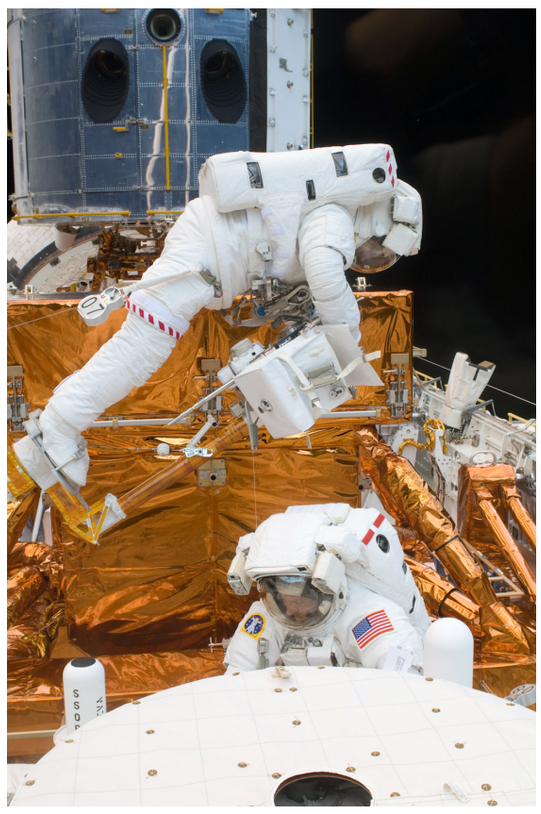
LAB 3
Cleaning HST Data

GU4303: Astrostatistics

Quick background...

2.4m UV-NIR telescope launched in 1990

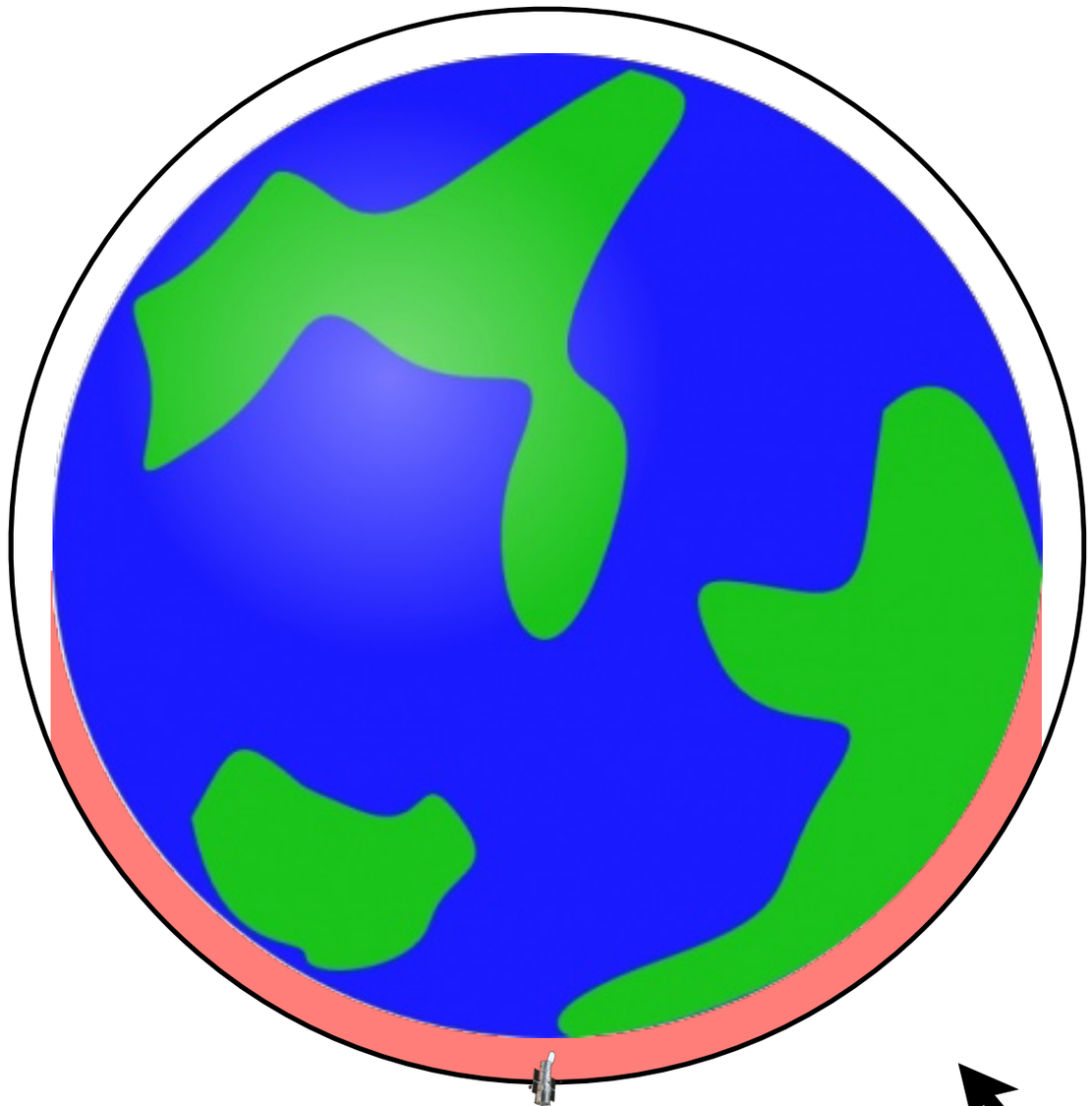
Mirror fixed in 1993, in first of five service missions



