A Python toolchain for variable star light curve analysis

Christian Dersch

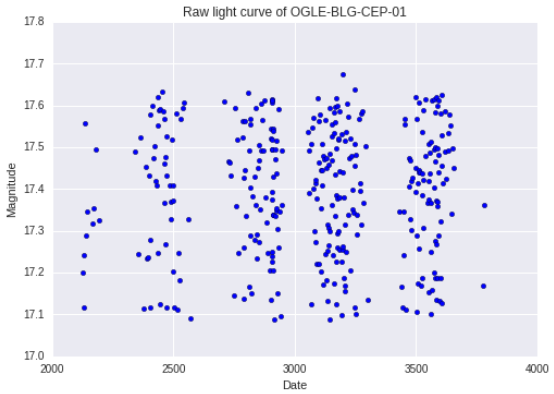
Philipps-University Marburg, Working Group Astronomy

15. September 2016

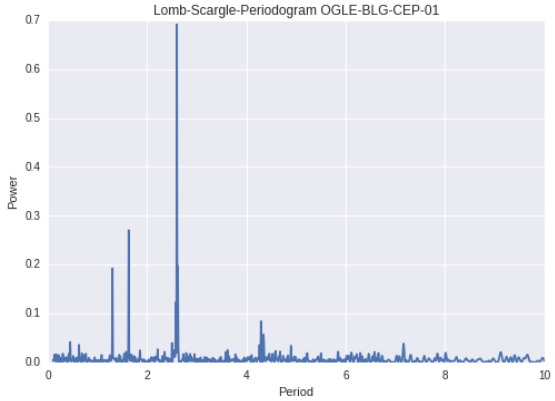
Motivation

- Python is a popular programming language, ACM: "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities"
- Python is **free software**
- Nonfree and often quite expensive software like IDL still very often used
- Many people use their own "historically grown" software
- Establish a free software toolchain available to everyone to establish some kind of standards
- **First GAIA data release**, toolchain allows playing with the shiny new data

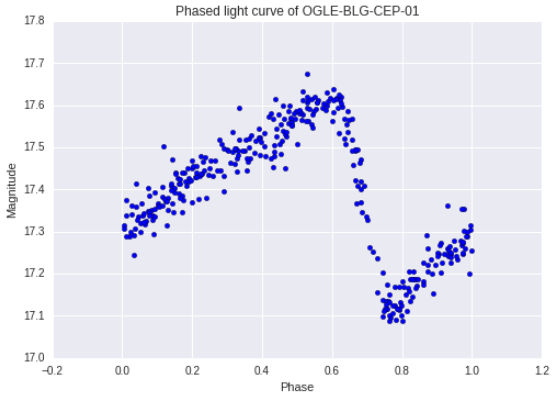
Standard tasks in light curve analysis



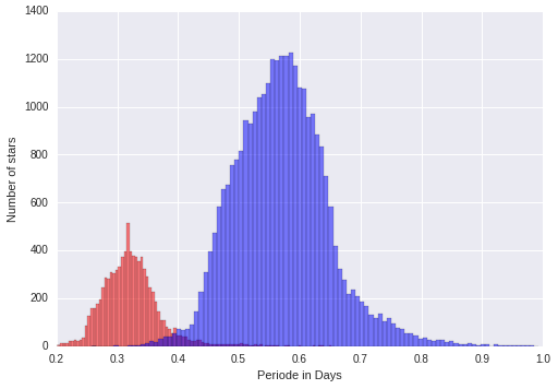
(a) raw light curve



(b) Periodogramm and Fourier components



(c) Phased light curve



(d) Classification

The SciPy-Stack

- Core modules: NumPy und SciPy
- Foundation for scientific computing with Python
- Makes use of C and Fortran under the hood
- Removes many performance issues you normally have with Python due to its nature as an interpreted language

Astropy

- Based on SciPy-Stack
- General astronomy related functions (e.g. coordinate transformations)
- Many affiliated packages, e.g. photutils for photometry
- The projects goal: Make Python a standard tool for astronomical data analysis

scikit-learn

- Based on SciPy-Stack too
- Implementation of many machine learning related algorithms/methods:
  - dimensionality reduction
  - projection methods
  - supervised learning
  - unsupervised learning

astroML, gatspy & Co

- Based on astropy and scikit-learning
- Data Mining for astronomy
- In addition: specialized packages like gatspy for time series analysis

Jake VanderPlas from University of Washington developed the modules in a high quality and well documented way!

