1 Background

1.1 Damped mass-spring systems

Many engineering pursuits center around modeling physical systems. If a system has been accurately modeled, then the difference between a model-predicted values and an experimentally measured values are low. As an example, consider a simplified damped mass-spring system (Figure 1). The position of the mass, x, can be modeled as a function of time, t, as follows:

$$x(t) = e^{\frac{-b}{2}t} \cos\left(t\sqrt{1-\frac{b^2}{4}}\right) \tag{1}$$

where b is the dampening coefficient. Eqn. 1 can be plotted for any non-negative value of b, and forms what is known as a family of curves parameterized by b (Figure 2). If we know the dampening coefficient of the system, we can easily predict the position of a block on a spring. However, what about the case where we don't know b? In other words, if we have a spring-mass system where we don't know the dampening coefficient, b, can we infer it from things we can measure (e.g. time-position measurements of the block)?

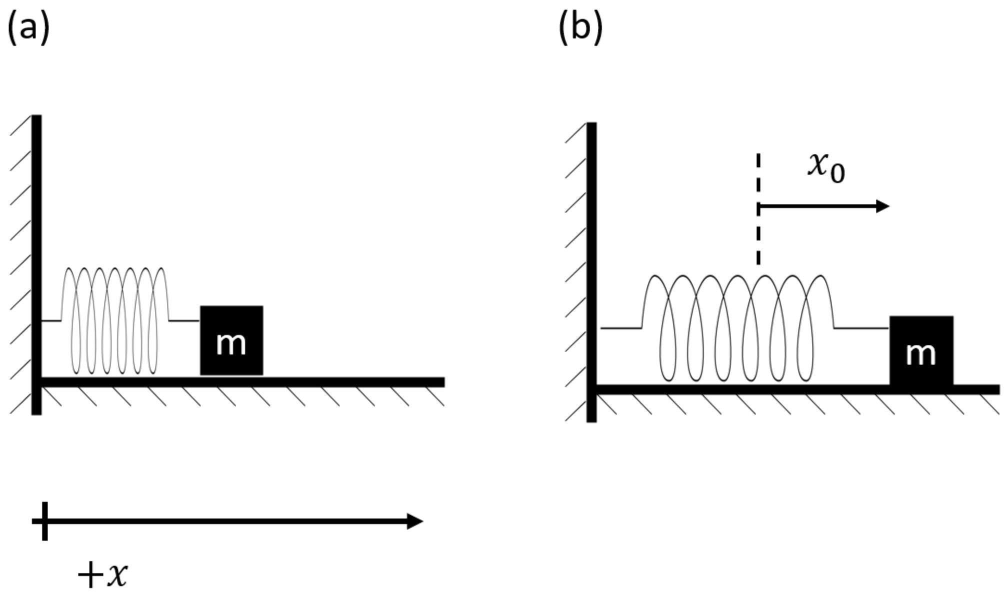

Figure 1: Diagram of a damped mass-spring system. (a) The equilibrium position of the mass. (b) The mass is pulled to some initial distance, x_0 and then released.

1.2 Error surfaces and parameter estimates

Suppose we collect 25 time-position measurements of the mass-spring system in Eqn. 1 where the b is unknown (Figure 3a). We want to find an estimate, $\hat{b}$ for the true value of b. One way to do this is to guess a value for $\hat{b}$, plug this value into Eqn. 1, check how well this guess matches our measurements (Figure 3b), and re-adjust our guess until we are satisfied the model matches the data.

This guess-and-check algorithm is the basis of both non-linear curve fitting and ML model training, but to understand the machinery it is important to formalize the process. First, we need

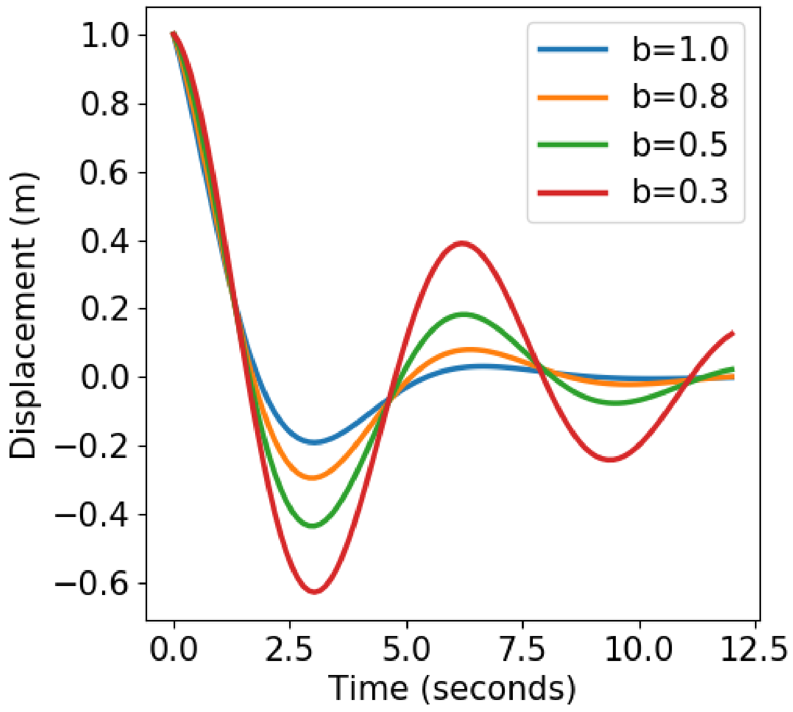

Figure 2: Diagram of a damped mass-spring system. (a) The equilibrium position of the mass. (b) The mass is pulled to some initial distance, x_0 and then released.

to define what it means for a guess to "match measurements well". This is done by defining an error function (also called an objective function). In this case, it is useful to define error to be the sum of squared residuals:

$$Error = f = \sum_{i=1}^{25} (x(t_i)^{model} - x_i^{measured})^2 \tag{2}$$

where $x(t_i)^{model}$ is the position of the block predicted by the model at time t_i, and $x_i^{measured}$ is the measured position of the block at the same time. Note that if we input an estimate for the dampening coefficient, $\hat{b}$, into Eqn. 1 which is close to the true value of of dampening in our system (i.e. $|\hat{b} - b| \approx 0$) then the error will be low (Figure 4). Similarly if we input estimates which are very different from the true value of dampening in our system (i.e. $|\hat{b} - b| >> 0$), the error will be high (Figure 4). With this in mind, it is clear that the value of $\hat{b}$ which minimizes Eq. 2, will be the value of $\hat{b}$ which best describes our system, and we should seek to minimize Eq. 2.

1.3 Gradient descent

The next challenge to address is related to how we minimize Eq. 2. Guess-and-check is slow, especially when many parameters are unknown. In the case of an ML model, we might be trying to estimate trillions of parameters; doing a brute-force search to optimize the set of parameters yielding a minimum error is computationally intractable. Luckily, gradient descent offers an intelligent way to perform parameter optimization.

Recalling that the error described in Eq. 2 is a function of our guess for $\hat{b}$, we will write the

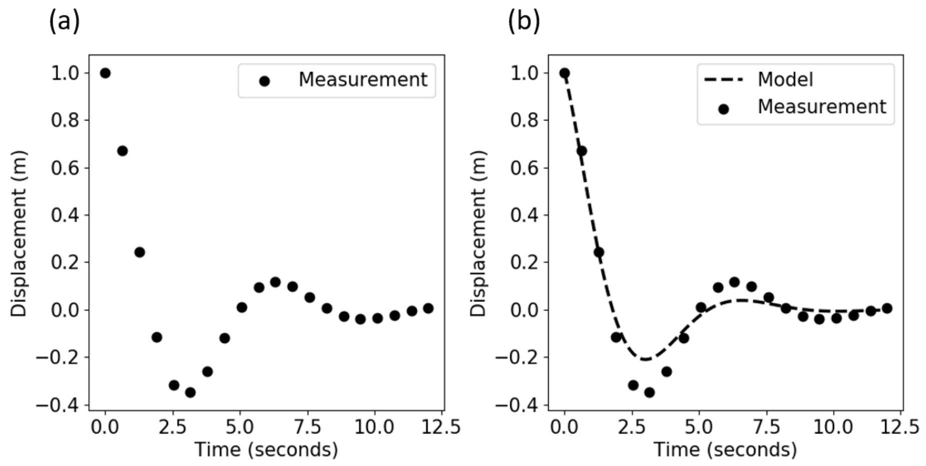

Figure 3: Time-position measurements of a mass-spring system with an unknown dampening coefficient, b. (a) 25 time-position measurements of a mass-spring system were recorded. (b) A guess for the dampening coefficient, $\hat{b}$ was made and Eqn. 1 was plotted using this value. In this case, the initial guess for $\hat{b}$ is too large, and the model predicts larger degrees of dampening than measured.

error as $f(\hat{b})$. Next, let $\hat{b}_0$ be our initial guess for the dampening coefficient. Finally, we will define the learning rate to be α. The gradient descent algorithm is as follows:

1. Calculate the gradient of the error function at the guess of $\hat{b}_0$: $\frac{df}{d\hat{b}}\big|_{\hat{b}_0}$
2. Calculate the updated guess, $\hat{b}_1$ as $\hat{b}_1 = \hat{b}_0 - \alpha \frac{df}{d\hat{b}}\big|_{\hat{b}_0}$
3. Repeat steps 1-2 until f does not change between parameter updates

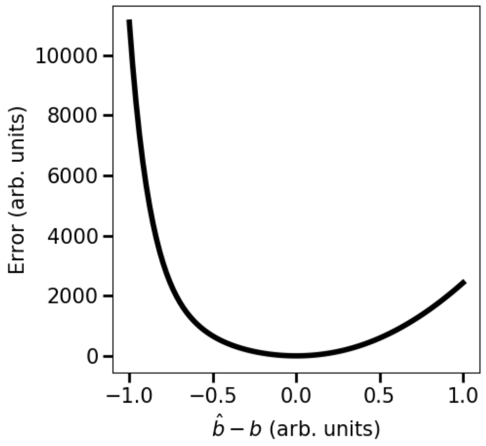

Figure 4: The error as a function of difference between the input dampening coefficient, $\hat{b}$ and true dampening coefficient b. As the input dampening coefficient approaches the true dampening coefficient, the error between observed values and model predicted values approaches 0.

2 Lab instructions

In this lab, we will be constructing our own gradient descent optimizer to minimize a black-box function function: $f(x)$

2.1 Install Python

Do it (steps will go here eventually)

2.2 Download lab files

Download function files from my Github into your working directory.

2.3 Guess-and-check optimization

You will begin by writing a program that uses guess and check to find the minimum value of $f(x)$.

1. Generate a set of guesses for x in the range $-6 < x_0 < 6$. You can do this anyway you want; x values can be random (you might find the np.random.uniform method useful), they can be equally spaced (you might find the np.linspace method useful), or any other way you can think up.
2. Calculate $f(x)$ for each guess in your set of guesses.
3. Find the minimum value of $f(x)$, and corresponding x.

2.4 Questions

1. What is the minimum value of $f(x)$? What value, x_{min} minimizes f?
2. How many guesses of x did you need to generate, for $f(x_{min})$ to be within 0.01% of the next lowest value?
3. Can we be sure we have found a global minimum? Why?

2.5 Gradient descent optimizer

Next, you will write a program to find the minimum value of $f(x)$ that uses the gradient descent method. Follow the steps in Section 1.3. Your initial guess of x_0 should be in the range $-6 < x_0 < 6$. You might find the np.random.uniform method useful.

2.6 Questions

1. What is the minimum value of $f(x)$? What value, x_{min} minimizes f?
2. How many guesses of x did you need to generate, for $f(x_{min})$ to be within 0.01% of the next lowest value?
3. Does your initial guess of x_0 change the minimum value of f? If so, why?
4. How does your learning rate, α, the number of guesses?
5. Does the learning rate, α, affect the calculated minimum? If so how? If not, why not?
6. Can we be sure we have found a global minimum? Why? What implications does this have for ML models?

Challenge In this lab we have implemented naive gradient descent for 1 dimension. Locate the algorithm for gradient descent with momentum and implement this.