Using Genetic Algorithms to Find the Period of a Single Exoplanet Orbit

Ceré L. Rettig

Lawrence Technological University

May 13, 2015

Abstract

Genetic algorithms are a way to randomly search through a parameter space in a relatively short time to avoid months of computation for one data set. It is a computing method which works for clean continuous data sets and sets with noise and data gaps. I found and modified python code for genetic algorithms to use for both spiral galaxy rotation and for a single exoplanet in a circular orbit around a star. I will discuss both galaxy rotation for spiral galaxies and for a single exoplanet in a circular orbit around a star.

I. Introduction

Genetic algorithms are a way to randomly search through a parameter space in a relatively short time to avoid months of computation for one data set. It is a computing method which works for clean continuous data sets and sets with noise and data gaps. Genetic algorithms borrow terminology from biology to breed successful individuals, and add mutation which allows you to randomly hop around parameter space. While trying to fit parameters to a model, the genetic algorithm allows us to vary these parameters and search through parameter space without strenuous computation searching for every point. There is a fitness condition which keeps the best of each run then "reshuffles" the data to find the best solution for the next run. If there are three parameters, then the genetic algorithm is essentially searching all the points in 3D-space. If there are more parameters there are more dimensions. Having three parameters, each with, for example, 10,000 possible answers, that is $(10{,}000)^3$ which would take months of strenuous computational power for one set of calculations, while using the genetic algorithm it goes through one set of calculations in less than one hour. In the genetic algorithm there are random mutations which occur that is based on a threshold that is set as well as breeding. This should allow for more successful fits to create a successful offspring. Both mutation and the breeding through directed selection are random fluctuations which allow for searching through a large parameter space. Anytime you need to search through parameter space, you should be able to find a genetic algorithm routine to help you do that.

II. Results and Discussion

A. Encoding/Decoding the Genetic Strings

While working with the genetic algorithm the encoding and decoding process illustrates what the genetic algorithm is doing. To set up the strings it is necessary to find out how big of a number is going to be used, how much resolution and how many decimal places are required. So, for example taking the magnitude of the range as 100 and changing the divisions within that range to be also 100, multiplying them, gets 10,000 possible numbers to search through. With genetic algorithms, the way the string is encoded is in binary, so it is necessary to use the next highest power of two, for this genetic algorithm it is 2^{14} so 14 bits of binary information are needed. For example a range can be coded from 0 to 9.999 and depending on the resolution the decimal place can be shifted within that four digit number. These are the ranges/resolutions chosen for this project, they can be adjusted as needed.

Borrowing from the language of biology, the individual is a string of binary numbers. Using three variables, they are grouped together to be one string. In the equations used there are three variables each 14 bits in length, which allows each string to have a length of 42 bits of binary. A string of 14 bits of binary in

base 2 encodes a number which gives 2^{14-1} divisions of a range. That string is an individual, each character in that string is a gene (1 or 0), and each 14 bit part of that string represents one of the three parameters which can be converted into a number. Once those parameters are decoded from the string, they can be put into the model equation. Having real world (x, y) data, when values for the parameters are obtained they can be plugged into the model equation for the same x-values and calculate a y-model, which can then be compared again a real y-value and use a sum of least squares as the fitness – the lower the number, the more fit. If the program finds the same value, for example 1000 times in a row, then the genetic algorithm has locked onto a good answer and outputs the results. Now that the encoding and decoding process is understood, an understanding of the genetic algorithm process can be understood.

B. The Genetic Algorithm Process

In the GA process there are four steps: (1) initialization, (2) the evolution phase which include: selection, crossover, and mutation, (3) the evaluation (decoding) phase, (4) determining if our data is a good fit and what happens if the data is not a good fit. In the initialization phase an initial population is randomly generated. In the evolution phase we select tournament style, so we select five randomly throughout the population and pick the best one, (evaluating the fitness of each string) then we select five randomly again, pick the best one; we can breed those two. There is a probability of taking from one parent or the other parent to form a new individual we keep repeating that to form a new population from the old one. Then we go through and mutate every member in that population, there is a small probability of mutating any individual gene within each string. So the randomizing lets us search through all the space but the breeding lets us try and pick the ones that have a better fitness value to give us better individuals. Then we evaluate those individuals to see which is the best one, keep the best one. The evaluation tests the parameters in the model to seek a low least-squares value. Keep in mind, this is directed selection rather than natural selection; it is more like dog breeding, we are selecting for particular traits. Each cycle, the best string is kept. If that value repeats consecutively a chosen number of time, say 1000, then the process seems to have locked onto the relatively best string, thus the best parameters.

Figure 1. A graphical representation of the genetic algorithm process. Starting with initialization, this generates a population. From initialization the GA moves into the evolution phase, this is the breeding stage. From evolution it continues on to the evaluation phase, this is where the GA decides if the number is a good fit with our data. If it is not a good fit it goes back to evolution and cycles through until a good fit is found. Once it is a good fit the GA prints the number out. [C.L. Rettig]

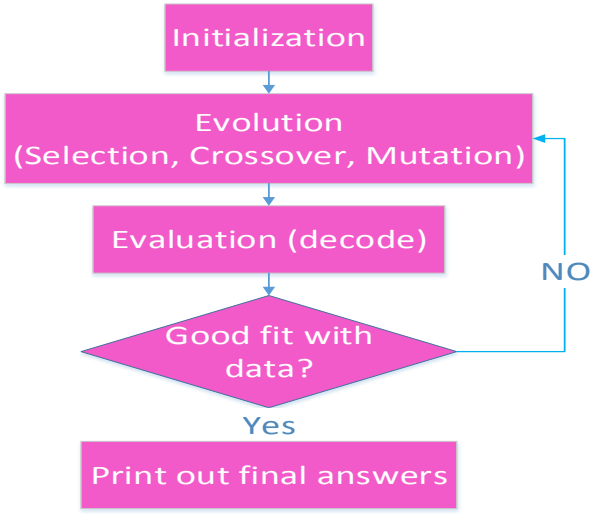


Figure 2. A biological approach to understanding of the breeding process of the genetic algorithm. [P. Charbonneau]

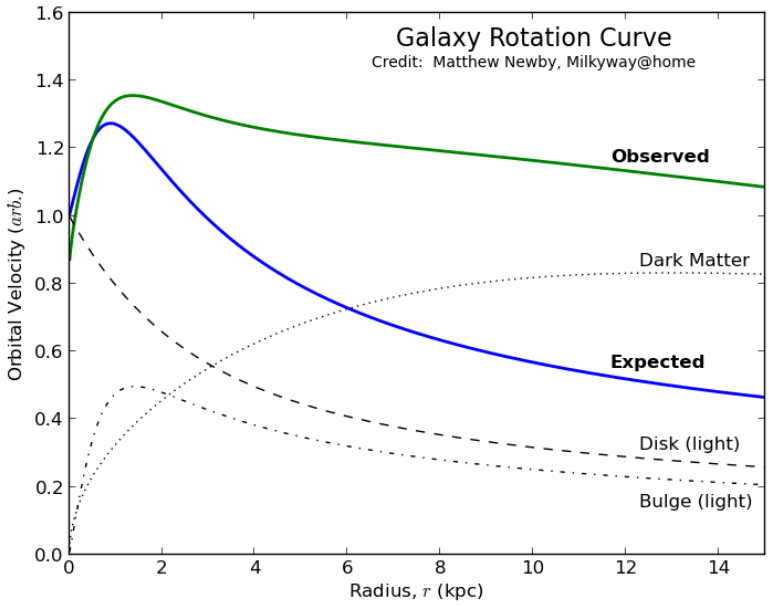
	Ph(P1)	x=0.14429628	y=0.72317247	[01]
Encoding:	Ph(P2)	x=0.71281369	y=0.83459991	[02]
		↓	↓	
		71281369	83459991	[03]
	Gn(P2)	7128136983459991		[04]
Breeding:	Gn(P1)	1442962872317247		[05]
	Gn(P2)	7128136983459991		[06]
(a) Crossover (gene=4):				
		144 2962872317247		[07]
		↓↑		
		712 8136983459991		[08]
		144 8136983459991		[09]
		712 2962872317247		[10]
	Gn(01)	1448136983459991		[11]
	Gn(02)	7122962872317247		[12]
(b) Mutation (Offspring=02, gene=10):				
	Gn(02)	7122962872317247		[13]
		712296287 2 317247		[14]
		712296287 6 317247		[15]
	Gn(02)	7122962878317247		[16]
Decoding:	Gn(02)	7122962878317247		[17]
		71229628	78317247	[18]
		↓	↓	
	Ph(02)	x=0.71229628	y=0.78317247	[19]
	Ph(01)	x=0.14481369	y=0.83459991	[20]

FIG. 2.—Encoding, breeding, and decoding in genetic algorithms. Phenotypes are defined in terms of two real numbers and are encoded as a string of 16 decimal digits (eight per real number). "Ph(P1)" means "phenotype of parent P1," "Gn(O2)" is "genotype of offspring 2," and so on. Encoding is shown only for Ph(P2), and decoding for Gn(O2). Note that a breeding event produces *two* offspring, and that both crossover and mutation operations occur only if a probability test yields true (*see text*).

C. Genetic Algorithms for Spiral Galaxy Rotation

Vera Rubin is an American astronomer who pioneered work on galaxy rotation rates. She uncovered the discrepancy between the predicted angular motion of galaxies and the observed motion, by studying galaxy rotation curves – this became known as the galaxy rotation problem. A galaxy rotation curve is a graphical analysis obtained by plotting the orbital velocities of visible stars in a particular galaxy against how far out from the center of the galaxy. One should see the velocity dropping off but the velocity stays relatively constant. Dark matter could be the reason for this. Dark matter accounts for effects that appear to be the result of mass, where such mass cannot be seen. It attracts or exerts a gravitational inward pull on the visible matter surrounding it. It is not dark energy because dark energy repels or pushes outward. Working with a galaxy rotation model, there are three variables and the python code works for three variables. A problem was found while looking in the literature research for the model and data sets. There was only a starting point for the model equations for dark matter – and some of the data sets were not readily available. Although while networking with Dr. Timothy Mc.Kay data sets from observation were obtained, working with the genetic algorithm became limited since there was only part of the information necessary for dark matter and no equations for the disk or bulge. This led to using the genetic algorithm for another project.

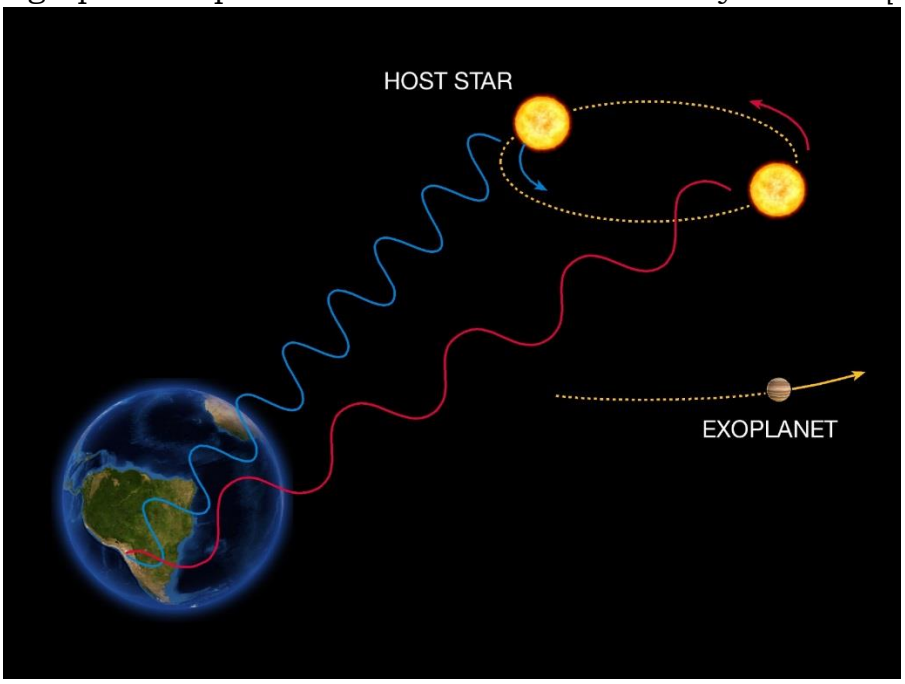
Figure 3. A Galaxy Rotation Curve [A. Treason]



D. Genetic Algorithms for Finding the Period of a Single Exoplanet Orbit

While there were difficulties with using the genetic algorithm for spiral galaxy rotation, it was used for finding a planet in a circular orbit around a star. If a planet is orbiting around a star, the star and the planet both orbit a common center of mass. The star will make a small orbit itself around that center of mass, which means it is sometimes coming toward the Earth and sometimes going away from the Earth (if the orientation of the system is "edge-on" to the Earth). The star's light would be blue shifted or red shifted, and the Doppler shift velocities are obtained. Having the Doppler shift velocity of a star it can be plotted as a function of time. The relative toward/away motion of that star system relative to our own is subtracted to give only the planet-induced motion of the star.

Figure 4. A graphical representation of the radial velocity method. [EOS]



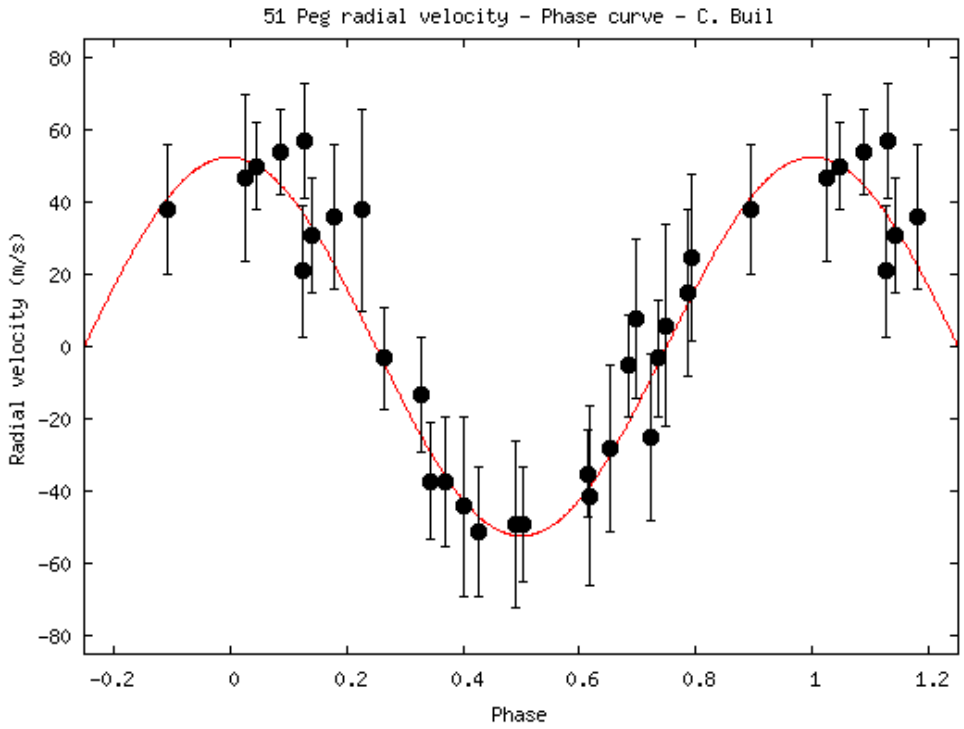
E. Finding the Period of 51 Pegasus

Looking closely at 51 Pegasus there is a velocity as a function of phase or relative place in one period of that orbit. This phase plot is obtained by looking at the time series, starting with a peak and looking forward in time to pick another peak with the same period at 0 and plot the velocity relative to phase rather than time. Plotting velocity against phase rather than time easily shows any outliers in the data set. The graphs produced use the function

$$v = A \sin\left(\frac{2\pi t}{B} + C\right) \tag{1}$$

and will plot velocity as a function of time. A is the amplitude velocity, B is the period (in days) and C is the phase angle. The amplitude is important because it gives information on the mass of the planet – the bigger the amplitude the bigger the planet. If planet is close to a star the amplitude will be larger (stronger force) and there will be a change in the period, this give the mass of planet as long the mass of the star is known. The period is the distance from the star and while the phase angle is not particularly important, the purpose is it shows when the observer began looking at the planet which is the offset needed to lock onto a period and amplitude. These three parameters: the amplitude, period and phase angle will be used in the genetic algorithm. The x-data is time (in days), and the y-data is the velocity (in km/s). When the three parameters and a real time are placed into the equation a value is obtained to find a predicted velocity and compare to the known actual velocity. This can be put into a sum of least squares to check for fitness. Once the most-fit set of parameters is found, they can be compared to the published data in the Exoplanet Database.

Figure 5. 51 Pegasus Radial Velocity as a Function of Phase Angle. [C. Buil]



F. The Exoplanet Database

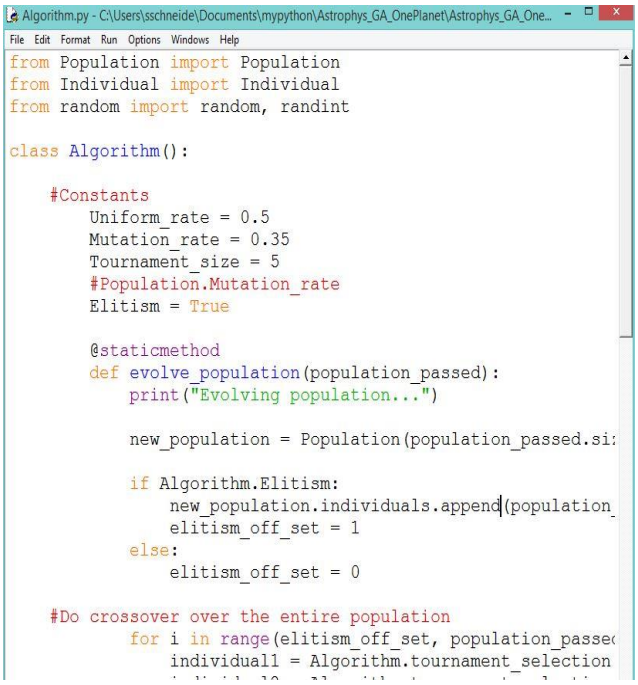
The exoplanet database shows confirmed exoplanets. There are 1831 total, so they need to be sorted through. They can be filtered by using the columns needed and taking out what is not needed. A single planet with a small eccentricity is needed to work with planets that have a circular orbit. A single planet and a circular orbit is needed since the genetic algorithm is coded for

three variables. Since plotting the velocity as a function of time, the radial velocity is also needed. The examples that will be used are 51 Pegasus since historically it is the first star found to have an orbiting exoplanet. Tau Boo has an interesting name, but I later found it is interesting since scientists have observed the flipping of its magnetic field in 2007, which shows it is a star much like our sun. HD195019 was chosen because it has a period greater than 10 days but it happens to be a perfect star for this project, as I will show.

G. Genetic Algorithm Code Example

Looking at an example of the code for HD195019, in Figure 6, the uniform rate which is part of the breeding, states that there is 50% chance of picking from one parent or the other. The mutation rate states that 35% of the time we mutate a particular gene. These numbers can be varied. The tournament size of 5 says to pick 5 randomly then pick the best of that five, pick 5 randomly then pick the best of that five and repeat that process until a new population is formed.

Figure 6: A Sample of the GA Code. [C.L. Rettig]

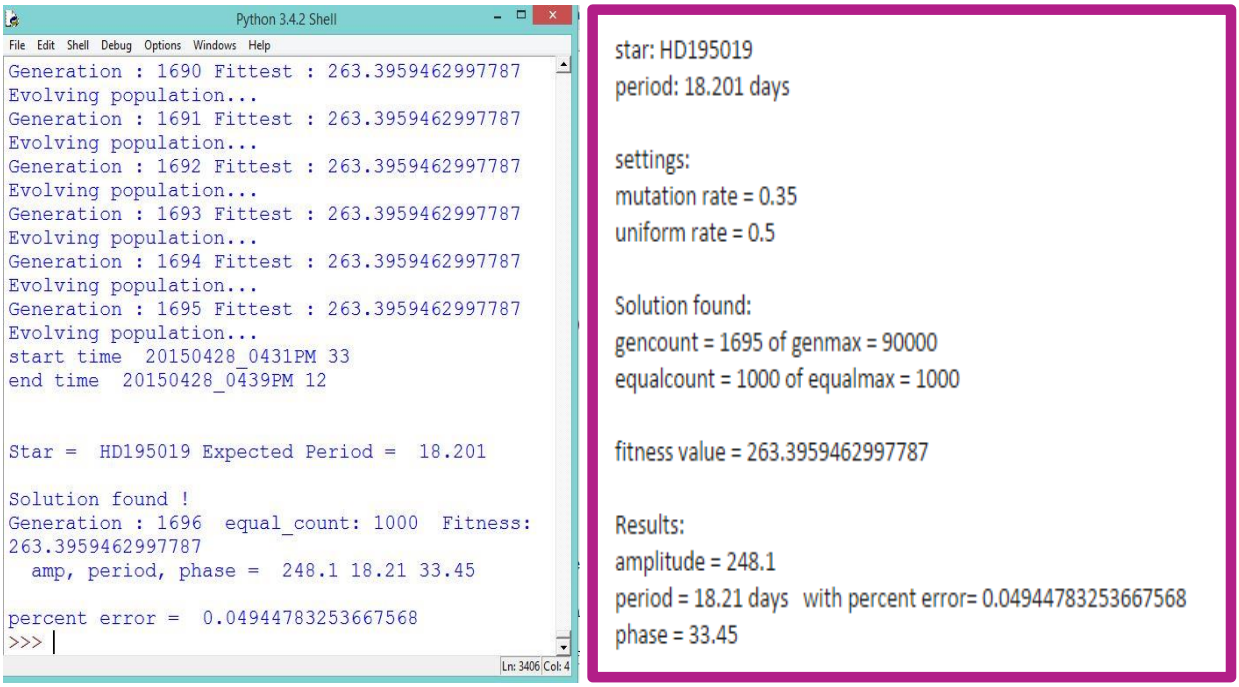


H. Genetic Algorithm Ouput Example

Looking at the code output for HD195019, in Figure 7, shows the fitness is the same for the last six runs but the equal count shows that the genetic algorithm came up the same answer 1,000 times and stopped since this was the threshold that was set. If it didn't stop at 1,000 times then it would have run for a maximum of 90,000 times. None of the data runs ever made it to the

genmax of 90,000 times or whatever value was chosen for the genmax for a particular run. So for this example found a fitness of 263.4 to be the best, 1000 times in a row. The percent error is for this particular run of HD195019 is 0.049%, this is a good data run but, actually not the best. Now to look at the data of the three star-exoplanet examples.

Figure 7. Genetic Algorithm Code Output. [C.L. Rettig]



I. Looking at the Exoplanet Orbits of 51 Pegasus, Tau Boo, HD195019

This is the best data run for these three particular examples, unlike the previous example which was just randomly chosen to further explain the genetic algorithm. Knowing the period, amplitude and phase angle one can see how well the data fits by plotting the model data against the actual data to see how well it fits. Noticing there are four significant figures, these can be shifted as needed. 51 Pegasus and Tau Boo had a range of 10 while HD195019 had a range for 100, because there are only 14 bits, it is limited to four decimal places but it was reasonable for this model.

Table I. Best Data run for the three examples. [C.L. Rettig]

Star	Expected Period	Calculated Period	Percent Error (%)
51PEG	4.230785 days	4.240 days	0.002
TauBoo	3.3124 days	3.313 days	0.018
HD195019	18.201 days	18.20 days	0.005

1. 51 Pegasus Data

Looking again at 51 Pegasus, the overall period is a good fit with an expected period of 4.23079 days and a calculated period of 4.240 days. The early part of the data series shows this, since there is more data earlier on it dominates the series, even though the later series seems to not match as well. With fewer later data points it does not affect the data series as much, but as previously stated the period agreement is good. This is the largest percent error of the three stars. There were some stars that didn't lock on which would require more investigation of the data files. This method works well for a reasonable number of the stars.

Table II. 51 Pegasus Data. [C.L. Rettig]

Star	Expected Period (days)	Calculated Period (days)	Percent Error (%)	Amplitude (km/s)	Phase Angle (rad)
51PEG	4.2308	4.240	0.218	49.58	4.34

2. Tau Boo Data

With Tau Boo the later data series fits well, but the early data series does not. This is because there is fewer data for the earlier series so the later series dominates. While the amplitude does not match it does find a reasonable period value. Tau boo has a robust period agreement with an expected period of 3.3124 days and a calculated period of 3.313 days.

Table III. Tau Boo Data [C.L. Rettig]

Star	Expected Period (days)	Calculated Period (days)	Percent Error (%)	Amplitude (km/s)	Phase Angle (rad)
TauBoo	3.3124	3.313	0.018	40.63	1.28

3. HD 195019 Data

HD195019 is practically perfect in every way, since the period and amplitude match well for both the earlier and later data series. The expected period is 18.201 days and the calculated period is 18.20 days. Notice the gaps in the data, although data was not taken every day the code still locks onto the period having those gaps. These gaps could be due to the planet being on the other side of HD195019 so it would not be viewable or it could be due to poor weather conditions. It is not made clear why these data gaps exist. This method works well for single planet systems with a circular orbit. With more data there is a better fit, and a much smaller percent error, not that any of the data runs shown are particularly bad. Single planet systems with an elliptical orbit and multiple planet systems would be a more a bit more complicated.

Table IV. HD195019 Data. [C.L. Rettig]

Star	Expected Period (days)	Calculated Period (days)	Percent Error (%)	Amplitude (km/s)	Phase Angle (rad)
TauBoo	3.3124	3.313	0.018	40.63	1.28

III. Conclusion and Future Directions

A. Conclusion

In this project I was able to obtain a vast knowledge and understand of genetic algorithms, programming in python, and the many uses of genetic algorithms. While I was unable to complete the work on genetic algorithms with spiral galaxy rotation, I was able to use the genetic algorithm with minimal modification for a single exoplanet orbit around a star. I was able to produce good results which shows not only my depth of knowledge in genetic algorithms but applying them to different uses in astronomy and astrophysics.

B. Future Directions

For future directions I would like to go back to the galaxy rotation and find or create all of the model equations necessary to use the genetic algorithm for spiral galaxy rotation. Then once this is accomplished I would like to see if this works for lenticular galaxies.

For furthering the project with a single planet orbit around a star either I or future students could modify the code for a multiple planet system or for an elliptical orbit. For a multiple planet system it should be reasonably simple, the string would just need to be expanded. A multiple planet system would add sine

curves together, one for each planet. A single planet system with an elliptical orbit would be a bit more challenging as a new fitness equation would need to be found and changed in the genetic algorithm. It would not be using sine curves anymore so it would be more complicated. The data sets that did not work out could be investigated further to see if there are any outliers that may need to be eliminated. This code can be used for other astronomy and astrophysics projects or with some modifications any project searching through a parameter space.

IV. Acknowledgements

I would like to acknowledge: Lawrence Technological University for having senior projects as part of the undergraduate curriculum. The Department of Natural Sciences for having a well developed senior project curriculum. Dr. Timothy McKay at the University of Michigan for his help in obtaining galaxy rotation data. Dr. Scott Schneider for being the most helpful and supportive senior project adviser, I could not have made it this far alone. My friends and family for always being there, or letting me be when I needed time for my project.

References

J.Q. Feng and C.F. Gallo, Res. Astron. Astrophys. **11** 1429-1449, 1 (2011).

P. Charbonneau, Astrophys. J. Suppl. Ser. **101** 309-334, 1 (1995).

L. Jacobson, theProjectSpot, http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3 (2012).

A. Rebelo, GitHubGist, https://gist.github.com/arthurrebelo/6689655 (2013).

A. Treason, Milky Way, http://milkyway.cs.rpi.edu/download/images/gal_rotation_curve.png (2015).

L. Doyle, SETI Institute, http://archive.seti.org/seti/seti-science/detecting_new/wobble_method.php (2015).

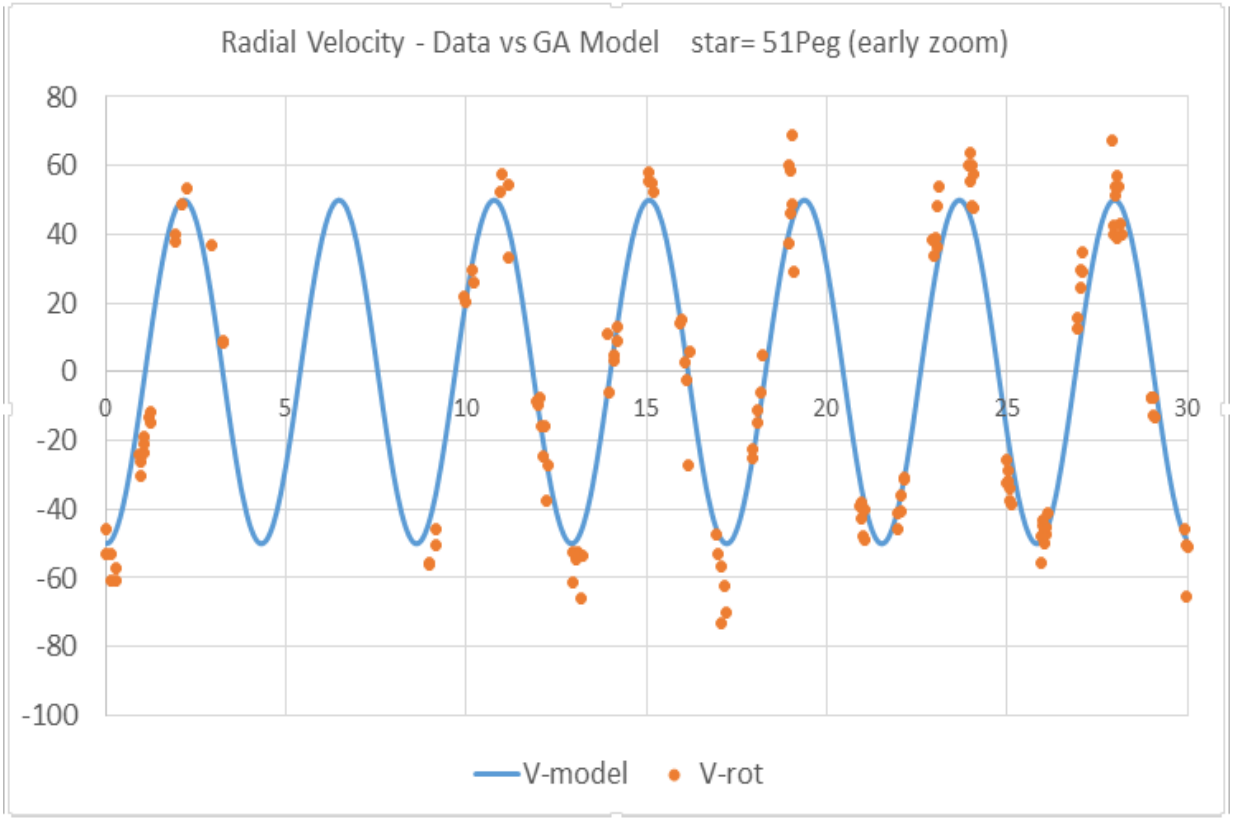
EOS, Radial Velocity Method, http://upload.wikimedia.org/wikipedia/commons/3/33/ESO_-_The_Radial_Velocity_Method_(by).jpg (2007).

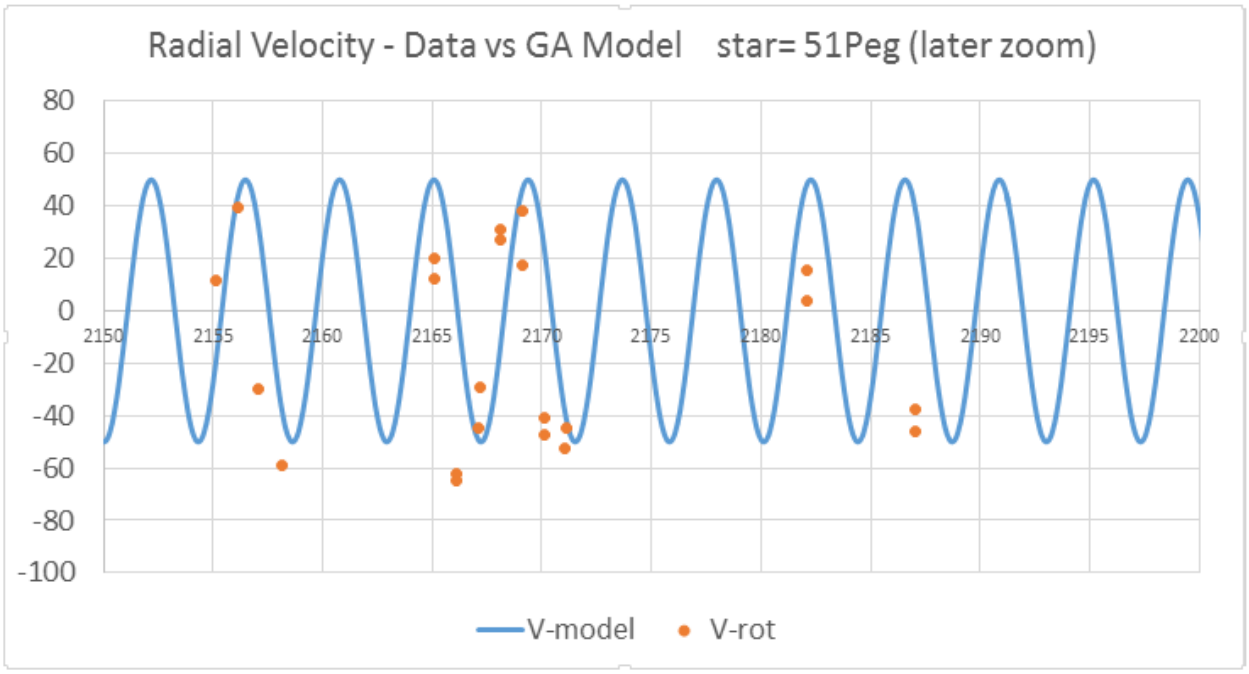
C. Buil, Spectroscopy, CCD and Astronomy, http://www.astrosurf.com/buil/exoplanet2/fig/51peg_phase.png (2015).

R.L. Akeson et al, NASA Exoplanet Science Institute, http://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=planets (2015).

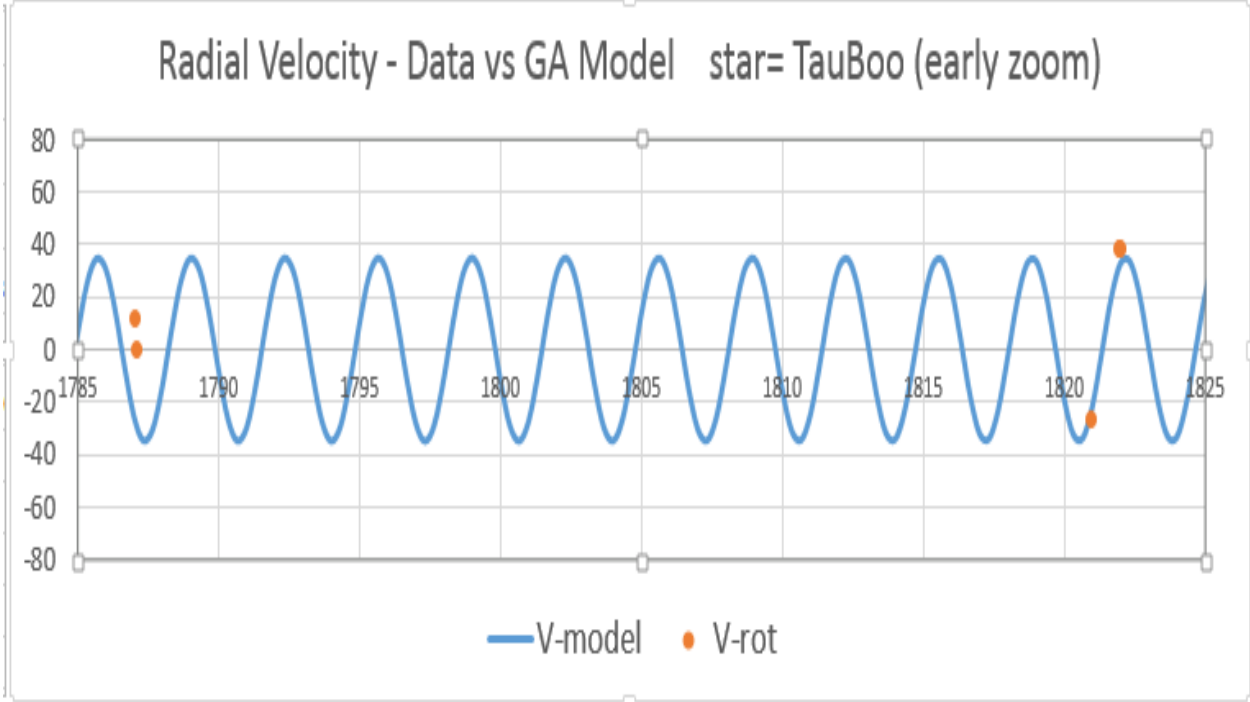
Appendix A: Results

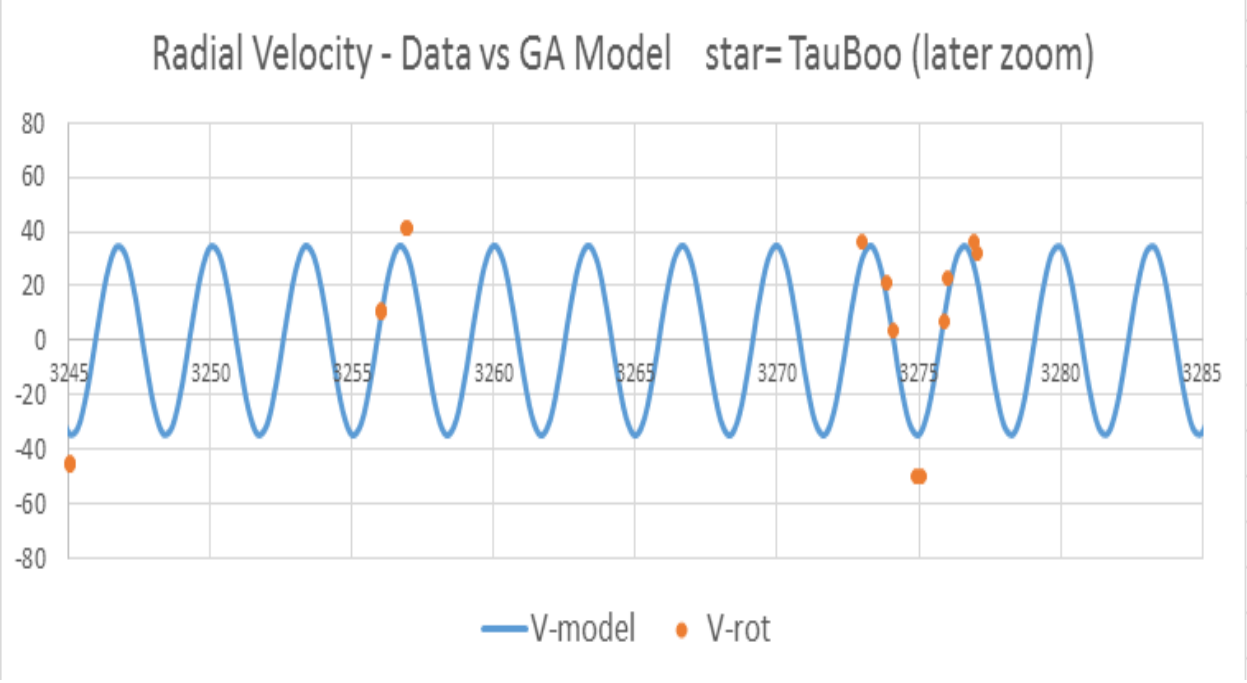
51 Pegasus



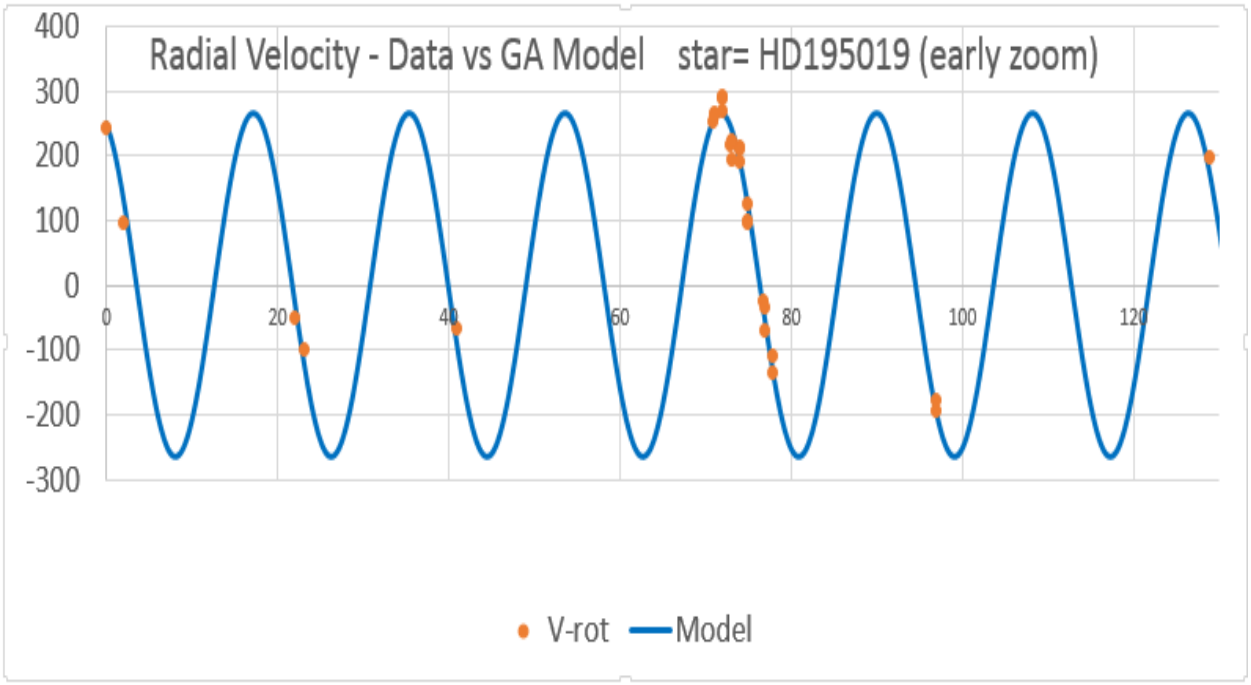


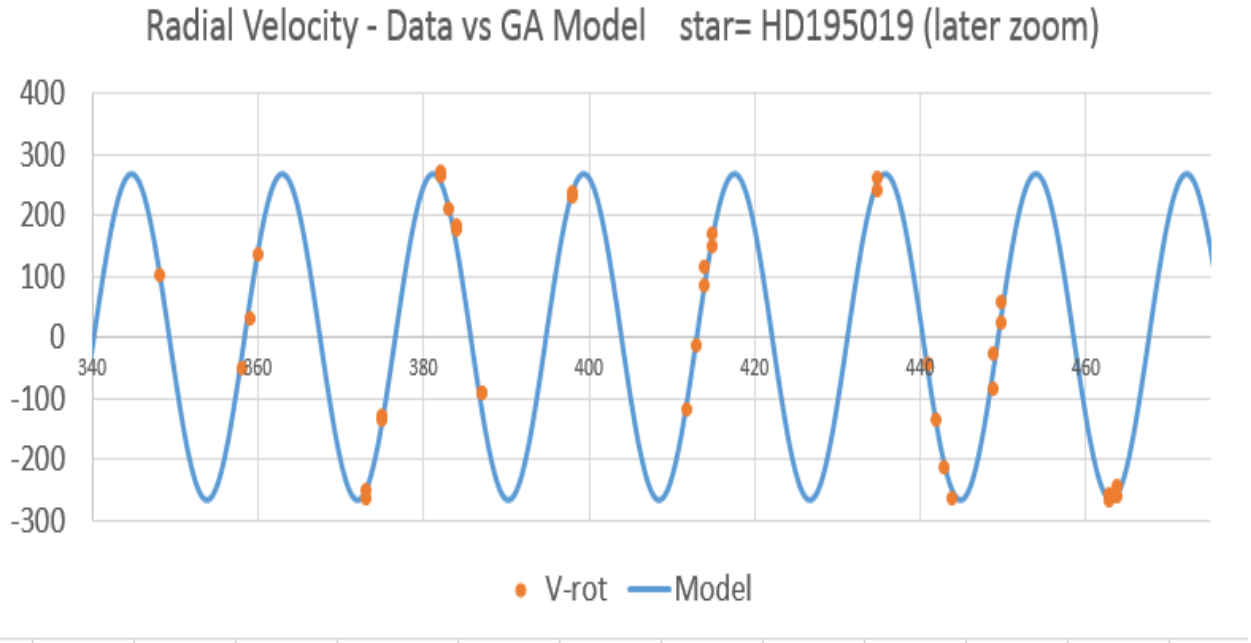
Tau Boo


Radial Velocity - Data vs GA Model star= TauBoo (early zoom)
80
60
40
20
0
-20
-40
-60
-80
1785
1790
1795
1800
1805
1810
1815
1820
1825
V-model
V-rot


Radial Velocity - Data vs GA Model star= TauBoo (later zoom)
80
60
40
20
0
-20
-40
-60
-80
3245
3250
3255
3260
3265
3270
3275
3280
3285
V-model
V-rot

HD 195019


Radial Velocity - Data vs GA Model star= HD195019 (early zoom)
400
300
200
100
0
-100
-200
-300
0
20
40
60
80
100
120
V-rot
Model


Radial Velocity - Data vs GA Model star= HD195019 (later zoom)
400
300
200
100
0
-100
-200
-300
340
360
380
400
420
440
460
V-rot
Model

Appendix B: Python Code for Genetic Algorithm

GA.py

```
#from FitnessCalc import FitnessCalc
from Population import Population
from Algorithm import Algorithm
from time import time
from Individual import Individual
from datetime import datetime, date, time
import math

start = time()

#FitnessCalc.set_solution("1001110001000010011100010000100111000010000")

my_pop = Population(50, True)


generation_count = 0
gen_max = 800000
equal_count = 0
equal_max = 1000

old_fitness = 0

while (my_pop.fitness_of_the_fittest() >= 0 and generation_count<gen_max and equal_count
< equal_max):
        generation_count += 1
        print("Generation : %s Fittest : %s " % (generation_count,
my_pop.fitness_of_the_fittest()))
        my_pop = Algorithm.evolve_population(my_pop)
        if (my_pop.fitness_of_the_fittest() == old_fitness):
```

```
                equal_count = equal_count +1
        if (my_pop.fitness_of_the_fittest() != old_fitness):
                equal_count = 0
        old_fitness = my_pop.fitness_of_the_fittest()


my_fittest = my_pop.get_fittest()

AA=0
for i in range(Individual.DefaultGeneLength-29,0,-1):
        j = Individual.DefaultGeneLength-i-29
        AA = AA + my_fittest.genes[i]*(2**j)
AA = AA*100/(2**14-1)
AA = round(AA,2)

BB=0
for i in range(Individual.DefaultGeneLength-15,14,-1):
        j = Individual.DefaultGeneLength-i-15
        BB = BB + my_fittest.genes[i]*(2**j)
BB = BB*100/(2**14-1)
BB = round(BB,2)

CC=0
for i in range(Individual.DefaultGeneLength-1,28,-1):
        j = Individual.DefaultGeneLength-i-1
        CC = CC + my_fittest.genes[i]*(2**j)
CC = CC*100/(2**14-1)
CC = round(CC,2)

goodperiod = float(Population.starperiod)
percenterror = 100*math.fabs(goodperiod-BB)/goodperiod

print("\n\nStar = ", Population.starname, "Expected Period =
",Population.starperiod,"\n")
```

```
print("Solution found !\nGeneration : %s  equal_count: %s  Fitness: %s \n  amp, period,
phase = " % (generation_count + 1,equal_count, my_pop.fitness_of_the_fittest()),AA, BB,
CC,"\n")
print("percent error = ",percenterror)

#print("Genes of the Fittest : %s " % (genes_the_fittest))

d = datetime.today()

goodperiod = float(Population.starperiod)
percenterror = 100*math.fabs(goodperiod-BB)/goodperiod

f = open(Population.starname+'_'+d.strftime("%Y%m%d_%I%M%p")+'.txt','w')
f.write('star: '+Population.starname+'\n')
f.write('period: '+Population.starperiod+' days \n\n')
f.write('settings:'+'\n')
f.write('mutation rate = '+str(Algorithm.Mutation_rate)+'\n')
f.write('uniform rate = '+str(Algorithm.Uniform_rate)+'\n\n')
f.write('Solution found:'+'\n')
f.write('gencount = '+str(generation_count)+' of genmax = '+str(gen_max)+'\n')
f.write('equalcount = '+str(equal_count)+' of equalmax = '+str(equal_max)+'\n\n')
f.write('fitness value = '+str(my_pop.fitness_of_the_fittest())+'\n\n')
f.write('Results:'+'\n')
f.write('amplitude = '+str(AA)+'\n')
f.write('period = '+str(BB)+' days   with percent error= '+str(percenterror)+'\n')
f.write('phase = '+str(CC)+'\n')
f.close()
```

Population.py

```
from Individual import Individual
#from FitnessCalc import FitnessCalc
import math

class Population():

    starname = 'TauBoo'
    starperiod = '3.3124'
#    starname = 'hd4308'
#    starperiod = '15.56'
#    starname = '51peg'
#    starperiod = '4.230785'
#    starname = 'HD83443'
#    starperiod = '2.985698'
#    starname = 'HD284968'
#    starperiod = '8.7836'
#    starname = 'HD195019'
#    starperiod = '18.201'

    def star_HD195019(self):
# Xvalues      star = HD195019b
        self.dataX = [0, 2.008611, 21.970278, 22.994873, 40.961667, 70.891099, 70.914896,
\
                      70.937222, 71.83603, 71.858287, 71.880683, 72.907951, 72.930683, \
                      72.952836, 73.851794, 73.874768, 73.89743, 74.89206, 74.913032, \
                      74.932951, 76.778032, 76.800382, 76.823368, 77.788831, 77.807581, \
                      96.832882, 96.855417, 128.763912, 295.131574, 348.070544,
358.056053, \
                      359.073299, 360.071192, 372.971238, 372.992697, 375.001562,
375.023935, \
                      382.013264, 382.036111, 383.077164, 384.013727, 384.035972,
387.033333, \
```

```
                    387.056018, 397.91706, 397.939363, 411.867789, 412.902222,
413.857245, \
                    413.881632, 414.794988, 414.817234, 434.765752, 434.788113,
440.837349, \
                    441.798461, 442.767199, 443.841204, 448.801042, 448.820752,
449.787847, \
                    449.810347, 462.7914, 462.813657, 463.757072, 463.779005, 491.7314,
\
                    491.754329, 674.054954, 695.060555, 695.082801, 745.953183,
746.977384, \
                    769.916609, 770.949942, 797.852847, 797.875012, 798.850833,
798.873241, \
                    807.826802, 807.848949, 811.789468, 811.818854, 818.77081,
818.800289, \
                    820.757986, 820.78713, 829.740081, 829.769398, 836.757685,
836.787118, \
                    1069.038277, 1069.068345, 1069.096542, 1071.07873, 1072.051434,
1110.016875, \
                    1110.046308, 1116.932396, 1117.977974, 1122.977766, 1123.008194,
1131.927697, \
                    1131.957512, 1152.848125, 1154.841771, 1155.895718, 1168.815544,
1168.84544, \
                    1175.79391, 1175.822858, 1178.795972, 1178.843623, 1444.099826,
1511.917141, \
                    1511.939745, 1511.962303]
#  Yvalues      star = hd195019b
        self.dataY = [242, 97, -49, -99, \
                    -67, 253, 267, 264, 289, 268, \
                    293, 217, 222, 195, 213, 212, \
                    190, 127, 96, 100, -25, -33, \
                    -69, -135, -109, -176, -194, 199, \
                    -51.9, 104, -49.1, 32.3, 136.8, -248, \
                    -263, -133, -127, 272, 264, 210, \
                    183, 178, -89, -92, 231, 237, \
```

```
                    -115.9, -13.7, 115, 87, 151, 169, \
                    261, 242, -41.4, -136, -210.9, -262.8, \
                    -84, -26, 24, 58, -268, -257, \
                    -261, -243, 230, 226, 213.8, 6, \
                    -11, 249, 205.1, -159.6, -233.5, 220, \
                    232, 250, 278, -253, -233, -128, \
                    -125, 260, 256, 159.5, 164, -154, \
                    -182, 300, 283, 50, 42, 95, \
                    206.1, 231.9, 270, 257, -227.2, -265.7, \
                    16, 46, 51, 10, -203.4, -259, \
                    -246.3, -25, -20, -140, -139, 136, \
                    89, -227.1, 202, 207, 182]


    def star_hd4308(self):
# Xvalues      star = hd4308b
        self.dataX = [0, 0.990558, 3.0165, 3.99956, 5.006756, 6.027873, 32.912893, \
                    34.943398, 36.942106, 273.208007, 277.164706, 278.208274,
279.163962, \
                    280.20039, 281.19853, 282.208481, 343.032293, 344.058, 345.049093,
\
                    346.02164, 347.009896, 348.051362, 349.068996, 351.090313,
353.022602, \
                    371.926708, 373.005438, 373.934791, 374.933486, 375.948144,
446.821622, \
                    447.822796, 448.829736, 449.827375, 453.806768, 454.800974,
455.815892, \
                    460.834832, 461.804359, 462.812048, 463.808086, 464.810547,
465.804867, \
                    466.817301, 468.807459, 469.811276, 470.828447, 471.825197,
472.857963, \
                    475.811996, 480.798691, 484.780129, 488.787938, 489.798738,
588.207913, \
```

```
                      590.126773, 591.201807, 592.185066, 669.037396, 670.134762,
671.144435, \
                      673.186304, 704.013934, 706.121556, 707.019058, 708.065272,
710.060853, \
                      759.958168, 763.892645, 764.826188, 765.885066, 766.87772]
# Yvalues      star = hd4308b
        self.dataY = [-0.3, -0.6, 2.8, 0.6, 4.5, 3.6, -9, \
                      -0.7, 1.1, 5.9, -0.1, 2.6, -1.3, \
                      -3.8, -4.6, -5.5, -1.1, 0, 0.2, \
                      -2.4, -2.5, -0.2, -2.6, -1.5, -0.1, \
                      8.5, 4.6, 0.4, -2.3, -0.4, 3.1, \
                      4.1, 0.3, -0.4, -8.9, -6.7, -2.4, \
                      1.7, 2.6, 3.7, 0.5, 1.1, -0.1, \
                      -0.2, -2.3, 0.5, -1.5, 3.1, 6.6, \
                      7.5, 5, -4.6, 1.6, 4.5, 0.4, \
                      -3.1, -2.6, -3.6, 3, 7.8, -3.7, \
                      -8.4, -0.3, -1.5, 5.5, 2.7, 0.6, \
                      0.2, -2, -4.9, -6.9, -5.8]



    def star_TauBoo(self):
#  xvalues  star = TauBoo
        self.dataX = [0, 236.2514, 236.2566, 236.2619, 265.1271, 265.1318, \
                      265.1373, 266.0712, 266.076, 372.9607, 372.9761, 372.9924, \
                      393.9539, 393.9686, 393.9829, 415.9111, 415.9144, 433.9029, \
                      619.1482, 749.981, 960.2441, 1060.0657, 1418.0203, 1478.9666, \
                      1712.2125, 1765.0523, 1765.0695, 1787.0025, 1787.0596, 1820.9484, \
                      1821.9555, 1821.9779, 2109.1625, 2109.1841, 2138.1137, 2138.1351, \
                      2390.3088, 2453.2048, 2522.9569, 2808.2364, 2898.9251, 2898.9479, \
                      3110.262438, 3110.284938, 3113.264139, 3114.291859, 3215.119174,
3215.144568, \
                      3245.022785, 3245.045494, 3256.024035, 3256.046246, 3256.9176,
3256.940343, \
```

```
                        3272.982855, 3273.892866, 3274.113653, 3274.933433, 3275.057797,
3275.882588, \
                        3276.042866, 3276.93275, 3277.040864, 3304.886385, 3340.888156,
3544.20451, \
                        3545.173665, 3546.009244, 3546.021257, 3546.255285, 3579.117114,
3604.985193, \
                        3654.959591, 3655.973653, 3686.920204, 3696.906616, 3835.279776,
4018.934081, \
                        4018.946639, 4046.922612, 4248.23076, 4345.036825, 4345.975112,
4346.975112, \
                        4578.291674, 4582.257311, 4669.057218, 4670.065297, 4981.225632,
4981.251651, \
                        5083.048619, 5083.056142, 5097.039544, 5375.189301, 5376.180586,
5380.174, \
                        5422.006107, 5422.025112]
# yvalues   star = TauBoo
        self.dataY = [61.2, 38.5, 40.7, 37.6, 37.2, 37.2, 38.7, \
                        40.7, 38.6, -33.1, -27.6, -34.5, 14.5, \
                        15.2, 15, -20.2, -18, 29.9, 11.2, \
                        18, 3.5, 57.4, 59.7, 9, -4.1, \
                        3.3, 3.4, 12.2, 0.2, -26.4, 38.7, \
                        38.8, -46.4, -43.2, -0.3, -0.5, -32.3, \
                        -28.4, -34.9, -35, 46.3, 48.1, 4.2, \
                        7.3, -20, 45.8, -43.5, -42.6, -44.6, \
                        -46.2, 11.3, 10.4, 41.3, 41.4, 36.3, \
                        21, 3.8, -49.6, -49.9, 7.3, 22.9, \
                        36.3, 32, -47.8, -38.6, 6.4, 39.9, \
                        -23.3, -23.1, -40.2, -19.4, 26.1, 10.2, \
                        -47.7, 27.9, 28.9, -28.2, 36.4, 32.2, \
                        -48.7, -26.5, -54.6, 15.2, 27.5, 39.3, \
                        20.1, -39.3, -31.5, -47, -42.2, -38.3, \
                        -37.3, -50.8, -51.6, 15, 34.6, -32.5, -31.4]
```

```
    def star_HD83443(self):
#  Xdata HD83443
        self.dataX = [0,0.982673, 1.989328, 2.984502, 73.860462, 74.84706, 75.706261, \
                      76.754062, 83.857372, 84.841144, 85.867881, 105.702094, 108.80576,
\
                      109.714618, 164.664652, 166.642954, 321.033984, 338.040506,
345.053184, \
                      409.815155, 435.867381, 436.754779, 464.843946, 702.974444,
813.777384, \
                      1091.012233, 1471.990439, 1528.851088]
# ydata  HD83443
        self.dataY = [21.8, -55.3, 34.3, 19.2, 44.9, -6.7, -57.2,45.3, \
                      -8.9, -45.6,49.4,-41.5,-33.2,52.5,-34.1,58.7,-13.2,-48.4, \
                      -1.9, -54.4,23,-54.8,38.9,-35.9,9.4, -46.4,15.3, 6.4]



    def star_51peg(self):
        #51_peg
        self.dataX = [0,0.018645,0.134525,0.150266,0.289016,0.302592,0.945682,\
                      0.961203,0.977268,1.050891,1.066793,1.081793,1.221168,\
                      1.23662,1.251388,1.936013,1.953842,2.119629,2.241558,\
                      2.938952,3.252071,3.265844,8.96559,8.990624,9.160879,\
                      9.183194,9.959155,9.981712,10.191805,10.214618,10.946967,\
                      10.96993,11.157326,11.176655,11.966712,11.988553,12.04581,\
                      12.068819,12.138368,12.161122,12.230405,12.253865,12.948217,\
                      12.97052,13.06853,13.091574,13.187673,13.211226,13.934444,\
                      13.964699,14.084826,14.106805,14.170705,14.194641,15.057442,\
                      15.081145,15.171446,15.192986,15.93728,15.958472,16.088958,\
                      16.112326,16.180277,16.202592,16.945937,16.968043,17.057789,\
                      17.078981,17.172349,17.194444,17.936492,17.959236,18.063645,\
                      18.087291,18.176608,18.198657,18.939027,18.96133,18.984108,\
                      19.00699,19.032777,19.056261,19.078935,20.922569,20.94471,\
                      20.96728,20.99008,21.012812,21.035347,21.060138,21.964594,\
```

```
                    21.982488,22.052222,22.067465,22.143645,22.158842,22.94331,\
                    22.966643,23.038391,23.060786,23.084027,23.106481,23.935856,\
                    23.956805,23.978993,24.001793,24.024872,24.048564,24.070856,\
                    24.970694,24.992638,25.017534,25.035104,25.062546,25.084131,\
                    25.106388,25.928981,25.952199,25.978148,26.000601,26.023553,\
                    26.046134,26.068692,26.091319,26.114224,26.93118,26.953483,\
                    27.023055,27.045671,27.068344,27.091759,27.925462,27.94574,\
                    27.96956,27.995277,28.016955,28.041886,28.062187,28.085416,\
                    28.156631,28.17956,28.984108,29.006828,29.029155,29.051423,\
                    29.07427,29.096666,29.925208,29.947847,29.970312,29.993078]
        self.dataY = [-52.9,-45.8,-60.8,-53.3,-60.9,-57.3,\
                    -24.2,-30.5,-26,-19.2,-23.5,-21.2,-13.5,\
                    -14.7,-11.9,39.7,38,48.8,53.2,36.6,\
                    8.7,  8.2,-55.7,-56.4,-45.6,-50.6,21.7,20.3,\
                    29.5,26,52.3,57.3,54.3,33,-8.8,-9.5,\
                    -7.7,-15.6,-24.4,-15.8,-37.6,-27,-52.5,\
                    -61.1,-54.7,-52.7,-65.8,-53.8,10.8,\
                    -5.8,4.9,3.2,9.2,13.1,58,55.4,55.1,52.3,14.3,\
                    15.1,2.5,-2.3,-27,6.1,-47.6,-53.1,-56.8,-73.1,\
                    -62.2,-69.9,-25.3,-22.4,-14.8,-11.2,-6,4.9,\
                    37.3,60.2,58.6,46.2,48.9,69,29,-39.2,\
                    -38.2,-42.7,-48.1,-39.9,-40.1,-48.8,\
                    -41.3,-46,-40.6,-36,-31.1,-30.7,38.2,33.8,39,\
                    48.3,36.3,53.7,60,55.2,63.9,48,60.2,\
                    57.7,47.6,-32.4,-25.5,-31.9,-29,-33.9,\
                    -37.5,-38.6,-47.7,-55.8,-43.1,-44.6,\
                    -49.9,-46.3,-47.3,-45.1,-41.3,12.4,\
                    15.7,24.6,29.8,34.7,28.9,67.1,\
                    39.9,42.4,54,51.4,57,39,54.1,42.8,40,-7.4,\
                    -7.8,-7.8,-7.4,-12.9,-13.2,-45.9,\
                    -50.4,-65.7,-51.2]

    def star_HD284968(self):
# star = hd284968b           period = 8.7836
```

```
        self.dataX = [0, 211.323, 233.349, 331.141, 389.02, 571.369, 711.107, \
                      742.094, 1043.236, 1061.235, 1396.082, 1468.05, 1495.011, \
                      1676.373, 1696.321, 1734.183, 1735.279, 2058.364, 2462.33, \
                      2559.134, 2885.197, 3001.983, 3142.297, 3244.286, 3299.135, \
                      3557.123, 3558.171, 3559.213, 3560.227, 3590.255, 3615.159, \
                      3616.157, 3621.071, 3622.132, 3625.131, 3651.078, 3652.093, \
                      3705.024, 3705.031, 3706.001, 3706.01]
        self.dataY = [-8.45, -14.26, -11.15, -3.02, -9.8, -16.45, \
                      -16.55, 7.14, -5.92, -10.07, -2.7, -6.69, \
                      -0.62, 1.92, -11.89, 3.21, 4.4, 5.05, \
                      1.92, 4.79, 4.43, -1.47, -15.5, 1.24, \
                      -6.06, 3.72, 4.6, 10.72, 11.2, 0, \
                      0.25, 2.05, 10.51, 5.32, -1.92, -4.3, \
                      1.16, -2.33, -0.41, 3.41, 0.37]



    def __init__(self, population_size, initialise):
        self.individuals = []

        #Creates the individuals
        if (initialise):
            for i in range(population_size):
                new_individual = Individual()
                self.individuals.append(new_individual)

    def get_fitness(self, individual_passed):
        fitness = 0
        sumdiff=0

        self.star_TauBoo()
#        self.star_hd4308()
#        self.star_51peg()
#        self.star_HD83443()
```

```
#         self.star_HD284968()
#         self.star_HD195019()


        self.A = 0
        for i in range(Individual.DefaultGeneLength-29,0,-1):
            j = Individual.DefaultGeneLength-i-29
            self.A = self.A + individual_passed.genes[i]*(2**j)
        self.A = self.A*100/(2**14-1)
        self.A = round(self.A,2)

        self.B = 0
        for i in range(Individual.DefaultGeneLength-15,14,-1):
            j = Individual.DefaultGeneLength-i-15
            self.B = self.B + individual_passed.genes[i]*(2**j)
        self.B = self.B*100/(2**14-1)
        self.B = round(self.B,2)
        if self.B == 0:
            self.B = 0.000001

        self.C = 0
        for i in range(Individual.DefaultGeneLength-1,28,-1):
            j = Individual.DefaultGeneLength-i-1
            self.C = self.C + individual_passed.genes[i]*(2**j)
        self.C = self.C*100/(2**14-1)
        self.C = round(self.C,2)

        for i in range(len(self.dataX)):
            sumdiff = sumdiff + (self.dataY[i]-
self.A*math.sin(2*math.pi*self.dataX[i]/self.B+self.C))**2
        fitness = math.sqrt(sumdiff)

        return fitness
```

```
def fitness_of_the_fittest(self):
    fitness_of_the_fittest = self.get_fitness(self.get_fittest())
    return fitness_of_the_fittest

def get_fittest(self):
    fittest = self.individuals[0]
    for i in range(len(self.individuals)):
        #switched from lessthan or equal to greater than or equal to
        if self.get_fitness(fittest) >= self.get_fitness(self.individuals[i]) :
            fittest = self.individuals[i]
    return fittest

def size(self):
    return len(self.individuals)

def get_individual(self, index):
    return self.individuals[index]

def save_individual(self, index, individual_passed):
    self.individuals[index] = individual_passed
```

