**Intro slide:**

Now that you know how to represent numbers in machine form, we will start performing computations with these numbers

**Slide 1:**

Some of the mathematical properties you are familiar with and often use when calculating things by-hand will not always be applied in floating point operations. For example, Floating point operations are not necessarily associative. The order of the computation matters. In this example, if we first sum this large number, one to the power of 100, to the number pi, then we subtract the large number, and the result will be zero. However, if we change the location of the parenthesis, and therefore the order of the computation, the result is equal to pi.

Operations are also not necessarily distributive. For example, multiplying 0.1 and 0.2 by 100 and taking the sum of these two results is not the same computation as summing 0.1 and 0.2 and multiplying that by 100.

**Slide 2:**

When performing floating point arithmetic, we fist compute the exact result, and then we round the result by computing its machine number representation.

We can see this loss of precision happening in other operations as well. Assume…

7 zeros!

**Second last example:**

Now that we know of the potential loss of significance when we subtract two numbers that are similar, how can we avoid these type of situations?

Suppose you are trying to compute the result of this function f(x) when x is a small number (let’s say, very close to zero). You can see you will have a potential subtraction of a number close to one and the number one itself.

Compute f(10^-3) assuming you are using a machine with 5 decimal accurate digits.

**Last example:**

You are given two real numbers x and y. Compute the relative error in the computation of (x-y) in a machine with five decimal digits of accuracy?

**TAYLOR**

In numerical computation, we will often deal with two types of errors.

First we can get errors due to rounding, since we cannot represent all numbers exactly with finite precision.

We can also get truncation errors, **which are made when we** truncate an infinite sum and approximate them by a finite sum.

In this video, we will describe how Taylor series can be used to approximate functions, and the truncation errors associated with that.

We will first approximate the function f(x) with a polynomial, where a sub-index I are the coefficients.

**Monte Carlo**

**Slide 1:**

What type of problems can we solve with the help of random numbers?

We can create models for the stock market and make predictions about your investments. And also use stochastic models to make predictions about odds to win a competition.

**Slide 2:**

Since computers are deterministic, how can we generate random numbers inside such a machine?

Let’s take a look at some of the random functions available in Python:

**Slide Random Variables:**

We will define a random variable X that maps the outcome of a random process to a numerical quantity.

For example, the random process can be the outcome of dropping a buttered bread on the floor. The random variable X can assume two values, one that indicates buttered side up and another one that indicates buttered side down.

Another typical example is a coin toss. The random variable X is the result of the toss, that here we assume as 0 for tail and 1 for head. Since the random variable X can assume two different values, the probability of each one is 0.5 or 50%.

The expected value of a discrete random variable is defined as the summation of the probability of each event multiplied by its value. For the coin toss, we will have the expected value equal to 1 (the value of the heads) times 0.5 (the probability) plus 0 (the value of the tails) times 0.5 (the probability), which is equal to 0.5.

**Slide Monte Carlo Methods:**

Monte Carlo methods are algorithms that rely on repeated random sampling to approximate desired quantities.

For example, we can use this method to approximate the expected value for a coin toss experiment, which we know to be 0.5. But let’s compute our numerical approximation. We will toss the coin N times, where each toss is a sample point. If the toss is head, the function f(x) evaluates as 1 otherwise it evaluates as zero. We sum all of these function evaluations, and finally divide by the total number of tosses to get the expected value.

**Slide Monte Carlo to approximate integrals:**

One of the most important applications of Monte Carlo methods is in estimating volumes and areas that are difficult to compute analytically.

For the sake of simplicity, in this class we will use Monte Carlo Methods to compute “easier” integrals such as the one represented by this double integration to compute areas.

**Arrays**

So far we have mostly looked at computations using scalars. Problems become more interesting when we start looking at a collection of these scalars. One way of looking at an array of numbers is the use of vectors.

We will be re-introducing a lot of the concepts that you learned in the linear algebra class, at a much faster speed, without including the proofs and derivations.  
  
----

A vector Space is a set V of vectors and a field F of scalars with two operations:

If two vectors are in the vector space, then the summation of these two vectors are also in the vector space

If a vector v is in the vector space and a scalar alpha is in the field space, then the multiplication of the scalar with the vector is also in the vector space.

\_\_

The addition and multiplication must also satisfy all the different properties listed here, so you can review them later.

\_\_\_ Go to notebook

Most of the methods we will learn in this course can be represented by linear functions or operators. But what do we mean by that?

A function can be represented as a mapping that takes vectors in the X set and transforms them into vectors in the Y set. A function f is linear if

f of u plus v is equal to f of u plus f of v, and

f of alpha times u is equal to alpha times f of u for any scalar alpha.

Let’s take a look at two examples…

We can use matrices to present our linear functions. We will call them linear operators. Think of a matrix A as a linear function that takes vectors in x and transforms them into vectors y

We have y(x) = f(x) or y = A x where Ax is a matrix vector multiplication

Based on our definition of linear operators, we have:

We will take a look at some typical linear transformations that you learned in linear algebra.

In this example, the dataset x represents the collection of points that produces the image of a gingerbread man. WE can apply different linear transformations of the dataset x, yielding a resulting dataset y by computing y = A times x. Here we see the result of a rotation.

We will first start with the shear operator:

**Norms**

How can we generalize the concept of absolute value when we are dealing with vectors? The norm of a vector is a function that takes a vector in Rn and returns a scalar that provides a sense of magnitude of this vector.

To be defined as a norm, a function needs to satisfy the following properties:

1. It returns a scalar that is strictly positive, for any vector that is different than zero
2. If you take the norm of a vector x multiplied by a scalar gamma, the result is the same as the absolute value of that scalar gamma multiplied by the norm of the vector x.
3. It satisfies the triangle inequality.

One example of a norm is called the p-norm, which we will mostly use in this course.

The p-norm takes the absolute value of each entry of the vector, raise to the power of p, sum all these terms, and then raise to the power of (1/p).

There are 3 p-norms that are particularly important:

P=1 : describe (this is the democratic norm, where every entry of the vector contributes equally)

P=2: describe (this is the Euclidean norm, commonly used to measure the Euclidean distance between two points)

P=infinity (this is the dictator norm, where only the largest entry contributes)

**Sparse matrices**

Some matrices contain many zeros, and storing all these zeros can waste a lot of storage, and impact the computational efficiency of the methods that use these matrices.

Sparse matrices are defined as matrices that have very few non-zero entries.

An m by n matrix would in general need to store m times n entries. For practical purposes, such a matrix would be called sparse if it has up to either m or n non-zero entries, whatever is the smaller value.

It is important to note that once we have the matrices stored using sparse format, we can also perform computations efficiently, without computing operations on zeros. This will be essential when solving real simulations, such as vehicle impact and fluid flow.

In your next MP, you will be performing computations in a truss system, such as the one illustrated by this crane, where you will be solving a system of linear equations K u = F, and K has a sparse and banded representation.

**Condition number**

I am sure that you have faced with this problem in your life: you want to set the temperature of the water for your shower. You probably have that perfect faucet position that gives your favorite water temperature. A little bit to the left, and it is too hot, a little to the right, and it is too cold. This is a simple example of a system in which small changes in the input ( here the position of the faucet), generates big changes in the output (here the big swings in water temperature). How can we characterize such systems?

In this video, we will investigate the impact of perturbations on the input when we are solving linear system of equations.

When solving A x = b, we can think of A and b as input values, and x as the output, or the result of the solve function (MAKE DRAWING)

The input has uncertainties that may ,for example, arise from the finite representation of numbers inside the computer.

Once you select your numerical method, (in this case, which algorithm you are going to use for the solve function), then you need to be able to answer the question: How sensitive is your result to perturbations in the input?

Suppose we start with a non-singular matrix A and we want to solve Ax=b. We will first assume that we have perturbations on the right-hand size given as delta b. This change in the input will cause a change in the output delta x. Can we estimate how large is this change?

We will start by defining the ratio between the output relative error and the input relative error.

First we will look at one example. We want to solve A x = b, starting by defining A as a random matrix.

Here is the formal definition: a condition number is a ….

IT is measured as the product of the norm of the inverse of A with the norm of A.

Recall that the induced matrix norm depends on the choice of the p value of the vector norm.

Because of that, we should specify which condition number we want to compute, by the use of the index for the p value. If the index is omitted, it is assumed the default value of p=2.

We know that we can compute the induced p=1 norm of a matrix by taking the maximum absolute column sum, the p=infinity norm by taking the maximum absolute row sum, and the p=2 norm by taking the maximum singular value.

So if you have a diagonal matrix, it should be straight-forward to compute the condition number:

We saw that the optimal condition number of a matrix is equal to one. It is possible to show that orthogonal matrices always have optimal condition numbers, and as such, they are very well-behaved in computation.

We say that a matrix with low condition number is well-conditioned and a matrix with large condition number is ill-conditioned.