In [83]:
```
#Stars and stuff
from astroquery.gaia import Gaia
from astropy.io.votable import parse, parse_single_table
from astropy.timeseries import LombScargle
import pandas as pd

#Math
import numpy as np
from scipy.linalg import lstsq
import scipy as sp
import pymc3 as pm
import theano.tensor as tt

#Machine Learning
from sklearn.model_selection import train_test_split

#Plotting
import matplotlib.pyplot as plt
from corner import corner

#Caching
from joblib import Memory
location = "./cachedir"
memory = Memory(location, verbose=0)

#File manipulation
from urllib.request import urlopen
import io
from glob import glob
import os
import sys
```

In [4]:
```
def get_gaia_query_rrlyrae(num_stars = 100, num_clean_epochs = 40, cond
s=None, verbose=False):
    add = ""
    if conds is not None:
        add = f"AND {conds}"
    query = f'''
        SELECT TOP {num_stars} *
        FROM gaiadr2.gaia_source as gaia
        JOIN gaiadr2.vari_rrlyrae using (source_id)
        WHERE
            num_clean_epochs_g > {num_clean_epochs}
    ''' + add
    if verbose:
        print(query)
    job = Gaia.launch_job_async(query)
    return job.get_results()

get_gaia_query_rrlyrae_cached = memory.cache(get_gaia_query_rrlyrae)
```

In [5]:
```
def get_gaia_query_general(query):
    job = Gaia.launch_job_async(query)
    return job.get_results()
get_gaia_query_general_cached = memory.cache(get_gaia_query_general)
```

In [6]:
```
def rmse(a, b):
    return np.sqrt(np.mean(np.square(a-b)))
```

In [7]:
```
def mse(a,b):
    return np.mean(np.square(a-b))
```

In [8]:
```
def get_Gband(data,index=None,source_id=None):
    assert index!=None or source_id!=None, "Must pass in either index o
r source_id"
    if index!=None:
        if source_id!=None:
            print(f"Using index: {index} instead of source_id: {source_
id}")
        selected_row=data[index]
        source_id=selected_row['source_id']
        print(f"Analyzing star with source_id: {source_id}")
    if sys.platform=="linux":
        dirname = "lightcurve_xmls_win"
    else:
        dirname = "lightcurve_xmls_mac"
    lc_f = f"{dirname}/{source_id}.xml"
    if os.path.exists(lc_f):
        votable = parse_single_table(lc_f)
    else:
        url = selected_row['epoch_photometry_url']
        if type(url) == bytes:
            url = url.decode('utf-8')
        votable = parse(url)
        votable.format = 'binary'
        with open(lc_f, 'w') as d:
            votable.to_xml(d)
        votable = parse_single_table(lc_f)
    Gstring = "G" if sys.platform=="linux" else b'G'
    Gband = votable.array[votable.array['band'] == Gstring]
    return Gband
```

In [9]:
```
def get_maxmin_freqs(data):
    if any([type(x)==np.ma.core.MaskedConstant for x in data['pf']]):
        col_name = 'p1_o'
    else:
        col_name = 'pf'
    max_freq, min_freq = 1/np.min(data[col_name]), 1/np.max(data[col_na
me])
    return max_freq, min_freq
```

In [10]:
```
def estimate_period(Gband, max_freq=2.465, min_freq=1.044, p=False):
    mag = Gband['mag']
    flux = Gband['flux']
    time = Gband['time']
    flux_err = Gband['flux_error']

    freq, power = LombScargle(time, flux, flux_err).autopower(maximum_f
requency=max_freq,
                                                              minimum_f
requency=min_freq,
                                                              nyquist_f
actor=100)
    period = 1/freq[np.argmax(power)]

    phase = time % period
    if p:
        plt.figure(figsize=(8,5))
        plt.plot(freq, power, '-k')
        plt.xlabel("Frequency")
        plt.ylabel("Spectral Power")

        fig, (ax1,ax2) = plt.subplots(1,2, figsize=(16,5))

        ax1.scatter(phase, flux)
        ax1.set(xlabel="Phase", ylabel="Flux")
        ax1.grid()

        ax2.scatter(phase, mag)
        ax2.set(xlabel="Magnitude", ylabel="Flux")
        ax2.grid()

        plt.show()

        print(f"Estimated period: 1/{freq[np.argmax(power)]:.5f} = {per
iod:.5f}")
#         print(f"Period as reported by vari_rrlyrae: {recorded_perio
d:.5f}")
#         print(f"RMSE: {np.sqrt(np.mean(np.square(period - recorded_pe
riod)))}")

    return period
```

In [11]:
```
def plot_magnitude(Gband):
    time = Gband['time']
    mag = Gband['mag']
    mag_uncertainty = 1.09/Gband['flux_over_error']

    plt.figure(figsize=(8,5))
    plt.title("Magnitude w/ Magnitude Uncertainty")
    plt.fill_between(time, mag+mag_uncertainty/2,mag-mag_uncertainty/2,
color='blue', alpha=0.5)
    plt.xlabel("Time")
    plt.ylabel("Magnitude")
    plt.grid()
    plt.show()
    print(f"Estimated mean: {np.log(np.average(np.exp(mag)))}")
```

In [12]:
```
def setup_fit(flux, time, omega, k):
    num_samps = len(time)
    k_s = np.arange(1,k+1)
    tk_tiling = np.outer(time, k_s)*omega
    X = np.zeros((num_samps, 2*k+1))
    X[:,0] = np.ones(num_samps)
    X[:,1:k+1] = np.sin(tk_tiling)
    X[:,k+1:] = np.cos(tk_tiling)
    return X
```

In [13]:
```
def pseudo_fourier(omega, t, A0, a, b):
    assert len(a) == len(b), f"Length of a and length of b must be the
same"
    K = len(a)
    s,c = np.zeros(len(t)), np.zeros(len(t))
    for k in range(1, K+1):
        s += a[k-1]*np.sin(k*omega*t)
        c += b[k-1]*np.cos(k*omega*t)
    return A0 + s + c
```

In [14]:
```
def get_prediction(Gband, max_freq, min_freq, k, p=False):

    flux,time=Gband['flux'], Gband['time']
    source_id=Gband['source_id'][0]
    period=estimate_period(Gband, max_freq=max_freq, min_freq=min_freq)
    omega=2*np.pi/period

    X = setup_fit(flux, time, omega, k)
    beta = lstsq(X, flux.data)[0]

    A0=beta[0]
    a=beta[1:k+1]
    b=beta[k+1:]

    time_interp=np.arange(np.min(time), np.max(time), 1)
    f_interp = pseudo_fourier(omega=2*np.pi/period, t=time_interp, A0=A
0, a=a, b=b)
    phase_interp, flux_interp = sort_by_phase(time_interp%period, f_int
erp)

    if p:
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(16,
7), gridspec_kw={'height_ratios':[3,1]})
        ax1.scatter(time%period, flux, label="Original",color='black')
        ax1.scatter(time%period, X.dot(beta), label="Predicted", color=
'green')
        ax1.set(title=f"SourceID: {source_id}\nRMSE: {rmse(flux, X.dot
(beta)):.5f}",ylabel="Flux")
        ax1.legend()
        ax1.grid()

        ax3.scatter(time%period, flux-X.dot(beta), label="Residual",col
or='red')
        ax3.set(xlabel="Phase")
        ax3.grid()

        ax2.scatter(time%period, flux, label="Original f")
        ax2.plot(phase_interp, flux_interp, label="Predicted f", color
="red")
        ax2.set(title="Interpolated Values", xlabel="Phase",ylabel="Flu
x")
        ax2.legend()
        ax2.grid()

        fig.delaxes(ax4)

        plt.show()
```

In [15]:
```
def sort_by_phase(phase, x):
    sorted_phase, sorted_x = np.array(sorted(zip(phase, x), key=lambda
pair: pair[0])).T
    return sorted_phase, sorted_x
```

In [16]:
```
def sampler(func, num_samps=10000):
    x = np.zeros(num_samps)
    x[0] = np.random.randn()
    for num_samp in range(1, num_samps):
        curr = x[num_samp-1]
        nxt = curr + np.random.randn()
        r = func(nxt)/func(curr)
        p = min(r, 1)
        if np.random.binomial(1,p):
            x[num_samp] = nxt
        else:
            x[num_samp] = curr
    return x
```

In [17]:
```
gauss=lambda x:1/(2*np.pi*(0.1**2))**0.5*np.exp(-(x-1)**2/(2*(0.1**2)))
```

In [61]:
```
def likelihood(d, a, b, s):
    Mg = a*np.log(d["pf"])+b
    pdf_Mg = sp.stats.norm.pdf(Mg, loc=d["Mg"], scale=s)
    return pdf_Mg

def prior(a, b, s):
    pdf_a = sp.stats.norm.pdf(a, loc=0,scale=2)
    pdf_b = sp.stats.norm.pdf(b, loc=0, scale=2)
    pdf_s = sp.stats.norm.pdf(s, loc=0, scale=2)
    return pdf_a*pdf_b*pdf_s

def posterior(d, a, b, s):
    p = np.nansum(likelihood(d, a, b, s)*prior(a,b,s))
    return p
```

In [19]:
```
def sampler_data(data, func, num_samps=10000):
    x = np.zeros((3,num_samps))
    x[:,0] = np.random.randn(3)
    for num_samp in range(1, num_samps):
        curr = x[:, num_samp - 1]
        nxt = curr + np.random.randn(3)
        r = func(data, *nxt)/func(data, *curr)
        p = min(r, 1)
        if np.random.binomial(1,p):
            x[:, num_samp] = nxt
        else:
            x[:, num_samp] = curr
    return x
```

In [124]:
```
def run_mcmc(x, y, yerr):
    # set up the model
    with pm.Model() as model:

        # define priors
        a = pm.Uniform("a", lower=-13, upper=7)
        b = pm.Uniform("b", lower=-11, upper=9)
        logsig = pm.Uniform("logsig", lower=-10, upper=10)

        sig_s = tt.power(tt.exp(logsig), 2)
        sig_i = tt.power(yerr, 2)

        big_sig = tt.sqrt(sig_s + sig_i)
        mu = a*np.log(x)+b

        # define the log-likelihood function
        # note that numpy doesn't play nicely with PyMC3, so you shoul
d use their built in math functions
        pm.Normal("obs", mu = mu, sd=big_sig, observed=y)

        # now set up the model to run
        # default of PyMC3 is to use the no-turn sampler (NUTS)

        # pm.sample will run the sampler and store output in 'trace'
        trace = pm.sample(draws=1000, tune=1000, chains=2, cores=2)

        # traceplot is a routine for plotting the 'traces' from the sa
mples
        _ = pm.traceplot(trace, var_names=["a", "b", "logsig"])

        # pm.summary provides some useful summary and convergance stat
istics
        pm.summary(trace, var_names=["a", "b", "logsig"])
        # translate trace into pandas dataframe for plotting (you can
also plot numpy arrays with corner)
        samples = pm.trace_to_dataframe(trace, varnames=["a", "b", "lo
gsig"])

        # make the corner plot and plot results from Hubble's paper as
'truth'
        # overplot percentiles: 16, 50, 84 on 1d historgrams
        corner(samples)
    return samples
```

In [20]:
```
pf_cond = "pf IS NOT NULL"
rrlyrae_100 = get_gaia_query_rrlyrae_cached(num_stars = 100, num_clean_
epochs=40, conds=pf_cond)
rrlyrae_100[:10]
```

Out[20]: *Table length=10*

solution_id	designation	random_index	ref_epoch	ra	
			yr	deg	
int64	object	int64	float64	float64	
1635721458409799680	Gaia DR2 5866125710834119808	841033097	2015.5	212.93756378519396	1
1635721458409799680	Gaia DR2 5978435871487788288	1394969254	2015.5	256.52946118354015	0.
1635721458409799680	Gaia DR2 5704736782734774528	122877659	2015.5	132.81229544277778	0.
1635721458409799680	Gaia DR2 5816755332315333888	259791940	2015.5	254.94654730343584	0.
1635721458409799680	Gaia DR2 5821611776409134976	299065082	2015.5	246.3262948830741	0.0
1635721458409799680	Gaia DR2 5642603243216872576	1311594757	2015.5	132.9564341215911	0.
1635721458409799680	Gaia DR2 5813181197970338560	1546661016	2015.5	263.35817148226175	0.
1635721458409799680	Gaia DR2 5630421856972980224	923739124	2015.5	140.99364763000955	0.0
1635721458409799680	Gaia DR2 5810405553887250432	191264318	2015.5	268.50110356278077	0.0
1635721458409799680	Gaia DR2 5821156028840408576	1453327615	2015.5	244.35278635111487	0.0

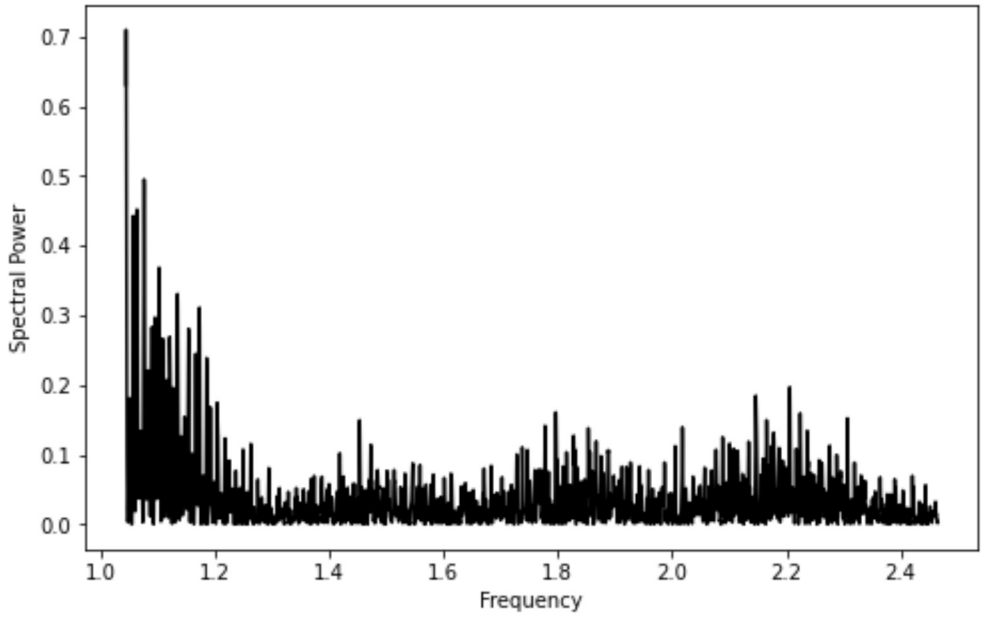
In [21]:
```
for row in rrlyrae_100:
    url = row['epoch_photometry_url'] if sys.platform=="linux" else row
['epoch_photometry_url'].decode('utf-8')
    source_id = row['source_id']
    if sys.platform=="linux":
        dirname = "lightcurve_xmls_win"
    else:
        dirname = "lightcurve_xmls_mac"
    dest = f"{dirname}/{source_id}.xml"
    if os.path.exists(dest):
#         print("It exists!")
        continue
    votable = parse(url)
    votable.format = 'binary'
    with open(dest, 'w') as d:
        votable.to_xml(d)
```

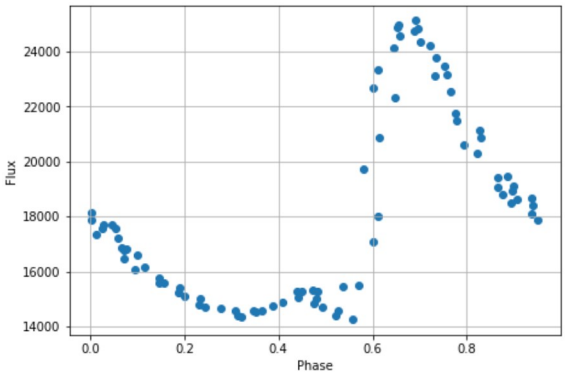
5813181197970338560

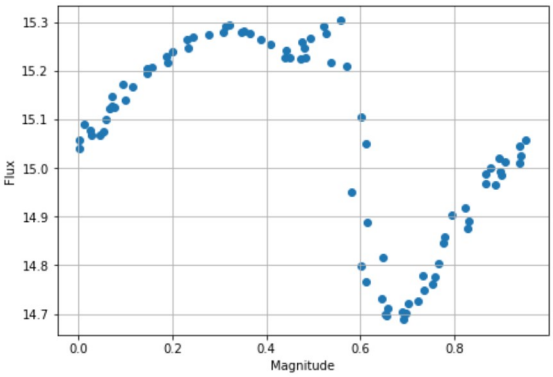
In [22]:
```
selected_index = 6
selected_Gband = get_Gband(rrlyrae_100, selected_index)
```

```
Analyzing star with source_id: 5813181197970338560
```

