing changes in the lightcurve, one can measure stellar oscillations.

It works out that you can characterize most of the waves inside of a star using three integers. The first, n, is the number of radial nodes inside of the star. The second, l, which for a given n value ranges from 0 to n, is the harmonic degree, and the third is the azimuthal order, m which ranges from -l to l. In order to extract the most information possible, you should compare these frequencies to your data. It works out that modelling all of the oscillations is rather complicated, so my analysis will ignore everything except for the l = 0 modes (Di Mauro 2016).

As you might expect, since we are dealing with waves, the way that you can actually get information out of these light curves is by taking a fourier transform, and looking for peaks. There will typically be many different peaks, representing different modes of oscillation on the inside of the star. The frequencies of these modes are pretty sensitive to the conditions inside the star, and can be used to infer a great deal of useful information. In my case I will only look at the mass and radius, because they are relatively simple to determine from these frequency peaks. Many other things are possible, but they rerquire appreciably more effort.

Shown in equations 1 and 2 are a set of scaling relations that can provide estimates for the mass and radius of a star given two parameters from the fourier series. The first is ν_{max}, which is the frequency of the strongest mode in the star. $< \Delta\nu >$ represents the average frequency difference between adjacent modes with the same l value. In our case, we will just look at the $l = 0$ cases, as these tend to produce the most power in a power spectrum (Kjeldsen & Bedding 1995) (Mathur et al. 2012).

$$\frac{R}{R_\odot} \approx \left(\frac{135\mu Hz}{< \Delta\nu >}\right)^2 \left(\frac{\nu_{max}}{3050\mu Hz}\right) \left(\frac{T_{eff}}{5777K}\right)^{\frac{1}{2}} \quad (1)$$

$$\frac{M}{M_\odot} \approx \left(\frac{135\mu Hz}{< \Delta\nu >}\right)^4 \left(\frac{\nu_{max}}{3050\mu Hz}\right)^3 \left(\frac{T_{eff}}{5777K}\right)^{\frac{3}{2}} \quad (2)$$

2. CODE

You can find all the code / videos that are described below in the Project folder of this github repository.

https://github.com/AdkPete/ASTP720.git

2.1. *How It Works*

The basic principles of this code are based upon the theory of natural selection. The idea is that we want to optimize some fitness function, $f(x)$, through an evolutionary process. The way this works is that we generate a bunch of sample solutions randomly selected from throughout our parameter space, which are called creatures. Each creature is then assigned a fitness, which is obtained by plugging the creatures parameters into our fitness function. Once this is done, we can proceed to update out population until we have found our answer.

The parameters for each creature is stored as a string full of 1s and 0s. This string is effectively the DNA of the creature, and will be used for all future operations on this creature. The exception, of course, is that we turn this back into a list of parameters in order to evaluate fitness, but otherwise everything gets done with this string. Each 1 and 0 can be called a gene, and the set of genes that fully specify a given parameter is called a

chromosome.

Each update is accomplished by selecting out some fraction of the population, preferentially selecting creatures with high fitness values. Then, these creatures are allowed to reproduce, so two parents produce a single offspring that borrows properties from both parents. Then we kill off creatures that have low fitness scores, to maintain our population size. This is then repeated for some number of generations, until we have found a suitable answer.

2.2. *Reproduction*

The goal of the reproduction code is to take in two parents, and produce a child with similar characteristics. The way we do this is by selecting a random number between 0 and the number of genes in the parent DNA, L. Every gene in the DNA string before this number comes from parent number 1, and everything afterwards comes from parent 2. This way the child randomly borrows some genes from parent 1 and other genes from parent 2.

The last step here is to introduce a possiblility of mutations. The way that this works is that we go through the DNA of the creature one gene at a time. For each gene, we draw a random number U between 0 and 1. If $U < 1/L$, then we flip this gene froma 1 to a 0 or 0 to 1. This process helps to keep the code from converging too quickly, and increases the odds that you find a global maxmimum. The rate of mutations is selected such that roughly 1 in 2 children will have a mutation. We also require that the parameters after the mutation remain inside the bounds of our problem. Otherwise, we just keep the original DNA because there is no point sampling regions where we know there is no solution.

2.3. *Test Results*

In order to test the code, I set up an N-dimensional Gaussian Pdf, and then asked the algorithms to find the maximum. I've shown a plot below showing how the population evolved for a 3d case. You will also find a short 30 second movie that shows the algorithm converging on a solution in a test case with 3 dimensions.

You can see that the popultion moves into a smaller and smaller region around the correct answer over time. This run used a population size of 150, and gives a good result within about 10 iterations. In real problems it is usually best to run with at least several hundred creatures, and often thousands.

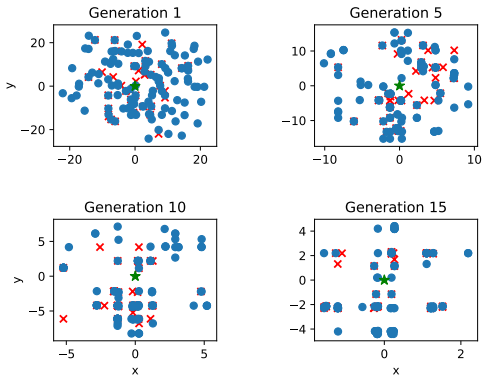

Figure 1. This figure shows the results of an algorithm test at 4 different generations. Blue dots represent old creatures, red x's mark new child creatures, and the green star at (0 , 0) shows the right answer.

