In [1]:

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
import statsmodels.formula.api as smf
from scipy import stats

%matplotlib inline
sns.set(style="white", font_scale=2.0)
```

In [2]:

```
# read in data and get x and y columns
data = pd.read_csv('brunhild.txt',delimiter='\t')
x = data['Hours']
y = data['Sulfate']

data.head()
```

Out[2]:

	Hours	Sulfate
0	2	15.11
1	4	11.36
2	6	9.77
3	8	9.09
4	10	8.48

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

a) Prepare a plot showing the data points and the regression line in log-log coordinates.

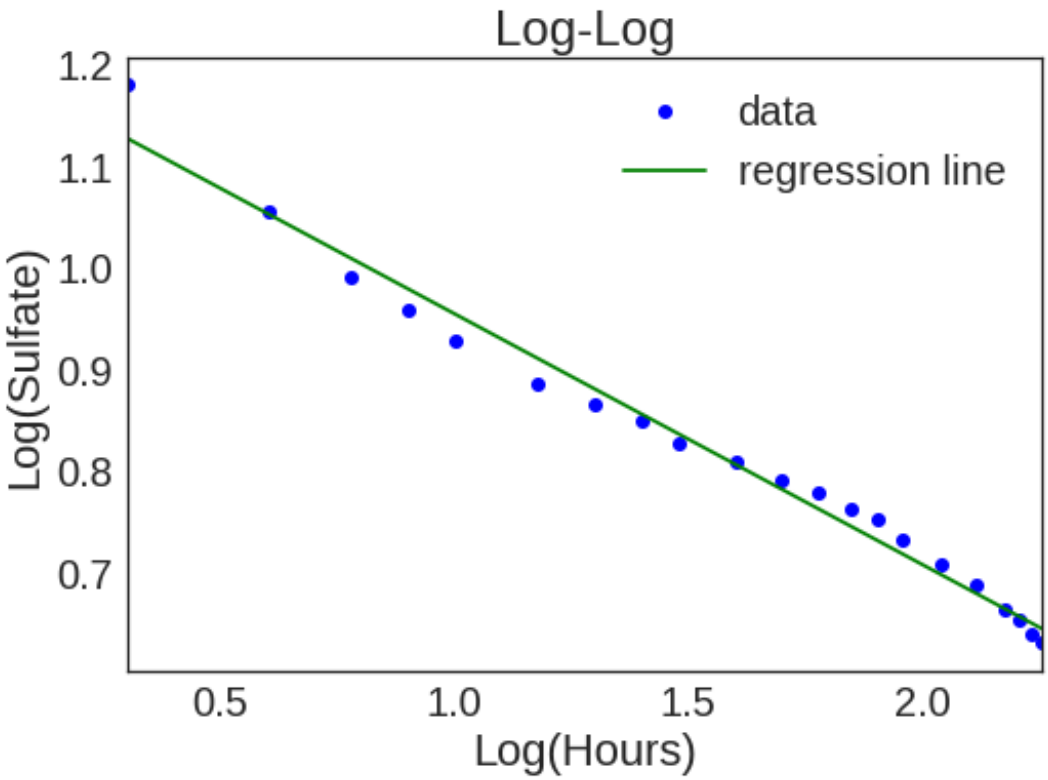
In [3]:

```
# get x and y in log coordinates
logx = np.log10(x)
logy = np.log10(y)
```

```
# fit linear regression model
p1 = np.poly1d(np.polyfit(logx,logy,1))

# plot data points
ax = plt.subplot()
plt.plot(logx,logy,'o',color='b',label='data')
ax.set(xlabel='Log(Hours)',ylabel='Log(Sulfate)',title='Log-Log')

# plot regression line
plt.plot(logx,p1(logx), c='g', label='regression line')

ax.set_xlim([min(logx),max(logx)])
ax.legend(loc=1)
```

Out[3]:

```
<matplotlib.legend.Legend at 0x7fe1689147b8>
```



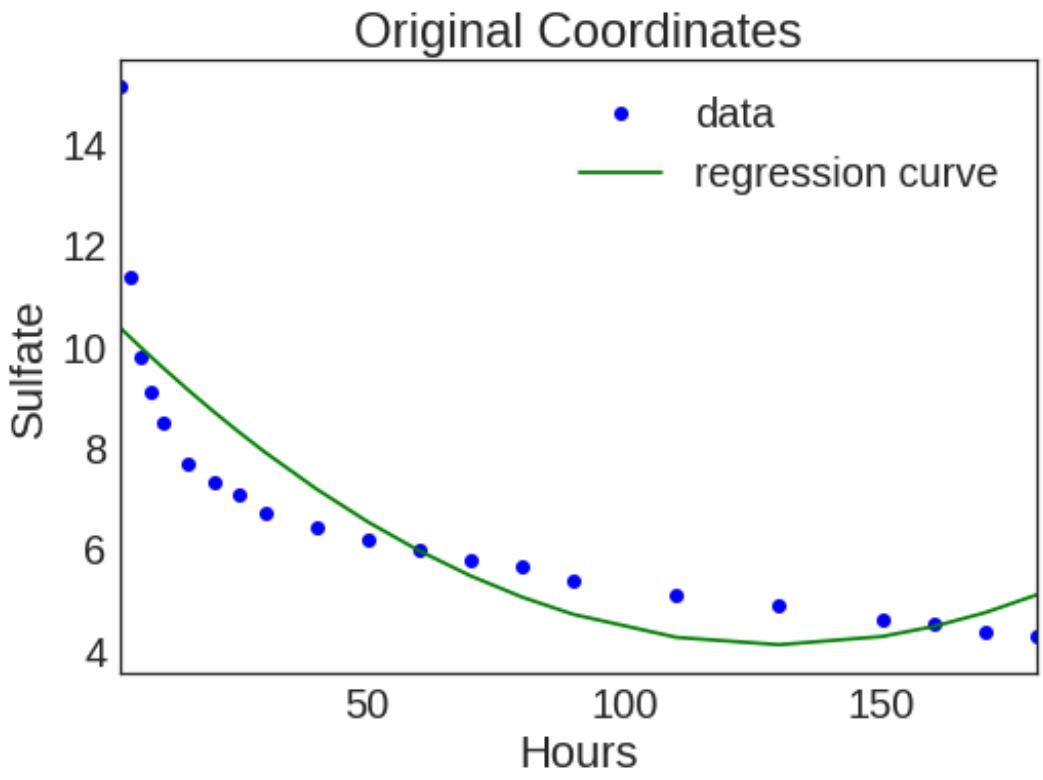
Prepare a plot showing the data points and the regression curve in the original coordinates.

In [4]:

```
# get curve fit
p2 = np.poly1d(np.polyfit(x,y,2))
# plot data points
ax = plt.subplot()
plt.plot(x,y,'o',color='b',label='data')
ax.set(xlabel='Hours',ylabel='Sulfate',title='Original Coordinates')

# plot regression line
plt.plot(x,p2(x),c='g', label='regression curve')

ax.set_xlim([min(x),max(x)])
ax.legend(loc=1)
```

Out[4]:

```
<matplotlib.legend.Legend at 0x7fe168872f98>
```



In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

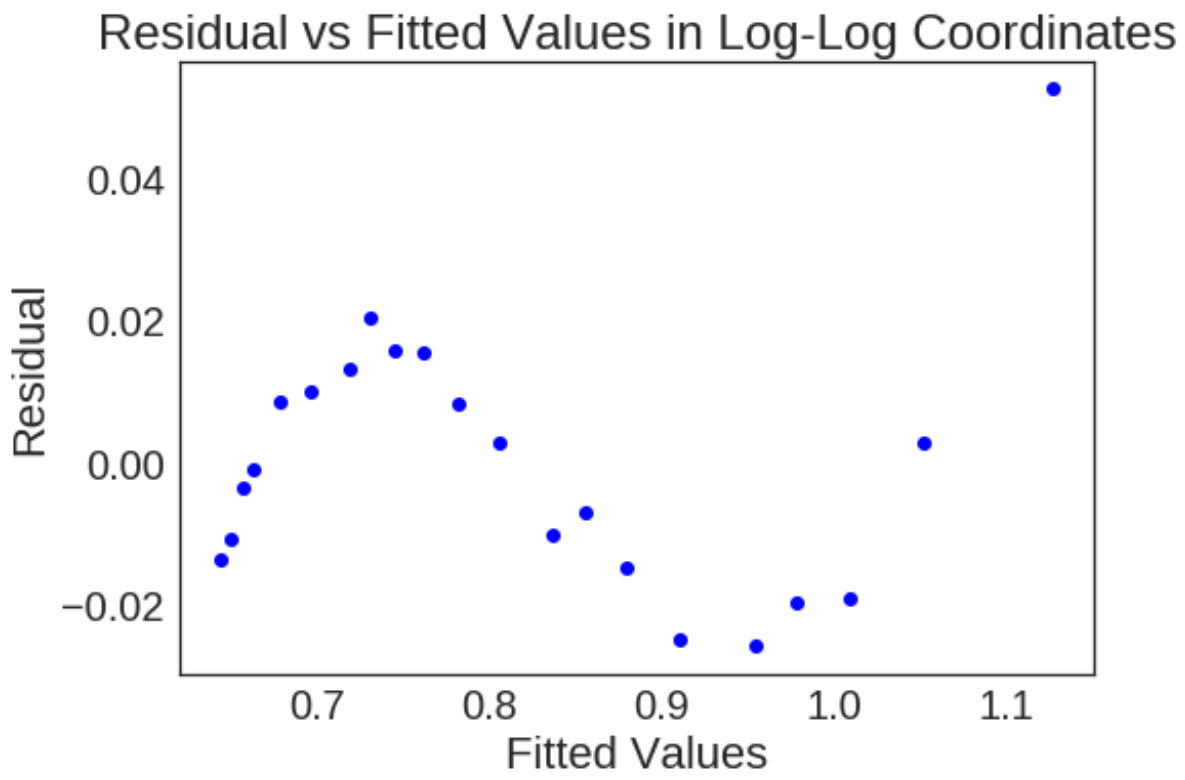
c) Plot the residual against the fitted values in log-log coordinates.

In [5]:

```
# residual in log-log coordinates
ax = plt.subplot()
log_yr = logy - p1(logx)

# yf is the fitted values and log_yr is the residual
plt.plot(p1(logx),log_yr,'o',color='b')

ax.set(xlabel='Fitted Values',
       ylabel='Residual',
       title='Residual vs Fitted Values in Log-Log Coordinates')
```

Out[5]:

```
[<matplotlib.text.Text at 0x7fe16883d080>,
 <matplotlib.text.Text at 0x7fe1688f32e8>,
 <matplotlib.text.Text at 0x7fe1687e75f8>]
```



In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

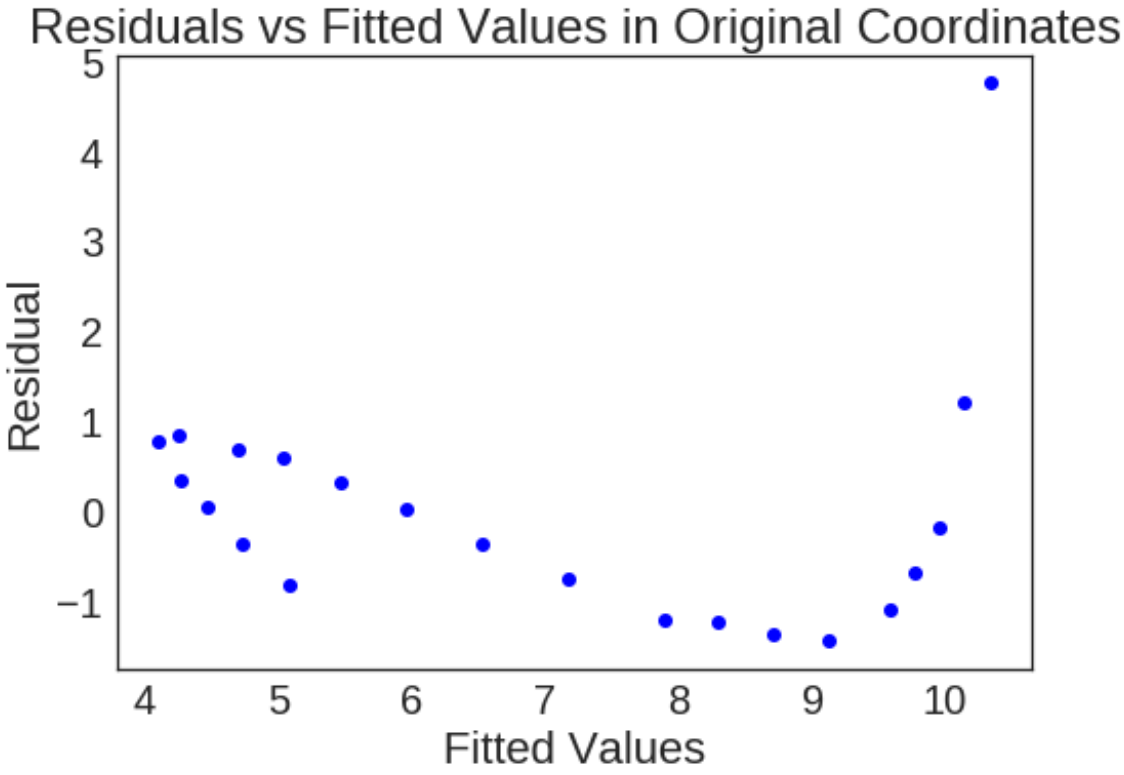
Plot the residual against the fitted values in original coordinates.

In [6]:

```
# residuals in original coordinates
ax = plt.subplot()
yr = y - p2(x)

ax = plt.subplot()

# yf2 is the fitted values and yr is the residual
plt.plot(p2(x),yr,'o',color='b')

ax.set(xlabel='Fitted Values',
       ylabel='Residual',
       title='Residuals vs Fitted Values in Original Coordinates')
```

Out[6]:

```
[<matplotlib.text.Text at 0x7fe1687c7a90>,
 <matplotlib.text.Text at 0x7fe168944828>,
 <matplotlib.text.Text at 0x7fe168770080>]
```



d) Use your plots to explain whether your regression is good or bad and why

In [7]:

```
# find r-squared values for log-log and original plots
def rsquared(x, y):
    slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
    return r_value**2
print(rsquared(logx,logy))
print(rsquared(x,y))
```

```
0.983925093101
0.586567257519
```

The r^2 value for the regression fit in log-log coordinates is much higher than the one for the curved fit in orginal coordinates, which means the linear regression fit in log-log coordinates is a better fit.

In [ ]:

```

```