In [245]:
```
#Stars and stuff
from astroquery.gaia import Gaia
from astropy.io.votable import parse, parse_single_table
from astropy.timeseries import LombScargle

#Math
import numpy as np
from scipy.linalg import lstsq

#Machine Learning
from sklearn.model_selection import train_test_split

#Matplotlib
import matplotlib.pyplot as plt

#Caching
from joblib import Memory

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

In [380]:
```
def get_gaia_query_rrlyrae(num_stars = 100, num_clean_epochs = 40, con
ds=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}
            AND pf IS NOT NULL
    ''' + add
    if verbose:
        print(query)
    job = Gaia.launch_job_async(query)
    return job.get_results()

location = "./cachedir"
memory = Memory(location, verbose=0)
get_gaia_query_rrlyrae_cached = memory.cache(get_gaia_query_rrlyrae)
```

In [127]:
```
def rmse(a, b):
    return np.sqrt(np.mean(np.square(a-b)))
```

In [249]:
```
def mse(a,b):
    return np.mean(np.square(a-b))
```

In [128]:
```
def get_Gband(data,index=None,source_id=None):
    assert index!=None or source_id!=None, "Must pass in either index
or 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}")
    lc_f = f"lightcurve_files/{source_id}.xml"
    if os.path.exists(lc_f):
        votable = parse_single_table(lc_f)
    else:
        url = selected_row['epoch_photometry_url']
        votable = parse_single_table(url)

    Gband = votable.array[votable.array['band'] == 'G']
    return Gband
```

In [417]:
```
def estimate_period(Gband, 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_
frequency=2.465,
                                                              minimum_f
requency=1.044,
                                                              nyquist_f
actor=1)
    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} = {pe
riod:.5f}")
#         print(f"Period as reported by vari_rrlyrae: {recorded_perio
d:.5f}")
#         print(f"RMSE: {np.sqrt(np.mean(np.square(period - recorded_p
eriod)))}")