Algorithm.py

```
from Population import Population
from Individual import Individual
from random import random, randint

class Algorithm():

    #Constants
        Uniform_rate = 0.5
        Mutation_rate = 0.35
        Tournament_size = 5
        #Population.Mutation_rate
        Elitism = True

        @staticmethod
        def evolve_population(population_passed):
            print("Evolving population...")

            new_population = Population(population_passed.size(),False)

            if Algorithm.Elitism:
                new_population.individuals.append(population_passed.get_fittest())
                elitism_off_set = 1
            else:
                elitism_off_set = 0

    #Do crossover over the entire population
            for i in range(elitism_off_set, population_passed.size()):
                individual1 = Algorithm.tournament_selection(population_passed)
                individual2 = Algorithm.tournament_selection(population_passed)
                new_individual = Algorithm.crossover(individual1, individual2)
                new_population.individuals.append(new_individual)
```

```
#Do mutation randomly
        for i in range(elitism_off_set, population_passed.size()):
            Algorithm.mutate(new_population.get_individual(i))

        return new_population

    @staticmethod
    def crossover(individual1_passed, individual2_passed):
        new_sol = Individual()
        for i in range(individual1_passed.size()):
            if random() <= Algorithm.Uniform_rate:
                new_sol.set_gene(i, individual1_passed.get_gene(i))
            else:
                new_sol.set_gene(i, individual2_passed.get_gene(i))

        return new_sol

    @staticmethod
    def mutate(individual_passed):
        for i in range(individual_passed.size()):
            if random() <= Algorithm.Mutation_rate:
                gene = randint(0,1)
                individual_passed.set_gene(i, gene)

    @staticmethod
    def tournament_selection(population_passed):
        #Tournament pool
        tournament = Population(Algorithm.Tournament_size, False)

        """ Tournament selection technique.
           How it works: The algorithm choose randomly five
           individuals from the population and returns the fittest one """
        for i in range(Algorithm.Tournament_size):
            random_id = int(random() * population_passed.size())
```

```
tournament.individuals.append(population_passed.get_individual(random_id))

            fittest = tournament.get_fittest()
            return fittest
```

Individual.py

```
from random import randint
#from FitnessCalc import FitnessCalc

class Individual():

                                DefaultGeneLength = 42
                                #DefaultGeneLength = len(FitnessCalc.Solution)

                                def __init__(self):
                                 self.genes = bytearray(Individual.DefaultGeneLength)
                                 for i in range(Individual.DefaultGeneLength):
                                      gene = randint(0,1)
                                      self.genes[i] = gene

                                def get_gene(self, index):
                                    return self.genes[index]

                                def set_gene(self, index, what_to_set):
                                    self.genes[index] =  what_to_set

                                def size(self):
                                    return len(self.genes)
```