Last service mission in 2009, orbit will decay in~2030

Low Earth Orbit (LEO), once every 95.48 minutes

direction of target star



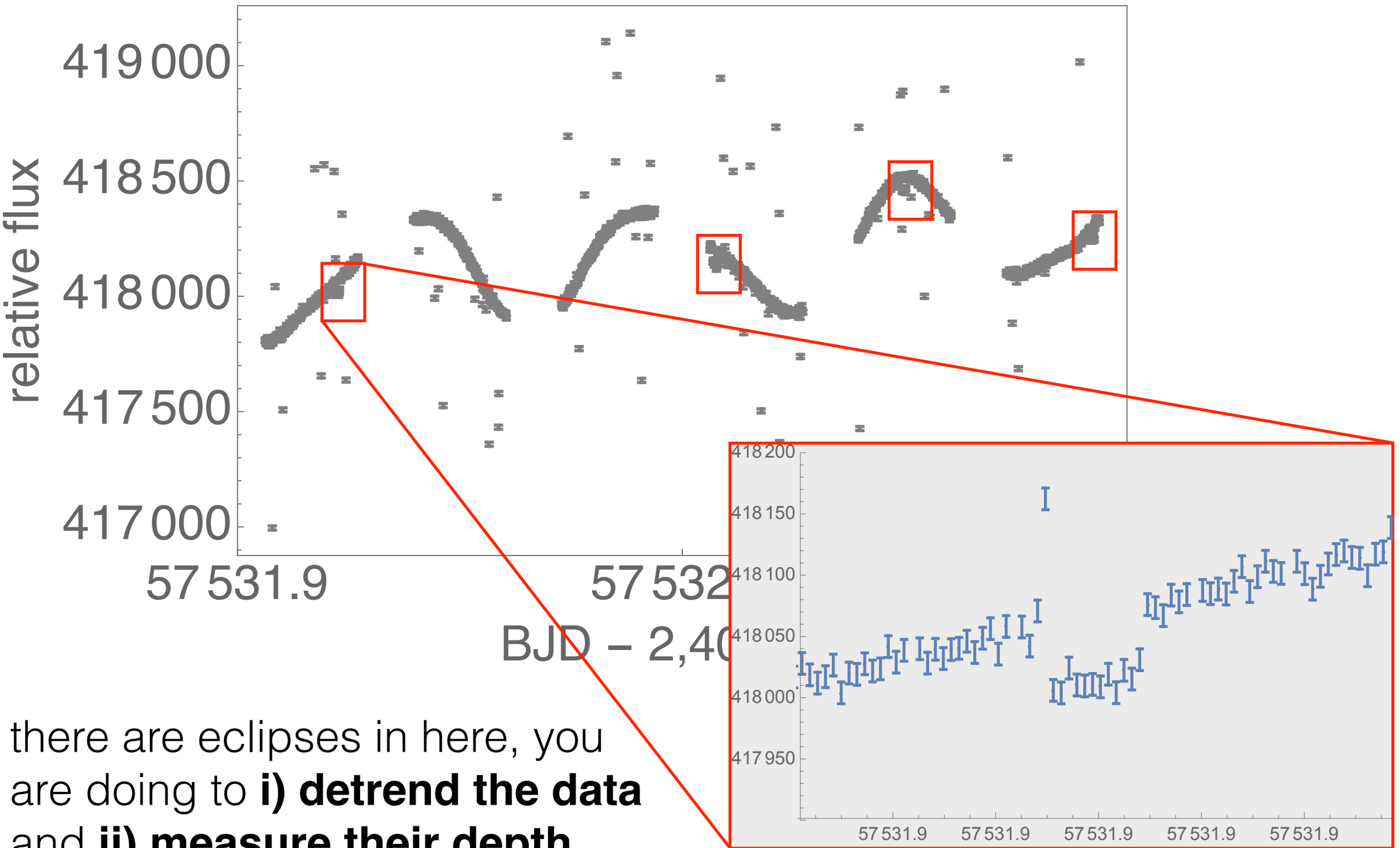
for most stars, observations are interrupted once every 95.48 mins => sparse data

(changing Sun light also heats and cools HST => phase-correlated noise)

we will ignore this!