    return period
```

In [418]:
```
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 [250]:
```
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 [169]:
```
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 [382]:
```
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 [134]:
```
rrlyrae_100 = get_gaia_query_rrlyrae_cached(num_stars = 100, num_clean
_epochs=40)
rrlyrae_100[:10]
```

Out[134]: *Table length=10*

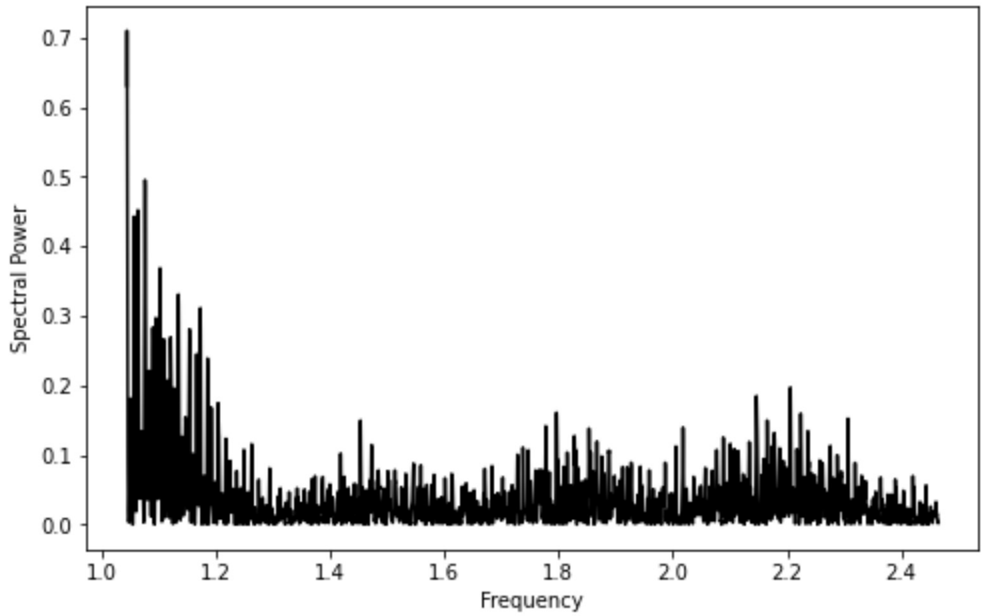
solution_id	designation	random_index	ref_epoch	ra	
			yr	deg	
int64	object	int64	float64	float64	
1635721458409799680	Gaia DR2 5866125710834119808	841033097	2015.5	212.93756378519396	1
1635721458409799680	Gaia DR2 5978435871487788288	1394969254	2015.5	256.52946118354015	0.
1635721458409799680	Gaia DR2 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