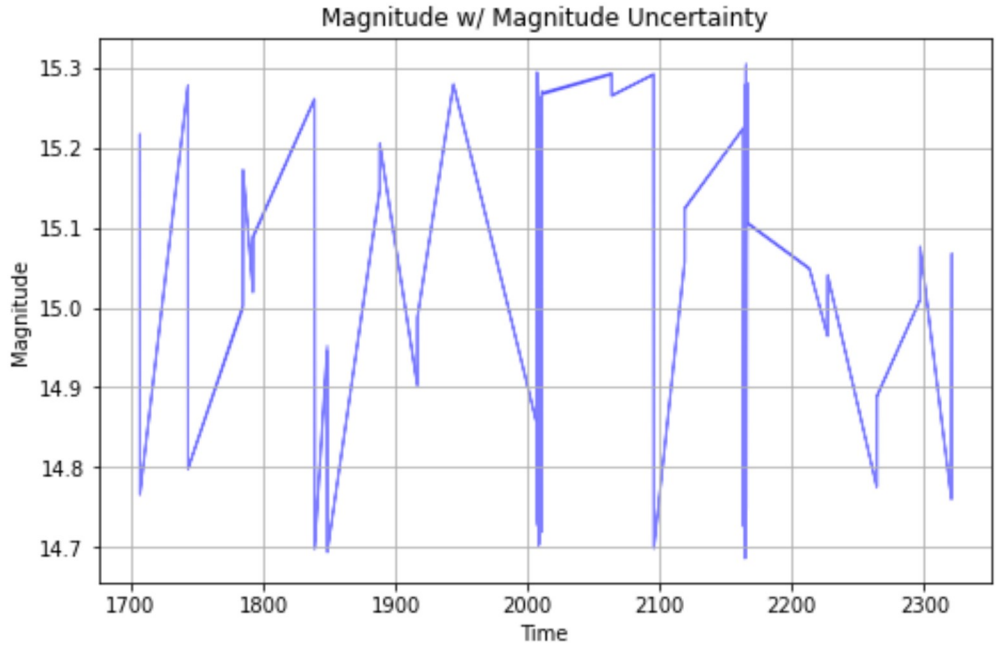
In [23]:
```
period = estimate_period(selected_Gband, p=True)
recorded_period = rrlyrae_100['pf'][selected_index]
print(f"Period as reported by vari_rrlyrae: {recorded_period:.5f}")
print(f"RMSE: {np.sqrt(np.mean(np.square(period - recorded_period)))}")
plot_magnitude(selected_Gband)
```







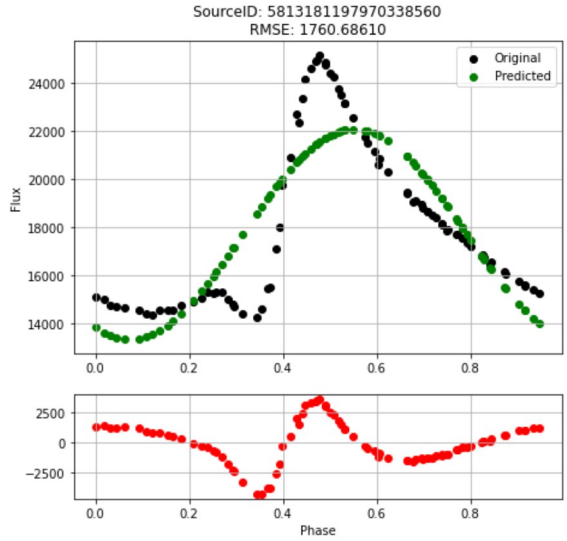

```
Estimated period: 1/1.04433 = 0.95756
Period as reported by vari_rrlyrae: 0.95765
RMSE: 9.488197693585665e-05
```

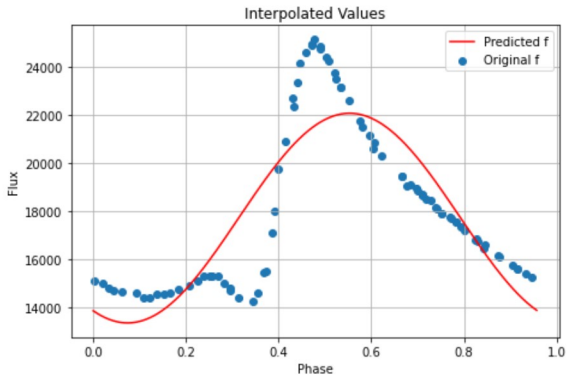


```
Estimated mean: 15.070074092017139
```

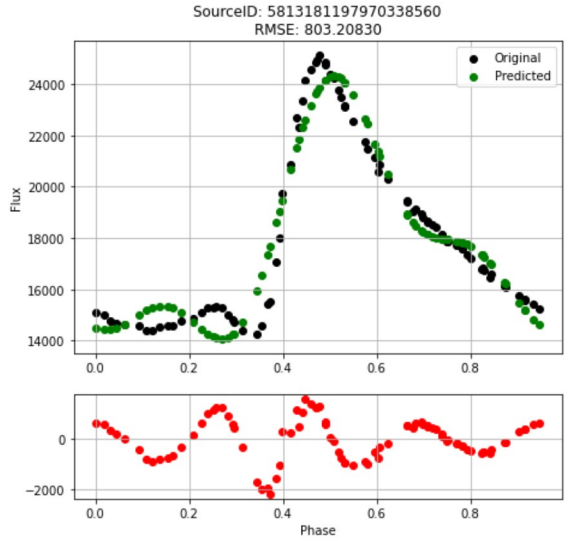
In [24]:
```
k_vals = [1,3,5,7,9]
max_freq, min_freq = get_maxmin_freqs(rrlyrae_100)
for k in k_vals:
    print(f"Analyzing fit for k={k}...")

    get_prediction(selected_Gband, max_freq=max_freq, min_freq=min_fre
q, k=k, p=True)
```

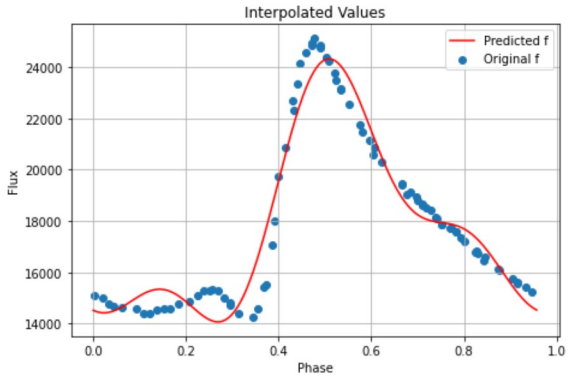
```
Analyzing fit for k=1...
```



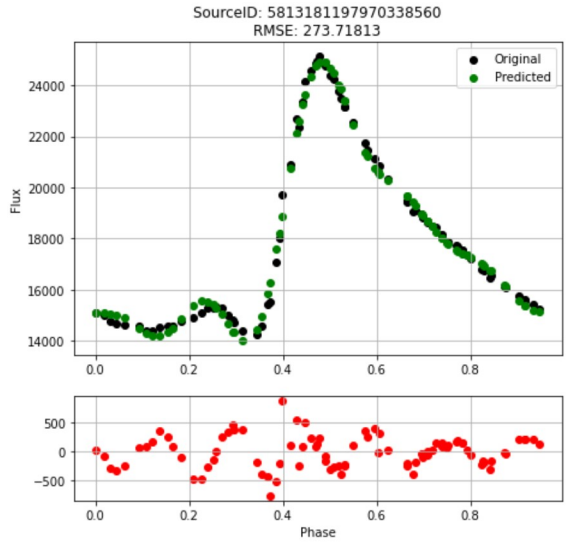


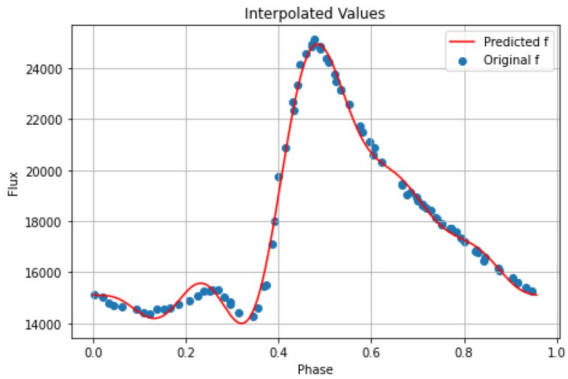

```
Analyzing fit for k=3...
```



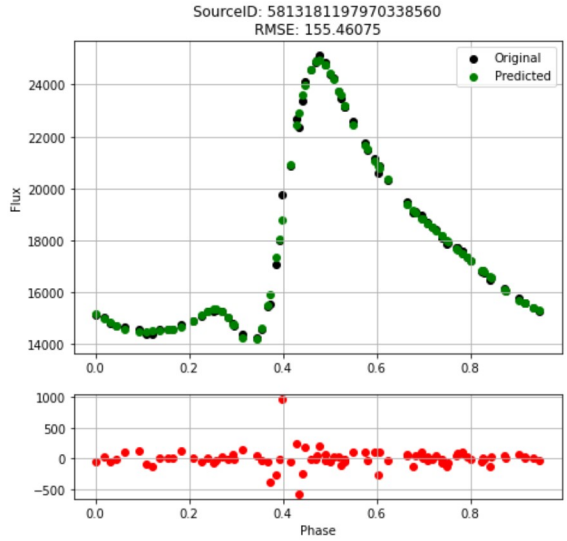


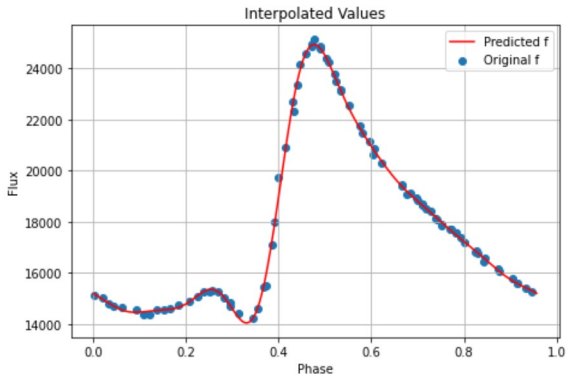

```
Analyzing fit for k=5...
```



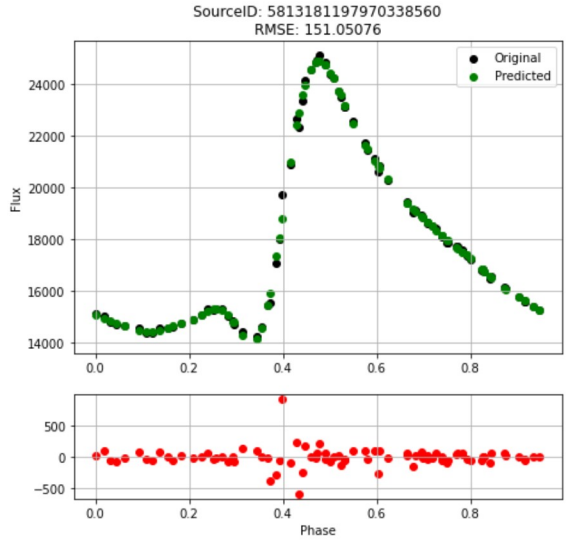


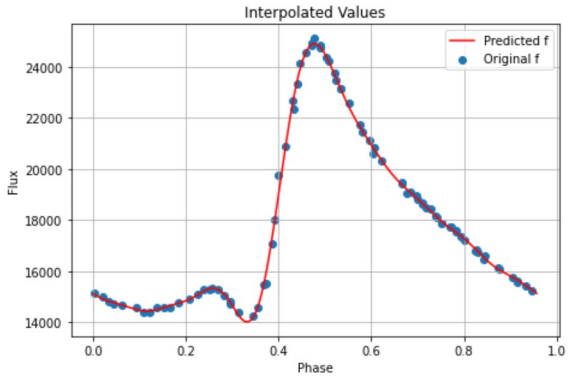
```
Analyzing fit for k=7...
```





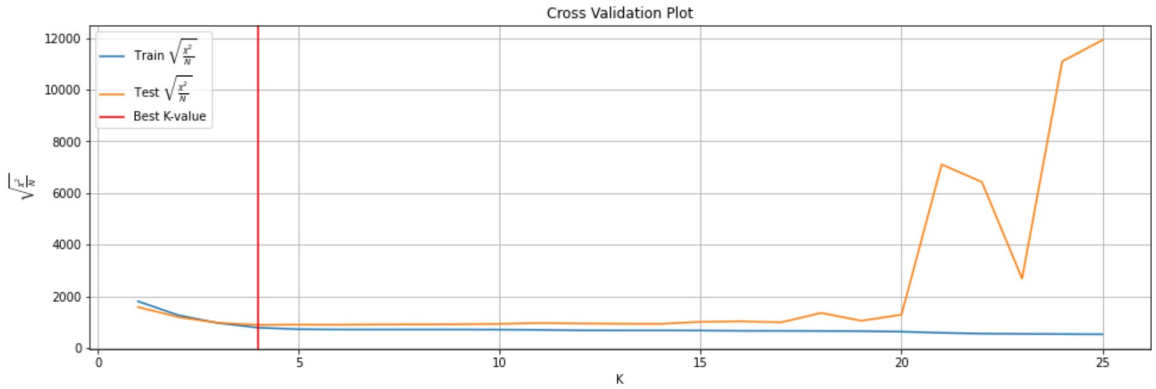

```
Analyzing fit for k=9...
```





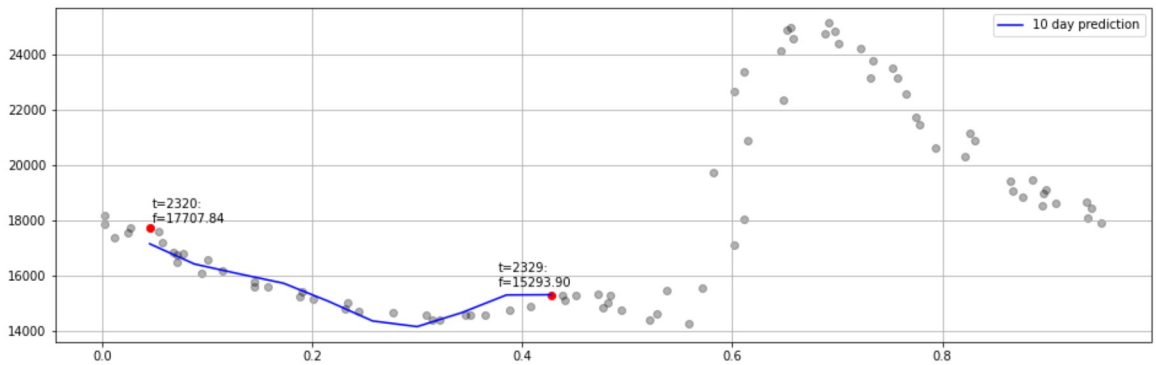
In [25]:
```
k_vals = np.arange(1,25+1)
train_mses = np.zeros(len(k_vals))
test_mses = np.zeros(len(k_vals))
time = selected_Gband['time']
flux = selected_Gband['flux']
period=estimate_period(selected_Gband)
omega=2*np.pi/period
time_train, time_test, flux_train, flux_test = train_test_split(time, f
lux, test_size=0.2)
```

In [26]:
```
p = False
for i,k in enumerate(k_vals):
#     print(f"Analyzing fit for k={k}...")

    X = setup_fit(flux_train, time_train, omega, k)
    beta_train = lstsq(X, flux_train.data)[0]
    A0=beta_train[0]
    a=beta_train[1:k+1]
    b=beta_train[k+1:]
    pred_train = pseudo_fourier(omega, time_train, A0, a, b)
    pred_test = pseudo_fourier(omega, time_test, A0, a, b)

    train_mse = rmse(flux_train, pred_train)
    train_mses[i] = train_mse

    test_mse = rmse(flux_test, pred_test)
    test_mses[i] = test_mse

    if p and k >= 20:
        fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,5))
        ax1.scatter(time_train%period,flux_train, label="Truth")
        ax1.scatter(time_train%period,pred_train,label="Pred")
        ax1.set(title=f"K={k}: $\\frac{{\\chi^2}}{{N}} = {train_mse:.3
f}$")
        ax1.legend()

        ax2.scatter(time_test%period,flux_test, label="Truth")
        ax2.scatter(time_test%period,pred_test,label="Pred")
        ax2.set(title=f"$\\frac{{\\chi^2}}{{N}} = {test_mse:.3f}$")
        ax2.legend()
        plt.show()

# print(test_mses)
fig,ax = plt.subplots(figsize=(16,5))
ax.plot(k_vals, train_mses, label="Train $\\sqrt{{\\frac{{\\chi^
2}}{{N}}}}$")
ax.plot(k_vals, test_mses, label="Test $\\sqrt{{\\frac{{\\chi^
2}}{{N}}}}$")
ax.axvline(np.argmin(test_mses)+1,color='red', label="Best K-value")
ax.legend()
ax.grid()
ax.set(xlabel="K", ylabel="$\\sqrt{{\\frac{{\\chi^2}}{{N}}}}$", title="
Cross Validation Plot")
plt.show()
```



In [27]:
```
k_best = np.argmin(test_mses)+1
X = setup_fit(flux, time, omega, k_best)
beta_best = lstsq(X, flux.data)[0]
A0_best=beta_best[0]
a_best=beta_best[1:k_best+1]
b_best=beta_best[k_best+1:]
last_flux, last_time = flux[-1], time[-1]
print(last_flux, last_time)
final_time = last_time+10
time_extrap = np.arange(last_time, final_time)
pred_extrap = pseudo_fourier(omega, time_extrap, A0_best, a_best, b_bes
t)
phase_extrap, flux_extrap = sort_by_phase(time_extrap%period, pred_extr
ap)
fig,ax = plt.subplots(figsize=(16,5))
ax.scatter(time%period, flux,color='black', alpha=0.3)
ax.scatter(last_time%period, last_flux,color='red')
ax.annotate(f"t={last_time:.0f}:\nf={last_flux:.2f}", (last_time%perio
d+0.002, last_flux+200))
ax.plot(phase_extrap, flux_extrap, color='blue', label="10 day predicti
on")
ax.scatter(phase_extrap[-1], flux_extrap[-1],color='red')
ax.annotate(f"t={time_extrap[-1]:.0f}:\nf={flux_extrap[-1]:.2f}", (phas
e_extrap[-1]-0.05, flux_extrap[-1]+300))
ax.grid()
ax.legend()
plt.show()
```

```
17707.83890462764 2320.202939518489
```



In [28]:
```
rrc_conds = f"""
best_classification='RRc'
AND int_average_g > 15
"""
rrlyrae_rrc = get_gaia_query_rrlyrae_cached(num_stars = 3, num_clean_ep
ochs=80, conds=rrc_conds)
rrlyrae_rrc
```

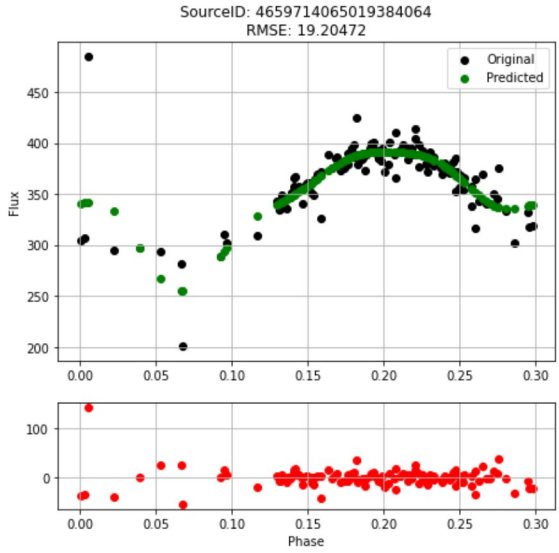
Out[28]: *Table length=3*

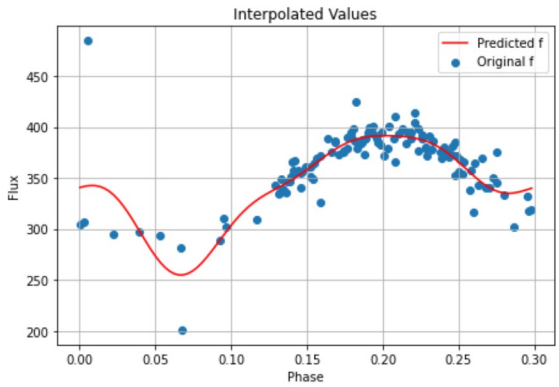
solution_id	designation	random_index	ref_epoch	ra	
			yr	deg	
int64	object	int64	float64	float64	
1635721458409799680	Gaia DR2 4659714065019384064	1255299599	2015.5	89.71078260193757	0.3
1635721458409799680	Gaia DR2 413458020354821504	1251053435	2015.5	19.111954139381382	0.0
1635721458409799680	Gaia DR2 5438044842945235712	865958970	2015.5	142.96098504851844	0.0

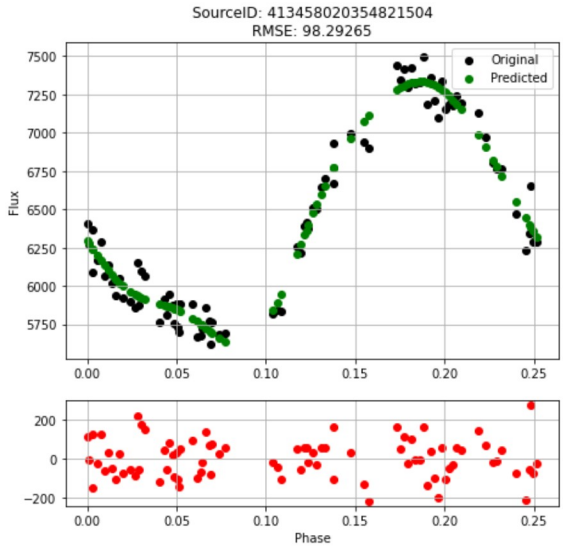
In [29]:
```
rrc_Gbands = [get_Gband(rrlyrae_rrc, i) for i, row in enumerate(rrlyrae
_rrc)]
```

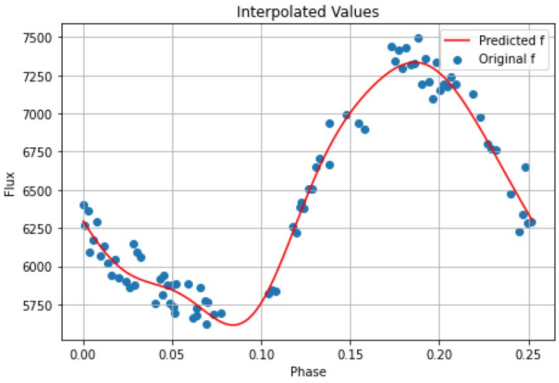
```
Analyzing star with source_id: 4659714065019384064
Analyzing star with source_id: 413458020354821504
Analyzing star with source_id: 5438044842945235712
```

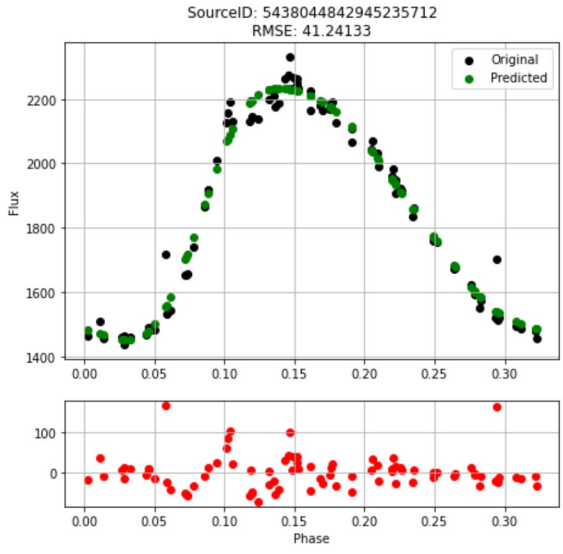
In [30]:
```
max_freq, min_freq = get_maxmin_freqs(rrlyrae_rrc)
for Gband in rrc_Gbands:
    get_prediction(Gband,max_freq=max_freq, min_freq=min_freq, k=k_best, p=True)
```

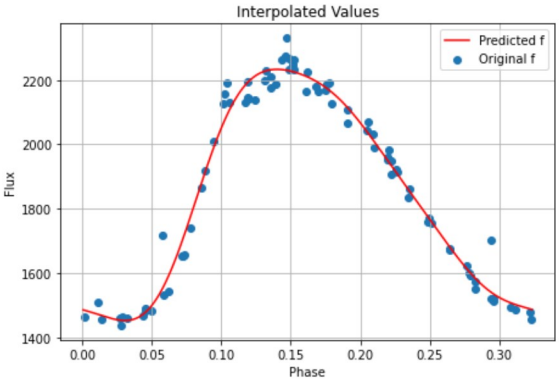












In [31]:
```
rrab_conds = f"""
best_classification='RRab'
AND int_average_g > 15
"""
rrlyrae_rrab = get_gaia_query_rrlyrae_cached(num_stars = 3, num_clean_epochs=80, conds=rrab_conds)
rrlyrae_rrab
```

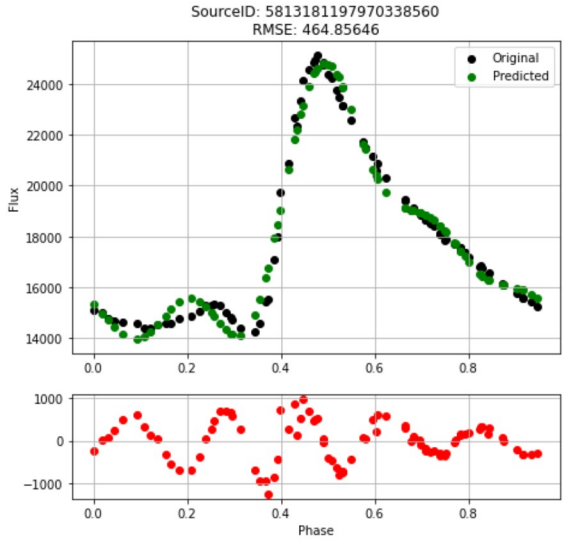
Out[31]: *Table length=3*

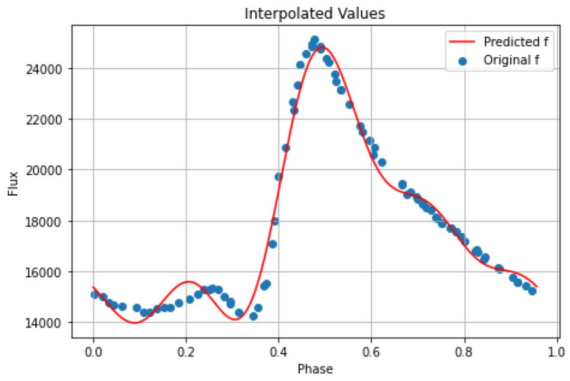
solution_id	designation	random_index	ref_epoch	ra	
			yr	deg	
int64	object	int64	float64	float64	
1635721458409799680	Gaia DR2 5813181197970338560	1546661016	2015.5	263.35817148226175	0.
1635721458409799680	Gaia DR2 6063663425261644544	307736288	2015.5	197.52702038202472	0.0
1635721458409799680	Gaia DR2 4180530812989384 96	516077856	2015.5	8.774802583044732	0.

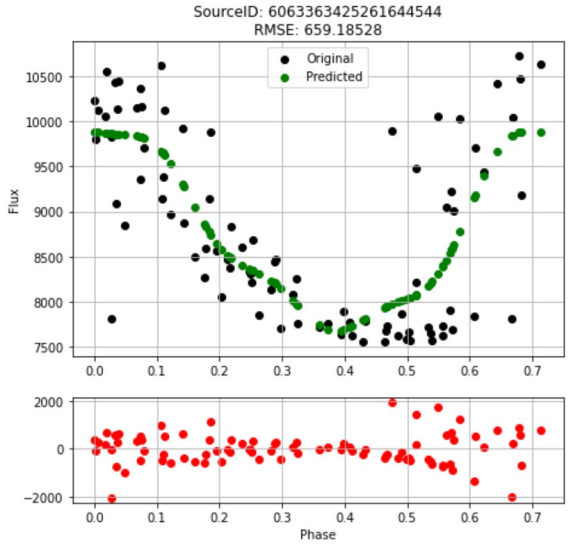
In [32]:
```
rrab_Gbands = [get_Gband(rrlyrae_rrab, i) for i, row in enumerate(rrlyrae_rrab)]
```

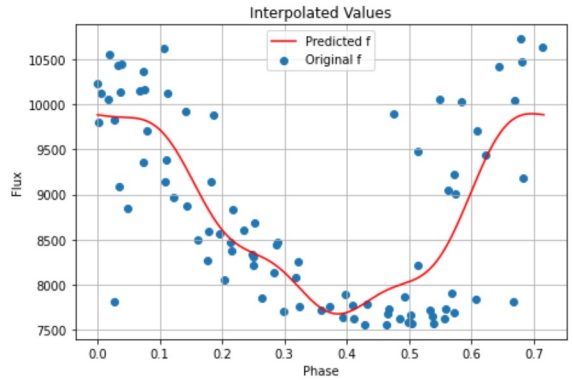
```
Analyzing star with source_id: 5813181197970338560
Analyzing star with source_id: 6063663425261644544
Analyzing star with source_id: 4180530812989384 96
```

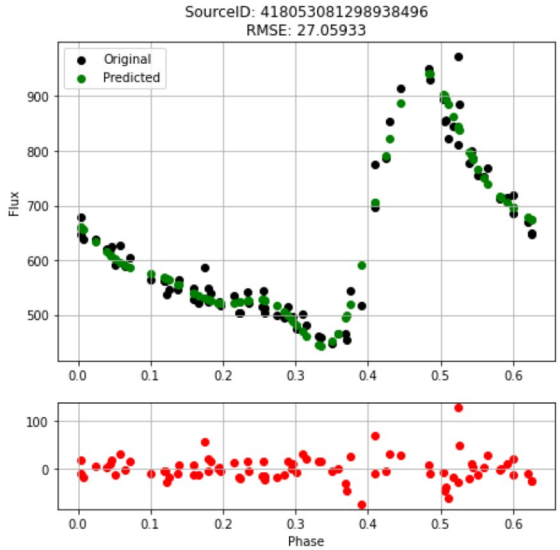
In [33]:
```
max_freq, min_freq = get_maxmin_freqs(rrlyrae_rrab)
for Gband in rrab_Gbands:
    get_prediction(Gband, max_freq=max_freq, min_freq=min_freq, k=k_best, p=True)
```

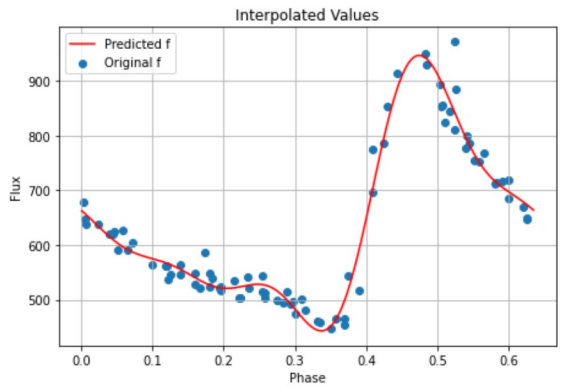












In [34]:
```
query="""
SELECT *
FROM gaiaedr3.gaia_source as gaia3
JOIN gaiaedr3.dr2_neighbourhood as gaia2
    ON gaia2.dr3_source_id = gaia3.source_id
JOIN gaiadr2.vari_rrlyrae as rrlyrae
    ON gaia2.dr2_source_id = rrlyrae.source_id
JOIN external.gaiaedr3_distance as dists
    ON gaia3.source_id = dists.source_id
WHERE
    abs(b)>30
    AND parallax>0.25
    AND parallax_over_error>5
    AND pf IS NOT NULL
"""
rrlyrae_gaia3 = get_gaia_query_general_cached(query)
print(f"Got {len(rrlyrae_gaia3)} stars")
rrlyrae_gaia3[:10]
```

