Problem Set 2

Part I. Exercises for Chapter 4: Graphical Analysis

Problem 1:

$$F(x) = x^2 - 1.1$$

(a)

$$\begin{aligned} F(x) &= x & &\text{This must be true for x to be fixed} \\ x^2 - 1.1 &= x \\ x^2 - x &= 1.1 \\ x^2 - x + 0.25 &= 1.35 & &\text{Left side of equation made perfect square} \\ (x-0.5)^2 &= 1.35 \\ x - 0.5 &= \pm\sqrt{1.35} \\ x &= 0.5 \pm \sqrt{1.35} \end{aligned}$$

(b)

$$\begin{aligned} F^2(x) &= x & &\text{This must me true for x} \\ (x^2-1.1)^2 - 1.1 &= x & &\text{By substituti} \\ x^4 - 2.2x^2 + 1.21 - 1.1 &= x \\ x^4 - 2.2x^2 - x + 0.11 &= 0 \\ (x^2 - x - 1.1)(x^2 + x - 0.1) &= 0 & &\text{Factoring possible because fixed points a} \\ x = 0.5(-1 \pm \sqrt{1 - 4(-0.1)}) & & &\text{Quadratic equation o} \\ x = -0.5 \pm \sqrt{0.35} \end{aligned}$$

$F(x)$ has fixed points at $x_0 = 0.5 \pm \sqrt{1.35}$ and 2-cycles at $x_0 = -0.5 \pm \sqrt{0.35}$.

Problem 2:

(a)

$$F(x) = \frac{1}{x}$$

Finding fixed points:

$$F(x) = x$$
$$\frac{1}{x} = x$$
$$x^2 = 1$$
$$x = \pm 1$$

Finding 2-cycles:

$$F^2(x) = x$$
$$\frac{1}{\frac{1}{x}} = x$$
$$x = x$$
$$x \in \mathbb{R}$$

$F(x) = \frac{1}{x}$ has fixed points at $x_0 = \pm 1$ and all other seed values lie on periodic 2-cycl

(b)

$$F(x) = e^x$$

$F(x) = e^x > x$ for all $x \in \mathbb{R}$. To demonstrate this, let's express the difference between e^x and x at a given x-value as $d(x) = e^x - x$. If $e^x = x$ at some x-value c, then $d(c)$ will equal 0. Let's find $d(x)$'s critical points. $d'(x) = e^x - 1$, which is only ever equal to 0 at $x = 0$. $d''(0) = e^0 = 1$, so $x = 0$ is a relative minimum of $d(x)$ and, as the only critical point of the function, must be the absolute minimum of $d(x)$. $d(0) = 1 > 0$, meaning that $d(x)$ is always greater than 0, so $e^x > x$ for all $x \in \mathbb{R}$. Thus, all orbits on $F(x)$ diverge to ∞.

(c)

$$F(x) = x^2 + 1$$

$F(x) = x^2 + 1 > x$ for all $x \in \mathbb{R}$. Like in (b), we can consider the difference $d(x) = x^2 - x + 1$. Because the discriminant for this function is negative and the leading coefficient is positive, we know that $d(x) > 0$ and consequently that $x^2 + 1 > x$ for all $x \in \mathbb{R}$. Therefore, all orbits on $F(x)$ diverge to ∞.

Problem 4:

$$F(x) = x \sin x$$

Setting Up Tools:

In [ ]:
```
from math import *
import numpy as np
import matplotlib.pyplot as plt
```

Functions to generate cobweb diagrams:

In [ ]:
```
# Driver function to make cobweb diagram
def cobweb_diagram(f, x_0, iter=25, window=(-10,10,-10,10)):
    # Plot the function
    xs = np.linspace(window[0], window[1], 100)
    plt.plot(xs, f(xs), 'b')

    # Plot the line y = x
    plt.plot([window[0], window[1]], [window[0], window[1]], 'k')

    # Call plot_cobweb() on seed value(s)
    if isinstance(x_0, (list, tuple, np.array, np.ndarray)):
        for seed in x_0:
            plot_cobweb(f, seed, iter, window)
    else:
        plot_cobweb(f, x_0, iter, window)

    # Display plot
    plt.axis('square')
    plt.axis(window)
    plt.grid()
    plt.show()

# Plots the "cobweb"
def plot_cobweb(f, x_0, iter, window):
    A = [] # stores x-coordinates of points on web
    B = [] # stores y-coordinates

    for i in range(iter):
        # If seed value is outside the window, stop looping
        if x_0 < window[0] or x_0 > window[1]:
            break

        # Add point on line y=x
        A.append(x_0)
        B.append(x_0)

        # Add point on line y=f(x)
        A.append(x_0)
        B.append(f(x_0))

        # Update seed value
        x_0 = f(x_0)

    # Plot cobweb
    plt.plot(A, B, 'r')
```

Function to calculate horizontal intercepts with $F(x)$:

In [ ]:
```
# Modifies Newton's Method to find horizontal intercepts
# WARNING: No protection against runtime errors, use with caution
def calculate_intercept(f, f_prime, y, guess, tol=1e-9):
    seed = guess
    while abs(f(seed) - y) > tol:
        seed += (y - f(seed)) / f_prime(seed)
    return seed
```

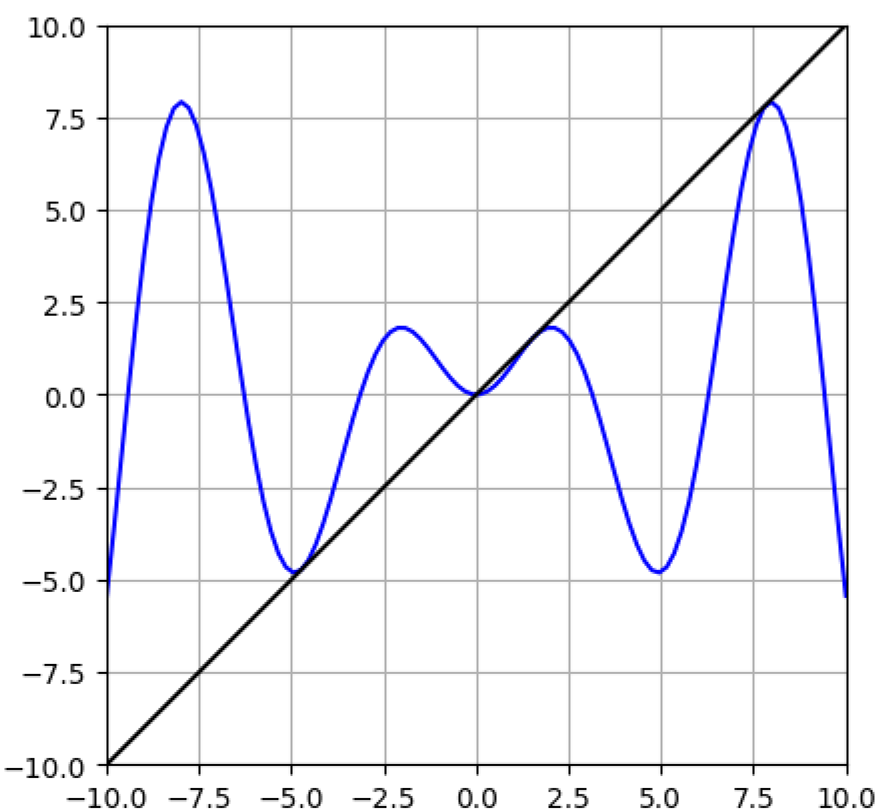
Conducting Orbit analysis

We can find the fixed points of $F(x)$ by setting $F(x)$ equal to x.

$$\begin{aligned} F(x) &= x \\ x\sin x &= x \\ x\sin x - x &= 0 \\ x(\sin x - 1) &= 1 \end{aligned}$$

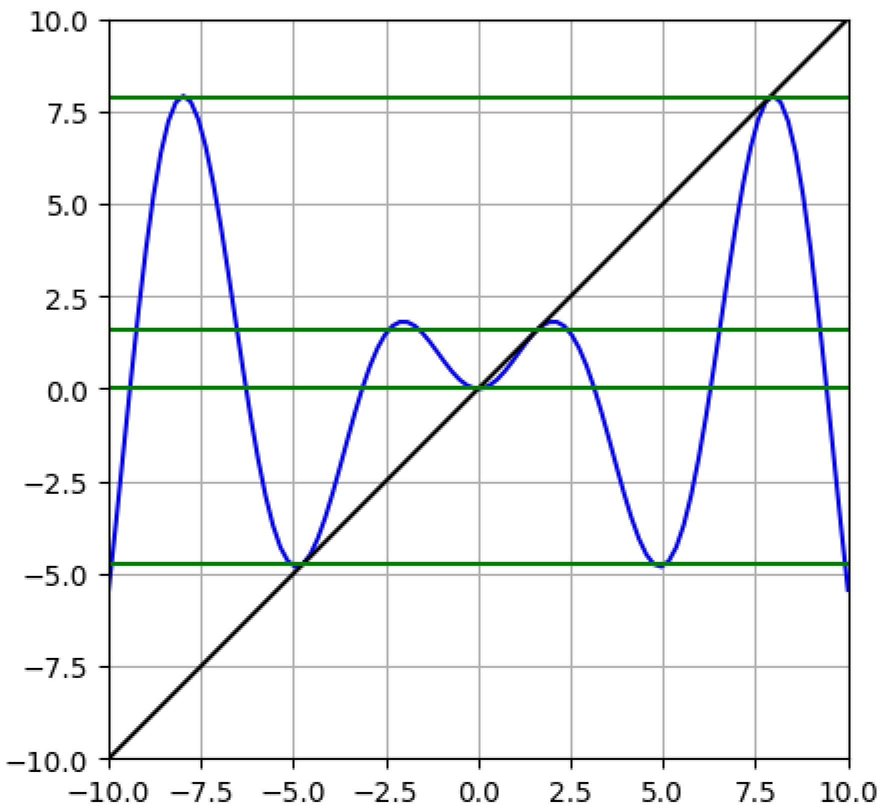
This will be satisfied when $x = 0$ or when $\sin(x) - 1 = 0$, which will occur for all $x = \frac{\pi}{2} + 2\pi k$ where $k \in \mathbb{Z}$. Thus, there are infinite fixed points, evenly spaced with the exception of $x = 0$. Let's verify this by making a cobweb diagram with these points.

In [ ]:
```
F = lambda x: x * np.sin(x)
cobweb_diagram(F, x_0=(-1.5*pi,0,0.5*pi,2.5*pi))
```



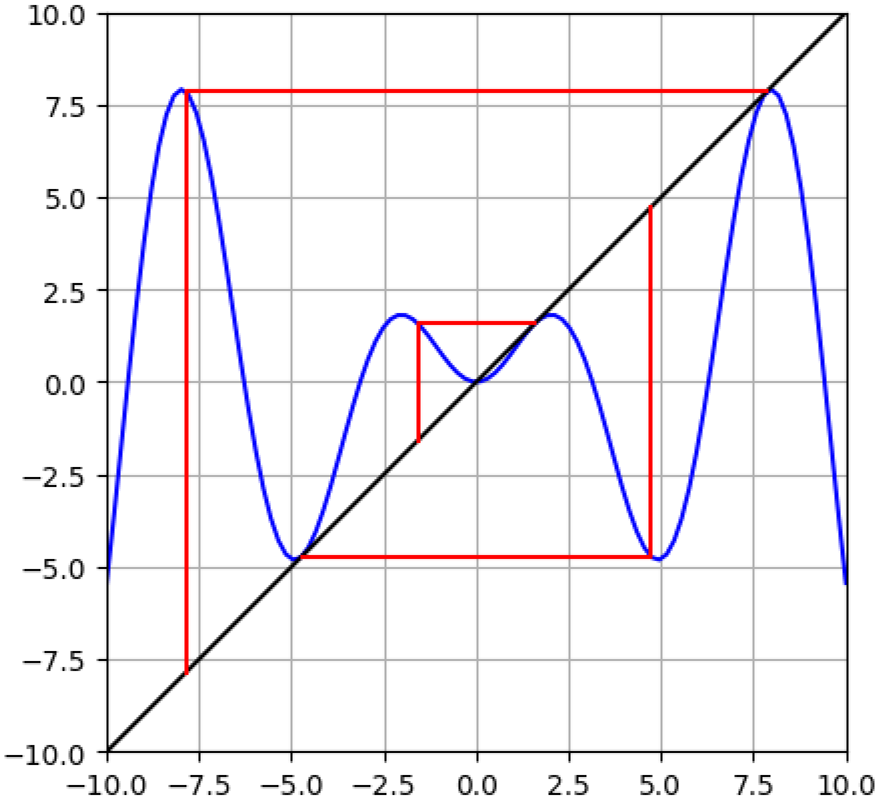
If we draw horizontal lines out from these fixed points, we can see the points that are fixed after a single iteration.

In [ ]:
```
xs = np.linspace(-10, 10, 100)
plt.plot(xs, F(xs), 'b')
plt.plot([-10,10], [-10,10], 'k')

# Draw horizontal lines
plt.plot([-10,10], [-1.5*pi,-1.5*pi], 'g')
plt.plot([-10,10], [0,0], 'g')
plt.plot([-10,10], [0.5*pi,0.5*pi], 'g')
plt.plot([-10,10], [2.5*pi,2.5*pi], 'g')

# Make it look pretty
plt.axis('square')
plt.axis((-10,10,-10,10))
plt.grid()
plt.show()
```



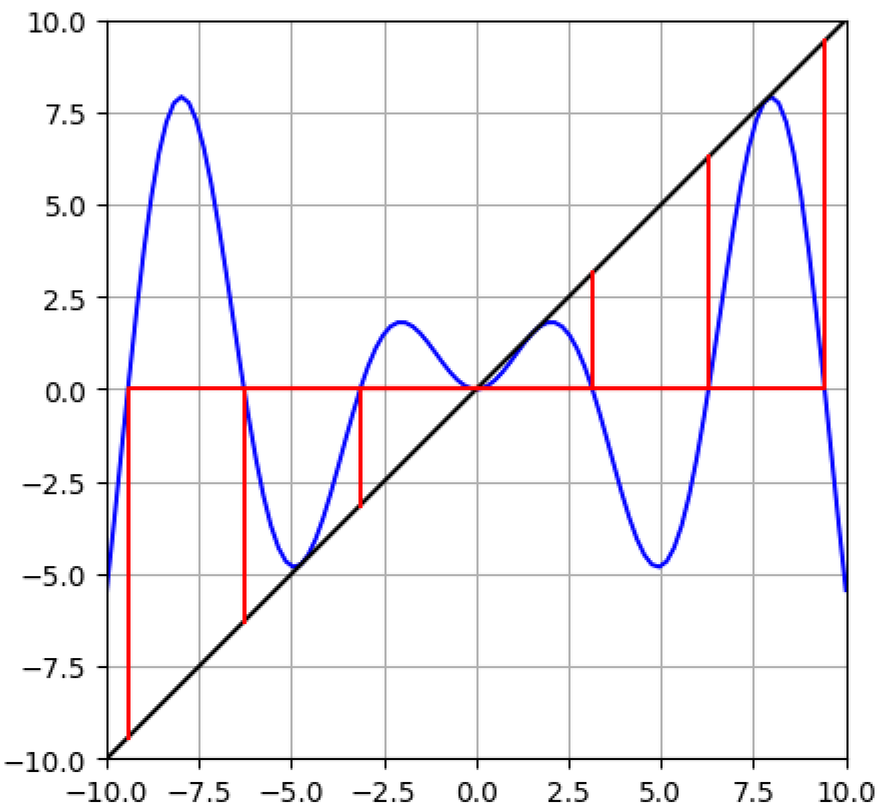
We can use the fact that $x \sin x$ is symmetric to find some of these. Let's check the mirrors of the fixed points.

In [ ]:
```
cobweb_diagram(F, x_0=(-2.5*pi,-0.5*pi,1.5*pi))
```



And intersections with $y = 0$ are fairly easy to find. These occur for all $x = \pi k$ where $k \in \mathbb{Z}$.

```
In [ ]: cobweb_diagram(F, x_0=(-3*pi,-2*pi,-pi,pi,2*pi,3*pi))
```



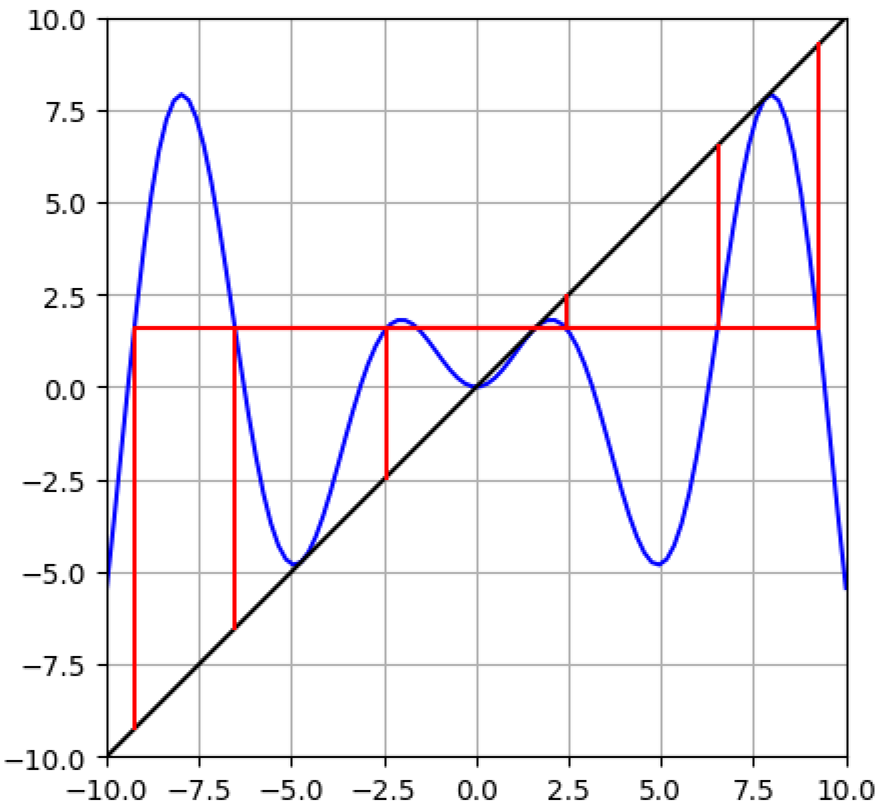
But the rest of the intersecions can't be calculated mathematically (at least to my knowledge), so we'll estimate them using numerical methods. For this, we'll use the derivative $F'(x) = x\cos x + \sin x$.

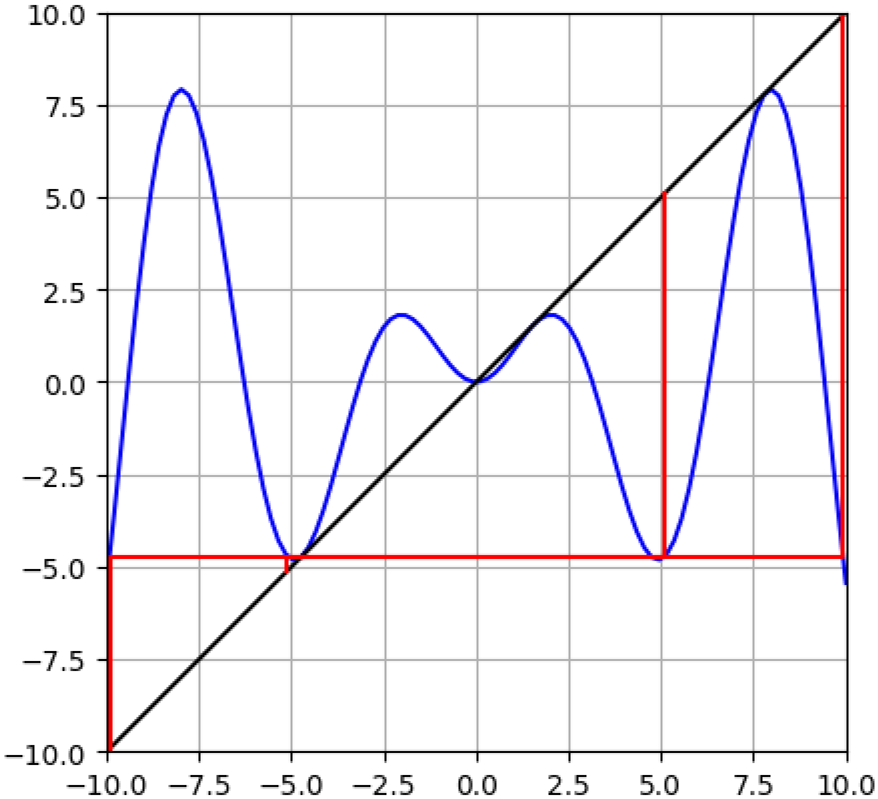
In [ ]:
```
F_prime = lambda x: x * cos(x) + sin(x)

# Intersections with y = 0.5*pi
print('Intersections with y = 0.5*pi:')
print(calculate_intercept(F, F_prime, 0.5*pi, -9))
print(calculate_intercept(F, F_prime, 0.5*pi, -6.5))
print(calculate_intercept(F, F_prime, 0.5*pi, -2.5))
print(calculate_intercept(F, F_prime, 0.5*pi, 2.5))
print(calculate_intercept(F, F_prime, 0.5*pi, 6.5))
print(calculate_intercept(F, F_prime, 0.5*pi, 9))
print()

# Intersections with y = -1.5*pi
print('Intersections with y = -1.5*pi:')
print(calculate_intercept(F, F_prime, -1.5*pi, -10))
print(calculate_intercept(F, F_prime, -1.5*pi, -5.5))
print(calculate_intercept(F, F_prime, -1.5*pi, 5.5))
print(calculate_intercept(F, F_prime, -1.5*pi, 10))
print()

# Intersections with y = 2.5*pi
print('Intersections with y = 2.5*pi:')
print(calculate_intercept(F, F_prime, 2.5*pi, -8.5))
print(calculate_intercept(F, F_prime, 2.5*pi, 8.5))

# Store results in a dictionary
intersections = {
    '0.5pi' : [calculate_intercept(F, F_prime, 0.5*pi, -9),
               calculate_intercept(F, F_prime, 0.5*pi, -6.5),
               calculate_intercept(F, F_prime, 0.5*pi, -2.5),
               calculate_intercept(F, F_prime, 0.5*pi, 2.5),
               calculate_intercept(F, F_prime, 0.5*pi, 6.5),
               calculate_intercept(F, F_prime, 0.5*pi, 9)],

    '-1.5pi' : [calculate_intercept(F, F_prime, -1.5*pi, -10),
                calculate_intercept(F, F_prime, -1.5*pi, -5.5),
                calculate_intercept(F, F_prime, -1.5*pi, 5.5),
                calculate_intercept(F, F_prime, -1.5*pi, 10)],

    '2.5pi' : [calculate_intercept(F, F_prime, 2.5*pi, -8.5),
               calculate_intercept(F, F_prime, 2.5*pi, 8.5)]
}
```

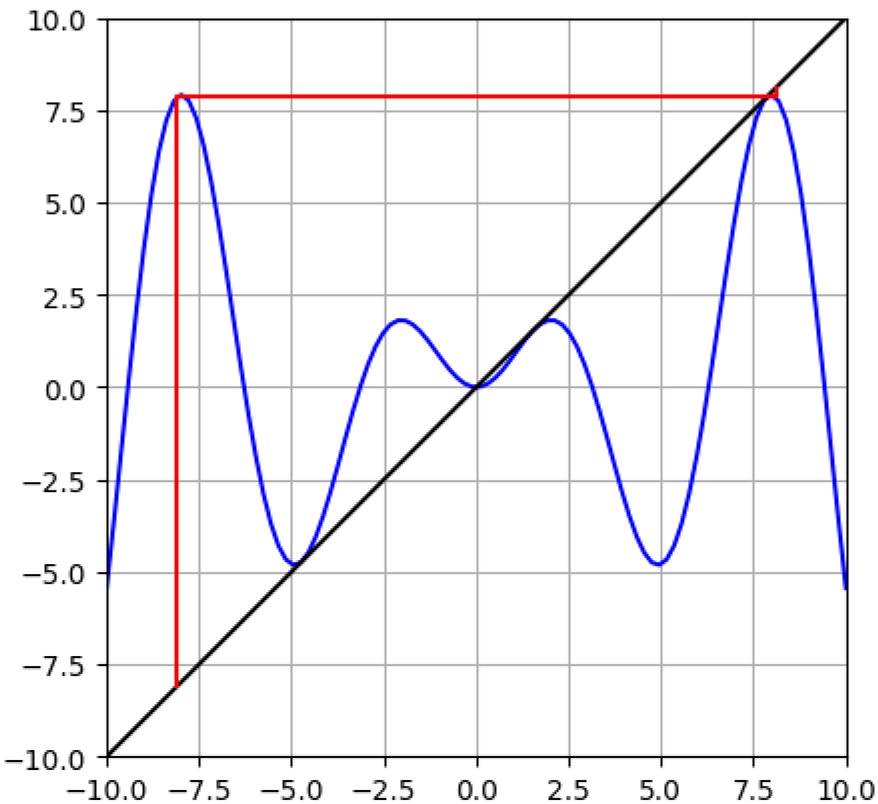
```
Intersections with y = 0.5*pi:
-9.254213650017624
-6.526260523514655
-2.443322686566392
2.443322686566392
6.526260523514655
9.254213650017624

Intersections with y = -1.5*pi:
-9.919797568933081
-5.109025786412278
5.109025786412278
9.919797568933081

Intersections with y = 2.5*pi:
-8.102101487957542
8.102101487957542
```

Let's verify these points graphically.

In [ ]:
```
cobweb_diagram(F, x_0=intersections['0.5pi'])
cobweb_diagram(F, x_0=intersections['-1.5pi'])
cobweb_diagram(F, x_0=intersections['2.5pi'])
```



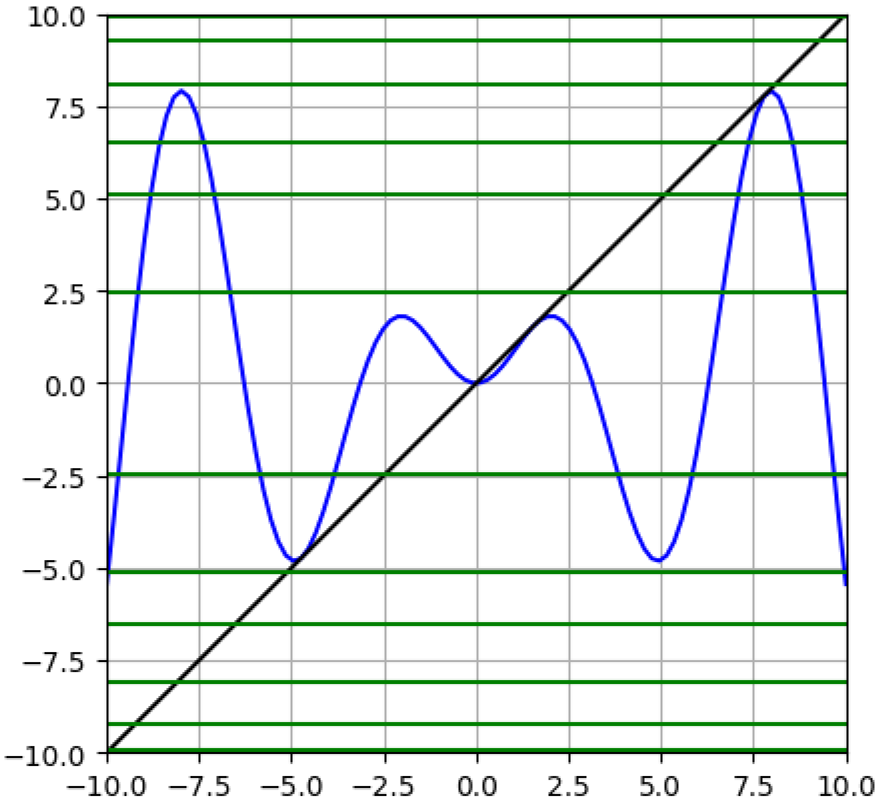




Now using the points we've found so far, we can find the seed values that become fixed after two iterations.

In [ ]:
```
xs = np.linspace(-10, 10, 100)
plt.plot(xs, F(xs), 'b')
```

```
plt.plot([-10,10], [-10,10], 'k')

# Draw horizontal lines
for intersect in intersections:
    for seed in intersections[intersect]:
        plt.plot([-10,10], [seed,seed], 'g')

# Make it look pretty
plt.axis('square')
plt.axis((-10,10,-10,10))
plt.grid()
plt.show()
```



In [ ]:

```
# Intersections with y = -5.11
print('Intersections with y = -5.11:')
print(calculate_intercept(F, F_prime, intersections['-1.5pi'][1], -10))
print(calculate_intercept(F, F_prime, intersections['-1.5pi'][1], 10))
print()

# Intersections with y = -pi
print('Intersections with y = -pi')
print(calculate_intercept(F, F_prime, -pi, -10))
print(calculate_intercept(F, F_prime, -pi, -6))
print(calculate_intercept(F, F_prime, -pi, 6))
print(calculate_intercept(F, F_prime, -pi, 10))
print()

# Intersections with y = -2.44
print('Intersections with y = -2.44:')
print(calculate_intercept(F, F_prime, intersections['0.5pi'][2], -10))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][2], -6))
```

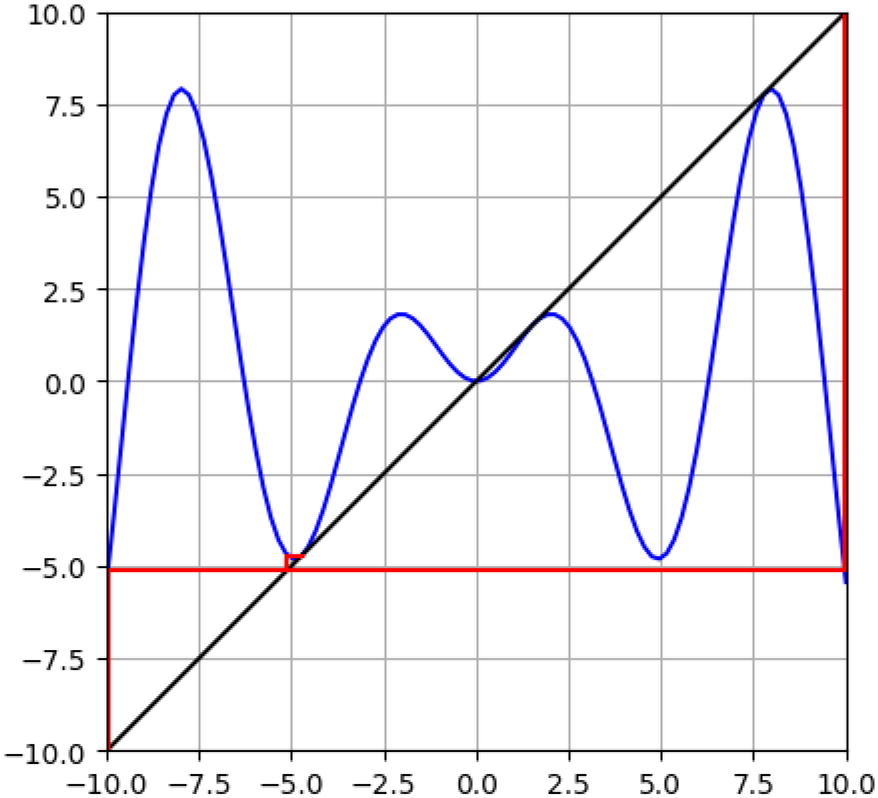
```
print(calculate_intercept(F, F_prime, intersections['0.5pi'][2], -4))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][2], 4))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][2], 6))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][2], 10))
print()

# Intersections with y = -0.5pi
print('Intersections with y = -0.5pi')
print(calculate_intercept(F, F_prime, -0.5*pi, -10))
print(calculate_intercept(F, F_prime, -0.5*pi, -6))
print(calculate_intercept(F, F_prime, -0.5*pi, -4))
print(calculate_intercept(F, F_prime, -0.5*pi, 4))
print(calculate_intercept(F, F_prime, -0.5*pi, 6))
print(calculate_intercept(F, F_prime, -0.5*pi, 10))
print()

# Intersections with y = 2.44
print('Intersections with y = 2.44')
print(calculate_intercept(F, F_prime, intersections['0.5pi'][3], -9))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][3], -6))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][3], 6))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][3], 9))
print()

# Intersections with y = pi
print('Intersections with y = pi')
print(calculate_intercept(F, F_prime, pi, -9))
print(calculate_intercept(F, F_prime, pi, -6))
print(calculate_intercept(F, F_prime, pi, 6))
print(calculate_intercept(F, F_prime, pi, 9))
print()

# Intersections with y = 5.11
print('Intersections with y = 5.11')
print(calculate_intercept(F, F_prime, intersections['-1.5pi'][2], -9))
print(calculate_intercept(F, F_prime, intersections['-1.5pi'][2], -7))
print(calculate_intercept(F, F_prime, intersections['-1.5pi'][2], 7))
print(calculate_intercept(F, F_prime, intersections['-1.5pi'][2], 9))
print()

# Intersections with y = 2pi
print('Intersections with y = 2pi')
print(calculate_intercept(F, F_prime, 2*pi, -9))
print(calculate_intercept(F, F_prime, 2*pi, -7))
print(calculate_intercept(F, F_prime, 2*pi, 7))
print(calculate_intercept(F, F_prime, 2*pi, 9))
print()

# Intersections with y = 6.53
print('Intersections with y = 6.53')
print(calculate_intercept(F, F_prime, intersections['0.5pi'][4], -8))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][4], -7))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][4], 7))
print(calculate_intercept(F, F_prime, intersections['0.5pi'][4], 8))

# Add results to previous dictionary
```

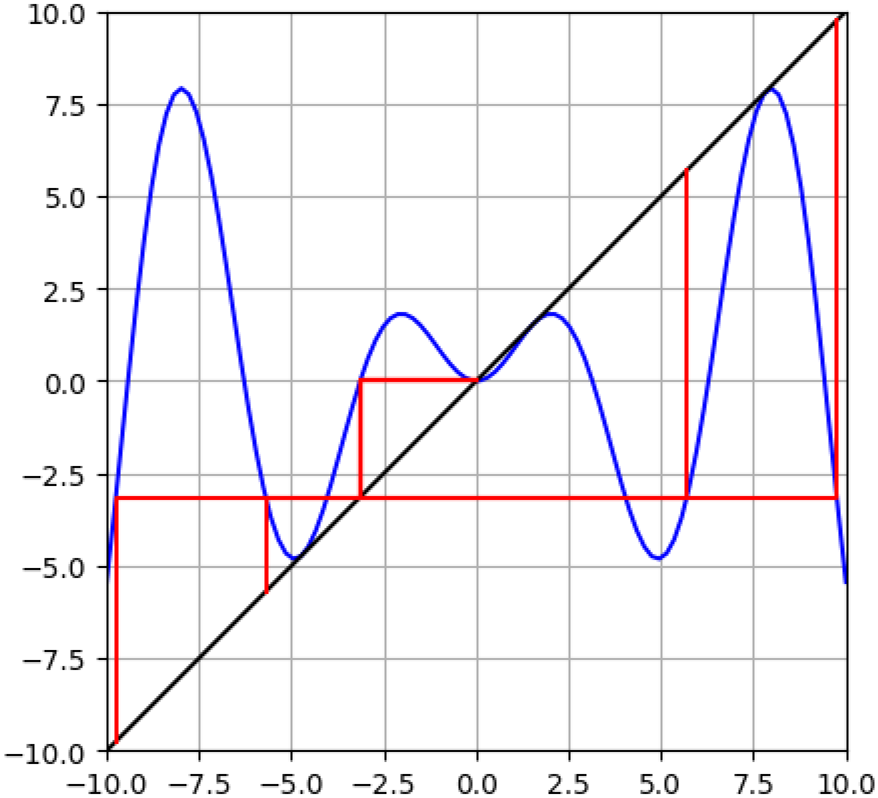
```
intersections['-5.11'] = [calculate_intercept(F, F_prime, intersections['-1.
                          calculate_intercept(F, F_prime, intersections['-1.

intersections['-pi'] = [calculate_intercept(F, F_prime, -pi, -10),
                        calculate_intercept(F, F_prime, -pi, -6),
                        calculate_intercept(F, F_prime, -pi, 6),
                        calculate_intercept(F, F_prime, -pi, 10)]

intersections['-2.44'] = [calculate_intercept(F, F_prime, intersections['0.5
                          calculate_intercept(F, F_prime, intersections['0.5
                          calculate_intercept(F, F_prime, intersections['0.5
                          calculate_intercept(F, F_prime, intersections['0.5
                          calculate_intercept(F, F_prime, intersections['0.5
                          calculate_intercept(F, F_prime, intersections['0.5

intersections['-0.5pi'] = [calculate_intercept(F, F_prime, -0.5*pi, -10),
                           calculate_intercept(F, F_prime, -0.5*pi, -6),
                           calculate_intercept(F, F_prime, -0.5*pi, -4),
                           calculate_intercept(F, F_prime, -0.5*pi, 4),
                           calculate_intercept(F, F_prime, -0.5*pi, 6),
                           calculate_intercept(F, F_prime, -0.5*pi, 10)]

intersections['2.44'] = [calculate_intercept(F, F_prime, intersections['0.5p
                         calculate_intercept(F, F_prime, intersections['0.5p
                         calculate_intercept(F, F_prime, intersections['0.5p
                         calculate_intercept(F, F_prime, intersections['0.5p

intersections['pi'] = [calculate_intercept(F, F_prime, pi, -9),
                       calculate_intercept(F, F_prime, pi, -6),
                       calculate_intercept(F, F_prime, pi, 6),
                       calculate_intercept(F, F_prime, pi, 9)]

intersections['5.11'] = [calculate_intercept(F, F_prime, intersections['-1.5
                         calculate_intercept(F, F_prime, intersections['-1.5
                         calculate_intercept(F, F_prime, intersections['-1.5
                         calculate_intercept(F, F_prime, intersections['-1.5

intersections['2pi'] = [calculate_intercept(F, F_prime, 2*pi, -9),
                        calculate_intercept(F, F_prime, 2*pi, -7),
                        calculate_intercept(F, F_prime, 2*pi, 7),
                        calculate_intercept(F, F_prime, 2*pi, 9)]

intersections['6.53'] = [calculate_intercept(F, F_prime, intersections['0.5p
                         calculate_intercept(F, F_prime, intersections['0.5p
                         calculate_intercept(F, F_prime, intersections['0.5p
                         calculate_intercept(F, F_prime, intersections['0.5p
```

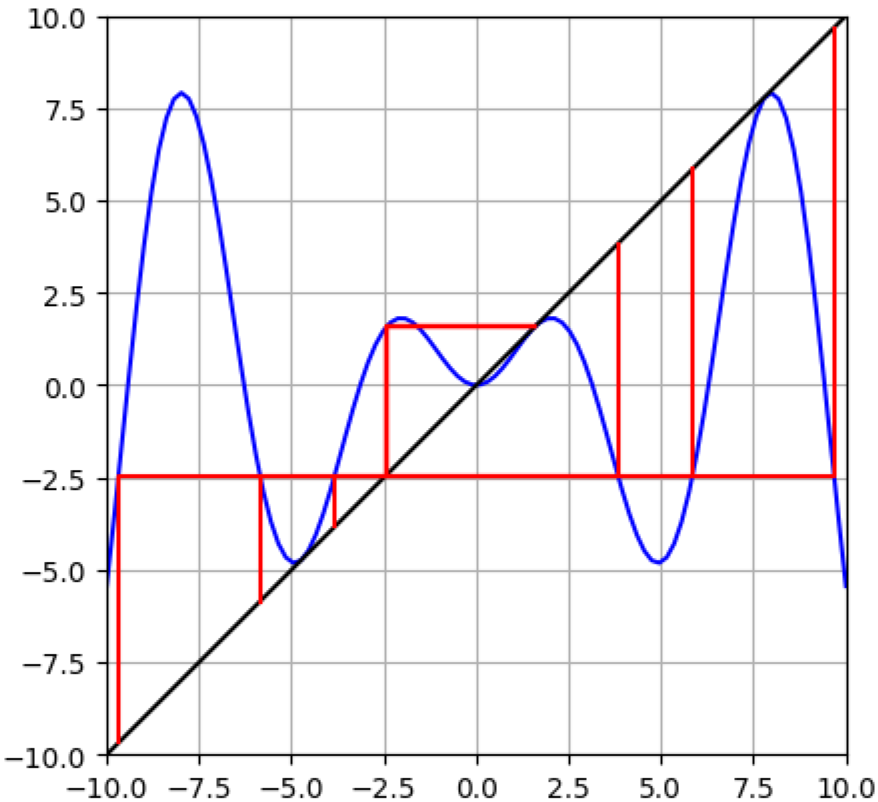
```
Intersections with y = -5.11:
-9.963208487016399
9.963208487016399

Intersections with y = -pi
-9.752749948040064
-5.699361557484731
5.699361557484731
9.752749948040064

Intersections with y = -2.44:
-9.679948783602313
-5.852511989430715
-3.832809970424864
3.832809970424864
5.852511989430715
9.679948783602313

Intersections with y = -0.5pi
-9.589326257258293
-6.019162645251092
-3.5939302541635367
3.5939302541635367
6.019162645251092
9.589326257258293

Intersections with y = 2.44
-9.154607833717213
-6.658888236701425
6.658888236701425
9.154607833717213

Intersections with y = pi
-9.07112290375477
-6.766048790452746
6.766048790452746
9.07112290375477

Intersections with y = 5.11
-8.80581913799646
-7.088122226634311
7.088122226634311
8.80581913799646

Intersections with y = 2pi
-8.606364308557731
-7.316132286079827
7.316132286079827
8.606364308557731

Intersections with y = 6.53
-15.266241940756972
-7.370628484763827
7.370628484763827
15.266241940756972
```

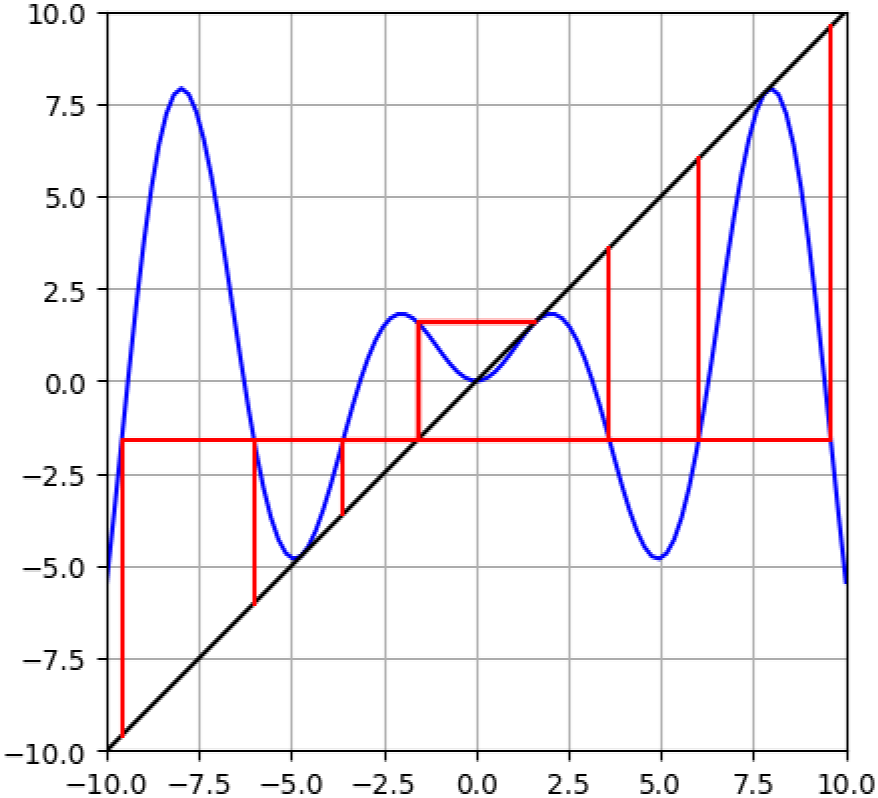
Now let's verify these points graphically.

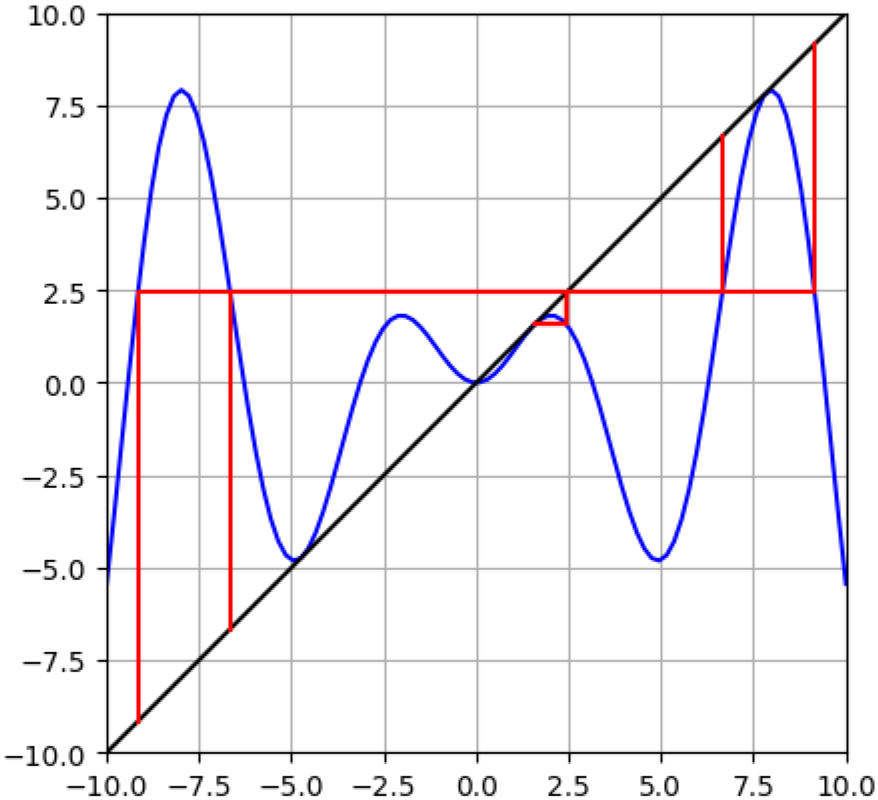
In [ ]:
```
cobweb_diagram(F, x_0=intersections['-5.11'])
cobweb_diagram(F, x_0=intersections['-pi'])
cobweb_diagram(F, x_0=intersections['-2.44'])
cobweb_diagram(F, x_0=intersections['-0.5pi'])
cobweb_diagram(F, x_0=intersections['2.44'])
cobweb_diagram(F, x_0=intersections['pi'])
cobweb_diagram(F, x_0=intersections['5.11'])
cobweb_diagram(F, x_0=intersections['2pi'])
cobweb_diagram(F, x_0=intersections['6.53'])
```

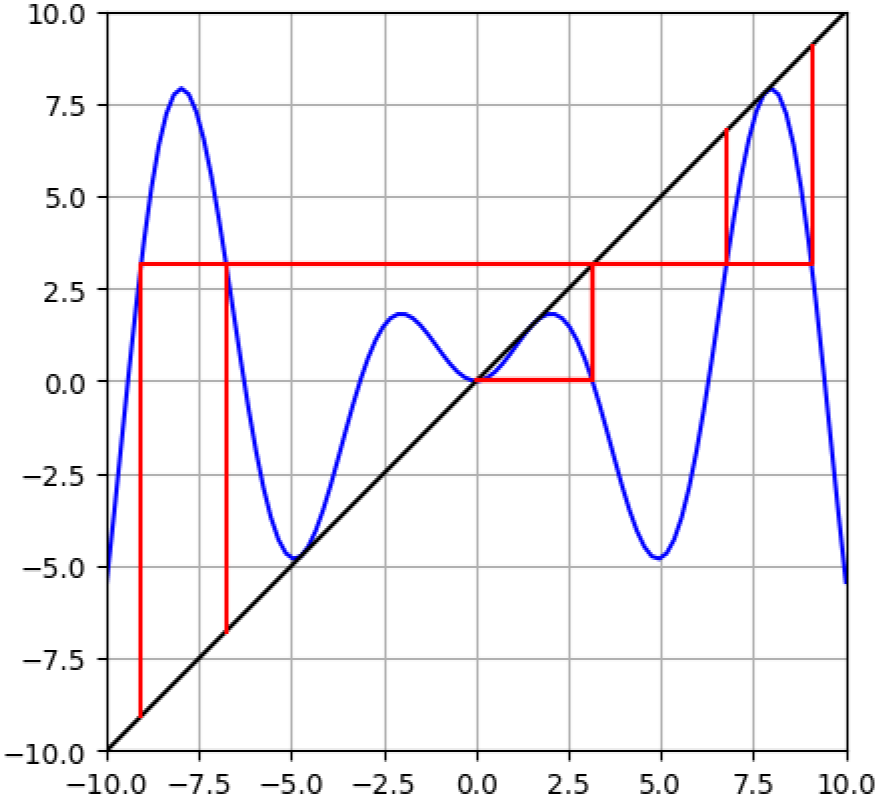


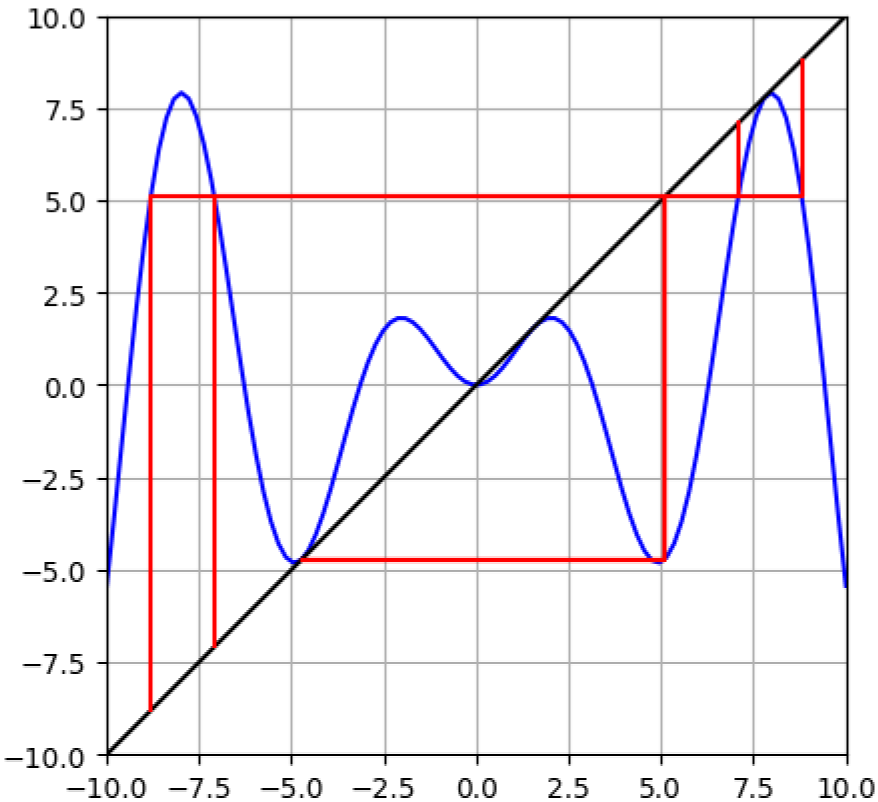

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

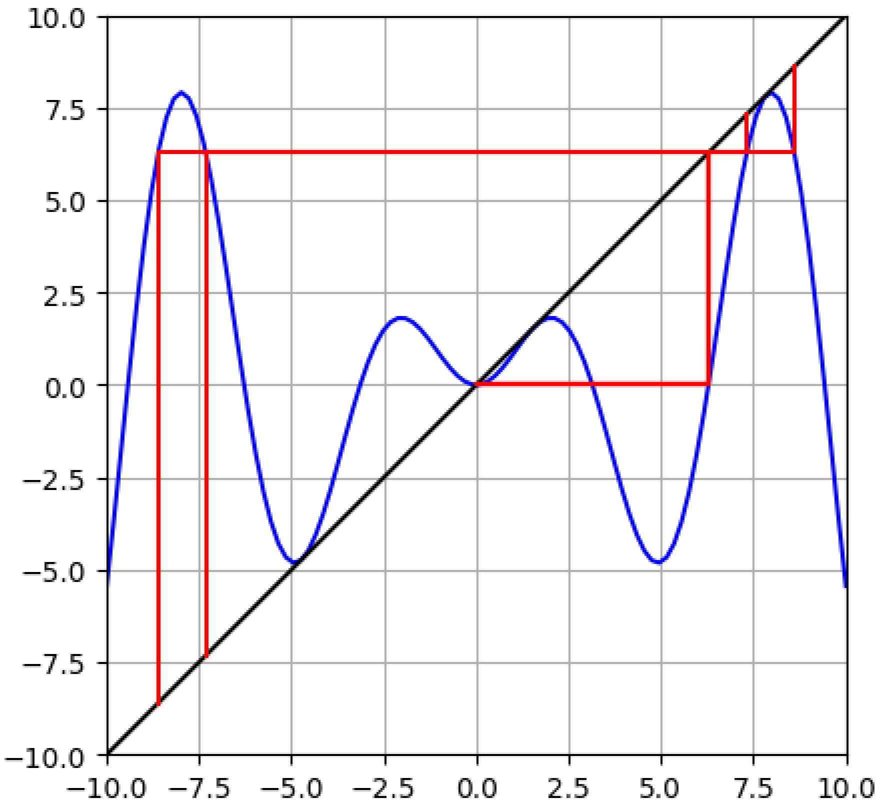

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

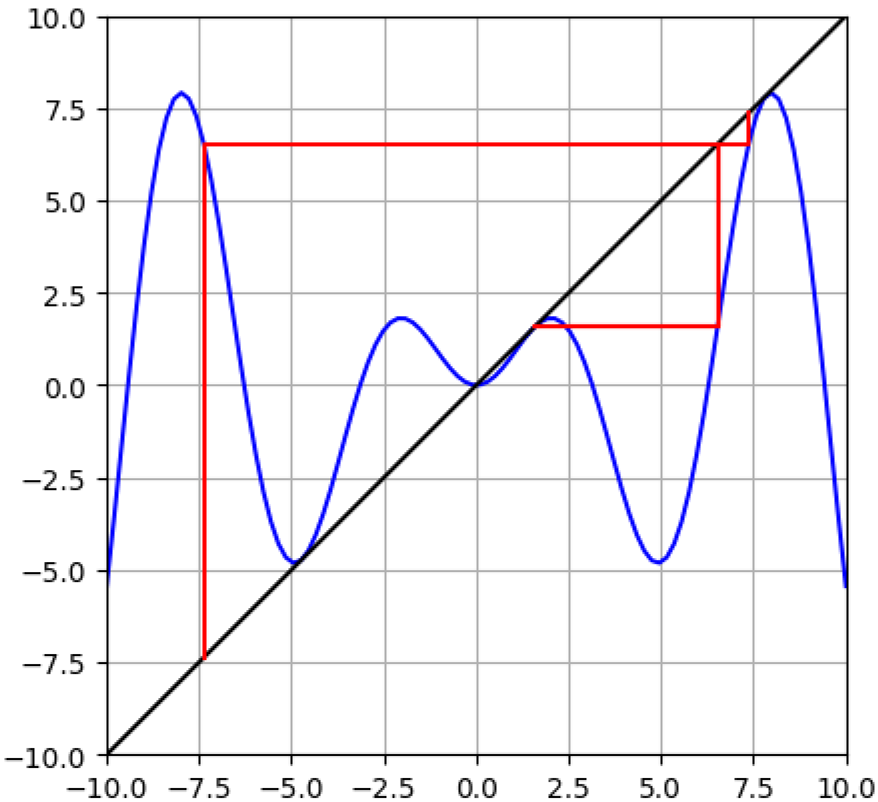

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0



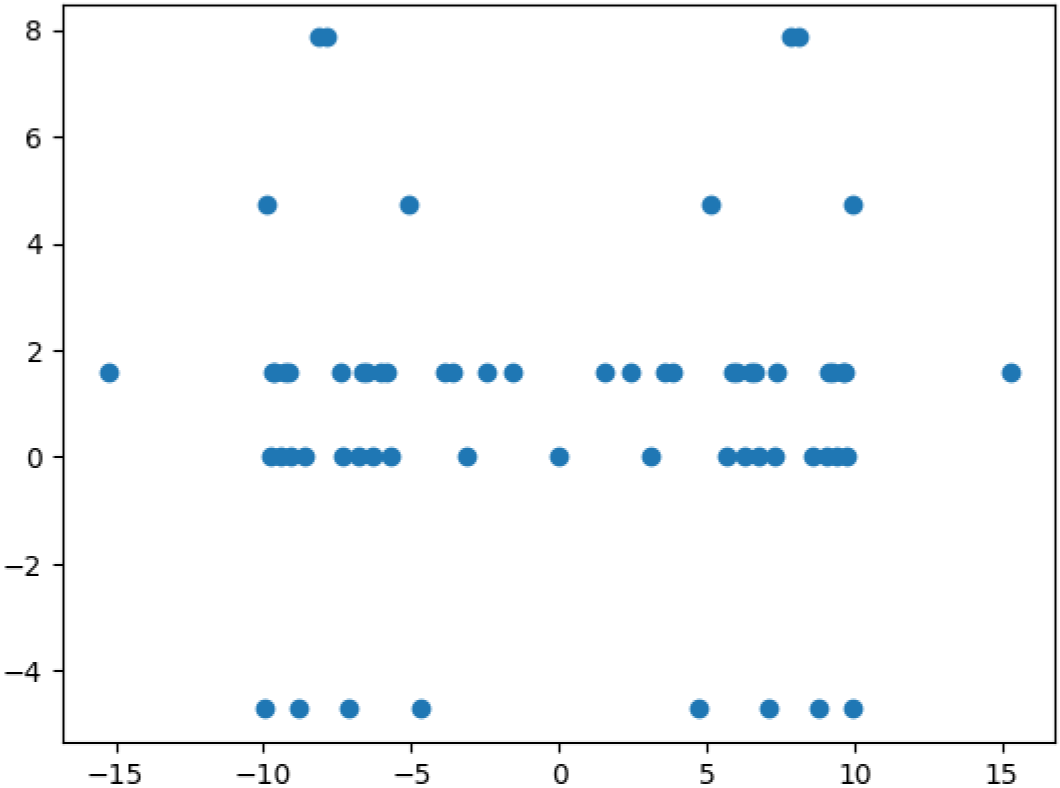


Now, let's put all of our current and eventual fixed points into a scatter plot and see if we can detect any patterns.

In [ ]:
```
xs = [-1.5*pi,0,0.5*pi,2.5*pi,-2.5*pi,-0.5*pi,1.5*pi,-3*pi,-2*pi,-pi,pi,2*pi
xs += intersections['0.5pi'] + intersections['-1.5pi'] + intersections['2.5p
```

```
xs += intersections['-5.11'] + intersections['-pi'] + intersections['-2.44']
xs += intersections['-0.5pi'] + intersections['2.44'] + intersections['pi']
xs += intersections['5.11'] + intersections['2pi'] + intersections['6.53']

ys = [-1.5*pi,0,0.5*pi,2.5*pi,2.5*pi,0.5*pi,-1.5*pi,0,0,0,0,0,0]
ys += [0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,1.5*pi,1.5*pi,1.5*pi,1.5*pi
ys += [-1.5*pi,-1.5*pi,0,0,0,0,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi]
ys += [0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi,0.5*pi
ys += [-1.5*pi,-1.5*pi,-1.5*pi,-1.5*pi,0,0,0,0,0.5*pi,0.5*pi,0.5*pi,0.5*pi]

plt.scatter(xs, ys)
```

Out[ ]: <matplotlib.collections.PathCollection at 0x7f43478512d0>



Unfortunately, this was as far as I was able to get with my analysis. The eventual fixed points don't seem to reveal much of a pattern, and my experimentation outside of the notebook in its submitted form reveal the difficulty of maintaining accurate estimations after a few iterations, so I may be out of luck for this assignment. In this next cell, I will combine all of the information I found into the one dictionary and print it. I apologize for the lack of completeness in my analysis.

In [ ]:
```
intersections['2.5pi'].append(-2.5*pi)
intersections['0.5pi'].append(-0.5*pi)
intersections['-1.5pi'].append(1.5*pi)
intersections['0'] = [-3*pi,-2*pi,-pi,pi,2*pi,3*pi]

for key in intersections:
    print(f'Intersections with y = {key}:')
    for value in intersections[key]:
```

```
        print(value)
    print()
```

```
Intersections with y = 0.5pi:
-9.254213650017624
-6.526260523514655
-2.443322686566392
2.443322686566392
6.526260523514655
9.254213650017624
-1.5707963267948966
-1.5707963267948966

Intersections with y = -1.5pi:
-9.919797568933081
-5.109025786412278
5.109025786412278
9.919797568933081
4.71238898038469
4.71238898038469

Intersections with y = 2.5pi:
-8.102101487957542
8.102101487957542
-7.853981633974483
-7.853981633974483

Intersections with y = -5.11:
-9.963208487016399
9.963208487016399

Intersections with y = -pi:
-9.752749948040064
-5.699361557484731
5.699361557484731
9.752749948040064

Intersections with y = -2.44:
-9.679948783602313
-5.852511989430715
-3.832809970424864
3.832809970424864
5.852511989430715
9.679948783602313

Intersections with y = -0.5pi:
-9.589326257258293
-6.019162645251092
-3.5939302541635367
3.5939302541635367
6.019162645251092
9.589326257258293

Intersections with y = 2.44:
-9.154607833717213
-6.658888236701425
6.658888236701425
9.154607833717213
```

```
Intersections with y = pi:
-9.07112290375477
-6.766048790452746
6.766048790452746
9.07112290375477

Intersections with y = 5.11:
-8.80581913799646
-7.088122226634311
7.088122226634311
8.80581913799646

Intersections with y = 2pi:
-8.606364308557731
-7.316132286079827
7.316132286079827
8.606364308557731

Intersections with y = 6.53:
-15.266241940756972
-7.370628484763827
7.370628484763827
15.266241940756972

Intersections with y = 0:
-9.42477796076938
-6.283185307179586
-3.141592653589793
3.141592653589793
6.283185307179586
9.42477796076938
```

Part II. Computer Exercise

$$F(x) = x(x^2 - 4)$$

C-1:

Adapting Newton's Method code to find the basin of attraction for each fixed point of F

Finding fixed points:

$$\begin{aligned} F(x) &= x \\ x(x^2 - 4) &= x \\ x^3 - 5x &= 0 \\ x(x^2 - 5) &= 0 \\ x &= 0, \pm\sqrt{5} \end{aligned}$$

Writing basin of attraction function:

In [ ]:

```
# Returns true if a seed value converges to fixed point within max iteration
def newton_method(f, fixed_pt, seed, max_iter, tol, max_val):
    iterations = 0

    while iterations < max_iter:
        # Calculate new guess
        seed = f(seed)
        iterations += 1

        # Check if orbit has diverged
        if abs(seed) > max_val:
            return False

        # Check if method has reached desired precision
        if abs(fixed_pt - seed) < tol:
            return True

    return False

# Returns a tuple with the minimum and maximum value of the basin of attract
def find_basin(f, fixed_pt, max_iter=25, init_step=0.1, tol=1e-9, max_val=1e
    # Find lower limit of basin of attraction
    step = init_step
    seed = fixed_pt - step
    infinite_limit = False

    # Move away from fp to the left, doubling step each time
    while newton_method(f, fixed_pt, seed, max_iter, tol, max_val):
        step *= 2
        seed -= step
        if seed < -max_val:
            lower_limit = -inf
            infinite_limit = True

    # Zero in on lower edge of basin
    if not infinite_limit:
        while step > tol:
            step *= 0.5
            if newton_method(f, fixed_pt, seed, max_iter, tol, max_val):
                seed -= step
            else:
                seed += step

        lower_limit = seed

    # Find upper limit of basin of attraction
    step = init_step
    seed = fixed_pt + step

    # Move away from fp to the right, doubling step each time
    while newton_method(f, fixed_pt, seed, max_iter, tol, max_val):
        step *= 2
        seed += step
        if seed > max_val:
            upper_limit = inf
```

```
            return (lower_limit, upper_limit)

    # Zero in on upper edge of basin
    while step > tol:
        step *= 0.5
        if newton_method(f, fixed_pt, seed, max_iter, tol, max_val):
            seed += step
        else:
            seed -= step

    upper_limit = seed

    # Return basin of attraction
    return (lower_limit, upper_limit)
```

Finding basins of attraction:

Basins of attraction on F:

In [ ]:
```
F = lambda x: x * (x**2 - 4)

print('-sqrt(5) basin of attraction:')
print(find_basin(F, -sqrt(5)), '\n')

print('0 basin of attraction:')
print(find_basin(F, 0), '\n')

print('sqrt(5) basin of attraction')
print(find_basin(F, sqrt(5)))
```

```
-sqrt(5) basin of attraction:
(-2.236067978244848, -2.2360679767547316)

0 basin of attraction:
(-7.450580596923829e-10, 7.450580596923829e-10)

sqrt(5) basin of attraction
(2.2360679767547316, 2.236067978244848)
```

Basins of attraction on N (fixed points will be 0's on F, so -2, 0, and 2):

In [ ]:
```
N = lambda x: x - F(x) / (3*x**2 - 4)

print('-2 basin of attraction:')
print(find_basin(N, -2), '\n')

print('0 basin of attraction:')
print(find_basin(N, 0), '\n')

print('2 basin of attraction:')
print(find_basin(N, 2))
```

```
-2 basin of attraction:
(-9401.959419571604, -1.1547714568674563)

0 basin of attraction:
(-0.8944271914660935, 0.8944271914660935)

2 basin of attraction:
(1.1547714568674563, 9401.959419571604)
```

C-2:

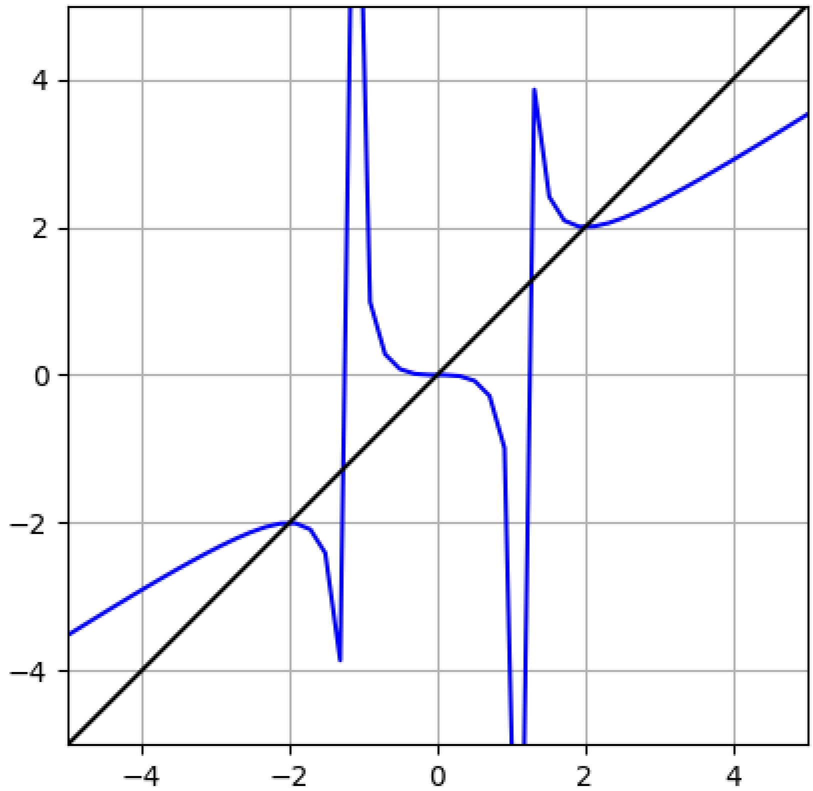
This structure of basins of attraction imply that certain values can cause Newton's method to fail. These values seem to coincide with the values where $F'(x)$ is near 0.

C-3:

The set of initial values in the domain of F for which Newton's method will fail is approximately $(-1.155, -0.894) \cup (0.894, 1.155)$. Let's look at a graph of N to see why this is the case

In [ ]:
```
xs = np.linspace(-10, 10, 100)
plt.plot(xs, N(xs), 'b')
plt.plot([-10,10], [-10,10], 'k')
plt.grid()
plt.axis('square')
plt.axis((-5,5,-5,5))
```

Out[ ]:
```
(-5.0, 5.0, -5.0, 5.0)
```



It looks like the regions where Newton's method failed lie near or within vertical asymptotes. This would imply that near 0 values of $F'(x)$ cause Newton's method to fail.