Diffusion Notes

Nicholas Leland

October 2024

Contents

Chapter 1

Step by Step Diffusion: An Elementary Tutorial

The first source that we will be taking a look at is **Step by Step Diffusion: An Elementary Tutorial**. This is meant to pose as an introductory course to the concept of Diffusion, and should help us gain a general understanding of the topic.

1.1 Fundamentals of Diffusion

1.1.1 Gaussian Diffusion

Gaussian diffusion is the act of taking images that have had noise applied to them, *denoising* the image in order to gain a deeper understanding of the framework that the image is built on. We will begin by exploring how we inject our images with noise.

Definition 1. *the* Forward Process *can be described as follows:*

$$x_{t+1} := x_1 + n_t, n_t \ N(0, \sigma^2)$$

Essentially, we are creating a target distribution $p*$ that is generated using a data set we have identified. Then, using this formula, we generate multiple random variables $(x_0, x_1, x_2, \ldots, x_t$. With these random variables (in our case, images), we can identify that we slowly begin to approach the **Gaussian Distribution**, therefore, we can sample the Gaussian rather then our originally defined $p*$ distribution.

We can evaluate the two distributions using a formula known as KL Divergence.

Definition 2. *The* ***Kullback-Leibler (KL) divergence*** *between two distributions P and Q is defined as:*

$$D_{KL}(P||Q) = \sum_x P(x) log \frac{P(x)}{Q(x)}.$$

or for a continuous distribution:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(X) log \frac{P(x)}{Q(x)}.$$

- *$P(x)$ is the true probability distribution*
- *$\mathscr{Q}(x)$ is the approximating (or target) probability distribution*
- *$D_{KL}(P||Q)$ is the KL Divergence which is non-negative and measures how much information is lost when Q is used to approximate P.*

So would we be utilizing Continuous or Discrete variables? You might think initially that due to the format of image files given as:

$$P_{ij} \subset [0, 255]^3 for i = 1, \ldots, W and j = 1, \ldots, H.$$

Realistically though, our model will think within the space of 0-1 instances which we will then convert into our discrete space. We will find later on that this quantization is actually quite a problematic idea with images.

It might be important to have these definition's on hand as well, mainly just as a refresher.

Definition 3. *Probability Density Function*

$$P(x) = \frac{1}{(2\pi)^{\frac{d}{2}}|\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T \sum^{-1}(x-\mu)).$$

where:

- *x is a d-dimensional random vector.*
- *μ is the mean vector*
- *$\sum$ is the covariance matrix.*

Definition 4. *Covariance Matrix*

$$Cov(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

where:

- *$\mathbb{E}[X]$. is the expected value (mean) of X.*
- *$\mathbb{E}[Y]$. is the expected value (mean) of Y.*

Now that we have initial definitions out of the way, let's reevaluate our problem. We have an image, we want to define some way to reconstruct an image of that nature. Our sub problem is essentially:

"Given a sample marginally distributed as p_t, introduce a sample marginally distributed as p_{t-1} "

Now that we have established our noisy image, the goal is to create a model which can identify features and effectively denoise our image. This method is known as a **reverse sampler**.

We would effectively move through each position until indefinitely reaching our original, target distribution ($p_0 = p^*$). This leads us to the primary idea behind diffusion, rather then learning how to go from a noised image back in one step (more in line to how a VAE functions), we instead go through multiple steps and learn the process behind each.

We can construct reverse samplers in many different ways, the **DDPM Sampler** is known as the standard diffusion sampler. This uses a very simple strategy, at a time t, given the input z, where z is a sample from p_t, output a sample from the conditional distribution.

$$p(x_{t-1}|x_t = z).$$

This seems very simple but realistically there is much more to the entire process. If we were to have a model for every single step of x_t, this would be quite a large process. The key is identifying the *per step noise* or σ.

If this amount of noise that we are adding per step is small enough, the distribution becomes simple.

Definition 5. ***Diffusion Reverse Process****: For small σ, the Gaussian diffusion process defined in (1), the conditional distribution $p(x_{t-1}|x_t$ is itself close to Gaussian. That is, for all the times t and conditioning $z \subset \mathbb{R}^d$, there exists some mean parameter $\mu \subset \mathbb{R}^d$ such that:*

$$p(x_{t-1}|x_t = z) \approx \mathcal{N}(x_{t-1} : \mu, \sigma^2.$$

There is a lot going on here, let's talk about it in a bit more detail. Essentially speaking, if we have the distribution that we are trying to understand, by targeting a certain point and minimizing our σ value, the distribution approaches that of the Gaussian at a much more aggressive rate. This allows us to have one missing value to focus on, the mean (μ).

This is a really important statement, so let's go through this one more time.

- For a given time t and a conditioning value x_t, learning the mean of $p(x_{t-1}|x_t$ is sufficient to learn the full conditional distribution $p(x_{t-1}|x_t$.
- Now that we know we must just determine the mean (μ), $wecanjustutilizeasimpletoollike$**simple linear regression**$!tobegi$
- Our goal is to estimate $\mathbb{E}[x_{t-1}, x_t]$

Definition 6. ***Standard Regression Loss*** : *Remember, for any distribution over* (x, y)*, we have:*

$$\arg\min_f \mathbb{E}\left[||f(x) - y||^2\right] = \mathbb{E}[y|x]$$
$$\mu_{t-1}(z) := \mathbb{E}[x_{t-1}|x_t = z]$$
$$\mu_{t-1} = \arg\min_{f:\mathbb{R}^d \to \mathbb{R}^d} \mathbb{E}_{x_t, x_{t-1}} ||f(x_t) - x_{t-1}||_2^2$$
$$\arg\min_{f:\mathbb{R}^d \to \mathbb{R}^{dxd`}} \mathbb{E}_{x_t, x_{t-1}, t} ||f(x_{t-1} + _t) - x_{t-1}||_2^2$$

Essentially, we are optimizing the μ for the minidifference between our given distribution and the noise that is generated on the image. x_0 is from our target distribution p^*. When target p^* is a distribution on images, then the corresponding regression problem is exactly an **image denoising objective**. This objective can then be approached with deep learning techniques such as traditional CNN's.

Through this process we have taken our goal of **learning to sample from an arbitrary distribution** to the standard **problem of regression**.

1.1.2 Diffusions in the Abstract

Forget about the Gaussian now. Let's break down our *diffusion-like generative model.*

- Start with a target distribution (p^*)
- Pick a base distribution ($q(x)$) that is easy to sample from (Standard Gaussian or i.i.d bits)
- Construct a sequence of distributions which interpolate between the target p^* and the base distribution q. We now have a set of distributions $p_0, p_1, p_2, \ldots, p_t$.
- With these distributions, $p_0 = p^*$ (Our Target), $p_t = q$ (base distribution) and the rest of the distributions, $p_t - 1, pt$, are marginally close to one another. The smaller steps the more accurately we can assume the base distribution.
- Then we learn a *reverse sampler* to take our p_t distributions and transform them into p_{t-1}. This is how we train the model, or our *learning step.*

Definition 7. ***Reverse Sampler*** *Given a sequence of marginal distributions* p_t*, a reverse sampler for step* t *is a potentially stochastic function* F_t *such that if* x_t p_t*, then the marginal distribution of* $F_t(x_t)$ *is exactly* p_{t-1}*:*

$$F_t(z) : z \ p_t \equiv p_{t-1}$$

Remember, stochastic functions are just functions that have an unpredictable outcome, which is entirely valid based on the assumption of how noise is added.

There are many possible reverse samplers that are available, some are even *deterministic*, or include no randomness. In some cases, it is possible to instantiate in a discrete setting where the distribution p^* is over a finite set and there are corresponding "interpolating distributions" and reverse samplers.

We will be evaluating three very popular reverse samplers here:

- The DDPM sampler
- The DDIM Sampler (More deterministic
- The Family of *flow-matching models*, a generalization of DDIM

1.1.3 Discretization

Very quickly, we run into the problem regarding dealing with discrete values (float vs int). The process of taking a function that is normally a continuous function and transferring that into one that is discrete is known as **discretization**. The goal is to reduce the amount of error down to a set *negligible amount* by the end of the process.

Currently we have explained the process of de-noising an image as finding a step value (p_{t-1} where the value is *extremely small*, therefore we can draw the same assumptions about the baseline distribution that we are using. We run into a problem when evaluating how small this step (p_t, p_{t-1} actually ends up being.

As a refresher, we have a time-evolving function, $p(x,t)$, which starts from the target distribution (p_0 at time $t = 0$) and ends at the noisy distribution (p_t at time $t = 1$). It is important to note, this is not a traditional function but rather the current status of our distribution at the set points (x and t). To reiterate, $p(t_0, x)$ is our original distribution (our clean image) and $p(t_1, x)$ is our distribution where we represent our target distribution.

$$p(x, k\Delta t) = p_k(x), where \Delta t = \frac{1}{T}.$$

Here, T is the *fineness* of the steps that we are taking. The more steps (and presumably the smaller the σ value), the closer the image is from step to step.

We run into a very interesting problem now. We need to ensure that the variance of the final distribution p_t is *independent of the number discretization steps.* To ensure that this is a reality, we need to be more specific about our incremental variances.

Let's look at the following given fact. If $x_k = x_{k-1} + \mathcal{N}(0, \sigma^2)$ then $x_t \sim \mathcal{N}(x_0, T\sigma^2)$

Essentially, because we are pulling from our target distribution so many times, normally, this would cause an overlap to occur, which in turn would aggressively scale our variance. This would result in our variance scaling at a factor of $T \cdot \sigma^2$. This would be quite problematic as the stacking variance would skew our distribution. To accommodate for this, we will instead by scaling the variance by the number of steps.

$$Total\ Variance = T \cdot (\sigma^2 \cdot \Delta t) = T \cdot (\sigma^2 \cdot \frac{1}{T}) = \sigma_2^2.$$

Remember, $\Delta t = \frac{1}{T}$ is the amount that we will be scaling by. With this fact, we will be choosing.

$$\sigma = sigma_q \sqrt{\Delta t}.$$

where σ_q^2 is the *desired terminal variance.* This concept of $\sqrt{\Delta t}$ scaling will be very important, and it's the baseline that SDE (Stochastic Differential Equation) formulation will build on later. By choosing σ_q^2 with the desired terminal variance, we ensure that the variance of P_t is always σ_q^2 regardless of our T.

At this point, we will be taking a temporary pause with this section and instead be jumping to the next chapter of these notes. This will build up the fundamentals regarding Stochastic Differential Equations, which will be very relative for the upcoming diffusion notes.

1.2 Stochastic Sampling: DDPM

DDPM : Denoising Diffusion Probabilistic Models

Here, we will be discussing the DDPM-like reverse sampler which are conceptually the same as that popularized byu *Denoising Diffusion Probabilistic Models* (DDPM) by Ho et al. [2020].

Just to reiterate, previously we discussed using Δt as a scaling factor within discretization of our variance to prevent exponential scaling. A footnote here discusses that we are still using a "Variance Exploding" diffusion forward process. The variance is still growing, but instead, it is growing at a *controllable rate* that we are able to evaluate as it grows towards our target distribution. Now back to the continued topics..

Let's consider the setup that we had reviewed previously with a target distribution p^* and a joint distribution of noisy samples $(x_0, x_{\Delta t}, \ldots, x_1)$. A DDPM will require estimates to be pulled from the following expectations:

$$\mu_t(z) := \mathbb{E}[x_t | x_{t+\Delta t} = z].$$

We are defining a set of functions, μ_t, for each of our noise generation steps. When we are within the **training** phase, our goal is to estimate samples of x_0 by optimizing the *denoising regression objective*

$$\mu_t = arg\,max_{f:\mathbb{R}^d \to \mathbb{R}^d} \quad \mathbb{E}_{x_t, x_{t+\Delta t}} \mid (x_{t+\Delta t}) - x_t \mid\mid_2^2 .$$

This training process typically utilizes neural-network optimization parameterization f. The next step is where inferencing occurs, where we take the output of our estimated functions and apply them to a new algorithm.

Algorithm 1. ***Stochastic Reverse Sampler (DDPM-like)***
For input sample x_t, and timestep t, output:

$$x_{t-\Delta t} \leftarrow \mu_{t-\Delta t}(x_t) + \mathcal{N}(0, \sigma_q^2 \Delta t).$$

Remember our original goal, we are establishing a regression problem where we are chasing after the μ of our target distribution. So what does the process of utilizing this algorithm look like?

First, we need to generate a sample. We sample x_1 as an isotropic Gaussian $x_1 \approx \mathcal{N}(0, \sigma_q^2)$, then we run the iteration of Algorithm 1 all the way down to $t = 0$ to produce a generated image. We will call this image, $\hat{x_0}$. This should, in theory, be a re-established image from our target distribution.

In order to really **prove** out this entire process though, we need to establish the **proof** of one specific step. The missing piece here is that the **true conditional** $p(x_{t-\Delta t}|x_t)$ **can be well-approximated by a Gaussian**, and that this approximation increases as we decrease the amount of change from steps closer and closer to zero.

Intuitively, my first guess for implementation is to establish away to inject noise into an image at a given step rate, then utilizing that, measure how close a distribution is to the Gaussian based on an evaluation of probability metrics, something like a KL-Divergence.

Definition 8. ***Kullback-Leibler (KL) divergence*** *also called* ***relative entropy*** *and* ***I-divergence*** *is a type of statistical distance: a measure of how one reference probability distribution P is different from a second probability distribution Q.*

$$D_{KL}(P \mid\mid Q) \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)}).$$

A cool way to remember this is to think about the expected surprise when using a distribution Q vs a distribution P.

1.2.1 Correctness of DDPM

Assume that we have a smooth density over $\mathbb{R}^d$, we will call this $p_{t-\Delta t}(x)$. We want to evaluate a joint distribution (map of two distributions with each as a perpendicular axis. We will define this joint distribution as $(x_{t-\Delta t}, x_t,$ where $x_{t-\Delta t} \sim p_{t-\Delta t}$ and $x_t \sim x_{t-\Delta t} + \mathcal{N}(0, \sigma_q^2 \Delta t)$.
For a sufficiently small Δt, the following holds. For every conditioning of $z \in \mathbb{R}^d$, there exists a μ_z such that:

$$p(x_{t-\Delta t}|x_t = z) \approx \mathcal{N}(x_{t-\Delta t}; \mu_z, \sigma_q^2 \Delta t).$$

for some constant μ_z depending only on z. Moreover, it suffices to take

$$\mu_z := \exists_{(}x_{t-\Delta t, x_t})[x_{t-\Delta t}|x_t = z].$$
$$= z + (\sigma_q^2 \Delta t) \nabla \log p_t(z), .$$

where p_t is the marginal distribution of x_t

So we are establishing z as a given point and then determining the average(μ at that point based on the predicted value. This predicted value, $\mathbb{E}$ is something that would be optimized through regression.
$\nabla \log p_t(z)$ is known as our *score function*, where we are basically searching for the highest point within the probability distribution. Essentially, we are trying to move in the direction where we are more likely to have values, which is very important because when the steps $(t - \Delta t)$ are small, it is easy to find the optimal position of where our μ is located.
It is important to note that this value μ_z is **not** actually something that we are trying to calculate! It is just important that we know a value like this does, in fact, exist. We need to know that it exists so that we **can** learn it from samples.
Overall this is good, I was struggling with thinking about this for a while because computationally this would be a very intensive process to try and work through for each individual pixel and for each individual step.
Let's work on an informal proof of this concept.
We will show an argument as to why the score shows within the reverse process. Let's start by reviewing this with the aspect of Bayes Rule.

$$p(x_{t-\Delta t}|x_t) = p(x_t|x_{t-\Delta t})p_{t-\Delta t}(x_{t-\Delta t})/p_t(x_t).$$

After working through some math, we end up with the following expression.

$$\log p(x_{t-\Delta t}|x_t) = -\frac{1}{2\sigma_q^2 \Delta t} \mid\mid x_{t-\Delta t} - \mu \mid\mid_2^2 .$$

This is identical to the *log-density* of a Normal distribution with μ and the variable σ_q^2. This therefore gives us a proof to show;

$$p(x_{t-\Delta t} \mid x_t) \approx \mathscr{N}(x_{t-\Delta t}; \mu, \sigma_q^2 \Delta t).$$

The primary takeaway here is to understand that both the forward process and the backwards process both utilize $p(x_t \mid x_{t-\Delta t})$ which is intuitively why they both have the same **functional form**.

1.2.2 Algorithms

Pseudocode 1. ***DDPM train loss***

```
*Input:* Neural network $f_\theta$ ; Sample-access to target distribution $p$.
*Data:* Terminal variance $\sigma_q$ ; step-size $\Delta t$
*Output:* Stochastic loss $L_\theta$
$x_0 \leftarrow$ Sample(p)
$t \leftarrow$ Unif $[0,1]$
$x_t \leftarrow x_0 + \mathscr{N}(0, \sigma_q^2 t)$
$x_{t+\Delta t} \leftarrow x_t + \mathscr{N}(0, \sigma_q^2 \Delta t)$
$L \leftarrow \mid\mid f_\theta(x_{t+\Delta t}, t + \Delta t) - x_t \mid\mid_2^2$
*return* $L_\theta$
```

Pseudocode 2. ***DDPM Sampling (Algorithm 1)***

```
*Input:* Trained model $f_\theta$
*Data:* Terminal Variance $\sigma_q$ ; step-size $\Delta t$
*Output:* $x_0$
$x_1 \leftarrow \mathscr{N}(0, \sigma_q^2)$
*for* $t = 1, (1 - \Delta t), (1 - 2\Delta t), \ldots, \Delta t$ *do*
| $\eta \leftarrow \mathscr{N}(0, \sigma_q^2 \Delta t)$
| $x_{t-\Delta t} \leftarrow f_\theta(x_t, t) + \eta$
*endreturn*$x_0$
```

Pseudocode 3. ***DDPM sampling (Algorithm 2)***

```
*Input:* Trained model $f_\theta$
*Data:* Terminal Variance $\sigma_q$; step-size $\Delta t$
*Output:* $x_0$
$x_1 \leftarrow \mathscr{N}(0, \sigma_q^2)$
```

```
for $t = 1, (1-\Delta t), (2-\Delta t), \ldots, \Delta t, 0$ do
| $\lambda \leftarrow \frac{\sqrt{t}}{\sqrt{t-\Delta t + \sqrt{t}}}$
| $x_{t-\Delta t} \leftarrow x_t + \lambda(f_\theta(x_t, t) - x_t)$
end
return $x_0$
```

With these sets of pseudocode established, we can begin to discuss and think about the systematic integration of these. We will be primarily focusing on the **Jax** integration as it is my preffered machine learning framework due to efficiency.

1.2.3 Variance Reduction: Predicting x_0

We have been training our diffusion models with the goal of predicting $\mathbb{E}[x_{t-\Delta t}|x_t]$.
We can argue that technically, there is a method we can follow in or der to instead, estimate $\mathbb{E}[x_0 \mid x_t]$ and then divide by $(\frac{t}{\Delta t}$, or the number of steps that have been taken so far.
We can define this through an equation as shown:

$$\mathbb{E}[(x_{t-\Delta t} - x_t) \mid x_t] = \frac{\Delta t}{t}\mathbb{E}[(x_0 - x_t) \mid x_t].$$

or equivalently:

$$\mathbb{E}[x_{t-\Delta t} \mid x_t] = (\frac{\Delta t}{t})\mathbb{E}[x_0 \mid x_t] + (1 - \frac{\Delta t}{t})x_t.$$

Let's break this down into simpler terms. Our goal is to estimate $\mathbb{E}[x_{t-\Delta t} \mid x_t]$. This is the previous noise step given the current noise step. We are going to do this by taking the number of steps so far, and multiplying directly by the estimated sample given by the current time step $((\frac{\Delta t}{t})\mathbb{E}[x_0 \mid x_t])$. After this we add an additional remaining noise step? I think? I believe this has something to do with $x_t = x_0 + \sum_i \eta_i$.
When we predict $x_{t-\Delta t}$, it is just like predicting the last noise step. We can define the last noise step as $\eta_{t-\Delta t} = (x_t - x_{t-\Delta t})$. Remember, our equation to add noise to a component is:

$$x_{t+\Delta t} := x_t + \eta_t, \eta_t \approx \mathcal{N}(0, \sigma_q^2 \Delta t).$$

If we are only given the final x_t, **all of the previous noise steps intuitively "look the same"** $(\{\eta_i\}_{i<t})$. This is relatively simple to think about, there is no way to determine if noise was added during step 3, or if it was added during step 20.
The paper then concludes that we can define **individual noise steps are distributed identically (not independently) given x_t** . Rather then estimate a single noise step, we instead estimate the average of all prior noise steps. There are $(\frac{t}{\Delta t})$ elapsed noise steps at time t, therefore we divide noise by the quantity (This is where the additional previous component is coming from).

Word of Warning: Remember, within diffusion, our goal is to estimate *expectations* ($\mathbb{E}$). We are **NOT** trying to sample directly from a distribution $p(x_0 \mid x_t$, but instead estimate what an expected sample of x_0 could be. This is a rather interesting thing to think about. It should be revisited, I feel as though this is very important.

1.2.4 Diffusion as SDEs [Optional]

This is kinda scary, marked as advanced material. Both the paper and my *internal thoughts* believe that this would be better to revisit later on (probably once I actually gain a grasp on SDE's as a subject LOL).

1.2.5 Implimentation

Well, this would begin the raw implimentation of our traditional DDPM.
Before we begin, we will be jumping to section 1.5, **Diffusion in Practice**, as it will cover some important things that might not have been covered directly in the theory section of this tutorial.

1.3 Deterministic Sampling: DDIM

1.4 Flow Matching

1.5 Diffusion in Practice

We are going to skip ahead here and begin to identify different things that might be important prior to implimentation.

Chapter 2

Stochastic Differential Equations

2.1 Differential Equations

This section will be primarily reviewing general differential equations and their fundamentals. We will be going over the baselines with some basic videos, and then diving into more traditional mathematical approaches after that.

2.1.1 Differential Equations, a tourist's guide

The first resource that we will be evaluating is the 3Blue1Brown video series regarding differential equations. This is avalible here: Differential Equations, a Tourist's Guide.

Differential Equations are essentially a function in which it is easier to document the change in numbers rather then the direct values.

Why they grow or shrink over why it is one value over another.

Differential equations are one of two: Ordinary Differential Equations (one input) or Partial Differential Equations (multiple inputs)

- Ordinary Differential Equations or **ODE**'s, are typically involving time. This is very commonly associated with physics calculations.
- Partial Differential Equations or **PDE**'s, deal with multiple inputs like a temperature at points in a solid body or the velocity of a fluid at different points in space.

Say we have an item falling. Based on earths gravity, our object is falling at a rate of $-9.8\frac{\frac{m}{s}}{s}$. This itself can be expressed as a differential equation, where it's derivative, gives the velocity, and the second derivative as the acceleration. We turn this into a differential equation problem when we begin to ask the reverse question. If we look at a generalized problem, we find that as we work through this problem backwards, we begin to introduce additional constants that are free to change. If we want to find the function which has an acceleration of 9.8, we are looking for a function with $-g$ as its derivative. This ends up introducing a t constant, which we know as time. If we do the same with our new function, looking for the original function given our acceleration and time constant, we now introduce an additional constant, one that changes the starting position. There can be many functions with each derivative, which is why we now need to clarify given the new constants. Again, if my acceleration is a second or d er derivative, -9.8. When I am trying to find the inverse of the derivative, there are many functions that might end up with this derivative, we need to isolate with the additional time position.

What if the forces depend on the location of the body? At this rate, we have an interchangeable exchange between a potential location and the resulting differential variable.

Often in differential equations, the puzzles you face involve finding a function whose derivative or higher order derivative which is defined in terms of the function itself.

In physics, it is often that you only deal with Second Or d er Differential Equations. $\ddot{x} = -\mu\dot{x}(t) - wx(t)$ where $\ddot{x}$ is our curvature, $\dot{x}$ is our slope and $x(t)$ is the height. We can continue to push our ODE's to higher degrees.

In a way, we are chasing our tail over and over again, our goal is to identify the rate of change of a rate of change. The common example to review here, is the pendulum. Let's setup this problem. X is our arc, and θ is our angle. We will clarify $x = L\theta$ and assume the typical gravity. We are essentially looking for not the downward force but the tangential force to the x arc. The tangential force would equate to $-g \cdot sin(\theta)$ due to the fact that the θ position from our pendulum is simultaneous with the θ of the tangential force.

We will consider the θ positive on the right, and negative on the left. We can then define the acceleration as follows:

$$a = \ddot{x} = -g \cdot sign(\theta).$$

Now at this point, we would try to identify the second order differential equation for θ.

$$\ddot{\theta}(t) = -\mu\dot{\theta}(t) - \frac{g}{L} \cdot sin(\theta(t)).$$

You may notice the additional term $-\mu\dot{\theta}(t)$ in our equation. This is a general blanket term that we will use to describe air resistance, or the slow from acceleration for our pendulum. Typically you may see some assumption to sin waves with early physics talks and the pendulum problem, it is very important to forget this assumption and instead to remember that it is different. When viewing the assumption, θ is the function, while our more ac curate model is using θ as the input to our function.

If we were to remove the term that we included for air resistance, we could reduce this term down and actually solve this problem, but it turns into a very complex problem to solve! Let's evaluate, what we gain from solving these types of problems.

Ideally, we take a differential equation, determine a solution, and then use that solution for one of two things, **Understanding** or **Computation**.

For any function, of ten times we could define a new function that satisfies the ODE, however, this is pointless unless you can then perform **computations** or **understanding** from the solution. Because of this, often we skip the solution, and instead jump right to attempting to create an understanding or performing computations from the equation alone.

What does this look like with our pendulum problem? We could define the individual positions on a 2-Dimensional diagram that gives all of the possible positions. We define a **state space** which is an abstraction of our state. This is a relatively common term within computer science, used to represent the set of all possible configurations of a system. θ is our x axis while $\dot{\theta}$ is our y axis. If we were to graph these, then use one of the positions as our starting value, you would notice that we essentially graph a spiral, slowly reducing to the origin of our $\theta, \dot{\theta}$ graph.

Using this, we can visualize the state space as a vector field. We can create vectors defined as follows:

$$\frac{d}{dt}\begin{bmatrix}\theta(t)\\ \dot{\theta}(t)\end{bmatrix} = \begin{bmatrix}\dot{\theta}(t)\\ \ddot{\theta}(t)\end{bmatrix}.$$

We would interpret this as being the resulting vector attached to a point in space. If we latch on to a position or begin at a position within our state space, the vector would define how the point would continue to move. The first item here tells us our acceleration moving from right to left. The second component of our vector can actually be re-represented in terms of the first or d er derivative of $\dot{\theta}(t)$.

$$\frac{d}{dt}\begin{bmatrix}\theta(t)\\ \dot{\theta}(t)\end{bmatrix} = \begin{bmatrix}\dot{\theta}(t)\\ \ddot{\theta}(t)\end{bmatrix} = \begin{bmatrix}\dot{\theta}(t)\\ -\mu\dot{\theta}(t) - \frac{g}{L}sin(\theta(t))\end{bmatrix}.$$

Realize, we have essentially broken up a difficult **Second Order ODE** to its first order component over a system. It might even be worth moving from repeating θ and $\dot{\theta}$ to another variable all together, in order to really drive home the fact that we are dealing with another theoretical component all together.

This is an important trick, move from higher form differential equations to a system of the lower form differential equation.

There is another important fact to observe here as well. With higher degrees of freedom, we increase potential dimensional space that our theoretical point would flow through. We can call a vector space like this a **phase space**. Within physics this is known as a space that represents position and time. Phase Space is strong because it helps to visualize the spectrum of initial conditions. With these vector charts, we can then represent something we will call **phase flow**.

Another aspect that we will tend to explore are whether or not certain points are **stable** within our phase space. For instance, back to our pendulum example, A point on the x axis (our θ) could be evaluated as stable if we were curious as to whether or not the pendulum would return to that exact position. Think of this like an inherent loop.

Knowing the physical properties of a pendulum it is easy to assume that this wouldn't be the case, but how about if we were to evaluate our matrix?

$$\begin{bmatrix} \dot{\theta}(t) \\ -\mu\dot{\theta}(t) - \frac{g}{L}sin(\theta(t)) \end{bmatrix}.$$

We now lose out the intuition that we would have. Instead, we would evaluate the contractual or expansive look at flow.

Remember, a cute way to think about differential equations are to evaluate them to *love*. We can't exactly describe the current position, but we can express them as the changing emotion and then even further if we have two variables evaluating two individuals that are interested in one another.

This previous section dealt with the understanding portion, but how do we deal with **computation**? Ideally, we would define a similar vector space, but now we would add an additional term, Δt. This new term is essentially our definitive time step amount. We computationally run through a small step and then determine an approximation point. The smaller a Δt, the more accurate the resulting view.

Interestingly enough, there begins to emerge some other odd values here that deal with **Chaos theory** (easily the coolest sounding math name ever). This deals with systems that are extremely sensitive to initial conditions that disrupt underlying patterns.

2.2 18.S096 Lecture 21: Stochastic Differential Equations

This next section of notes will be directly from an MiT lecture avalible on youtube at Lecture 21: Stochastic Differential Equations. This is primarily from the background of a FinTech domain, but should also go through some aspects that will be applicable for diffusion before we jump into the next section.

Say we are given a Differential Equation.

$$dX = \mu(t, X(t))dt + \sigma(t, X(t)dB(t).$$

Our goal is to find a stochastic process $X(t)$ that satisfies a given function.