Lecture 1
Introduction

1.1 Origins of my teaching this course

I have taught Math 32 many times since I came to Brandeis in 1960—in fact probably as often as everyone else in the department combined. But this time it is going to be somewhat different and I want to tell you a little about how I happen to be teaching it again seven years after I became emeritus.

I became interested in mathematical visualization (using computer graphics) about a dozen years ago, and I have been working on a program called 3D-XplorMath that displays a large number of mathematical objects and processes ever since. In fact I retired from teaching earlier than I otherwise might have in order to have more time to think about Mathematical Visualization and continue development of the program. If you are a Mac user and would like to try out 3D-XplorMath, it is available from my web-site:

http://rsp.math.brandeis.edu/3D-XplorMath/TopLevel/download.html

or from VersionTracker.

The program should be thought of as a Mathematical Museum. Although it wasn't originally designed as a teaching tool many people started using it as an adjunct to their instruction and wrote back to me encouraging me to add more features for that purpose. So about two years ago I applied for an NSF CCLI grant for funding to turn 3DXM into a curriculum enhancement tool. I am the Principle Investigator on the grant and about six other mathematicians from all over the world (The 3DXM Consortium) are working with me on it.

As part of the grant activity we proposed to develop curricula for courses in ODE and differential geometry, using 3DXM to enhance the teaching in those subjects, and to give some courses in these subjects to test the new curricula. We gave the first of those pilot courses last winter in Taiwan to a class of about this size. It was a "crash course" that met six hours a week for six weeks and I was part of a team of five who participated in the instruction.

I think that we were all very surprised at how well the course went and at the enthusiasm of the students for the way it was taught. In the end we all felt that the success of the course came not just from using 3DXM to help the students visualize the concepts of the course, but even more from another learning technique that we stressed. Namely we had the students form into teams, tht worked together to write their own software to implement the theoretical material of the course algorithmically. In fact I was so impressed by the students enthusiasm that I asked to teach Math 32a this Fall and use the same approach.

What does it mean to "implement the theoretical material of the course algorithmically"? That may sound like just fancy jargon, but I have something very real and specific in mind and since it will play an important role in the way this course is taught, I want to try to explain it to you now—or at least start.

Mathematical theorems are often described as being either constructive or non-constructive. What exactly is the difference? Let me illustrate with two very famous theorems,

Banach Contraction Principle. *If X is a closed subset of $\mathbf{R}^n$ and $F : X \to X$ satisfies $\|F(x) - F(y)\| \leq K\|x - y\|$ for all $x, y \in X$ with $K < 1$ then there is a unique point p of X such that $F(p) = p$, and moreover for any point x of X the sequence $F(x), F(F(x)), F(F(F(x))), \ldots$ converges to p.*

Brouwer Fixed Point Theorem. *If D s the unit disk in the plane and F is any continuous map of D into itself, then there is a point p of D such that $F(p) = p$.*

These two theorems may appear superficially similar in that both assert the existence of point p left fixed by a particular kind of map. However, while in the first case the proof (and the very statement of the theorem) give an algorithm for finding p, in the second case there is no such algorithm, Instead the proof is by contradiction—it shows that if there were no such p, then it would be possible to continuously map the disk onto its boundary by a map leaving each point of the boundary fixed, and this is known to be impossible.

▷ **1.1—Exercise 1.** Let $X = [0, 1]$ and define $F : X \to X$ by $F(x) = \cos(x)$. Use the Mean Value Theorem to prove that F satisfies the hypothesis of the Banach Contraction Principle, and use a hand calculator to estimate the fixed point of F to two decimal places.

1.2 Algorithmic Mathematics

By doing mathematics algorithmically I mean using a computer and some programming system to actually "create" mathematical objects from constructive proofs of their existence. But what does it really mean to construct some desired mathematical object on a computer?

1) First one has to define data structures that describe the mathematical object one is trying to create and also the other objects that arise in the existence proof. (For example, a point in the plane is described by a pair (x,y) of floating point numbers, and a triangle is described by three such pairs.)
2) Then one has to translate the mathematical algorithms for constructing the mathemtatical object into subroutines in the programming system that act on the given data sructures to construct a data structure describing the desired object.
3) Finally, one has to write graphical procedures to display the data describing the created object in a meaningful way.

If one has to start from scratch, using a standard low-level programming system like Pascal, or C or Java, this can be a very difficult task and time-consuming task. But fortunately there are several excellent "high-level" mathematical programming systems that have already done much of the work of defining good mathematical data structures and writing important functions and subroutines that act on these data structures as well as easy to use routines for displaying these mathematical data structures visually. Perhaps the three most popular are Matlab, Mathematica and Maple.

Mathematica and Maple are fairly similar. Both were designed to work primarily with symbolic expressions, so they are particularly suited for doing algebra and simplifying complicated expressions. Matlab is primarily a system for doing numerical analysis. Its basic data structures are vectors and matrices, so it excels in problems that involve numerical linear algebra. And all three have good visualization "back-ends" for displaying the results of computation.

It turns out that Matlab's strong points make it particularly well-suited to carrying out the sort of projects we will encounter in this course.

In the Taiwan course, we used both Mathematica and Matlab, and students were able to carry out the projects with both, but our experience was that it was easier to teach the sudents Matlab and the students found it easier to work with. However, if some of you already know Mathematica (or Maple) and would likem to use it to do the projects, that is fine with me.

What are the main programming projects I have in mind. Three of the central theorems in curve and surface theory have very beautiful construcive proofs. When I taught these theorems before I never stressed their constructive nature. But in fact actually translating these theorems into code is quite feasible even for programming beginners, and doing so will not only make the meaning of the theorems and the proofs much clearer for you, but also it will be an excellent way for you to become expert in Matlab, a skill that you should find very useful in the years ahead,

1.3 What next?

That finishes my introduction to the course. In the next lecture we will begin by trying to make sense of the question, "What is "Geometry?" Geometry seems such a familiar and ancient notion that you may be surprised to hear that the mathematicians current conception of the subject underwent a substantial reformulation a little over a century ago by the German mathematician Felix Klein in his so-called "Erlanger Program". As preparation for my lecture, try looking up "Felix Klein" and "Erlanger Program" on Google.