Sun light

even though photon noise is very low (thanks 2.4m aperture), there are plenty of trends/affects to clean



there are eclipses in here, you are doing to **i) detrend the data** and **ii) measure their depth**

but, you, like a real astronomy, know some things about the effects in play...

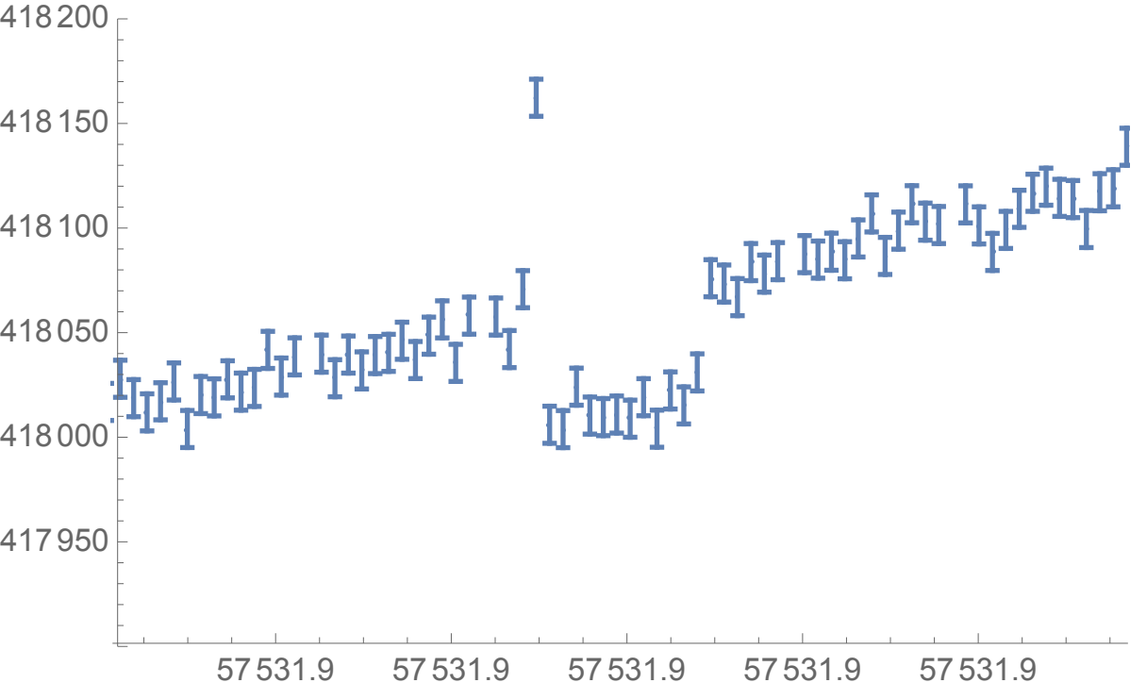
- photon noise (reported error bars)

these need to be removed

- overall slow linear trend in time
- sinusoidal variation of the star of order hours timescale
- outliers due to cosmic rays
- expected times and duration of the eclipses

you need to correct for these three effects to measure the depth, prize for the winner who comes closest to the true solution!

other information for your consideration...



exposures = 13.2 seconds
P(HST) = 95.48 mins
start of observations = 57531.87312
exactly 6 orbits of data

duration of eclipses = 166.484 seconds
central time of first transit = 57531.91792371155
period of eclipses = 2.025026511149096 hours

data will be...
{time, flux, error on flux}

YOUR TASK: Use this data set to measure the depth of the eclipses

1] Remove outliers (think carefully about bandwidth!)

2] Choose a detrending method (global parametric vs local parametric vs non-parametric) and apply it to the time series

3] Measure the depth **and error** using the weighted average

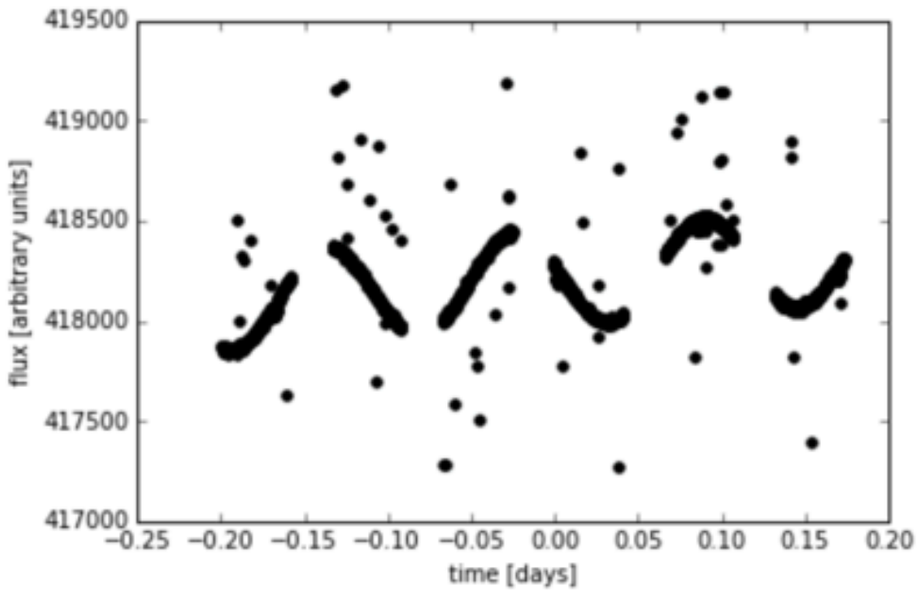
4a] If you used parametric, try propagating the uncertainty of your detrending into your depth estimate via MCMC noise model fit

4b] If you used non-parametric, experiment with different bandwidths and window to test robustness of result

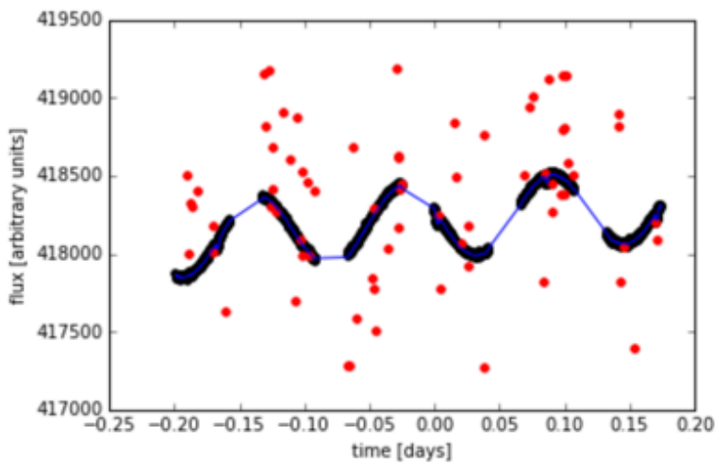
5] Make some nice plots showing the i) quality of your detrending ii) RMS vs bin size plot on residuals iii) weighted residuals histogram

6] Write up your slides

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import ascii

[redacted]
tpivot = np.median(data[0][:])    # pivot point, which i'll use later
data = np.stack((dataraw[0][:]-tpivot,dataraw[1][:],dataraw[2][:]))

time=data[0][:]     # Time [d]
flux=data[1][:]     # Flux [arbitrary units]
fluxerr=data[2][:]  # Error on flux [arbitrary units]

# Some information we know
mins = 1.0/(24.0*60.0)       # one minute [days]
Phst = 95.48*mins            # period of HST
dur = 166.484/86400.0        # duration of the eclipse [days]
t0 = 57531.91792371155       # mid-time of first eclipse
Ptran = 0.08437610463121233 # period of eclipses [days]
cadence = 13.2/86400.0       # cadence [days]

%matplotlib inline

plt.scatter(time, flux, color='k')
plt.xlabel('time [days]')
plt.ylabel('flux [arbitrary units]')
plt.show()
```



In [4]:
```
# TASK 1: REMOVE OUTLIERS

# Define a moving median function
def movingmedian(x,window):
[redacted]

# Calculate moving medians
smoothwindow = 11
timesmooth = movingmedian(time,smoothwindow)
fluxsmooth = movingmedian(flux,smoothwindow)

# Append the edges of the windows
[redacted]

# Split data into good vs bad
sigmas = 3
[redacted]
data2=data2[2:,:]
baddata=baddata[2:,:]

%matplotlib inline

plt.scatter(data2[:,0], data2[:,1], color='k')
plt.plot(time, fluxpred, color='b')
plt.scatter(baddata[:,0], baddata[:,1], color='r')
plt.xlabel('time [days]')
plt.ylabel('flux [arbitrary units]')
plt.show()
print 100.0*len(baddata)/len(flux),'% of the original data is an outlier'
```

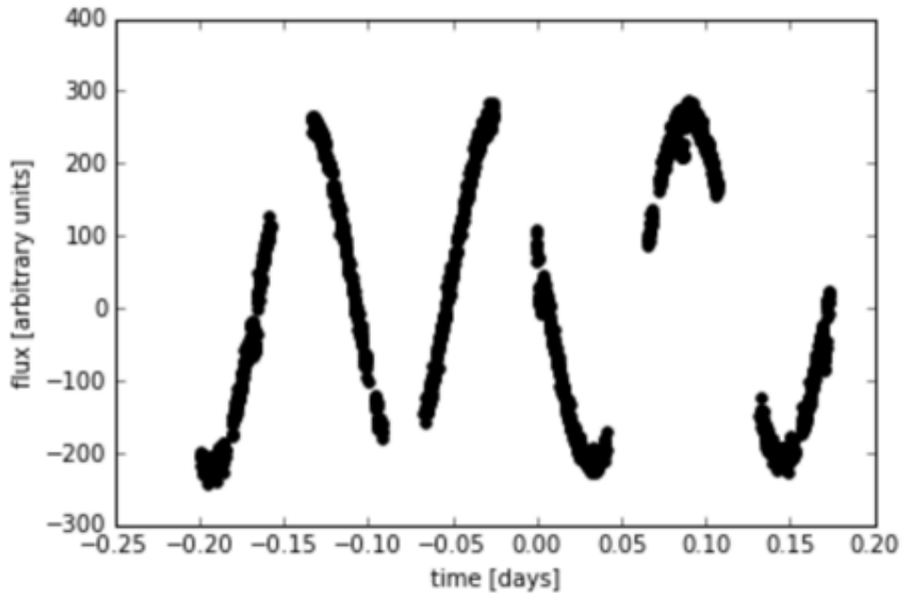



```
4.48877805486 % of the original data is an outlier
```

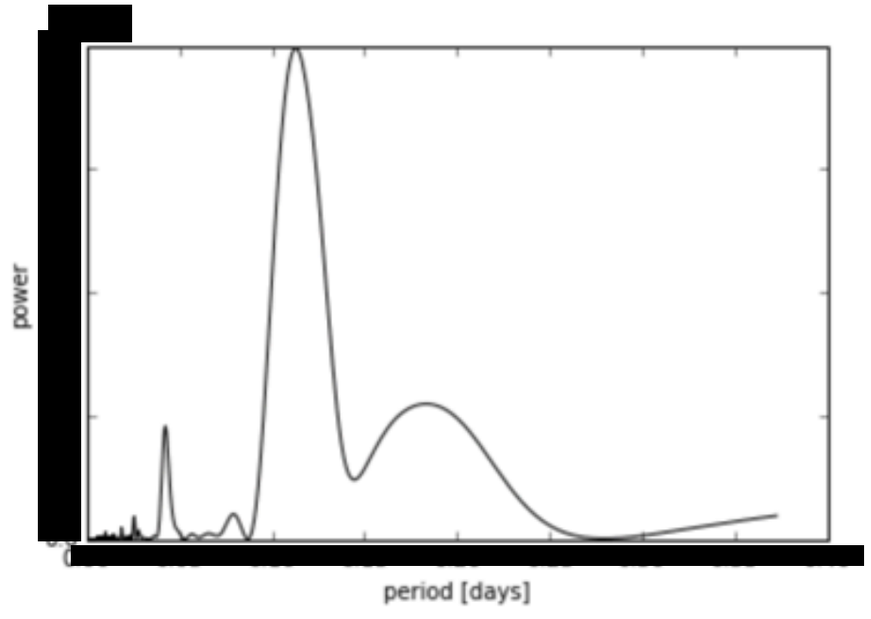
In [5]:
```
# TASK 2: FIND THE SINUSOIDAL SIGNAL's PERIOD
# Task 2a: Remove the transits first, which could mess things up
```

[redacted]

```
# Task 2b: Remove a linear trend first, before passing through to periodogram
tempslope = np.polyfit(data3[:,0], data3[:,1], 1)
periodogramdata = data3[:,1] - ( tempslope[1] + tempslope[0]*data3[:,0] )

%matplotlib inline

plt.scatter(data3[:,0], periodogramdata, color='k')
plt.xlabel('time [days]')
plt.ylabel('flux [arbitrary units]')
plt.show()
```

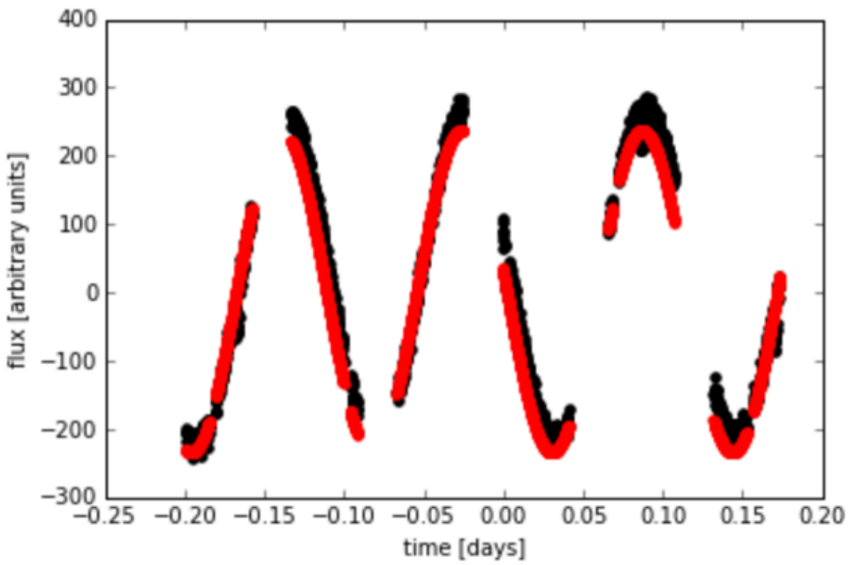


In [6]:
```
# Task 2c: Now run a periodogram through it
import scipy.signal as signal

nfreq = 50000
[redacted]
pgram = signal.lombscargle(data3[:,0], periodogramdata, freqs)
Pbest = periods[np.argmax(pgram)]
# periodogram has a peak at (A**2) * N/4, according to scipy dcoumentation, so
Abest = np.sqrt(4.0*pgram[np.argmax(pgram)]/len(data3[:,0]))
[redacted]
```



best period = [redacted]
best amp = [redacted]

In [8]:
```
# TASK 3: REMOVE THE SINUSOID + TREND
# Task 3a: Start by plotting our the vanilla sinusoid and manually tune the phase

fittedsinudoid = Abest*np.sin( 2.0*np.pi*data3[:,0]/Pbest + 3.0 )

%matplotlib inline

plt.scatter(data3[:,0], periodogramdata, color='k')
plt.scatter(data3[:,0], fittedsinudoid, color='r')
plt.xlabel('time [days]')
plt.ylabel('flux [arbitrary units]')
plt.show()
```



A much better solution is to write your own LS periodogram, which would yield the phase + could even codify linear trend!

$$f(A,B,C,D;t) = A + B*t + C*\sin(\omega t) + D*\cos(\omega t)$$

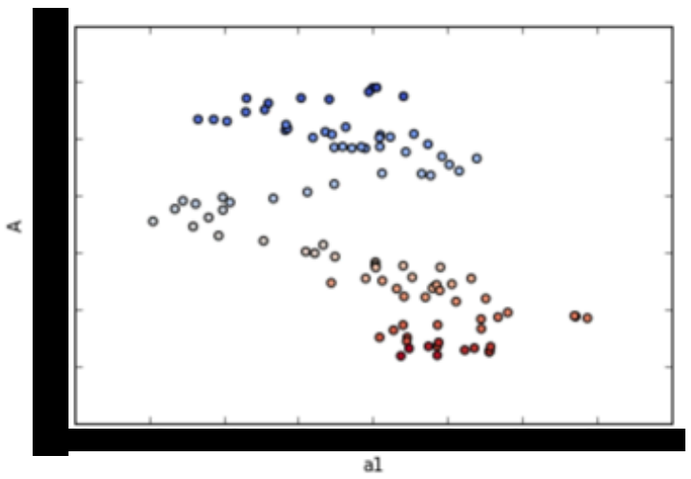
In [9]:
```
# TASK 3: REMOVE THE SINUSOID + A LINEAR TREND TOGETHER
# I'm going to do using the MCMC fitter we made last time

time=data3[:,0]     # Time [d]
flux=data3[:,1]     # Flux [arbitrary units]
fluxerr=data3[:,2]  # Error on flux [arbitrary units]

# Define the model
def model(t,theta):
    [redacted]
    return modflux

# Set a definition for the loglikelihood, assuming normally distributed data
def log_like(t,f,ferr,theta):
    [redacted]
    return loglike

# Set an initial chain point, seed from a "reasonable" solution
[redacted]

# Calculate the associated modified loglike
[redacted]
```

A better solution would be to use an optimize function here, since MCMC is not designed to aggressively find maximal likelihood

In [10]:
```
# Define the proposal jump size
a0jump = 0.1*np.median(fluxerr)
a1jump = 0.1*np.median(fluxerr)/(np.amax(time) - np.amin(time))
Ajump =
Pjump =
phijump
thetajump=np.array([a0jump,a1jump,\
                    Ajump,Pjump,phijump])

# Starting walking
j=0
jmax=100
while True:

    # Generate a proposal (or jump)


    # Compute Metropolis Rule


    if j==jmax:
        break
```

short run

In [11]:
```
%matplotlib inline

jlist=np.arange(len(thetachain))
plt.scatter(thetachain[:,1], thetachain[:,2], c=jlist, cmap='coolwarm')
plt.xlabel('a1')
plt.ylabel('A')
plt.show()
```



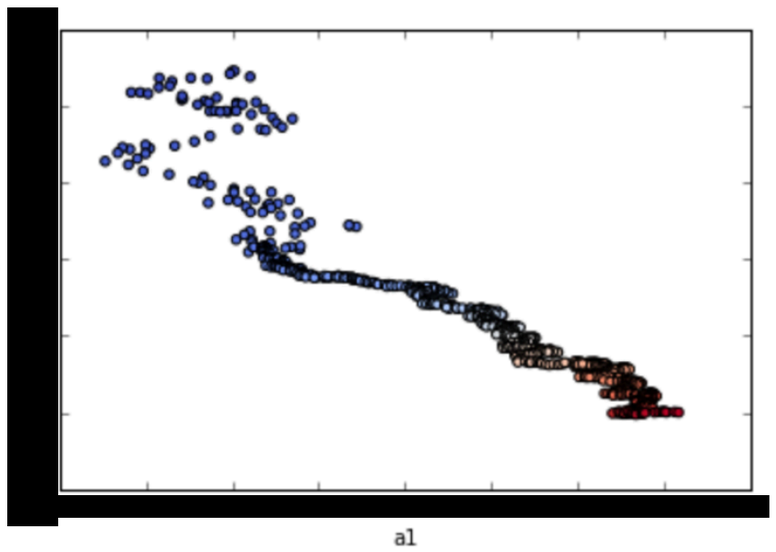
In [12]:
```
# See what the standard deviation of the second half of the initial MCMC chain was...
thetavar=[np.std(thetachain[50:jmax,0]),np.std(thetachain[50:jmax,1]),np.std(thetachain[50:jmax,2]),\
np.std(thetachain[50:jmax,3]),np.std(thetachain[50:jmax,4])]
print 'Check that polynomial coeff jump params are ~less than stdevs...'
print thetavar[0],thetavar[1]
print thetajump[0],thetajump[1]
print ' '
print 'Check that sinusoidal coeff jump params are ~less than stdevs...'
print thetavar[2],thetavar[3],thetavar[4]
print thetajump[2],thetajump[3],thetajump[4]
```

```
Check that polynomial coeff jump params are ~less than stdevs...
5.69173303923 5.26877272037
0.8864388 2.37792822655

Check that sinusoidal coeff jump params are ~less than stdevs...
6.36045808499 0.0020926678203 0.0653913740347
0.8864388 0.000566235551288 0.00628318530718
```

In [13]:
```
# Restart the chain, but modify the jump size
[redacted]

# Starting walking
j=100
jmax=1000
while True:

    # Generate a proposal (or jump)
    [redacted]

    # Compute Metropolis Rule
    [redacted]

    if j==jmax:
        break
```

medium run

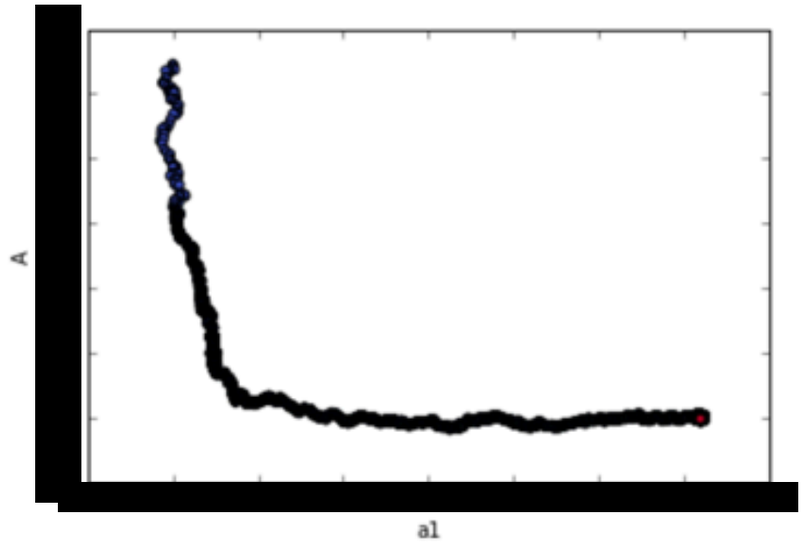
```
%matplotlib inline

jlist=np.arange(len(thetachain))
plt.scatter(thetachain[:,1], thetachain[:,2], c=jlist, cmap='co
plt.xlabel('a1')
plt.ylabel('A')
plt.show()
```



In [15]:
```
# Starting walking
j=1000
jmax=10000
while True:

    # Generate a proposal (or jump)
    [redacted]

    # Compute Metropolis Rule
    [redacted]

    if j==jmax:
        break
```

long run

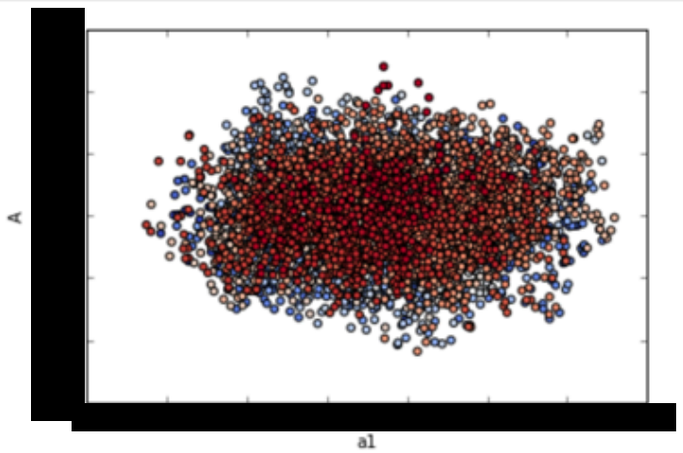
In [16]:
```
%matplotlib inline

jlist=np.arange(len(thetachain))
plt.scatter(thetachain[:,1], thetachain[:,2], c=jlist, cmap='coolwarm')
plt.xlabel('a1')
plt.ylabel('A')
plt.show()
```



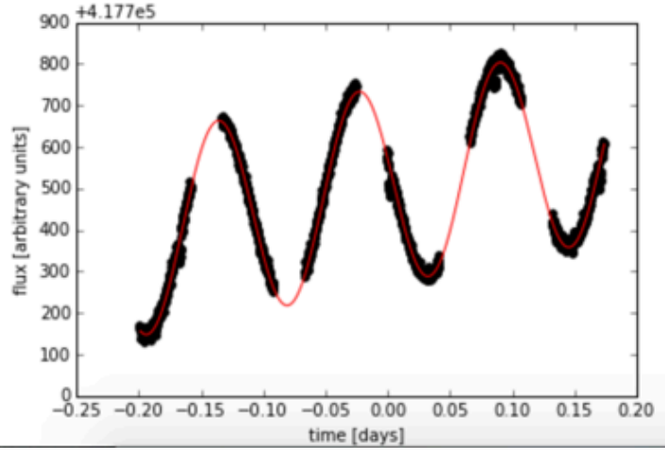
In [17]:
```
loglikeburn=np.median(loglikechain)
j=-1
while True:
    j=j+1
    if loglikechain[j] > loglikeburn:
        break
burnj=j
print 'Burn point = ',burnj
```

```
Burn point =  3068
```

In [18]:
```
%matplotlib inline

thetachainburnt=thetachain[burnj:,:]
loglikechainburnt=loglikechain[burnj:]

jlist=np.arange(len(thetachainburnt))
plt.scatter(thetachainburnt[:,1], thetachainburnt[:,2], c=jlist, cmap='coolwarm')
plt.xlabel('a1')
plt.ylabel('A')
plt.show()
```

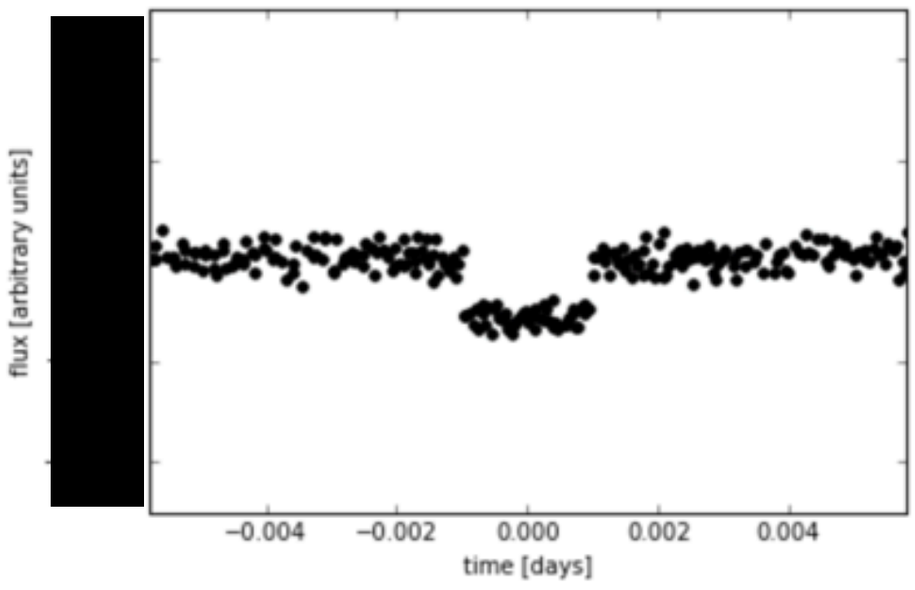


In [19]:
```
thetabest=thetachainburnt[np.argmax(loglikechainburnt),:]
densetimes=np.arange(data2[0,0],data2[len(data2)-1,0],cadence)
densemodel=model(densetimes,thetabest)

%matplotlib inline

plt.scatter(data2[:,0], data2[:,1], color='k')
plt.plot(densetimes, densemodel, color='r')
plt.xlabel('time [days]')
plt.ylabel('flux [arbitrary units]')
plt.show()
```



a plot proving my detrending is good

In [20]:
```
# TASK 4: NOW CO-ADD THE TRANSITS TO GET DEPTH

[redacted]
for i in range(len(time)):
    timefold[i] = time[i] - (t0-tpivot) - Ptran*round( ( time[i] - (t0-tpivot) )/Ptran )

%matplotlib inline

plt.scatter(timefold, norm, color='k')
plt.axis([-3.0*dur, 3.0*dur, -0.0005,0.0005])
plt.xlabel('time [days]')
plt.ylabel('flux [arbitrary units]')
plt.show()
```



Make you sure you a) fold b) divide data by detrending function [not subtract!]

In [21]:
```
[redacted]
Depth = [redacted] +/- [redacted]
```