HW 28

In [1]:
```
import jax
import jax.numpy as jnp
import numpy as np
import pandas as pd
# import PIL
# import scipy
# import sympy as sp
from matplotlib import pyplot as plt
from scipy.optimize import minimize
```

The file *ENSO.txt* contains weather data. The independent variable is the monthly averaged atmospheric pressure differences between Easter Island and Darwin, Australia and the dependent variable is time in months. This difference drives the trade winds in the southern hemisphere. The data contains three cyclical signals and should be fit to the model

$$y_\theta(t) = \theta_1 + \theta_2\cos(2\pi t/12) + \theta_3\sin(2\pi t/12) + \theta_5\cos(2\pi t/\theta_4) \\ +\theta_6\sin(2\pi t/\theta_4) + \theta_8\cos(2\pi t/\theta_7) + \theta_9\sin(2\pi t/\theta_7)$$

The annual cycle is the strongest and corresponds to parameters θ_2 and θ_3 in the model. (Observe that these parameters scale Fourier terms with periods of 12 months.) Two other cycles with unknown periods (given by θ_4 and θ_7) are described by the model. These cycles correspond to the El Niño and the Southern Oscillation.

One of the challenges in nonlinear fits is finding a reasonable starting guess for an optimization algorithm. In this problem you will use the data to reason about reasonable initial guesses for the parameters.

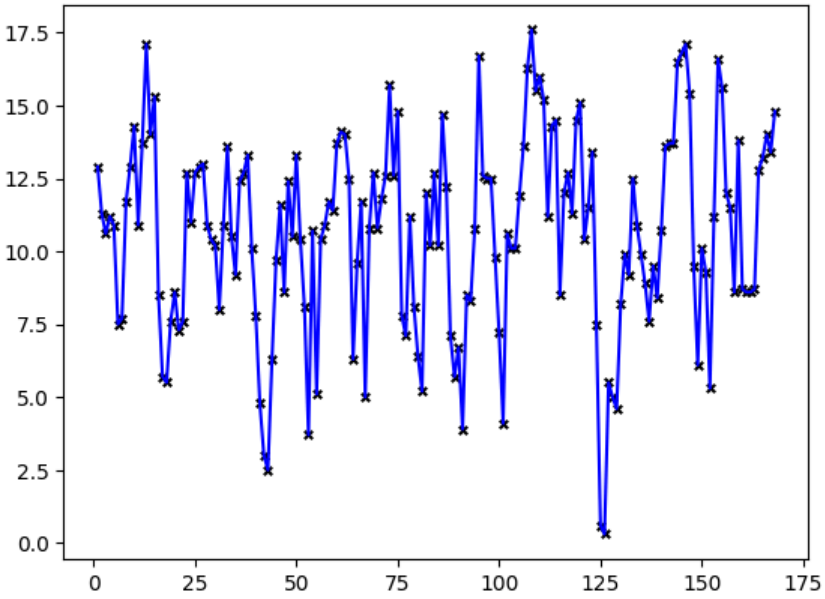
In [2]:
```
data = pd.read_csv("enso.txt", sep=r'\s+', header=None).values
t = data[:, 1] # months
y = data[:, 0] # atmospheric pressure
```

In [3]:
```
def f(t: jnp.ndarray, th1: float, th2: float, th3: float, th4: float, th5: float, th6: float, th7: float, th8: float, th9: float) -> jnp.ndarray:
    out = jnp.array(
        [
            jnp.full(t.shape, th1)+
            th2*jnp.cos(2*jnp.pi*t*(1/12))+   # annual cycle
            th3*jnp.sin(2*jnp.pi*t*(1/12))+   # annual cycle
            th5*jnp.cos(2*jnp.pi*t*(1/th4))+  # el nino
            th6*jnp.sin(2*jnp.pi*t*(1/th4))+  # el nino
            th8*jnp.cos(2*jnp.pi*t*(1/th7))+  # southern oscillation
            th9*jnp.sin(2*jnp.pi*t*(1/th7))   # southern oscillation
        ]
```

```
    ).T
    return out
```

In [4]:
```
# let's plot data for an initial visualization
plt.scatter(t, y, color="black", marker="x", s=15)
plt.plot(t, y, color="blue")
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x1181487d0>]
```



a. Argue that θ_1 corresponds to the average value of the data and use that to suggest a reasonable starting guess for θ_1.

In [5]:
```
# Theta_0 should correspond the the average of the average atmospheric pressure because it shifts the
# whole distribution of outcomes equally.

th1_init = jnp.mean(y) # near 10.6
```

b. Argue that θ_2, θ_3, θ_5, θ_6, θ_8 and θ_9 must correspond to the size of the variation in the signal from the mean and suggest a reasonable range of starting guesses for these parameters.

In [6]:
```
# The parameters applied outside the sin/cosine functions listed in (b) must correspond to the variation in signal
# attributable to effects of time because they are associated with fluctuations from the mean associated with a
# certain wavelength.

# Based on this, and in considering the plot above, I recommend:
th2_init = jnp.float32(7) # annual fluctuation, could be 6-8
```

```
th3_init = jnp.float32(7) # annual fluctuation, could be 6-8
th5_init = jnp.float32(5) # notable fluctuation near month 125, could be 4-6
th6_init = jnp.float32(5) # notable fluctuation near month 125, could be 4-6
th8_init = jnp.float32(5) # general fluctuation over multiple years, as in month 0-75, could be 4-6
th9_init = jnp.float32(5) # general fluctuation over multiple years, as in month 0-75, could be 4-6
```

c. Argue that θ_4 and θ_7 correspond to the period of the two other signals in the data. What is the shortest period you could reasonably infer from data? What is the longest period you could reasonably infer from the data? Based on these arguments, suggest a range of starting guesses for these parameters.

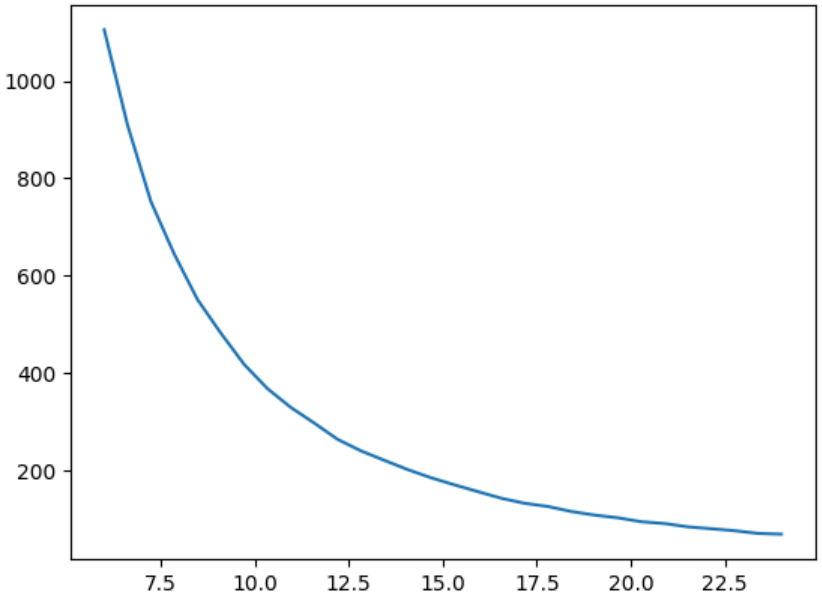
In [7]:
```
# I promted ChatGPT with "teach me about the time fluctuation of the souther oscilation and el nino."

# I learned that the el nino-southern oscilation fluctuates over the course of 2-7 years, but I see
# patterns that may be associated with a shorter 6-18 month period. Additionally,
# there is a general pattern that has a period near 75 in the data, as well as another downward
# spike around month 125. Potentially we could also assume some signal that is general across
# the whole set, with a period of twice the length of the data.

# Based on this, I suggests a range near 6-24 months
th4_init = jnp.float32(18)
th7_init = jnp.float32(18)
```

d. Denote the model map for this learning problem by $\boldsymbol{F}(\theta)$. Evaluate the $\|\partial \boldsymbol{F}/\partial\theta_4\|$ and $\|\partial \boldsymbol{F}/\partial\theta_7\|$ for values of θ_4 and θ_7 that are within the range you suggested in the previous part and compare them to values evaluated well outside that range. What do you observe about the norm of these derivatives when the parameters θ_4 and θ_7 are pushed to extreme values outside your suggested range??

In [8]:
```
partial_th4 = jax.jacrev(f, argnums=4)
partial_th7 = jax.jacrev(f, argnums=7)
```

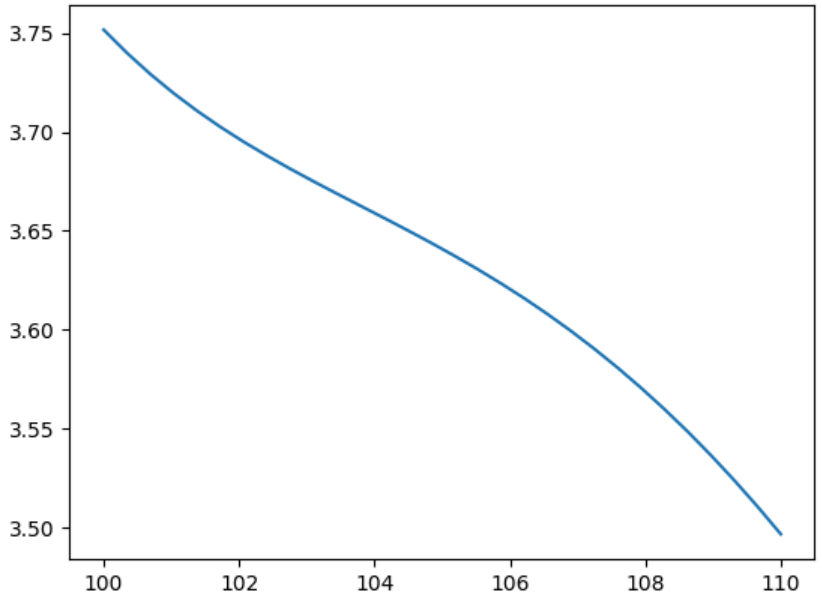
In [9]:
```
# Getting norms of gradient with respect to theta4 based on varied theta 4 inside our plausible range
plotting_norms = np.ones((30, 2))
for i, th4 in enumerate(jnp.linspace(6, 24, 30)):
    norm = jnp.linalg.norm(partial_th4(t, th1_init, th2_init, th3_init, th4, th5_init, th6_init, th7_init, th8_init, th9_init), ord=2)
    plotting_norms[i] = np.array([th4, norm])

# plot norms over theta 4
plt.plot(plotting_norms[:, 0], plotting_norms[:, 1])
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x1189a6c10>]
```



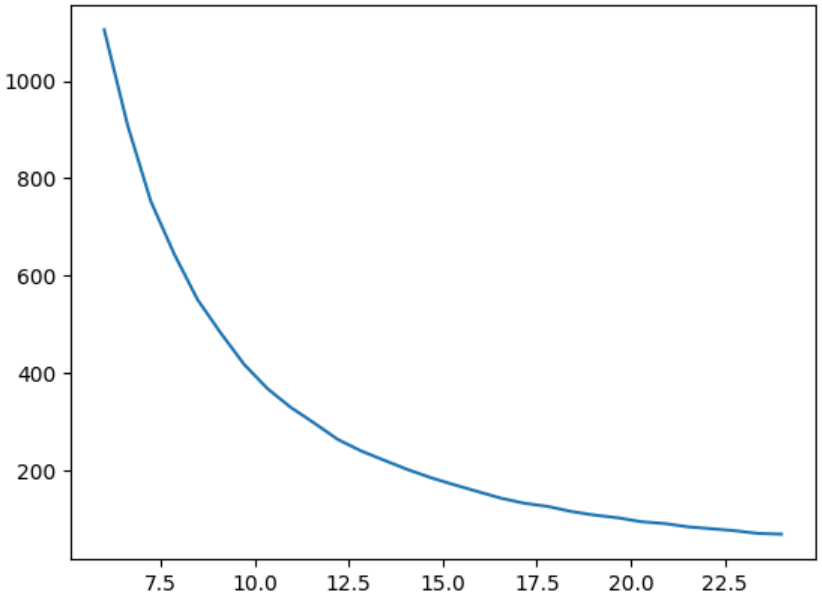
In [10]:
```
# Getting norms of gradient with respect to theta4 based on varied theta 4 outside our plausible range
plotting_norms = np.ones((30, 2))
for i, th4 in enumerate(jnp.linspace(100, 110, 30)):
    norm = jnp.linalg.norm(partial_th4(t, th1_init, th2_init, th3_init, th4, th5_init, th6_init, th7_init, th8_init, th9_init), ord=2)
    plotting_norms[i] = np.array([th4, norm])

# plot norms over theta 4
plt.plot(plotting_norms[:, 0], plotting_norms[:, 1])
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x118af1590>]
```



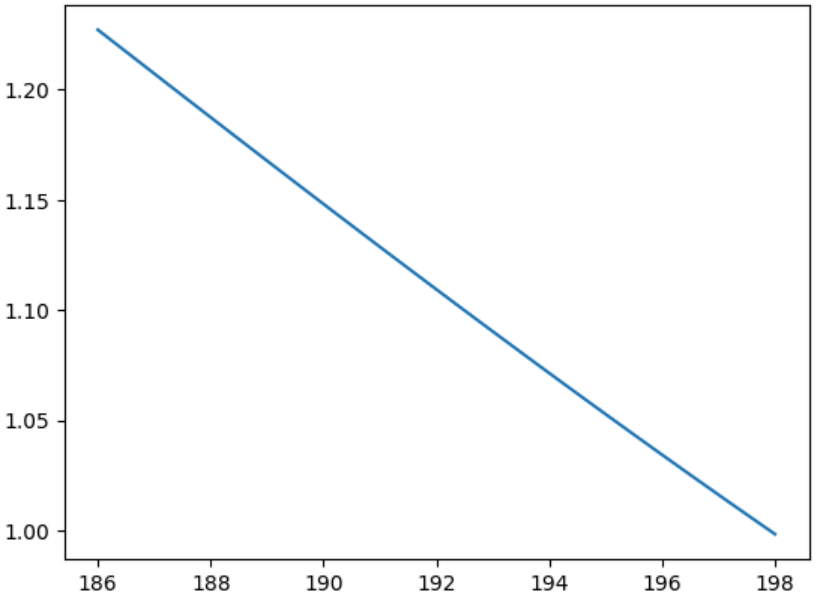
In [11]:
```
# Getting norms of gradient with respect to theta7 based on varied theta 7 inside our plausible range
plotting_norms = np.ones((30, 2))
for i, th7 in enumerate(jnp.linspace(6, 24, 30)):
    norm = jnp.linalg.norm(partial_th7(t, th1_init, th2_init, th3_init, th4_init, th5_init, th6_init, th7, th8_init, th9_init), ord=2)
    plotting_norms[i] = np.array([th7, norm])

# plot norms over theta 4
plt.plot(plotting_norms[:, 0], plotting_norms[:, 1])
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x118b67390>]
```



In [12]:
```
# Getting norms of gradient with respect to theta7 based on varied theta 7 outside our plausible range
plotting_norms = np.ones((30, 2))
for i, th7 in enumerate(jnp.linspace(186, 198, 30)):
    norm = jnp.linalg.norm(partial_th7(t, th1_init, th2_init, th3_init, th4_init, th5_init, th6_init, th7, th8_init, th9_init), ord=2)
    plotting_norms[i] = np.array([th7, norm])

# plot norms over theta 4
plt.plot(plotting_norms[:, 0], plotting_norms[:, 1])
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x118bf91d0>]
```



In [13]:
```
# I observe that the normed gradients are much greater for good theta values
# of the parameters. This suggests that the cost surface is very steep at those
# locations in parameter space, which may be indicative that these are near a basin
```

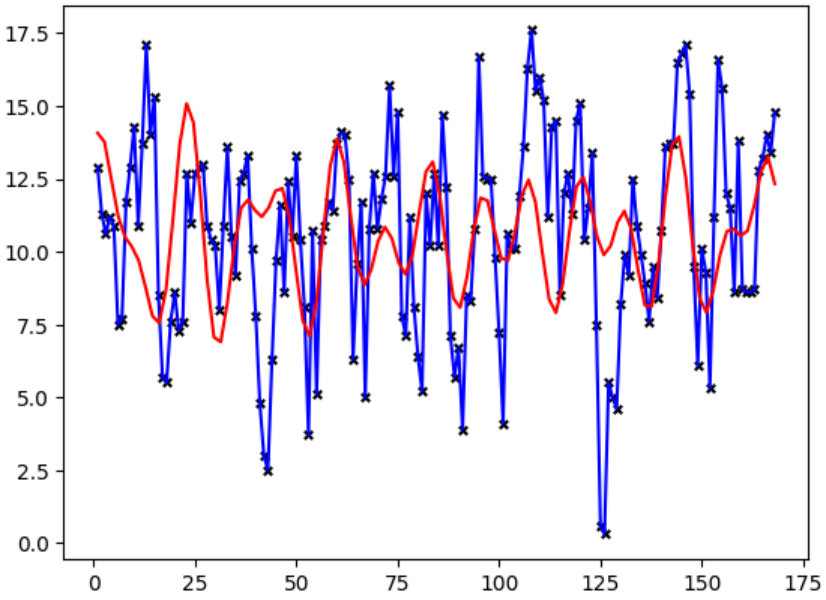
e. Fit this model to the data. Begin your fit from a few different starting points, including points near your intial guesses for θ_4 and θ_7, as well as those far away. Can you choose initial parameter values for which your fitting algorithms fail.

In [14]:
```
def c(thet: jnp.ndarray, t: jnp.ndarray, y: jnp.ndarray): # cost for minimization
    y_hat = f(t, *thet)
    cost = jnp.linalg.norm(y_hat-y, ord=2)
    return cost

t_plotting = jnp.linspace(t.min(), t.max(), 100) # for plotting
```

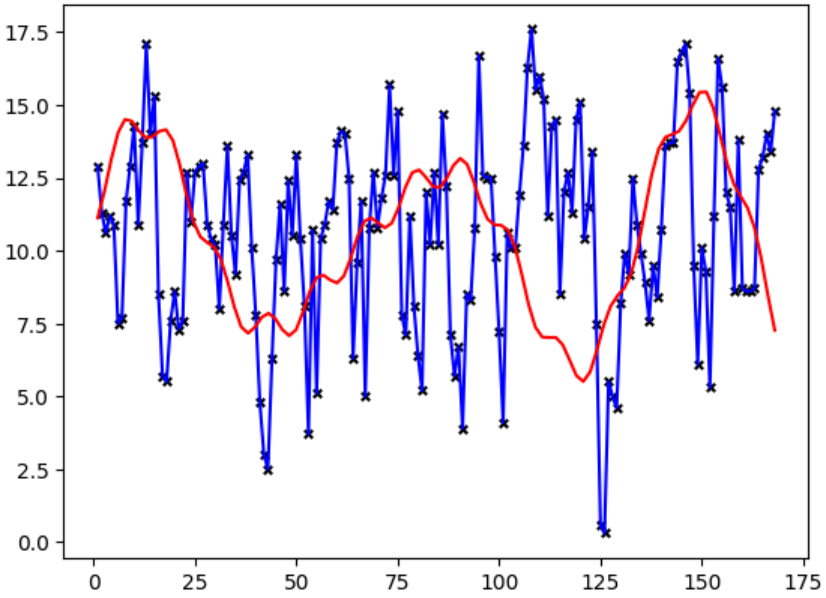
In [15]:
```
inits = [
    jnp.array([th1_init, th2_init, th3_init, th4_init, th5_init, th6_init, th7_init, th8_init, th9_init]),
    jnp.array([th1_init, 10, 10, 48, th5_init, th6_init, 48, 1, 1]),
    jnp.array([th1_init, 100, 100, 100, 100, 100, 100, 100, 100]),
    jnp.array([th1_init, 100, 100, 10,  100, 100, 100, 100, 100]),
    jnp.array([th1_init, 5, 5, 24, 5, 5, 84, 5, 5])
]

for init in inits:
    fig, ax = plt.subplots()
    sol = minimize(c, init, args=(t, y), method="Powell", tol=1e-10)
    # plot
```

```
    y_plotting = f(t_plotting, *sol.x)
    plt.scatter(t, y, color="black", marker="x", s=15)
    plt.plot(t, y, color="blue")
    plt.plot(t_plotting, y_plotting, color="red")
    plt.title(init)
```

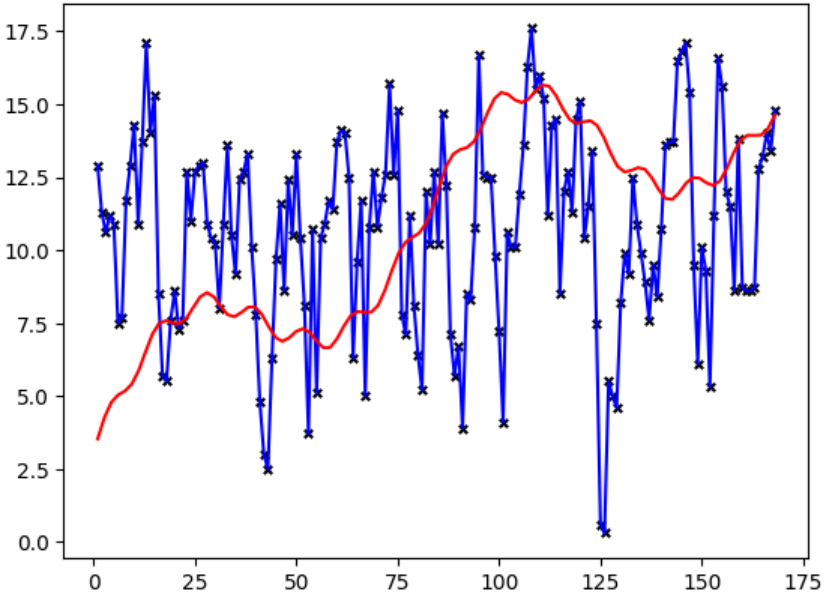
```
[10.641668  7.      7.     18.      5.      5.     18.
  5.      5.    ]
```



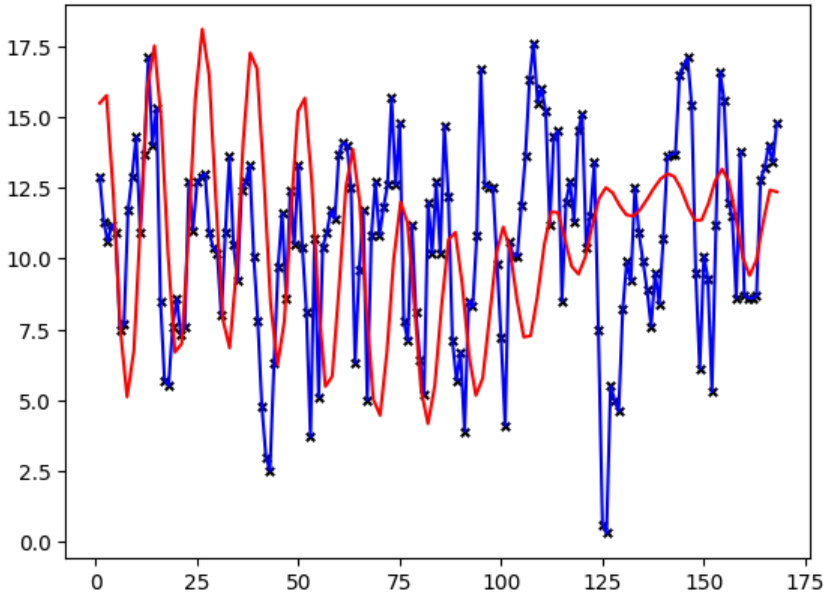
```
[10.641668 10.      10.      48.       5.       5.      48.
  1.       1.     ]
```



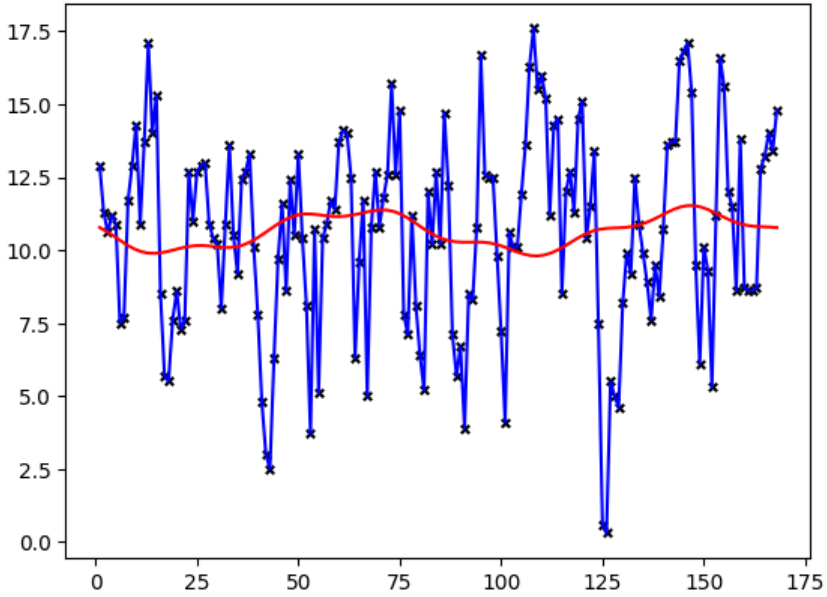

```
[ 10.641668 100.      100.      100.      100.      100.
  100.      100.      100.     ]
```



```
[ 10.641668 100.      100.       10.      100.      100.
  100.      100.      100.      ]
```




```
[10.641668  5.       5.      24.       5.       5.      84.
  5.       5.      ]
```



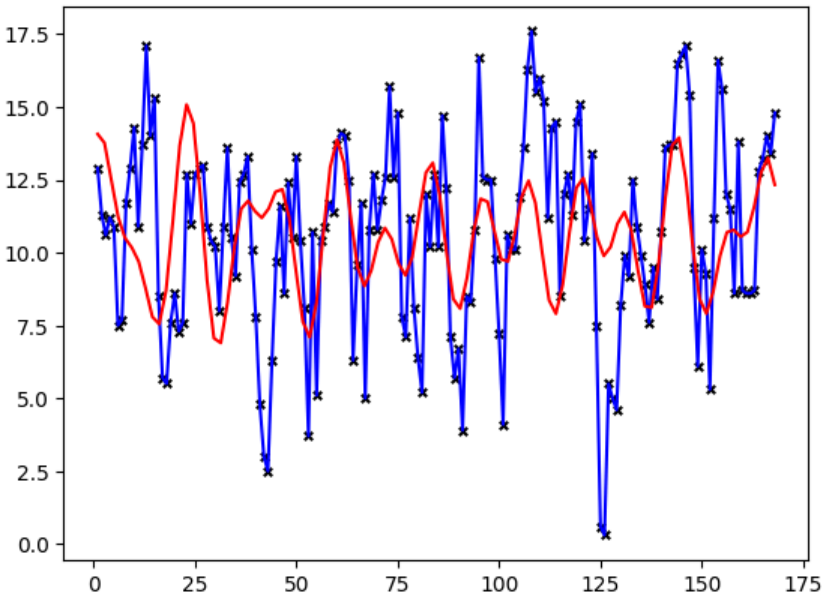
In [16]:
```
# I found that setting theta 4 and theta 7 to very different values, with the rest of the initializations
# relatively low, gives the worst fit. I also found that in general, fitting works best if the initializations
# for theta 4 and theta 7 are very similar
```

f. Plot the data and your model at the best fit parameters. Calculate the Fisher Information Matrix at the best fit and report the variance in each of the parameter estimates. Don't forget to scale your results by σ. Assume that σ^2 is the sum of the squared error divided by the degrees of freedom

In [17]:
```
sigma_squared = len(t.flatten())-9 # data - parameters
```

In [18]:
```
# plotting data and model for best fit
inits = [
    jnp.array([th1_init, th2_init, th3_init, th4_init, th5_init, th6_init, th7_init, th8_init, th9_init])
]

for init in inits:
    fig, ax = plt.subplots()
    sol = minimize(c, init, args=(t, y), method="Powell", tol=1e-10)
    # plot
    y_plotting = f(t_plotting, *sol.x)
    plt.scatter(t, y, color="black", marker="x", s=15)
    plt.plot(t, y, color="blue")
    plt.plot(t_plotting, y_plotting, color="red")
    plt.title(init)
```

```
[10.641668  7.      7.     18.      5.      5.     18.
  5.      5.     ]
```



In [19]:
```python
# FIM

# redefine f to take an array
def f(t: jnp.ndarray, thet) -> jnp.ndarray:
    th1, th2, th3, th4, th5, th6, th7, th8, th9 = thet
    out = jnp.array(
        [
            jnp.full(t.shape, th1)+
            th2*jnp.cos(2*jnp.pi*t*(1/12))+   # annual cycle
            th3*jnp.sin(2*jnp.pi*t*(1/12))+   # annual cycle
            th5*jnp.cos(2*jnp.pi*t*(1/th4))+  # el nino
            th6*jnp.sin(2*jnp.pi*t*(1/th4))+  # el nino
            th8*jnp.cos(2*jnp.pi*t*(1/th7))+  # southern oscillation
            th9*jnp.sin(2*jnp.pi*t*(1/th7))   # southern oscillation
        ]
    ).T
    return out



J = jax.jacrev(f, argnums=1)
ja = J(t, jnp.array([th1_init, th2_init, th3_init, th4_init, th5_init, th6_init, th7_init, th8_init, th9_init]))
ja = ja[:, 0, :]
FIM = np.matmul(ja.T, ja) * (1/sigma_squared)
pd.DataFrame(FIM)
```

Out[19]:

	0	1	2	3	4	5	6	7	8
0	1.056604e+00	2.549129e-08	4.123592e-08	-0.024398	0.010728	0.029475	-0.024398	0.010728	0.029475
1	2.549129e-08	5.283018e-01	-2.099283e-08	0.181778	-0.017361	-0.019177	0.181778	-0.017361	-0.019177
2	4.123592e-08	-2.099283e-08	5.283020e-01	-0.481038	0.032015	-0.018484	-0.481038	0.032015	-0.018484
3	-2.439776e-02	1.817779e-01	-4.810380e-01	93.908333	-4.241397	4.390851	93.908333	-4.241397	4.390850
4	1.072795e-02	-1.736075e-02	3.201550e-02	-4.241397	0.522202	0.005118	-4.241397	0.522202	0.005118
5	2.947470e-02	-1.917655e-02	-1.848431e-02	4.390851	0.005118	0.534402	4.390851	0.005118	0.534402
6	-2.439776e-02	1.817779e-01	-4.810380e-01	93.908333	-4.241397	4.390851	93.908333	-4.241397	4.390850
7	1.072795e-02	-1.736075e-02	3.201550e-02	-4.241397	0.522202	0.005118	-4.241397	0.522202	0.005118
8	2.947470e-02	-1.917656e-02	-1.848431e-02	4.390850	0.005118	0.534402	4.390850	0.005118	0.534402

In [20]:
```
# Covariance Matrix - holds variance at each of the parameter estimates on diagonal, covariances off diagonal
Cov = np.linalg.pinv(FIM) # the FIM was singular here, so I'll put the pseudoinverse
pd.DataFrame(Cov)
```

Out[20]:

	0	1	2	3	4	5	6	7	8
0	0.949388	-0.005392	0.000269	0.003808	0.021644	-8.572461e-02	0.003808	0.021644	-2.980968e-02
1	-0.005392	1.903662	-0.003496	-0.008350	-0.037022	2.610855e-01	-0.008350	-0.037022	-5.468180e-02
2	0.000269	-0.003496	1.902988	0.002745	-0.036205	7.556293e-02	0.002745	-0.036205	-5.429448e-02
3	0.003808	-0.008350	0.002745	0.011073	0.090573	-1.240596e-01	0.011073	0.090573	-6.004233e-02
4	0.021644	-0.037022	-0.036205	0.090573	1.222085	-1.041312e+00	0.090573	1.222085	-4.742323e-01
5	-0.085725	0.261085	0.075563	-0.127582	-1.041211	-5.592402e+06	-0.124135	-1.041413	5.592406e+06
6	0.003808	-0.008350	0.002745	0.011073	0.090573	-1.276568e-01	0.011073	0.090573	-5.644511e-02
7	0.021644	-0.037022	-0.036205	0.090573	1.222085	-1.041312e+00	0.090573	1.222085	-4.742329e-01
8	-0.029810	-0.054682	-0.054294	-0.056520	-0.474333	5.592406e+06	-0.059967	-0.474132	-5.592406e+06

Acknowledgment

Work in this repository and with associated assignments and projects may be adapted or copied from similar files used in my prior academic and industry work (e.g., using a LaTeX file or Dockerfile as a starting point). Those files and any other work in this repository may have been developed with the help of LLM's like ChatGPT. For example, to provide context, answer questions, refine writing, understand function call syntax, and assist with repetitive tasks. In these cases, deliverables and associated work reflect my best efforts to optimize my learning and demonstrate my capacity, while using available resources and LLM's to facilitate the process.

ChatGPT Conversation