CSCC37

Main Notes

Contents

1 Floating Point Arithmetic **2**
1.1 Overview . . . 2

2 Solving Linear Systems of Equations **3**
2.1 Overview . . . 3
2.2 Direct Method . . . 3
2.3 Indirect Method . . . 4
2.4 Iterative Refinement/Improvement . . . 4

3 Solving Non-linear Single Equations **5**
3.1 Overview . . . 5

1 Floating Point Arithmetic

1.1 Overview

A broad overview of this chapter is studying how computers represent real numbers and how we manipulate them. This is because while some real numbers are much easier, like integers, dealing with irrational values, even just seemingly simple decimal values like 0.1 can be challenging. Often when dealing with floating point values we deal with issues such as rounding or even chopping, which lead to round off error, and can subsequently lead to error propagation. For example, if we have a computer that can only handle four digit floating point values, and we subtract 0.1234367 and 0.1234216, we would expect something like $1.51 * 10^{-4}$, but because our computer can only do four digits, we actually get 0. While this may seem close enough, "close enough" is relative. If we are only dealing with very small values, this is a massive issue.

Definition – Conditioning/Stability

These two both refer to how we measure how small changes in the initial input impact the final result. While both do this, they apply to different things. Conditioning applies specifically to functions, where we say the condition of f or $\text{cond}(f)$. Stability meanwhile refers to algorithms where we can have multiple inputs and outputs. The value is always a number that indicates the amplification of the impact.

2 Solving Linear Systems of Equations

2.1 Overview

Notation

A in this context refers to a square matrix. We should only be handling square matrices because non-squares are handled in the next course. Row/Column vectors are denoted with an underline such as $\underline{x}$ or $\underline{b}$.

When referring to the triangular matrices, we use $\triangle^r$. Refer to B24 notes for definitions on what is an upper/lower (strict) matrix.

Here we will be exploring how to solve these systems, how error propagation affects them and how to control the error propagation.

Review Linear Algebra

Review concepts such as triangular matrices, square matrices and their properties.

2.2 Direct Method

So as a recap, our problem is $A\underline{x} = \underline{b}$. We know A and $\underline{b}$, and we want to solve for $\underline{x}$.

Do you want to learn more?

This is the method that Pancer is using for this course. More information can be found in the textbook Scientific Computing: An Introductory Survey.

The direct method is the method already introduced in A22 but with some extra steps. Remember that while the process might be like "why do it like this, why these inbetweener steps", its all to simplify the calculations.

1. Factorize A
   We will factorize $A = LU$, where L is a unit lower $\triangle^r$ and V is an upper $\triangle^r$.

   Why are they square

   L and U don't necessarily have to be square, but as you will see in the later steps, it helps that they are.

   Note that by doing this, we have turned A, which is an unstructured matrix that is likely dense, into two highly structured matrices.

2. Restate the problem
   So now:
   $$A\underline{x} = \underline{b} \iff LU\underline{x} = \underline{b} \tag{1}$$

3. Next, we will do a change of variable and make
   $$L\underline{d} = \underline{b} \tag{2}$$
   Where
   $$\underline{d} = U\underline{x} \tag{3}$$

4. Next, solve for $\underline{d}$. So if we expand we will have something like:

$$\begin{bmatrix} 1 & & 0 \\ \ell_{21} & 1 & \\ \vdots & \ddots & \\ \ell_{n1} & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \ell_{21}d_1 + d_2 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \end{bmatrix} \tag{4}$$

   Which honestly is not too bad, second result depends on first, third on second, etc. It just unfolds like that. If this looks confusing, *remember* that we have ℓ_{ii} and we have b_i. We just have to work our way down. This is called **forward elimination**.

5. Then solve $U\underline{x} = \underline{d}$ for $\underline{x}$. We have to do this because the point is not to solve D, but to solve $\underline{x}$.

$$\begin{bmatrix} u_{11} > 0 & & \cdots & u_{1n} > 0 \\ 0 & u_{22} > 0 & & \\ \vdots & & \ddots & \\ 0 & \cdots & \cdots & u_{nn} > 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} \tag{5}$$

   And from this it is obvious that $x_n = \frac{d_n}{u_{nn}}$. And thus we can calculate x_i. Here we have to work our way up as opposed to step 4 where we worked down. this is called **backwards substitution**.

Just note that this process has a lot of error propagation. One of the reasons is because this algorithm has a runtime of $\mathcal{O}(n^3)$ and as we run more, we perform more operations which causes more rounding or chopping. This will be elaborated on later.

2.3 Indirect Method

Definition – X-Test

TBA

Definition – F-Test

TBA

2.4 Iterative Refinement/Improvement

3 Solving Non-linear Single Equations

3.1 Overview

For this section we are only doing single equations because doing multiple requires knowledge of multivariable calculus (and fuck that shit).
Non-linear problems are harder because of the fact that we don't always know how many solutions there will be.

Definition – Fixed Point Problem – FPP

The FPP relates directly with the FPI, or Fixed Point Iteration

Definition – Fixed Point Iteration – FPI

TBA