Proposed Toolchain

Module	Usage
astroML	Machine Learning (enhances scikit-learn)
Astropy	General astronomy related functions
gatspy	Time series analysis
IPython	Interactive workflow
Matplotlib	Plotting stuff
NumPy	Basic module of SciPy-Stack
Pandas	data organization
scikit-learn	Data Mining / Machine Learning
SciPy	Enhanced scientific calculations, statistics
Seaborn	Additional Plotting
Statsmodels	Statistics

Efforts like Debian Astro and Fedora Astronomy already provide the toolchain!

Advantages

- Again: We are working entirely with free software now!
- As the modules are based on SciPy-Stack they integrate very well, even when they are developed completely independent
- Python is easy to learn and already quite common
- Supports the effort of astropy and other projects

Testing against OGLE-III CVS

- OGLE-III CVS: OGLE-III catalog of variable stars
- deeply analyzed data set
- $\rightarrow$ Test the toolchain against this data set to show it's working
- Periods are very close to OGLE ones: $\Delta P = 0.0001$ days
- Same for Fourier components:

$$\Delta R_{ij} = 0.02$$
$$\Delta \phi_{ij} = 0.001$$

- Test used a probe of 2500 stars with periods < 100 days

Note: This is just an example, we made more tests

Outlook: Sonneberg Archive

- Sonneberg: One of the biggest photo plate archives in the world
- Data coverage: More than 70 years
- Data have higher uncertainty ($\sigma = 0.12$ mag) than OGLE CCD data
- $\rightarrow$ We added more noise ($\sigma = 0.2$ mag) to OGLE data and performed our analysis again
- Results:
  - Error in period increased to

$$\Delta P = 0.001 \text{ Tage}$$

  - For Fourier components:

$$\Delta R_{ij} = 0.11$$
$$\Delta \phi_{ij} = 0.01$$

- Still good enough!

A short demo using two classes of RR-Lyrae (RRab and RRc)

Thank you very much for your attention!
Do you have any questions?

References

- OGLE-III: `http://ogledb.astrouw.edu.pl/~ogle/CVS/`
- Astropy: `http://www.astropy.org/`
- astroML, gatspy: `http://www.astroml.org/`
- Pandas: `http://pandas.pydata.org`
- scikit-learn: `http://scikit-learn.org/`
- SciPy-Stack: `http://scipy.org/`
- statsmodels: `http://statsmodels.sourceforge.net/`

Demo of a short light curve analysis, with shiny new GAIA data!

- Two types of RR-Lyrae (RRab and RRc)

First load the required modules, data and just plot the light curve

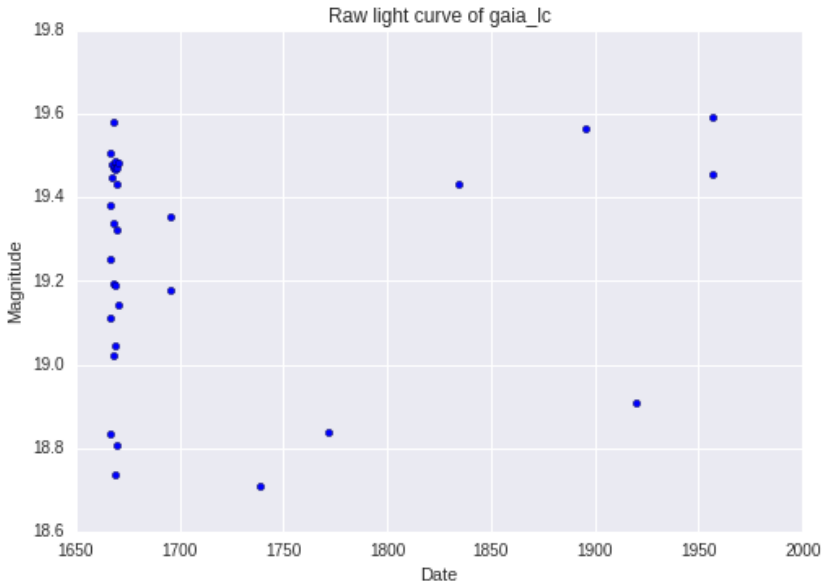
In [1]:
```
%matplotlib inline
import numpy as np
from gatspy.periodic import LombScargleFast
from scipy.optimize import curve_fit

import matplotlib.pyplot as plt
import seaborn
seaborn.set()

# GAIA source 4656702360931582080
star = "gaia_lc"

lightcurve = np.loadtxt(star+".dat")
plt.scatter(lightcurve[:,0],lightcurve[:,1])
plt.xlabel("Date")
plt.ylabel("Magnitude")
plt.title("Raw light curve of "+star)
```

Out[1]: <matplotlib.text.Text at 0x7fbb38629978>

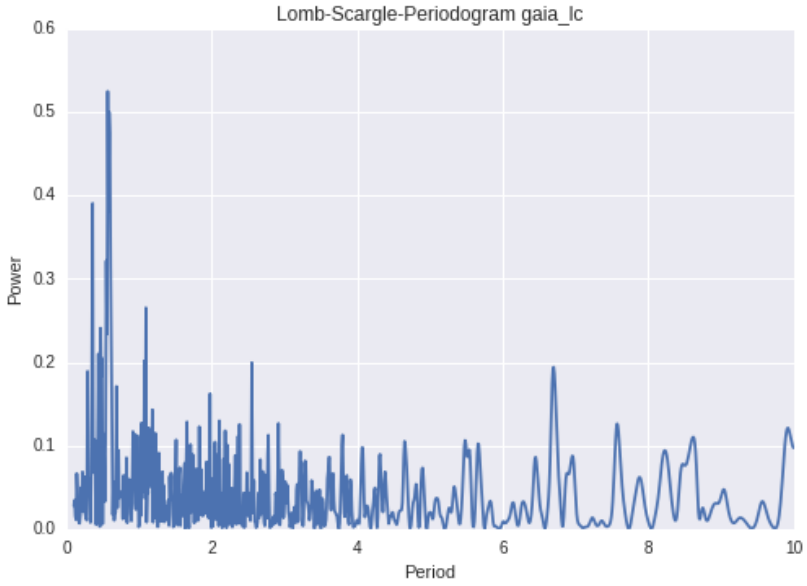


Calculate the Lomb-Scarge-Periodogram and find the most significant period

In [2]:
```
ls = LombScargleFast()
ls.optimizer.period_range = (0.1,10)
ls.fit(lightcurve[:,0],lightcurve[:,1])
period = ls.best_period
print("Best period: " + str(period) + " days")
# periodogramm
periods = np.linspace(0.1,10,1000)
scores = ls.score(periods)
plt.plot(periods,scores)
plt.xlabel("Period")
plt.ylabel("Power")
plt.title("Lomb-Scargle-Periodogram "+star)
```

```
Finding optimal frequency:
 - Estimated peak width = 0.0217
 - Using 5 steps per peak; omega_step = 0.00433
 - User-specified period range:  0.1 to 10
 - Computing periods at 14357 steps
Zooming-in on 5 candidate peaks:
 - Computing periods at 1000 steps
Best period: 0.567111700018 days
```

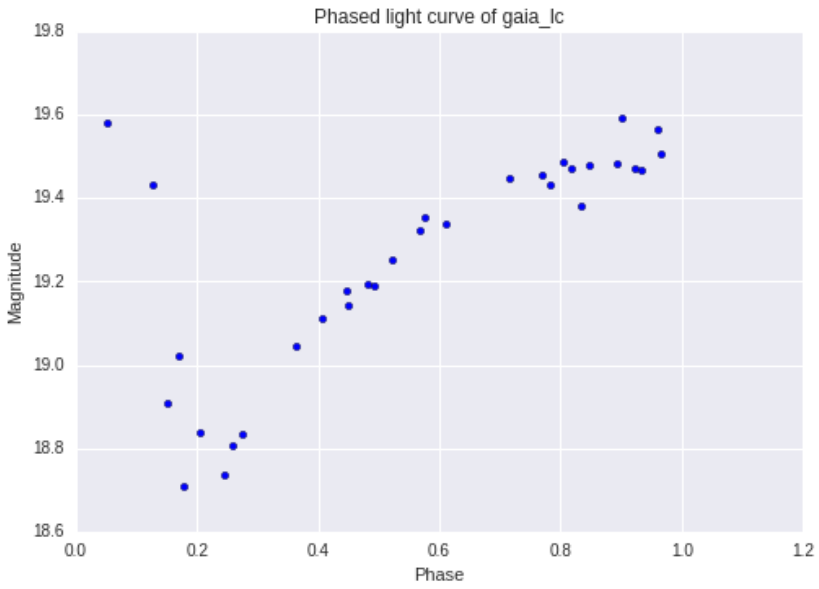
Out[2]: <matplotlib.text.Text at 0x7fbb38580c18>



Phased light curve

In [3]:
```
# phased light curve
#period = 0.566943974650772
phased_lc = np.zeros(shape=lightcurve.shape)
phased_lc[:,0] = np.fmod(lightcurve[:,0]/period,1)
phased_lc[:,1] = lightcurve[:,1]
#phased_lc[:,2] = lightcurve[:,2]
plt.scatter(phased_lc[:,0],phased_lc[:,1])
plt.xlabel("Phase")
plt.ylabel("Magnitude")
plt.title("Phased light curve of "+star)
```

Out[3]: <matplotlib.text.Text at 0x7fbb38566e80>



Calculate Fourier components by fitting a 4 mode Fourier series to the data

In [4]:
```
def fourier4(tau, c0, c1, phi1, c2, phi2, c3, phi3, c4, phi4):
  # tau = (2*pi*t)/P im Argument setzen
  return c0+c1*np.cos(tau+phi1)+c2*np.cos(2*tau+phi2)+c3*np.cos(3*tau+p
hi3)+c4*np.cos(4*tau+phi4)

def fit_fourier4(lc,period):
  phi_min = 0
  phi_max = 2*np.pi
  c_min = 0
  c_max = np.inf
  bound_min = [c_min,c_min,phi_min,c_min,phi_min,c_min,phi_min,c_min,ph
i_min]
  bound_max = [c_max,c_max,phi_max,c_max,phi_max,c_max,phi_max,c_max,ph
i_max]
  bound = (bound_min,bound_max)
  popt, pcov = curve_fit(fourier4, lc[:,0]*2*np.pi/period, lc[:,1], bou
nds=bound)
  c_0 = popt[0]
  c_1 = popt[1]
  c_2 = popt[3]
  c_3 = popt[5]
  c_4 = popt[7]
  phi_1 = popt[2]
  phi_2 = popt[4]
  phi_3 = popt[6]
  phi_4 = popt[8]
  # Hier werden beispielhaft R21 und phi21 zurueckgegeben
  return c_2/c_1, np.mod(phi_2 - 2*phi_1, 2*np.pi)

R21, phi21 = fit_fourier4(lightcurve,period)
print("R21: %f\nPhi21: %f" % (R21, phi21))
```

```
R21: 0.544417
Phi21: 4.071997
```

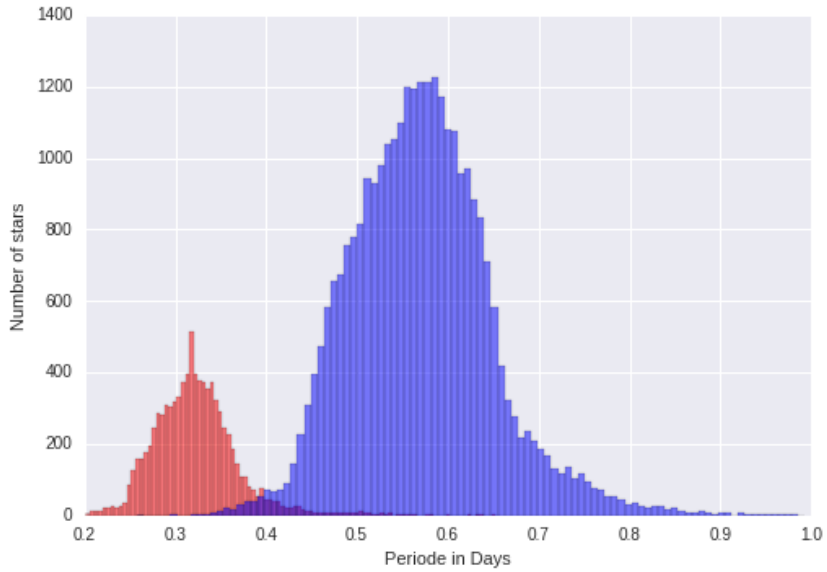
Now some classification

In [5]:
```
import pandas as pd
from sklearn.cluster import KMeans

# Load RR-Lyrae data from OGLE-III
datafile = "startable_rrlyrae.dat"
table = pd.read_csv(datafile,delimiter="\t",header=6,index_col=0)

# Define data set to be used (table from OGLE also contains info like
# positions we do not want to use)
#
# Replace the numerical value -99.99 with abstract value np.nan
#
# Remove rows containing NaN
data = table.query("A_1>0 and V>0 and I>0 and R21_1>0 and phi21_1>0")[[
"P_1","A_1","R21_1","phi21_1"]]

# Finally: Cluster using Fourier component R21_1 and period
prediction = KMeans(n_clusters=2,n_init=1000,init="random",random_state
=10).fit_predict(data[["P_1","R21_1"]])
# Create histogram plot
plt.hist((data["P_1"][prediction==0]),bins=100,color="red",alpha=0.5)
plt.hist((data["P_1"][prediction==1]),bins=100,color="blue",alpha=0.5)
plt.xlabel("Periode in Days")
plt.ylabel("Number of stars")
plt.savefig("ogle_period.png")
plt.show()
```



Important: This examples are missing (statistical) tests you should perform @serious analysis!

Thank you very much for your attention! Do you have any questions?

In [ ]: