

Week 2 Optimisation for Machine Learning

Neimhin Robinson Gunning, 16321701

January 31, 2024

1 Part (a)

1.1 Part (a) (i)

To represent the expression x^4 using sympy we first create a symbol object for x (line 1 in Listing 3), and then raise it to the fourth power using standard python arithmetic notation. The `**` operation is overloaded such that `x**4` yields an object representing the expression x^4 . If we assume $y(x) = x^4$ then the derivative dy/dx is found with `(x**4).diff()`. Running the code we find this derivative to be $4x^3$.

Listing 1: Source code to find the derivative of x^4 using sympy. The result is $4x^3$.

```
src/ai.py Thu Jan 11 09:02:38 2024 1
1: import sympy as sp
2: x = sp.symbols('x')
3: y = x**4
4: print(y.diff()) # 4*x**3
5:
```

1.2 Part (a) (ii)

Listing 2: Ellided source code to estimate the derivative of x^4 using the finite difference method. The function `diff_with_pert` can be used to estimate the derivative of an arbitrary function of x . The full script is in the appendix and named `src/a11.py`.

```
src/finite_diff_eg.py Wed Jan 31 14:28:44 2024 1
1: import sympy as sp
2: # finite difference
3: def diff_with_pert(f, xval, pert=0.01):
4:     global x
5:     delta_x = pert
6:     return (f.subs(x, xval + delta_x) - f.subs(x, xval)) / (delta_x)
7: x = sp.symbols('x')
8: y = x**4
9: dydx = y.diff()
10: analytic_ys = [dydx.subs(x, i) for i in my_range()]
11: # ...
12: for pert in np.array([0.01, 0.1, 0.15]):
13:     dydx_finite = diff_with_pert(y, x, pert=pert)
14:     # ...
15:
```

1.3 Part (a) (iii)

The accuracy of the finite difference approximation of the derivative is greater for smaller values of δ . The reason for this is related to the definition of the derivative. The derivative is defined as

$$\lim_{\delta \rightarrow 0} \frac{f(x + \delta) - f(x)}{x + \delta - x}$$

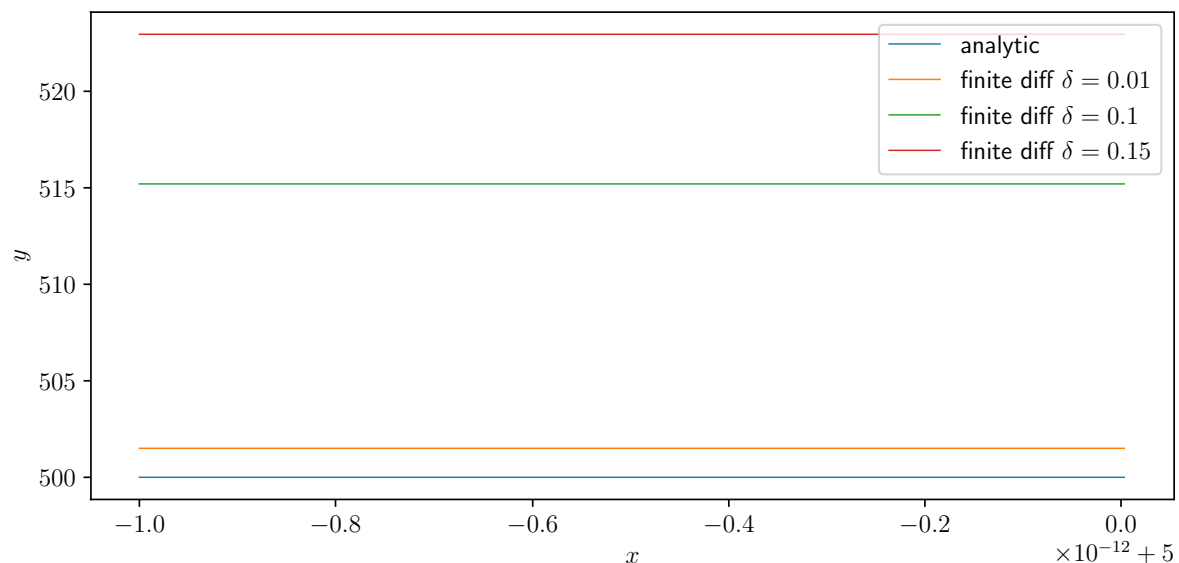


Figure 1: A comparison of the analytic derivation of $dy/dx = 4x^3$ and some approximations of dy/dx using the finite difference method. The finite difference approximation was $\frac{\hat{dy}}{dx}(x) = \frac{(x+\delta)^4 - x^4}{\delta}$.

, i.e. the derivative is the function approached by this expression as δ approaches 0. As we decrease δ and apply the finite difference method we are approaching the true derivative.

2 (b)

2.1 (b) (i)

Listing 3 shows ellided source code for a python class that can be used to run gradient descent. Ellided functions allow for specification of the function to be optimized, the convergence condition, the max number of iterations etc. Line 13 calculates the gradient at the current value, and line 14 updates the value based on the gradient and step size. The convergence condition is a function of x_n and x_{n-1} . Examples of this class in use are in the appendix, e.g. in `src/bi.py`, in which the gradient function is computed with `sympy`.

Listing 3: Ellided source code a python class that implements gradient descent with a fixed step size.

```
src/gradient_descent_listing.py    Wed Jan 31 15:38:46 2024    1
1: class GradientDescent():
2:     # ...
3:     def iterate(self):
4:         import math
5:         x_value = self._start
6:         old_x_value = None
7:         iteration = 0
8:         while True:
9:             yield [iteration, float(x_value), float(self._function(x_value))]
10:            iteration += 1
11:            if self._max_iter > 0 and iteration > self._max_iter:
12:                break
13:            grad_value = self._gradient(x_value)
14:            x_value -= self._step_size * grad_value # Update step
15:            if old_x_value is not None and self._converged(x_value, old_x_value):
16:                yield [iteration, float(x_value), float(self._function(old_x_value))]
17:                print("converged")
18:                break
19:            old_x_value = x_value
```

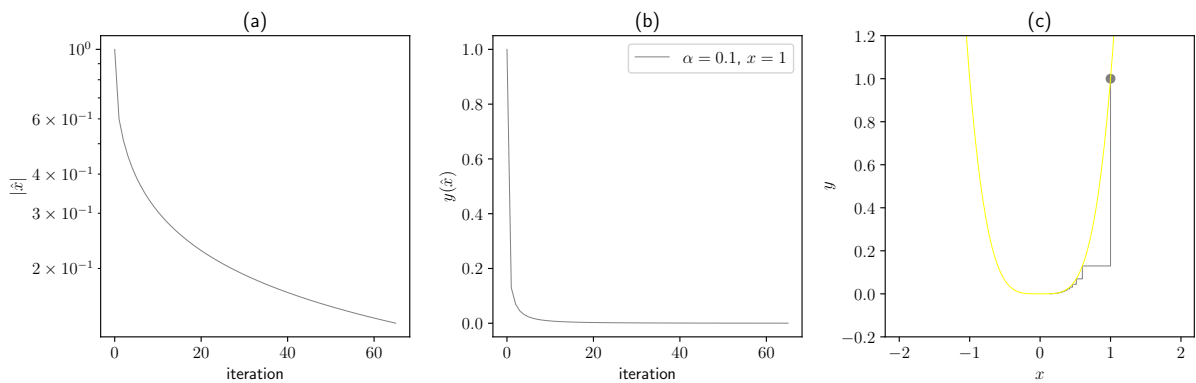


Figure 2: Running gradient descent on x^4 with initial value $x_0 = 1$ and step size $\alpha = 0.1$. The yellow line in subplot (c) is x^4 .

2.2 (b) (ii)

In Figure 2 is a plot of gradient descent on x^4 with $x_0 = 1$ and $\alpha = 0.1$. We see that updates to $|\hat{x}|$ and $y(\hat{x})$ are initially large but quickly become tiny. Because x^4 is very flat around the minimum (see subplot (c)) we have a very small gradient, and thus very small step sizes, so the algorithm updates extremely slowly towards $\hat{x} = 0$, but cannot reach it in a finite number of iterations.

In Figure 3 several runs of gradient descent on x^4 are visualized. Subplot (a) shows $|\hat{x}|$ for each iteration, which is the distance of the estimate \hat{x} from the optimal value, $x = 0$. With initial value $x = 1$ and step size $\alpha = 0.1$ we see that the steps are initially large, i.e. moving quickly to the optimum, but on successive iterations the steps slow down dramatically. In subplot (b) we see the value of $y(\hat{x})$ on each iteration.

The convergence criterion for gradient descent adopted here is when $|\alpha \Delta x| < 0.001$ or x is $\pm \infty$ ¹. For the function x^4 we can see from Table 1 that the final guess, i.e. the estimate of $\arg \min x^4$ is most accurate and converges fastest with larger values of α , **unless** α is too large and the iterations of gradient descent degenerate, i.e. subsequent iterations give worse or unchanged estimates. The reason for this is that the gradient of x^4 is tiny as x approaches the minimum. So, a smaller value of α means the convergence condition, $|\alpha \Delta x| < 0.001$, can be met with a higher value of Δx , but in the case of x^4 , Δx is higher when x is further from the optimal value 0. This is why the final guess is worse for smaller values of α , but why does a smaller α lead to slower convergence time?

Since α is constant for a single run of gradient descent, the convergence condition depends on Δx , namely we need $|\Delta x| < 0.001/\alpha$. A larger α entails a larger step, but there are two factors at play here:

1. A larger step size may entail a smaller Δx on the subsequent iteration.
2. A larger step size moves us away from meeting the convergence condition.

It turns out for x^4 that the first of these factors is more important, in cases where gradient descent is converging. Thus larger α results in faster convergence (for x^4).

¹Meaning the floating point values `math.inf` or `-math.inf`

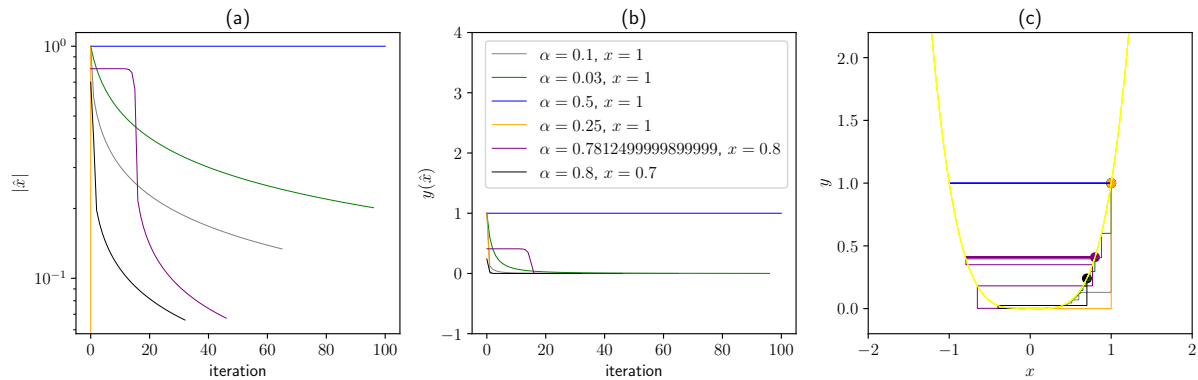


Figure 3: Various runs of gradient descent on x^4 . One run, where $\alpha = 0.5$ and $x_0 = 1$, is stuck in a 2-step loop, flipping \hat{x} back and forth between 1 and -1. The yellow line in subplot (c) is x^4 .

Table 1: Various runs of gradient descent on x^4 with different outcomes. The maximum number of iterations was set to 100, and the convergence threshold to 0.001. Some of the α values were purposefully selected to lie near the margin of non-convergence, which can be calculated with $\alpha_m = \frac{2x_0}{4x_0^3}$. So rows 8 and 9 use $\alpha_m \pm \epsilon$.

	α	start	convergence time	final guess
0	0.01	1.0	100	0.3323366923926806
1	0.02	1.0	100	0.24119415115009837
2	0.03	1.0	97	0.20146846304616808
3	0.04	1.0	89	0.18252923619780276
4	0.05	1.0	83	0.1691984331910481
5	0.1	1.0	66	0.13379629115760108
6	0.4999999999999999	1.0	55	-0.07827643923095963
7	0.5	1.0	100	-1.0
8	0.7812500000000009	0.8	29	inf
9	0.7812499999999989	0.8	24	0.02474194758833489
10	0.8	0.7	33	-0.06609857197723801

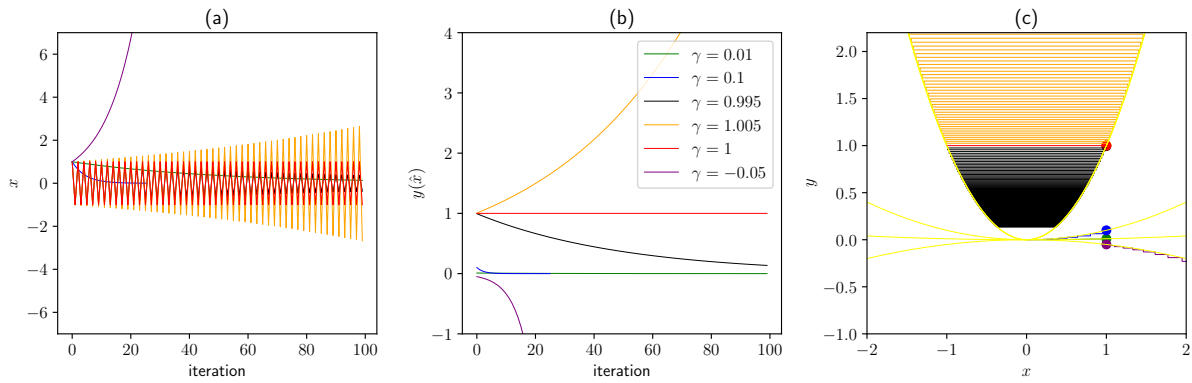


Figure 4: Various runs of gradient descent on γx^2 . Subplot (a) shows the estimate of x for each iteration. The yellow lines in subplot (c) are plots of the various γx^2 functions.

3 (c)

3.1 (c) (i) Optimising γx^2

Firstly, note that if γ is negative, then γx^2 and $\gamma|x|$ have no minimum, and thus gradient descent will behave weirdly. For instance, in my implementation, with enough iterations it will actually finish with `math.inf`, which is reasonable. If $\gamma = 0$ then gradient descent terminates immediately, and any value for x minimizes $\gamma x^2 = 0x^2$. Increasing γ increases the magnitude of the gradient, resulting in larger steps, but the convergence time and accuracy will then depend on the initial value.

Let $d_0 = |x_0 - \arg \min_x \gamma x^2|$ be the initial distance to the optimal value of x . Let's assume x_0 is 'far' from

$$\arg \min_x \gamma x^2$$

, i.e. d_0 is large, meaning gradient descent will at first take several steps in the same direction. Increasing γ to γ' , i.e. $\gamma' > \gamma$, results in a larger gradient $2\gamma'x'_0 > 2\gamma x_0$ meaning the first step is larger. If after the first iteration the distance has improved, $d_1 < d_0$ then gradient descent will converge, if not it will diverge. Let's assume we are in a converging condition. While the second gradient $2\gamma'x'_1$ is not necessarily larger than $2\gamma x_1$, the updated x'_1 is now closer to the minimum than x_1 . The result of these dynamics is that with a larger γ' there is a fewer number iterations until the first 'overshoot'.

For certain values this means that convergence time is smaller for the larger γ' than for γ .

3.2 (c) (ii) Optimising $\gamma|x|$

```
1: import sympy as sp
2: import numpy as np
3: import matplotlib.pyplot as plt
4:
5: LINEWIDTH = 0.7
6:
7:
8: def my_range():
9:     return np.arange(5, 5.00000000000001, step=0.00000000000001)
10:
11:
12: # finite difference
13: def diff_with_pert(f, xval, pert=0.01):
14:     global x
15:     delta_x = pert
16:     return (f.subs(x, xval + delta_x) - f.subs(x, xval)) / (delta_x)
17:
18:
19: x = sp.symbols('x')
20: y = x**4
21: dydx = y.diff()
22: analytic_ys = [dydx.subs(x, i) for i in my_range()]
23:
24: plt.figure(figsize=(8, 4))
25: plt.plot(
26:     np.array(list(my_range())),
27:     analytic_ys,
28:     linewidth=LINEWIDTH,
29:     label="analytic")
30:
31: for pert in np.array([0.01, 0.1, 0.15]):
32:     dydx_finite = diff_with_pert(y, x, pert=pert)
33:     finite_diff_ys = [dydx_finite.subs(x, i) for i in my_range()]
34:     plt.plot(
35:         my_range(),
36:         finite_diff_ys,
37:         linewidth=LINEWIDTH,
38:         label=f"finite diff  $\Delta$ ={pert}")
39: plt.xlabel("$x$")
40: plt.ylabel("$y$")
41: plt.legend()
42: plt.tight_layout()
43: plt.savefig("fig/finite-diff.pdf")
```

```
1: import sympy as sp
2: x = sp.symbols('x')
3: y = x**4
4: print(y.diff())      # 4*x**3
5:
```

```
1: import sympy as sp
2: import sys
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import seaborn as sns
6: import pandas as pd
7: from lib import GradientDescent
8:
9: LINEWIDTH = 0.5
10: x = sp.symbols('x')
11: y = x**4
12: dydx = y.diff()
13:
14: fig, ax = plt.subplots(1, 3, figsize=(12, 4))
15:
16: blowup = 0.8
17:
18: # alpha * (blowup ** 3) * 4 = 1.2
19:
20: results = {
21:     "alpha": [],
22:     "start": [],
23:     "convergence time": [],
24:     "final guess": [],
25: }
26: iota = 0.000000000001
27: for (step_size, start, color) in [
28:     (0.1, 1, 'gray'),
29:     (0.03, 1, 'green'),
30:     (0.5, 1, 'blue'),
31:     (0.25, 1, 'orange'),
32:     # ((2*blowup)/((blowup**3)*4) + iota, blowup, 'red'),
33:     ((2*blowup)/((blowup**3)*4) - iota, blowup, 'purple'),
34:     (0.8, 0.7, 'black')
35: ]:
36:     print(step_size, start, color)
37:     g = GradientDescent()
38:     g.max_iter(100)
39:     g.step_size(step_size)
40:     g.start(start)
41:     g.function(lambda x1: float(y.subs(x, x1)))
42:     y_diff = y.diff()
43:     g.gradient(lambda x1: float(y_diff.subs(x, x1)))
44:     g.debug(True)
45:
46:     def is_inf(x):
47:         import math
48:         if x == math.inf or x == -math.inf:
49:             return True
50:
51:     def converged(x1, x2):
52:         if is_inf(x1) or is_inf(x2):
53:             return True
54:         abs = np.abs(x1-x2)
55:         print(abs, x1, x2)
56:         return abs < 0.001
57:     g.converged(converged)
58:     iterations, estimates, y_of_x = zip(*[
59:         (x[0], x[1], x[2]) for x in g.iterate())
60:     results["alpha"].append(step_size)
61:     results["start"].append(start)
62:     results["convergence time"].append(len(iterations))
63:     results["final guess"].append(estimates[-1])
64:     print('y_of_x', y_of_x)
65:     print('iterations', iterations)
66:     print('estimates', estimates)
67:     sns.lineplot(
68:         x=iterations,
69:         y=np.abs(np.array(estimates)),
70:         ax=ax[0],
71:         linewidth=LINEWIDTH,
72:         legend=False,
73:         color=color,
74:         label=f"$\\alpha={step_size}$, $x={start}$")
75:     sns.lineplot(
76:         x=iterations,
77:         y=y_of_x,
78:         ax=ax[1],
79:         linewidth=LINEWIDTH,
80:         color=color,
81:         label=f"$\\alpha={step_size}$, $x={start}$")
82:     ax[2].step(
83:         estimates,
84:         y_of_x,
85:         linewidth=LINEWIDTH,
86:         color=color,
87:         label=f"$\\alpha={step_size}$, $x={start}$")
88:     xs = np.arange(-2, 2, 0.01)
89:     ys = [y.subs(x, xi) for xi in xs]
90:     ax[2].plot(
91:         xs,
92:         ys,
93:         linewidth=LINEWIDTH,
94:         label="$x^4$",
95:         color='yellow',
96:     )
97:     ax[2].scatter(
98:         start,
99:         g._function(start),
100:         color=color)
```



```
101:
102:
103: ax[0].set_ylabel("$|\\hat{x}|$")
104: ax[0].set_xlabel("iteration")
105: ax[0].set_yscale('log')
106: ax[0].set_title("(a)")
107: ax[1].set_ylabel("$y(\\hat{x})$")
108: ax[1].set_xlabel("iteration")
109: ax[1].set_title("(b)")
110: ax[2].set_xlabel("$x$")
111: ax[2].set_ylabel("$y$")
112: ax[2].set_title("(c)")
113: # ax[0].set_ylim([-7, 7])
114: ax[1].set_ylim([-1, 4])
115: ax[2].set_ylim([-0.2, 2.2])
116: ax[2].set_xlim([-2, 2])
117: plt.tight_layout()
118:
119: outfile = "fig/gradient-descent-b1.pdf"
120: if len(sys.argv) > 1:
121:     outfile = sys.argv[1]
122: plt.savefig(outfile)
123: df = pd.DataFrame(results)
124: print(df)
125: df.to_csv("fig/gradient-descent-b1.csv")
```

```
1: import sympy as sp
2: import sys
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import seaborn as sns
6: import pandas as pd
7: from lib import GradientDescent
8:
9: LINEWIDTH = 0.5
10: CONVERGENCE_THRESHOLD = 0.001
11: x = sp.symbols('x')
12: y = x**4
13: dydx = y.diff()
14:
15: fig, ax = plt.subplots(1, 3, figsize=(12,4))
16:
17: blowup = 0.8
18:
19: # alpha * (blowup ** 3) * 4 = 1.2
20:
21: results = {
22:     "$\\alpha$": [],
23:     "start": [],
24:     "convergence time": [],
25:     "final guess": [],
26: }
27: iota = 0.000000000000000001
28: for (step_size, start, color) in [
29:     (0.01, 1, 'green'),
30:     (0.02, 1, 'green'),
31:     (0.03, 1, 'green'),
32:     (0.04, 1, 'green'),
33:     (0.05, 1, 'green'),
34:     (0.1, 1, 'green'),
35:     (0.5 - iota, 1, 'green'),
36:     (0.5, 1, 'blue'),
37:     ((2*blowup)/((blowup**3)*4) + iota, blowup, 'red'),
38:     ((2*blowup)/((blowup**3)*4) - iota, blowup, 'purple'),
39:     (0.8, 0.7, 'black')
40: ]:
41:     print(step_size, start, color)
42:     g = GradientDescent()
43:     g.max_iter(99)
44:     g.step_size(step_size)
45:     g.start(start)
46:     g.function(lambda x1: float(y.subs(x, x1)))
47:     y_diff = y.diff()
48:     g.gradient(lambda x1: float(y_diff.subs(x, x1)))
49:     g.debug(True)
50:
51:     def converged(x1, x2):
52:         import math
53:         if x1 == math.inf or x2 == math.inf:
54:             print(x1, x2, x1 == math.inf, x1 is math.inf)
55:             return True
56:         abs = np.abs(x1-x2)
57:         print(abs, x1, x2)
58:         return abs < CONVERGENCE_THRESHOLD
59:
60:     g.converged(converged)
61:     iterations, estimates, y_of_x = zip(*[
62:         (x[0], x[1], x[2]) for x in g.iterate())
63:     results["$\\alpha$"].append(step_size)
64:     results["start"].append(start)
65:     results["convergence time"].append(len(iterations))
66:     results["final guess"].append(estimates[-1])
67:     print('y_of_x', y_of_x)
68:     print('iterations', iterations)
69:     print('estimates', estimates)
70:     sns.lineplot(
71:         x=iterations,
72:         y=estimates,
73:         ax=ax[0],
74:         linewidth=LINEWIDTH,
75:         legend=False,
76:         color=color,
77:         label=f"$\\alpha$={step_size}$, $x$={start}$")
78:     sns.lineplot(
79:         x=iterations,
80:         y=y_of_x,
81:         ax=ax[1],
82:         linewidth=LINEWIDTH,
83:         color=color,
84:         label=f"$\\alpha$={step_size}$, $x$={start}$")
85:     ax[2].step(
86:         estimates,
87:         y_of_x,
88:         linewidth=LINEWIDTH,
89:         color=color,
90:         label=f"$\\alpha$={step_size}$, $x$={start}$")
91:     xs = np.arange(-2, 2, 0.01)
92:     ys = [ y.subs(x, xi) for xi in xs]
93:     ax[2].plot(
94:         xs,
95:         ys,
96:         linewidth=LINEWIDTH,
97:         label=f"$x^4$",
98:         color='yellow',
99:     )
100:     ax[2].scatter(
```

```
101:         start,
102:         g._function(start),
103:         color=color)
104:
105:
106: ax[0].set_ylabel("estimate of  $\arg\min_x x^4$ ")
107: ax[0].set_xlabel("iteration")
108: ax[1].set_ylabel("$y(\hat{x})$")
109: ax[1].set_xlabel("iteration")
110: ax[0].set_ylim([-7, 7])
111: ax[1].set_ylim([-1, 10])
112: ax[2].set_ylim([-0.2, 2.2])
113: ax[2].set_xlim([-2, 2])
114: plt.tight_layout()
115:
116: outfile = "fig/gradient-descent-b2.pdf"
117: if len(sys.argv) > 1:
118:     outfile = sys.argv[1]
119: plt.savefig(outfile)
120: df = pd.DataFrame(results)
121: print(df)
122: df.to_csv("fig/gradient-descent-b2.csv", index=False)
```

```
1: import sympy as sp
2: import sys
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import seaborn as sns
6: from lib import GradientDescent
7:
8: LINEWIDTH = 0.7
9: x = sp.symbols('x')
10: y = x**4
11: dydx = y.diff()
12:
13: fig, ax = plt.subplots(1, 2)
14:
15: for step_size in np.array([0.5]):
16:     for start in np.array([1.00001]):
17:         print(step_size, start)
18:         g = GradientDescent()
19:         g.max_iter(100)
20:         g.step_size(step_size)
21:         g.start(start)
22:         g.function(lambda x1: float(y.subs(x, x1)))
23:         y_diff = y.diff()
24:         g.gradient(lambda x1: float(y_diff.subs(x, x1)))
25:         g.debug(True)
26:
27:         def converged(x1, x2):
28:             abs = np.abs(x1-x2)
29:             print(abs, x1, x2)
30:             return abs < 0.001
31:         g.converged(converged)
32:         iterations, estimates, y_of_x = zip(*[(x[0], x[1], x[2]) for x in g.iterate()])
33:         print('y_of_x', y_of_x)
34:         print('iterations', iterations)
35:         print('estimates', estimates)
36:         sns.lineplot(
37:             x=iterations,
38:             y=estimates,
39:             ax=ax[0],
40:             linewidth=LINEWIDTH,
41:             legend=False,
42:             label=f"$\\alpha={step_size}$, $x={start}$")
43:         sns.lineplot(
44:             x=iterations,
45:             y=y_of_x,
46:             ax=ax[1],
47:             linewidth=LINEWIDTH,
48:             label=f"$\\alpha={step_size}$, $x={start}$")
49:
50: ax[0].set_ylabel("estimate of  $\arg\min_x x^4$ ")
51: ax[0].set_xlabel("iteration")
52: ax[1].set_ylabel(" $\hat{y}(x)$ ")
53: ax[1].set_xlabel("iteration")
54: ax[0].set_ylim([-10000, 10000])
55: ax[1].set_ylim([-100, 10000])
56: plt.tight_layout()
57:
58: outfile = "fig/gradient-descent-x^4-crazy.pdf"
59: if len(sys.argv) > 1:
60:     outfile = sys.argv[1]
61: print(outfile)
62: plt.savefig(outfile)
```

```
1: import sympy as sp
2: import sys
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import seaborn as sns
6: import pandas as pd
7: from lib import GradientDescent
8:
9: LINEWIDTH = 0.5
10: x = sp.symbols('x')
11: y = x**4
12: dydx = y.diff()
13:
14: fig, ax = plt.subplots(1, 3, figsize=(12, 4))
15:
16: blowup = 0.8
17:
18: # alpha * (blowup ** 3) * 4 = 1.2
19:
20: results = {
21:     "alpha": [],
22:     "start": [],
23:     "convergence time": [],
24:     "final guess": [],
25: }
26: iota = 0.000000000001
27: for (step_size, start, color) in [
28:     (0.1, 1, 'gray'),
29: ]:
30:     print(step_size, start, color)
31:     g = GradientDescent()
32:     g.max_iter(100)
33:     g.step_size(step_size)
34:     g.start(start)
35:     g.function(lambda x1: float(y.subs(x, x1)))
36:     y_diff = y.diff()
37:     g.gradient(lambda x1: float(y_diff.subs(x, x1)))
38:     g.debug(True)
39:
40:     def is_inf(x):
41:         import math
42:         if x == math.inf or x == -math.inf:
43:             return True
44:
45:     def converged(x1, x2):
46:         if is_inf(x1) or is_inf(x2):
47:             return True
48:         abs = np.abs(x1-x2)
49:         print(abs, x1, x2)
50:         return abs < 0.001
51:     g.converged(converged)
52:     iterations, estimates, y_of_x = zip(*[
53:         (x[0], x[1], x[2]) for x in g.iterate())
54:     results["alpha"].append(step_size)
55:     results["start"].append(start)
56:     results["convergence time"].append(len(iterations))
57:     results["final guess"].append(estimates[-1])
58:     print('y_of_x', y_of_x)
59:     print('iterations', iterations)
60:     print('estimates', estimates)
61:     sns.lineplot(
62:         x=iterations,
63:         y=np.abs(np.array(estimates)),
64:         ax=ax[0],
65:         linewidth=LINEWIDTH,
66:         legend=False,
67:         color=color,
68:         label=f"$\\alpha={step_size}$, $x={start}$")
69:     sns.lineplot(
70:         x=iterations,
71:         y=y_of_x,
72:         ax=ax[1],
73:         linewidth=LINEWIDTH,
74:         color=color,
75:         label=f"$\\alpha={step_size}$, $x={start}$")
76:     ax[2].step(
77:         estimates,
78:         y_of_x,
79:         linewidth=LINEWIDTH,
80:         color=color,
81:         label=f"$\\alpha={step_size}$, $x={start}$")
82:     xs = np.arange(-2, 2, 0.01)
83:     ys = [y.subs(x, xi) for xi in xs]
84:     ax[2].plot(
85:         xs,
86:         ys,
87:         linewidth=LINEWIDTH,
88:         label="$x^4$",
89:         color='yellow',
90:     )
91:     ax[2].scatter(
92:         start,
93:         g._function(start),
94:         color=color)
95:
96:
97: ax[0].set_ylabel("$\\hat{x}$")
98: ax[0].set_xlabel("iteration")
99: ax[0].set_yscale('log')
100: ax[0].set_title("(a)")
```

```
101: ax[1].set_ylabel("$y(\\hat{x})$")
102: ax[1].set_xlabel("iteration")
103: ax[1].set_title("(b)")
104: ax[2].set_xlabel("$x$")
105: ax[2].set_ylabel("$y$")
106: ax[2].set_title("(c)")
107: # ax[0].set_ylim([-7, 7])
108: # ax[1].set_ylim([-1, 4])
109: ax[2].set_ylim([-0.2, 1.2])
110: # ax[2].set_xlim([-2, 2])
111: plt.tight_layout()
112:
113: outfile = "fig/gradient-descent-bi.pdf"
114: if len(sys.argv) > 1:
115:     outfile = sys.argv[1]
116: plt.savefig(outfile)
117: df = pd.DataFrame(results)
118: print(df)
119: df.to_csv("fig/gradient-descent-bi.csv")
```

```
1: import sympy as sp
2: import sys
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import seaborn as sns
6: import pandas as pd
7: from lib import GradientDescent
8:
9: LINEWIDTH = 0.1
10: x = sp.symbols('x')
11: y = x**4
12: dydx = y.diff()
13:
14: fig, ax = plt.subplots(1, 3, figsize=(12,4))
15:
16: blowup = 0.8
17:
18: # alpha * (blowup ** 3) * 4 = 1.2
19:
20: results = {
21:     "alpha": [],
22:     "start": [],
23:     "gamma": [],
24:     "$f(x)$": [],
25:     "convergence time": [],
26:     "final guess": [],
27: }
28: iota = 0.005
29: for (gamma, start, color) in [
30:     (0.1, 1, 'blue'),
31:     (1, 1.5, 'black'),
32:     (1, 1, 'orange'),
33:     (1.2, 1, 'red'),
34:     (-0.005, 1, 'purple'),
35: ]:
36:     g = GradientDescent()
37:     g.max_iter(30)
38:     alpha = 1
39:     g.step_size(alpha)
40:     g.start(start)
41:     y = gamma * sp.Abs(x)
42:     g.function(lambda x1: float(y.subs(x, x1)))
43:     y_diff = y.diff()
44:     g.gradient(lambda x1: gamma if x1 > 0 else -gamma if x1 < 0 else 0)
45:     g.debug(True)
46:
47:     def is_inf(x):
48:         import math
49:         if x == math.inf or x == -math.inf:
50:             return True
51:
52:     def converged(x1, x2):
53:         if is_inf(x1) or is_inf(x2):
54:             return True
55:         abs = np.abs(x1-x2)
56:         print(abs, x1, x2)
57:         return abs < 0.001
58:     g.converged(converged)
59:     iterations, estimates, y_of_x = zip(*[
60:         (x[0], x[1], x[2]) for x in g.iterate())
61:     results["alpha"].append(alpha)
62:     results["gamma"].append(gamma)
63:     results["$f(x)$"].append(str(y))
64:     results["start"].append(start)
65:     results["convergence time"].append(len(iterations))
66:     results["final guess"].append(estimates[-1])
67:     sns.lineplot(
68:         x=iterations,
69:         y=estimates,
70:         ax=ax[0],
71:         linewidth=LINEWIDTH,
72:         legend=False,
73:         color=color,
74:         label=f"$\\gamma={gamma}$, $x_0={start}$")
75:     sns.lineplot(
76:         x=iterations,
77:         y=y_of_x,
78:         ax=ax[1],
79:         linewidth=LINEWIDTH,
80:         color=color,
81:         label=f"$\\gamma={gamma}$, $x_0={start}$")
82:     ax[2].step(
83:         estimates,
84:         y_of_x,
85:         linewidth=LINEWIDTH,
86:         color=color,
87:         label=f"$\\gamma={gamma}$, $x_0={start}$")
88:     xs = np.arange(-2, 2, 0.01)
89:     ys = [y.subs(x, xi) for xi in xs]
90:     ax[2].plot(
91:         xs,
92:         ys,
93:         linewidth=LINEWIDTH,
94:         label="$\\gamma x^2$",
95:         color='yellow',
96:     )
97:     ax[2].scatter(
98:         start,
99:         g._function(start),
100:         color=color)
```

```
101:
102:
103: ax[0].set_ylabel("$x$")
104: ax[0].set_xlabel("iteration")
105: ax[0].set_title("(a)")
106: ax[1].set_ylabel("$y(\hat{x})$")
107: ax[1].set_xlabel("iteration")
108: ax[1].set_title("(b)")
109: ax[2].set_xlabel("$x$")
110: ax[2].set_ylabel("$y$")
111: ax[2].set_title("(c)")
112: ax[0].set_ylim([-1.2, 2])
113: ax[1].set_ylim([-1, 4])
114: ax[2].set_ylim([-1, 2.2])
115: ax[2].set_xlim([-2, 2])
116: plt.tight_layout()
117:
118: outfile = "fig/gradient-descent-cii.pdf"
119: if len(sys.argv) > 1:
120:     outfile = sys.argv[1]
121: plt.savefig(outfile)
122: df = pd.DataFrame(results)
123: print(df)
124: df.to_csv("fig/gradient-descent-cii.csv")
```



```
1: import sympy as sp
2: import sys
3: import numpy as np
4: import matplotlib.pyplot as plt
5: import seaborn as sns
6: import pandas as pd
7: from lib import GradientDescent
8:
9: LINEWIDTH = 0.1
10: x = sp.symbols('x')
11: y = x**4
12: dydx = y.diff()
13:
14: fig, ax = plt.subplots(1, 3, figsize=(12,4))
15:
16: blowup = 0.8
17:
18: # alpha * (blowup ** 3) * 4 = 1.2
19:
20: results = {
21:     "alpha": [],
22:     "start": [],
23:     "gamma": [],
24:     "$f(x)$": [],
25:     "convergence time": [],
26:     "final guess": [],
27: }
28: iota = 0.005
29: def run(gamma, color, max_iter=99, plot=True):
30:     g = GradientDescent()
31:     g.max_iter(max_iter)
32:     alpha = 1
33:     start = 1
34:     g.step_size(alpha)
35:     g.start(start)
36:     y = gamma * (x**2)
37:     g.function(lambda x1: float(y.subs(x, x1)))
38:     y_diff = y.diff()
39:     g.gradient(lambda x1: float(y_diff.subs(x, x1)))
40:     g.debug(True)
41:
42:     def is_inf(x):
43:         import math
44:         if x == math.inf or x == -math.inf:
45:             return True
46:
47:     def converged(x1, x2):
48:         if is_inf(x1) or is_inf(x2):
49:             return True
50:         abs = np.abs(x1-x2)
51:         print(abs, x1, x2)
52:         return abs < 0.001
53:     g.converged(converged)
54:     iterations, estimates, y_of_x = zip(*[
55:         (x[0], x[1], x[2]) for x in g.iterate())])
56:     results["alpha"].append(alpha)
57:     results["gamma"].append(gamma)
58:     results["$f(x)$"].append(str(y))
59:     results["start"].append(start)
60:     results["convergence time"].append(len(iterations))
61:     results["final guess"].append(estimates[-1])
62:     if plot:
63:         sns.lineplot(
64:             x=iterations,
65:             y=estimates,
66:             ax=ax[0],
67:             linewidth=LINEWIDTH,
68:             legend=False,
69:             color=color,
70:             label=f"$\\gamma={gamma}$")
71:         sns.lineplot(
72:             x=iterations,
73:             y=y_of_x,
74:             ax=ax[1],
75:             linewidth=LINEWIDTH,
76:             color=color,
77:             label=f"$\\gamma={gamma}$")
78:         ax[2].step(
79:             estimates,
80:             y_of_x,
81:             linewidth=LINEWIDTH,
82:             color=color,
83:             label=f"$\\gamma={gamma}$")
84:         xs = np.arange(-2, 2, 0.01)
85:         ys = [y.subs(x, xi) for xi in xs]
86:         ax[2].plot(
87:             xs,
88:             ys,
89:             linewidth=LINEWIDTH,
90:             label="$\\gamma x^2$",
91:             color='yellow',
92:         )
93:         ax[2].scatter(
94:             start,
95:             g._function(start),
96:             color=color)
97:
98:
99: for (gamma, color) in [
100:     (0.01, 'green'),
```

```
101:         ( 0.1, 'blue'),
102:         ( 1 - iota, 'black'),
103:         (1 + iota, 'orange'),
104:         ( 1, 'red'),
105:         (-0.05, 'purple'),
106:     ]:
107:         run(gamma, color)
108:
109: run(-1000, 'pink', max_iter=10000, plot=False)
110:
111: ax[0].set_ylabel("$x$")
112: ax[0].set_xlabel("iteration")
113: ax[0].set_title("(a)")
114: ax[1].set_ylabel("$y(\hat{x})$")
115: ax[1].set_xlabel("iteration")
116: ax[1].set_title("(b)")
117: ax[2].set_xlabel("$x$")
118: ax[2].set_ylabel("$y$")
119: ax[2].set_title("(c)")
120: ax[0].set_ylim([-7, 7])
121: ax[1].set_ylim([-1, 4])
122: ax[2].set_ylim([-1, 2.2])
123: ax[2].set_xlim([-2, 2])
124: plt.tight_layout()
125:
126: outfile = "fig/gradient-descent-ci.pdf"
127: if len(sys.argv) > 1:
128:     outfile = sys.argv[1]
129: plt.savefig(outfile)
130: df = pd.DataFrame(results)
131: print(df)
132: df.to_csv("fig/gradient-descent-ci.csv")
```

```
1: #!/usr/bin/env python
2:
3: import pandas as pd
4: import sys
5: import subprocess
6: import os
7:
8:
9: def csv_to_latex_pdf(input_csv, output_pdf="output.pdf"):
10:     # Read the CSV file into a pandas DataFrame
11:     df = pd.read_csv(input_csv, dtype=str)
12:
13:     # Convert the DataFrame to LaTeX tabular format
14:     df_to_latex_pdf(df, output_pdf=output_pdf)
15:
16:
17: def format_float(x):
18:     if isinstance(x, float):
19:         import math
20:         if x == math.inf:
21:             return "$\\infty$"
22:         if x == -math.inf:
23:             return "$-\\infty$"
24:         if x == math.nan:
25:             return "NaN"
26:         return ("\\num{{{0:.2g}}}".format(x))
27:
28:
29: def df_to_latex_pdf(df, output_pdf="output.pdf"):
30:     # Create the tmp directory if it doesn't exist
31:     if not os.path.exists("tmp"):
32:         os.makedirs("tmp")
33:     latex_tabular = df.to_latex(float_format=format_float)
34:
35:     # Wrap the tabular code in a LaTeX document
36:     latex_document = r"""\documentclass{article}
37: \usepackage{booktabs}
38: \usepackage{siunitx}
39: \begin{document}
40: \thispagestyle{empty}
41: """ + latex_tabular + r"""\end{document}"""
42:
43:     output_tex = "tmp/output.tex"
44:
45:     # Save the LaTeX code to a file
46:     with open(output_tex, 'w') as f:
47:         f.write(latex_document)
48:
49:     # Compile the LaTeX file using pdflatex
50:     subprocess.run(["pdflatex", "-jobname=tmp/output", output_tex])
51:     subprocess.run(["pdfcrop", "tmp/output.pdf", output_pdf])
52:
53:     print(f"PDF generated as {output_pdf}")
54:
55:
56: if __name__ == "__main__":
57:     if len(sys.argv) != 3:
58:         print("Usage: python script_name.py input.csv output.pdf")
59:         sys.exit(1)
60:
61:     input_csv = sys.argv[1]
62:     output_pdf = sys.argv[2]
63:     csv_to_latex_pdf(input_csv, output_pdf)
```

```
1: import sympy as sp
2: # finite difference
3: def diff_with_pert(f, xval, pert=0.01):
4:     global x
5:     delta_x = pert
6:     return (f.subs(x, xval + delta_x) - f.subs(x, xval)) / (delta_x)
7: x = sp.symbols('x')
8: y = x**4
9: dydx = y.diff()
10: analytic_ys = [dydx.subs(x, i) for i in my_range()]
11: # ...
12: for pert in np.array([0.01, 0.1, 0.15]):
13:     dydx_finite = diff_with_pert(y, x, pert=pert)
14:     # ...
15:
```

```
1: class GradientDescent():
2:     # ...
3:     def iterate(self):
4:         import math
5:         x_value = self._start
6:         old_x_value = None
7:         iteration = 0
8:         while True:
9:             yield [iteration, float(x_value), float(self._function(x_value))]
10:            iteration += 1
11:            if self._max_iter > 0 and iteration > self._max_iter:
12:                break
13:            grad_value = self._gradient(x_value)
14:            x_value -= self._step_size * grad_value # Update step
15:            if old_x_value is not None and self._converged(x_value, old_x_value):
16:                yield [iteration, float(x_value), float(self._function(old_x_value))]
17:                print("converged")
18:                break
19:            old_x_value = x_value
```

```
1: class GradientDescent():
2:     def __init__(self):
3:         self._max_iter = 1000
4:         self._debug = False
5:         self._converged = lambda x1, x2: False
6:
7:     def step_size(self, a):
8:         self._step_size = a
9:         return self
10:
11:    def function(self, f):
12:        self._function = f
13:        return self
14:
15:    def gradient(self, g):
16:        self._gradient = g
17:        return self
18:
19:    def max_iter(self, m):
20:        self._max_iter = m
21:        return self
22:
23:    def start(self, s):
24:        self._start = s
25:        return self
26:
27:    def debug(self, d):
28:        self._debug = d
29:        return self
30:
31:    def converged(self, c):
32:        self._converged = c
33:        return self
34:
35:    def iterate(self):
36:        import math
37:        x_value = self._start
38:        old_x_value = None
39:        iteration = 0
40:        while True:
41:            yield [iteration, float(x_value), float(self._function(x_value))]
42:            iteration += 1
43:            if self._max_iter > 0 and iteration > self._max_iter:
44:                break
45:            grad_value = self._gradient(x_value)
46:            x_value -= self._step_size * grad_value # Update step
47:            if old_x_value is not None and self._converged(x_value, old_x_value):
48:                yield [iteration, float(x_value), float(self._function(old_x_value))]
49:                print("converged")
50:                break
51:            old_x_value = x_value
```

```
1: import sympy as sp
2:
3: x = sp.symbols('x')
4: print(x)
5: f = x ** 4
6: print(f)
7: print(f.diff())
8: print(f.subs(x, x**2))
9: print(f.conjugate())
10: print(f)
11: print(f.subs())
```