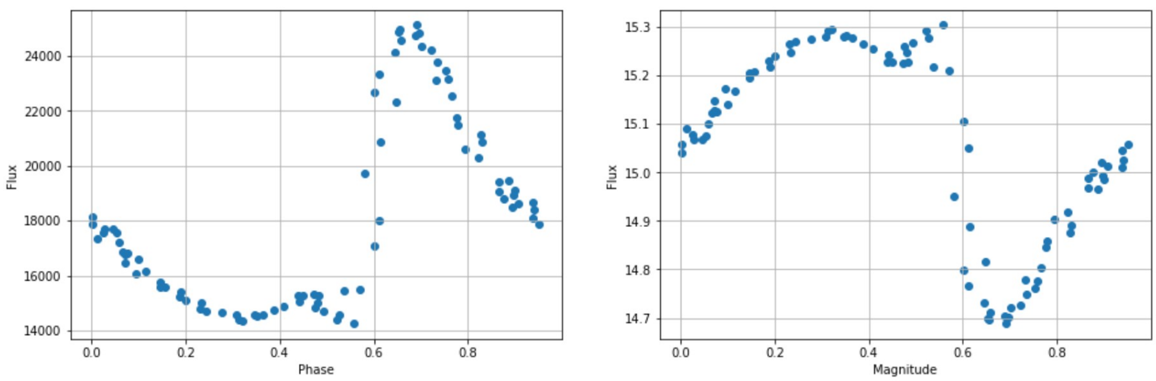
In [135]:
```
for row in rrlyrae_100:
    url = row['epoch_photometry_url']
    source_id = row['source_id']
    dest = f"lightcurve_xmls/{source_id}.xml"
    if os.path.exists(dest):
#         print("It exists!")
        continue
    votable = parse(url)
    votable.format = 'binary'
    with open(dest, 'w') as d:
        votable.to_xml(d)
```

In [240]:
```
selected_index = 6
selected_Gband = get_Gband(rrlyrae_100, selected_index)
```

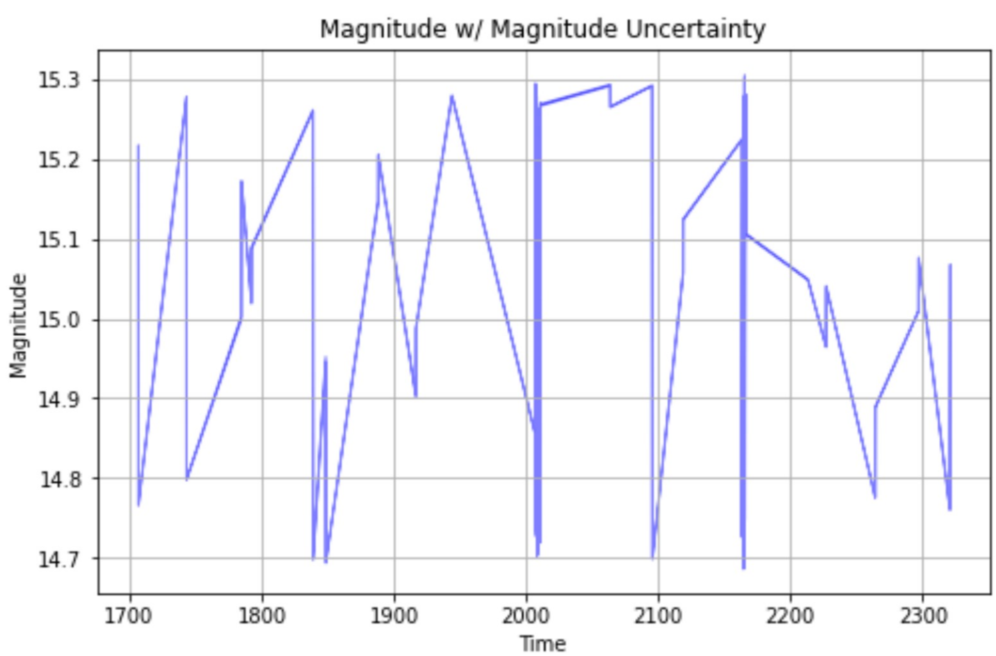
```
Analyzing star with source_id: 5813181197970338560
```

In [419]:
```
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_perio
d)))}")
plot_magnitude(selected_Gband)
```





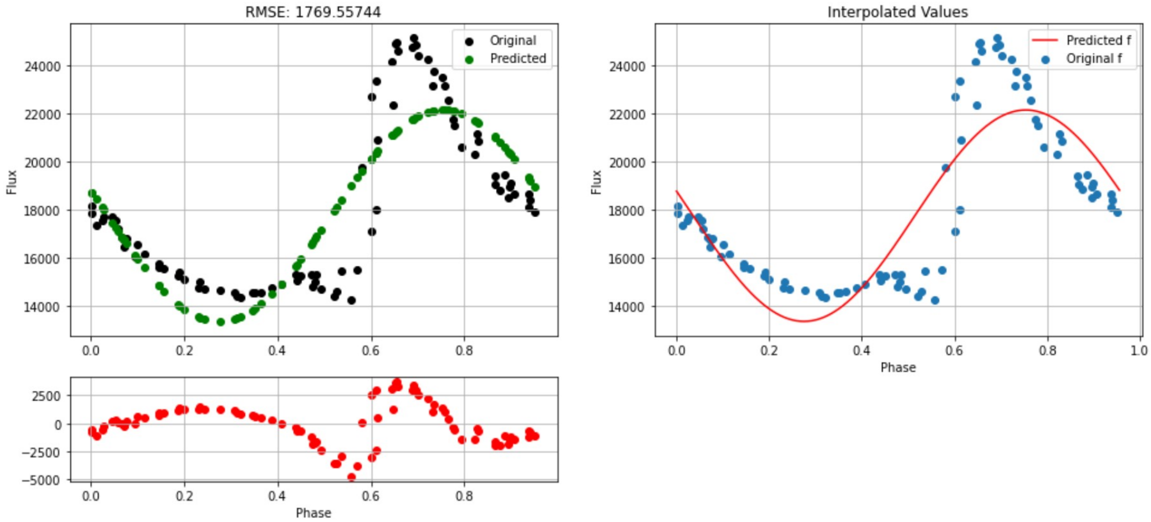

```
Estimated period: 1/1.04433 = 0.95756
Period as reported by vari_rrlyrae: 0.95765
RMSE: 9.488197693585665e-05
```



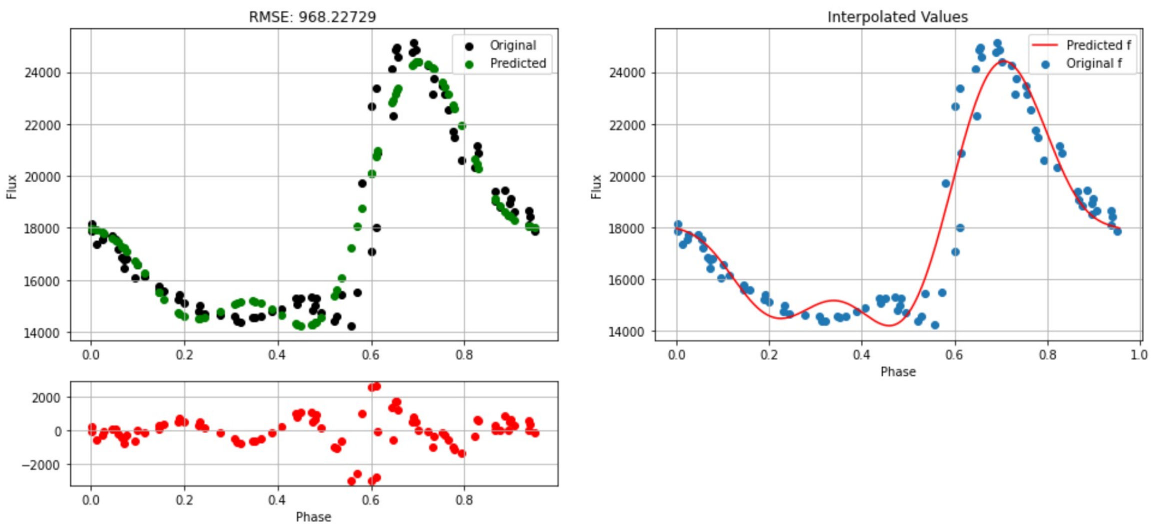
```
Estimated mean: 15.070074092017139
```

In [383]:
```
k_vals = [1,3,5,7,9]
flux,time=selected_Gband['flux'], selected_Gband['time']
period=estimate_period(selected_Gband)
omega=2*np.pi/period
time_interp=np.arange(np.min(time), np.max(time), 1)
for k in k_vals:
    print(f"Analyzing fit for k={k}...")