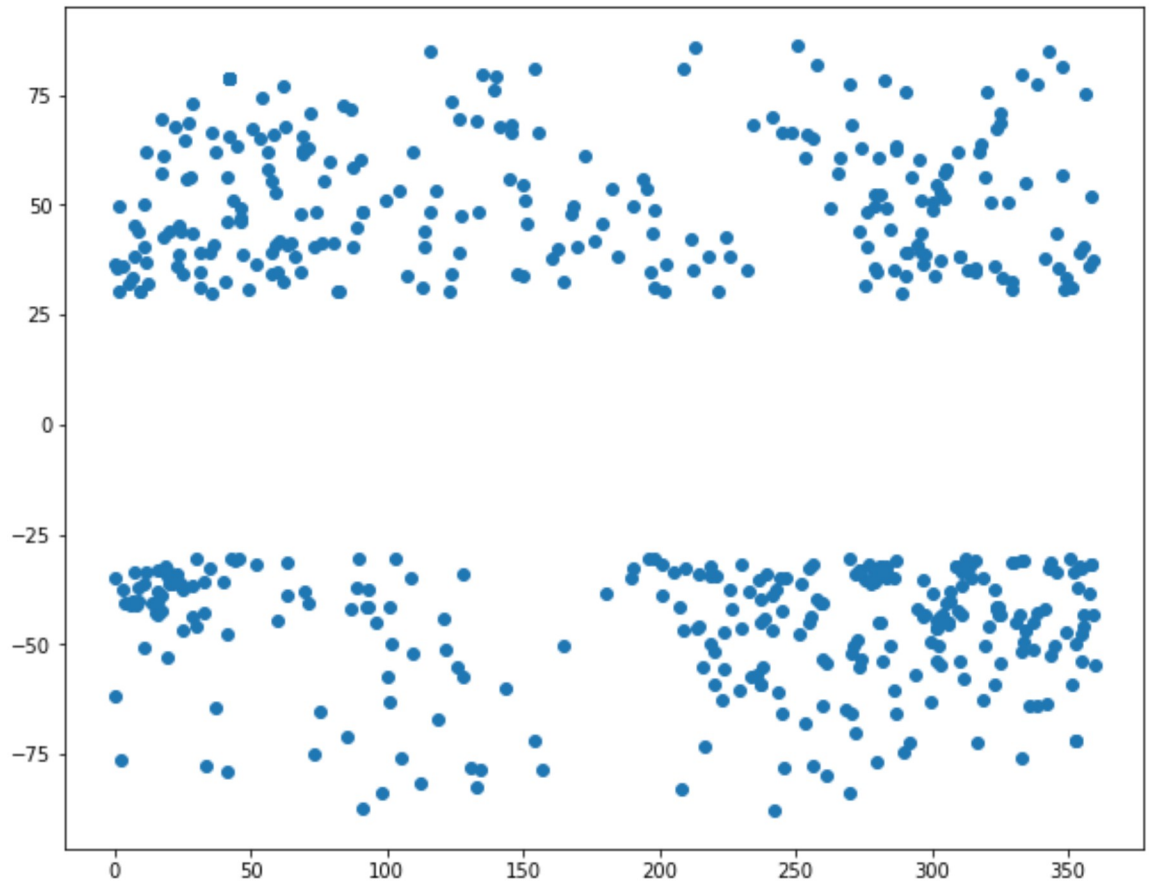
```
Got 548 stars
```

Out[34]: *Table length=10*

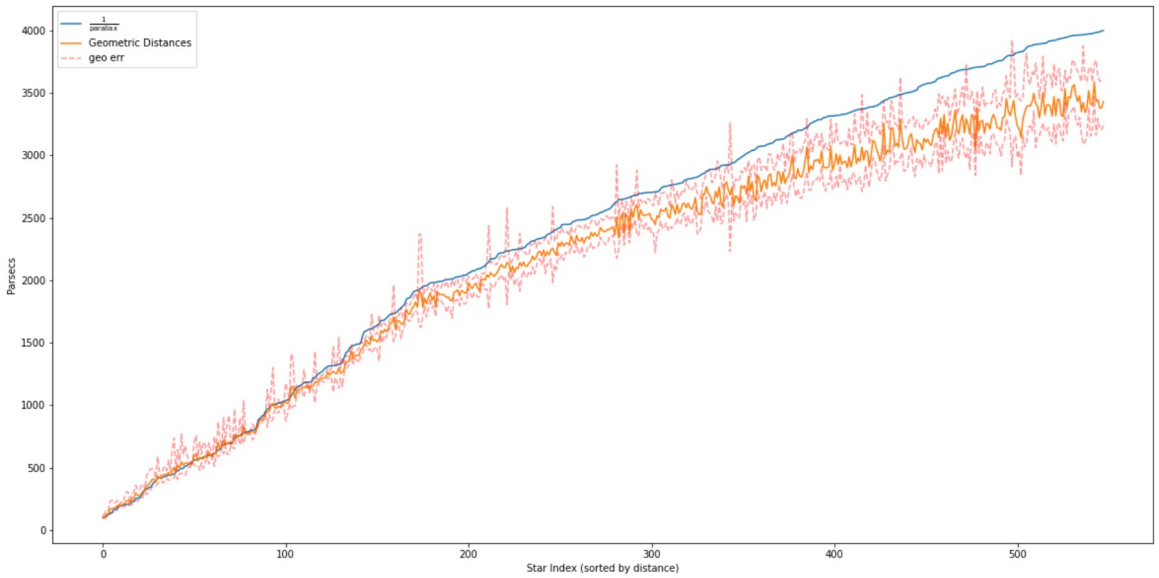
solution_id	designation	source_id	random_index	ref_epoch	
				yr	
int64	object	int64	int64	float64	
1636042515805110273	Gaia EDR3 4657904848780392832	4657904848780392832	1297321964	2016.0	4
1636042515805110273	Gaia EDR3 4798586678570579072	4798586678570579072	279121884	2016.0	4
1636042515805110273	Gaia EDR3 4685757887726594816	4685757887726594816	1480418546	2016.0	1
1636042515805110273	Gaia EDR3 4689637956899105792	4689637956899105792	1084076130	2016.0	5
1636042515805110273	Gaia EDR3 4689637961175354240	4689637961175354240	47042557	2016.0	5
1636042515805110273	Gaia EDR3 4700326863447344768	4700326863447344768	267955142	2016.0	3
1636042515805110273	Gaia EDR3 4690886594062571008	4690886594062571008	1401140051	2016.0	1
1636042515805110273	Gaia EDR3 4658012051209225984	4658012051209225984	847616141	2016.0	8
1636042515805110273	Gaia EDR3 4658145092114070144	4658145092114070144	1681799523	2016.0	1
1636042515805110273	Gaia EDR3 4685834544315524224	4685834544315524224	398366609	2016.0	1

In [35]:
```
plt.figure(figsize=(10,8))
plt.scatter(rrlyrae_gaia3['l'], rrlyrae_gaia3['b'])
```

Out[35]: <matplotlib.collections.PathCollection at 0x7f387f1ee320>



In [36]:
```
rrlyrae_gaia3_df = rrlyrae_gaia3.to_pandas()
dists = pd.concat(
        [rrlyrae_gaia3_df[["r_med_geo", "r_hi_geo", "r_lo_geo"]], 1000/
rrlyrae_gaia3_df['parallax']],
        axis=1
    ).rename(columns={"parallax":"distance"}).sort_values("distance")
n = np.arange(dists.shape[0])
plt.figure(figsize=(20,10))
plt.plot(n,dists["distance"], label=r"$\frac{1}{\mathrm{parallax}}$")
plt.plot(n, dists["r_med_geo"], label="Geometric Distances")
plt.plot(n, dists["r_hi_geo"], "--", color="red", alpha=0.4, label="geo
err")
plt.plot(n, dists["r_lo_geo"], "--", color="red", alpha=0.4)
plt.ylabel("Parsecs")
plt.xlabel("Star Index (sorted by distance)")
plt.legend()
plt.show()
```



In [37]:
```
query = """
    SELECT *
    FROM gaiadr2.gaia_source as gaia2
    JOIN gaiaedr3.dr2_neighbourhood as link
        ON gaia2.source_id = link.dr2_source_id
    JOIN gaiadr2.vari_rrlyrae as rrlyrae
        ON rrlyrae.source_id = gaia2.source_id
    WHERE
        parallax_over_error > 5
        AND abs(b) > 30
        AND parallax > 0.25
        AND pf IS NOT NULL
"""
rrlyrae_gaia2 = get_gaia_query_general_cached(query)
print(f"Got {len(rrlyrae_gaia2)} stars")
rrlyrae_gaia2[:10]
```

```
Got 518 stars
```

Out[37]: *Table length=10*

solution_id	designation	source_id	random_index	ref_epoch	
				yr	
int64	object	int64	int64	float64	
1635721458409799680	Gaia DR2 4654631533876789632	4654631533876789632	502718668	2015.5	
1635721458409799680	Gaia DR2 4654631533876789632	4654631533876789632	502718668	2015.5	
1635721458409799680	Gaia DR2 4685757887726594816	4685757887726594816	812747682	2015.5	1
1635721458409799680	Gaia DR2 4689637956899105792	4689637956899105792	1530397305	2015.5	
1635721458409799680	Gaia DR2 4689637956899105792	4689637956899105792	1530397305	2015.5	
1635721458409799680	Gaia DR2 4700326863447344768	4700326863447344768	642465858	2015.5	
1635721458409799680	Gaia DR2 4658186602866639872	4658186602866639872	1491505400	2015.5	
1635721458409799680	Gaia DR2 4658186602866639872	4658186602866639872	1491505400	2015.5	
1635721458409799680	Gaia DR2 4632502526615290624	4632502526615290624	1511122055	2015.5	3
1635721458409799680	Gaia DR2 4632502526615290624	4632502526615290624	1511122055	2015.5	3

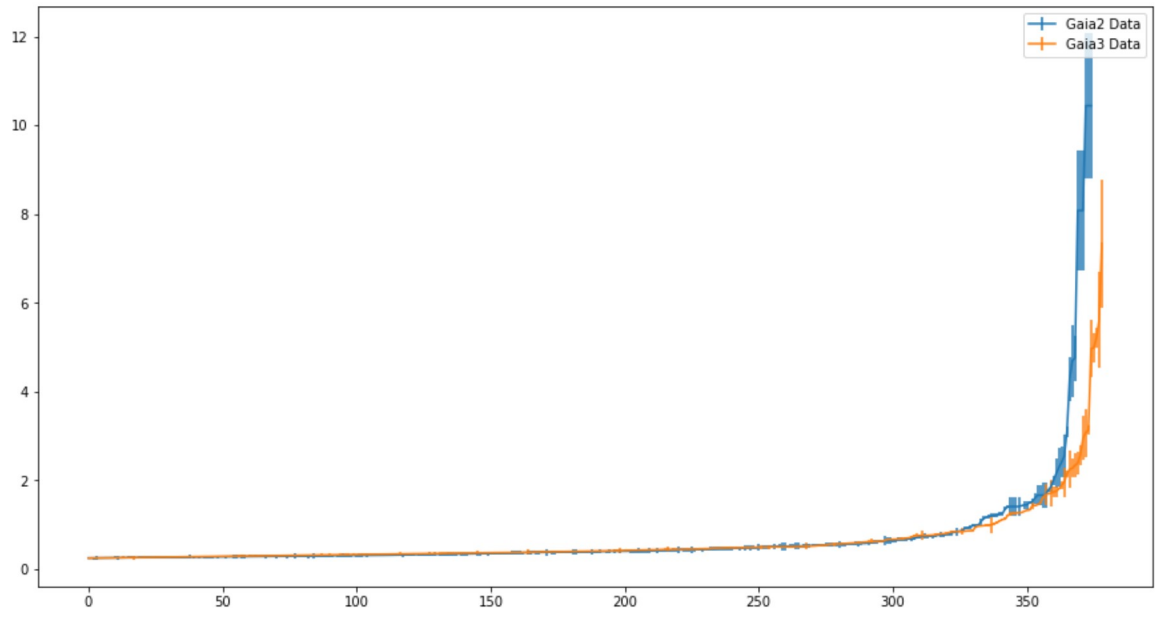
In [38]:
```
mask = [(source_id in rrlyrae_gaia2["dr3_source_id"]) for source_id in
rrlyrae_gaia3["source_id"]]
gaia3_data = rrlyrae_gaia3[mask].to_pandas().sort_values("parallax")
gaia3_data["dr3_source_id"]
```

Out[38]:
```
271     3186033648144893696
340     6910756005950942336
93      3945610468551775616
171      612591803903942016
183     1540009731422069504
                ...
61      4943063090574101376
63      4955988365155969792
3       4700326863447344768
279     4998160202357316096
179     1328057321618952064
Name: dr3_source_id, Length: 379, dtype: int64
```

In [39]:
```
mask2 = [(source_id in rrlyrae_gaia3["dr3_source_id"]) for source_id in
rrlyrae_gaia2["source_id"]]
gaia2_data = rrlyrae_gaia2[mask2].to_pandas().sort_values("parallax")
len(gaia2_data)
```

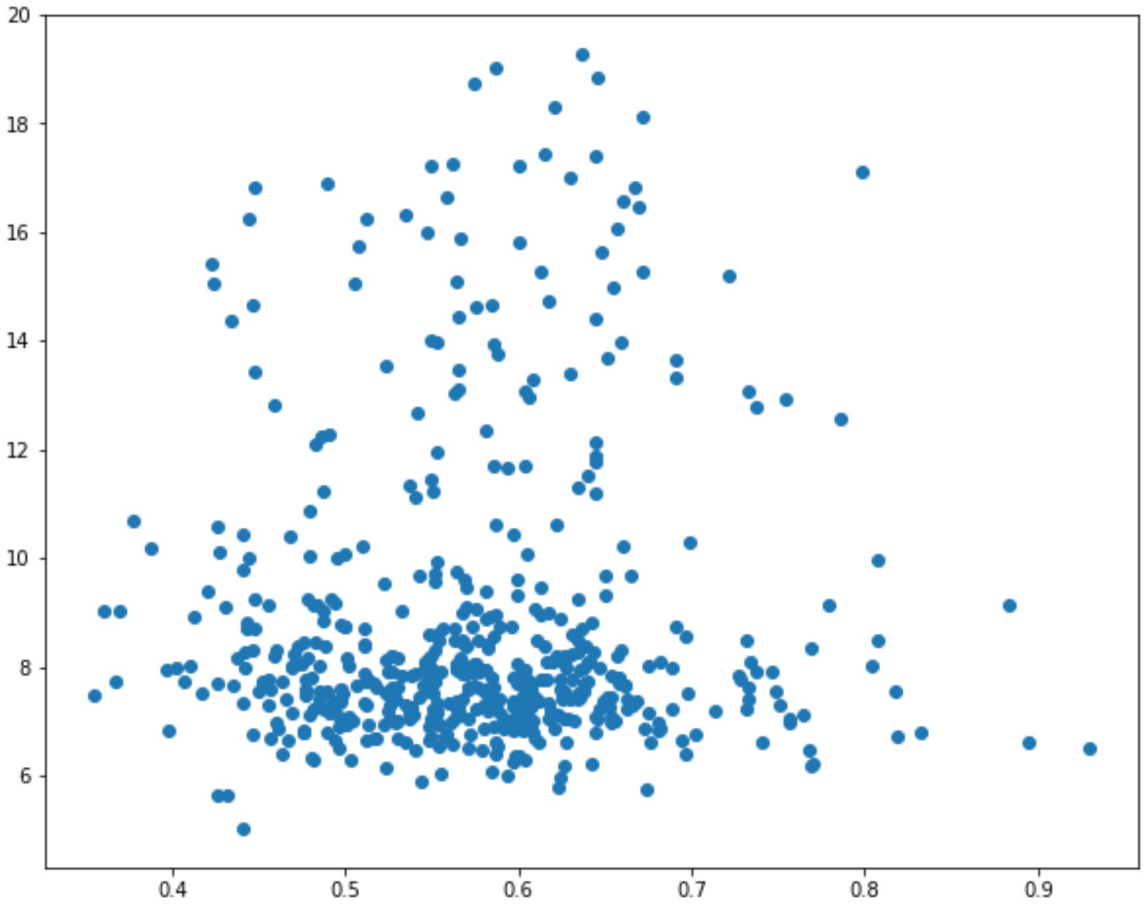
Out[39]:
```
375
```

In [40]:
```
plt.figure(figsize=(15,8))
plt.errorbar(np.arange(len(gaia2_data)), gaia2_data["parallax"], yerr=g
aia2_data["parallax_error"], label="Gaia2 Data")
plt.errorbar(np.arange(len(gaia3_data)), gaia3_data["parallax"], yerr=g
aia3_data["parallax_error"], label="Gaia3 Data")
plt.legend()
plt.show()
```



In [41]:
```
Mg = rrlyrae_gaia3["phot_g_mean_mag"]-5*np.log10((rrlyrae_gaia3["r_hi_geo"]-rrlyrae_gaia3["r_lo_geo"])/2)+5
plt.figure(figsize=(10,8))
plt.scatter(rrlyrae_gaia3["pf"], Mg)
```

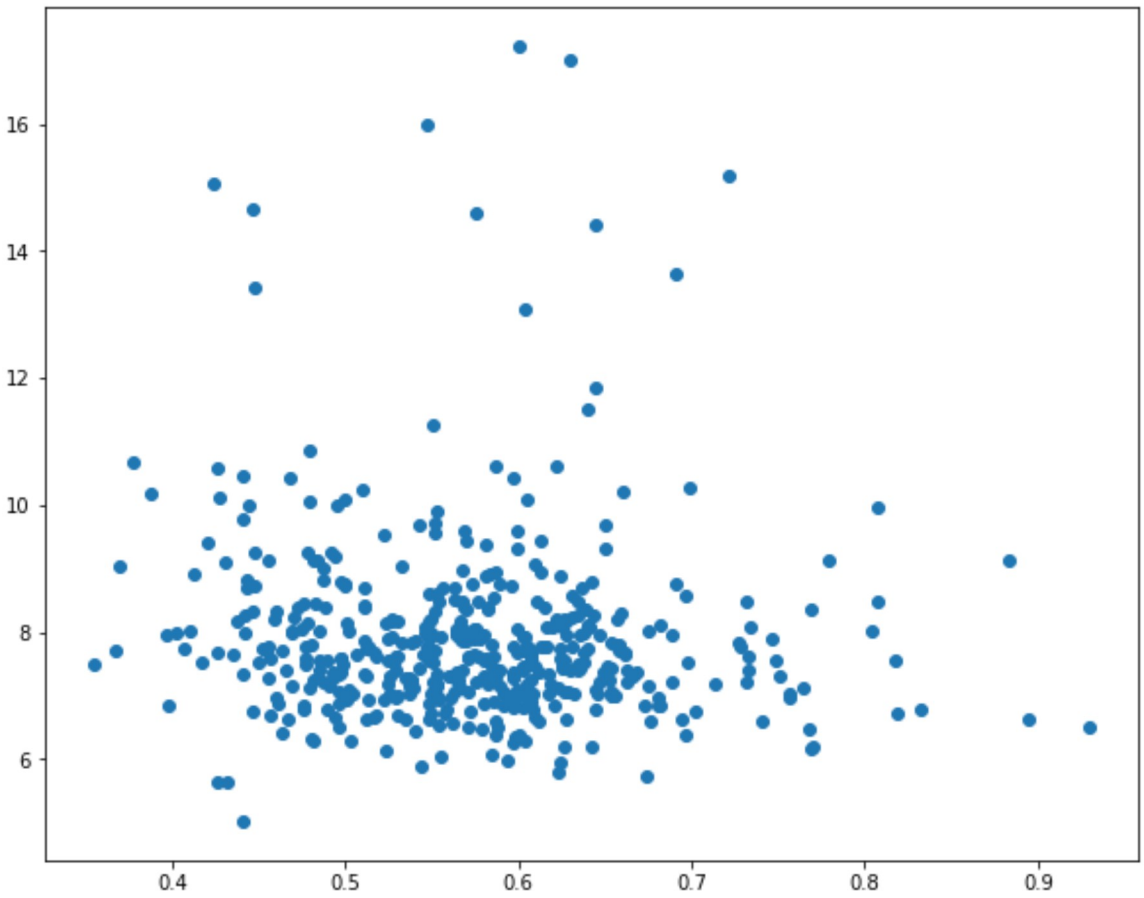
Out[41]:
```
<matplotlib.collections.PathCollection at 0x7f387d0e6f28>
```



In [42]:
```
query = """
    SELECT *
    FROM gaiaedr3.gaia_source as gaia
    JOIN gaiaedr3.dr2_neighbourhood as link
        ON gaia.source_id = link.dr3_source_id
    JOIN gaiadr2.vari_rrlyrae as lyrae
        ON lyrae.source_id = link.dr2_source_id
    JOIN external.gaiaedr3_distance as dists
        ON gaia.source_id = dists.source_id
    WHERE
        parallax_over_error > 5
        AND abs(b) > 30
        AND parallax > 0.25
        AND pf IS NOT NULL
        AND astrometric_excess_noise < 1
        AND 1 + 0.015*power(bp_rp, 2) < phot_bp_rp_excess_factor
        AND phot_bp_rp_excess_factor < 1.3 + 0.06*power(bp_rp, 2)
"""

rrlyrae_gaia3_filt = get_gaia_query_general_cached(query)
```

In [43]:
```
Mg = rrlyrae_gaia3_filt["phot_g_mean_mag"]-5*np.log10((rrlyrae_gaia3_filt["r_hi_geo"]-rrlyrae_gaia3_filt["r_lo_geo"])/2)+5
plt.figure(figsize=(10,8))
plt.scatter(rrlyrae_gaia3_filt["pf"], Mg)
```

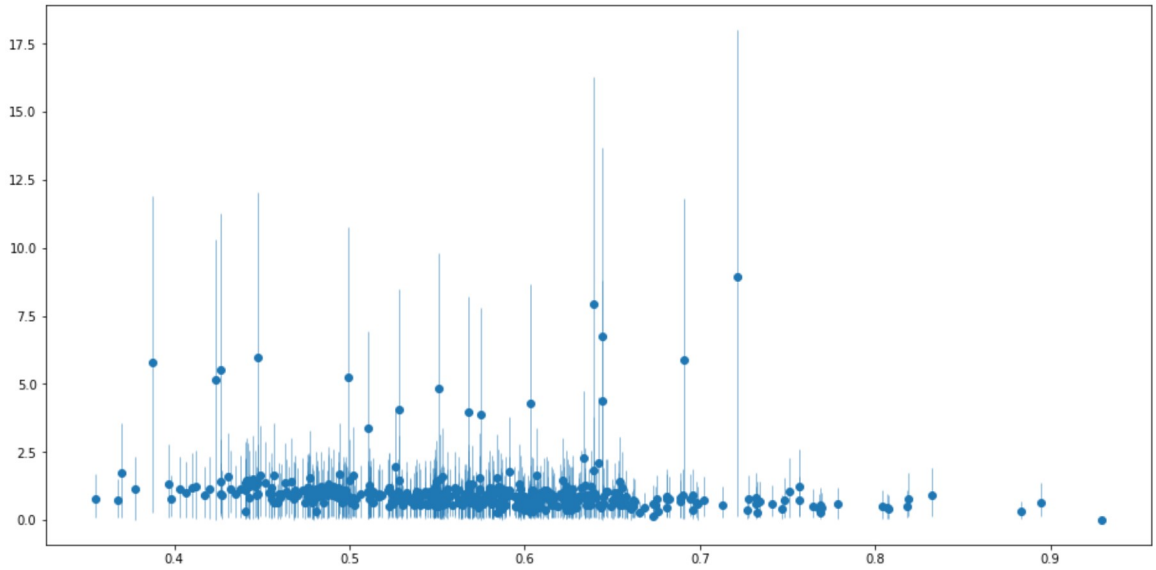
Out[43]: <matplotlib.collections.PathCollection at 0x7f387cfe9e48>



In [77]:
```
rrlyrae_gaia3_filt_df = rrlyrae_gaia3_filt.to_pandas()
Mg = rrlyrae_gaia3_filt_df["phot_g_mean_mag"]-5*np.log10(rrlyrae_gaia3_filt_df["r_med_geo"])+5
rrlyrae_gaia3_filt_df = pd.concat([rrlyrae_gaia3_filt_df,
                                   Mg.rename("Mg")],
                                   axis=1)
rrlyrae_gaia3_filt_df = rrlyrae_gaia3_filt_df[rrlyrae_gaia3_filt_df["Mg"]<9]
```

In [78]:
```
high_error = rrlyrae_gaia3_filt_df["phot_g_mean_mag"]-5*np.log10(rrlyrae_gaia3_filt_df["r_hi_geo"])+5
low_error = rrlyrae_gaia3_filt_df["phot_g_mean_mag"]-5*np.log10(rrlyrae_gaia3_filt_df["r_lo_geo"])+5
error = np.stack((high_error, low_error))
```

In [79]:
```
plt.figure(figsize=(16,8))
plt.errorbar(rrlyrae_gaia3_filt_df["pf"], rrlyrae_gaia3_filt_df["Mg"],
yerr=error, fmt="o", elinewidth=0.5)
plt.show()
```



In [47]:
```
query = f"""
    SELECT *
    FROM gaiaedr3.gaia_source as gaia
    JOIN gaiaedr3.dr2_neighbourhood as link
        ON gaia.source_id = link.dr3_source_id
    JOIN gaiadr2.vari_rrlyrae as lyrae
        ON lyrae.source_id = link.dr2_source_id
    JOIN external.gaiadr2_geometric_distance as dists
        ON link.dr2_source_id = dists.source_id
    WHERE
        parallax_over_error > 5
        AND abs(b) > 30
        AND parallax > 0.25
        AND pf IS NOT NULL
        AND astrometric_excess_noise < 1
        AND 1 + 0.015*power(bp_rp, 2) < phot_bp_rp_excess_factor
        AND phot_bp_rp_excess_factor < 1.3 + 0.06*power(bp_rp, 2)
"""

rrlyrae_gaia2_filt = get_gaia_query_general_cached(query)
rrlyrae_gaia2_filt[:10]
```

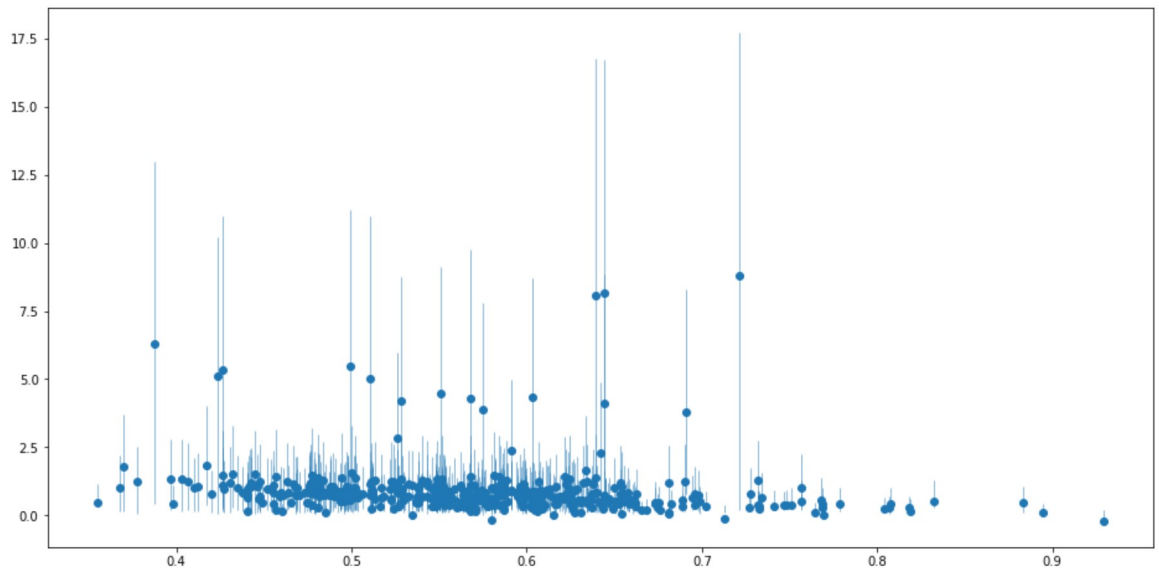
Out[47]: *Table length=10*

solution_id	designation	source_id	random_index	ref_epoch	
				yr	
int64	object	int64	int64	float64	
1636042515805110273	Gaia EDR3 4685757887726594816	4685757887726594816	1480418546	2016.0	1
1636042515805110273	Gaia EDR3 4793950312913699328	4793950312913699328	989614984	2016.0	
1636042515805110273	Gaia EDR3 6271733007667343360	6271733007667343360	1511020438	2016.0	2
1636042515805110273	Gaia EDR3 6275283468151655424	6275283468151655424	985339877	2016.0	2
1636042515805110273	Gaia EDR3 6321036891467501952	6321036891467501952	945181857	2016.0	
1636042515805110273	Gaia EDR3 6327036861995922560	6327036861995922560	676464905	2016.0	
1636042515805110273	Gaia EDR3 6189279973568784384	6189279973568784384	282814367	2016.0	2
1636042515805110273	Gaia EDR3 1481133009762718720	1481133009762718720	1242872450	2016.0	2
1636042515805110273	Gaia EDR3 6190268571961010944	6190268571961010944	710567451	2016.0	
1636042515805110273	Gaia EDR3 6308159961037880576	6308159961037880576	705715556	2016.0	2

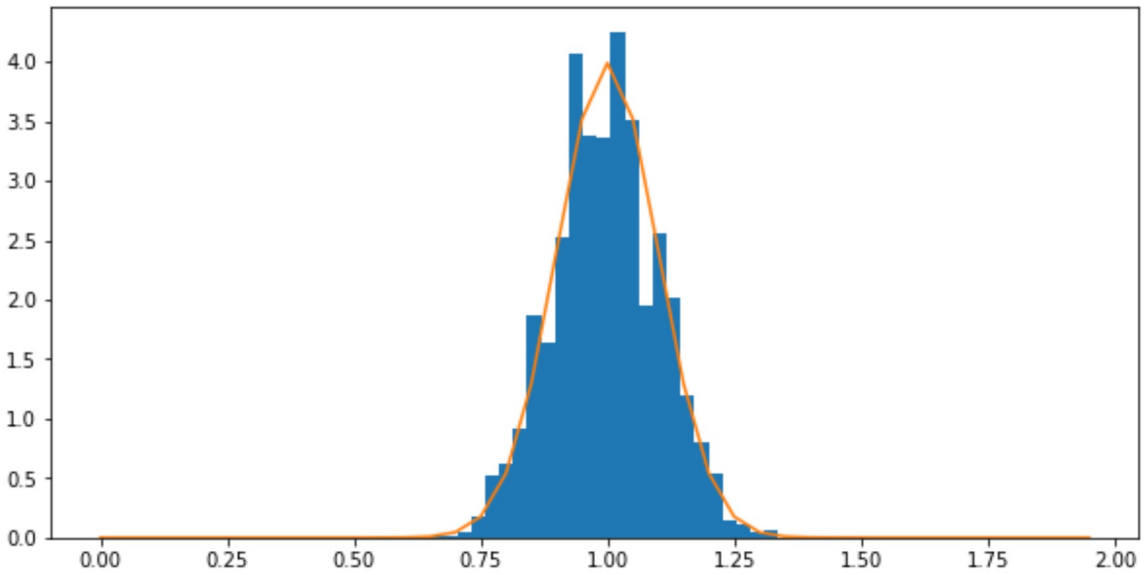
In [91]:
```
rrlyrae_gaia2_filt_df = rrlyrae_gaia2_filt.to_pandas()
Mg = rrlyrae_gaia2_filt_df["phot_g_mean_mag"]-5*np.log10(rrlyrae_gaia2_
filt_df["r_est"])+5
rrlyrae_gaia2_filt_df = pd.concat([rrlyrae_gaia2_filt_df, Mg.rename("M
g")],axis=1)
rrlyrae_gaia2_filt_df = rrlyrae_gaia2_filt_df[rrlyrae_gaia2_filt_df["M
g"]<9]
```

In [92]:
```
high_error = rrlyrae_gaia2_filt_df["phot_g_mean_mag"]-5*np.log10(rrlyra
e_gaia2_filt_df["r_hi"])+5
rrlyrae_gaia2_filt_df = pd.concat([rrlyrae_gaia2_filt_df, high_error.re
name("Mg_hi")],axis=1)
low_error = rrlyrae_gaia2_filt_df["phot_g_mean_mag"]-5*np.log10(rrlyrae
_gaia2_filt_df["r_lo"])+5
rrlyrae_gaia2_filt_df = pd.concat([rrlyrae_gaia2_filt_df, low_error.ren
ame("Mg_lo")],axis=1)
error = np.stack((high_error, low_error))
```

In [93]:
```
plt.figure(figsize=(16,8))
plt.errorbar(rrlyrae_gaia2_filt_df['pf'], rrlyrae_gaia2_filt_df['Mg'],
yerr=error, fmt="o", elinewidth=0.5)
plt.show()
```

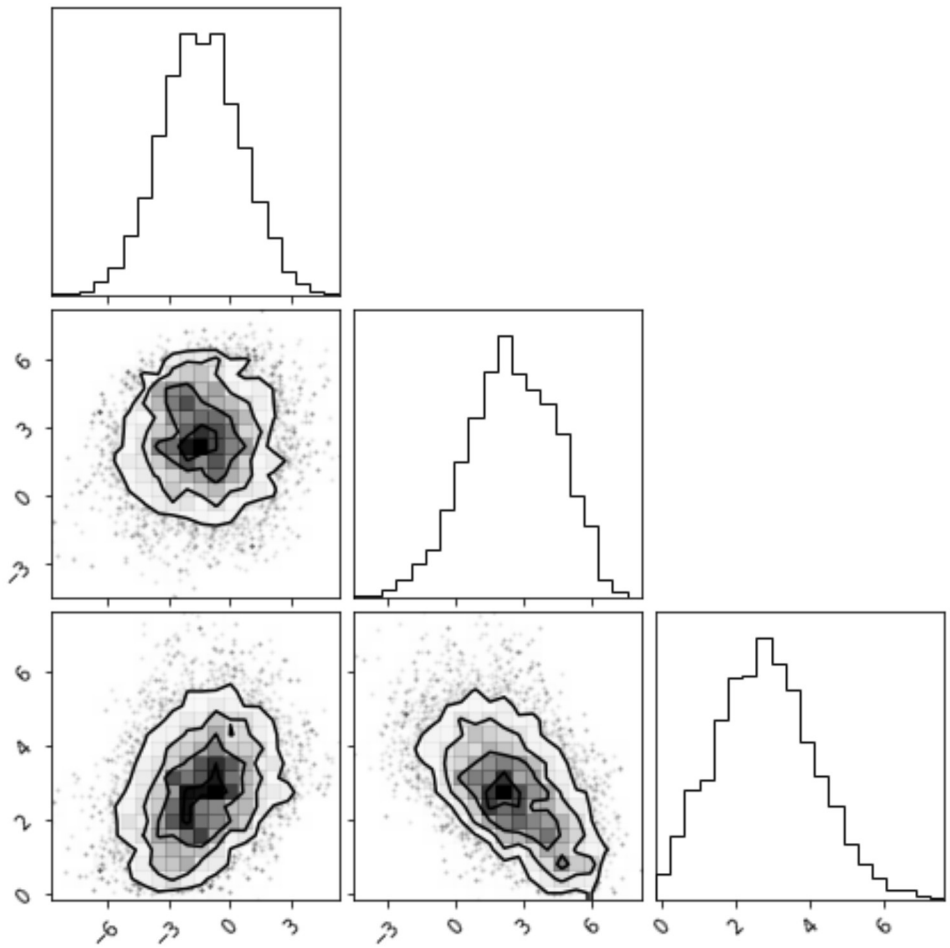


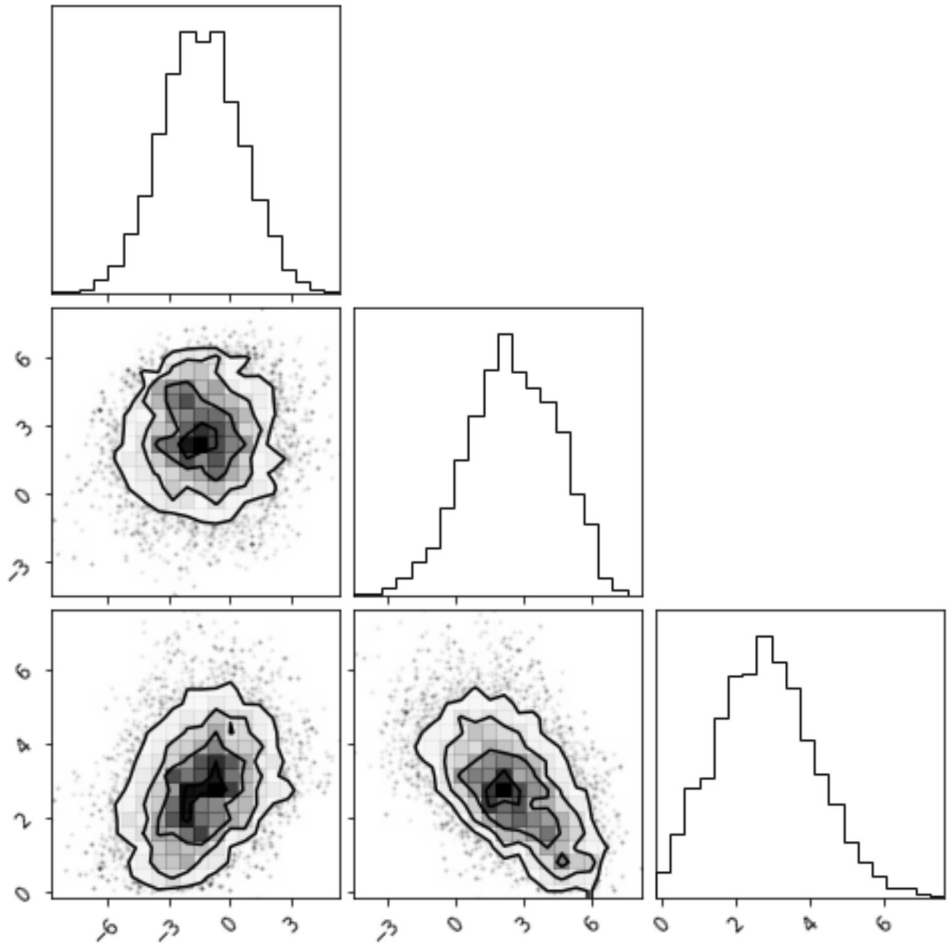
In [53]:
```
x = sampler(gauss)
r = np.arange(0, 2, 0.05)
plt.figure(figsize = (10, 5))
plt.hist(x, density=True, bins=25)
plt.plot(r, gauss(r))
plt.show()
```



In [65]:
```
x_data = sampler_data(rrlyrae_gaia3_filt_df, posterior)
corner(x_data.T)
```

Out[65]:




6
3
0
−3
6
4
2
0
−6
−3
0
3
−3
0
3
6
0
2
4
6

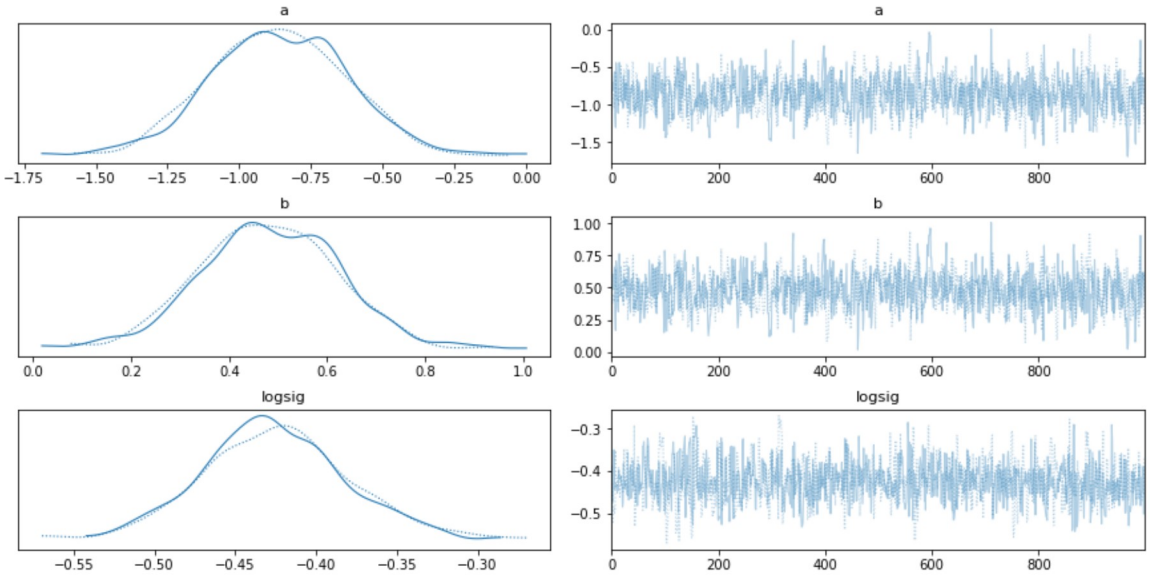
In [125]:
```
y_err = abs(-5*(rrlyrae_gaia3_filt_df["r_hi_geo"] - rrlyrae_gaia3_filt
_df["r_lo_geo"])/rrlyrae_gaia3_filt_df["r_med_geo"])
y = rrlyrae_gaia3_filt_df["Mg"]
x = rrlyrae_gaia3_filt_df["pf"]
samples = run_mcmc(x, y, y_err)
```

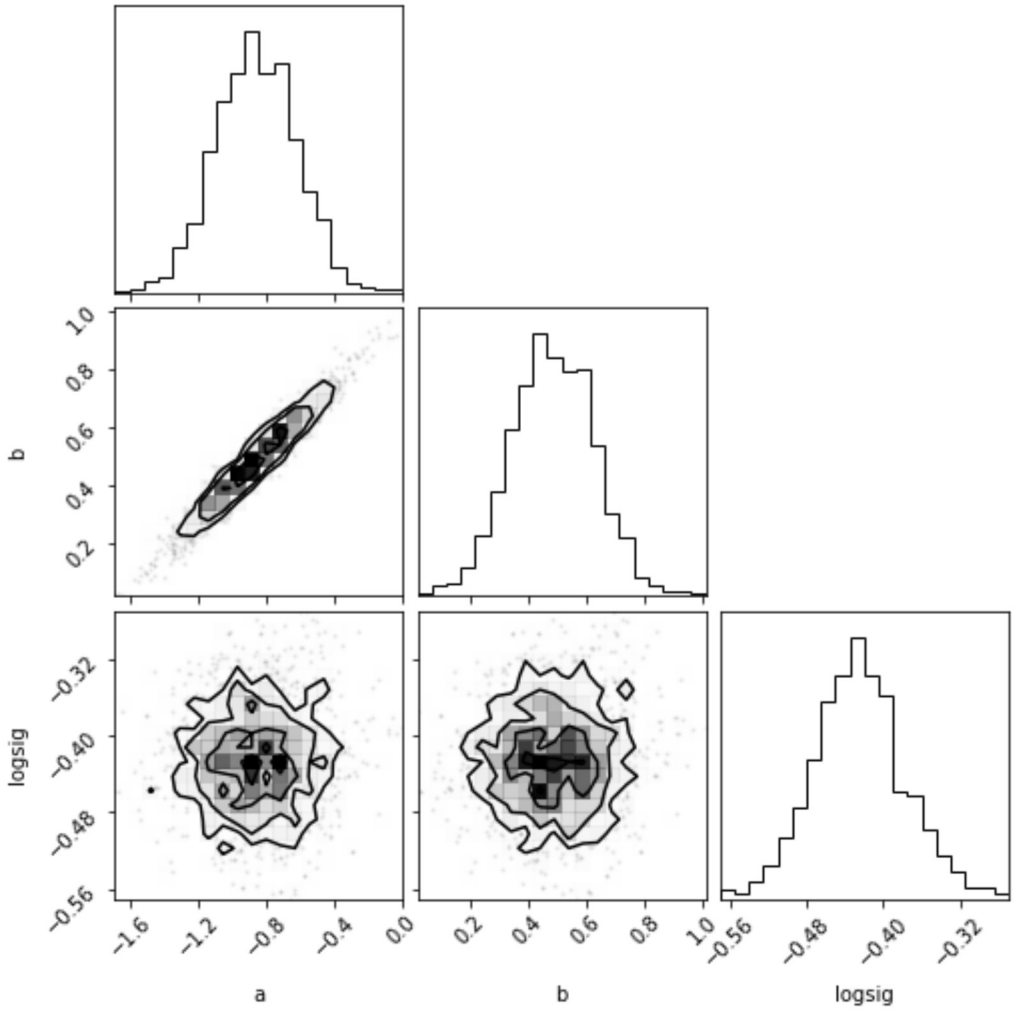
```
/home/rafferino/.local/lib/python3.6/site-packages/pymc3/sampling.py:
468: FutureWarning: In an upcoming release, pm.sample will return an
`arviz.InferenceData` object instead of a `MultiTrace` by default. Yo
u can pass return_inferencedata=True or return_inferencedata=False to
be safe and silence this warning.
  FutureWarning,
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [logsig, b, a]
```

100.00% [4000/4000 00:03<00:00 Sampling

2 chains, 0 divergences]

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2
_000 draws total) took 4 seconds.
```

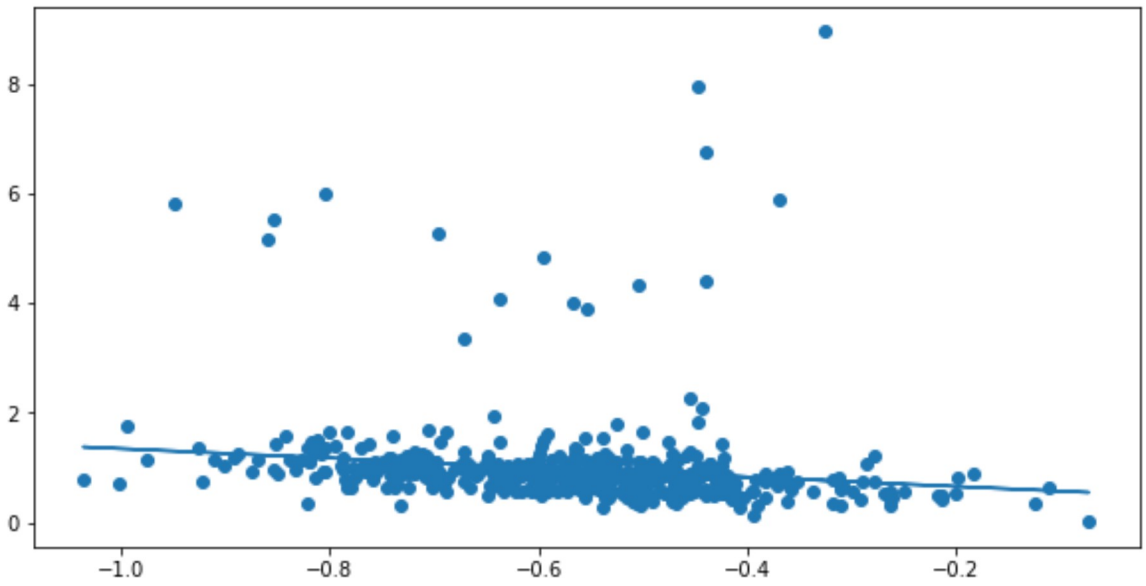




In [126]:
```
a_est, b_est, logsig_est = np.mean(samples, axis=0)
print(a_est, b_est)
```

```
-0.8631928819783391 0.4810903397930271
```

In [127]:
```
plt.figure(figsize=(10,5))
plt.scatter(np.log(rrlyrae_gaia3_filt_df["pf"]), rrlyrae_gaia3_filt_df
["Mg"])
plt.plot(np.log(rrlyrae_gaia3_filt_df["pf"]), a_est*np.log(rrlyrae_gai
a3_filt_df["pf"])+b_est)
plt.show()
```



In [111]:
```
query = f"""
    SELECT *
    FROM gaiaedr3.gaia_source as gaia
    JOIN gaiaedr3.dr2_neighbourhood as link
        ON gaia.source_id = link.dr3_source_id
    JOIN gaiadr2.vari_rrlyrae as lyrae
        ON lyrae.source_id = link.dr2_source_id
    JOIN gaiadr2.allwise_best_neighbour as wise_link
        ON wise_link.source_id = link.dr2_source_id
    JOIN gaiadr1.allwise_original_valid as wise
        ON wise.allwise_oid = wise_link.allwise_oid
    WHERE
        parallax_over_error > 5
        AND abs(b) > 30
        AND parallax > 0.25
        AND pf IS NOT NULL
        AND astrometric_excess_noise < 1
        AND 1 + 0.015*power(bp_rp, 2) < phot_bp_rp_excess_factor
        AND phot_bp_rp_excess_factor < 1.3 + 0.06*power(bp_rp, 2)
"""
rrlyrae_wise_gaia3 = get_gaia_query_general_cached(query)
rrlyrae_wise_gaia3[:10]
```

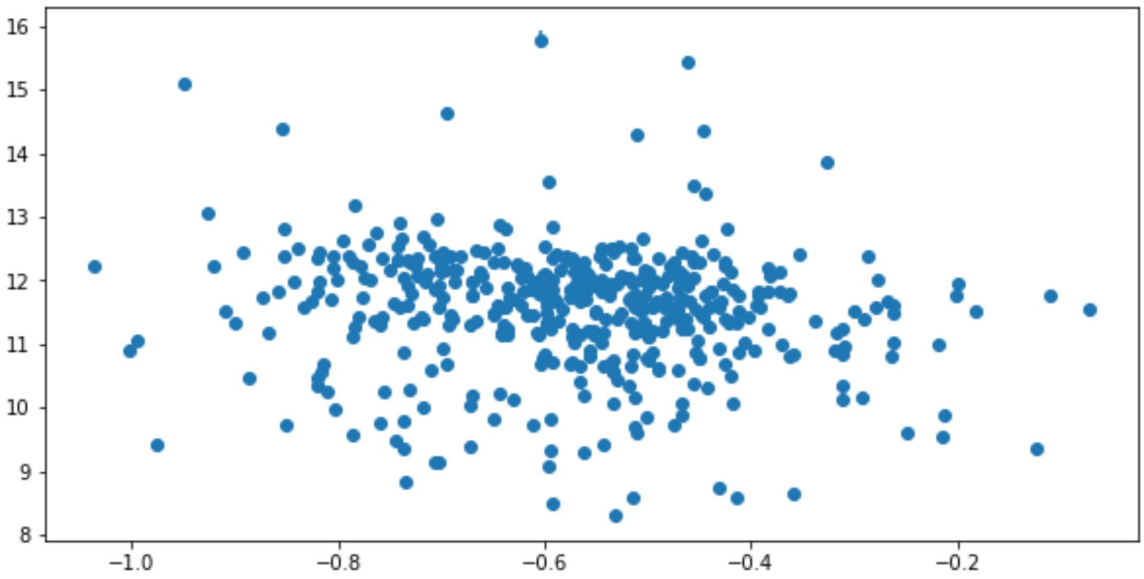
```
INFO: Query finished. [astroquery.utils.tap.core]
```

Out[111]: *Table length=10*

solution_id	designation	source_id	random_index	ref_epoch	
				yr	
int64	object	int64	int64	float64	
1636042515805110273	Gaia EDR3 4685757887726594816	4685757887726594816	1480418546	2016.0	1
1636042515805110273	Gaia EDR3 4793950312913699328	4793950312913699328	989614984	2016.0	
1636042515805110273	Gaia EDR3 6271733007667343360	6271733007667343360	1511020438	2016.0	2
1636042515805110273	Gaia EDR3 6275283468151655424	6275283468151655424	985339877	2016.0	2
1636042515805110273	Gaia EDR3 6321036891467501952	6321036891467501952	945181857	2016.0	
1636042515805110273	Gaia EDR3 6327036861995922560	6327036861995922560	676464905	2016.0	
1636042515805110273	Gaia EDR3 6189279973568784384	6189279973568784384	282814367	2016.0	2
1636042515805110273	Gaia EDR3 1481133009762718720	1481133009762718720	1242872450	2016.0	2
1636042515805110273	Gaia EDR3 6190268571961010944	6190268571961010944	710567451	2016.0	
1636042515805110273	Gaia EDR3 6308159961037880576	6308159961037880576	705715556	2016.0	2

In [115]:
```
plt.figure(figsize=(10,5))
plt.errorbar(np.log(rrlyrae_wise_gaia3["pf"]), rrlyrae_wise_gaia3["w2mpro"], yerr=rrlyrae_wise_gaia3["w2mpro_error"], fmt="o")
plt.plot()
```

Out[115]:
```
[]
```



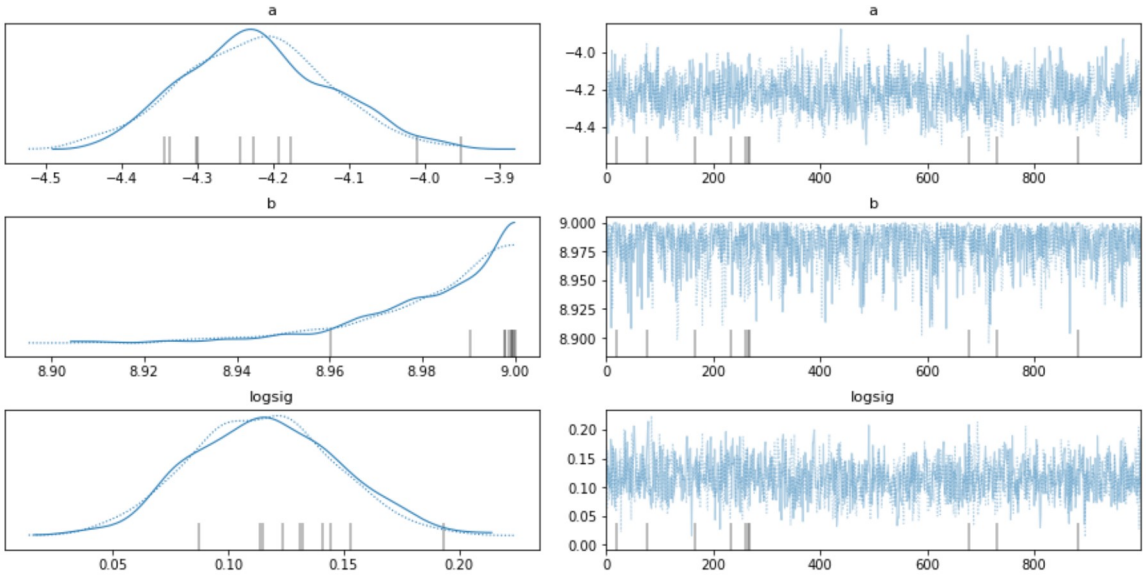
In [130]:
```
y = rrlyrae_wise_gaia3["w2mpro"]
x = rrlyrae_wise_gaia3['pf']
yerr = rrlyrae_wise_gaia3["w2mpro_error"]
samples = run_mcmc(x, y, yerr)
```

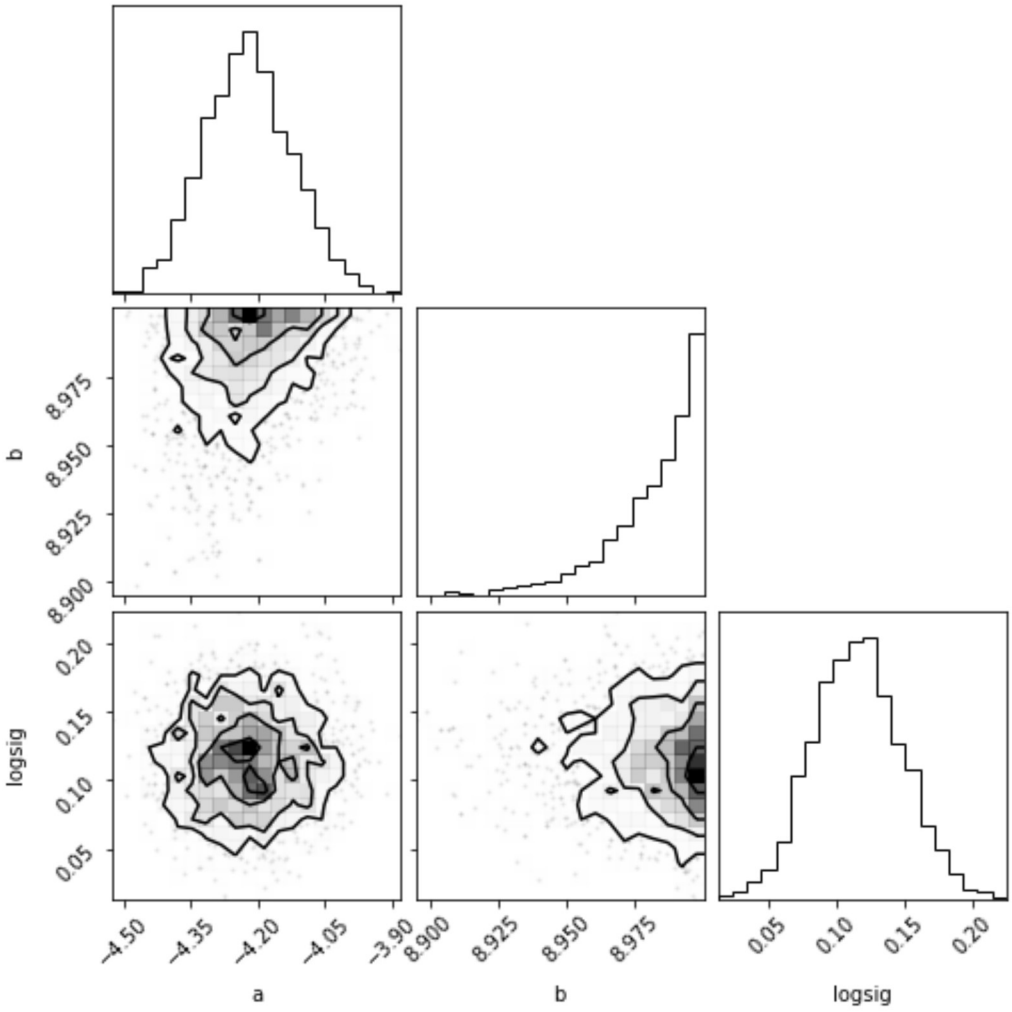
```
/home/rafferino/.local/lib/python3.6/site-packages/pymc3/sampling.py:
468: FutureWarning: In an upcoming release, pm.sample will return an
`arviz.InferenceData` object instead of a `MultiTrace` by default. Yo
u can pass return_inferencedata=True or return_inferencedata=False to
be safe and silence this warning.
  FutureWarning,
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [logsig, b, a]
```

100.00% [4000/4000 00:02<00:00 Sampling

2 chains, 10 divergences]

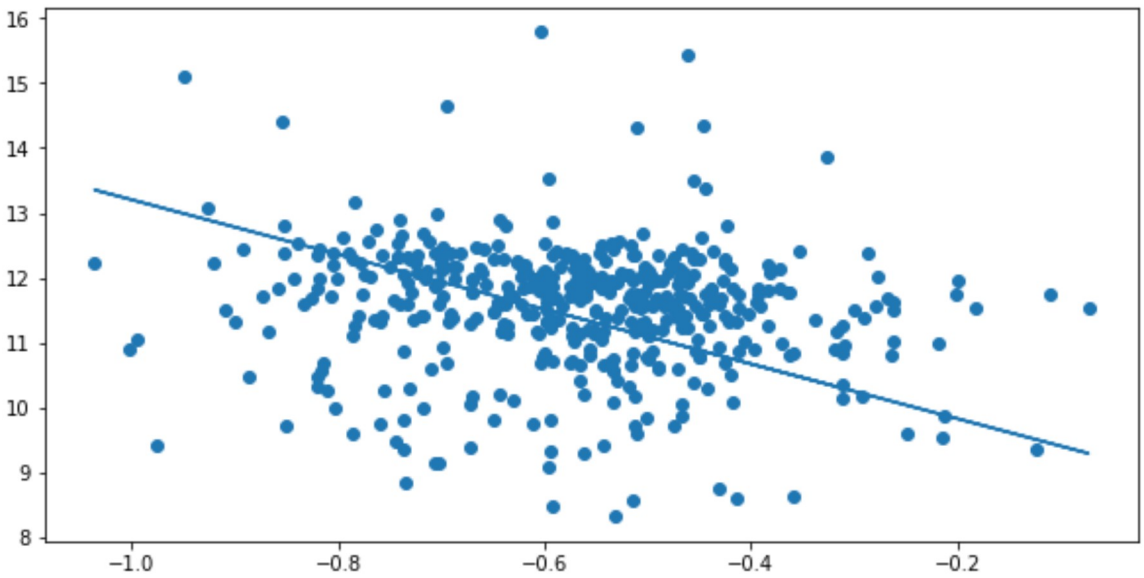
```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2
_000 draws total) took 2 seconds.
There were 7 divergences after tuning. Increase `target_accept` or re
parameterize.
There were 3 divergences after tuning. Increase `target_accept` or re
parameterize.
```




b
8.975
8.950
8.925
8.900
logsig
0.20
0.15
0.10
0.05
−4.50
−4.35
−4.20
−4.05
−3.90
a
8.900
8.925
8.950
8.975
b
0.05
0.10
0.15
0.20
logsig

In [133]:
```
a_est, b_est, logsig_est = np.mean(samples, axis=0)
print(a_est, b_est)
plt.figure(figsize=(10,5))
plt.scatter(np.log(x), y)
plt.plot(np.log(x), a_est*np.log(x)+b_est)
plt.show()
```

```
-4.222285372451877 8.980894055121672
```



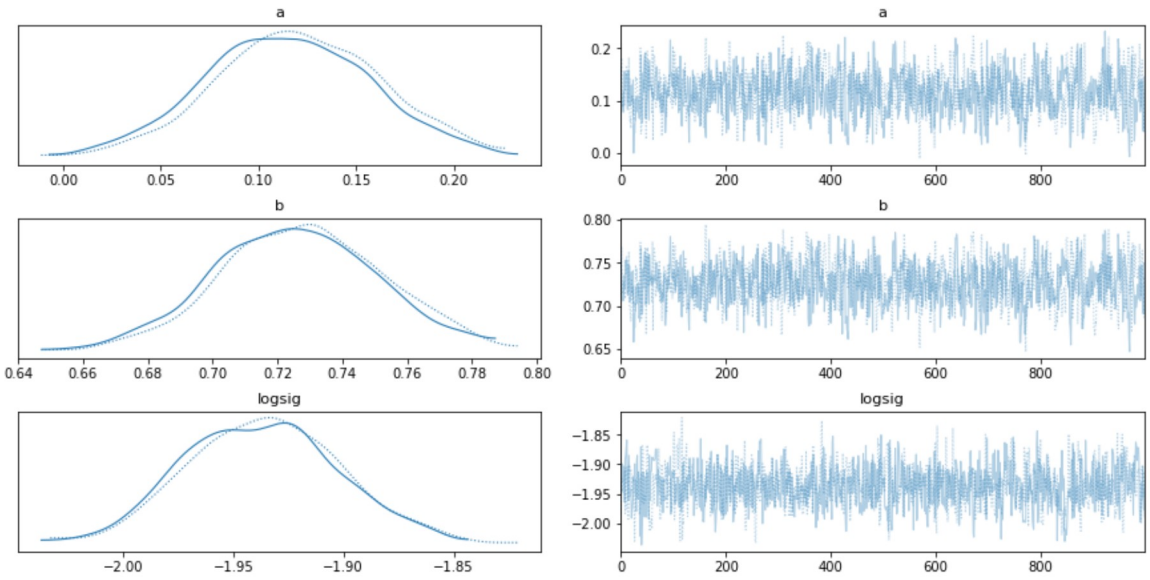
In [137]:
```
y = rrlyrae_wise_gaia3["bp_rp"]
x = rrlyrae_wise_gaia3['pf']
yerr = 1.09/np.sqrt(rrlyrae_wise_gaia3["phot_bp_mean_flux_over_error"]
**2 + rrlyrae_wise_gaia3["phot_rp_mean_flux_over_error"]**2)
samples = run_mcmc(x, y, yerr)
```

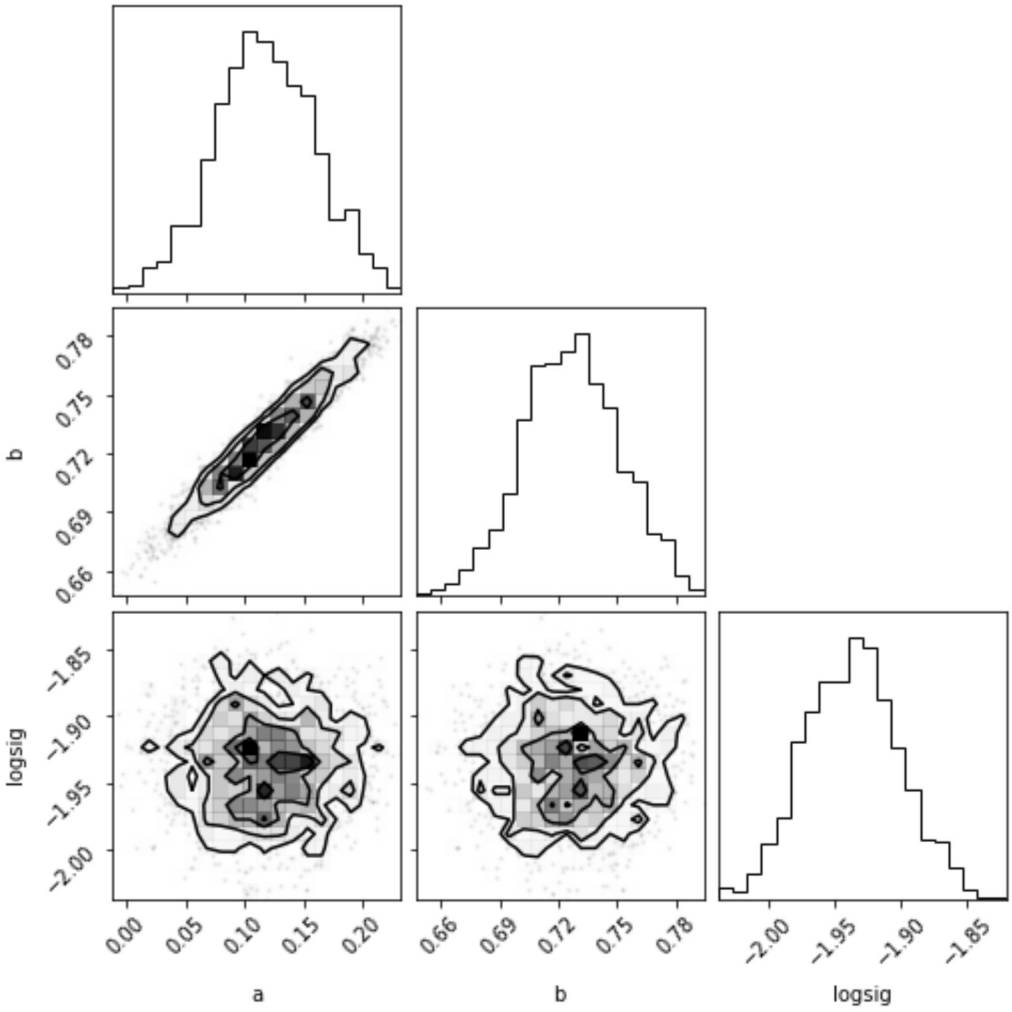
```
/home/rafferino/.local/lib/python3.6/site-packages/pymc3/sampling.py:
468: FutureWarning: In an upcoming release, pm.sample will return an
`arviz.InferenceData` object instead of a `MultiTrace` by default. Yo
u can pass return_inferencedata=True or return_inferencedata=False to
be safe and silence this warning.
  FutureWarning,
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [logsig, b, a]
```

100.00% [4000/4000 00:03<00:00 Sampling

2 chains, 0 divergences]

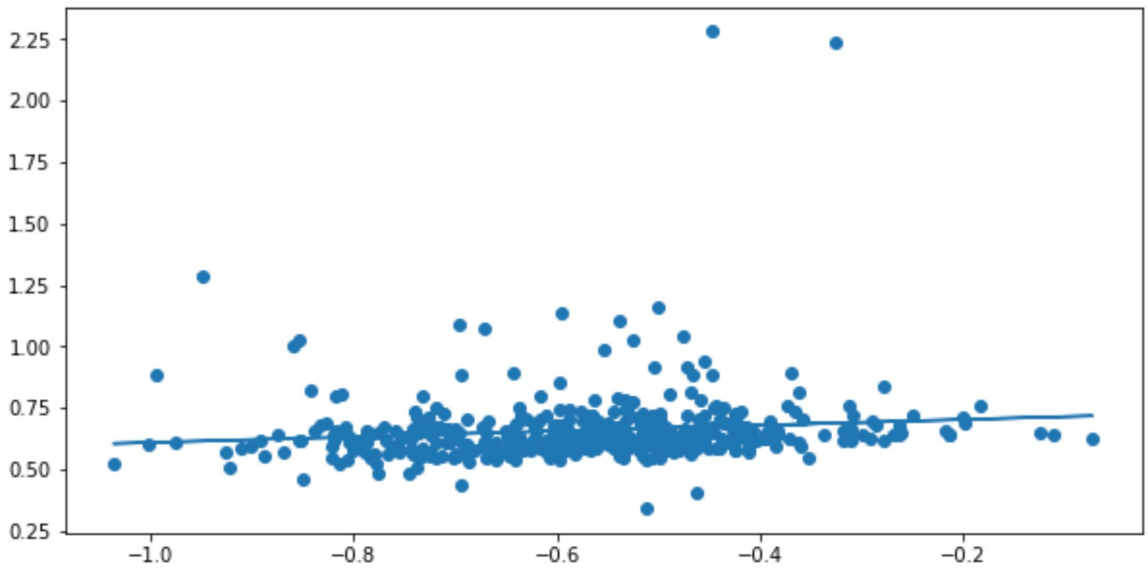
```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2
_000 draws total) took 4 seconds.
```




b
0.78
0.75
0.72
0.69
0.66
logsig
−1.85
−1.90
−1.95
−2.00
0.00
0.05
0.10
0.15
0.20
a
0.66
0.69
0.72
0.75
0.78
b
−2.00
−1.95
−1.90
−1.85
logsig

In [138]:
```
a_est, b_est, logsig_est = np.mean(samples, axis=0)
print(a_est, b_est)
plt.figure(figsize=(10,5))
plt.scatter(np.log(x), y)
plt.plot(np.log(x), a_est*np.log(x)+b_est)
plt.show()
```

```
0.11731060134689694 0.726440768620609
```



In [141]:
```
query = f"""
    SELECT *
    FROM gaiaedr3.gaia_source as gaia
    JOIN gaiaedr3.dr2_neighbourhood as link
        ON gaia.source_id = link.dr3_source_id
    JOIN gaiadr2.vari_rrlyrae as lyrae
        ON lyrae.source_id = link.dr2_source_id
    WHERE
        pf IS NOT NULL
"""
rrlyrae_all = get_gaia_query_general_cached(query)
rrlyrae_all[:10]
```

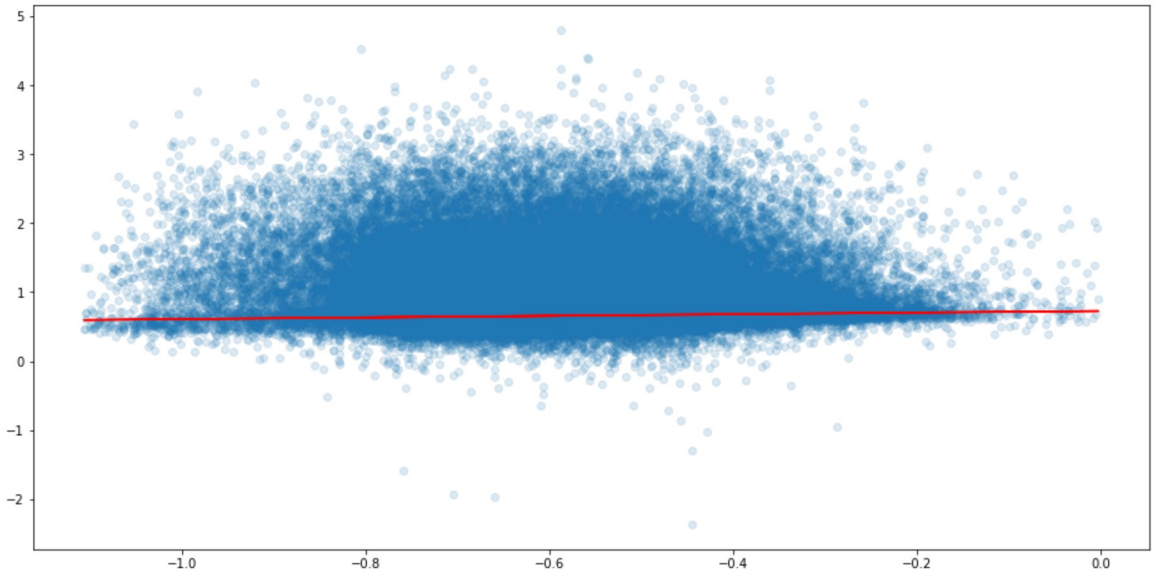
```
INFO: Query finished. [astroquery.utils.tap.core]
```

Out[141]: *Table length=10*

solution_id	designation	source_id	random_index	ref_epoch	
				yr	
int64	object	int64	int64	float64	
1636042515805110273	Gaia EDR3 5884553250832851200	5884553250832851200	826396290	2016.0	1
1636042515805110273	Gaia EDR3 5870760633362254848	5870760633362254848	64698055	2016.0	1
1636042515805110273	Gaia EDR3 5865284545760288512	5865284545760288512	41064784	2016.0	2(
1636042515805110273	Gaia EDR3 5865347428404130560	5865347428404130560	678640914	2016.0	2(
1636042515805110273	Gaia EDR3 5865198410283237632	5865198410283237632	389404988	2016.0	2(
1636042515805110273	Gaia EDR3 5865198410283237760	5865198410283237760	356811117	2016.0	2
1636042515805110273	Gaia EDR3 5877954085112175488	5877954085112175488	1422901300	2016.0	2:
1636042515805110273	Gaia EDR3 5877954085128759296	5877954085128759296	1233192230	2016.0	2:
1636042515805110273	Gaia EDR3 5877954085112175616	5877954085112175616	415323865	2016.0	2:
1636042515805110273	Gaia EDR3 5879361185109392896	5879361185109392896	694268089	2016.0	1

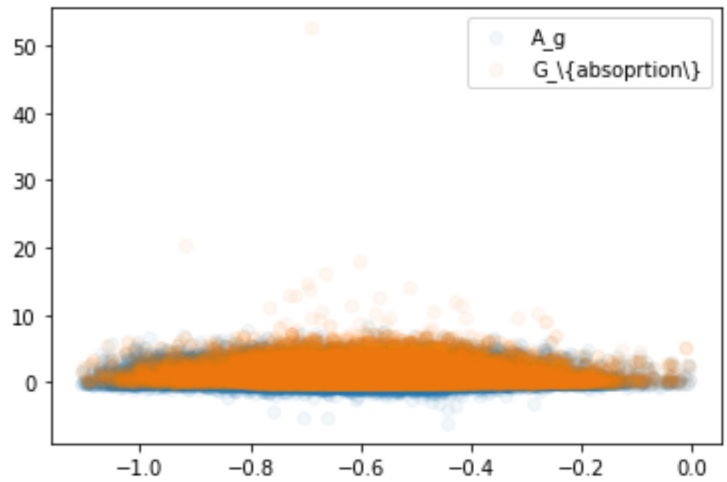
In [143]:
```
bprp_int = a_est*np.log(rrlyrae_all["pf"])+b_est
bprp_obs = rrlyrae_all["bp_rp"]
col_exc = bprp_obs - bprp_int
```

In [148]:
```
plt.figure(figsize=(16,8))
plt.scatter(np.log(rrlyrae_all["pf"]), bprp_obs, alpha=0.15)
plt.plot(np.log(rrlyrae_all["pf"]), bprp_int, c="red")
plt.show()
```



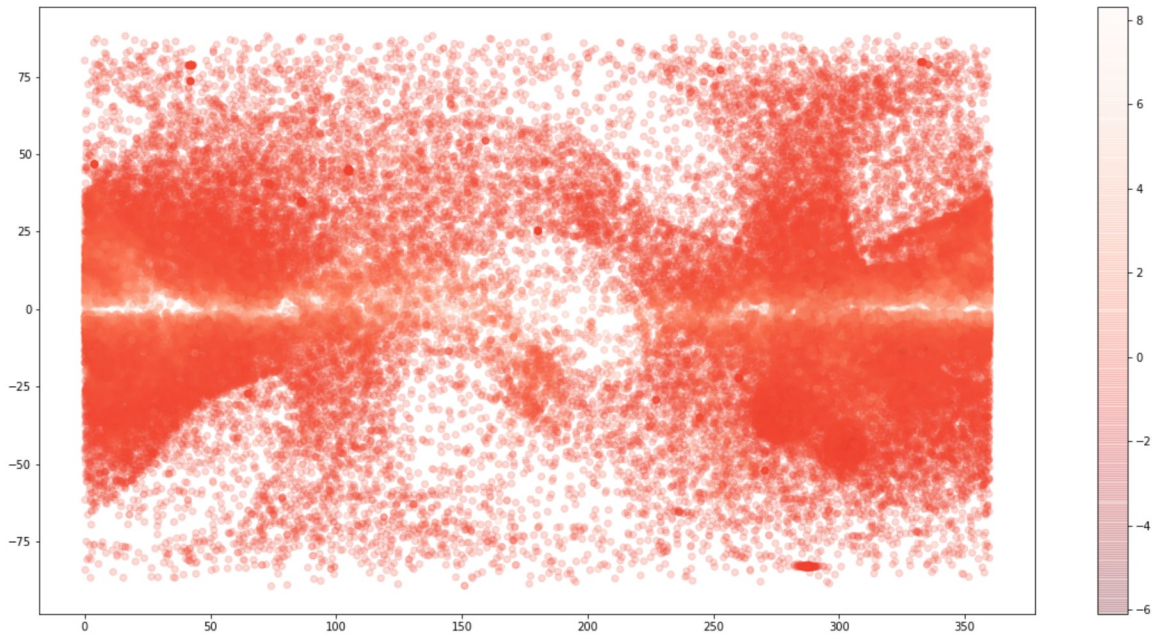
In [151]:
```
Ag = 2*col_exc
g_abso = rrlyrae_all['g_absorption']
plt.scatter(np.log(rrlyrae_all["pf"]), Ag, alpha=0.05, label="A_g")
plt.scatter(np.log(rrlyrae_all["pf"]), g_abso, alpha=0.05, label="G
_\{absoprtion\}")
plt.legend()
```

Out[151]:
```
<matplotlib.legend.Legend at 0x7f386c713e10>
```



In [155]:
```
lon = rrlyrae_all['l']
lat = rrlyrae_all['b']
plt.figure(figsize=(20,10))
plt.scatter(lon, lat, c=Ag, alpha=0.2, cmap='Reds_r')
plt.colorbar()
plt.plot()
```

Out[155]: []



In [ ]: