HW3

1 Homework 3: This HW is based on the code for Lecture 7 (Gradient Descent).

1.1 Instructions:

Place the answer to your code only in the area specified. Also, make sure to run all your code, meaning, press » to "Restart Kernel and Run All Cells". This should plot all outputs including your answers to homework questions. After this, go to file (top left) and select "Print". Save your file as a PDF and upload the PDF to Canvas.

2 Question:

Try to a build a simple optimizer to minimize:

`f(w) = a[0] + a[1]*w + a[2]*w^2 + ... + a[d]*w^d`

for the coefficients `a = [0,0.5,-2,0,1]`.

- Plot the function f(w) (2 points)
- Can you see where the minima is? (1 point)
- Write a function that outputs f(w) and its gradient (3 points).
- Run the optimizer on the function to see if it finds the minima (3 poimts).
- Print the funciton value and number of iterations (3 points).
- Instead of writing the function for a specific coefficient vector `a`, create a class that works for an arbitrary vector `a` (3 points).

You may wish to use the `poly.polyval(w,a)` method to evaluate the polynomial.

[ ]:
```
import asyncio
import typing
import numpy.polynomial.polynomial as poly
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

a = [0, 0.5, -2, 0, 1]

print(poly.polyval(-1, a))
print(poly.polyval(0, a))
```

```
print(poly.polyval(0.25, a))
print(poly.polyval(1, a))
print(poly.polyval(2, a))
```

```
-1.5
0.0
0.00390625
-0.5
9.0
```

Checking by hand

`f(w) = a[0] + a[1]*w + a[2]*w^2 + ... + a[d]*w^d` for the coefficients `a = [0,0.5,-2,0,1]`.

$f(w) = \frac{1}{2}w - 2w^2 + 0 + w^4$

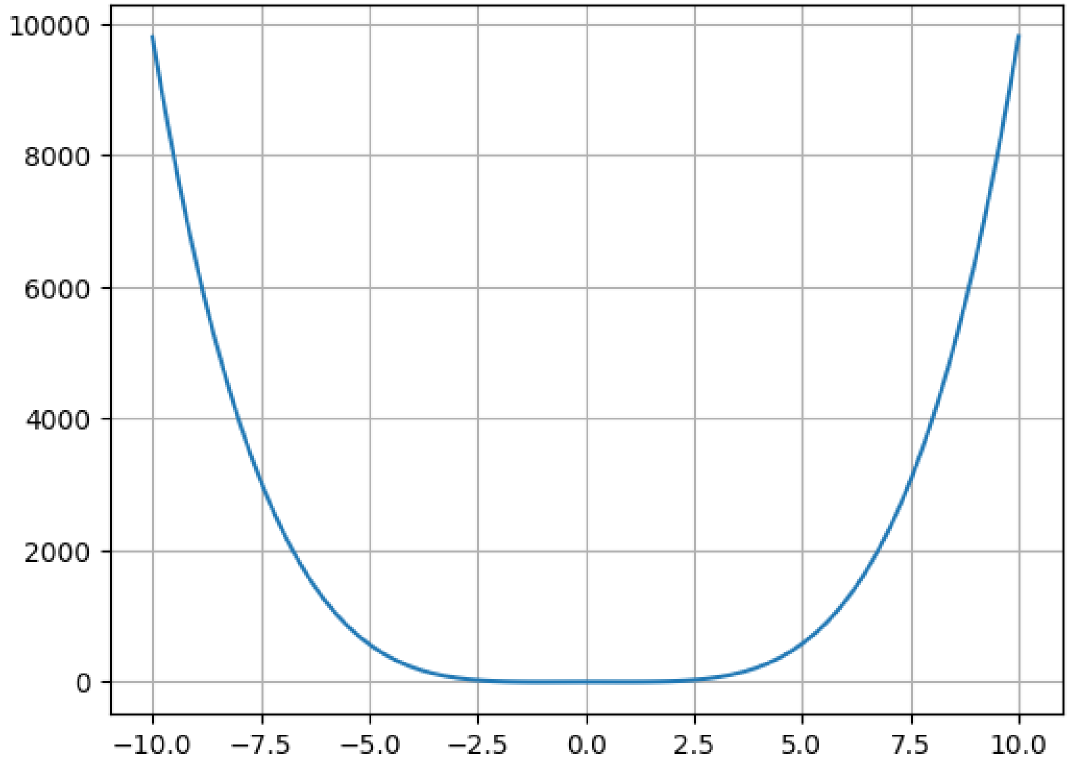
$w = 1; 0.5 - 2 + 1 = 0.5 - 1 = -0.5$

$w = 2; 1 - 8 + 16 = 1 + 8 = 9$

2.1 Plot the function f(w) (2 points)

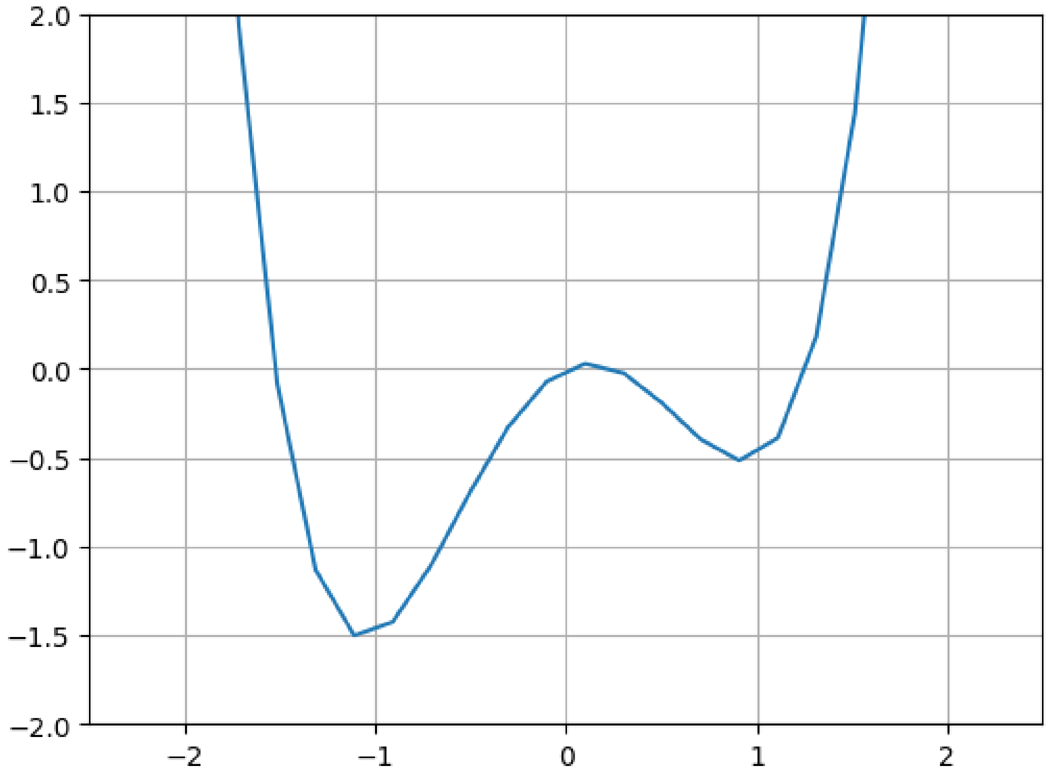
[ ]:
```
w = np.linspace(-10, 10, 100)
v = poly.polyval(w, a)

plt.plot(w,v)
plt.grid(True)
```



2.2 Can you see where the minima is? (1 point)

[ ]:
```
plt.plot(w,v)
plt.xlim(-2.5, 2.5)
plt.ylim(-2, 2)
plt.grid(True)
```



Looking at the macro graph, one could assume $f(0)$ would have been the minima but this is not true. Zooming in to the graph $f(\approx -1)$ is the true minima while $f(\approx 1)$ is a local minima.

2.3 Write a function that outputs f(w) and its gradient (3 points).

[ ]:
```
p = np.polynomial.Polynomial(a)
print(p)

d = p.deriv()
print(d)

assert p(1) == -0.5, "Our polynomial is wrong..."
```

```
0.0 + 0.5 x - 2.0 x**2 + 0.0 x**3 + 1.0 x**4
0.5 - 4.0 x + 0.0 x**2 + 4.0 x**3
```

[ ]:
```
# our gradient function
feval = lambda w: (p(w), p.deriv()(w))

w0 = np.random.randn(1)

# Perturb the point
```

```python
step = 1e-6
w1 = w0 + step*np.random.randn(1)

# Measure the function and gradient at w0 and w1
f0, fgrad0 = feval(w0)
print(f0, fgrad0)
f1, fgrad1 = feval(w1)
print(f1, fgrad1)

# Predict the amount the function should have changed based on the gradient
df_est = fgrad0.dot(w1-w0)
print(df_est)

# Print the two values to see if they are close
print("Actual f1-f0    = %12.4e" % (f1-f0))
print("Predicted f1-f0 = %12.4e" % df_est)

assert np.isclose(f1-f0, df_est), "Actual and Predicted value do not match!"
```

```
[-0.39107492] [1.53090923]
[-0.39107624] [1.53089991]
-1.3229735478990994e-06
Actual f1-f0    =  -1.3230e-06
Predicted f1-f0 =  -1.3230e-06
```

2.4 Run the optimizer on the function to see if it finds the minima (3 points).

[ ]:
```python
async def grad_opt_sim(feval: typing.Callable, winit: np.array, lr: float=1e-3,␣
 ↪nit: int=1000) -> typing.Tuple:
    """
    Simple gradient descent optimization

    feval:  A function that returns f, fgrad, the objective
            function and its gradient
    winit:  Initial estimate
    lr:     learning rate
    nit:    Number of iterations
    """
    # Initialize
    w0 = winit
    f0, fgrad0 = feval(w0)

    # Loop over iterations
    for it in range(nit):

        # Evaluate the function and gradient
        temp_f0, fgrad0 = feval(w0)
```

```
        # https://numpy.org/doc/stable/reference/generated/numpy.isclose.html
        # using to determine if we aren't making "enough" progress
        if np.isclose(temp_f0, f0, rtol=1e-05, atol=1e-08) and it > 0:
            break
        f0 = temp_f0

        # Take a gradient step
        w0 = w0 - lr*fgrad0

    return (w0, f0, it)

# We need multiple random points to find a true minima vs local minma
winits = np.array([np.random.randn(1)[0] for i in range(15)])

data = asyncio.gather(*[grad_opt_sim(feval, winit) for winit in winits])
await data
```

[ ]:
```
[(0.6367990229171355, -0.3271556089271581, 999),
 (-0.9937295213879747, -1.496402714509664, 999),
 (-1.070086776260489, -1.5139786632587726, 356),
 (0.9134230000018356, -0.5158346054591425, 999),
 (0.9413653484936304, -0.5163546200852214, 387),
 (-1.0443045777344577, -1.513933986900044, 874),
 (0.7405611127477955, -0.42510027589677113, 999),
 (-1.044276005002227, -1.5139304429142142, 544),
 (-1.0701016676897654, -1.5139768244001859, 305),
 (-1.0443148721131452, -1.513935261901335, 464),
 (0.5032053741302854, -0.18970560573021664, 999),
 (0.9413406378715726, -0.5163564053353972, 109),
 (-1.040450163973884, -1.513387974660023, 999),
 (0.895871212529062 7, -0.5130484076585351, 999),
 (-1.0442434880471358, -1.513926400526765, 436)]
```

2.5 Print the funciton value and number of iterations (3 points).

[ ]:
```
df = pd.DataFrame(columns=["init_x", "init_y", "grad_x", "grad_y"])
df["init_x"] = winits
df["init_y"] = p(winits)
df[["grad_x", "grad_y", "iters"]] = data.result()

df
```

[ ]:

	init_x	init_y	grad_x	grad_y	iters
0	0.144875	0.030900	0.636799	-0.327156	999.0
1	0.077826	0.026836	-0.993730	-1.496403	999.0
2	-1.837919	3.735669	-1.070087	-1.513979	356.0

```
3   0.259492 -0.000392  0.913423 -0.515835  999.0
4   1.097689 -0.409162  0.941365 -0.516355  387.0
5  -0.031867 -0.017963 -1.044305 -1.513934  874.0
6   0.154071  0.030123  0.740561 -0.425100  999.0
7  -0.433276 -0.556853 -1.044276 -1.513930  544.0
8  -1.383997 -0.853956 -1.070102 -1.513977  305.0
9  -0.593715 -0.877598 -1.044315 -1.513935  464.0
10  0.137966  0.031276  0.503205 -0.189706  999.0
11  0.952835 -0.515099  0.941341 -0.516356  109.0
12  0.040085  0.016832 -1.040450 -1.513388  999.0
13  0.214484  0.017352  0.895871 -0.513048  999.0
14 -0.649219 -0.989930 -1.044243 -1.513926  436.0
```

2.6 Instead of writing the function for a specific coefficient vector a, create a class that works for an arbitrary vector a (3 points).

[ ]:
```
# It should already work for any arbitrary coef vector

class GradientPolynomial(object):
    def __init__(self, a: np.array) -> None:
        self.__p = np.polynomial.Polynomial(a)
        self.__inits = []
        self.__vals = []

    def __repr__(self) -> str:
        return f"<GradientPolynomial: f{self.__p}>"

    @property
    def __d(self) -> np.polynomial.Polynomial:
        return self.__p.deriv()

    def feval(self, w: np.array) -> typing.Tuple:
        return (self.__p(w), self.__d(w))

    async def optimizer(self, winit: np.array, eta=1e-3, nit=1000) -> typing.
 ↪Tuple:
        w0 = winit
        f0, fgrad0 = self.feval(w0)

        for it in range(nit):
            temp_f0, fgrad0 = self.feval(w0)

            # https://numpy.org/doc/stable/reference/generated/numpy.isclose.
 ↪html
            # using to determine if we aren't making "enough" progress
            if np.isclose(temp_f0, f0, rtol=1e-07, atol=1e-08) and it > 0:
                break
```

```
            f0 = temp_f0

            w0 -= eta*fgrad0

        return (w0, f0, it)


    async def plot(self,
                   n: int=1000,
                   domain: tuple=(-10, 10),
                   initial_points: typing.Union[np.array, int, None]=None,
                   eta: float=1e-3,
                   nit: int=1000,
                   xlim: typing.Union[tuple, None]=None,
                   ylim: typing.Union[tuple, None]=None) -> pd.DataFrame:
        x, y = self.__p.linspace(n, domain=domain)

        if isinstance(initial_points, int):
            self.__inits = np.random.choice(x, initial_points)
            self.__vals = []
        elif isinstance(initial_points, (np.array, list, tuple)):
            self.__inits = initial_points
            self.__vals = []
        elif not self.__inits:
            raise Exception("No array of initial points or number of inital␣
↪points to be generated was provided")

        if not self.__vals:
            self.__vals = asyncio.gather(*[self.optimizer(i, eta, nit) for i in␣
↪self.__inits])
            await self.__vals

        df = pd.DataFrame(columns=["init_x", "init_y", "grad_x", "grad_y"])
        df["init_x"] = self.__inits
        df["init_y"] = self.__p(self.__inits)
        df[["grad_x", "grad_y", "iters"]] = self.__vals.result()

        plt.plot(x, y)
        plt.plot(*self.__d.linspace(n, domain=domain))
        plt.scatter(df["init_x"], df["init_y"], marker='o', c='black', s=30)
        plt.scatter(df["grad_x"], df["grad_y"], marker='o', c='red', s=30)

        if xlim:
            plt.xlim(*xlim)
        if ylim:
            plt.ylim(*ylim)
        plt.grid(True)
```

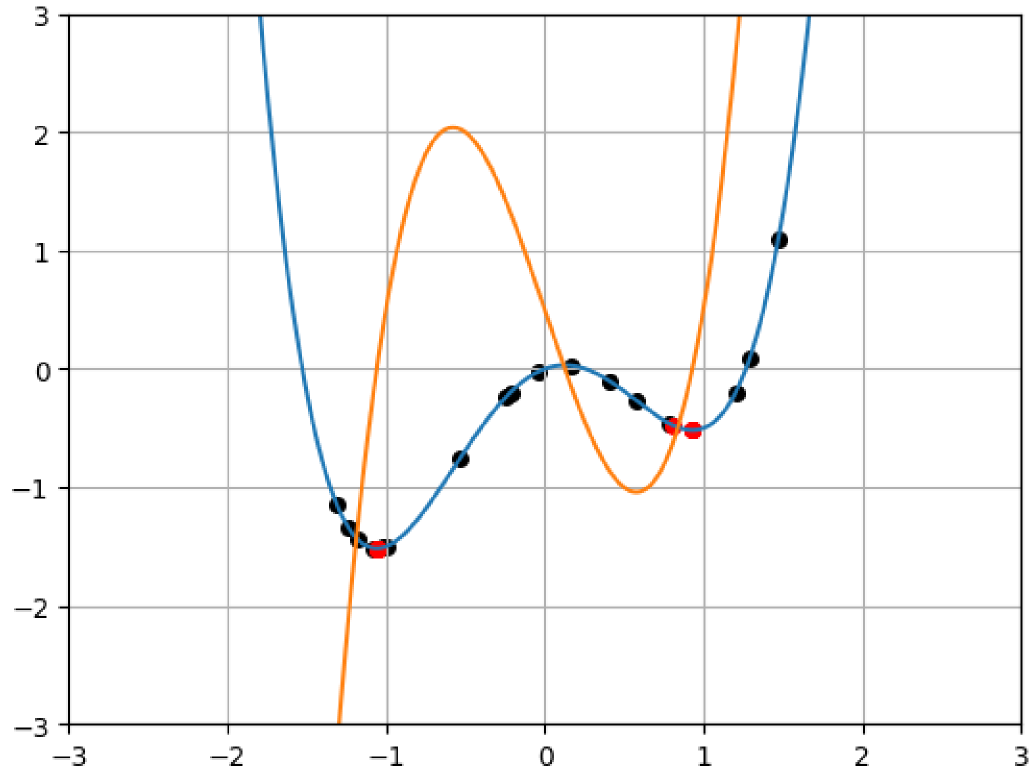
```python
        return df

gp = GradientPolynomial(a)
print(gp)
await gp.plot(domain=(-2, 2), initial_points=20, xlim=(-3, 3), ylim=(-3, 3))
```

```
<GradientPolynomial: f0.0 + 0.5 x - 2.0 x**2 + 0.0 x**3 + 1.0 x**4>
```

[ ]:

	init_x	init_y	grad_x	grad_y	iters
0	-1.871872	4.333601	-1.058786	-1.514745	594.0
1	-1.179179	-1.437128	-1.058784	-1.514745	461.0
2	0.410410	-0.103297	0.926385	-0.516697	999.0
3	1.291291	0.091113	0.931616	-0.516744	809.0
4	1.203203	-0.197965	0.931616	-0.516744	782.0
5	1.735736	3.919145	0.931620	-0.516744	867.0
6	0.582583	-0.272320	0.929096	-0.516743	999.0
7	-1.947948	5.835263	-1.058783	-1.514745	598.0
8	-1.239239	-1.332630	-1.058783	-1.514745	496.0
9	1.471471	1.093493	0.931617	-0.516744	842.0
10	-1.307307	-1.150898	-1.058791	-1.514745	521.0
11	-1.083083	-1.511589	-1.058786	-1.514745	309.0
12	-0.990991	-1.495174	-1.056123	-1.514745	423.0
13	-0.214214	-0.196777	-1.056123	-1.514745	923.0
14	0.782783	-0.458645	0.929180	-0.516744	793.0
15	-0.242242	-0.235040	-1.056118	-1.514745	902.0
16	-0.530531	-0.748969	-1.056114	-1.514745	738.0
17	0.166166	0.028623	0.812140	-0.477669	999.0
18	-0.038038	-0.021911	-1.053764	-1.514689	999.0
19	-1.023023	-1.509342	-1.056115	-1.514745	348.0



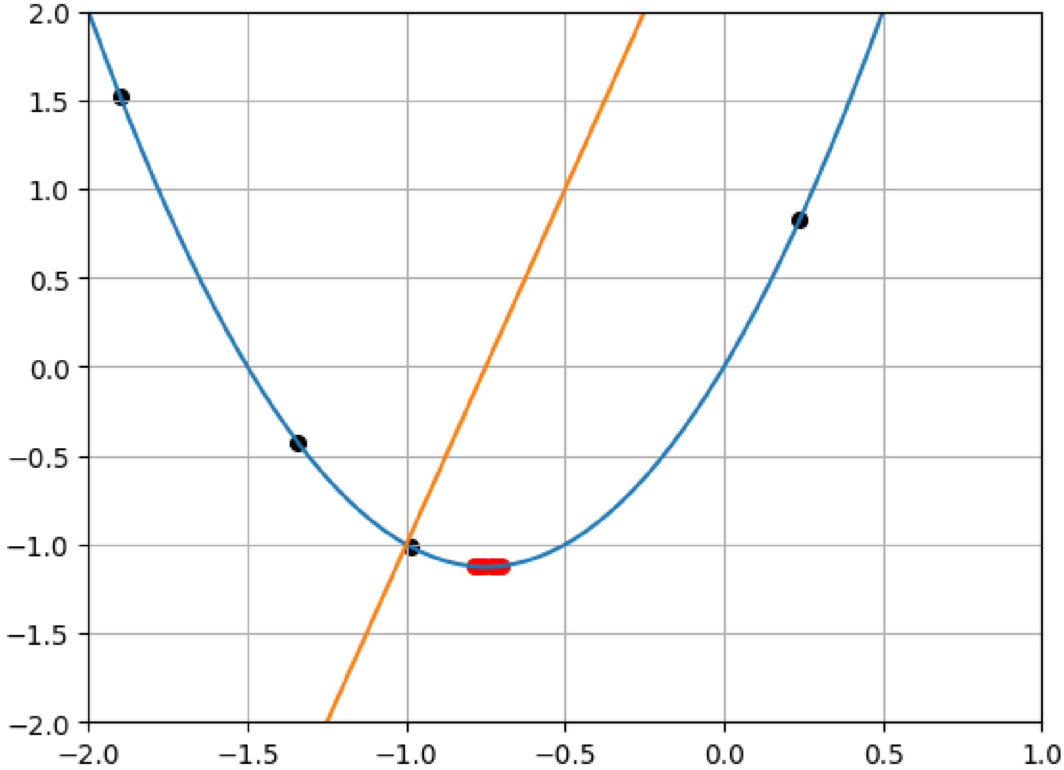
[ ]:

```
gp = GradientPolynomial([0, 3, 2])
print(gp)
await gp.plot(domain=(-3, 3), initial_points=10, xlim=(-2, 1), ylim=(-2, 2))
```

```
<GradientPolynomial: f0.0 + 3.0 x + 2.0 x**2>
```

[ ]:

	init_x	init_y	grad_x	grad_y	iters
0	-1.342342	-0.423261	-0.760762	-1.124766	999.0
1	0.237237	0.824275	-0.732063	-1.124351	999.0
2	0.795796	3.653969	-0.721914	-1.123410	999.0
3	-2.093093	2.482798	-0.774403	-1.123799	999.0
4	0.525526	2.128931	-0.726825	-1.123917	999.0
5	-0.987988	-1.011723	-0.754324	-1.124962	999.0
6	-2.585586	5.613749	-0.783351	-1.122757	999.0
7	-1.900901	1.524146	-0.770911	-1.124118	999.0
8	1.798799	11.867751	-0.703690	-1.120676	999.0
9	-2.393393	4.276484	-0.779859	-1.123202	999.0

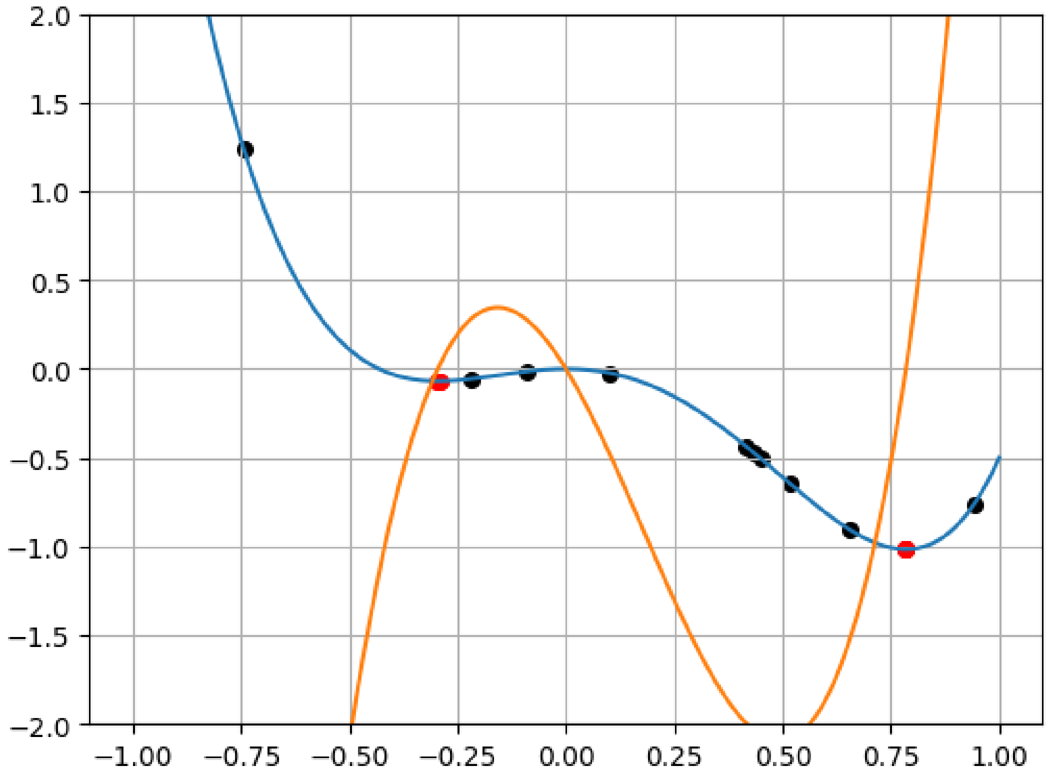


[ ]:
```
gp = GradientPolynomial([0, 0, -2, -3, 4, 0.5])
print(gp)
await gp.plot(domain=(-1, 1), initial_points=10, ylim=(-2, 2))
```

```
<GradientPolynomial: f0.0 + 0.0 x - 2.0 x**2 - 3.0 x**3 + 4.0 x**4 + 0.5 x**5>
```

[ ]:

	init_x	init_y	grad_x	grad_y	iters
0	-0.089089	-0.013503	-0.291243	-0.067801	999.0
1	0.431431	-0.467121	0.784172	-1.015678	460.0
2	0.451451	-0.508118	0.784170	-1.015678	450.0
3	0.417417	-0.438893	0.784172	-1.015678	467.0
4	0.653654	-0.902494	0.784173	-1.015678	346.0
5	0.517518	-0.645980	0.784177	-1.015678	419.0
6	-0.743744	1.238042	-0.296714	-0.067857	995.0
7	-0.219219	-0.055524	-0.295170	-0.067857	940.0
8	0.941942	-0.762095	0.785444	-1.015678	316.0
9	0.099099	-0.022170	0.784177	-1.015679	754.0



2.7 Extra/Alt Implementation

[ ]:
```
x, y = p.linspace(1000, domain=(-3, 3))

# choose 10 random points from domain -3 to 3
initial_points = np.random.choice(x, 10)
print(f"Random initial points: {initial_points}")

async def gradient_descent(pt, eta=0.01, steps=5000, allow_early_out=True) ->␣
 ↪float:
    result = pt
    for i in range(steps):
        temp_result = result - eta * d(result)
        if allow_early_out and (p(temp_result) >= p(result)):
            break
        result = temp_result
    return result

data = asyncio.gather(*[gradient_descent(init_pt) for init_pt in␣
 ↪initial_points])
await data
```

```python
data = np.array(data.result())

print(f"Gradient Descent x-values: {data}")
print(f"Gradient Descent y-valyes: {[p(d) for d in data]}")

# this also works for plotting
plt.plot(x, y)
plt.plot(*d.linspace(1000, domain=(-10, 10)))
plt.scatter(initial_points, p(initial_points), marker='o', c='black', s=30)
plt.scatter(data, p(data), marker='o', c='red', s=30)
plt.xlim(-3.5, 3.5)
plt.ylim(-3.5, 3.5)
plt.grid(True)
```

```
Random initial points: [ 1.67867868  1.1981982  -2.89189189 -1.996997
1.88288288 -2.58558559
  0.8018018   2.24324324  0.36936937 -1.04804805]
Gradient Descent x-values: [ 0.93040295  0.93040294 -1.05745378 -1.05745379
0.93040296 -1.05745379
  0.93040291  0.93040295  0.93040291 -1.05745376]
Gradient Descent y-valyes: [-0.5167485082852478, -0.5167485082852483,
-1.5147536412757052, -1.5147536412757043, -0.5167485082852467,
-1.5147536412757039, -0.5167485082852478, -0.5167485082852481,
-0.5167485082852488, -1.514753641275705]
```