    X = setup_fit(flux, time, omega, k)
    beta = lstsq(X, f.data)[0]
#     print(rmse(np.dot(X, beta), f))
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2, figsize=(16,7),
gridspec_kw={'height_ratios':[3,1]})
    # plt.plot(t, f, label="Truth")
    # plt.plot(t, X.dot(b), label="Predicted")
    ax1.scatter(t%period, flux, label="Original",color='black')
    ax1.scatter(t%period, X.dot(beta), label="Predicted", color='green
')
    ax1.set(title=f"RMSE: {rmse(f, X.dot(beta)):.5f}",ylabel="Flux")
    ax1.legend()
    ax1.grid()

    ax3.scatter(t%period, flux-X.dot(beta), label="Residual",color='re
d')
    ax3.set(xlabel="Phase")
    ax3.grid()

    A0=beta[0]
    a=beta[1:k+1]
    b=beta[k+1:]
    f_interp = pseudo_fourier(omega=2*np.pi/period, t=time_interp, A0=
A0, a=a, b=b)
    phase_interp, flux_interp = sort_by_phase(time_interp%period, f_in
terp)
    ax2.scatter(time%period, flux, label="Original f")
    ax2.plot(phase_interp, flux_interp, label="Predicted f", color="re
d")
    ax2.set(title="Interpolated Values", xlabel="Phase",ylabel="Flux")
    ax2.legend()
    ax2.grid()
    fig.delaxes(ax4)
    plt.show()
```

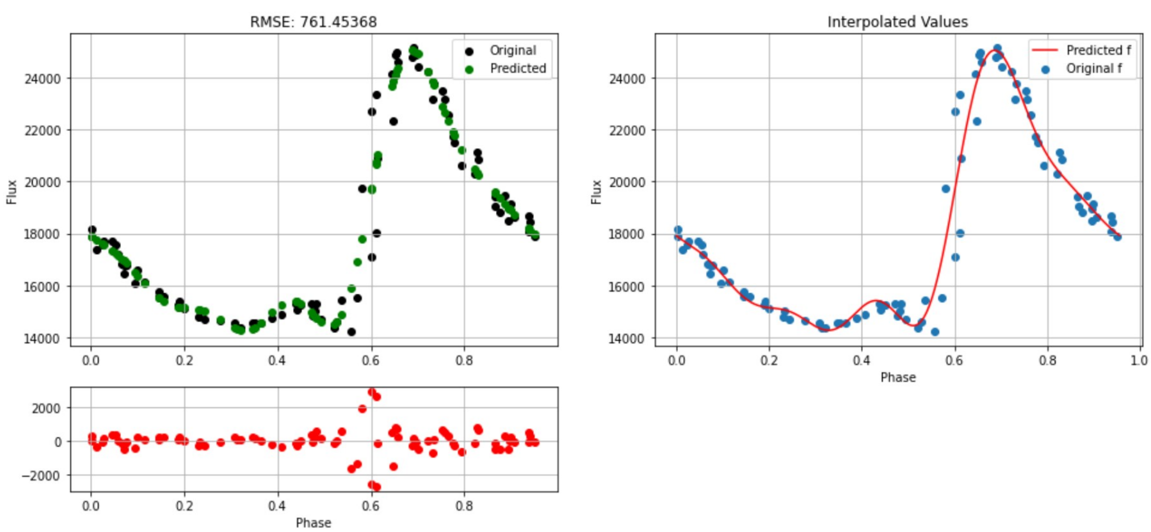
```
Analyzing fit for k=1...
```



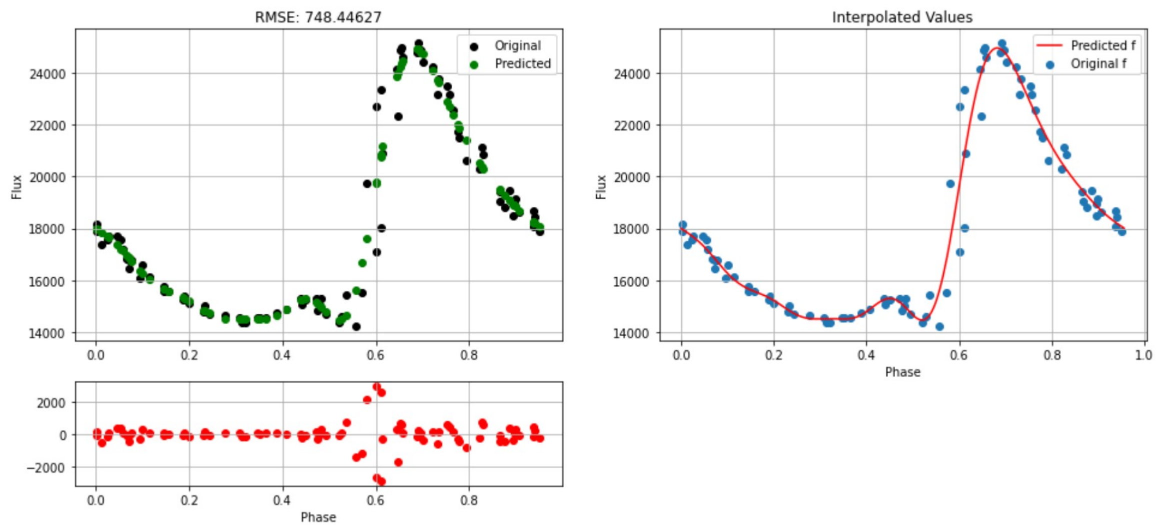