3. RESULTS

I used a Python package called lightkurve to obtain my data. This package basically lets you download lightcurves from the Kepler mission (Lightkurve Collaboration et al. 2018). In this case, I have selected a star with no transiting exoplanets, which was fairly bright, and was observed many times by the satellite. I have shown the raw light curve data in figure 2. The next step, which is also done by the python package, is just to generate a periodogram for this data, which I have shown in Figure 3. The peaks that can be seen in the right panel are caused by the oscillations in the star, and are what we are interested in.

The next bit is a bit hacked together at the moment. Basically we want to select out only the peaks in the periodogram here, and then fit a gaussian to those peaks using my genetic algorithms code. I've written Python codes to pull these peaks out for this source, but it would require some fiddling to use with another source. So then the genetic algorithm tries to find the least squares fit to this data set. What we really care about is effectively just the mean of this distribution, which will be ν_{max}. For now, I have used a method built into the lightkurve package to determine $< \Delta\nu >$.

In Table 1 you will find the parameters for this star determined by my code from this data set. The real parameters for ν_{max} and $\Delta\nu$ are both taken from other methods implemented in the lightkurve package, for comparison to my values. You can see that this has actually produced a pretty good estimate for the mass and radius of the star.

4. CONCLUSIONS

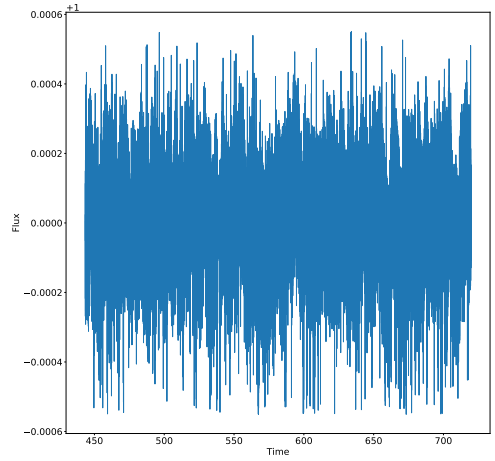

Figure 2. This figure shows the lightcurve from our star. This lightcurve was produced through many observations of the Kepler satellite.

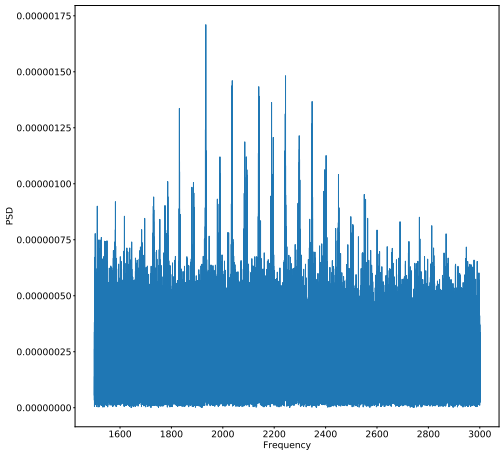

Figure 3. This figure shows a periodogram of our light curve. The spikes correspond to different modes of oscillation inside of our star.

Table 1. Table showing my estimates for a number of parameters, along with real values determined by other methods.

Parameter	Genetic Value	Real Value
ν_{max}	2191	2175.00 μHz
R	1.263	$1.2R_{\odot}$
M	1.173	$1.1M_{\odot}$

I used a genetic algorithm to measure stellar properties from a light curve using astrosiesmology. I was able to sucessfully measure the mass and radius of this star with this method using Kepler obtained light curves, which I think is pretty exciting. Over the course of this project, I have learned a lot about genetic algorithms, including how they work, why they work and what kinds of problems that they are good at. It turns out that a genetic algorithm like this one is a good way to solve a wide variety of optimization problems, and they are especially useful for problems with high dimensionality.

There are still some assorted improvements that I would like to make to this code. These include finding a better way to handle mutations and reproduction, adding support for more randomly selecting which individual creatures are selected to have children. I'd also like to add code that requires that no two creatures can have the same DNA, as this leads to evaluating the fitness at the same set of parameters more than once, which is decidedly inefficient.

I'd also like to try this out on some harder optimization problems. The things that I have thrown it so far are relatively simple, often for the sake of keeping the run time low in order to get things ready in a timely fasion. It would be interesting to run it on some standard optimization test problems and see how it compares to other algorithms, like ones implemented in scipy.

REFERENCES

Di Mauro, M. P. 2016, in Frontier Research in Astrophysics II (FRAPWS2016), 29. https://arxiv.org/abs/1703.07604

Kjeldsen, H., & Bedding, T. R. 1995, A&A, 293, 87. https://arxiv.org/abs/astro-ph/9403015

Lightkurve Collaboration, Cardoso, J. V. d. M., Hedges, C., et al. 2018, Lightkurve: Kepler and TESS time series analysis in Python, Astrophysics Source Code Library. http://ascl.net/1812.013

Mathur, S., Metcalfe, T. S., Woitaszek, M., et al. 2012, ApJ, 749, 152, doi: 10.1088/0004-637X/749/2/152