```
Analyzing fit for k=3...
```



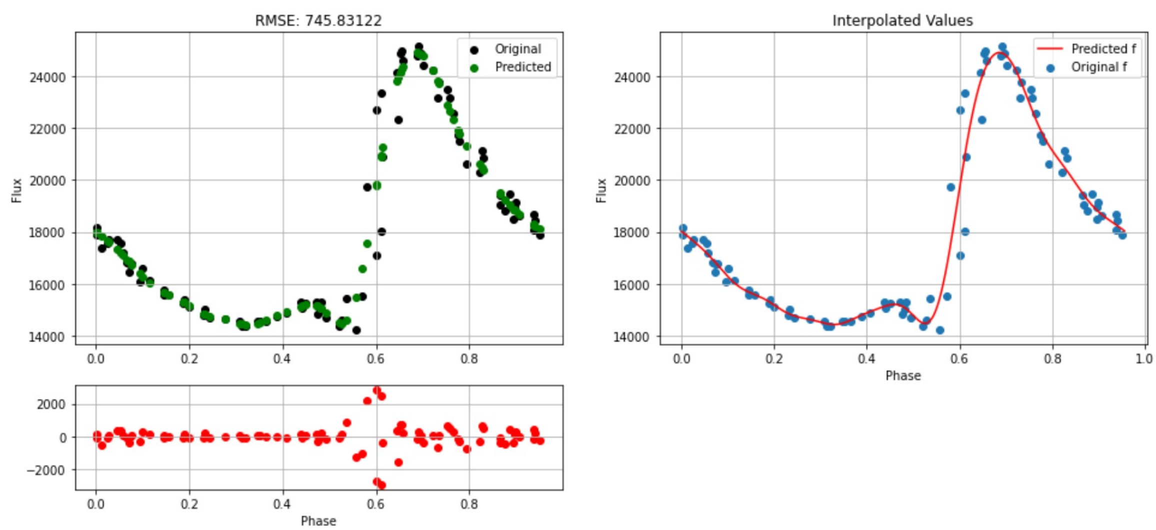

```
Analyzing fit for k=5...
```




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

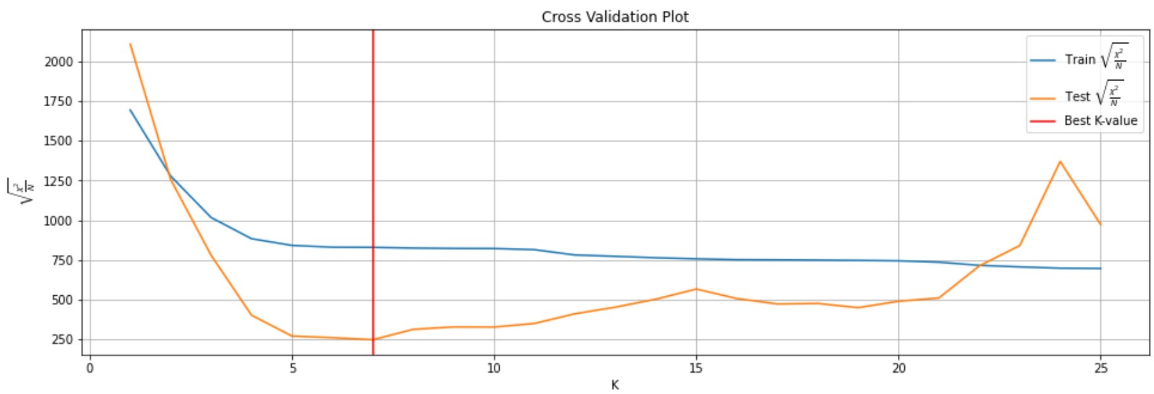



```
Analyzing fit for k=9...
```



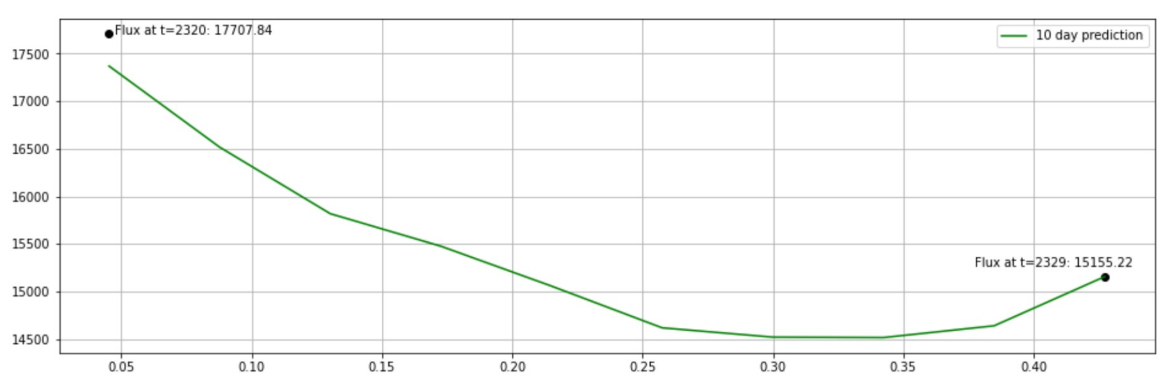
In [367]:
```
k_vals = np.arange(1,25+1)
train_mses = np.zeros(len(k_vals))
test_mses = np.zeros(len(k_vals))
time_train, time_test, flux_train, flux_test = train_test_split(time,
flux, test_size=0.2)
```

In [420]:
```python
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 [421]:
```python
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_be
st)
phase_extrap, flux_extrap = sort_by_phase(time_extrap%period, pred_ext
rap)
fig,ax = plt.subplots(figsize=(16,5))
ax.scatter(last_time%period, last_flux,color='black')
ax.annotate(f"Flux at t={last_time:.0f}: {last_flux:.2f}", (last_time%
period+0.002, last_flux))
ax.plot(phase_extrap, flux_extrap, color='green', label="10 day predic
tion")
ax.scatter(phase_extrap[-1], flux_extrap[-1],color='black')
ax.annotate(f"Flux at t={time_extrap[-1]:.0f}: {flux_extrap[-1]:.2f}",
(phase_extrap[-1]-0.05, flux_extrap[-1]+100))
ax.grid()
ax.legend()
plt.show()
```

```
17707.83890462764 2320.202939518489
```



In [ ]: