A Modern Course on Curves and Surfaces

Richard S. Palais

Contents

Lecture 1. Introduction **1**
Lecture 2. What is Geometry **4**
Lecture 3. Geometry of Inner-Product Spaces **7**
Lecture 4. Linear Maps and the Euclidean Group **11**
Lecture 5. Adjoints of Linear Maps and the Spectral Theorem **14**
Lecture 6. Differential Calculus on Inner-Product Spaces **18**
Lecture 7. Normed Spaces and Integration **22**
Lecture 8. Ordinary Differential Equations (aka ODE) **30**
Lecture 9. Linear ODE and Numerical Methods **36**
Lecture 10. The Theorem of Frobenius **41**
Lecture 11. Differenttable Parametric Curves **47**
Lecture 12. Curves in 3-Space **54**
Lecture 13. The Fundamental Forms of a Surface **60**
Lecture 14. The Fundamental Theorem of Surface Theory **68**
Appendix I. The Matlab Projects **75**
Appendix II. Homework and Exams **94**
Appendix III. Matlab Notes **109**

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.

Lecture 2
What is Geometry?

2.1 Groups of Transformations

Let X be some set. In the following we will refer to X as "space". By a *transformation* or *bijection* of X we will mean a mapping $f : X \to X$ that is "one-to-one and onto". This means that:

1) if $f(x_1) = f(x_2)$ then $x_1 = x_2$,
2) every y in X is of the form $f(x)$ for some x in X—which by 1) is clearly unique,

and then the inverse of f, denoted by f^{-1}, is defined using 2) to be the mapping of X to itself $f^{-1}(y) := x$.

▷ **2.1—Exercise 1.** Check that f^{-1} is also a bijection of X.

Recall that if f and g are any two self-mappings of X, then their *composition* $f \circ g$ is the mapping of X to X defined by $(f \circ g)(x) := f(g(x))$.

▷ **2.1—Exercise 2.** Show that the composition of two bijections of X is again a bijection of X. Show by an example that composition of bijections is **not** necessarily commutative, i.e., it is not true in general that $f \circ g = g \circ f$. (Hint: Take for X the three element set $\{1, 2, 3\}$.) On the other hand, show that $f \circ f^{-1}$ and $f^{-1} \circ f$ are always both equal to the *identity transformation* of X, i.e., the transformtion of X that maps each elemet of X to itself. Show that the inverse of $f \circ g$ is $g^{-1} \circ f^{-1}$.

2.1.1 Definition. A set G of transformations of X is called a *group* of transformations of X if:

1) it contains the identity transformation,
2) it is closed under composition, and
3) if a transformation f is in G then so also is f^{-1}.

Clearly the set Biject(X) of all transformations of X is the largest group of transformations of X, and the set consisting of only the identity map of X is the smallest.

▷ **2.1—Exercise 3.** Show that the intersection of any collection of groups of transformations of X is itself a group of transformations of X. Deduce that if S is any set of transformations of X, then there is is a smallest group of transformations of X that includes S. This group is called the group generated by S.

2.1.2 Felix Klein's Erlanger Program. In 1872, the German mathematician Felix Klein, recently appointed to a chair at Erlanger, proposed a new viewpoint towards Geometry. For centuries after Euclid, there was only Euclidean geometry, but then, in the 1800s many new and different geometries were introduced and studied; for example spherical geometry, hyperbolic geometry, projective geometry, affine geometry, and more. Eventually mathematicians felt the need to elucidate just what a geometric theory was and how to classify the various different geometries. Klein observed that each geometry had associated to it a group of transformations, the symmetry group of the geometry, and two objects of the geometry could be considered equivalent if there was an element of that group that carried one object to the other. For example, the symmetry group of Euclidean geometry is the so-called Euclidean Group—generated by the rotations and translations of Euclidean space—and two triangles are considered to be "equivalent" in Euclidean geometry if they are congruent, meaning that there is some element of the Euclidean group that carries one triangle into another. To quote Klein:

> "...geometrical properties are characterized by their invariance under a group of transformations."

That last sentence epitomizes what has come to be known as Klein's Erlanger Program, and just what it means should gradually become more clear as we proceed.

2.2 Euclidean Spaces and Their Symmetries

We shall as usual denote by $\mathbf{R}^n$ the space of all ordered n-tuples $x = (x_1, \ldots, x_n)$ of real numbers. The space $\mathbf{R}^n$ has two important structures. The first is algebraic, namely $\mathbf{R}^n$ is a real vector space (of dimension n). If α is a "scalar", i.e., a real number, then the product αx is the n-tuple $(\alpha x_1, \ldots, \alpha x_n)$, and if $y = (y_1, \ldots, y_n)$ is a second element of $\mathbf{R}^n$, then the vector sum $x + y$ is the n-tuple $(x_1 + y_1, \ldots, x_n + y_n)$. The second important structure is the so-called *inner product*, which is a real-valued function on $\mathbf{R}^n \times \mathbf{R}^n$, namely $(x, y) \mapsto \langle x, y\rangle := x_1y_1 + x_2y_2 + \cdots + x_ny_n$. As we shall see, it is this inner product that allows us to define geometric concepts like length and angle in $\mathbf{R}^n$. Sometimes the inner product is referred to as the "dot-product" and written as $x \cdot y$.

The inner product has three characteristic properties that give it its importance, namely:

1) Symmetry: $\langle x, y\rangle = \langle y, x\rangle$ for all $x, y \in \mathbf{R}^n$
2) Positive Definiteness: $\langle x, x\rangle \geq 0$, with equality if and only if $x = 0$.
3) Bilinearity: $\langle \alpha x + \beta y, z\rangle = \alpha \langle x, z\rangle + \beta \langle y, z\rangle$, for all $x, y, z \in \mathbf{R}^n$ and all $\alpha, \beta \in \mathbf{R}$.

▷ **2.2—Exercise 1.** Verify that $\langle x, y\rangle$ has these three properties.

More generally:

2.2.1 Definition. . If V is a real vector space, then a real-valued function on $V \times V$, $(v_1, v_2) \mapsto \langle v_1, v_2\rangle$ is called an *inner product for* V if it is symmetric, positive definite, and bilinear. An *inner-product space* is a real vector space V together with a fixed choice of inner product for V.

We recall that that for any $x \in \mathbf{R}^n$, we define its norm, $\|x\|$, by $\|x\| := \sqrt{\langle x, x\rangle}$, and of course, we make this same definition in any inner product space.

▷ **2.2—Exercise 2.** Show that if $x, y \in \mathbf{R}^n$ and $t \in \mathbf{R}$, then $\|tx + y\|^2$ is a quadratic polynomial function of t, namely:

$$\|tx + y\|^2 = \langle tx + y, tx + y\rangle = \|x\|^2 t^2 + 2\langle x, y\rangle t + \|y\|^2 .$$

and note the important special case

$$\|x + y\|^2 = \|x\|^2 + 2\langle x, y\rangle + \|y\|^2 .$$

Finally, for reasons we shall see a little later, the two vectors x an y are called *orthogonal* if $\langle x, y\rangle = 0$, so in this case we have:

Pythagorean Identity. *If x and y are orthogonal vectors in an inner product space then* $\|x + y\|^2 = \|x\|^2 + \|y\|^2$.

Now let me remind you of some basic facts from high-school mathematics concerning an arbitrary real polynomial $P(t) = at^2 + bt + c$ in a variable t. (For simplicity, we will assume tha $a > 0$.) Recall that the *discriminant* of $P(t)$ is the quantity $b^2 - 4ac$, and it distinguishes what kind of roots the polynomial has. In fact, the so-called "Quadratic Formula" says that the two (possible complex) roots of $P(t)$ are $(-b \pm \sqrt{b^2 - 4ac})/2a$. Thus there are three cases:

Case 1: $b^2 - 4ac > 0$. Then $P(t)$ has two real roots. Between these roots $P(t)$ is negative and outside of the interval between the roots it is positive.

Case 2: $b^2 - 4ac = 0$. Then $P(t)$ has only the single real root $-b/2a$, and elsewhere $P(t) > 0$.

Case 3: $b^2 - 4ac < 0$. Then $P(t)$ has no real roots and $P(t)$ is positive for all real t.

In the case of the polynomial $\|tx + y\|^2$, we see that $a = \|x\|^2$, $c = \|y\|^2$, and $b = 2\langle x, y\rangle$, so the discriminant is $4(|\langle x, y\rangle|^2 - \|x\|^2 \|y\|^2)$. Now Case 1 is ruled out by positive definiteness. In Case 2, we have $|\langle x, y\rangle| = \|x\| \|y\|$, so if t is the root of the polynomial then $\|x + ty\| = 0$, so x = -ty, and we see that in this case x and y are linearly dependent. Finally, in Case 3, $|\langle x, y\rangle| < \|x\| \|y\|$, and since $x + ty$ is never zero, x and y are linearly independent. This proves one of the most important inequalities in all of mathematics,

Schwartz Inequality. *For all $x, y \in \mathbf{R}^n$, $|\langle x, y\rangle| \leq \|x\| \|y\|$, with equality if and only if x and y are linearly dependent.*

Of course, since our proof only used the three properties that define an inner product, the Schwartz Inequality is valid in any inner-product space.

▷ **2.2—Exercise 3.** Use the Schwartz Inequality to deduce the Triangle Inequality:

$$\|x + y\| \leq \|x\| + \|y\| .$$

(Hint: Square both sides.)

2.2—Example 1. Let $C([a, b])$ denote the vector space of continuous real-valued functions on the interval $[a, b]$ (with pointwise vector operations, as usual). For $f, g \in C([a, b])$ define $\langle f, g\rangle = \int_a^b f(x)g(x)\,dx$. it is easy to check that this satisfies our three conditions for an inner product. What does the Schwartz Inequality say in this case?

Lecture 3
Geometry of Inner Product Spaces

3.1 Angles

Let x and y be two non-zero vectors in an inner product space. Then by the Schwartz inequality, the ratio $\langle x, y\rangle \,/\, \|x\|\,\|y\|$ lies in the interval $[-1, 1]$, so there is a unique angle θ between 0 and π such that $\cos(\theta) = \langle x, y\rangle \,/\, \|x\|\,\|y\|$. In other words, we define θ to make the identity $\langle x, y\rangle = \|x\|\,\|y\|\cos(\theta)$ hold. What is the geometric meaning of θ? Let's first consider a special case. Namely take for x the unit vector in the x direction, $(1, 0)$, and let y be an arbitrary vector in $\mathbf{R}^2$. If $r = \|y\|$ and ϕ is the angle between x and y (the so-called polar angle of y), then clearly $y = (r\cos(\phi), r\sin(\phi))$, so it follows that $\langle x, y\rangle = (1)(r\cos(\phi)) + (0)(r\sin(\phi)) = r\cos(\phi)$ and hence $\langle x, y\rangle \,/\, \|x\|\,\|y\| = \cos(\phi)$, so in this case the angle θ is exactly the angle ϕ between x and y.

▷ **3.1—Exercise 1.** Carry out the computation for the general case of two non-zero vectors in the plane with lengths r_1 and r_2 and polar angles ϕ_1 and ϕ_2, so that $x = (r_1\cos(\phi_1), r_1\sin(\phi_1))$ and $y = (r_2\cos(\phi_2), r_2\sin(\phi_2))$. Show that in this case too the ratio $\langle x, y\rangle \,/\, \|x\|\,\|y\|$ is the cosine of the angle $(\phi_1 - \phi_2)$ between x and y. (Hint: use the Cosine Addition Formula: $\cos(A \pm B) = \cos(A)\cos(B) \mp \sin(A)\sin(B)$.)

Henceforth we will refer to θ as the angle between x and y. In particular, if $\langle x, y\rangle = 0$, so that $\theta = \pi/2$, then we say that x and y are *orthogonal*.

3.2 Orthonormal Bases for an Inner Product Space

We begin by recalling the basic facts concerning linear dependence, dimension, and bases in a vector space V. (If you prefer to be concrete, you may think of V as being $\mathbf{R}^n$.) We say that vectors $v_1, \ldots, v_n$ in V are *linearly dependent* if there are scalars $\alpha_1, \ldots, \alpha_n$, **not all zero**. such that the linear combination $\alpha_1 v_1 + \cdots + \alpha_n v_n$ is the zero vector. It is easy to see that this is equivalent to one of the v_i being a linear combination of the others. If $v_1, \ldots, v_n$ are **not** linearly dependent, than we say that they are linearly independent. The vectors $v_1, \ldots, v_n$ are said to *span* V if every element of V can be written as a linear combination of the v_i, and if V is spanned by some finite set of vectors then we say that V finite dimensional, and we define the dimension of V, $\dim(V)$, to be the least number of vectors needed to span V. A finite set of vectors $v_1, \ldots, v_n$ in V is called a *basis* for V if it is both linearly independent and spans V. It is easy to see that this is equivalent to demanding that every element of V is a **unique** linear combination of the v_i. The following is a the basic theorem tying thse concepts together.

Theorem. *If V is an n-dimensional vector space, then every basis for V has exactly n elements. Moreover, if $v_1, \ldots, v_n$ is any set of n elements of V, then they form a basis for V if and only if they are linearly independent or if and only if they span V. In other words, n elements of V are linearly independent if and only if they span V.*

In what follows, we assume that V is an inner-product space. If $v \in V$ is a non-zero vector, we define a unit vector e with the same direction as V by $e := v/\|v\|$. This is called *normalizing* v, and if v already has unit length then we say that v is *normalized*. We say that k vectors $e_1, \ldots, e_k$ in V are *orthonormal* if each e_i is normalized and if the e_i are mutually orthogonal. Note that these conditions can be written succinctly as $\langle e_i, e_j \rangle = \delta^i_j$, where δ^i_j is the so-called Kronecker delta symbol and is defined to be zero if i and j are different and 1 if they are equal.

▷ **3.2—Exercise 1.** Show that if $e_1, \ldots, e_k$ are orthonormal and v is a linear combination of the e_i, say $v = \alpha_1 v_1 + \cdots + \alpha_k v_k$, then the α_i are uniquely determined by the formulas $\alpha_i = \langle v, e_i \rangle$. Deduce from this that orthonormal vectors are automatically linearly independent.

Orthonormal bases are also referred to as *frames* and as we shall see they play an extremely important role in all things having to do with explicit computation in inner-product spaces. Note that if $e_1, \ldots, e_n$ is an orthonormal basis for V then every element of V is a linear combination of the e_i, so that by the exercise each $v \in V$ has the expansion $v = \sum_{i=1}^n \langle v, e_i \rangle \, e_i$.

3.2—Example 1. The "standard basis" for $\mathbf{R}^n$, is $\delta^1, \ldots, \delta^n$, where $\delta^i = (\delta^1_1, \ldots, \delta^i_n)$. It is clearly orthonormal.

3.3 Orthogonal Projection

Let V be an inner product space and W a linear subspace of V. We recall that the *orthogonal complement* of W, denoted by $W^\perp$, is the set of those v in V that are orthogonal to every w in W.

▷ **3.3—Exercise 1.** Show that $W^\perp$ is a linear subspace of V and that $W \cap W^\perp = 0$.

If $v \in V$, we will say that a vector w in W is its orthogonal projection on W if $u = v - w$ is in $W^\perp$.

▷ **3.3—Exercise 2.** Show that there can be st most one such w. (Hint: if w' is another, so $u' = v - u \in W^\perp$ then $u - u' = w' - w$ is in both W and $W^\perp$.)

3.3.1 Remark. Suppose $\omega \in W$. Then since $v - \omega = (v - w) + (w - \omega)$ and $v - w \in W^\perp$ while $(w - \omega) \in W$, it follows from the Pythagorean identity that $\|v - \omega\|^2 = \|v - w\|^2 + \|w - \omega\|^2$. Thus, $\|v - \omega\|$ is strictly greater than $\|v - w\|$ unless $\omega = w$. In other words, **the orthogonal projection of v on w is the unique point of W that has minimum distance from v.**

We call a map $P : V \to W$ *orthogonal projection of V onto W* if $v - Pv$ is in $W^\perp$ for all $v \in V$. By the previous exercise this mapping is uniquely determined if it exists (and we will see below that it always does exist).

▷ **3.3—Exercise 3.** Show that if $P : V \to W$ is orthogonal projection onto W, then P is a linear map. Show also that if $v \in W$, then $Pv = v$ and hence $P^2 = P$.

▷ **3.3—Exercise 4.** Show that if $e_1, \ldots, e_n$ is an orthonormal basis for W and if for each $v \in V$ we define $Pv := \sum_{i=1}^{n} \langle v, e_i \rangle e_i$, then P is orthogonal projection onto W. In particular, orthogonal projection onto W exists for any subspace W of V that has some orthonormal basis. (Since the next section shows that any W has an orthonormal basis, orthogonal projction on a subspace is always defined.)

3.4 The Gram-Schmidt Algorithm

There is a beautiful algorithm, called the Gram-Schmidt Procedure, for starting with an arbitrary sequence $w_1, w_2, \ldots, w_k$ of linearly independent vectors in an inner product space V and manufacturing an orthonormal sequence $e_1, \ldots, e_k$ out of them, Moreover it has the nice property that for all $j \leq k$, the sequence $e_1, \ldots, e_j$ spans the same subspace W_j of V as is spanned by $w_1, \ldots, w_j$.

In case $k = 1$ this is easy. To say that w_1 is linearly independent just means that it is non-zero, and we take e_1 to be its normalization: $e_1 := w_1 / \|w_1\|$. Surprisingly, this trivial special case is the crucial first step in an inductive procedure.

In fact, suppose that we have constructed orthonormal vectors $e_1, \ldots, e_m$ (where $m < k$) and that they span the same subspace W_m that is spanned by $w_1, \ldots, w_m$. How can we make the next step and construct e_{m+1} so that $e_1, \ldots, e_{m+1}$ is orthonormal and spans the same subspace as $w_1, \ldots, w_{m+1}$?

First note that since the $e_1, \ldots, e_m$ are linearly independent and span W_m, they are an orthonormal basis for W_m, and hence we can find the orthogonal projection ω_{m+1} of w_{m+1} onto W_m using the formula $\omega_{m+1} = \sum_{i=1}^{m} \langle w_{m+1}, e_i \rangle e_i$. Recall that this means that $\epsilon_{m+1} = w_{m+1} - \omega_{m+1}$ is orthogonal to W_m, and in particular to $e_1, \ldots, e_m$. Now ϵ_{m+1} **cannot be zero!** Why? Because if it were then we would have $w_{m+1} = \omega_{m+1} \in W_m$, so w_{m+1} would be a linear combination of $w_1, \ldots, w_m$, contradicting the assumption that $w_1, \ldots, w_k$ were linearly independent. But then we can define e_{m+1} to be the normalization of ϵ_{m+1}, i.e., $e_{m+1} := \epsilon_{m+1} / \|\epsilon_{m+1}\|$, and it follows that e_{m+1} is also orthogonal to $e_1, \ldots, e_m$, so that $e_1, \ldots, e_{m+1}$ is orthonormal. Finally, it is immediate from its definition that e_{m+1} is a linear combination of $e_1, \ldots, e_m$ and w_{m+1} and hence of $w_1, \ldots, w_{m+1}$, completing the induction. Let's write the first few steps in the Gram-Schmidt Process explicitly.

1 $e_1 := w_1 / \|w_1\|$. % Normalize w_1 to get e_1.

2a $\omega_2 := \langle w_2, e_1 \rangle e_1$. % Get projection ω_2 of w_2 on W_1,

2b $\epsilon_2 := w_2 - \omega_2$. % subtract ω_2 from w_2 to get $W_1^\perp$ component ϵ_2 of w_2,

2c $e_2 := \epsilon_2 / \|\epsilon_2\|$. % and normalize it to get e_2.

3a $\omega_3 := \langle w_3, e_1 \rangle e_1 + \langle w_3, e_2 \rangle e_2$. % Get projection ω_3 of w_3 on W_2,

3b $\epsilon_3 := w_3 - \omega_3$. % subtract ω_3 from w_3 to get $W_2^\perp$ component ϵ_3 of w_3,

3c $e_3 := \epsilon_3 / \|\epsilon_3\|$. % and normalize it to get e_3.

...

If W is a k-dimensional subspace of an n-dimensional inner-product space V then we can start with a basis for for W and extend it to a basis for V. If we now apply Gram-Schmidt to this basis, we end up with an orthonormal basis for V with the first k elements in W and with the remaining $n-k$ in $W^\perp$. This tells us several things:

- $W^\perp$ has dimension $n-k$.
- V is the direct sum of W and $W^\perp$. This just means that every element of V can be written uniquely as the sum $w+u$ where $w \in W$ and $u \in W^\perp$.
- $(W^\perp)^\perp = W$.
- If P is the orthogonal projection of V on W and I denotes the identity map of V then $I-P$ is orthogonal projection of V on $W^\perp$.

▹Project 1. Implement Gram-Schmidt as a Matlab Function

In more detail, create a Matlab M-file GramSchmidt.m in which you define a Matlab function GramSchmidt(M) taking as input a rectangular matrix M of real numbers of arbitrary size $m \times n$, and assuming that the m rows of M are linearly independent, it should transform M into another $m \times n$ matrix in which the rows are orthonormal, and moreover such that the subspace spanned by the first k rows of the output matrix is the same as the space spanned by the first k rows of the input matrix. Clearly, in writting your algorithm, you will need to know the number of rows, m and the number of columns n of M. You can find these out using the Matlab size function. In fact, size(M) returns (m,n) while size(M,1) returns m and size(M,2) returns n. Your algorithm will have to do some sort of loop, iterating over each row in order. Be sure to test your function on a number of different matrices of various sizes. What happens to your function if you give it as input a matrix with linearly dependent rows. (Ideally it should report this fact and not just return garbage!)

Lecture 4
Linear Maps And The Euclidean Group.

I assume that you have seen the basic facts concerning linear transformations and matrices in earlier courses. However we will review these facts here to establish a common notation. In all the following we assume that the vector spaces in question have finite dimension.

4.1 Linear Maps and Matrices

Let V and W be two vector spaces. A function T mapping V into W is called a *linear map* if $T(\alpha v_1 + \beta v_2) = \alpha T(v_1) + \beta T(v_2)$ for all scalars α, β and all $v_1, v_2 \in V$. We make the space $L(V, W)$ of all linear maps of V into W into a vector space by defining the addition and scalar multiplication laws to be "pointwise". i.e., if $S, T \in L(V, W)$, then for any $v \in V$ we define $(\alpha T + \beta S)(v) := \alpha T(v) + \beta S(v)$

4.1.1 Remark. If $v_1, \ldots, v_n$ is any basis for V and $\omega_1, \ldots, \omega_n$ are arbitrary elements of W, then there is a unique $T \in L(V, W)$ such that $T(v_i) = \omega_i$. For if $v \in V$, then v has a unique expansion of the form $v = \sum_{i=1}^n \alpha_i v_i$, and then we can define T by $T(v) := \sum_{i=1}^n \alpha_i \omega_i$, and it is easily seen that this T is linear, and that it is the unique linear transformation with the required properties.

In particular, if $w_1, \ldots, w_m$ is a basis for W, then for $1 \leq i \leq n$ and $1 \leq j \leq m$ we define E_{ij} to be the unique element of $L(V, W)$ that maps v_i to w_j and maps all the other v_k to the zero element of W.

4.1.2 Definition. Suppose $T : V \to W$ is a linear map, and that as above we have a basis $v_1, \ldots, v_n$ for V and a basis $w_1, \ldots, w_m$ for W. For $1 \leq j \leq n$, the element Tv_j of W has a unique expansion as a linear combination of the w_i, $T(v_j) = \sum_{i=1}^m T_{ij} w_i$. These mn scalars T_{ij} are called *the matrix elements of* T relative to the two bases v_i and w_j.

4.1.3 Remark. It does not make sense to speak of the matrix of a linear map until bases are specified for the domain and range. However, if T is a linear map from $\mathbf{R}^n$ to $\mathbf{R}^m$, then by its matrix we always understand its matrix relative to the standard bases for $\mathbf{R}^n$ and $\mathbf{R}^m$.

4.1.4 Remark. If V is a vector space then we abreviate $L(V, V)$ to $L(V)$, and we often refer to a linear map $T : V \to V$ as a *linear operator* on V. To define the matrix of a linear operator on V we only need one basis for V.

▷ **4.1—Exercise 1.** Suppose that $v \in V$ has the expansion $v = \sum_{j=1}^n \alpha_j v_j$, and that $Tv \in W$ has the expansion $Tv = \sum_{i=1}^m \beta_i w_i$. Show that we can compute the components β_i of Tv from the components α_j of v and the matrix for T relative to the two bases, using the formula $\beta_i = \sum_{j=1}^n T_{ij} \alpha_j$.

Caution! Distinguish carefully between the two formulas: $T(v_j) = \sum_{i=1}^m T_{ij} w_i$ and $\beta_i = \sum_{j=1}^n T_{ij} \alpha_j$. The first is essentially the definition of the matrix T_{ij} while the second is the formula for computing the components of Tv relative to the given basis for W from the components of v relative to the given basis for V.

▷ **4.1—Exercise 2.** Show that $T = \sum_{i=1}^n \sum_{j=1}^m T_{ij} E_{ij}$, and deduce that E_{ij} is a basis for $L(V, W)$, so in particular, $L(V, W)$ has dimension nm, the product of the dimensions of V and W.

4.2 Isomorphisms and Automorphisms

If V and W are vector spaces, then a linear map $T : V \to W$ is called an *isomorphism* of V with W if it is bijective (i.e., one-to-one and onto), and we say that V and W are *isomorphic* if there exists an isomorphism of V with W. An isomorphism of V with itself is called an *automorphism* of V, and we denote the set of all automorphisms of V by $\boldsymbol{GL}(V)$. ($\boldsymbol{GL}(V)$ is usually referred to as the general linear group of V—check that it is a group.)

▷ **4.2—Exercise 1.** If $T : V \to W$ is a linear map and $v_1, \ldots . v_n$ is a basis for V then show that T is an isomorphism if and only if $Tv_1, \ldots, Tv_n$ is a basis for W. Deduce that two finite-dimensional vector spaces are isomorphic if and only if they have the same dimension.

There are two important linear subspaces associated to a linear map $T : V \to W$. The first, called the *kernel* of T and denoted by $\ker(T)$, is the subspace of V consisting of all $v \in V$ such that $T(v) = 0$, and the second, called the *image* of T, and denoted by $\mathrm{im}(T)$, is the subspace of W consisting of all $w \in W$ of the form Tv for some $v \in V$.

Notice that if v_1 and v_2 are in V, then $T(v_1) = T(v_2)$ if and only if $T(v_1 - v_2) = 0$, i.e., if and only if v_1 and v_2 differ by an element of $\ker(T)$. Thus T is one-to-one if and only if $\ker(T)$ contains only the zero vector.

Proposition. *A necessary and sufficient condition for $T : V \to W$ to be an isomorphism of V with $\mathrm{im}(T)$ is for $\ker(T)$ to be the zero subspace of V.*

Theorem. *If V and W are finite dimensional vector spaces and $T : V \to W$ is a linear map, then $\dim(\ker(T)) + \dim(\mathrm{im}(T)) = \dim(V)$.*

PROOF. Choose a basis $v_1, \ldots, v_k$ for $\ker(T)$ and extend it to a basis $v_1, \ldots, v_n$ for all of V. It will suffice to show that $T(v_{k+1}), \ldots, T(v_n)$ is a basis for $\mathrm{im}(T)$. We leave this as an (easy) exercise.

Corollary. *If V and W have the same dimension then a linear map $T : V \to W$ is an isomorphism of V with W if it is* **either** *one-to-one or onto.*

Recall that if V is an inner product space and $v_1, v_2 \in V$, then we define the *distance between v_1 and v_2* as $\rho(v_1, v_2) := \|v_1 - v_2\|$. This makes any inner-product space into a metric space. A mapping $f : V \to W$ between inner-product spaces is called an *isometry* if it is distance preserving, i.e., if for all $v_1, v_2 \in V$, $\|f(v_1) - f(v_2)\| = \|v_1 - v_2\|$.

4.2.1 Definition. If V is an inner product space then we define the *Euclidean group* of V, denoted by $\boldsymbol{Euc}(V)$, to be the set of all isometries $f : V \to V$. We define the *orthogonal group* of V, denoted by $\boldsymbol{O}(V)$ to be the set of $f \in \boldsymbol{Euc}(V)$ such that $f(0) = 0$.

4.2.2 Remark. We will justify calling $\boldsymbol{Euc}(V)$ a group shortly. It is clear that $\boldsymbol{Euc}(V)$ is closed under composition, and that elements of $\boldsymbol{Euc}(V)$ are one-to-one, but at this point it is not clear that an element f of $\boldsymbol{Euc}(V)$ maps onto all of V, so f might not have an inverse in $\boldsymbol{Euc}(V)$. A similar remark holds for $\boldsymbol{O}(V)$.

Proposition. *If $f \in \boldsymbol{O}(V)$ then f preserves inner-products, i.e., if $v_1, v_2 \in V$ then $\langle fv_1, fv_2\rangle = \langle v_1, v_2\rangle$.*

PROOF. Clearly f preserves norms, since $\|f(v)\| = \|f(v) - f(0)\| = \|v - 0\| = \|v\|$, and we also know that, $\|f(v_1) - f(v_2)\|^2 = \|v_1 - v_2\|^2$. Then $\langle fv_1, fv_2\rangle = \langle v_1, v_2\rangle$ now follows easily from the polarization identity in the form: $\langle v, w\rangle = \frac{1}{2}(\|v\|^2 + \|w\|^2 - \|v - w\|^2)$.

Theorem. *$\boldsymbol{O}(V) \subseteq \boldsymbol{GL}(V)$, i.e., elements of $\boldsymbol{O}(V)$ are invertible linear transformations.*

PROOF. Let $e_1, \ldots, e_n$ be an orthonormal basis for V and let $\epsilon_i = f(e_i)$. By the preceding proposition $\langle \epsilon_i, \epsilon_j\rangle = \langle e_i, e_j\rangle = \delta^i_j$, so that the ϵ_i also form an orthonormal basis for V. Now suppose that $v_1, v_2 \in V$ and let α_i, β_i and , γ_i be respectively the components of v_1, v_2, and $v_1 + v_2$ relative to the orthonormal basis e_i, and similarly let α'_i, β'_i and , γ'_i be the components of $f(v_1), f(v_2)$, and $f(v_1 + v_2)$ relative to the orthonormal basis ϵ_i. To prove that $f(v_1 + v_2) = f(v_1) + f(v_2)$ it will suffice to show that $\gamma'_i = \alpha'_i + \beta'_i$. Now we know that $\gamma_i = \alpha_i + \beta_i$, so it will suffice to show that $\alpha'_i = \alpha_i$, $\beta'_i = \beta_i$, and $\gamma'_i = \gamma_i$. But since $\alpha_i = \langle v_1, e_i\rangle$ while $\alpha'_i = \langle f(v_1), \epsilon_i\rangle = \langle f(v_1), f(e_i)\rangle$, $\alpha'_i = \alpha_i$ follows from the fact that f preserves inner-products, and the other equalities follow likewise.. A similar argument shows that $f(\alpha v) = \alpha f(v)$. Finally, since f is linear and one-to-one, it follows that f is invertible. ∎

4.2.3 Remark. It is now clear that we can equivalently define $\boldsymbol{O}(V)$ to be the set of linear maps $T : V \to V$ that preserves inner-products.

Every $a \in V$ gives rise to a map $\tau_a : V \to V$ called *translation by a*, defined by, $\tau_a(v) = v + a$. The set $\mathcal{T}(V)$ of all $\tau_a, a \in V$ is clearly a group since $\tau_{a+b} = \tau_a \circ \tau_b$ and τ_0 is the identity. Moreover since $(v_1 + a) - (v_2 + a) = v_1 - v_2$, it follows that τ_a is an isometry, i.e. $\mathcal{T}(V) \subseteq \boldsymbol{Euc}(V)$

Theorem. *Every element f of $\boldsymbol{Euc}(V)$ can be written uniquely as an orthogonal transformation O followed by a translation τ_a.*

PROOF. Define $a := f(0)$. Then clearly the composition $\tau_{-a} \circ f$ leaves the origin fixed, so it is an element O of $\boldsymbol{O}(V)$, and it follows that $f = \tau_a \circ O$. (We leave uniqueness as an exercise.)

Corollary. *Every element f of $\boldsymbol{Euc}(V)$ is a one-to-one map of V onto itself and its inverse is also in V, so $\boldsymbol{Euc}(V)$ is indeed a group of transformations of V.*

PROOF. In fact we see that $f^{-1} = O^{-1} \circ \tau_{-a}$.

Lecture 5
Adjoints of Linear Maps and The Spectral Theorem

5.1 The Dual Vector Space of a Vector Space

If V is a vector space then the vector space $L(V, \mathbf{R})$ of linear maps of V into the one-dimensional vector space of scalars, $\mathbf{R}$ plays an important role in many considerations. It is called the *dual space* of V and is denoted by V^*. Elements of V^* are called *linear functionals* on V.

5.1.1 Remark. If $v_1, \ldots, v_n$ is any basis for V, then (since 1 is a basis for $\mathbf{R}$) it follows from 4.1.1 that there is a uniquely determined basis $\ell_1, \ldots, \ell_n$ for V^* such that $\ell_i(v_j) = \delta^i_j$. This is called the *dual basis* to $v_1, \ldots, v_n$. In particular, V^* has the same dimension as V.

5.2 The Self-Duality of Inner-Product Spaces

If V is an inner-product space, then there is an important way to represent elements of V^*. Namely, recall that the inner-product is linear in each variable when the other variable is held fixed. This means that for each vector $v \in V$ we can define an element $v^* : V \to \mathbf{R}$ of V^* by the formula $v^*(x) := \langle x, v\rangle$. Since the inner product is linear in **both** variables, it follows that the map $v \mapsto v^*$ is linear. If $v^* = 0$, then in particular $0 = v^*(v) = \langle v, v\rangle$, so by positive definiteness $v = 0$, i.e., the kernel of the map $v \mapsto v^*$ is zero, and since V and V^* have the same dimension, it follows that this map is an isomorphism—i.e., every linear functional on an inner-product space V can be expressed uniquely as the inner-product with some fixed element of V. This is often expressed by saying that inner-product spaces are "self-dual".

5.2.1 Remark. Note that if e_i is an orthonormal basis for V, then e^*_i is the dual basis for V^*.

5.3 Adjoint Linear Maps

Now let V and W be finite dimensional inner-product spaces and $T : V \to W$ a linear map. We will next define a linear map $T^* : W \to V$ called the adjoint of T that satisfies the identity $\langle Tv, w\rangle = \langle v, T^*w\rangle$ for all $v \in V$ and all $w \in W$.

If we fix w in W, then to define T^*w we note that the map $v \to \langle Tv, w\rangle$ is clearly a linear functional on V, i.e., an element of V^*, so by self-duality there is a uniquely defined element T^*w in V such that $\langle v, T^*w\rangle = \langle Tv, w\rangle$ for all v in V.

We recapitulate the above as a formal definition.

5.3.1 Definition. Let V, W be finite dimensional inner-product spaces and $T \in L(V, W)$. The *adjoint* of T is the unique element $T^* \in L(W, V)$ satisfying the identity:

$$\langle v, T^* w \rangle = \langle Tv, w \rangle .$$

Here are some exercises involving adjoints and their basic properties.

▷ **5.3—Exercise 1.** Show that $(T^*)^* = T$.

▷ **5.3—Exercise 2.** Recall that if T_{ij} is an $m \times n$ matrix (i.e., m rows and n columns) and S_{ji} an $n \times m$ matrix, then S_{ji} is called the *transpose* of T_{ij} if $T_{ij} = S_{ji}$ for $1 \leq i \leq m$, $1 \leq j \leq n$. Show that if we choose orthonormal bases for V and W, then the matrix of T^* relative to these bases is the transpose of the matrix of T relative to the same bases.

▷ **5.3—Exercise 3.** Show that $\ker(T)$ and $\mathrm{im}(T^*)$ are orthogonal complements in V, and similarly, $\mathrm{im}(T)$ and $\ker(T^*)$ are each other's orthogonal complements in W. (Note that by Exercise 1, you only have to prove one of these.)

▷ **5.3—Exercise 4.** Show that a linear operator T on V is in the orthogonal group $\boldsymbol{O}(V)$ if and only if $TT^* = I$ (where I denotes the identity map of V) or equivalently, if and only if $T^* = T^{-1}$.

If $T : V \to V$ is a linear operator on V, then T^* is also a linear operator on V, so it makes sense to compare them and in particular ask if they are equal.

5.3.2 Definition. A linear operator on an inner-product space V is called *self-adjoint* if $T^* = T$, i.e., if $\langle Tv_1, v_2 \rangle = \langle v_1, Tv_2 \rangle$ for all $v_1, v_2 \in V$.

Note that by Exercise 3 above, self-adjoint operators are characterized by the fact that their matrices with respect to an orthonormal basis are symmetric.

▷ **5.3—Exercise 5.** Show that if W is a linear subspace of the inner-product space V, then the orthogonal projection P of V on W is a self-adjoint operator on V.

5.3.3 Definition. If T is a linear operator on V, then a linear subspace $U \subseteq V$ is called a T-invariant subspace if $T(U) \subseteq U$, i.e., if $u \in U$ implies $Tu \in U$.

5.3.4 Remark. Note that if U is a T-invariant subspace of V, then T can be regarded as a linear operator on U by restriction, and clearly if T is self-adjoint, so is its restriction.

▷ **5.3—Exercise 6.** Show that if $T : V \to V$ is a self-adjoint operator, and $U \subseteq V$ is a T-invariant subspace of V, the $U^\perp$ is also a T-invariant subspace of V.

5.4 Eigenvalues and Eigenvectors of a Linear Operator.

In this section, $T : V \to V$ is a linear operator on a real vector space V. If λ is a real number, then we define the linear subspace $E_\lambda(T)$ of V to be the set of $v \in V$ such that $Tv = \lambda v$. In other words, if I denotes the identity map of V, then $E_\lambda(T) = \ker(T - \lambda I)$.

Of course the zero vector is always in $E_\lambda(T)$. If $E_\lambda(T)$ contains a non-zero vector, then we say that λ is an *eigenvalue* of T and that $E_\lambda(T)$ is the λ-eigenspace of T. A non-zero vector in $E_\lambda(T)$ is called an *eigenvector* of T belonging to the eigenvector λ. The set of all eigenvalues of T is called the *spectrum* of T (the name comes from quantum mechanics) and it is denoted by $\mathrm{Spec}(T)$.

▷ **5.4—Exercise 1.** Show that a linear operator T on V has a diagonal matrix in a particular basis for V if and only if each element of the basis is an eienvector of T, and that then $\mathrm{Spec}(T)$ consists of the diagonal elements of the matrix.

The following is an easy but very important fact.

Theorem. *If T is a self-adjoint linear operator on an inner-product space V and λ_1, λ_2 are distinct real numbers, then $E_{\lambda_1}(T)$ and $E_{\lambda_2}(T)$ are orthogonal subspaces of V. In particular, eigenvectors of T that belong to different eigenvalues are orthogonal.*

▷ **5.4—Exercise 2.** Prove this theorem. (Hint: Let $v_i \in E_{\lambda_i}(T), i = 1, 2$. You must show that $\langle v_1, v_2\rangle = 0$. Start with the fact that $\langle Tv_1, v_2\rangle = \langle v_1, Tv_2\rangle$.)

A general operator T on a vector space V need not have any eigenvalues, that is, $\mathrm{Spec}(T)$ may be empty. For example a rotation in the plane (by other than π or 2π radians) clearly has no eigenvectors and hence no eigenvalues. On the other hand self-adjoint operators always have at least one eigenvector. That is:

Spectral Lemma. *If V is an inner-product space of positive, finite dimension and if $T : V \to V$ is a self-adjoint operator on V, then $\mathrm{Spec}(T)$ is non-empty, i.e., T has at least one eigenvalue and hence at least one eigenvector.*

We will prove this result later after some preparation. But next we show how it leads to an easy proof of the extremely important:

Spectral Theorem for Self-Adjoint Operators. *If V is an inner inner-product space of posiive, finite dimension and $T : V \to V$ is a self-adjoint operator on V, then V has an orthonormal basis consisting of eigenvectors of T. In other words, T has a diagonal matrix in some orthonormal basis for V.*

PROOF. We prove this by induction on the dimension n of V. If $n = 1$ the theorem is trivial, since any non-zero vector in V is clearly an eigenvector, Thus we can assume that the theorem is valid for all self-adjoint operators on inner-product spaces of dimension less than n. By the Spectral Lemma, we can find at least one eigenvector w for T. Let $e_1 = w/\|w\|$ and let W be the one-dimensional space spanned by w. The fact that w is an eigenvector implies that W is a T-invariant linear subspace of V, and by 5.3, so is $W^\perp$. Since $W^\perp$ has dimension $n - 1$, by the inductive hypothesis T restricted to $W^\perp$ has an orthonormal basis $e_2, \ldots . e_n$ of eigenvectors of T, and then $e_1, \ldots, e_n$ is an orthonormal basis of V consisting of eigenvectors of T.

5.5 Finding One Eigenvector of a Self-Adjoint Operator

In this section we will outline the strategy for finding an eigenvector v for a self-adjoint operator T on an inner product space V. Actually carrying out this strategy will involve some further preparation. In particular, we will need first to review the basic facts about differential calculus in vector spaces (i.e., "multivariable calculus").

We choose an orthonormal basis $e_1, \ldots, e_n$ for V, and let T_{ij} denote the matrix of T in this basis.

We define a real-valued function F on V by $F(x) = \langle Tx, x\rangle$. F is called the *quadratic form on V defined by T*. The name comes from the following fact.

▷ **5.5—Exercise 1.** Show that if $x = x_1e_1 + \cdots + x_ne_n$, then $F(x) = \sum_{i,j=1}^n T_{ij}x_ix_j$, and using the symmetry of T_{ij} deduce that $\frac{\partial F}{\partial x_i} = 2\sum_{k=1}^n T_{ik}x_k$.

We will denote by $S(V)$ the set $\{x \in V \mid \|x\| = 1\}$, i.e., the "unit sphere" of normalized vectors. Note that if $x = x_1e_1 + \cdots + x_ne_n$, then $x \in S(V)$ if and only if $x_1^2 + \cdots + x_n^2 = 1$, so if we define $G : V \to \mathbf{R}$ by $G(x) := x_1^2 + \cdots + x_n^2 - 1$, then $S(V) = \{x \in V \mid G(x) = 0\}$.

5.5.1 Remark. Now $T_{ij} = \frac{\partial F}{\partial x_i} = 2\sum_{k=1}^n T_{ik}x_k$ is just twice the i-th component of Tx in the basis e_j. On the other hand $\frac{\partial G}{\partial x_i} = 2x_i$, which is twice the i-th component of x in this basis. It follows that a point x of $S(V)$ is an eigenvector of T provided there is a constant λ (the eigenvalue) such that $\frac{\partial F}{\partial x_i}(x) = \lambda\frac{\partial G}{\partial x_i}(x)$. If you learned about constrained extrema and Lagrange multipliers from advanced calculus, this should look familiar. Let me quote the relevant theorem,

Theorem on Constrained Extrema. *Let F and G be two differentiable real-valued functions on $\mathbf{R}^n$, and define a "constraint surface" $S \subseteq \mathbf{R}^n$ by $S = \{x \in V \mid G(x) = 0\}$. Let $p \in S$ be a point where F assumes its maximum value on S, and assume that the partial derivatives of G do not all vanish at p. Then the partial derivatives of F and of G are proportional at p, i.e., there is a constant λ (the "Lagrange Multiplier") such that $\frac{\partial F}{\partial x_i}(x) = \lambda\frac{\partial G}{\partial x_i}(x)$ for $i = 1, \ldots, n$.*

(We will sketch the proof of the Theorem on Constrained Extrema in the next lecture.)

Thus, **to find an eigenvector of T, all we have to do is choose a point of $S(V)$ where the quadratic form $F(x) = \langle Tx, x\rangle$ assumes its maximum value on $S(V)$.**

Lecture 6
Differential Calculus on Inner-product Spaces

In this section, we will use without proof standard facts that you should have seen in your multi-variable calculus classes.

6.1 Review of Basic Topology.

V will denote a finite dimensional inner-product space, and $e_1, \ldots, e_n$ some orthonormal basis for V. We will use this basis to identify $\mathbf{R}^n$ with V, via the map $(x_1, \ldots, x_n) \mapsto x_1e_1 + \cdots + x_ne_n$, and we recall that this preserves inner-products and norms, and hence distances between points. In particular, this means that a sequence $v^k = v_1^k e_1 + \ldots + v_n^k e_n$ of vectors in V converges to a vector $v = v_1e_1 + \ldots + v_ne_n$ if an only if each of the n component sequences v_i^k of real numbers converges to the corresponding real number v_i, and also, that the sequence v^k is Cauchy if and only if each component sequence of real numbers is Cauchy. This allows us to reduce questions about convergence of sequences in V to more standard questions about convergence in $\mathbf{R}$. For example, since $\mathbf{R}$ is complete (i.e., every Cauchy sequence of real numbers is convergent) it follows that V is also complete.

Recall that a subset S of $\mathbf{R}^n$ is called *compact* if any sequence of points in S has a subsequence that converges to a point of S, and the Bolzano-Weierstrass Theorem says that S is compact if and only if it is closed and bounded. (Bounded means that $\{\|s\| \mid s \in S\}$ is a bounded set of real numbers, and closed means that any limit of a sequence of points of S is itself in S.) Because of the distance preserving identification of V with $\mathbf{R}^n$ it follows that for subsets of V too, compact is the same as closed and bounded.

▷ **6.1—Exercise 1.** Recall that a set $O \subseteq V$ is called *open* if whenever $p \in O$ there is an $\epsilon > 0$ such that $\|x - p\| < \epsilon$ implies that $x \in O$. Show that O is open in V if and only if its complement is closed.

If $S \subseteq V$ and $f : S \to W$ is a map of S into some other inner-product space, then f is called *continuous* if whenever a sequence s_k in S converges to a point s of S, it follows that $f(s_k)$ converges to $f(s)$. (More succinctly, $\lim_{k\to\infty} f(s_k) = f(\lim_{k\to\infty} s_k)$, so one sometimes says a map is continuous if it "commutes with taking limits".)

▷ **6.1—Exercise 2.** Show that if $f : S \to W$ is continuous, then if A is an open (resp. closed) subset of W, then $f^{-1}(A)$ is open (resp. closed) in S. Deduce from this that $S(V)$, the unit sphere of V, is a closed and hence compact subset of V.

▷ **6.1—Exercise 3.** Show that any continuous real-valued function on a compact subset S of V must be bounded above and in fact there is a point s of S where f assumes its maximum value. (Hint # 1: If it were not bounded above, there would be a sequence s_n such that $f(s_n) > n$. Hint #2: Choose a sequence s_n so that $f(s_n)$ converges to the least upper bound of the values of f.)

6.2 Differentiable maps Between Inner-Product Spaces.

In this section, V and W are inner-product spaces.

The key concept in differential calculus is approximating a non-linear map $f : V \to W$ near some point p, by a linear map, T, called its *differential* at p. To be more precise, if v is close to zero, we should have $f(p+v) = f(p) + Tv + R_p(v)$ where the error term $R_p(v)$ should be "small" in a sense we shall make precise below.

We start with the one-dimensional case. A map $\sigma : (a,b) \to V$ of an interval (a,b) into V is called a *curve* in V, and it is said to be *differentiable* at the point $t_0 \in (a,b)$ if the limit $\sigma'(t_0) := \lim_{h\to 0} \frac{\sigma(t+h)-\sigma(t_0)}{h}$ exists. Note that this limit $\sigma'(t_0)$ is a vector in V, called the *derivative* (or the tangent vector or velocity vector) of σ at t_0. In this case, the differential of σ at t_0 is the linear map $D\sigma_{t_0} : \mathbf{R} \to V$ defined by $D\sigma_{t_0}(t) = t\sigma'(t_0)$, so that the error term is $R_p(t) = \sigma(t_0+t) - \sigma(t_0) - t\sigma'(t_0)$. Thus, not only is $R_p(t)$ small when t is small, but even when we divide it by t the result, $\frac{\sigma(t_0+t)-\sigma(t_0)}{t} - \sigma'(t_0)$, is still small and in fact it approaches zero as $t \to 0$. That is, we have $\sigma(t_0+t) = \sigma(t_0) + D\sigma_{t_0}(t) + |t|\rho(t)$ where $|\rho(t)| \to 0$ as $t \to 0$. We use this to define the notion of differentiability more generally by analogy.

6.2.1 Definition. . Let O be open in V, $F : O \to W$ a map, and $p \in O$. We say that F is *differentiable* at p if there exists a linear map $T : V \to W$ such that for v near 0 in V, $F(p+v) = F(p) + T(v) + \|v\|\,\rho(v)$ where $\rho(v) := \frac{F(p+v)-F(p)-T(v)}{\|v\|} \to 0$ as $\|v\| \to 0$. We call T the *differential of F at p* and denote it by DF_p. If $F : O \to W$ is differentiable at each point of O and if $DF : O \to L(V,W)$ is continuous we say that F is *continuously differentiable (or C^1) in O*.

6.2.2 Definition. Assuming $F : O \to W$ is as above and is differentiable at $p \in O$, then for $v \in V$, we call $DF_p(v)$ the *directional derivative of F at p in the direction v*.

▷ **6.2—Exercise 1.** Show that DF_p is well-defined, i.e., if $S : V \to W$ is a second linear map satisfying the same property as T, then necessarily $S = T$. (Hint: By subtraction one finds that for all v close to zero $\frac{\|(S-T)(v)\|}{\|v\|} \to 0$ as $\|v\| \to 0$. If one now replaces v by tv and uses the linearity of S and T, one finds that for fixed v, $\frac{\|(S-T)(v)\|}{\|v\|} \to 0$ as $t \to 0$.)

Chain Rule. *Let U, V, W be inner-product spaces, Ω an open set of U and O and open set of V. Suppose that $G : \Omega \to V$ is differentiable at $\omega \in \Omega$ and that $F : O \to W$ is differentiable at $p = G(\omega) \in O$. Then $F \circ G$ is differentiable at ω and $D(F \circ G)_\omega = DF_p \circ DG_\omega$.*

▷ **6.2—Exercise 2.** Prove the Chain Rule.

▷ **6.2—Exercise 3.** Show that if $p, v \in V$ then the map $\sigma : \mathbf{R} \to V$, $\sigma(t) = p + tv$ is differentiable at all $t_0 \in \mathbf{R}$ and that $\sigma'(t_0) = v$ for all $t_0 \in \mathbf{R}$. More generally, if $w_0 \in W$ and $T \in L(V,W)$ show that $F : V \to W$ defined by $F(v) := w_0 + Tv$ is differentiable at all v_0 in V and that $DF_{v_0} = T$. (So a linear map is its own differential at every point.)

Using the Chain Rule, we have a nice geometric interpretation of the directional derivative.

▷ **6.2—Exercise 4.** Let $F : O \to W$ be differentiable at p, let $v \in V$. Let $\sigma : \mathbf{R} \to V$ be any curve in V that is differentiable at t_0 with $\sigma(t_0) = p$ and $\sigma'(t_0) = v$. Then the curve in W, $F \circ \sigma : \mathbf{R} \to W$ is also differentiable at t_0 and its tangent vector at t_0 is $DF_p(v)$, the directional derivative of F at p in the direction v.

6.3 But Where Are All the Partial Derivatives?

Let's try to tie this up with what you learned in multi-variable calculus. As above, let us assume that O is open in V and that $F : O \to W$ is differentiable at $p \in O$. Let $e_1, \ldots, e_n$ be an orthonormal basis for V and $\epsilon_1, \ldots, \epsilon_m$ be an orthonormal basis for W, and let $p = p_1e_1 + \cdots + p_ne_n$.

If $x = x_1e_1 + \cdots + x_ne_n$ is in O, then its image $F(x) \in W$ will have an expansion in the basis ϵ_i, $F(x) = F_1(x)\epsilon_1 + \cdots + F_m(x)\epsilon_m$. If as usual we identify x with its n-tuple of components $(x_1, \ldots, x_n)$, then we have m functions of n variables, $F_1(x_1, \ldots, x_n), \ldots, F_m(x_1, \ldots, x_n)$ that describe the mapping F relative to the two bases e_i and ϵ_j.

▷ **6.3—Exercise 1.** Show that the partial derivatives of the F_i at $(p_1, \ldots, p_n)$ all exist, and in fact, show that the $\frac{\partial F_i(p_1,\ldots,p_n)}{\partial x_j}$ are the components of the directional derivative of F at p in the direction e_j relative to the basis ϵ_i.

▷ **6.3—Exercise 2.** The $n \times m$ matrix $\frac{\partial F_i(p_1,\ldots,p_n)}{\partial x_j}$ is called the *Jacobian matrix* of F at p (relative to the two bases e_j and ϵ_i). Show that it is the matrix of DF_p relative to these two bases, so that if $v = v_1e_1 + \cdots + v_ne_n$ then the i-th component of the directional derivative of F at p in the direction v is $\sum_{j=1}^{n} F_{ij}v_j$.

6.3.1 Remark. It is clear from these exercises that differentials and Jacobian matrices are logically equivalent. So which should one use? For general theoretical discussions it is usually more natural and intuitive (and easier) to reason directly about differentials of maps. However, when it comes to a question concerning a particular map F that requires computing some of its properties, then one often must work with its partial derivatives to carry out the necessary computations.

The following is a standard advanced calculus result that provides a simple test for when a map $F : O \to W$ such as above is C^1.

Theorem. *A necessary and sufficient condition for a map $F : O \to W$ as above to be C^1 is that all its partial derivatives $\frac{\partial F_i(x_1,\ldots,x_n)}{\partial x_j}$ are continuous functions on O.*

▷ **6.3—Exercise 3.** Consider the map $F : \mathbf{R}^2 \to \mathbf{R}$ defined by $F(x, y) = \frac{2xy^2}{x^2+y^2}$ for $(x, y) \neq (0, 0)$ and $F(0, 0) = 0$. Show that the partial derivatives of F exist everywhere and are continuous except at the origin. Show also that F is actually linear on each straight line through the origin, but nevertheless F is **not** differentiable at the origin. (Hint: In polar coordinates, $F = r\sin(2\theta)\cos(\theta)$.)

6.4 The Gradient of a Real-Valued Function

Let's specialize to the case $W = \mathbf{R}$, i.e., we consider a differentiable **real-valued** function $f : V \to \mathbf{R}$ (perhaps only defined on an open subset O of V). In this case it is customary to denote the differential of f at a point p by df_p rather than Df_p. Notice that df_p is in the dual space $V^* = L(V, R)$ of V. We recall the "meaning" of df_p, namely $df_p(v)$ is the directional derivative of f in the direction v, i.e., the rate of change of f along any path through p in the direction v. So if $\sigma(t)$ is a smooth curve in V with $\sigma(0) = p$ and with tangent vector $\sigma'(0) = v$ at p, then $df_p(v) = (\frac{d}{dt})_{t=0} f(\sigma(t))$.

Next recall the self-duality principle for inner-product spaces: any ℓ of V^* can be expressed in the form $\ell(v) = \langle v, \omega \rangle$ for a unique $\omega \in V$. So in particular, for each $p \in O$, there is a unique vector ∇f_p in V such that

$$df_p(v) = \langle v, \nabla f_p \rangle ,$$

and the vector ∇f_p defined by this identity is called the gradient of f at p.

A set of the form $f^{-1}(a) = \{v \in V \mid f(v) = a\}$ (where $a \in \mathbf{R}$) is called a *level set* of f and more precisely the a-level of f.

▷ **6.4—Exercise 1.** Show that if the image of a differentiable curve $\sigma : (a, b) \to V$ is in a level set of f then $\nabla f_{\sigma(t)}$ is orthogonal to $\sigma'(t)$. (Hint: The derivative of $f(\sigma(t))$ is zero.)

A point p where df (or ∇f) vanishes is called a *critical point* of f. So for example any local maximum or local minimum of f is a critical point.

▷ **6.4—Exercise 2.** If p is not a critical point of f then show that $\frac{\nabla f_p}{\|\nabla f_p\|}$ is the unit vector in the direction in which f is increasing most rapidly, and that $\|\nabla f_p\|$ is the magnitude of this rate of increase. (Hint: Use the Schwartz inequality.)

▷ **6.4—Exercise 3.** Suppose that $\sigma : (a, b) \to V$ and $\gamma : (a, b) \to V$ are differentiable curves. Show that $\frac{d}{dt} \langle \sigma(t), \gamma(t) \rangle = \langle \sigma'(t), \gamma(t) \rangle + \langle \sigma(t), \gamma'(t) \rangle$, and in particular $\frac{d}{dt} \|\sigma(t)\|^2 = 2 \langle \sigma(t), \sigma'(t) \rangle$. Deduce that if $\sigma : (a, b) \to V$ has its image in the unit sphere $S(V)$ then $\sigma(t)$ and $\sigma'(t)$ are orthogonal.

▷ **6.4—Exercise 4.** For $p \in S(V)$, define $T_pS(V)$, the tangent space to $S(V)$ at p, to be all $v \in V$ of the form $\sigma'(t_0)$ where $\sigma(t)$ is a differentiable curve in $S(V)$ having $\sigma(t_0) = p$. By the previous exercise, p is orthogonal to everything in $T_pS(V)$. Show that conversely any v orthogonal to p is in $T_pS(V)$, so the tangent space to $S(V)$ at p is exactly the orthogonal complement of p. (Hint: Define $\sigma(t) := \cos(\|v\| t)p + \sin(\|v\| t)\frac{v}{\|v\|}$. Check that $\sigma(0) = p$, $\sigma'(t) = v$, and that $\sigma(t) \in S(V)$ for all t.

▷ **6.4—Exercise 5.** Let T be a self-adjoint operator on V and define $f : V \to \mathbf{R}$ by $f(v) := \frac{1}{2} \langle Tv, v \rangle$. Show that f is differentiable and that $\nabla f_v = Tv$.

Proof of Spectral Lemma. We must find a p in $S(V)$ that is an eigenvector of the self-adjoint operator $T : V \to V$. By Exercise 4 we must show that ∇f_p is a multiple of p. But by Exercise 4, the scalar multiples of p are just those vectors orthogonal to $T_pS(V)$, so it will suffice to find p witht ∇f_p orthogonal to $T_pS(V)$. If we choose p a point of $S(V)$ where f assumes its maximum value on $S(V)$, that is automatic.

Lecture 7
Normed Spaces and Integration

7.1 Norms for Vector Spaces.

Recall that a metric space is a very simple mathematical structure. It is given by a set X together with a distance function for X, which is a mapping ρ from $X \times X \to \mathbf{R}$ that satisfies three properties:

- Positivity: $\rho(x_1, x_2) \geq 0$ with equality if and only if $x_1 = x_2$.
- Symmetry: $\rho(x_1, x_2) = \rho(x_2, x_1)$.
- Triangle Inequality: $\rho(x_1, x_3) \leq \rho(x_1, x_2) + \rho(x_2, x_3)$.

(You should think of the triangle inequality as saying that "things close to the same thing are close to each other".)

For the metric spaces that turn up in practice, X is usually some subset of a vector space V and the distance function ρ has the form $\rho(x_1, x_2) = N(x_1 - x_2)$ where the function $N : V \to \mathbf{R}$ is what is called a "norm" for V,

7.1.1 Definition. A real-valued function on a vector space V is called a *norm* for V if it satisfies the following three properties:

- Positivity: $N(v) \geq 0$ with equality if and only if $v = 0$.
- Positive Homogeneity: $N(\alpha v) = |\alpha| N(v)$.
- Triangle Inequality: $N(x_1 + x_2) \leq N(x_1) + N(x_2)$.

If N is a norm for V then we call $\rho_{_N}(x_1, x_2) := N(x_1 - x_2)$ the *associated distance function (or metric) for* V. A vector space V together with some a choice of norm is called a *normed space*, and the norm is usually denoted by $\| \ \|$. If V is complete in the associated metric (i.e., every Cauchy sequence converges), then V is called a *Banach spacee*.

▷ **7.1—Exercise 1.** Show that if N is a norm for V then $\rho_{_N}$ really defines a distance function for V.

7.1.2 Remark. We have seen that for an inner-product space V, we can define a norm for V by $\|v\| := \sqrt{\langle v, v \rangle}$. Moreover, we could then recover the inner-product from this norm by using the so-called polarization identity: $\|x + y\|^2 = \|x\|^2 + \|y\|^2 + 2\langle x, y \rangle$. It is natural to wonder if every norm "comes from an inner product" in this way, or if not which ones do. The answer is quite interesting. If we replace y by $-y$ in the polarization identity and then add the result to the original polarization identity, we get $\|x + y\|^2 + \|x - y\|^2 = 2\|x\|^2 + 2\|y\|^2$. This has a nice interpretation: it says that the sum of the squares of the two diagonals of any parallelogram is equal to the sum of the squares of the four sides—and so it is called the *Parallelogram Law*. It is a pretty (and non-obvious!) fact that the Parallelogram Law is not only a necessary but also a sufficient condition for a norm to come from an inner product.

There is a remarkable fact about linear maps between normed spaces: if they have even a hint of continuity then they are very strongly continuous. To be precise:

7.1.3 Theorem. *Let V and W be normed spaces and $T : V \to W$ a linear map from V to W. Then the following are equivalent:*

1) There is at least one point $v_0 \in V$ where T is continuous.

2) T is continuous at 0.

3) There is a constant K such that $\|Tv\| \leq K\|v\|$ for all $v \in V$.

4) T satisfies a Lipschitz condition.

5) T is uniformly continuous.

PROOF. First note that since $\|(x_n + y) - (x + y)\| = \|x_n - x\|$ it follows that $x_n \to x$ is equivalent to $(x_n + y) \to (x + y)$, a fact we will use several times

Assume 1) and let $v_n \to 0$. Then $(v_n + v_0) \to v_0$, so by 1) $T(v_n + v_0) \to Tv_0$, or by linearity, $Tv_n + Tv_0 \to Tv_0$; which is equivalent to $Tv_n \to 0$, hence 1) implies 2). Assuming 2), we can find a $\delta > 0$ such that if $\|x\| \leq \delta$ then $\|T(x)\| < 1$. Now $\left\| \frac{\delta v}{\|v\|} \right\| = \delta$, so $\left\| T\left(\frac{\delta v}{\|v\|} \right) \right\| < 1$, hence $\|Tv\| < K\|v\|$ where $K = \frac{1}{\delta}$, so 2) implies 3). Since $Tv_1 - Tv_2 = T(v_1 - v_2)$, 3) implies that K is a Lipschitz constant for T, and finally 4) $\Rightarrow$ 5) $\Rightarrow$ 1) is trivial.

▷ **7.1—Exercise 2.** Show that if V and W are finite dimensional inner-product spaces then any linear map $T : V \to W$ is automatically continuous. Hint: Choose bases, and look at the matrix representation of T.

7.1.4 Remark. This theorem shows that if $T : V \to W$ is continuous, then T is bounded on the unit sphere of V. Thus $|||\, T \,||| := \sup_{\|v\|=1} \|Tv\|$ is well-defined and is called the norm of the linear map T. It is clearly the smallest Lipschitz constant for T.

7.1.5 Extension Theorem For Continuous Linear Maps. *Let V be a normed space, U a linear subspace of V, W a Banach space, and $T : U \to W$ a continuous linear map. Then T has a unique extension to a linear map $\tilde{T} : \bar{U} \to W$ (where as usual $\bar{U}$ denotes the closure of U in V). Moreover $|||\, \tilde{T} \,||| = |||\, T \,|||$.*

PROOF. Suppose $v \in \bar{U}$. Then there is a sequence $u_n \in U$ such that $u_n \to v$. Then u_n is a Cauchy sequence, and since T is Lipschitz it follows that Tu_n is Cauchy in W, and since W is by assumption a Banach space (complete) it follows that Tu_n converges to a limit $w \in W$, and we define $\tilde{T}v := w$. We leave further details as an exercise. ∎

▷ **7.1—Exercise 3.** If u'_n is a second sequence in U that converges to v show that Tu'_n has the same limit as Tu_n. (Hint: Consider the sequence $u_1, u'_1, u_2, u'_2, \ldots$.)

▷ **7.1—Exercise 4.** Why does $|||\, \tilde{T} \,||| = |||\, T \,|||$?

7.2 The Space $B([a,b],V)$ of Bounded Functions on V.

7.2.1 Definition. Let V be a finite dimensional inner-product space and $[a,b] \subseteq \mathbf{R}$ a closed interval of real numbers. We denote by $B([a,b],V)$ the vector space of all bounded functions $\sigma : [a,b] \to V$, with pointwise vector operations, and we define a function $\sigma \mapsto \|\sigma\|_\infty$ on $B([a,b],V)$ by $\|\sigma\|_\infty = \sup_{a\le t\le b} \|\sigma(t)\|$.

▷ **7.2—Exercise 1.** Show that $\| \ \|_\infty$ really defines a norm for $B([a,b],V)$.

Henceforth we shall always regard $B([a,b],V)$ as a metric space with the distance function defined by the norm $\| \ \|_\infty$. But what does convergence mean in this metric space?

▷ **7.2—Exercise 2.** Let $\sigma_n : [a,b] \to V$ be a sequence in $B([a,b],V)$ and $\sigma \in B([a,b],V)$. To say that the sequence σ_n converges to σ of course means by definition that the sequence of real numbers $\|\sigma_n - \sigma\|_\infty$ converges to zero. Show that this is the case if and only if the sequence of functions $\sigma_n(t)$ converges **uniformly** to the function $\sigma(t)$ on the interval $[a,b]$, i.e., if and only if for every $\epsilon > 0$ there is a positive integer N such that the inequality $\|\sigma_n(t) - \sigma(t)\|$ holds **for all** $t \in [a,b]$ provided $n > N$.

7.2.2 Remark. A similar proof shows that the sequence σ_n is Cauchy in $B([a,b],V)$ if and only if the sequence of functions $\sigma_n(t)$ is uniformly Cauchy on $[a,b]$, i.e., if and only if for every $\epsilon > 0$ there is a positive integer N such that the inequality $\|\sigma_n(t) - \sigma_m(t)\|$ holds for all $t \in [a,b]$ provided both m and n are greater than N. Now if $\sigma_n(t)$ is uniformly Cauchy on $[a,b]$, then *a fortiori* σ_n is a Cauchy sequence in V for each $t \in [a,b]$, and since V is complete, $\sigma_n(t)$ converges to some element $\sigma(t)$ in V, and then it is easy to see that σ is in $B([a,b],V)$ and that $\sigma - \sigma_n \to 0$. This proves that $B([a,b],V)$ is a Banach space, i.e., any Cauchy sequence in $B([a,b],V)$ converges to an element of $B([a,b],V)$.

7.3 Quick and Dirty Integration

We will now see that it is quite easy to define the integral, $\int_a^b f(t)\,dt$, of a continuous map $f : [a,b] \to V$, and in fact we shall define it for a class of f considerably more general than continuous. In all of the following we assume that V is a finite dimensional inner-product space.

7.3.1 Definition. A *partition* Π of $[a,b]$ is a finite sequence $a = t_0 \le t_1 \le \ldots \le t_n = b$, and we say that a function $f : [a,b] \to V$ is *adjusted* to the partition Π if f is constant on each of the sub-intervals (t_i, t_{i+1}). We call f a *step-function* if there exists a partition to which it is adjusted, and we denote by $\mathcal{S}([a,b],V)$ the set of all step functions. Clearly $\mathcal{S}([a,b],V) \subseteq B([a,b],V)$. If f_1 and f_2 are two step-functions, and Π_i is a partition adjusted to f_i, then any partition that contains all the points of Π_1 and Π_2 is adjusted to both f_1 and f_2 and hence to any linear combination of them. This shows that $\mathcal{S}([a,b],V)$ **is a linear subspace of** $B([a,b],V)$.

7.3.2 Definition. If $f : [a,b] \to V$ is a step-function and if f is adjusted to a partition $\Pi = t_0 \le t_1 \le \ldots \le t_n$, then we define its *integral*, $\mathcal{I}(f) \in V$, by $\mathcal{I}(f) := \sum_{i=1}^n (t_i - t_{i-1})v_i$, where v_i is the constant value of f on the interval (t_{i-1}, t_i).

▷ **7.3—Exercise 1.** Show that the integral $\mathcal{I} : \mathcal{S}([a,b],V) \to V$ is a well-defined linear map and satisfies $\|\mathcal{I}(f)\| \leq (b-a)\,\|f\|_\infty$, so that $\mathcal{I}$ is continuous and has norm $|||\,\mathcal{I}\,||| = (b-a)$. Hint: The only (slightly) tricky point is to show that $\mathcal{I}(f)$ does not depend on the choice of a partition to which f is adjusted. Reduce this to showing that when you subdivide one subinterval of a partition, the integral does not change.

We can now use our Extension Theorem For Continuous Linear Maps to conclude that $\mathcal{I}$ has a unique extension to a continuous linear map $\mathcal{I} : \bar{\mathcal{S}}([a,b],V) \to V$, where $\bar{\mathcal{S}}([a,b],V)$ denotes the closure of the step functions in the space $B([a,b],V)$ of bounded functions. For $f \in \bar{\mathcal{S}}([a,b],V)$ we will also denote its integral, $\mathcal{I}(f)$, by $\int_a^b f(t)\,dt$, and we will refer to elements of $\bar{\mathcal{S}}([a,b],V)$ as *integrable* functions. Note that according to the extension theorem, the inequality $\left\|\int_a^b f(t)\,dt\right\| \leq (b-a)\,\|f\|_\infty$ continues to hold for any integrable function f.

7.3.3 Proposition. *Integration commutes with linear maps. That is, if $T : V \to W$ is linear and $f : [a,b] \to V$ is integrable, then $T \circ f : [a,b] \to W$ is integrable and $T(\int_a^b f(t)\,dt) = \int_a^b T(f(t))\,dt$.*

PROOF. The Proposition is obvious for step functions, and it then follows for integrable functions by the uniqueness of the extension.

It is natural to wonder just how inclusive the space of integrable functions is. We note next that it contains all continuous functions.

7.3.4 Proposition. *The space $C([a,b],V)$ of continuous maps of $[a,b]$ into V is a linear subspace of the space $\bar{\mathcal{S}}([a,b],V)$ of integrable functions.*

PROOF. By a standard result of analysis, if $f \in C([a,b],V)$ then f is uniformly continuous. That is, given $\epsilon > 0$, there is a $\delta > 0$ such that if t_1 and t_2 are in $[a,b]$ and $|t_1 - t_2| < \delta$, then $\|f(t_1) - f(t_2)\| < \epsilon$. Choose an integer N so large that $\frac{b-a}{N} < \delta$ and partition $[a,b]$ into N equal subintervals, and let ϕ be the step function that is constant on each subinterval of this partition and agrees with f at the left endpoint. Then clearly $\|f - \phi\|_\infty < \epsilon$, proving that f is in the closure of $\mathcal{S}([a,b],V)$ ∎

▷ **7.3—Exercise 2.** The "standard result of analysis" says that, if X is a compact metric space then a continuous map $f : X \to Y$ is uniformly continuous. Prove this. Hint: One approach is to prove the "contrapositive" statement, i.e., show that if X is **not** uniformly continuous then there is at least one point x of X where f is not continuous. For this, pick an ϵ so that for each integer n there are two points x_n and x'_n with $\rho(x_n, x'_n) < \frac{1}{n}$ but $\rho(f(x_n), f(x'_n)) > \epsilon$. By compactness, a subsequence x_{n_k} converges to some point x. Show that x'_{n_k} also converges to x and deduce that f is not continuous at x.

7.3.5 Remark. The above proof generalizes to show that even a piecewise continuous $f : [a,b] \to V$ is integrable, where $f : [a,b] \to V$ is called piecewise continuous if there is a

partition of $[a, b]$ such that f is continuous on each of its open subintervals, with a limit at each endpoint of the subinterval.

▷ **7.3—Exercise 3.** Show that if $f : [a, b] \to V$ is integrable then $\left\| \int_a^b f(t)\, dt \right\| \leq \int_a^b \|f(t)\|\, dt$. Hint: For a step function, this follows from the triangle inequality for norms.

▷ **7.3—Exercise 4.** . Show that if $f : [a, b] \to V$ is integrable, and $a < c < b$, then $\int_a^b f(t)\, dt = \int_a^c f(t)\, dt + \int_c^b f(t)\, dt$

▷ **7.3—Exercise 5.** Let $f : [a, b] \to V$ be continuous and define $F : [a, b] \to V$ by $F(t) := \int_a^t f(s)\, ds$. Show that F is differentiable and that $F' = f$.
Hint: $\frac{1}{h}[F(t_0 + h) - F(t_0)] - f(t_0) = \frac{1}{h} \int_{t_0}^{t_0+h} (f(s) - f(t_0))\, ds$.

We next prove the vector version of the Fundamental Theorem of Integral Calculus (allowing us to evaluate a definite integral if we know an anti-derivative for the integrand). As in the classic case it depends on knowing that only constant functions have a zero derivative.

7.3.6 Lemma. *If $f : [a, b] \to V$ is differentiable and f' is identically zero, then f is constant.*

PROOF. This can be reduced to the classic special case that $V = \mathbf{R}$ (proved in elementary Calculus as an easy consequence of Rolle's Theorem). In fact, if $\ell \in V^*$, then $(\ell \circ f)' = \ell \circ f' = 0$ so $(\ell \circ f)$ is constant by the special case. But since this holds for *every* $\ell \in V^*$ it follows easily that f itself must be constant. ∎

▷ **7.3—Exercise 6.** Prove the vector version of the Fundamental Theorem of Integral Calculus, That is, let $f : [a, b] \to V$ be continuous and let $\Phi : [a, b] \to V$ be an antiderivative of f, i.e., Φ is differentiable and $\Phi' = f$. Show that $\int_a^b f(t)\, dt = \Phi(b) - \Phi(a)$. Hint: If we define $F(t) := \int_a^t f(s)\, ds$, then we know F and Φ have the same derivative, so $F - \Phi$ has derivative 0, and by the lemma F and Φ differ by a constant vector.

7.3.7 Finite Difference Formula. *Let O be a convex open set in V and let $F : O \to W$ be differentiable. If $v_0, v_1 \in O$, then $F(v_1) - F(v_0) = \int_0^1 DF_{\sigma(t)}(v_1 - v_0)\, dt$, where $\sigma(t) = v_0 + t(v_1 - v_0)$, $0 \leq t \leq 1$ is the line segment joining v_0 to v_1.*

PROOF. By the chain rule, $(F \circ \sigma)'(t) = DF_{\sigma(t)}(\sigma'(t))$, and clearly $\sigma'(t) = v_1 - v_0$. Thus $(F \circ \sigma)$ is an anti-derivative for $DF_{\sigma(t)}(v_1 - v_0)$, and the result follows from the Fundamental Theorem.

7.3.8 Corollary. *Let O be a convex open set in V and let $F : O \to W$ be continuously differentiable. Then F satisfies a Lipschitz condition on any closed, bounded subset S of O. In fact, if K is an upper bound for $|||\, DF_p\, |||$ for $p \in S$, then K is a Lipschitz bound for F.*

▷ **7.3—Exercise 7.** Use the Finite Difference Formula to prove the corollary.

7.4 Numerical Integration (or Quadrature Rules)

Since one usually cannot find an anti-derivative for an integrand in closed form, it is important to be able to "evaluate an integral numerically"—meaning approximate it with arbitrary precision. In fact, this is so important that there are whole books devoted the study of numerical integration methods (aka quadrature rules). We will consider only two such methods, one known as the Trapezoidal Rule and the other as Simpson's Rule. In what follows, we will assume that the integrand f is always at least continuous, but for the error estimates that we will mention to be valid, we will need f to have several continuous derivatives.

7.4.1 Definition. By a *quadrature rule* we mean a function M that assigns to each continuous function $f : [a, b] \to V$ (mapping a closed interval $[a, b]$ into an inner-product space V) a vector $M(f, a, b) \in V$—which is supposed to be an approximation of the integral, $\int_a^b f(t)\,dt$. A particular quadrature rule M is usually given by specifying a linear combination of the values of f at certain points of he interval $[a, b]$; that is, it has the general form $M(f, a, b) := \sum_{i=1}^{n} w_i f(t_i)$, where the points $t_i \in [a, b]$ are called the *nodes* of M and the scalars w_i are called its *weights*. The *error* of M for a particular f and $[a, b]$ is defined as $\mathrm{Err}(M, f, a, b) := \left\| \int_a^b f(t)\,dt - M(f, a, b) \right\|$.

7.4—Example 1. The Trapezoidal Rule: $M^T(f, a, b) := \frac{b-a}{2}[f(a) + f(b)]$.
In this case, there are two nodes, namely the two endpoints of the interval, and they have equal weights, namely half the length of the interval. Later we shall see the origin of this rule (and explain its name).

7.4—Example 2. Simpson's Rule: $M^S(f, a, b) := \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$.
So now the nodes are the two endpoints, as before, and in addition the midpoint of the interval. And the weights are $\frac{b-a}{6}$ for the two endpoints and $\frac{2(b-a)}{3}$ for the midpoint.

7.4.2 Remark. Notice that in both examples the weights add up to $b - a$. This is no accident; any "reasonable" quadrature rule should have a zero error for a constant function, and this easily implies that the weights must add to $b - a$.

7.4.3 Proposition. *If $f : [a, b] \to V$ has two continuous derivatives, and $\|f''(t)\| < C$ for all $t \in [a, b]$ then $\mathrm{Err}(M^T, f, a, b) \le C\frac{(b-a)^3}{12}$. Similarly, if $f : [a, b] \to V$ has four continuous derivatives, and $\|f''''(t)\| < C$ for all $t \in [a, b]$ then $\mathrm{Err}(M^S, f, a, b) \le C\frac{(b-a)^5}{90}$*

7.4.4 Remark. The proof of this proposition is not difficult—it depends only the Mean Value Theorem—but it can be found in any numerical analysis text and will not be repeated here.

7.4.5 Definition. If M is a quadrature rule then we define a sequence M_n of *derived* quadrature rules by $M_n(f, a, b) := \sum_{i=0}^{n-1} M(f, a + ih, a + (i + 1)h)$ where $h = \frac{b-a}{n}$. We say that the rule M is *convergent* for f on $[a, b]$ if the sequence $M_n(f, a, b)$ converges to $\int_a^b f(t)\,dt$.

In other words, to estimate the integral $\int_a^b f(t)\,dt$ using the n-th derived rule M_n, we simply divide the interval $[a,b]$ of integration into n equal sub-intervals, estimate the integral on each sub-interval using M, and then add these estimates to get the estimate of the integral on the whole interval.

7.4.6 Remark. We next note an interesting relation between the errors of M and of M_n. Namely, with the notation just used in the above definition, we see that by the additivity of the integral, $\int_a^b f(t)\,dt = \sum_{i=0}^{n-1} \int_{a+ih}^{a+(i+1)h} f(t)\,dt$, hence from the definition of M_n and the triangle inequality, we have $\mathrm{Err}(M_n, f, a, b) \le \sum_{i=0}^{n-1} \mathrm{Err}(M, f, a+ih, a+(i+1)h)$. We can now use this together with Proposition 7.4.3 to prove the following important result:

7.4.7 Theorem. *If $f : [a,b] \to V$ has two continuous derivatives, and $\|f''(t)\| < C$ for all $t \in [a,b]$ then $\mathrm{Err}(M_n^T, f, a, b) \le C\frac{(b-a)^3}{12n^2}$. Similarly, if $f : [a,b] \to V$ has four continuous derivatives, and $\|f''''(t)\| < C$ for all $t \in [a,b]$ then $\mathrm{Err}(M_n^S, f, a, b) \le C\frac{(b-a)^5}{90n^4}$*

▷ **7.4—Exercise 1.** Fill in the details of the proof of this theorem.

7.4.8 Remark. This shows that both the Trapezoidal Rule and Simpson's Rule are convergent for any reasonably smooth function. But it also shows that Simpson's Rule is far superior to the Trapezoidal Rule. For just fifty per cent more "effort" (measured by the number of evaluations of f) one gets a far more accurate result.

Where did the formulas for the Trapezoidal Rule and Simpson's Rule come from? It helps to think of the classical case of a real-valued function f, so we can regard $\int_a^b f(t)\,dt$ as representing the area under the graph of f between a and b. Now, if f is differentiable and the interval $[a,b]$ is short, then the graph of f is well-approximated by the straight line segment joining $(a,(f(a))$ to $(b,f(b))$, so the area under of the graph of f should be well-approximated by the area between the x-axis and this line segment. The latter area is of course a trapezoid (whence the name) and it has the area given by the Trapezoidal Rule formula. Simpson's Rule arises if instead of interpolating f by a linear function that agrees with f at a and b we instead interpolate by a quadratic function cx^2+dx+e that agrees with f at a and b and also at the mid-point $\frac{a+b}{2}$.

▷ **7.4—Exercise 2.** Using the method of "undetermined coefficients", show that there is a unique choice of coefficients c, d, e such that the quadratic polynomial cx^2+dx+e agrees with the function f at the three points a, b and $\frac{a+b}{2}$. Find c, d, and e explicitly and integrate the polynomial from a to b, and check that this gives $M^S(f,a,b)$.

7.5 Second Matlab Project.

The second Matlab project is to develop Matlab code to implement the Trapezoidal Rule and Simpson's Rule, and then to do some experimentation with your software, checking that the error estimates of theorem 7.4.7 are satisfied for some test cases where the function f has a known anti-derivative and so can be evaluated exactly. In more detail:

1) Write a Matlab function M-file defining a function TrapezoidalRule(f,a,b,n). This should return the value of $M_n^T(f,a,b)$. Here of course the parameters a and b represent real numbers and the parameter n a positive integer. But what about the parameter f, i.e., what should it be legal to substitute for f when the TrapezoidalRule(f,a,b,n) is called? Answer: f should represent a function of a real variable whose values are arrays (of some fixed size) of real numbers. The function that you are permitted to substitute for f should either be a built-in Matlab function (such as sin) or an inline function in the Matlab Workspace, or a function that is defined in some other M-File.

2) Write a second Matlab function M-file defining a function SimpsonsRule(f,a,b,n) that returns $M_n^S(f,a,b)$.

3) Recall that $\int_0^t \frac{dx}{1+x^2} = \arctan(t)$, so that in particular $\int_0^1 \frac{4\,dx}{1+x^2} = 4\arctan(1) = \pi$. Using the error estimates for the Trapezoidal Rule and Simpson's Rule, calculate how large n should be to calculate π correct to d decimal places from this formula using Trapezoidal and Simpson. Set format long in Matlab and get the value of π to fifteen decimal places by simply typing pi. Then use your Trapezoidal and Simpson functions from parts 1) and 2) to see how large you actually have to choose n to calculate π to 5, 10, and 15 decimal places.

4) Be prepared to discuss your solutions in the Computer Lab.

Lecture 8
Ordinary Differential Equations (aka ODE)

8.1 The Initial Value Problem for an ODE.

Suppose we know the wind velocity at every point of space and at every instant of time. A puff of smoke drifts by, and at a certain moment we observe the precise location of a particular smoke particle. Can we then predict where that particle will be at all future times? By making this metaphorical question precise we will be led to the concept of an initial value problem for an ordinary differential equation.

We will interpret "space" to mean $\mathbf{R}^n$, or more generally an inner-product space V, and an "instant of time" will be represented by a real number t. Thus, knowing the wind velocity at every point of space and at all instants of time means that we have a function $X : V \times \mathbf{R} \to V$ that associates to each (v, t) in $V \times \mathbf{R}$ a vector $X(v, t)$ in V representing the wind velocity at v at time t. Such a mapping is called a *time-dependent vector field on* V. We will always be working with such X that are at least continuous, and usually X will even be continuously differentiable. In case $X(v, t)$ does not actually depend on t then we call X a *time-independent vector field on* V, or simply a vector field on V. Note that this is the same as giving a map $X : V \to V$.

How should we model the path taken by the smoke particle? An ideal smoke particle is characterized by the fact that it "goes with the flow", i.e., it is carried along by the wind, meaning that if $x(t)$ is its location at a time t, then its velocity at time t will be the wind velocity at that point and time, namely $X(x(t), t)$. But the velocity of the particle at time t is $x'(t) = \frac{dx}{dt}$, so the path of a smoke particle will be a differentiable curve $x : (a, b) \to V$ such that $x'(t) = X(x(t), t)$ for all $t \in (a, b)$. Such a curve is called a *solution curve* of the time-dependent vector field X and we also say that"x satisfies the ordinary differential equation $\frac{dx}{dt} = X(x, t)$".

Usually we will be interested in solution curves of a differential equation $\frac{dx}{dt} = X(x, t)$ that satisfy a particular *initial condition*. This means that we have singled out some special time t_0 (often, $t_0 = 0$), and some specific point $v_0 \in V$, and we look for a solution of the ODE that satisfies $x(t_0) = v_0$. (In our smoke particle metaphor, this corresponds to observing the particle at v_0 as our clock reads t_0.) The pair of equations $\frac{dx}{dt} = X(x, t)$ and $x(t_0) = v_0$ is called an "initial value problem" (abbreviated as IVP) for the ODE $\frac{dx}{dt} = X$. The reason that it is so important is that the so-called Existence and Uniqueness Theorem for ODE says (more or less) that, under reasonable assumptions on X, the initial value problem has a "unique" solution.

8.1.1 Remark. If the vector field X is time-independent, then the ODE $\frac{dx}{dt} = X$ is often called *autonomous*.

8.1.2 Remark. In the case $V = \mathbf{R}^n$, $X(x, t) = (X_1(x, t), \ldots, X_n(x, t))$ so that written out in full, the ODE $\frac{dx}{dt} = X$ looks like $\frac{dx_i}{dt} = X_i(x_1(t), \ldots, x_n(t), t)$, $i = 1, \ldots n$. In this form it is usually referred to as a "system of ordinary differential equations".

8.1—Example 1. X a constant vector field, i. e., $X(v,t) = u$, where u is some fixed element of V. The solution with initial condition $x(t_0) = v_0$ is clearly the straight line $x(t) := v_0 + (t - t_0)u$.

8.1—Example 2. X is the "identity" vector field, $X(v,t) = v$. The solution with initial condition $x(t_0) = v_0$ is clearly $x(t) := e^{(t-t_0)}v_0$. (Later we will see how to generalize this to an arbitrary linear vector field, i.e., one of the form $X(v,t) = Tv$ where $T : V \to V$ is a continuous linear map.)

8.1—Example 3. A vector field that is "space-independent", i. e., $X(v,t) = f(t)$ where $f : \mathbf{R} \to V$ is continuous. The solution with initial condition $x(t_0) = v_0$ is $x(t) = v_0 + \int_{t_o}^{t} f(s)\,ds$.

8.2 The Local Existence and Uniqueness Theorem.

In what follows, $X : V \times \mathbf{R} \to V$ is a time dependent vector field on V, $v_0 \in V$, $I = [a,b]$ is an interval of real numbers, and $t_0 \in I$. We define a map $F = F^X_{t_0,v_0}$ of $C(I,V)$ to itself by: $F(\sigma)(t) := v_0 + \int_{t_0}^{t} X(\sigma(s),s)\,ds$.

The following proposition follows immediately from the definitions and the Fundamental Theorem of Integral Calculus.

Proposition. *A necessary and sufficient condition for $\sigma : I \to V$ to be a solution of the initial value problem $\frac{dx}{dt} = X(x,t)$ and $x(t_0) = v_0$ is that σ be a fixed point of $F^X_{t_0,v_0}$.*

This immediately suggests using successive approximations as a strategy for solving the initial value problem. Start say with the constant curve $x_0 : I \to V$ given by $x_0(t) = v_0$, and define $x_n : I \to V$ inductively by $x_{n+1} := F^X_{t_0,v_0}(x_n)$, and attempt to show that x_n converges to a fixed point of $F^X_{t_0,v_0}$, perhaps by showing that the Contraction Principle applies. As we shall now see, this simple idea actually works for a very general class of ODE.

▷ **8.2—Exercise 1.** Carry out the above strategy for the case of the time-independent vector field $X(v) := v$ with $t_0 = 0$. We saw above that the solution in this case is $x(t) = e^t v_0$. Show by induction that $x_n(t) = P_n(t)v_0$, where P_n is the n-th order Taylor polynomial for e^t, i.e., $P_n(t) = \sum_{k=0}^{n} \frac{t^k}{k!}$.

Local Existence and Uniqueness Theorem For ODE. *Let $X : V \times \mathbf{R} \to V$ be a C^1 time-dependent vector field on V, $p \in V$, and $t_0 \in \mathbf{R}$. There are positive constants ϵ and δ depending on X, p, and t_0 such that if $I = [t_0 - \delta, t_0 + \delta]$, then for each $v_0 \in V$ with $\|v_0 - p\| < \epsilon$ the differential equation $\sigma'(t) = X(\sigma(t),t)$ has a unique solution $\sigma : I \to V$ satisfying $\sigma(t_0) = v_0$.*

PROOF. If $\epsilon > 0$, then using the technique explained earlier we can find a Lipschitz constant M for X restricted to the set of $(x,t) \in V \times \mathbf{R}$ such that $\|x - p\| \leq 2\epsilon$ and $|t - t_0| \leq \epsilon$. Let B be the maximum value of $\|X(x,t)\|$ on this same set, and choose $\delta > 0$ so that $K = M\delta < 1$ and $B\delta < \epsilon$, and define Y to be the set of σ in $C(I,V)$ such that

$\|\sigma(t) - p\| \leq 2\epsilon$ for all $|t| \leq \delta$. It is easy to see that Y is closed in $C(I, V)$, hence a complete metric space. The theorem will follow from the Banach Contraction Principle if we can show that for $\|v_0\| < \epsilon$, $F = F^X_{t_0,v_0}$ maps Y to itself and has K as a Lipschitz bound.

If $\sigma \in Y$ then $\|F(\sigma)(t) - p\| \leq \|v_0 - p\| + \int_0^t \|X(\sigma(s), s)\|\, ds \leq \epsilon + \delta B \leq 2\epsilon$, so F maps Y to itself. And if $\sigma_1, \sigma_2 \in X$ then $\|X(\sigma_1(t), t) - X(\sigma_2(t), t)\| \leq M \|\sigma_1(t) - \sigma_2(t)\|$, so

$$\begin{aligned}\|F(\sigma_1)(t) - F(\sigma_2)(t)\| &\leq \int_0^t \|X(\sigma_1(s), s) - X(\sigma_2(s), s)\|\, ds \\ &\leq \int_0^t M \|\sigma_1(s) - \sigma_2(s)\|\, ds \\ &\leq \int_0^t M\rho(\sigma_1, \sigma_2)\, ds \\ &\leq \delta M \rho(\sigma_1, \sigma_2) \leq K\rho(\sigma_1, \sigma_2).\end{aligned}$$

and it follows that $\rho(F(\sigma_1), F(\sigma_2) \leq K\rho(\sigma_1, \sigma_2)$. ∎

▷ **8.2—Exercise 2.** Show that continuity of V is **not** sufficient to guarantee uniqueness for an IVP. Hint: The classic example (with $V = \mathbf{R}$) is the initial value problem $\frac{dx}{dt} = \sqrt{x}$, and $x(0) = 0$. (Note that this is C^1, except at the point $x = 0$.) Show that for each $T > 0$, we get a distinct solution $x_{_T}(t)$ of this IVP by defining $x_{_T}(t) = 0$ for $t < T$ and $x_{_T}(t) = \frac{1}{4}(t - T)^2$ for $t \geq T$.

8.2.1 Remark. The existence and uniqueness theorems tell us that for a given initial condition we can solve our initial value problem (uniquely) for a short time interval. The next question we will take up is for just how long we can "follow a smoke particle". You might guess for each initial condition p in V we should have a solution $x_p : \mathbf{R} \to V$ with $x_p(t_0) = p$. But such global in time existence is too much to expect in general. For example, take $V = \mathbf{R}$ and consider the differential equation $\frac{dx}{dt} = x^2$ with the initial condition $x(0) = x_0$. An easy calculation shows that the unique solution is $x(t) = \frac{x_0}{1 - tx_0}$. Note that x_0, this solution "blows up" at time $T = \frac{1}{x_0}$, and by the uniqueness theorem, no solution can exist for a time greater than T.

You may object that a particle of smoke will never go off to infinity in a finite amount of time! Perhaps the smoke metaphor isn't so good after all. The answer is that a real, physical wind field has bounded velocity, and it isn't hard to show that in this case we do indeed have global in time existence.

We are now going to make a simplification, and restrict attention to time-**independent** vector fields (which we shall simply call vector fields). That may sound like a tremendous loss of generality, but in fact it is no loss of generality at all!

▷ **8.2—Exercise 3.** Let $X(v, t)$ be a time-dependent vector field in V, and define an associated time independent vector field $\tilde{X}$ in $V \times \mathbf{R}$ by $\tilde{X}(y) = (X(y), 1)$. Show that $y(t) = (x(t), f(t))$ is a solution of the differential equation $\frac{dy}{dt} = \tilde{X}(y)$ if and only if $f(t) = t + c$ and $x(t)$ is a solution of $\frac{dx}{dt} = X(x, t + c)$. Deduce that if $y(t) = (x(t), f(t))$ solves the IVP $\frac{dy}{dt} = \tilde{X}(y)$, $y(t_0) = (x_0, t_0)$, then $x(t)$ solves $\frac{dx}{dt} = X(x, t)$, $x(t_0) = x_0$.

This may seem like a swindle—we don't seem to have done much beyond coalescing the original time variable t with the space variables, i.e., we have switched from a space + time description to a space-time description. But there is another important difference, namely $\tilde{X}$ takes values in $V \times \mathbf{R}$. In any case, this is a true reduction of the non-autonomous case to the autonomous case, and it is important, since autonomous differential equations have special properties that make them easier to study. Here is one such special property of autonomous systems.

8.2.2 Proposition. *If $x : (a, b) \to V$ is any solution of the autonomous differentiable equation $\frac{dx}{dt} = X(x)$ and $t_0 \in \mathbf{R}$, then $y : (a + t_0, b + t_0) \to V$ defined by $y(t) = x(t - t_0)$ is also a solution of the same equation.*

▷ **8.2—Exercise 4.** Prove the above Proposition.

Consequently, when considering the IVP for an autonomous differentiable equation we can assume that $t_0 = 0$. For if $x(t)$ is a solution with $x(0) = p$, then $x(t - t_0)$ will be a solution with $x(t_0) = p$.

8.2.3 Remark. There is another trick that allows us to reduce the study of higher order differential equations to the case of first order equations. Consider the second order differential equation: $\frac{d^2x}{dt^2} = f(x, \frac{dx}{dt}, t)$. Introduce a new variable v, (the velocity) and consider the following related system of first order equations: $\frac{dx}{dt} = v$ and $\frac{dv}{dt} = f(x, v, t)$. It is pretty obvious there is a close relation between curves $x(t)$ satisfying $x''(t) = f(x(t), x'(t), t)$ and pairs of curves $x(t)$, $v(t)$ satisfying $x'(t) = v(t)$ and $v'(t) = f(x(t), v(t), t)$.

▷ **8.2—Exercise 5.** Define the notion of an initial value problem for the above second order differential equation, and write a careful statement of the relation between solutions of this initial value problem and the initial value problem for the related system of first order differential equations.

We will now look more closely at the uniqueness question for solutions of an initial value problem. The answer is summed up succinctly in the following result.

8.2.4 Maximal Solution Theorem. *Let $\frac{dx}{dt} = X(x)$ be an autonomous differential equation in V and p any point of V. Among all solutions $x(t)$ of the equation that satisfy the initial condition $x(0) = p$, there is a maximum one, σ_p, in the sense that any solution of this IVP is the restriction of σ_p to some interval containing zero.*

▷ **8.2—Exercise 6.** If you know about connectedness you should be able to prove this very easily. First, using the local uniqueness theorem, show that any two solutions agree on the intersection of their domains. Then define σ_p to be the union of all solutions.

Henceforth whenever we are considering some autonomous differential equation, σ_p will denote this maximal solution curve with initial condition p. The interval on which σ_p is defined will be denoted by $(\alpha(p), \omega(p))$, where of course $\alpha(p)$ is either $-\infty$ or a negative real number, and $\omega(p)$ is either ∞ or a positive real number.

We have seen that the maximal solution need **not** be defined on all of $\mathbf{R}$, and it is important to know just how the solution "blows up" as t approaches a finite endpoint of its interval of definition. *A priori* it might seem that the solution could remain in some bounded region, but it is an important fact that this is impossible—if $\omega(p)$ is finite then the reason the solution cannot be continued past $\omega(p)$ is simply that $\sigma(t)$ escapes to infinity (in the sense that $\|\sigma_p(t)\| \to \infty$) as t approaches $\omega(p)$.

8.2.5 No Bounded Escape Theorem. *If $\omega(p) < \infty$ then $\lim_{t\to\omega(p)} \|\sigma_p(t)\| = \infty$, and similarly, if $\alpha(p) > -\infty$ then $\lim_{t\to\alpha(p)} \|\sigma_p(t)\| = \infty$.*

▷ **8.2—Exercise 7.** Prove the No Bounded Escape Theorem.
(Hint: If $\lim_{t\to\omega(p)} \|\sigma(p)\| \neq \infty$, then by Bolzano-Weierstrass there would be a sequence t_k converging to $\omega(p)$ from below, such that $\sigma_p(t_k) \to q$. Then use the local existence theorem around q to show that you could extend the solution beyond $\omega(p)$. Here is where we get to use the fact there is a neighborhood O of q such that a solution exists with any initial condition q' in O and defined **on the whole interval** $(-\epsilon, \epsilon)$. For k sufficiently large, we will have both $\sigma_p(t_k)$ in O and $t_k > \omega - \epsilon$, which quickly leads to a contradiction.)

Here is another interesting and important special properties of autonomous systems.

▷ **8.2—Exercise 8.** Show that the images of the σ_p partition V into disjoint smooth curves (the "streamlines" of smoke particles). These curves are referred to as the *orbits* of the ODE. (Hint: If $x(t)$ and $\xi(t)$ are two solutions of the same autonomous ODE and if $x(t_0) = \xi(t_1)$ then show that $x(t_0 + s) = \xi(t_1 + s)$.)

▷ **8.2—Exercise 9.** Show that if the vector field $X : V \to V$ is C^k then each maximal solution $\sigma_p : (\alpha_p, \omega_p) \to V$ is C^{k+1}. (Hint: To begin with we know that σ_p is differentiable and hence continuous, so $t \mapsto X(\sigma_p(t))$ is at least continuous. Now use the relation $\sigma_p'(t) = X(\sigma_p(t))$ to argue by induction.)

8.2.6 Remark. In the next section we will discuss the smoothness of $\sigma_p(t)$ as a function of p and t jointly, and see that it is of class C^k.

8.2.7 Definition. A C^k vector field $X : V \to V$ (and also the autonomous differential equation $\frac{dx}{dt} = X(x)$) is called *complete* if $\alpha(p) = -\infty$ and $\omega(p) = \infty$ for all p in V. In this case, for each $t \in \mathbf{R}$ we define a map $\phi_t : V \to V$ by $\phi_t(p) = \sigma_p(t)$. The mapping $t \mapsto \phi_t$ is called the *flow* generated by the differential equation $\frac{dx}{dt} = X(x)$.

8.2.8 Remark. Using our smoke particle metaphor, the meaning of ϕ_t can be explained as follows: if a puff of smoke occupies a region U at a given time, then t units of time later it will occupy the region $\phi_t(U)$. Note that ϕ_0 is clearly the identity mapping of $\mathbf{R}^n$

▷ **8.2—Exercise 10.** Show that the ϕ_t satisfy $\phi_{t_1+t_2} = \phi_{t_1}\phi_{t_2}$, so that in particular $\phi_{-t} = \phi_t^{-1}$. By the next section, each ϕ_t is actually a "diffeomorphism" of V, i.e., both it and its inverse are C^1. So the flow generated by a complete, autonomous C^1 vector field is a homomorphism of the additive group of real numbers into the group of diffeomorphisms of V.

8.3 Smoothness of Solutions of ODE.

As we saw above, it is an easy exercise to show that each solution of the differential equation $\frac{dx}{dt} = X(x)$ will be C^{k+1} if X is C^k. But in practice it is also important to know how solutions of an ODE depend on initial conditions. Also, if we have a family of $X(x, \alpha)$ of vector fields depending on some parameters $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbf{R}^k$, we would like to know how the solutions of the corresponding IVP depend on these parameters. The following theorem answers all these questions. We will not give the proof here. It can be found in any good textbook on ODE. (One excellent source is *Differential Equations, Dynamical Systems, and Linear Algebra* by Morris Hirsch and Stephen Smale. See in particular Chapter 15.)

Theorem. *Let $X : V \times \mathbf{R}^k \to V$ be a C^k map. We regard $X(x, \alpha)$ as a family of vector fields on V depending on the parameter $\alpha \in \mathbf{R}^k$, and denote by $t \mapsto \sigma_p^\alpha(t)$ the maximal solution of the ODE $\frac{dx}{dt} = X(x, \alpha)$ with $\sigma_p^\alpha(0) = p$. Then the map $(p, t, \alpha) \mapsto \sigma_p^\alpha(t)$ is of class C^k.*

Lecture 9
Linear ODE and Numerical Methods

9.1 Linear Differential Equations.

If $A : V \to V$ is a continuous linear operator on V then we can also consider A as a vector field on V. The corresponding autonomous ODE, $\frac{dx}{dt} = Ax$ is called a *linear differential equation on* V.

Denote by $C(\mathbf{R}, V)$ the continuous maps of $\mathbf{R}$ into V, and let $F = F^A_{x_0}$ be the map of $C(\mathbf{R}, V)$ to itself defined by $F(x)(t) := x_0 + \int_0^t A(x(s))\,ds$. Since A is linear, this can also be written as $F(x)(t) := x_0 + A\int_0^t x(s)\,ds$. We know that the solution of the IVP with initial value x_0 is just the unique fixed point of F, so let's try to find it by successive approximations starting from the constant path $x^0(t) = x_0$. If we recall that the sequence of successive approximations, x^n, is defined recursively by $x^{n+1} = F(x^n)$, then an elementary induction gives $x^n(t) = \sum_{k=0}^n \frac{1}{k!}(tA)^k x_0$, suggesting that the solution to the initial value problem should be given by the limit of this sequence, namely the infinite series $\sum_{k=0}^\infty \frac{1}{k!}(tA)^k x_0$. Now (for obvious reasons) given a linear operator T acting on V, the limit of the infinite series of operators $\sum_{k=0}^\infty \frac{1}{k!}T^k$ is denoted by e^T or $\exp(T)$ so we can also say that the solution to our IVP should be $e^{tA}x_0$.

The convergence properties of the series for $e^T x$ follow easily from the Weierstrass M-test. If we define $M_k = \frac{1}{k!}|||\,T\,|||^k r$, then $\sum M_k$ converges to $e^{|||\,T\,|||}r$, and since $\left\|\frac{1}{k!}T^k x\right\| < M_k$ when $\|x\| < r$, it follows that $\sum_{k=0}^\infty \frac{1}{k!}T^k x$ converges absolutely and uniformly to a limit, $e^T x$, on any bounded subset of V.

▷ **9.1—Exercise 1.** Provide the details for the last statement.
(Hint: Since the sequence of partial sums $\sum_{k=0}^n M_k$ converges, it is Cauchy, i.e., given $\epsilon > 0$, we can choose N large enough that $\sum_m^{m+k} M_k < \epsilon$ provided $m > N$. Now if $\|x\| < r$, $\left\|\sum_{k=0}^{m+k} \frac{1}{k!}T^k x - \sum_{k=0}^m \frac{1}{k!}T^k x\right\| < \sum_m^{m+k} M_k < \epsilon$, proving that the infinite series defining $e^T x$ is uniformly Cauchy and hence uniformly convergent in $\|x\| < r$.)

Since the partial sums of the series for $e^T x$ are linear in x, so is their limit, so e^T is indeed a linear operator on V.

Next observe that since a power series in t can be differentiated term by term, it follows that $\frac{d}{dt}e^{tA}x_0 = Ae^{tA}x_0$, i.e., $x(t) = e^{tA}x_0$ is a solution of the ODE $\frac{dx}{dt} = Ax$. Finally, substituting zero for t in the power series gives $e^{0A}x_0 = x_0$. This completes the proof of the following proposition.

9.1.1 Proposition. *If A is a linear operator on V then the solution of the linear differential equation $\frac{dx}{dt} = Ax$ with initial condition x_0 is $x(t) = e^{tA}x_0$.*

As a by-product of the above discussion we see that a linear ODE $\frac{dx}{dt} = Ax$ is complete, and the associated flow ϕ_t is just e^{tA}. By a general fact about flows it follows that $e^{(s+t)A} = e^{sA}e^{tA}$, and $e^{-A} = (e^A)^{-1}$, so $\exp : A \mapsto e^A$ is a map of the vector space $L(V)$

of all linear maps of V into the group $\boldsymbol{GL}(V)$ of invertible elements of $L(V)$ and for each $A \in L(V)$, $t \mapsto e^{tA}$ is a homomorphism of the additive group of real numbers into $\boldsymbol{GL}(V)$.

▷ **9.1—Exercise 2.** Show more generally that if A and B are commuting linear operators on V then $e^{A+B} = e^A e^B$. (Hint: Since A and B commute, the Binomial Theorem is valid for $(A+B)^k$, and since the series defining e^{A+B} is absolutely convergent, it is permissible to rearrange terms in the infinite sum. For a different proof, show that $e^{tA}e^{tB}x_0$ satisfies the initial value problem $\frac{dx}{dt} = (A+B)x$, $x(0) = x_0$, and use the uniqueness theorem.)

▷ **9.1—Exercise 3.** Now assume that V is a finite dimensional inner-product space. Show that $\left(e^A\right)^* = e^{A^*}$. (Hint: Recall that $(AB)^* = B^*A^*$.) Show that if A is skew-adjoint (meaning $A^* = -A$) than e^A is in the orthogonal group. Conversely show that if e^{tA} is in the orthogonal group for all $t \in \mathbf{R}$ then A must be skew-adjoint.

▷ **9.1—Exercise 4.** Let's say that $A \in L(V)$ is tangent to the orthogonal group $\boldsymbol{O}(V)$ at the identity if there is a differentiable path $\sigma : (a,b) \to L(V)$ defined near 0, such that $\sigma(t) \in \boldsymbol{O}(V)$ for all t, $\sigma(0) = I$, and $\sigma'(0) = A$. From what we have just seen, every skew-adjoint A is tangent to $\boldsymbol{O}(V)$ at I. Show that conversely any A that is tangent to $\boldsymbol{O}(V)$ at I must be skew-adjoint. (Hint: Differentiate the identity $\sigma(t)\sigma(t)^* = I$ at $t = 0$.)

9.2 Numerical Solutions of ODE.

Once we go beyond the linear case, only a few rather special initial value problems can be solved in closed form using standard elementary functions. For the general case it is necessary to fall back on constructing a solution numerically. But what algorithm should we use? A natural first guess is successive approximations. But while that is a powerful theoretical tool for studying general properties of initial value problems—and in particular for proving existence and uniqueness—it turns out to be so inefficient as an approach to calculating numerical solutions, that it is rarely used.

In fact there is no unique answer to the question of what numerical algorithm to use for solving ODEs, for there is no one method that is "best" in all situations. There are integration routines that are fast and accurate when used with the majority of equations one meets. Perhaps the most popular of these is the fourth order Runge-Kutta algorithm (which we will consider below). But there are many special situations that require a more sophisticated approach. Indeed, this is still a very active area of research, and there are literally dozens of books on the subject.

The initial goal will be to give you a quick first impression of how numerical methods work by describing one of the oldest numerical approaches to solving an initial value problem, the so-called "Euler Method". While rarely a good choice for practical computation, it is intuitive, simple, and effective, and it is also the basis for some of the more sophisticated algorithms. This makes it an excellent place to become familiar with the basic concepts that enter into the numerical integration of ODE.

In what follows we will suppose that X is a C^1 time-dependent vector field on V, and given t_0 in $\mathbf{R}$ and x^0 in V we will denote by $\sigma(X, x^0, t_0, t)$ the maximal solution, $x(t)$, of the differential equation $\frac{dx}{dt} = X(x, t)$ satisfying the initial condition $x(t_0) = x^0$. The goal in the numerical integration of ODE is to devise effective methods for approximating $x(t)$ on an interval $I = [t_0, T]$. The strategy that many methods use is to interpolate N equally spaced gridpoints $t_1, \ldots t_N$ in the interval I, defined by $t_k := t_0 + k\Delta t$ with $\Delta t = \frac{T-t_0}{N}$, and then use some rule to define values $x^1, \ldots, x^N$ in V, in such a way that when N is large each x^k is close to the corresponding $x(t_k)$. The quantity $\max_{1\leq k\leq N} \left\|x^k - x(t_k)\right\|$ is called the *global error* of the algorithm, and if it converges to zero as N tends to infinity (for every choice of X, t_0, x^0, and T), then we say that we have a *convergent algorithm*. Euler's Method is a convergent algorithm of this sort.

One common way to construct the algorithm that produces the values $x^1, \ldots, x^N$ uses a recursion based on a so-called "stepping procedure", namely a function, $\Sigma(X, x^0, t_0, \Delta t)$, having as inputs:

1) a time-dependent vector field X on V,

2) an initial condition x^0 in V,

3) an initial time t_0 in $\mathbf{R}$, and

4) a "time-step" Δt in $\mathbf{R}$,

and with output a point of V that for small Δt approximates $\sigma(X, x^0, t_0, t_0 + \Delta t)$ well. More precisely, the so-called "local truncation error", defined by

$$\left\|\sigma(X, x^0, t_0, t_0 + \Delta t) - \Sigma(X, x^0, t_0, \Delta t)\right\|,$$

should approach zero at least quadratically in the time-step Δt. Given such a stepping procedure, the approximations x^k of the $x(t_k)$ are defined recursively by $x^{k+1} = \Sigma(X, x^k, t_k, \Delta t)$. Numerical integration methods that follow this general pattern are referred to as finite difference methods.

9.2.1 Remark. There are two sources that contribute to the global error, $\left\|x^k - x(t_k)\right\|$. First, each stage of the recursion will give an additional local truncation error added to what has already accumulated up to that point. But, in addition, after the first step, there will be an error because the recursion uses $\Sigma(X, x^k, t_k, \Delta t)$ rather than the unknown $\Sigma(X, x(t_k), t_k, \Delta t)$. (In practice there is a third source of error, namely machine round-off error from using floating-point arithmetic. We will usually ignore this and pretend that our computers do precise real number arithmetic, but there are situations where it is important to take it into consideration.)

For Euler's Method the stepping procedure is simple and natural. It is defined by:

Euler Step

$\Sigma^E(X, x^0, t_0, \Delta t) := x^0 + \Delta t\, X(x^0, t_0)$.

It is easy to see why this is a good choice. If as above we denote $\sigma(X, x^0, t_0, t)$, by $x(t)$, then by Taylor's Theorem,

$$\begin{aligned} x(t_0 + \Delta t) &= x(t_0) + \Delta t\, x'(t_0) + O(\Delta t^2) \\ &= x^0 + \Delta t\, X(x^0, t_0) + O(\Delta t^2) \\ &= \Sigma^E(X, x^0, t_0, \Delta t) + O(\Delta t^2), \end{aligned}$$

so $\big\|\sigma(X, x^0, t_0, t_0 + \Delta t) - \Sigma(X, x^0, t_0, \Delta t)\big\|$, the local truncation error for Euler's Method, does go to zero quadratically in Δt. When we partition $[0, T]$ into N equal parts, $\Delta t = \frac{T - t_0}{N}$, each step in the recursion for computing x^k will contribute a local truncation error that is $O(\Delta t^2) = O(\frac{1}{N^2})$. Since there are N steps in the recursion and at each step we add $O(\frac{1}{N^2})$ to the error, this suggests that the global error will be $O(\frac{1}{N})$, and hence will go to zero as N tends to infinity. However, because of the remark above, this is not a complete proof that Euler's Method is convergent, and we will not give the details of the rigorous argument.

▷ **9.2—Exercise 1.** Show that Euler's Method applied to the initial value problem $\frac{dx}{dt} = x$ with $x(0) = 1$ gives $\lim_{N\to\infty}(1 + \frac{t}{N})^N = e^t$.

Two positive features of Euler's method are it's intuitiveness and the ease with which it can be programmed. Beyond that there is little to recommend it as a practical method for solving real-world problems. It requires very small time steps to get reasonable accuracy, making it too slow for serious applications, and in fact it is rarely used except for pedagogical purposes.

On the other hand, as we have already said, there is one excellent general purpose finite difference method for solving IVPs, and it goes by the name Runge-Kutta—or more properly the fourth order Runge-Kutta Method—since there is a whole family of Runge-Kutta methods. The stepping procedure for fourth order Runge-Kutta is:

Runge-Kutta Step

$\Sigma^{RK^4}(X, x_0, t_0, \Delta t) := x_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$, where:
$k_1 = \Delta t\, X(x_0, t_0)$
$k_2 = \Delta t\, X(x_0 + \frac{1}{2}k_1, t_0 + \frac{\Delta t}{2})$
$k_3 = \Delta t\, X(x_0 + \frac{1}{2}k_2, t_0 + \frac{\Delta t}{2})$
$k_4 = \Delta t\, X(x_0 + k_3, t_0 + \Delta t)$

It is of course natural to ask where this formula comes from. But if you recall Simpson's Rule then the above should not look all that unreasonable, and indeed if $f(x, t) = \phi(t)$ then recall that the solution of the IVP reduces to the integral of ϕ and in this case the Runge-Kutta formula reduces precisely to Simpson's Rule. But even better, like Simpson's Rule, Runge-Kutta is fourth order, meaning that the local truncation error goes to zero as the fifth power of the step-size, and the global error as the fourth power. So if for a fixed step-size we have attained an accuracy of 0.1, then with one-tenth the step-size (and so ten times the number of steps and ten times the time) we can expect an accuracy of 0.00001, whereas with the Euler method, ten times the time would only increase accuracy from 0.1 to 0.01.

▷ **9.2—Exercise 2.** Write a Matlab M-File function `Euler(X,x0,T,n)` that estimates $x(T)$, the solution at time T of the initial value problem $\frac{dx}{dt} = X(x)$, $x(0) = x0$ by applying the Euler stepping method to the interval $[0, T]$ with n time steps. Similarly write such a function `RungeKutta(X,x0,T,n)` that uses the Runge-Kutta stepping method. Make some experiments using the case $V = \mathbf{R}$, $\frac{dx}{dt} = x$ with $x0 = 1$ and $T = 1$, so that $x(T) = e$. Check how large n has to be to get various degrees of accuracy using the two methods.

Lecture 10
The Theorem of Frobenius

10.1 What if Time were Two-dimensional?

With our study of ODE, we have completed the review of the analytical tools that will be required for our study curve theory. However, when we turn later to the study of surfaces, there is an additional tool we will need. This is embodied in a theorem of Frobenius that we consider next.

One approach to the Frobenius Theorem is consider what would become of the local existence and uniqueness theorem for the IVP for ODE if "time", instead of being one-dimensional, was two-dimensional. That is, suppose that an instant of time is represented not by a single real number $t \in \mathbf{R}$, but by an ordered pair (t_1, t_2) of real numbers. (We still assume that "space" is represented by points of a vector space, V, although shortly we will specialize to the case $V = \mathbf{R}^n$.) What is the natural generalization of an ODE and its associated initial value problem?

The history of a smoke particle—that is, a particle that streams along with the wind—will now be described not by a "world-line" $t \mapsto x(t)$ from $\mathbf{R}$ to V but by a "world-sheet" $(t_1, t_2) \mapsto x(t_1, t_2)$, a map from our pair of "time" parameters in $\mathbf{R}^2$ to V. And since their are now two time coordinates, the particle will have not one but rather two velocity vectors associated to it at time (t_1, t_2), namely $\frac{\partial x(t_1,t_2)}{\partial t_1}$ and $\frac{\partial x(t_1,t_2)}{\partial t_2}$. Thus the "wind" must now be described not by a single time-dependent vector field $X : V \times \mathbf{R} \to V$, but by a pair of time-dependent vector fields $X^1 : V \times \mathbf{R}^2 \to V$ and $X^2 : V \times \mathbf{R}^2 \to V$. Operationally, the definition of $X^i(v, t_1, t_2)$ is as follows: place a smoke particle at v, and at time (t_1, t_2) let it go, and measure its velocity $\frac{\partial x(t_1,t_2)}{\partial t_i}$. In all that follows we will assume that these vector fields X^1 and X^2 are C^k, $k \geq 2$.

The ordinary differential equation IVP $\frac{dx}{dt} = X(x, t)$, $x(t^0) = v^0$ describing a smoke particle motion when time is one-dimensional is now replaced by a similar IVP for a "system of first order partial differential equations":

$$(*) \qquad \begin{aligned} &1) \quad x(t_1^0, t_2^0) = v^0, \\ &2) \quad \frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2), \\ &3) \quad \frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2), \end{aligned}$$

An Algorithm for Constructing Solutions of the System $(*)$.

We now describe a simple and intuitive algorithm that we will refer to as Algorithm F for constructing the solution $x(t_1, t_2)$ of the initial value problem $(*)$ near $(t_1, t_2) = (t_1^0, t_2^0)$, provided a solution exists. To be more precise, the algorithm will produce a map $(t_1, t_2) \mapsto x(t_1, t_2)$, defined for (t_1, t_2) in a neighborhood U of (t_1^0, t_2^0) defined by $|t_i - t_i^0| < \epsilon$, $i = 1, 2$, and for which the following five statements a) to e) are valid:

a) $x(t_1, t_2)$ satisfies the initial value condition 1) of $(*)$,

b) $x(t_1, t_2)$ satisfies 2) of $(*)$, at least along the line $t_2 = t_2^0$.

c) $x(t_1, t_2)$ satisfies 3) of $(*)$ in all of U,

d) $x : U \to V$ is C^k,

e) The properties a), b), c) uniquely determine the function $x(t_1, t_2)$ near (t_1^0, t_2^0), hence it will be the unique solution of $(*)$ near (t_1^0, t_2^0), if any solution exists.

The strategy behind Algorithm F comes from a subtle change in point of view. Instead of regarding 2) and 3) of $(*)$ as a pair of coupled PDE with independent variables t_1 and t_2, we consider them as two independent ODEs, the first with t_1 as independent variable and t_2 as parameter, and the second with these roles reversed.

Algorithm F is defined as follows. First we solve the ODE initial value problem $\frac{dy}{dt} = Y(y, t)$ and $y(t_1^0) = v^0$, where $Y(v, t) := X^1(v, t, t_2^0)$. By the local existence and uniqueness theorem for ODE, if ϵ is sufficiently small then the solution will exist for $|t - t_1^0| < \epsilon$, and moreover $y(t)$ will be C^{k+1}. We now define $x(t_1, t_2^0) = y(t_1)$ for $|t_1 - t_1^0| < \epsilon$. This of course guarantees that no matter how we define (t_1, t_2) for other values of t_2, statements a) and b) will be valid. Moreover, by the uniqueness of solutions of the IVP for ODEs, it is clear that conversely if a) and b) are to hold then we **must** define $x(t_1, t_2^0)$ this way on $|t_1 - t_1^0| < \epsilon$.

Next we consider the ODE $\frac{dz}{dt} = Z(z, t, t_1)$ where t_1 is considered a parameter and the vector field Z is defined by $Z(v, t, t_1) := X^2(v, t_1, t)$. It again follows from the local existence and uniqueness theorem for ODE that if ϵ is sufficiently small, then for each t_1 with $|t_1 - t_1^0| < \epsilon$, the IVP $\frac{dz}{dt} = Z(z, t, t_1)$, $z(t_2^0) = y(t_1) = x(t_1, t_2^0)$ has a unique solution $z_{t_1}(t)$ for $|t - t_2^0| < \epsilon$. We now define $x(t_1, t_2)$ in U by $x(t_1, t_2) = z_{t_1}(t_2)$. We note that because of the initial condition $z(t_2^0) = y(t_1) = x(t_1, t_2^0)$, this extends the definition of $x(t_1, t_2)$ in the first part of the algorithm, so properties a) and b) still hold. But now in addition, c) also clearly holds, and moreover in order for c) to hold then by the uniqueness theorem for ODE the way we extended the definition of $x(t_1, t_2)$ to all of U was the only way possible; in other words property e) is established. Finally, property d) is immediate from the Theorem of Section 8.3 (concerning the smoothness of solutions of ODE with respect to initial conditions and parameters).

[That completes the formal description of Algorithm F. We now restate it in less formal and more intuitive language. First find $x(t_1, t_2)$ along the line $t_2 = t_2^0$ by freezing the value of t_2 at t_2^0 and regarding the partial differential equation $\frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2)$ as an ODE in which t_2 is just a parameter. Then, regard the PDE $\frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2)$ as an ODE in which t_1 is a parameter, and for each parameter value t_1 near t_1^0 solve this ODE, taking for initial value at $t_2 = t_2^0$ the value $x(t_1, t_2^0)$, found in the first step.]

The question we will consider next is under what conditions the function $x(t_1, t_2)$ defined by Algorithm F satisfies equation 2) of the system $(*)$ in all of U, and not just along the segment $t_1 = t_1^0$.

First we look at a couple of examples.

10.1—Example 1. Take $V = \mathbf{R}$, and define $X^1(x, t_1, t_2) = X^2(x, t_1, t_2) = x$, so the system of PDE is $\frac{\partial x}{\partial t_1} = \frac{\partial x}{\partial t_2} = x$. In this case Algorithm F leads to the function $x(t_1, t_2) = v^0 e^{(t_1 - t_1^0)} e^{(t_2 - t_2^0)}$ which clearly does solve the system $(*)$.

10.1—Example 2. Let $V = \mathbf{R}$, define $X^1(x, t_1, t_2) = x$, $X^2(x, t_1, t_2) = 1$ and take as initial condition $x(0, 0) = 1$. Now the system of partial differential equations is $\frac{\partial x}{\partial t_1} = x$ and $\frac{\partial x}{\partial t_2} = 1$. Applying Algorithm F, the first equation together with the initial condition gives $x(t_1, 0) = e^{t_1}$, and the second equation then implies that $x(t_1, t_2) = e^{t_1} + t_2$. In this case the first of the two partial differential equations is clearly **not** satisfied off the line $t_2 = 0$!

So we see life is not going to be quite as simple as with one-dimensional time. For certain choices of X^1 and X^2 it will be possible to solve the IVP locally for every choice of initial condition $x(t_1^0, t_2^0) = v^0$, while for other X^1 and X^2 this will not be so. Let's give a name to distinguish between these cases.

10.1.1 Definition. Let $X^1 : V \times \mathbf{R}^2 \to V$ and $X^2 : V \times \mathbf{R}^2 \to V$ be C^2 maps. We call the system of PDEs $\frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2)$ and $\frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2)$ is *integrable*, if for every $(t_1^0, t_2^0) \in \mathbf{R}^2$ and $v^0 \in V$, their is a solution x of this system that is defined in a neighborhood of (t_1^0, t_2^0) and that satisfies the initial condition $x(t(t_1^0, t_2^0)) = v^0$

What the Frobenius Theorem does is provide a necessary and sufficient condition for a system to be integrable. Moreover, this condition is both natural and easy and practical to apply. Since it becomes somewhat easier to formulate in the case that $V = \mathbf{R}^n$, we will assume we are in that case from now on. (Of course this is no real loss of generality, since it simply amounts to choosing a basis for V.)

The vector fields X^1 and X^2 can now be described by $2n$ C^k real-valued functions X_j^1 and X_j^2 on $\mathbf{R}^n \times \mathbf{R}^2$:

$$X^i(x, t_1, t_2) = X^i(x_1, \ldots, x_n, t_1, t_2) = (X_1^i(x_1, \ldots, x_n, t_1, t_2), \ldots, X_n^i(x_1, \ldots, x_n, t_1, t_2))$$

10.1.2 Definition. Let $X^1 : \mathbf{R}^n \times \mathbf{R}^2 \to \mathbf{R}^n$ and $X^2 : \mathbf{R}^n \times \mathbf{R}^2 \to \mathbf{R}^n$ be C^2 vector fields on $\mathbf{R}^n$ depending on two real parameters t_1 and t_2. We will call X^1 and X^2 *compatible* if the following n conditions hold identically:

$$\frac{\partial X_i^1}{\partial t_2} + \sum_{j=1}^{n} \frac{\partial X_i^1}{\partial x_j} X_j^2 = \frac{\partial X_i^2}{\partial t_1} + \sum_{j=1}^{n} \frac{\partial X_i^2}{\partial x_j} X_j^1, \qquad 1 \le i \le n.$$

Frobenius Theorem. *If $X^i : \mathbf{R}^n \times \mathbf{R}^2 \to \mathbf{R}^n$ $i = 1, 2$ are two C^2 vector fields on $\mathbf{R}^n$, then the system of PDEs $\frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2)$ and $\frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2)$ is integrable if and only if X^1 and X^2 are compatible.*

PROOF. We first check the necessity of the condition. We assume the system is integrable and show that the compatibility identities hold at an arbitrary point $(x^0, t_1^0, t_2^0) \in \mathbf{R}^n \times \mathbf{R}^2$. Let $x(t_1, t_2)$ be a solution of $\frac{\partial x}{\partial t_i} = X^i(x, t_1, t_2)$, $i = 1, 2$ defined near (t_1^0, t_2^0) and satisfying $x((t_1^0, t_2^0)) = x^0$. From d) and e) above we know that x is C^2, so in particular its second cross partial-derivatives at (x^0, t_1^0, t_2^0) exist and are equal. If we differentiate the equation $\frac{\partial x_i(t_1,t_2)}{\partial t_1} = X_i^1(x(t_1, t_2), t_1, t_2)$ with respect to t_2, using the chain-rule, we find:

$$\begin{aligned}\frac{\partial}{\partial t_2}\frac{\partial x(t_1,t_2)}{\partial t_1} &= \frac{\partial X_i^1(x,t_1,t_2)}{\partial t_2} + \sum_{j=1}^n \frac{\partial X_i^1(x,t_1,t_2)}{\partial x_j}\frac{\partial x_j(t_1,t_2)}{\partial t_2} \\ &= \frac{\partial X_i^1(x,t_1,t_2)}{\partial t_2} + \sum_{j=1}^n \frac{\partial X_i^1(x,t_1,t_2)}{\partial x_j} X_j^2(x,t_1,t_2),\end{aligned}$$

so if we interchange the roles of t_1 and t_2, set the cross-derivatives equal and evaluate at (x^0, t_1^0, t_2^0) we get precisely the compatibility identities at that point.

Now assume that X^1 and X^2 are compatible. Given $(x^0, t_1^0, t_2^0) \in \mathbf{R}^n \times \mathbf{R}^2$ use Algorithm F to define $x(t_1, t_2)$ in U and define $z(t_1, t_2)$ in U by $z(t_1, t_2) := \frac{\partial x(t_1,t_2)}{\partial t_1} - X^1(x(t_1, t_2), t_1, t_2)$. To complete the proof we must show that z is identically zero in U. But for that it will suffice to show that z satisfies the linear ODE $\frac{\partial z}{\partial t_2} = \sum_{j=1}^n \frac{\partial X^2}{\partial x_j} z_j$. For zero is a solution of this equation and z has the initial value zero at $t_2 = 0$ by property b) of Algorithm F, and then by uniqueness of solutions z must be zero. From the definition of z and the chain-rule,

$$\begin{aligned}\frac{\partial z}{\partial t_2} &= \frac{\partial}{\partial t_2}\frac{\partial x(t_1,t_2)}{\partial t_1} - \frac{\partial X^1}{\partial t_2} - \sum_{j=1}^n \frac{\partial X^1}{\partial x_j}\frac{\partial x_j}{\partial t_2} \\ &= \frac{\partial}{\partial t_1}\frac{\partial x(t_1,t_2)}{\partial t_2} - \left(\frac{\partial X^1}{\partial t_2} + \sum_{j=1}^n \frac{\partial X^1}{\partial x_j} X_j^2\right) \\ &= \frac{\partial}{\partial t_1} X^2(x(t_1,t_2),t_1,t_2) - \left(\frac{\partial X^1}{\partial t_2} + \sum_{j=1}^n \frac{\partial X^1}{\partial x_j} X_j^2\right) \\ &= \frac{\partial X^2}{\partial t_1} + \sum_{j=1}^n \frac{\partial X^2}{\partial x_j}\frac{\partial x_j}{\partial t_1} - \left(\frac{\partial X^1}{\partial t_2} + \sum_{j=1}^n \frac{\partial X^1}{\partial x_j} X_j^2\right) \\ &= \frac{\partial X^2}{\partial t_1} + \sum_{j=1}^n \frac{\partial X^2}{\partial x_j}\frac{\partial x_j}{\partial t_1} - \left(\frac{\partial X^2}{\partial t_1} + \sum_{j=1}^n \frac{\partial X^2}{\partial x_j} X_j^1\right) \\ &= \sum_{j=1}^n \frac{\partial X^2}{\partial x_j}\left(\frac{\partial x_j}{\partial t_1} - X_j^1\right) = \sum_{j=1}^n \frac{\partial X^2}{\partial x_j} z_j \quad \blacksquare\end{aligned}$$

10.2 Fifth Matlab Project.

Your assignment for the fifth project is to implement Algorithm F in Matlab. This should consist of an M-File, AlgorithmF.m, defining a Matlab function AlgorithmF(X1,X2,...), together with various auxilliary M-Files that implement certain subroutines required by the algorithm. (As usual, it is a matter of programming taste to what extent you use subfunctions as opposed to functions defined in separate M-Files.)

Let's consider in more detail just what the inputs and output to AlgorithmF should be. First, the **output** should represent the function $x(t_1, t_2)$ that solves the IVP $(*)$. But since we are going to get this solution by solving some ODEs numerically (using Runge-Kutta), in Matlab the output x will be a two-dimensional array x(i,j) of vectors of length n, To be specific, we take the domain U of x to be a rectangle a1 $\leq$ t1 $\leq$ b1 and a2 $\leq$ t2 $\leq$ b2, and we will use (a1,a2) for the point (t_1^0, t_2^0) where the initial condition $x(t_1^0, t_2^0) = x^0$ is given. So far then we see that the first line of our M-File will look something like:

```
function       x = AlgorithmF(X1,X2,x0,a1,a2,b1,b2,...)
```

We still have to specify the size of the output array x. This is given by two positive integers, T1Res and T2Res that give the number of subintervals into which we divide the intervals [a1,b1] and [a2,b2]. Let's define h1 := (b1 - a1)/T1Res and h2 := (b2 - a2)/T2Res. We take T1Res + 1 subdivision points in [a1,b1], a1 + i * h1, i = 0,1, ..., T1Res, and similarly we take T2Res + 1 subdivision points [a2,b2], a2 + j * h2, j = 0,1, ..., T2Res, It will be convenient to store these in arrays T1 and T2 of length T1Res + 1 and T2Res + 1 respectively. That is, T1(i) = a1 + i * h1 and T2(i) = a2 + i * h2. Then the array x(i,j) will have size T1Res + 1 by T2Res + 1. Namely, we will store at x(i,j) the value of the solution $x(t_1, t_2)$ of $(*)$ at the point (T1(i),T2(j)), or more exactly the approximate value found by our implementation of AlgorithmF in which we solve the ODEs approximately using Runge-Kutta. So now the first line of our M-File has become:

```
function       x = AlgorithmF(X1,X2,x0,a1,a2,b1,b2,T1Res,T2Res,...)
```

There is still one more input parameter needed, namely a real number StepSize that determines the accuracy of the Runge-Kutta algorithm. Strictly speaking StepSize is not the actual size of the steps used in the Runge-Kutta integration, but an upper bound for it. When we propagate the solution of an ODE y(t) from a value t = t0 where we already know it to a next value t = t0 + h where we need it, we will divide h in a number N of equal steps to make h/N less than StepSize and use that many steps in our Runge-Kutta method. (Since we will usually settle for accuracy of about 10^{-8} and Runge-Kutta is fourth order, in practice we usually take StepSize approximately 0.01). So finally the first line of our M-File has become:

```
function       x = AlgorithmF(X1,X2,x0,a1,a2,b1,b2,T1Res,T2Res,StepSize)
```

The first two input parameters X1 and X2 represent the vector fields defining the system of PDE we are dealing with. Each is a function of $n + 2$ variables, x1, x2,...,xn,t1,t2, In practice the actual functions substituted for these parameters will be taken from functions defined either in an M-File or an inline expression.

Once you understand the above discussion well you should find writing the actual code for AlgorithmF to be straightforward. We start by assigning to x(0,0) the value x0, Then, for i = 0 to T1Res, we inductively find x(i+1,0) from x(i,0) by using Runge-Kutta to solve the ODE $\frac{\partial x}{\partial t1} = X1(x, t1, a2)$ on the interval [T1(i),T1(i+1)] with initial value x(i,0) at time t1 = T1(i). Then, in a similar manner, for each i from 0 to T1Res, and each j from 0 to T2Res, we inductively find x(i,j+1) from x(i,j) by applying Runge-Kutta to solve the ODE $\frac{\partial x}{\partial t2}$ =X2(x,T1(i),t2) on the interval [T2(j),T2(j+1)] with initial value x(i,j) at time t2 = T2(j).

After the solution array x is constructed, it should be displayed either in wireframe (using meshgrid) or in patch mode (using surf).

Here is an "extra credit" addition to project 5. Write an M-File that defines a function that checks whether the two vector fields X1 and X2 are compatible. I suggest that you do this by checking numerically whether the two sides of the n compatibility conditions are equal at the points (T1(i),T2(j)). Here, to allow for roundoff errors, "equal" should mean that the absolute value of the difference is less than some tolerance. Use centered differences to compute the partial derivatives. See if you can make your test of equality "scale invariant". This means that if it succeeds or fails for X1 and X2, it should do the same if you multiply both X1 and X2 by the same scalar.

Lecture 11
Differentiable Parametric Curves

11.1 Definitions and Examples.

11.1.1 Definition. A *differentiable parametric curve* in $\mathbf{R}^n$ of class C^k ($k \geq 1$) is a C^k map $t \mapsto \alpha(t) = (\alpha_1(t), \ldots, \alpha_n(t))$ of some interval I (called the domain of α) into $\mathbf{R}^n$. We call α *regular* if its velocity vector, $\alpha'(t)$, is non-zero for all t in I. The image of the mapping α is often referred to as the *trace* of the curve α.

Some Conventions To avoid endless repetition, in all that follows we will assume that α is a regular, differentiable parametric curve in $\mathbf{R}^n$ of class C^k ($k \geq 2$), and we will abbreviate this by referring to α as a "parametric curve", or just as a "curve". Frequently we will take the domain I of α to be the closed interval $[a, b]$. In case $n = 2$ we will often write $\alpha(t) = (x(t), y(t))$ and similarly when $n = 3$ we will often write $\alpha(t) = (x(t), y(t), z(t))$.

11.1—Example 1. A Straight Line. Let $x^0, v^0 \in \mathbf{R}^n$ with $v_0 \neq 0$. Then $\alpha(t) = x^0 + tv^0$ is a straight line in $\mathbf{R}^n$ with constant velocity v^0. The domain is all of $\mathbf{R}$.

11.1—Example 2. Let $r > 0$, $n = 2$, take $I = [0, 2\pi]$ and define $\alpha(t) = (r\cos(t), r\sin(t))$. The trace of α is the set of (x, y) satisfying $x^2 + y^2 = r^2$, so α is a parameterization of the circle of radius r centered at the origin.

11.1—Example 3. Let $r, b > 0$, $n = 3$, $I = \mathbf{R}$, and define $\alpha(t) = (r\cos(t), r\sin(t), bt)$. The trace of α is a helix of radius r and pitch $\frac{2\pi}{b}$.

11.2 Reparametrizaion by Arclength.

Suppose $\phi : [c, d] \to \mathbf{R}$ is C^k and $\phi'(t) > 0$ for all $t \in [c, d]$. Then ϕ is monotonic and so a one-to-one map of $[c, d]$ onto $[a, b]$ where $a = \phi(c)$ and $b = \phi(d)$. Moreover, by the Inverse Function Theorem, $\psi = \phi^{-1}$ is C^k and $\psi'(\phi(t)) = 1/\phi'(t)$. If $\alpha : [a, b] \to \mathbf{R}^n$ is a parametric curve, then $\tilde{\alpha} := \alpha \circ \phi : [c, d] \to \mathbf{R}^n$ is a parametric curve with the same trace as α. In this setting, ϕ is called a parameter change and $\tilde{\alpha}$ is called a reparametrization of α. Since α and $\tilde{\alpha}$ have the same trace, in some naive sense at least, they represent the same "curve".

Of course for many purposes, the way a curve is parametric is of crucial importance—for example, reparametrizing a solution of an ODE will nearly always result in a non-solution. However in geometric considerations it is natural to concentrate on the trace and regard two parametric curves that differ by a change of parameterization as representing the same object. Formally speaking, differing by a change of parameterization is an equivalence relation on the set of parametric curves, and we regard the corresponding equivalence classes as being the primary objects of study in differential geometry. This raises a problem.

Whenever we define some property of parametric curves, then we should check that it is independent of the choice of parameterization. As we shall now see, there is an elegant way to avoid this complication. Namely, among all the parameterizations of a parametric curve α there is one that is the"most natural" choice, namely parameterization by arclength, and in our theoretical study of curves and their properties we will usually pefer this one over others and define properties of a curve using this parameterization.

Recall that in elementary Calculus, if $\alpha : [a, b] \to \mathbf{R}^n$ is a parametric curve, then its length L is defined as $L := \int_a^b \|\alpha'(t)\|\, dt$. More generally, the arclength along α is the function $s : [a, b] \to [0, L]$ defined by $s(t) = \int_a^t \|\alpha'(\tau)\|\, d\tau$. Since $s'(t) = \|\alpha'(t)\| > 0$, as remarked above it has a C^k inverse $t : [0, L] \to [a, b]$, and $\tilde{\alpha} : [0, L] \to \mathbf{R}^n$ defined by $\tilde{\alpha}(s) = \alpha(t(s))$ is a reparameterization of α called its parameterization by arclength. Note that by definition, the length of $\tilde{\alpha}$ between 0 and s is s, so the name is well-chosen.

▷ **11.2—Exercise 1.** Show that a parametric curve α is parametrized by arclength if and only if $\|\alpha'(t)\|$ is identically equal to 1.

11.2.1 Remark. From now on we will usually assume that α is parametrized by its arclength. It is traditional to signify this by using s as the variable name when dealing with such paths.

Notation If $\alpha(s)$ is a curve parametrized by arclength, then we will denote its unit tangent vector at s by $\overrightarrow{t}(s) := \alpha'(s)$.

11.2.2 Remark. In $\mathbf{R}^2$ there is an important orthogonal transformation that we shall denote by $R_{\frac{\pi}{2}}$. It is defined by $R_{\frac{\pi}{2}}(x, y) = (y, -x)$, and geometrically speaking it rotates any vector v through 90 degrees into a vector orthogonal to v. If α is a curve in $\mathbf{R}^2$ parametrized by arclength, then we define its normal at s by $\overrightarrow{n}(s) = R_{\frac{\pi}{2}}\, \overrightarrow{t}(s)$.

11.2.3 Remark. In $\mathbf{R}^n$ for $n > 2$, there is no natural way to assign a normal to a curve at every point that works in complete generality. To convince yourself of this, just think of the case of a straight line in $\mathbf{R}^3$—there is a whole circle of directions at each point that are normal to the line, and no way to prefer any one of them. However, at a point where $\alpha''(s) \neq 0$, we define the unit normal $\overrightarrow{n}(s)$ to α at s by $\overrightarrow{n}(s) := \frac{\alpha''(s)}{\|\alpha''(s)\|}$. (Recall that since $\|\alpha'(s)\|$ is identically equal to one, it follows that its derivative $\alpha''(s)$ is orthogonal to $\alpha'(s)$.)

▷ **11.2—Exercise 2.** Show that when the straight line $\alpha(t) = x^0 + tv^0$ is reparametrized by arclength the result is $\tilde{\alpha}(s) = x^0 + su$ where $u = \frac{v^0}{\|v^0\|}$

▷ **11.2—Exercise 3.** Consider the circle of radius r, $\alpha(t) = (r\cos(t), r\sin(t))$, with $I = [0, 2\pi]$. Show that $s(t) = rt$, so that $t(s) = s/r$, and deduce that reparameterization by arclength gives $\tilde{\alpha}(s) = (r\cos(s/r), r\sin(sr))$, and $\overrightarrow{t}(s) = (-\sin(s/r), \cos(s/r))$.

What is the Curvature of a Curve?

How should we define the "curvature" of a curve? For a plane curve, $\alpha(s) = (x(s), y(s))$

there is a natural and intuitive definition—namely the rate at which its unit tangent vector $\alpha'(s) = (x'(s), y'(s))$ is "turning". Now since $\alpha'(s)$ is a unit vector, we can write it as $\alpha'(s) = (\cos(\theta(s)), \sin(\theta(s)))$ where $\theta(s)$ is the angle $\alpha'(s)$ makes with the x-axis. Thus we can define the curvature $k(s)$ of α at s as $\theta'(s)$—the rate of change of this angle with respect to arclength. Notice that $\alpha''(s) = \theta'(s)(-\sin((\theta(s)), \cos((\theta(s))) = k(s)R_{\frac{\pi}{2}}\overrightarrow{t}(s) = k(s)\overrightarrow{n}(s)$, so we make the following definition:

11.2.4 Definition. If α is a curve in $\mathbf{R}^2$ that is parametrized by arclength, then its *curvature* at $\alpha(s)$ is defined to be the scalar $k(s)$ such that $\alpha''(s) = k(s)\overrightarrow{n}(s)$.

Note that in the plane, $\mathbf{R}^2$, the curvature $k(s)$ can be either positive or negative. Its absolute value is of course given by $|k(s)| = \|\alpha''(s)\|$.

11.2.5 Definition. If α is a curve in $\mathbf{R}^n$, $n > 2$ that is parametrized by arclength, then its *curvature* at $\alpha(s)$ is defined to be $k(s) := \|\alpha''(s)\|$.

▷ **11.2—Exercise 4.** Show that if $\alpha(t) = (x(t), y(t))$ is a plane parametrized curve that is not necessarily parametrized by arclength, then its curvature at $\alpha(t)$ is given by the following formula. Hint: $\theta = \tan^{-1}(y'/x')$.

$$\frac{x'(t)y''(t) - y'(t)x''(t)}{\left(x'(t)^2 + y'(t)^2\right)^{\frac{3}{2}}}.$$

11.2.6 Remark. Recall that when $n > 2$ at points where $k(s) := \|\alpha''(s)\| > 0$ the normal $\overrightarrow{n}(s)$ was defined by by $\overrightarrow{n}(s) := \frac{\alpha''(s)}{\|\alpha''(s)\|}$, so the equality $\alpha''(s) = k(s)\overrightarrow{n}(s)$ holds in this case too. But note the subtle difference; for a plane curve the curvature can be positive or negative, while in higher dimensions it is (by definition) always positive.

▷ **11.2—Exercise 5.** Show that a straight line has curvature zero, and that a circle of radius r has constant curvature $1/r$.

▷ **11.2—Exercise 6.** Show that straight lines are the only curves with zero curvature, but show that curves with positive constant curvature are not necessarily circles. (Hint: Show that a helix has constant curvature.) However, in $\mathbf{R}^2$, show that a curve of constant positive curvature k must be a circle of radius $1/k$.

Osculating Circles and Evolutes

At any point $\alpha(s)$ of a plane curve α there are clearly circles of any radius that are tangent to α at this point. In fact, just move out a distance r from $\alpha(s)$ along the normal $\overrightarrow{n}(s)$ and construct the circle with radius r centered at that point. However there is one special tangent circle that is "more tangent" than the others, in the sense that it has "second order contact" with α. This is the so-called *osculating circle* at $\alpha(s)$ and is the circle having the same curvature as α at this point, namely $k(s)$. Recalling that a circle of radius r has curvature $1/r$, we see that the radius of the osculating circle is $1/k(s)$, and its center, called the *center of curvature* of α at $\alpha(s)$ is clearly the point $c(s) := \alpha(s) - (1/k(s))\overrightarrow{n}(s)$.

As the point $\alpha(s)$ varies over the curve α, the corresponding centers of curvature trace out a curve called the *evolute* of the original curve α.

11.3 The Fundamental Theorem of Plane Curves

Recall that $\boldsymbol{Euc}(\mathbf{R}^n)$ denotes the Euclidean group of $\mathbf{R}^n$, i.e., all the distance preserving maps of $\mathbf{R}^n$ to itself. We have seen that every element of $\boldsymbol{Euc}(\mathbf{R}^n)$ can be written as the composition of a translation and an orthogonal transformation.

11.3.1 Definition. Two parametric curves α_1 and α_2 in $\mathbf{R}^n$ are called *congruent* if there is an element $g \in \boldsymbol{Euc}(n)$ such that $\alpha_2 = g \circ \alpha_1$.

11.3.2 Proposition. *The curvature function of a parametrized curve is invariant under congruence. That is, if two parametrized curves in $\mathbf{R}^n$ are congruent, then their curvature functions are identical.*

▷ **11.3—Exercise 1.** Prove this Proposition.

It is a remarkable fact that for plane curves the converse is true, so remarkable that it goes by the name The Fundamental Theorem of Plane Curves. The Fundamental Theorem actually says more—any continuous function $k(s)$ is the curvature function for some parametrized plane curve, and this curve is uniquely determined up to congruence. In fact, we will get an explicit formula below for the curve in terms of k.

11.3.3 Proposition. *Let $k : [0, L] \to \mathbf{R}$ be continuous and let $\alpha(s) = (x(t), y(t))$ be a parametrized plane curve that is parametrized by arclength. A neccessary and sufficient condition for α to have k as its curvature function is that θ, x, y be a solution on $[0, L]$ of the following system of first order ODE:*

$$\begin{aligned} \frac{d\theta}{ds} &= k(s), \\ \frac{dx}{ds} &= \cos(\theta), \\ \frac{dy}{ds} &= \sin(\theta). \end{aligned}$$

▷ **11.3—Exercise 2.** Prove this Proposition.

The first ODE of the above system integrates immediately to give $\theta(\sigma) = \theta_0 + \int_0^\sigma k(\tau)\, d\tau$. If we now substitute this into each of the remaining equations and integrate again we obtain the following important corollary.

11.3.4 Corollary. *If $k : [0, L] \to \mathbf{R}$ is a continuous function, then the set of plane curves $\alpha(s) = (x(s), y(s))$ that are parametrized by arclength and have k as their curvature*

function are just those of the form:

$$x(s) := x_0 + \int_0^s \cos\left(\theta_0 + \int_0^\sigma k(\tau)\, d\tau\right),$$
$$y(s) := y_0 + \int_0^s \sin\left(\theta_0 + \int_0^\sigma k(\tau)\, d\tau\right).$$

▷ **11.3—Exercise 3.** Use this corollary to rederive the fact that straight lines and circles are the only plane curves with constant curvature.

11.3.5 Remark. Note the geometric meaning of the constants of integration x_0, y_0 and θ_0. The initial point $\alpha(0)$ of the curve α is (x_0, y_0),while θ_0 is the angle that the initial tangent direction $\alpha'(0)$ makes with the x-axis. If in particular we take all three constants to be zero and call the resulting curve α_0, then we get the general solution from α_0 by first rotating by θ_0 and then translating by (x_0, y_0). This proves:

11.3.6 Fundamental Theorem of Plane Curves. *Two plane curves are congruent if and only if they have the same curvature function. Moreover any continuous function* $k : [0, L] \to \mathbf{R}$ *can be realized as the curvature function of a plane curve.*

Why the Fancy Name?

"Fundamental Theorem" sounds rather imposing— what's the big deal? Well, if you think about it, we have made remarkable progress in our understanding of curve theory in the past few pages—progress that actually required many years of hard work—and that progress is summed up in the Fundamental Theorem. There are two major insights involved in this progress. The first is that, from the viewpoint of geometry, we should consider curves that differ by parameterization as "the same", and that we can avoid the ambiguity of description this involves by choosing parameterization by arclength. (The lack of any analogous "canonical parameterization" for a surface will make our study of surface thory considerably more complicated.) The second insight is that, from the geometric viewpoiint again, congruent curves should also be regarded as "the same", and if we accept this then the simplest geometric description of a plane curve—one that avoids all redundency—is just its curvature function.

11.4 Sixth Matlab Project.

The Matlab project below is concerned in part with the visualization and animation of curves. Before getting into the details of the project, I would like to make a few general remarks on the subject of mathematical visualization that you should keep in mind while working on this project—or for that matter when you have any programming task that involves visualization and animation of mathematical objects.

1) How should you choose an error tolerance?

First, an important principle concerning the handling of errors in any computer graphics context. Books on numerical analysis tell you how to estimate errors and how to keep

them below a certain tolerance, but they cannot tell you what that tolerance should be—that must depend on how the numbers are going to be used. Beginners often assume they should aim for the highest accuracy their programming system can provide—for example fourteen decimal places for Matlab. But that will often be far more than is required for the task at hand, and as you have already seen, certain algorithms may require a very long time to attain that accuracy. The degree of one's patience hardly seems to be the best way to go about choosing an error tolerance.

In fact, there is often is a more rational way to choose appropriate error bounds. For example, in financial calculations it makes no sense to compute values with an error less than half the smallest denomination of the monetary unit involved. And when making physical calculations, it is useless to calculate to an accuracy much greater than can be measured with the most precise measuring instruments available. Similarly, in carpentry there is little point to calculating the length of a board to a tolerance less than the width of the blade that will make the cut.

This same principle governs in mathematical visualization. My approach is to choose a tolerance that is "about half a pixel", since any higher accuracy won't be visible anyway. It is usually fairly easy to estimate the size of a pixel. There are roughly 100 pixels per inch, so for example if you are are graphing in a six inch square window, and the axes go from minus one to one, then six hundred pixels equals two length units, so half a pixel accuracy means a tolerance of $\frac{1}{600}$ or roughly 0.002.

1) How should you represent a curve?

Mathematically a curve in $\mathbf{R}^n$ is given by a map of an interval $[a, b]$ into $\mathbf{R}^n$. We can only represent the curve on a computer screen when $n = 2$ or $n = 3$. Let's consider the case of plane curves ($n = 2$) first. If $\alpha(t) = (x(t), y(t))$ then for any N we can divide the interval $[a, b]$ into N equal subintervals of length $h = \frac{b-a}{N}$, namely $[t_k, t_{k+1}]$, where $t_k = a + kh$ and $k = 0, \ldots, N-1$. We associate to α and N an approximating "N-gon" α_N (i.e., a polygon with N sides) with vertices $v_k := (x(t_k), y(t_k))$. It is some α_N with N suitably large) that actually gets drawn on the computer screen when we want to display α. This reduces the actual drawing problem to that of drawing a straight line segment, and the latter is of course built into every computer system at a very low level.

In Matlab the code for plotting the curve α, or rather the polygon α_{30} would be:

```
N = 30
h = (b-a)/N;
t = a:h:b ;
plot(x(t),y(t)), axis equal;
```

To plot a curve $\alpha(t) = (x(t), y(t), z(t))$ in $\mathbf{R}^3$ is really no more difficult. In Matlab the only change is that the last line gets replaced by:

```
plot3(x(t),y(t),z(t)), axis equal;
```

only now one has to be more careful about interpreting just what it is that one sees on the screen in this case. The answer is that one again is seeing a certain polygon in the plane, but now it is the projection of the polygon in $\mathbf{R}^3$ with vertices at $v_k := (x(t_k), y(t_k), z(t_k))$. (The projection can be chosen to be either an orthographic projection in some direction

or else a perspective projection from some point.)

1) How do you create animations?

Viisualization can be a powerful tool for gaining insight into the nature of complex mathematical objects, and frequently those insights can be further enhanced by careful use of animation. Remember that time is essentially another dimension, so animations allow us to pack a lot more information onto a computer screen in a format that the human brain can easily assimilate. The number of ways that animation can be used are far to numerous to catalog here, but in addition to obvious ones, such as rotating a three dimensional object, one should also mention "morphing". Mathematical objects frequently depend on several parameters (e.g., think of the family of ellipses: $x = a\cos(\theta)$, $y = b\sin(\theta)$). Morphing refers to moving along a curve in the space of parameters and creating frames of an animation as you go.

All animation techniques use the same basic technique—namely showing a succession of "frames" on the screen in rapid succession. If the number of frames per second is fast enough, and the change between frames is small enough, then the phenomenon of "persistence of vision" creates the illusion that one is seeing a continuous process evolve. Computer games have become very popular in recent years, and they depend so heavily on high quality animation that the video hardware in personal computers has improved very rapidly. Still, there are many different methods (and tricks) involved in creating good animations, and rather than try to cover them here we will have some special lectures on various animation techniques, with particular emphasis on how to implement these techniques in Matlab.

Matlab Project # 6.

Your assignment for the sixth project is to implement the Fundamental Theorem of Plane Curves using Matlab. That is, given a curvature function $k : [0, L] \to \mathbf{R}$, construct and plot a plane curve $x : [0, L] \to \mathbf{R}^2$ that has k as its curvature function. To make the solution unique, take the initial point of x to be the origin and its initial tangent direction to be the direction of the positive x-axis. You should also put in an option to plot the evolute of the curve as well as the curve itself. Finally see if you can build an animation that plots the osculating circle at a point that moves along the curve x. For uniformity, name your M-File PlaneCurveFT, and let it start out:

```
function     x = PlaneCurveFT(k,L,option)
```

If option is not given (i.e., nargin = 2) or if option = 0, then just plot the curve x. If option = 1, then plot x and, after a pause, plot its evolute in red. Finally, if option = 2, then plot x and its evolute, and then animate the osculating circle (in blue) along the curve, also drawing the radius from the center of curvature.

[To find the curve x, you first integrate k to get $\overrightarrow{t} = x'$, and then integrate $\overrightarrow{t}$. The curvature, k, will be given as a Matlab function, so you can use the version of Simpson's Rule previously discussed for the first integration. But $\overrightarrow{t}$ will not be in the form of a Matlab function that you can substitute into that version of Simpson's Rule, so you will need to develop a slightly modified version of Simpson's. where the input is a matrix that gives the values of the integrand at the nodes rather than the integrand as a function.]

Lecture 12
Curves in 3-Space

Some Definitions and Notations. In this section $\alpha : [0, L] \to \mathbf{R}^3$ will denote a curve in $\mathbf{R}^3$ that is parametrized by arclength and is of class C^k, $k \geq 2$. We recall that the unit tangent to α at s is defined by $\overrightarrow{t}(s) := \alpha'(s)$. (That it is a unit vector is just the definition of α being parametrized by arclength.) The *curvature* of α is the non-negative real-valued function $k : [0, L] \to \mathbf{R}$ defined by $k(s) := \|\alpha''(s)\|$, and we call α a *Frenet curve* if its curvature function is strictly positive. For a Frenet curve, we define its normal vector $\overrightarrow{n}(s)$at s by $\overrightarrow{n}(s) := \frac{\alpha''(s)}{k(s)}$. Since $\overrightarrow{t}(s)$ has constant length one, it is orthogonal to its derivative $\overrightarrow{t}'(s) = \alpha''(s)$, hence:

12.0.1 First Frenet Equation. *If $\alpha : [0, L] \to \mathbf{R}^3$ is a Frenet curve then its normal vector $\overrightarrow{n}(s)$ is a unit vector that is orthogonal to $\overrightarrow{t}(s)$, and $\overrightarrow{t}'(s) = k(s)\overrightarrow{n}(s)$.*

For the remainder of this section we will assume that α is a Frenet curve.

12.0.2 Definition. We define the *binormal vector* to α at s by $\overrightarrow{b}(s) := \overrightarrow{t}(s) \times \overrightarrow{n}(s)$. The ordered triple of unit vectors $f(s) := (\overrightarrow{t}(s), \overrightarrow{n}(s), \overrightarrow{b}(s))$ is called the *Frenet frame* of α at $\alpha(s)$, and the mapping $s \mapsto f(s)$ of $[0, L] \to \left(\mathbf{R}^3\right)^3$ is called the *Frenet framing* of the curve α. The plane spanned by $\overrightarrow{t}(s)$ and $\overrightarrow{n}(s)$ is called the *osculating plane* to α at s, and the plane spanned by $\overrightarrow{n}(s)$ and $\overrightarrow{b}(s)$ is called the *normal plane* to α at $\alpha(s)$.

12.1 Quick Review of the Vector Product

We recall that for $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$ in $\mathbf{R}^3$, we define their *vector-product* $u \times v \in \mathbf{R}^3$ by $u \times v := (u_2v_3 - v_2u_3, u_3v_1 - u_1v_3, u_1v_2 - v_1u_2)$.

12.1.1 Remark. Symbolically we can write this definition as:

$$u \times v = \det \begin{pmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ e_1 & e_2 & e_3 \end{pmatrix},$$

where e_1, e_2, e_3 is the standard basis for $\mathbf{R}^3$. What this means is that we get a correct result if we use Cramer's rule to expand this "determinant" by minors of the third row. From this formula the following facts are immediate:

12.1.2 Proposition. *The vector product is bilinear and skew-symmetric. That, is it is linear in each argument and changes sign when the arguments are interchanged. The vector product of u and v vanishes if and only if u and v are linearly dependent, and $(u \times v) \cdot w$ is the determinant of the matrix with rows u,v, and w. In particular, $(u \times v) \cdot u$ and $(u \times v) \cdot v$ are both zero, i.e., $u \times v$ is orthogonal to both u and v.*

12.1.3 Lemma. $(u \times v) \cdot (x \times y) = \det \begin{pmatrix} u \cdot x & v \cdot x \\ u \cdot y & v \cdot y \end{pmatrix}$.

PROOF. Either direct verification, or use linearity in all four variables to reduce the result to the case that all of the variables are equal to one of the standard orthonormal basis vectors e_i, in which case it is obvious.

12.1.4 Proposition. *The norm of $u \times v$ is the area of the parallelogram spanned by u and v. That is, if θ is the angle between u and v then $\|u \times v\| = \|u\| \, \|v\| \sin(\theta)$.*

PROOF. $\|u \times v\|^2 = (u \times v) \cdot (u \times v) = \det \begin{pmatrix} u \cdot u & v \cdot u \\ u \cdot v & v \cdot v \end{pmatrix} = \|u\|^2 \|v\|^2 - \|u\|^2 \|v\|^2 \cos^2(\theta)$ $= \|u\|^2 \|v\|^2 (1 - \cos^2(\theta))$. ∎

▷ **12.1—Exercise 1.** Show that the "triple vector product" $(u \times v) \times w$ is given by the formula $(u \cdot w)v - (v \cdot w)u$. Hint. Since it is orthogonal to $u \times w$ it must be of the form $f(w)v + g(w)u$, where, since the result is linear in w, f and g have the form $a \cdot w$ and $b \cdot w$, so we have $(u \times v) \times w = (a \cdot w)v + (b \cdot w)u$, for some a and b.

▷ **12.1—Exercise 2.** If $u(t)$ and $v(t)$ are smooth curves in $\mathbf{R}^3$, show that $(u(t) \times v(t))' = u'(t) \times v(t) + (u(t) \times v'(t)$

12.2 The Frenet Formulas

We now return to the Frenet Frame $\vec{t}(s), \vec{n}(s), \vec{b}(s)$ for α. Recall that the binormal vector was defined by $\vec{b}(s) := \vec{t}(s) \times \vec{n}(s)$. In particular, as the vector product of orthogonal unit vectors it is a unit vector and so its derivative orthogonal $\vec{b}'(s)$ is orthogonal to $\vec{b}(s)$ and so it is a linear combination of $\vec{t}(s)$ and $\vec{n}(s)$. But in fact $\vec{b}'(s)$ is orthogonal to $\vec{t}(s)$ also, and hence it is a multiple of $\vec{n}(s)$. To see this we calculate:

$$\begin{aligned} \vec{b}'(s) &= (\vec{t}(s) \times \vec{n}(s))' \\ &= \vec{t}'(s) \times \vec{n}(s) + \vec{t}(s) \times \vec{n}'(s) \\ &= k(s)\vec{n}(s) \times \vec{n}(s) + \vec{t}(s) \times \vec{n}'(s) \\ &= \vec{t}(s) \times \vec{n}'(s) \end{aligned}$$

since $\vec{n}(s) \times \vec{n}(s) = 0$. Thus $\vec{b}'(s)$ is orthogonal to both $\vec{b}(s)$ and $\vec{t}(s)$ and so:

12.2.1 Proposition. $\vec{b}'(s)$ *is a multiple of* $\vec{n}(s)$.

12.2.2 Definition. We define the *torsion* of α at s to be the real number $\tau(s)$ such the $\vec{b}'(s) = \tau(s)\vec{n}(s)$.

12.2.3 Remark. Since $\vec{b}(s)$ is the normal to the osculating plane to the curve α at s, the torsion measures the rate at which the curve is twisting out of its osculating plane. Note for

example, that if $\tau(s)$ is identically zero, then $\overrightarrow{b}(s)$ is a constant b_0, and so the osculating plane is fixed, and it follows easily that $\alpha(s)$ lies in a plane parallel to its osculating plane. In fact, since $(\alpha(s)\cdot b_0)' = \overrightarrow{t}(s)\cdot b_0 = 0$, $(\alpha(s)-\alpha(0))\cdot b_0 = 0$, which says that α lies in the plane Π orthogonal to b_0 that contains $\alpha(0)$.

Now that we have computed $\overrightarrow{t}'(s)$ and $\overrightarrow{b}'(s)$, it is easy to compute $\overrightarrow{n}'(s)$. In fact, since $\overrightarrow{b}(s) = \overrightarrow{t}(s)\times\overrightarrow{n}(s)$, it follows that $n(s) = b(s)\times t(s)$, so

$$\begin{aligned}\overrightarrow{n}'(s) &= (\overrightarrow{b}(s)\times\overrightarrow{t}(s))'\\ &= \overrightarrow{b}(s)\times\overrightarrow{t}'(s) + \overrightarrow{b}'(s)\times\overrightarrow{t}(s)\\ &= \overrightarrow{b}(s)\times(k(s)n(s)) + \tau(s)\overrightarrow{n}(s)\times\overrightarrow{t}(s)\\ &= -k(s)\overrightarrow{t}(s) - \tau(s)\overrightarrow{b}(s)\end{aligned}$$

The equations that express the derivative of the Frenet frame in terms of the frame itself are referred to as the Frenet Equations. Let's rewrite them as a single matrix equation:

Frenet Equations

$$\begin{pmatrix}\overrightarrow{t}(s)\\ \overrightarrow{n}(s)\\ \overrightarrow{b}(s)\end{pmatrix}' = \begin{pmatrix}0 & k(s) & 0\\ -k(s) & 0 & -\tau(s)\\ 0 & \tau(s) & 0\end{pmatrix}\begin{pmatrix}\overrightarrow{t}(s)\\ \overrightarrow{n}(s)\\ \overrightarrow{b}(s)\end{pmatrix}$$

12.2.4 Remark. We can write this symbolicly as $f' = Af$ where $f = (\overrightarrow{t}(s), \overrightarrow{n}(s), \overrightarrow{b}(s))$ is the Frenet frame, and A is a 3×3 matrix with entries 0, $\pm k(s)$ and $\pm\tau(s)$. The fact that this matrix is skew-symmetric is, as we shall see next is just a reflection of the fact that $(\overrightarrow{t}(s), \overrightarrow{n}(s), \overrightarrow{b}(s))$ are orthonormal.

12.2.5 Proposition. *Let $f_i : [0,L]\to\mathbf{R}^n$, $i = 1,\dots,n$, be C^1 maps and suppose that the $f_i(t)$ are orthonormal for all t. Let $a_{ij}(t)$ be the $n\times n$ matrix that expresses the $f_i'(t)$ as a linear combination of the $f_i(t)$, i.e., $f_i'(t) = \sum_{j=1}^n a_{ij}(t)f_j(t)$ (so that $a_{ij}(t) = f_i'(t)\cdot f_j(t)$). Then $a_{ij}(t)$ is skew-symmetric. Conversely, if $t\mapsto a_{ij}(t)$ is a continuous map of $[0,L]$ into the skew-adjoint $n\times n$ matrices and $\phi_1,\dots,\phi_n$ is any orthonormal frame for $\mathbf{R}^n$, then there are unique differentiable maps $f_i : [0,L]\to\mathbf{R}^n$ such that $f_i(0) = \phi_i$ and $f_i'(t) = \sum_{j=1}^n a_{ij}(t)f_j(t)$, and these $f_i(t)$ are orthonormal for all t.*

PROOF. Differentiating $f_i(t)\cdot f_j(t) = \delta_{ij}$ gives $f_i'(t)\cdot f_j(t) + f_i(t)\cdot f_j'(t) = 0$ proving the first statement. Conversely, if we are given $a_{ij}(t)$ and the ϕ_i, then by the existence and uniqueness theorem for ODE, we can solve the IVP $f_i'(t) = \sum_{j=1}^n a_{ij}(t)f_j(t)$ and $f_i(0) = \phi_i$ uniquely on the interval $[0,L]$, and we only have to show that $s_{ij}(t) := f_i(t)\cdot f_j(t)$ is identically equal to δ_{ij}. We note that $s_{ij}(t)$ satisfies the IVP $s_{ij}(0) = \delta_{ij}$ and

$$\begin{aligned}\frac{d}{dt}\left(s_{ij}(t)\right) &= f_i'(t)\cdot f_j(t) + f_i(t)\cdot f_j'(t)\\ &= \sum_{k=1}^{n} a_{ik}(t) f_k(t)\cdot f_j(t) + \sum_{k=1}^{n} a_{jk}(t) f_i(t) f_k(t)\\ &= \sum_{k=1}^{n} a_{ik}(t) s_{kj}(t) + \sum_{k=1}^{n} a_{jk}(t) s_{ij}(t)\end{aligned}$$

Since the $a_{ij}(t)$ are skew symmetric it is clear that $s_{ij}(t) = \delta_{ij}$ is also a solution, so the converse follows from the uniquenes of the solution to this IVP. ∎

12.3 The Fundamental Theorem of Space Curves

We are now in a position to generalize our theory of plane curves to an analogous theory of space curves. We first make a number of simple observations.

Observation 1. The Frenet framing associated to a space curve is invariant under orthogonal transformations in the sense that if α and $\tilde{\alpha}$ are space curves and g is an element of the orthogonal group $\boldsymbol{O}(\mathbf{R}^3)$ such that $\tilde{\alpha} = g \circ \alpha$, then g maps the Frenet frame of α at s to the Frenet frame of $\tilde{\alpha}$ at s.

Observation 2. The curvature and torsion functions of a plane curve are likewise invariant under orthogonal transformation and also under translation.

▷ **12.3—Exercise 1.** Prove the validity of these two observations. Hint; They depend on little more than the definitions of the quantities involved and the fact that orthogonal transformations preserve lengths of curves.

We now in a position to prove the following analogue of the Fundamental Theorem of Plane Curves.

Fundamental Theorem of Space Curves. *Two space curves are congruent if and only if they have the same curvature and torsion functions. Moreover any pair of continuous functions $k : [0, L] \to (0, \infty)$ and $\tau : [0, L] \to \mathbf{R}$ can be realized as the curvature and torsion functions of some space curve.*

PROOF. Given k and τ, it follows from the preceding Proposition that we can find $\mathbf{R}^3$ valued functions $\overrightarrow{t}(t), \overrightarrow{n}(t), \overrightarrow{b}(t)$ defined on $[0, L]$ that are orthonormal and satisfy the Frenet Equations. Then, just as in the planar case, we can define a curve $\alpha(s) := \int_0^s \overrightarrow{t}(t)\, dt$. Since $\overrightarrow{t}(t)$ is a unit vector, α is parametrized by arclength and $\overrightarrow{t}$ is clearly its unit tangent vector, so it is a consequence of the Frenet Equations that $\overrightarrow{n}(t)$, and $\overrightarrow{b}(t)$ are its normal and binormal and k and τ its curvature and torsion.

Now suppose $\tilde{\alpha}$ is a second curve with the same curvature and torsion as α. If we translate α by $\tilde{\alpha}(0) - \alpha(0)$ and then rotate it by the rotation carrying the Frenet frames of α at 0 into that of $\tilde{\alpha}$ at 0, then we get a curve congruent to α that has the same initial point and

initial Frenet frame as $\tilde{\alpha}$, so it will sufice to show that if α and $\tilde{\alpha}$ have the *same* initial point and Frenet frame, then they are identical. But from the uniqueness of the solution of the IVP for the Frenet equations it now follows that α and $\tilde{\alpha}$ have the same Frenet frame at all $s \in [0, L]$, and it the follows that both $\alpha(s)$ and $\tilde{\alpha}(s)$ equal $\int_0^s \overrightarrow{t}(t)\, dt$. ∎

Matlab Project # 7.

As you have probably guessed, your assignment for the seventh Matlab project is to implement the Fundamental Theorem of Space Curves. That is, given a (positive) curvature function $k : [0, L] \to \mathbf{R}$, and a torsion function $\tau : [0, L] \to \mathbf{R}$, construct and plot a space curve x that has k as its curvature function and τ as its torsion function. To make the solution unique, take the initial point of x to be the origin and its initial tangent direction to be the direction of the positive x-axis. You should also use plot3 to plot the curve. See if you can create an animation that moves the Frenet Frame along the curve. For uniformity, name your M-File SpaceCurveFT, and let it start out:

```
function       x = SpaceCurveFT(k,tau,L)
```

12.4 Surface Theory: Basic Definitions and Examples

The theory of curves in the plane, three-space, and higher dimensions is deep and rich in detail, and we have barely scratched the surface. However I would like to save enough time to cover at least the basics of surface theory, so we will now leave the theory of curves.

How should we define a surface? As with curves, there are many possible answers, and we will select not the most general definition but one that is both intuitive and leads quickly to a good theory. The simplest curve is just an interval in the line, and we defined other curves to be maps of an interval into $\mathbf{R}^n$ with non-vanishing derivative, The natural two dimensional analog of an interval is a connected open set or *domain* $\mathcal{O}$ in $\mathbf{R}^2$.

12.4.1 Definition. A C^k *parametric surface* in $\mathbf{R}^3$ ($k \geq 3$) is a C^k map $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$ (where $\mathcal{O}$ is a domain in $\mathbf{R}^2$) such that its differential, $D\mathcal{F}_p$, has rank 2 at all points $p \in \mathcal{O}$.

Of course, intuitively speaking, it is the image of $\mathcal{F}$ that constitutes the surface, but as with curves we will allow ourselves a looseness of language and not always distinguish carefully between $\mathcal{F}$ and its image.

Notation. Just as it is traditional to use t as the parameter of a curve (or s if the parameter is arclength), it is traditional to use u and v to denote the parameters of a parametric surface, so a surface is given by a mapping $(u, v) \mapsto \mathcal{F}(u, v) = (\mathcal{F}_1(u, v), \mathcal{F}_2(u, v), \mathcal{F}_2(u, v))$. Instead of the $\mathcal{F}_i(u, v)$ it is also traditional to use $(x(u, v).y(u.v), z(u, v))$ to denote the three components of $\mathcal{F}(u, v)$, and we will often use this notation without explicit mention when it is clear what surface is under consideration.

▷ **12.4—Exercise 1.** Show that if $p_0 = (u_0, v_0)$, then the condition that $D\mathcal{F}_p$ has rank two is equuivalent to $\frac{\partial \mathcal{F}(u_0,v_0)}{\partial u}$ and $\frac{\partial \mathcal{F}(u_0,v_0)}{\partial v}$ being linearly independent.

12.4.2 Definition. If $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$ is a parametric surface in $\mathbf{R}^3$ and $p \in \mathcal{O}$, the *tangent space to $\mathcal{F}$ at p* is defined to be the image of the linear map $D\mathcal{F}_p : \mathbf{R}^2 \to \mathbf{R}^3$, and we denote it by $T\mathcal{F}_p$. We note that $T\mathcal{F}_p$ is by assumption a two-dimensional linear subspace of $\mathbf{R}^3$ and that $\frac{\partial \mathcal{F}(u_0,v_0)}{\partial u}$ and $\frac{\partial \mathcal{F}(u_0,v_0)}{\partial v}$ is clearly a basis. This is called the basis for $T\mathcal{F}_p$ defined by the parameters u, v. We define the *normal vector to $\mathcal{F}$ at p* to be the unit vector $\overrightarrow{\nu}(p)$ obtained by normalizing $\frac{\partial \mathcal{F}(u_0,v_0)}{\partial u} \times \frac{\partial \mathcal{F}(u_0,v_0)}{\partial v}$. The map $\overrightarrow{\nu} : \mathcal{O} \to \mathbf{S}^2$, $p \mapsto \overrightarrow{\nu}(p)$ of $\mathcal{O}$ to the unit sphere $\mathbf{S}^2 \subseteq \mathbf{R}^3$ is called the *Gauss map* of the surface $\mathcal{F}$.

12.4.3 Remark. You will probably guess that the "curvature" of the surface $\mathcal{F}$ (whatever it means) will somehow be measured by the rate at which the normal $\overrightarrow{\nu}(p)$ varies with p.

▷ **12.4—Exercise 2.** Show that the tangent space to $\mathcal{F}$ at p and the tangent space to $\mathbf{S}^2$ at $\overrightarrow{\nu}(p)$ are the same.

12.4.4 Remark. It is natural to try to use what we have learned about curves to help us investigate surfaces, and this approach turns out to be very effective. It $(u(t), v(t))$ is a smooth parametric curve in the domain $\mathcal{O}$ of the of the surface $\mathcal{F}$, then $\alpha(t) := \mathcal{F}(u(t), v(t))$ is a parametric curve in $\mathbf{R}^3$, and we shall call such curve a parametric curve on the surface $\mathcal{F}$. If we put $u_0 = u(t_0)$, $v_0 = v(t_0)$ and $p = (u_0, v_0)$, then the by the chain-rule, the tangent vector, $\alpha'(t_0)$, to $\alpha(t)$ at t_0 is $D\mathcal{F}_p(u'(t_0), v'(t_0))$, an element of the tangent space $T\mathcal{F}_p$ to $\mathcal{F}$ at p. In terms of the basis defined by the parameters u, v we have $\alpha'(t_0) = u'(t_0)\frac{\partial \mathcal{F}(u_0,v_0)}{\partial u} + v'(t_0)\frac{\partial \mathcal{F}(u_0,v_0)}{\partial v}$.

▷ **12.4—Exercise 3.** Show that every element of $T\mathcal{F}_p$ is the tangent vector to some curve on the surface $\mathcal{F}$ as above.

12.4.5 Remark. There is a special two-parameter "net" of curves on a surface $\mathcal{F}$ defined by taking the images of the straight lines in the domain $\mathcal{O}$ that are parallel to the u, v axes. Through each point $p_0 = (u_0, v_0)$ there are two such lines, $t \mapsto (u_0 + t, v_0)$ and $t \mapsto (u_0, v_0 + t)$, called the u-gridline through p and the v-gridline through p, and to visualize the surface, one plots the images under $\mathcal{F}$ of a more or less dense collection of these "gridlines".

▷ **12.4—Exercise 4.** Show that the tangent vectors to the u and v gridlines through p are just the elements of the basis for $T\mathcal{F}_p$ defined by the parameters u, v.

There are two types of surfaces that everyone learns about early in their mathematical training—graphs and surfaces of revolution.

12.4—Example 1. Graphs of Functions. Given a real-valued function $f : \mathcal{O} \to \mathbf{R}$ we get a parametric surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$ called the graph of f by $\mathcal{F}(u, v) := (u, v, f(u, v))$.

12.4—Example 2. Surfaces of Revolution. Let $t \mapsto \alpha(t) = (x(t), z(t))$ be a curve in the x, z-plane that does not meet the z-axis—i.e., $x(t) > 0$ for all t in the domain (a, b) of α, and let $\mathcal{O} = (0, 2\pi) \times (a, b)$. We define a surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$, called the surface of revolution (about the z-axis) defined from the curve α, by $\mathcal{F}(u, v) := (x(v)\cos(u), x(v)\sin(u), z(v))$.

Lecture 13
The Fundamental Forms of a Surface

In the following we denote by $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$ a parametric surface in $\mathbf{R}^3$, $\mathcal{F}(u,v) = (x(u,v), y(u,v), z(u,v))$. We denote partial derivatives with respect to the parameters u and v by subscripts: $\mathcal{F}_u := \frac{\partial \mathcal{F}}{\partial u}$ and $\mathcal{F}_v := \frac{\partial \mathcal{F}}{\partial u}$, and similarly for higher order derivative. We recall that if $p = (u_0, v_0) \in \mathcal{O}$ then $\mathcal{F}_u(p)$ and $\mathcal{F}_v(p)$ is a basis for $T\mathcal{F}_p$, the tangent space to $\mathcal{F}$ at p, the unit normal to $\mathcal{F}$ at p is $\overrightarrow{\nu}(p) := \frac{\mathcal{F}_u(p) \times \mathcal{F}_v(p)}{\|\mathcal{F}_u(p) \times \mathcal{F}_v(p)\|}$ and that we call the map $\overrightarrow{\nu} : \mathcal{O} \to \mathbf{S}^2$ the Gauss map of the surface $\mathcal{F}$.

13.1 Bilinear and Quadratic Forms

There are two important pieces of data associated to any surface, called its First and Second Fundamental Forms.

The First Fundamental Form encodes the "intrinsic data" about the surface—i.e., the information that you could discover by wandering around on the surface and making measurements **within** the surface.

The Second Fundamental Form on the other hand encodes the information about how the surface is embedded into the surrounding three dimensional space—explicitly it tells how the normal vector to the surface varies as one moves in different directions on the surface, so you could say it tells how the surface is curved in the embedding space.

These two "fundamental forms" are invariant under congruence, and moreover, they are a complete set of invariants for surfaces under congruence, meaning that if two surfaces have the same first and second fundamental forms then they are congruent. This latter fact is part of the Fundamental Theorem of Surfaces. But, it turns out that, unlike the curvature and torsion of a curve, not every **apparently** possible choice of First Fundamental Form and Second Fundamental Form for a surface can be realized by an actual surface. For this to be the case, the two forms must satisfy certain differential identities called the *Gauss-Codazzi Equations* and this fact is also part of the Fundamental Theorem of Surfaces.

Before considering the definitions of the fundamental forms on a surface, we make a short detour back into linear algebra to consider the general notions of bilinear and quadratic forms on a vector space.

13.1.1 Definition. Let V be a real vector space. A real-valued function $B : V \times V \to \mathbf{R}$ is called a *bilinear form* on V if it is linear in each variable separately when the other variable is held fixed. The bilinear form B is called *symmetric* (respectively *skew-symmetric*) if $B(v_1, v_2) = B(v_2, v_1)$ (respectively $B(v_1, v_2) = -B(v_2, v_1)$) for all $v_1, v_2 \in V$.

▷ **13.1—Exercise 1.** Show that every bilinear form on a vector space can be decomposed uniquely into the sum of a symmetric and a skew-symmetric bilinear form.

13.1.2 Definition. A real-valued function Q on a vector space V is called a *quadratic form* if it can be written in the form $Q(v) = B(v, v)$ for some symmetric bilinear form B on V. (We say that Q is *determined by* B.)

▷ **13.1—Exercise 2.** (Polarization Again.) Show that if Q is a quadratic form on V then the bilinear form B on V such that $Q(v) = B(v, v)$ is uniquely determined by the identity $B(v_1, v_2) = \frac{1}{2}(Q(v_1 + v_2) - Q(v_1) - Q(v_2))$.

Notation. Because of this bijective correspondence between quadratic forms and bilinear forms, it will be convenient to use the same symbol to denote them both. That is, if Q is a quadratic form then we shall also write Q for the bilinear form that determines it, so that $Q(v) = Q(v, v)$.

13.1.3 Remark. Suppose that V is an inner-product space. Then the inner product is a bilinear form on V and the quadratic form it determines is of course $Q(v) = \|v\|^2$. More generally, if $A : V \to V$ is any linear operator on V, then $B^A(v_1, v_2) = \langle Av_1, v_2\rangle$ is a bilinear form on V and B^A is symmetric (respectively, skew-symmetric) if and only if A is self-adjoint (respectively, skew-adjoint).

▷ **13.1—Exercise 3.** Show that any bilinear form on a finite dimensional inner-product space is of the form B^A for a unique choice of self-adjoint operator A on V, and hence any quadratic form on an inner-product space is of the form $Q^A(v) = \langle Av, v\rangle$ for a unique choice of self-adjoint operator A on V.

13.1.4 Remark. If B is a bilinear form on a vector space V and if $v_1, \ldots, v_n$ is a basis for V then the $b_{ij} = B(v_i, v_j)$ are called the matrix of coefficients of the form B in this basis. Clearly, if $u = \sum_i u_i v_i$ and $w = \sum_i w_i v_i$, than $B(u, w) = \sum_{ij} b_{ij} u_i w_j$. The bilinear form B is symmetric if and only if the matrix b_{ij} is symmetric, and in that case the quadratic form Q determined by B is $Q(u) = \sum_{ij} b_{ij} u_i u_j$.

13.1.5 Remark. Suppose that $T : V \to V$ is a self-adjoint operator on an inner-product space V, and that $v_1, \ldots, v_n$ is a basis for V. What is the relation between the matrix $b_{ij} = \langle Tv_i, v_j\rangle$ of the symmetric bilinear form B^T defined by T, and the matrix A of T in the basis $v_1, \ldots, v_n$? Your first guess may be that these two matrices are equal, however life is not quite that simple.

13.1.6 Proposition. *Let $T : V \to V$ be a self-adjoint operator on an inner-producr space V. If $b = (b_{ij})$ is the matrix of coefficients of the bilinear form B^T determined by T and A is the matrix of T, both with respect to the same basis $v_1, \ldots, v_n$ for V, then $A = g^{-1}b$, where g is the matrix of inner-products $g_{ij} = \langle v_i, v_j\rangle$, i.e., the matrix of coefficients of the bilinear form given by the inner-product.*

PROOF. By the definition of A, $Tv_i = \sum_{i=1}^n A_{ki} v_k$, hence $b_{ij} = \langle \sum_{i=1}^n A_{ki} v_k, v_j\rangle = A_{ki} g_{kj}$, i.e., $b = A^t g$, where A^t is the transpose of A, Hence $A^t = bg^{-1}$, and since b and g are symmetric, $A = (bg^{-1})^t = g^{-1}b$.

13.2 Quadratic Forms on a Surface

13.2.1 Definition. If $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$ is a parametric surface in $\mathbf{R}^3$, then a *quadratic form on* $\mathcal{F}$, Q, we mean a function $p \mapsto Q_p$ that assigns to each p in $\mathcal{O}$ a quadratic form Q_p on the tangent space $T\mathcal{F}_p$ of $\mathcal{F}$ at p.

13.2.2 Remark. Making use of the bases $\mathcal{F}_u(p), \mathcal{F}_v(p)$ in the $T\mathcal{F}_p$, a quadratic form Q on $\mathcal{F}$ is described by the symmetric 2×2 matrix of real-valued functions $Q_{ij} : \mathcal{O} \to \mathbf{R}$ defined by $Q_{ij}(p) := Q(\mathcal{F}_{x_i}(p), \mathcal{F}_{x_j}(p))$, (where $x_1 = u$ and $x_2 = v$). These three functions Q_{11}, Q_{12}, and Q_{22} on $\mathcal{O}$ determine the quadratic form Q on $\mathcal{F}$ uniquely: if $w \in T\mathcal{F}_p$, then $w = \xi\mathcal{F}_u(p) + \eta\mathcal{F}_v$, and $Q_p(w) = Q_{11}(p)\,\xi^2 + 2\,Q_{12}(p)\,\xi\,\eta + Q_{22}(p)\,\eta^2$. We call the Q_{ij} the *coefficients* of the quadratic form Q, and we say that Q is of class C^k if its three coefficients are C^k. Note that we can choose *any* three functions Q_{ij} and use the above formula for $Q_p(w)$ to define a unique quadratic form Q on $\mathcal{F}$ with these Q_{ij} as coefficients. This means that **we can identify quadratic forms on a surface with ordered triples of real-valued functions on its domain.**

Notation. Because of the preceding remark, it is convenient to have a simple way of referring to the quadratic form Q on a surface having the three coefficients A, B, C. There is a classical and standard notation for this, namely:

$$Q = A(u,v)\,du^2 + 2B(u,v)\,du\,dv + C(u,v)\,dv^2.$$

▷ **13.2—Exercise 1.** To see the reason for this notation—and better understand its meaning—consider a curve in $\mathcal{O}$ given parametrically by $t \mapsto (u(t), v(t))$, and the corresponding image curve $\alpha(t) := \mathcal{F}(u(t), v(t))$ on $\mathcal{F}$. Show that

$$Q(\alpha'(t)) = A(u(t),v(t))\left(\frac{du}{dt}\right)^2 + B(u(t),v(t))\left(\frac{du}{dt}\right)\left(\frac{dv}{dt}\right) + C(u(t),v(t))\left(\frac{dv}{dt}\right)^2.$$

The point is that curves on $\mathcal{F}$ are nearly always given in the form $t \mapsto \mathcal{F}(u(t), v(t))$, so a knowledge of the coefficients A, B, C as functions ot u, v is just what is needed in order to compute the values of the form on tangent vectors to such a curve from the parametric functions $u(t)$ and $v(t)$. As a first application we shall now develop a formula for the length of the curve α.

Definition of the First Fundamental Form of a Surface $\mathcal{F}$
Since $T\mathcal{F}_p$ is a linear subspace of $\mathbf{R}^3$ it becomes an inner-product space by using the restriction of the inner product on $\mathbf{R}^3$. Then the First Fundamental Form on $\mathcal{F}$, denoted by $I^{\mathcal{F}}$, is defined by $I^{\mathcal{F}}_p(w) := \|w\|^2$, and its coefficients are denoted by $E^{\mathcal{F}}, F^{\mathcal{F}}, G^{\mathcal{F}}$. When there is no chance of ambiguity we will omit the superscript from $I^{\mathcal{F}}$ and its coefficients. Thus:

$$I^{\mathcal{F}} = E^{\mathcal{F}}(u,v)\,du^2 + 2F^{\mathcal{F}}(u,v)\,du\,dv + G^{\mathcal{F}}(u,v)\,dv^2,$$

where the functions $E^{\mathcal{F}}$, $F^{\mathcal{F}}$, and $G^{\mathcal{F}}$ are defined by:

$$\begin{aligned} E^{\mathcal{F}} &:= \mathcal{F}_u \cdot \mathcal{F}_u = x_u^2 + y_u^2 + z_u^2, \\ F^{\mathcal{F}} &:= \mathcal{F}_u \cdot \mathcal{F}_v = x_u x_v + y_u y_v + z_u z_v, \\ G^{\mathcal{F}} &:= \mathcal{F}_v \cdot \mathcal{F}_v = x_v^2 + y_v^2 + z_v^2. \end{aligned}$$

The Length of a Curve on a Surface

Let $t \mapsto (u(t), v(t))$ be a parametric curve in $\mathcal{O}$ with domain $[a, b]$. By the above exercise, the length, L, of the curve $\alpha : t \mapsto \mathcal{F}(u(t), v(t))$ is:

$$\begin{aligned} L &= \int_a^b \sqrt{I(\alpha'(t))}\, dt \\ &= \int_a^b \sqrt{E(u(t),v(t))\left(\frac{du}{dt}\right)^2 + F(u(t),v(t))\left(\frac{du}{dt}\right)\left(\frac{dv}{dt}\right) + G(u(t),v(t))\left(\frac{dv}{dt}\right)^2}\, dt\,. \end{aligned}$$

Alternative Notations for the First Fundamental Form.
The First Fundamental Form of a surface is so important that there are several other standard notational conventions for referring to it. One whose origin should be obvious is to denote to it by ds^2, and call $ds = \sqrt{ds^2}$ the "line element" of the surface.

13.3 The Shape Operator and Second Fundamental Form

We next consider the differential $D\nu_p$ of the Gauss map $\nu : \mathcal{O} \to \mathbf{S}^2$ at a point p of $\mathcal{O}$. Strictly speaking it is a linear map of $\mathbf{R}^2 \to \mathbf{R}^3$, but as we shall now see it has a natural interpretation as a map of $T_p\mathcal{F}$ to itself. As such it plays a central role in the study of the extrinsic properties of $\mathcal{F}$ and is called the *shape operator* of $\mathcal{F}$ at p. Moreover, we shall also establish the important fact that the shape operator is a self-adjoint operator on $T_p\mathcal{F}$ and so defines a quadratic form on $\mathcal{F}$, the Second Fundamental Form of the surface.

In fact, since $D\mathcal{F}_p$ is by definition an isomorphism of $\mathbf{R}^2$ onto $T_p\mathcal{F}$, given $w \in T_p\mathcal{F}$, we can define $D\nu_p(w) := (\frac{d}{dt})_{t=0}\nu(\alpha(t))$, where α is any curve of the form $\alpha(t) := \mathcal{F}(\gamma(t))$ with $\gamma(t)$ a curve in $\mathcal{O}$ with $\gamma(0) = p$ such that $D\mathcal{F}_p(\gamma'(0)) = w$. Then since $\nu(\alpha(t)) \in \mathbf{S}^2$ for all t, it follows that $D\nu_p(w) \in T_{\nu(p)}\mathbf{S}^2 = \nu_p^\perp = T\mathcal{F}_p$, completing the proof that $D\nu_p$ maps $T\mathcal{F}_p$ to itself.

13.3.1 Definition. The linear map $-D\nu_p : T_p\mathcal{F} \to T_p\mathcal{F}$ is called the *shape operator* of the surface $\mathcal{F}$ at p.

13.3.2 Remark. The reason for the minus sign will appear later. (It gives the curvatures of the standard surfaces their correct sign.)

13.3.3 Theorem. *The shape operator is self-adjoint.*

▷ **13.3—Exercise 1.** Prove this. Hint—you must show that for all $w_1, w_2 \in T_p\mathcal{F}$, $\langle D\nu_p w_1, w_2\rangle = \langle w_1, D\nu_p w_2\rangle$. However it suffices to to prove this for w_1 and w_2 taken from some basis for $T_p\mathcal{F}$. (Why?) In particular you only need to show this when the w_i are taken from $\{\mathcal{F}_u, \mathcal{F}_v\}$. For this, take the partial derivatives of the identities $\langle \mathcal{F}_u, \nu\rangle = 0$ and $\langle \mathcal{F}_v, \nu\rangle = 0$ with respect to u and v (remembering that $\nu_u = D\nu(\mathcal{F}_u)$), so that for example $\langle D\nu(\mathcal{F}_u)), \mathcal{F}_v\rangle = \langle \nu_u, \mathcal{F}_v\rangle = (\langle \nu, \mathcal{F}_v\rangle)_u - \langle \nu, \mathcal{F}_{vu}\rangle = -\langle \nu, \mathcal{F}_{vu}\rangle$, etc.

Definition of the Second Fundamental Form of a Surface $\mathcal{F}$

We define the *Second Fundamental Form* of a surface $\mathcal{F}$ to be the quadratic form defined by the shape operator. It is denoted by $II^{\mathcal{F}}$, so for $w \in T_p\mathcal{F}$,

$$II^{\mathcal{F}}_p(w) = -\langle D\nu_p(w), w\rangle\,.$$

We will denote the components of the Second Fundamental Form by $L^{\mathcal{F}}, M^{\mathcal{F}}, N^{\mathcal{F}}$, so that

$$II^{\mathcal{F}} = L^{\mathcal{F}}(u,v)\,du^2 + 2M^{\mathcal{F}}(u,v)\,du\,dv + N^{\mathcal{F}}(u,v)\,dv^2,$$

where the functions $L^{\mathcal{F}}$, $M^{\mathcal{F}}$, and $N^{\mathcal{F}}$ are defined by:

$$\begin{aligned} L^{\mathcal{F}} &:= -D\nu(\mathcal{F}_u)\cdot\mathcal{F}_u = \nu\cdot\mathcal{F}_{uu},\\ M^{\mathcal{F}} &:= -D\nu(\mathcal{F}_u)\cdot\mathcal{F}_v = \nu\cdot\mathcal{F}_{uv},\\ N^{\mathcal{F}} &:= -D\nu(\mathcal{F}_v)\cdot\mathcal{F}_v = \nu\cdot\mathcal{F}_{vv}. \end{aligned}$$

As with the First Fundamental Form, we will usually omit the superscript $\mathcal{F}$ from $II^{\mathcal{F}}$ and its components when it is otherwise clear from the context.

Matrix Notation for First and Second Fundamental Form Components

It is convenient when making computations involving the two fundamental forms to have a more uniform matrix style notation for their components relative to the standard basis $\mathcal{F}_u, \mathcal{F}_v$ for $T_p\mathcal{F}$. In such situations we will put $t_1 = u$ and $t_2 = v$ and write $I = \sum_{i,j} g_{ij}\,dt_i\,dt_j$ and $II = \sum_{i,j}\ell_{ij}\,dt_i\,dt_j$. Thus $g_{11} = E$, $g_{12} = g_{21} = F$, $g_{22} = G$, and $\ell_{11} = L$, $\ell_{12} = \ell_{21} = M$, $\ell_{22} = N$. The formulas giving the g_{ij} and ℓ_{ij} in terms of partial derivatives of $\mathcal{F}$ are more uniform with this notation (and hence easier to compute with): $g_{ij} = \mathcal{F}_{t_i}\cdot\mathcal{F}_{t_j}$, and $\ell_{ij} = -\nu_{t_i}\cdot\mathcal{F}_{t_j} = \nu\cdot\mathcal{F}_{t_it_j}$.

We will refer to the 2×2 matrix g_{ij} as g and its inverse matrix by g^{-1}, and we will denote the matrix elements of the inverse matrix by g^{ij}. By Cramer's Rule:

$$g^{-1} = \frac{1}{\det(g)}\begin{pmatrix} g_{22} & -g_{12}\\ -g_{12} & g_{11}\end{pmatrix},$$

i.e., $g^{11} = g_{22}/\det(g)$, $g^{22} = g_{11}/\det(g)$, and $g^{12} = g^{21} = -g_{12}/\det(g)$.

13.3.4 Remark. By Proposition 13.1.6, the matrix of the Shape operator in the basis $\mathcal{F}_{t_1}, \mathcal{F}_{t_2}$ is $g^{-1}\ell$.

▷ **13.3—Exercise 2.** Show that $\|\mathcal{F}_u\times\mathcal{F}_v\|^2 = \det(g) = EG - F^2$, so that the unit normal to $\mathcal{F}$ is $\nu = \frac{\mathcal{F}_u\times\mathcal{F}_v}{\sqrt{EG-F^2}}$. Hint—recall the formula $(u\times v)\cdot(x\times y) = \det\begin{pmatrix} u\cdot x & v\cdot x\\ u\cdot y & v\cdot y\end{pmatrix}$ from our quick review of the vector product.

Geometric Interpretation of the Second Fundamental Form

Definition. Let $\alpha(s) = \mathcal{F}(u(s), v(s))$ be a regular curve on $\mathcal{F}$ and let $n(s)$ denote its unit normal, so $\alpha''(s) = k(s)\, n(s)$, where $k(s)$ is the curvature of α. We define the *normal curvature* to α, denoted by $k_n(s)$, to be the component of $\alpha''(s)$ in the direction normal to $\mathcal{F}$, i.e., the dot product of $\alpha''(s) = k(s)\, n(s)$ with $\nu(\alpha(s))$, so that $k_n(s) = k(s)\cos(\theta(s))$, where $\theta(s)$ is the angle between the normal $n(s)$ to α and the normal $\nu(\alpha(s))$ to $\mathcal{F}$.

13.3.5 Meusnier's Theorem. *If $\alpha(s)$ is a regular curve on a surface $\mathcal{F}$, then its normal curvature is given by the formula $k_n(s) = I\!I^{\mathcal{F}}(\alpha'(s))$. In particular, if two regular curves on $\mathcal{F}$ pass through the same point p and have the same tangent at p, then they have the same normal curvature at p.*

PROOF. Since α is a curve on $\mathcal{F}$, $\alpha'(s)$ is tangent to $\mathcal{F}$ at $\alpha(s)$, so $\alpha'(s) \cdot \nu(\alpha(s))$ is identically zero. Differentiating gives $\alpha''(s) \cdot \nu(\alpha(s)) + \alpha'(s) \cdot D\nu(\alpha'(s)) = 0$, so $k_n(s) := \alpha''(s) \cdot \nu(\alpha(s)) = -\alpha'(s) \cdot D\nu(\alpha'(s)) = I\!I(\alpha'(s))$. ∎

13.3.6 Remark. Recall that the curvature of a curve measures its second order properties, so the remarkable thing about Meusnier's Theorem is that it says, for a curve α that lies on a surface $\mathcal{F}$, k_n, the normal component of the curvature of α depends only on its first order properties (α') and the second order properties of $\mathcal{F}$ ($D\nu$). The obvious conclusion is that k_n measures the curvature of α that is a consequence of its being constrained to lie in the surface.

13.3.7 Remark. If w is a unit tangent vector to $\mathcal{F}$ at p, then w and $\nu(p)$ determine a plane Π through p that cuts $\mathcal{F}$ in a curve $\alpha(s)$ lying on $\mathcal{F}$ with $\alpha(0) = p$ and $\alpha'(0) = w$, This curve α is called the *normal section* of $\mathcal{F}$ in the direction by w. Since α lies in the plane Π, $\alpha''(0)$ is tangent to Π, and since it is of course orthogonal to $w = \alpha'(0)$, it follows that $\alpha''(0)$ must be parallel to $\nu(p)$—i.e., the angle θ that $\alpha''(0)$ makes with $\nu(p)$ is zero, and hence by the definition of the normal curvature, $k_n = k\cos(\theta) = k$, i.e., for a normal section, the normal curvature is just the curvature, so we could equivalently define the Second Fundamental Form of $\mathcal{F}$ by saying that for a unit vector $w \in T_p\mathcal{F}$, $I\!I(w)$ is the curvature of the normal section at p in the direction w. (This is how I always think of $I\!I$.)

▷ **13.3—Exercise 3.** Show that the First and Second Fundamental Forms of a Surface are invariant under congruence. That is, if g is an element of the Euclidean group $\boldsymbol{Euc}(\mathbf{R}^3)$, then $g \circ \mathcal{F}$ has the same First and Second Fundamental Forms as $\mathcal{F}$.

The Principal Directions and Principal Curvatures

Since the Shape operator, $-D\nu_p$, is a self-adjoint operator on $T_p\mathcal{F}$, by the Spectral Theorem there an orthonormal basis e_1, e_2 for $T_p\mathcal{F}$ consisting of eigenvectors of the Shape operator. The corresponding eigenvalues λ_1, λ_2 are called the *principal curvatures* at p, and e_1 and e_2 are called *principal directions* at p. Recall that in general, if $T : V \to V$ is a self-adjoint operator, then a point on the unit sphere of V where the corresponding quadratic form $\langle Tv, v\rangle$ assumes a minimum or maximum value is an eigenvector of T. Since $T_p\mathcal{F}$ is two-dimensional, we can define λ_1 and λ_2 as respectively the minimum and maximum values

of $I\!I_p(w)$ on the unit sphere (a circle!) in $T_p\mathcal{F}$, and e_1 and e_2 as unit vectors where these minimum and maximum values are assumed. We define the *Gaussian Curvature* K and the *Mean Curvature* H at p to be respectively the determinant and trace of the Shape operator $-D\nu_p$, so $K = \lambda_1\lambda_2$ and $H = \lambda_1 + \lambda_2$.

13.3.8 Remark. It is important to have a good formulas for K, H, and the principal curvatures in terms of the coefficients of the first and second fundamental forms (which themselves can easily be computed from the parametric equations for the surface). Recalling from 13.3.4 that the matrix of the Shape operator in the usual basis $\mathcal{F}_u, \mathcal{F}_v$ is $g^{-1}\ell$, it follows that:

$$K = \frac{\det(\ell)}{\det(g)} = \frac{\ell_{11}\ell_{22} - \ell_{12}^2}{g_{11}g_{22} - g_{12}^2}.$$

▷ **13.3—Exercise 4.** Show that the Mean Curvature is given in terms of the coefficients of the first and second fundamental forms by the formula:

$$H = \frac{g_{22}\ell_{11} - 2g_{12}\ell_{12} + g_{11}\ell_{22}}{g_{11}g_{22} - g_{12}^2}.$$

(Hint: The trace of an operator is the sum of the diagonal elements of its matrix with respect to **any** basis.)

13.3.9 Remark. Now that we have formulas for H and K in terms of the g_{ij} and ℓ_{ij}, it is easy to get formulas for the principal curvatures λ_1, λ_2 in terms of H and K (and so in terms of g_{ij} and ℓ_{ij}). Recall that the so-called characteristic polynomial of the Shape operator is $\chi(\lambda) := \det(-D\nu - \lambda I) = (\lambda - \lambda_1)(\lambda - \lambda_2) = \lambda^2 - H\lambda + K$, so that its roots, which are the principal curvatures λ_1, λ_2 are given by $\lambda_1 = \frac{H - \sqrt{H^2 - 4K}}{2}$ and $\lambda_2 = \frac{H + \sqrt{H^2 - 4K}}{2}$.

13.3.10 Remark. There is a special case one should keep in mind, and that is when $\lambda_1 = \lambda_2$, i.e., when $I\!I_p$ is constant on the unit sphere of $T_p\mathcal{F}$. Such a point p is called an *umbillic point* of $\mathcal{F}$. While at a non-umbillic point the principal directions e_1 and e_2 are uniquely determined up to sign, at an umbilic point every direction is a principal direction and we can take for e_1, e_2 any orthonormal basis for the tangent space at p.

Parallel Surfaces

We define a one-parameter family of surfaces $\mathcal{F}(t) : \mathcal{O} \to \mathbf{R}^3$ associated to the surface $\mathcal{F}$ by $\mathcal{F}(t)(u, v) = \mathcal{F}(u, v) - \nu(u, v)$. Clearly $\mathcal{F}(0) = \mathcal{F}$ and $\|\mathcal{F}(t)(u, v) - \mathcal{F}(u, v)\| = t$. Also, $D\mathcal{F}(t)_p = D\mathcal{F}_p + tD\nu_p$, and since $D\nu_p$ maps $T_p\mathcal{F}$ to itself, it follows that $T_p\mathcal{F}(t) = T_p\mathcal{F}$ (at least for t sufficiently small). So, for obvious reasons, we call $\mathcal{F}(t)$ the parallel surface to $\mathcal{F}$ at distance t.

▷ **13.3—Exercise 5.** Since $T_p\mathcal{F}(t) = T_p\mathcal{F}$, it follows that the First Fundamental Forms $I^{\mathcal{F}(t)}$ of the parallel surfaces can be regarded as a one-parameter family of quadratic forms on $\mathcal{F}$. Show that $I\!I^{\mathcal{F}} = \left(\frac{d}{dt}\right)_{t=0} I^{\mathcal{F}(t)}$.

13.3—Example 1. A plane is a surface $\mathcal{F} : \mathbf{R}^2 \to \mathbf{R}^3$ given by a map of the form $p \mapsto x_0 + T(p)$ where $T : \mathbf{R}^2 \to \mathbf{R}^3$ is a linear map of rank two. If we call Π the image of P (a two-dimensional linear subspace of $\mathbf{R}^3$), then clearly the image of $\mathcal{F}$ is $x_0 + \Pi$, the tangent space to $\mathcal{F}$ at every point is Π, and the normal vector ν_p is the same at every point (one of the two unit vectors orthogonal to Π). Here are three ways to see that the Second Fundamental Form of such a surface is zero:

a) The normal sections are all straight lines, so their curvatures vanish.

b) Since ν is constant, the parallel surfaces $\mathcal{F}(t)$ are obtained from $\mathcal{F}$ by translating it by $t\nu$, a Euclidean motion, so all of the First Fundamental Forms $I^{\mathcal{F}(t)}$ are the same, and by the preceding exercise $II^{\mathcal{F}} = 0$.

c) Since the Gauss Map $\nu : \mathbf{R}^2 \to \mathbf{S}^2$ is a constant, the Shape operator $-D\nu$ is zero.

13.3—Example 2. The sphere of radius r. We have already seen how to parametrize this using longitude and co-latitude as the parameters. Also, any hemisphere can be parametrized in the usual way as a graph. However we will not need any parmetrization to compute the Second Fundamental Form. We use two approaches.

a) The normal sections are all great circles, so in particular they are circles of radius r, and so have curvature $\frac{1}{r}$. Thus the Shape operator is $\frac{1}{r}$ times the identity.

b) If $\mathcal{F}(t)$ is the parallel surface at distance t, then clearly $\mathcal{F}(t) = \frac{r+t}{r}\mathcal{F} = (1 + \frac{t}{r})\mathcal{F}$, so $I^{\mathcal{F}(t)} = (1 + \frac{t}{r})I^{\mathcal{F}}$, and this time the exercise gives $II^{\mathcal{F}} = \frac{1}{r}I^{\mathcal{F}}$.

It follows that the Gauss Curvature of the sphere is $K = \frac{1}{r^2}$, and its mean curvature is $H = \frac{2}{r}$.

Lecture 14
The Fundamental Theorem of Surface Theory

Review of Notation.

- In what follows, $(t_1, t_2) \mapsto \mathcal{F}(t_1, t_2)$ is a parametric surface in $\mathbf{R}^3$, $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$.
- Partial derivatives with respect to t_1 and t_2 are denoted by subscripts: $\mathcal{F}_{t_i} := \frac{\partial \mathcal{F}}{\partial t_i}$, $\mathcal{F}_{t_i t_j} := \frac{\partial^2 \mathcal{F}}{\partial t_i t_j}$, etc.
- The standard basis for $T_p\mathcal{F}$—the tangent space to $\mathcal{F}$ at a point $p \in \mathcal{O}$—is $\mathcal{F}_{t_1}, \mathcal{F}_{t_2}$.
- The unit normal to $\mathcal{F}$ at p is $\nu(p) = \frac{\mathcal{F}_{t_1}(p) \times \mathcal{F}_{t_2}(p)}{\left\| \mathcal{F}_{t_1}(p) \times \mathcal{F}_{t_2}(p) \right\|}$.
- The matrix g of the First Fundamental Form with respect to the standard basis is the 2×2 matrix $g_{ij} = \mathcal{F}_{t_i}(p) \cdot \mathcal{F}_{t_j}(p)$.
- The Shape operator at p is the self-adjoint operator $-D\nu_p : T_p\mathcal{F} \to T_p\mathcal{F}$.
- The Shape operator defines the Second Fundamental Form which has the 2×2 matrix of coefficients ℓ given by $\ell_{ij} = -\nu_{t_i} \cdot \mathcal{F}_{t_j} = \nu \cdot \mathcal{F}_{t_i t_j}$.
- The matrix of the Shape operator in the standard basis is $g^{-1}\ell$.

14.1 The Frame Equations.

At each point $p \in \mathcal{O}$ we define the *standard frame* at p, $\mathfrak{f}(p) = (\mathfrak{f}_1(p), \mathfrak{f}_2(p), \mathfrak{f}_3(p))$ to be the basis of $\mathbf{R}^3$ given by $\mathfrak{f}_1(p) := \mathcal{F}_{t_1}(p), \mathfrak{f}_2(p) := \mathcal{F}_{t_2}(p), \mathfrak{f}_3(p) := \nu(p)$. Note that $\mathfrak{f}_1(p), \mathfrak{f}_2(p)$ is just the standard basis for $T_p\mathcal{F}$. We will regard $\mathfrak{f}$ as a map from $\mathcal{O}$ into 3×3 matrices, with the rows being the three basis elements. Since $\mathfrak{f}(p)$ is a basis for $\mathbf{R}^3$, any $v \in \mathbf{R}^3$ can be written uniquely as a linear combination of the $\mathfrak{f}_i(p)$: $v = \sum_i c_i \mathfrak{f}_i(p)$. In particular, we can take in turn for v each of $(\mathfrak{f}_j)_{t_k}(p)$ and this defines uniquely a 3×3 matrix $P^k_{ji}(p)$ such that $(\mathfrak{f}_j)_{t_k}(p) = \sum_j P^k_{ji}(p)\mathfrak{f}_i(p)$. We can write these equations as a pair of equations between five matrix-valued functions $\mathfrak{f}$, $\mathfrak{f}_{t_1}$, $\mathfrak{f}_{t_2}$, P^1, and P^2, defined on $\mathcal{O}$, namely:

$$\mathfrak{f}_{t_1} = \mathfrak{f}\, P^1$$
$$\mathfrak{f}_{t_2} = \mathfrak{f}\, P^2$$

and we call these equations the *frame equations* for the surface $\mathcal{F}$.

What makes these equations so interesting and important is that, as we will see below, the matrix-valued functions P^1 and P^2 can be calculated explicitly from formulas that display them as fixed expressions in the coeffients g_{ij} and ℓ_{ij} of the First and Second Fundamental Forms and their partial derivatives. Thus, **if the First and Second Fundamental Forms are known, we can consider the Frame Equations as a coupled pair of first order PDE for the frame field** $\mathfrak{f}$, and it follows from the Frobenius Theorem that we can solve these equations and find the frame field, and then with another integration we can recover the surface $\mathcal{F}$. Thus the frame equations are analogous to the Frenet equations of curve theory, and lead in the same way to a Fundamental Theorem. Now the fun begins!

14.1.1 Lemma. *Let V be an inner-product space, $\mathfrak{f} = (\mathfrak{f}_1, \dots, \mathfrak{f}_n)$ a basis for V, and G the matrix of inner-products $G_{ij} = \langle \mathfrak{f}_i, \mathfrak{f}_j \rangle$. Given $x = \sum_{i=1}^n x_i \mathfrak{f}_i$ in V, let $\xi_i := \langle x, \mathfrak{f}_i \rangle$. Then $\xi_i = \sum_{j=1}^n G_{ji} x_j = \sum_{j=1}^n G^t_{ij} x_j$, or in matrix notation, $(\xi_1, \dots, \xi_n) = (x_1, \dots, x_n) G^t$, so $(x_1, \dots, x_n)^t = G^{-1} (\xi_1, \dots, \xi_n)^t$.*

PROOF. $\xi_i = \left\langle \sum_{j=1}^n x_j \mathfrak{f}_j, \mathfrak{f}_i \right\rangle = \sum_{j=1}^n x_j \langle \mathfrak{f}_j, \mathfrak{f}_i \rangle$. ∎

14.1.2 Remark. Here is another way of phrasing this result. The basis $\mathfrak{f}_1, \dots, \mathfrak{f}_n$ for V determines two bases for the dual space V^*, the dual basis ℓ_i defined by $\ell_i(\mathfrak{f}_j) = \delta_{ij}$ and the basis $\mathfrak{f}_i^*$, defined by $\mathfrak{f}_i^*(v) := \langle v, \mathfrak{f}_i \rangle$, and these two bases are related by $\mathfrak{f}_i^* = \sum_{j=1}^n G_{ij} \ell_j$.

14.1.3 Theorem.

$$P^1 = G^{-1} A^1 = \begin{pmatrix} g^{11} & g^{12} & 0 \\ g^{12} & g^{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2}(g_{11})_{t_1} & \frac{1}{2}(g_{11})_{t_2} & -\ell_{11} \\ (g_{12})_{t_1} - \frac{1}{2}(g_{11})_{t_2} & \frac{1}{2}(g_{22})_{t_1} & -\ell_{12} \\ \ell_{11} & \ell_{12} & 0 \end{pmatrix}$$

$$P^2 = G^{-1} A^2 = \begin{pmatrix} g^{11} & g^{12} & 0 \\ g^{12} & g^{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{2}(g_{11})_{t_2} & (g_{12})_{t_2} - \frac{1}{2}(g_{22})_{t_1} & -\ell_{12} \\ \frac{1}{2}(g_{22})_{t_1} & \frac{1}{2}(g_{22})_{t_2} & -\ell_{22} \\ \ell_{12} & \ell_{22} & 0 \end{pmatrix}$$

PROOF. In the lemma (with $n = 3$) take $\mathfrak{f} = \mathfrak{f}(p)$, the standard frame for $\mathcal{F}$ at p, so that

$$G = \begin{pmatrix} g_{11} & g_{12} & 0 \\ g_{12} & g_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and hence

$$G^{-1} = \begin{pmatrix} g^{11} & g^{12} & 0 \\ g^{12} & g^{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(recall that the $g^{-1} = g^{ij}$ is the matrix inverse to $g = g_{ij}$), and take $x = (\mathfrak{f}_j)_{t_k}$, so by the conclusion of the Lemma, $A^k_{ij} = \xi_i = (\mathfrak{f}_j)_{t_k} \cdot \mathfrak{f}_i$. Thus, for example, for $i = 1, 2$:

$$A^k_{ii} = (\mathfrak{f}_i)_{t_k} \cdot \mathfrak{f}_i = \mathcal{F}_{t_i t_k} \cdot \mathcal{F}_{t_i} = \frac{1}{2} (\mathcal{F}_{t_i} \cdot \mathcal{F}_{t_i})_{t_k} = \frac{1}{2} (g_{ii})_{t_k}.$$

Next note that $(g_{12})_{t_1} = (\mathcal{F}_{t_1} \cdot \mathcal{F}_{t_2})_{t_1} = \mathcal{F}_{t_1 t_1} \cdot \mathcal{F}_{t_2} + \mathcal{F}_{t_1} \cdot \mathcal{F}_{t_1 t_2} = \mathcal{F}_{t_1 t_1} \cdot \mathcal{F}_{t_2} + \frac{1}{2}(g_{11})_{t_2}$, so:

$$A^1_{21} = (\mathfrak{f}_1)_{t_1} \cdot \mathfrak{f}_2 = \mathcal{F}_{t_1 t_1} \cdot \mathcal{F}_{t_2} = (g_{12})_{t_1} - \frac{1}{2}(g_{11})_{t_2}.$$

and interchanging the roles of t_1 and t_2 gives

$$A^2_{12} = (\mathfrak{f}_2)_{t_2} \cdot \mathfrak{f}_1 = \mathcal{F}_{t_2 t_2} \cdot \mathcal{F}_{t_1} = (g_{12})_{t_2} - \frac{1}{2}(g_{22})_{t_1}. \text{ Also}$$

$$A^1_{12} = (\mathfrak{f}_2)_{t_1} \cdot \mathfrak{f}_1 = \mathcal{F}_{t_1 t_2} \cdot \mathcal{F}_{t_1} = \frac{1}{2} (\mathcal{F}_{t_1} \cdot \mathcal{F}_{t_1})_{t_2} = \frac{1}{2} (g_{11})_{t_2} \text{ and}$$

$$A^2_{21} = (\mathfrak{f}_1)_{t_2} \cdot \mathfrak{f}_2 = \mathcal{F}_{t_1 t_2} \cdot \mathcal{F}_{t_2} = \frac{1}{2} (\mathcal{F}_{t_1} \cdot \mathcal{F}_{t_1})_{t_2} = \frac{1}{2} (g_{11})_{t_2}.$$

For $i = 1, 2$,

$$A^k_{i3} = (\mathfrak{f}_3)_{t_k} \cdot \mathfrak{f}_i = \nu_{t_k} \cdot \mathcal{F}_{t_i} = D\nu(\mathcal{F}_{t_k}) \cdot \mathcal{F}_{t_i} = -\ell_{ki},$$

and since $\mathfrak{f}_3$ is orthogonal to $\mathfrak{f}_i$, $0 = (\mathfrak{f}_3 \cdot \mathfrak{f}_i)_{t_k} = (\mathfrak{f}_3)_{t_k} \cdot \mathfrak{f}_i + \mathfrak{f}_3 \cdot (\mathfrak{f}_i)_{t_k}$, hence

$$A^k_{3i} = (\mathfrak{f}_i)_{t_k} \cdot \mathfrak{f}_3 = -(\mathfrak{f}_3)_{t_k} \cdot \mathfrak{f}_i = -A^k_{i3}.$$

Finally, since $\mathfrak{f}_3 \cdot \mathfrak{f}_3 = \|\nu\|^2 = 1$, $(\mathfrak{f}_3 \cdot \mathfrak{f}_3)_{t_k} = 0$, so

$$A^k_{33} = (\mathfrak{f}_3)_{t_k} \cdot \mathfrak{f}_3 = \frac{1}{2}(\mathfrak{f}_3 \cdot \mathfrak{f}_3)_{t_k} = 0. \quad \blacksquare$$

Henceforth we regard the 3×3 matrix-valued functions G, G^{-1}, A^k and P^k in $\mathcal{O}$ as being **defined** by the formulas in the statement of the above theorem.

14.1.4 Corollary (Gauss-Codazzi Equations). *If g_{ij} and ℓ_{ij} are the coefficients of the First and Second Fundamental Forms of a surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$, then the matrix-valued functions P^1 and P^2 defined in $\mathcal{O}$ by the above Theorem satisfy the matrix identity*

$$P^1_{t_2} - P^2_{t_1} = P^1P^2 - P^2P^1$$

called the Gauss-Codazzi Equations.

PROOF. Differentiate the first of the Frame Equations with respect to t_2 and the second with respect to t_1 and set $\mathfrak{f}_{t_1t_2} = \mathfrak{f}_{t_2t_1}$. This gives $\mathfrak{f}_{t_2}P^1 + \mathfrak{f}P^1_{t_2} = \mathfrak{f}_{t_1}P^2 + \mathfrak{f}P^2_{t_1}$. Substituting for $\mathfrak{f}_{t_1}$ and $\mathfrak{f}_{t_2}$ their values from the Frame Equations, gives

$$\mathfrak{f}\,(P^1_{t_2} - P^2_{t_1} - (P^1P^2 - P^2P^1)) = 0,$$

and since $\mathfrak{f}$ is a non-singular matrix, the corollary follows. $\blacksquare$

14.2 Gauss's "Theorema Egregium" (Remarkable Theorem).

Karl Friedrich Gauss was one of the great mathematicians of all time, and it was he who develped the deeper aspects of surface theory in the first half of the nineteenth century. There is one theorem that he proved that is highly surprising, namely that K, the Gauss Curvature of a surface (the determinant of the shape operator), is an **intrinsic** quantity that is it can be computed from a knowledge of only the First Fundamental Form, and so it can be found by doing measurements within the surface without reference to how the surface is embedded in space. Gauss thought so highly of this result that in his notes he called it the "Theorema Egregium", or the remarkable (or outstanding) theorem. Notice that by Theorem 14.1.3, the matrix entries of P^k_{ij} with $1 \le i, j \le 2$ depend only on the g_{ij} and their partial derivatives, so to prove the intrinsic nature of K it will suffice to get a formula for it involving only these quantities and the g^{ij}.

14.2.1 Theorema Egregium.

$$K = -\frac{(P^1_{12})_{t_2} - (P^2_{12})_{t_1} - \sum_{j=1}^2 (P^1_{1j}P^2_{j2} - P^2_{1j}P^1_{j2})}{g^{11}(g_{11}g_{22} - g^2_{12})}.$$

PROOF. In the matrix Gauss-Codazzi Equation, consider the equation for the first row and second column,

$$(P^1_{12})_{t_2} - (P^2_{12})_{t_1} = \sum_{j=1}^3 (P^1_{1j}P^2_{j2} - P^2_{1j}P^1_{j2}).$$

If we move all terms involving $P^k_{i,j}$ with $i, j < 3$ to the left hand side of the equation, the result is:

$$(P^1_{12})_{t_2} - (P^2_{12})_{t_1} - \sum_{j=1}^2 (P^1_{1j}P^2_{j2} - P^2_{1j}P^1_{j2}) = P^1_{13}P^2_{32} - P^2_{13}P^1_{32}.$$

Now use Theorem 14.1.3 to find the matrix elements on the right hand side of this equation: $P^1_{13} = -(g^{11}\ell_{11} + g^{12}\ell_{12})$, $P^2_{13} = -(g^{11}\ell_{12} + g^{12}\ell_{22})$, $P^1_{32} = \ell_{12}$, $P^2_{32} = \ell_{22}$. Thus:

$$P^1_{13}P^2_{32} - P^2_{13}P^1_{32} = -(g^{11}\ell_{11} + g^{12}\ell_{12})\ell_{22} + (g^{11}\ell_{12} + g^{12}\ell_{22})\ell_{12} = -g^{11}(\ell_{11}\ell_{22} - \ell^2_{12}).$$

Since by an earlier remark (13.3.8),

$$K = \frac{\det(\ell)}{\det(g)} = \frac{\ell_{11}\ell_{22} - \ell^2_{12}}{g_{11}g_{22} - g^2_{12}},$$

$P^1_{13}P^2_{32} - P^2_{13}P^1_{32} = -g^{11}(\ell_{11}\ell_{22} - \ell^2_{12}) = -g^{11}(g_{11}g_{22} - g^2_{12})K$
and the claimed formula for K follows. ∎

14.2.2 The Fundamental Theorem of Surfaces. *Congruent parametric surfaces in* $\mathbf{R}^3$ *have the same First and Second Fundamental Forms and conversely two parametric surfaces in* $\mathbf{R}^3$ *with the same First and Second Fundamental Forms are congruent. Moreover, if* $\mathcal{O}$ *is a domain in* $\mathbf{R}^2$ *and* $I = \sum_{ij=1}^2 g_{ij}\,dt_i\,dt_j$, $II = \sum_{ij=1}^2 \ell_{ij}\,dt_i\,dt_j$ *are* C^2 *quadratic forms in* $\mathcal{O}$ *with* I *positive definite, then there exists a parametric surface* $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$ *with* $I^{\mathcal{F}} = I$ *and* $II^{\mathcal{F}} = II$ *provided the Gauss-Codazzi equations are satisfied.*

PROOF. We have already discussed the first statement. If $\mathcal{F}^i : \mathcal{O} \to \mathbf{R}^3$, $i = 1, 2$ have the same First and Second Fundamental forms, then after translations we can assume that they both map some point $p \in \mathcal{O}$ to the origin. Then, since $\mathcal{F}^1_{t_i}(p) \cdot \mathcal{F}^1_{t_j}(p) = g_{ij}(p) = \mathcal{F}^2_{t_i}(p) \cdot \mathcal{F}^2_{t_j}(p)$, it follows that after transforming one of the surfaces by an orthogonal transformation we can assume that the standard frame $\mathfrak{f}^1$ for $\mathcal{F}^1$ and the standard frame $\mathfrak{f}^2$ for $\mathcal{F}^2$ agree at p. But then, since $\mathcal{F}^1$ and $\mathcal{F}^2$ have identical frame equations, by the uniqueness part of the Frobenius Theorem it follows that $\mathfrak{f}^1$ and $\mathfrak{f}^2$ agree in all of $\mathcal{O}$, so in

particular $\mathcal{F}^1_{t_i} = \mathcal{F}^2_{t_i}$. But then $\mathcal{F}^1$ and $\mathcal{F}^2$ differ by a constant, and since they agree at p, they are identical, proving the second statement.

To prove the third and final statement of the theorem, we first note that since g_{ij} is positive definite, it is in particular invertible, so the matrix G^{-1} and the matrices P^k of Theorem 14.1.3, are well-defined. Moreover, since the Gauss-Codazzi equations are just the compatibility conditions of Frobenius Theorem, it follows that we can solve the "frame equations" $\mathfrak{f}_{t_i} = \mathfrak{f}\, P^k$, $k = 1, 2$ uniquely given an arbitrary initial value for $\mathfrak{f}$ at some point $p \in \mathcal{O}$, and for this initial frame we choose a basis $\mathfrak{f}(p)$ for $\mathbf{R}^3$ such that $\mathfrak{f}_i(p) \cdot \mathfrak{f}_j(p) = G_{ij}(p)$ (which is possible since g_{ij} and hence G_{ij} is positive definite).

Having solved for the frame field $\mathfrak{f}$, we now need to solve the system $\mathcal{F}_{t_i} = \mathfrak{f}_i$, $i = 1, 2$ to get the surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$. This is another Frobenius problem, and now the compatibility condition is $(\mathfrak{f}_1)_{t_2} = (\mathfrak{f}_2)_{t_1}$ or by the frame equation, $\sum_{j=1}^3 P^2_{j1}\mathfrak{f}_j = \sum_{j=1}^3 P^1_{j2}\mathfrak{f}_j$. But by inspection, the second column of P^1 is indeed equal to the first column of P^2, so the compatibility equations are satisfied and we can find a unique $\mathcal{F}$ with $\mathcal{F}(p) = 0$ and $\mathcal{F}_{t_i} = \mathfrak{f}_i$, for $i = 1, 2$.

It remains to show that $\mathcal{F}$ is a surface in $\mathbf{R}^3$ with $\sum_{ij} g_{ij} dt_i\, dx_j$ and $\sum_{ij} \ell_{ij} dt_i\, dt_j$ as its first and second fundamental forms, i.e.,

- $\mathcal{F}_{t_1}$ and $\mathcal{F}_{t_2}$ are independent,
- $\mathfrak{f}_3$ is orthogonal to $\mathcal{F}_{t_1}$ and $\mathcal{F}_{t_2}$,
- $\|\mathfrak{f}_3\| = 1$,
- $\mathcal{F}_{t_i} \cdot \mathcal{F}_{t_j} = g_{ij}$, and
- $(\mathfrak{f}_3)_{x_i} \cdot f_{x_j} = -\ell_{ij}$.

The first step is to prove that the 3×3 matrix function $\Phi = (\mathfrak{f}_i \cdot \mathfrak{f}_j)$ is equal to G. We compute the partial derivatives of Φ. Since $\mathfrak{f}$ satisfy the frame equations,

$$\begin{aligned}(\mathfrak{f}_i \cdot \mathfrak{f}_j)_{t_1} &= (\mathfrak{f}_i)_{t_1} \cdot \mathfrak{f}_j + \mathfrak{f}_i \cdot (\mathfrak{f}_j)_{t_1} \\ &= \sum_k P^1_{ki}\mathfrak{f}_k \cdot \mathfrak{f}_j + P^1_{kj}\mathfrak{f}_k \cdot \mathfrak{f}_i = \sum_k P^1_{ki} g_{jk} + g_{ik} P^1_{kj} \\ &= (GP^1)_{ji} + (GP^1)_{ij} = (GP^1 + (GP^1)^t)_{ij}.\end{aligned}$$

But $GP^1 = G(G^{-1}A^1) = A^1$, so $\Phi_{t_1} = G_{t_1}$ and a similar computation gives $\Phi_{t_2} = G_{t_2}$. Thus Φ and G differ by a constant, and since they agree at p they are identical. Thus $\mathfrak{f}_i \cdot \mathfrak{f}_j = G_{ij}$, which proves all of the above list of bulleted items but the last.

To compute the Second Fundamental Form of $\mathcal{F}$, we again use the frame equations:

$$\begin{aligned}-(\mathfrak{f}_3)_{t_1} \cdot \mathfrak{f}_j &= (g^{11}\ell_{11} + g^{12}\ell_{12})\mathfrak{f}_1 \cdot \mathfrak{f}_j + (g^{12}\ell_{11} + g^{22}\ell_{12})\mathfrak{f}_2 \cdot \mathfrak{f}_j \\ &= (g^{11}\ell_{11} + g^{12}\ell_{12})g_{1j} + (g^{12}\ell_{11} + g^{22}\ell_{12})g_{2j} \\ &= \ell_{11}(g^{11}g_{1j} + g^{12}g_{2j}) + \ell_{12}(g^{21}g_{1j} + g^{22}g_{2j}) \\ &= \ell_{11}\delta_{1j} + \ell_{12}\delta_{2j}.\end{aligned}$$

So $-(\mathfrak{f}_3)_{t_1} \cdot \mathfrak{f}_1 = \ell_{11}$, $\quad -(\mathfrak{f}_3)_{t_1} \cdot \mathfrak{f}_2 = \ell_{12}$, and similar computations show that $-(\mathfrak{f}_2)_{t_2} \cdot \mathfrak{f}_j = \ell_{2j}$, proving that $\sum_{ij} \ell_{ij} dt_i\, dt_j$ is the Second Fundamental Form of $\mathcal{F}$. ∎

The Eighth and Final Matlab Project.

The primary Matlab M-File should be called SurfaceFT.m and should start out:

```
function   F = SurfaceFT(g11,g12,g22,l11,l12,l22,a1,b1,a2,b2,T1Res,T2Res)
```

where $I := \sum_{ij}$ `gij(t1,t2) dti dtj` and $II := \sum_{ij}$ `lij(t1,t2) dti dtj` are quadratic forms in $\mathcal{O}$, and the function $F : \mathcal{O} \to \mathbf{R}^3$ returned is supposed to be the surface having I and II as its First and Second Fundamental Forms. For this surface to exist, we know from the Fundamental Theorem that it is necessary and sufficient that I be positive definite and that the Gauss-Codazzi equations be satisfied.

Of course the heavy lifting of the SurfaceFT will be done by AlgorithmF (i.e., the solution of the Frobenius Problem) which you will apply to integrate the frame equations in order to get the frame field $\mathfrak{f}$—after which you must apply AlgorithmF a second time to get the surface $\mathcal{F}$ from $\mathfrak{f}_1$ and $\mathfrak{f}_2$.

But to carry out the first application of AlgorithmF, you must first compute the matrices $P^1 = G^{-1}A^1$ and $P^2 = G^{-1}A^2$ that define the right hand sides of the two frame equations. Recall that G^{-1} is the inverse of the 3×3 matrix

$$G = \begin{pmatrix} \texttt{g11(t1,t2)} & \texttt{g12(t1,t2)} & 0 \\ \texttt{g12(t1,t2)} & \texttt{g22(t1,t2)} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and so it can be easily computed from the `gij(t1,t2)` by using Cramer's Rule, while the two 3×3 matrices A^1 and A^2 are given explicitly (see 14.1.3 above) in terms of the `g11(t1,t2)` **and their partial derivatives with respect to the** t_i . (Of course, once you have both G^{-1} and A^i, you get P^i as their matrix product.)

In order to be able to compute the A^i, you will first need to define some auxilliary functions. Most of them can probably be subfunctions defined in the same file, though some could be separate M-Files. For example you will want to have a function called `g(t1,t2)` that returns a 2×2 matrix $\begin{pmatrix} \texttt{g11(t1,t2)} & \texttt{g12(t1,t2)} \\ \texttt{g12(t1,t2)} & \texttt{g22(t1,t2)} \end{pmatrix}$ and another called `l(t1,t2)` that returns a 2×2 matrix $\begin{pmatrix} \texttt{l11(t1,t2)} & \texttt{l12(t1,t2)} \\ \texttt{l12(t1,t2)} & \texttt{l22(t1,t2)} \end{pmatrix}$. You will then want to create the functions `G(t1,t2)` and `invG(t2,t2)` that return the 3×3 matrices G and G^{-1}.

There is another complication before you can define the Matlab functions `A1` and `A2` that represent A^1 and A^2. You not only need the functions `gij(t1,t2)` but also their first partial derivatives with respect to the variables `t1` and `t2`. I recommend that along with the function `g(t1,t2)` you also define two more functions `g_t1(t1,t2)` and `g_t2(t1,t2)` that return 2×2 matrices whose entries are the partial derivatives of the `gij(t1,t2)` with respect to `t1` and `t2` respectively. As usual you can compute these partial derivatives using symmetric differencing—you don't need to do it symbolically.

You should define a Matlab function GaussCodazziCheck that will check whether or not the Gauss-Codazzi Equations are satisfied. Once you have defined the two 3×3 matrix-

valued functions P^1 and P^2, it will be easy to write GaussCodazziCheck, since the Gauss-Codazzi equations are just $P^1_{t_2} - P^2_{t_1} = P^1P^2 - P^2P^1$. The idea is to check the identities numerically, matrix element by matrix element, at a sufficiently dense set of points, again using symmetric differencing to compute the derivatives.

Of course, when you are all done you will need some good test cases on which to try out you algorithm. We will discuss this elsewhere.

GOOD LUCK, AND HAVE FUN!

Appendix I
The Matlab Projects

▷Project 1. Implement Gram-Schmidt as a Matlab Function

In more detail, create a Matlab m-file GramSchmidt.m in which you define a Matlab function GramSchmidt(M) taking as input a rectangular matrix M of real numbers of arbitrary size $m \times n$, and assuming that the m rows of M are linearly independent, it should transform M into another $m \times n$ matrix in which the rows are orthonormal, and moreover such that the subspace spanned by the first k rows of the output matrix is the same as the space spanned by the first k rows of the input matrix. Clearly, in writting your algorithm, you will need to know the number of rows, m and the number of columns n of M. You can find these out using the Matlab size function. In fact, size(M) returns (m,n) while size(M,1) returns m and size(M,2) returns n. Your algorithm will have to do some sort of loop, iterating over each row in order. Be sure to test your function on a number of different matrices of various sizes. What happens to your function if you give it as input a matrix with linearly dependent rows. (Ideally it should report this fact and not just return garbage!)

Addenda

- The project is a little ambiguous. It asks for a function M-File that **transforms** the input matrix to a matrix with orthogonal rows. One legitimate interpretation (actually the one I had in mind) was that the input matrix should actually be changed by the function into one with orthogonal rows. However, if you prefer to write your function so that it does not actually change the input matrix but it instead returns a different matrix having orthogonal rows, that is acceptable. (But you should realize that there are often good reasons to do the orthogonalization "in place". For example, if the input matrix is very large (e.g., 10000×10000) then there might be a memory problem in creating a second matrix of that size.)
- Please put your name in a comment near the top of the M-File.
- Another comment (coming just after the line declaring the function) should explain in detail just what the function does and how to call it. This will be printed in the command window when a user types "help GramSchmidt".
- Several of you have asked how to handle the problem of an input matrix in which the rows are not linearly independent. You will detect this in the course of the computation by finding that the norm of a certain vector u is zero, so you cannot normalize u to get the next element of the orthonormal set. (By the way, you should be careful to check not just for $\|u\|$ being exactly zero, but say for $\|u\| < 0.00001$. The reason is that because of roundoff and other errors the computation will be too unreliable if the vectors are "almost linearly dependent".) In this case you should not try to return any matrix and should just use the disp() function to display an error string to tell the user about the problem. Something like:

```
disp('Input matrix rows are dependent; orthonormalization impossible.')
```

▷Project 2. Implement the Trapezoidal Rule and Simpson's Rule in Matlab

0.1 Review of Trapezoidal and Simpson's Rules.

One usually cannot find anti-derivatives in closed form, so it is important to be able to "evaluate an integral numerically"—meaning approximate it with arbitrary precision. In fact, this is so important that there are whole books devoted the study of numerical integration methods (aka quadrature rules). We will consider only two such methods, the Trapezoidal Rule and Simpson's Rule. In what follows, we will assume that the integrand f is always at least continuous.

0.1.1 Definition. By a *quadrature rule* we mean a function M that assigns to each continuous function $f : [a, b] \to V$ (mapping a closed interval $[a, b]$ into an inner-product space V) a vector $M(f, a, b) \in V$—which is supposed to be an approximation of the integral, $\int_a^b f(t)\,dt$. A particular quadrature rule M is usually given by specifying a linear combination of the values of f at certain points of he interval $[a, b]$; that is, it has the general form $M(f, a, b) := \sum_{i=1}^{n} w_i f(t_i)$, where the points $t_i \in [a, b]$ are called the *nodes* of M and the scalars w_i are called its *weights*. The *error* of M for a particular f and $[a, b]$ is defined as $\mathrm{Err}(M, f, a, b) := \left\| \int_a^b f(t)\,dt - M(f, a, b) \right\|$.

0.1—Example 1. The Trapezoidal Rule: $M^T(f, a, b) := \frac{b-a}{2}[f(a) + f(b)]$.
In this case, there are two nodes, namely the two endpoints of the interval, and they have equal weights, namely half the length of the interval. Later we shall see the origin of this rule (and explain its name).

0.1—Example 2. Simpson's Rule: $M^S(f, a, b) := \frac{b-a}{6}[f(a) + 4f(\frac{a+b}{2}) + f(b)]$.
So now the nodes are the two endpoints, as before, and in addition the midpoint of the interval. And the weights are $\frac{b-a}{6}$ for the two endpoints and $\frac{2(b-a)}{3}$ for the midpoint.

0.1.2 Remark. Notice that in both examples the weights add up to one. This is no accident; any "reasonable" quadrature rule should have a zero error for a constant function, and this easily implies that the weights must add to one.

0.1.3 Proposition. *If $f : [a, b] \to V$ has two continuous derivatives, and $\|f''(t)\| < C$ for all $t \in [a, b]$ then $\mathrm{Err}(M^T, f, a, b) \le C\frac{(b-a)^3}{12}$. Similarly, if $f : [a, b] \to V$ has four continuous derivatives, and $\|f''''(t)\| < C$ for all $t \in [a, b]$ then $\mathrm{Err}(M^S, f, a, b) \le C\frac{(b-a)^5}{90}$*

0.1.4 Remark. The proof of this proposition is not difficult—it depends only the Mean Value Theorem—but it can be found in any numerical analysis text and will not be repeated here.

0.1.5 Definition. If M is a quadrature rule then we define a sequence M_n of *derived* quadrature rules by $M_n(f, a, b) := \sum_{i=0}^{n-1} M(f, a + ih, a + (i + 1)h)$ where $h = \frac{b-a}{n}$. We say that the rule M is *convergent* for f on $[a, b]$ if the sequence $M_n(f, a, b)$ converges to $\int_a^b f(t)\,dt$.

In other words, to estimate the integral $\int_a^b f(t)\,dt$ using the n-th derived rule M_n, we simply divide the interval $[a,b]$ of integration into n equal sub-intervals, estimate the integral on each sub-interval using M, and then add these estimates to get the estimate of the integral on the whole interval.

0.1.6 Remark. We next note an interesting relation between the errors of M and of M_n. Namely, with the notation just used in the above definition, we see that by the additivity of the integral, $\int_a^b f(t)\,dt = \sum_{i=0}^{n-1} \int_{a+ih}^{a+(i+1)h} f(t)\,dt$, hence from the definition of M_n and the triangle inequality, we have $\mathrm{Err}(M_n, f, a, b) \le \sum_{i=0}^{n-1} \mathrm{Err}(M, f, a+ih, a+(i+1)h)$. We can now use this together with Proposition 7.4.3 to prove the following important result:

0.1.7 Theorem. *If $f : [a,b] \to V$ has two continuous derivatives, and $\|f''(t)\| < C$ for all $t \in [a,b]$ then $\mathrm{Err}(M_n^T, f, a, b) \le C\frac{(b-a)^2}{12n^2}$. Similarly, if $f : [a,b] \to V$ has four continuous derivatives, and $\|f''''(t)\| < C$ for all $t \in [a,b]$ then $\mathrm{Err}(M_n^S, f, a, b) \le C\frac{(b-a)^4}{90n^4}$*

0.1.8 Remark. This shows that both the Trapezoidal Rule and Simpson's Rule are convergent for any reasonably smooth function. But it also shows that Simpson's Rule is far superior to the Trapezoidal Rule. For just fifty per cent more "effort" (measured by the number of evaluations of f) one gets a far more accurate result.

The second Matlab project is to develop Matlab code to implement the Trapezoidal Rule and Simpson's Rule, and then to do some experimentation with your software, checking that the error estimates of theorem 7.4.7 are satisfied for some test cases where the function f has a known anti-derivative and so can be evaluated exactly. In more detail:

1) Write a Matlab function M-file defining a function TrapezoidalRule(f,a,b,n). This should return the value of $M_n^T(f,a,b)$. Here of course the parameters a and b represent real numbers and the parameter n a positive integer. But what about the parameter f, i.e., what should it be legal to substitute for f when the TrapezoidalRule(f,a,b,n) is called? Answer: f should represent a function of a real variable whose values are arrays (of some fixed size) of real numbers. The function that you are permitted to substitute for f should either be a built-in Matlab function (such as sin) or an inline function in the Matlab Workspace, or a function that is defined in some other M-File.
2) Write a second Matlab function M-file defining a function SimpsonsRule(f,a,b,n) that returns $M_n^S(f,a,b)$.
3) Recall that $\int_0^t \frac{dx}{1+x^2} = \arctan(t)$, so that in particular $\int_0^1 \frac{4\,dx}{1+x^2} = 4\arctan(1) = \pi$. Using the error estimates for the Trapezoidal Rule and Simpson's Rule, calculate how large n should be to calculate π correct to d decimal places from this formula using Trapezoidal and Simpson. Set format long in Matlab and get the value of π to fifteen decimal places by simply typing pi. Then use your Trapezoidal and Simpson functions from parts 1) and 2) to see how large you actually have to choose n to calculate π to 5, 10, and 15 decimal places.
4) Be prepared to discuss your solutions in the Computer Lab.

▷Project 3. Implement the Method of Successive Approximations

0.2 Review of The Contraction Principle.

If X is any set and $f : X \to X$ a mapping of X to itself, then for each positive integer n we define a mapping $f^{(n)} X \to X$ by composing f with itself n times. That is, $f^{(1)}(x) = f(x)$, $f^{(2)}(x) = f(f(x))$, $f^{(3)}(x) = f(f(f(x)))$, etc. To be more formal, we define the sequence $f^{(n)}$ inductively by: $f^{(1)} := f$ and $f^{(n+1)} := f \circ f^{(n)}$.

Elsewhere you have verified the following facts:

1) $f^{(n)} \circ f^{(k)} = f^{(n+k)}$.
2) If X is a metric space and that f satisfies a Lipschitz condition with constant K then $f^{(n)}$ satisfies a Lipschitz condition with constant K^n.
3) Assuming again that X is a metric space and that $f : X \to X$ is a contraction mapping, i.e., that f satisfies a Lipschitz condition with constant $K < 1$, we have the so-called Fundamental Inequality For Contraction Mappings, namely, for all $x_1, x_2 \in X$,

$$\rho(x_1, x_2) \leq \frac{1}{1-K}\Big(\rho(x_1, f(x_1)) + \rho(x_2, f(x_2)\Big).$$

4) With the same assumptions, if x is **any** point of X then

$$\rho(f^{(n)}(x), f^{(m)}(x)) \leq \left(\frac{K^n + K^m}{1-K}\right)\rho(x, f(x)),$$

5) If $f : X \to X$ is a contraction mapping and p is the unique fixed point of f, then for any x in X, $\rho(f^{(n)}(x), p) \leq \left(\frac{K^n}{1-K}\right)\rho(x, f(x))$

Remark. The sequence $\{f^{(n)}(x)\}$ is usually referred to as *the sequence of iterates of x under f*, and the process of locating the fixed point p of a contraction mapping f by taking the limit of a sequence of iterates of f goes by the name *"the method of successive approximations"*. To make this into a rigorous algorithm, we must have a "stopping rule". That is, since we cannot keep iterating f forever, we must know when to stop. One rather rough approach is to keep on iterating until successive iterates are "close enough", but a better method is provided by the previous problem. Suppose we decide to be satisfied with the approximation $f^{(n)}(x)$ if we can be sure that $\rho(f^{(n)}(x), p) \leq \epsilon$ where ϵ is some "tolerance" given in advance. We first compute $f(x)$, then $\rho(f(x), x)$, and then solve $\left(\frac{K^n}{1-K}\right)\rho(x, f(x)) = \epsilon$ for n and iterate $n-1$ more times to get our acceptable approximation $f^{(n)}(x)$ to p.

6) Solve $\left(\frac{K^n}{1-K}\right)\rho(x, f(x)) = \epsilon$ for n in terms of ϵ, K, and $\rho(x, f(x))$.

Third Matlab Project.

Write an Matlab M-file that implements the Successive Approximations algorithm. Name it SuccessiveApprox.m, and use it to define a Matlab function SuccessiveApprox($f, K, x,$ eps). Assume that $f : \mathbf{R}^n \to \mathbf{R}^n$ is known to be a contraction mapping with contraction constant K, that $x \in \mathbf{R}^n$, and you want to compute iterates of x until you are within eps of the fixed point p of f. Use a subfunction to compute the number of times n you need to iterate f starting from x to get within eps of p, and then use a loop and feval to iterate applying f to x the appropriate number of times.

▹Project 4. Implement Euler's Method and Runge-Kutta in Matlab

0.3 Review of Stepping Methods for Solving IVPs.

In what follows X is a C^1 time-dependent vector field on V, and given t_0 in $\mathbf{R}$ and x^0 in V we will denote by $\sigma(X, x^0, t_0, t)$ the maximal solution, $x(t)$, of the differential equation $\frac{dx}{dt} = X(x, t)$ satisfying the initial condition $x(t_0) = x^0$. The goal in the numerical integration of ODE is to devise effective methods for approximating $x(t)$ on an interval $I = [t_0, T]$. The strategy that many methods use is to interpolate N equally spaced gridpoints $t_1, \ldots t_N$ in the interval I, defined by $t_k := t_0 + k\Delta t$ with $\Delta t = \frac{T-t_0}{N}$, and then use some rule to define values $x^1, \ldots, x^N$ in V, in such a way that when N is large each x^k is close to the corresponding $x(t_k)$. The quantity $\max_{1\leq k\leq N} \left\|x^k - x(t_k)\right\|$ is called the *global error* of the algorithm, and if it converges to zero as N tends to infinity (for every choice of X, t_0, x^0, and T), then we say that we have a *convergent algorithm*.

One common way to construct the algorithm that produces the values $x^1, \ldots, x^N$ uses a recursion based on a so-called "stepping procedure", namely a function, $\Sigma(X, x^0, t_0, \Delta t)$, having as inputs:

1) a time-dependent vector field X on V,
2) an initial condition x^0 in V,
3) an initial time t_0 in $\mathbf{R}$, and
4) a "time-step" Δt in $\mathbf{R}$,

and with output a point of V that for small Δt approximates $\sigma(X, x^0, t_0, t_0 + \Delta t)$ well. More precisely, the so-called "local truncation error", defined by

$$\left\|\sigma(X, x^0, t_0, t_0 + \Delta t) - \Sigma(X, x^0, t_0, \Delta t)\right\|,$$

should approach zero at least quadratically in the time-step Δt. Given such a stepping procedure, the approximations x^k of the $x(t_k)$ are defined recursively by $x^{k+1} = \Sigma(X, x^k, t_k, \Delta t)$. Numerical integration methods that follow this general pattern are referred to as finite difference methods.

0.3.1 Remark. There are two sources that contribute to the global error, $\left\|x^k - x(t_k)\right\|$. First, each stage of the recursion will give an additional local truncation error added to what has already accumulated up to that point. But, in addition, after the first step, there will be an error because the recursion uses $\Sigma(X, x^k, t_k, \Delta t)$ rather than the unknown $\Sigma(X, x(t_k), t_k, \Delta t)$. (In practice there is a third source of error, namely machine round-off error from using floating-point arithmetic. We will usually ignore this and pretend that our computers do precise real number arithmetic, but there are situations where it is important to take it into consideration.)

For Euler's Method the stepping procedure is simple and natural. It is defined by:

Euler Step

$\Sigma^E(X, x^0, t_0, \Delta t) := x^0 + \Delta t\, X(x^0, t_0)$.

It is easy to see why this is a good choice. If as above we denote $\sigma(X, x^0, t_0, t)$, by $x(t)$, then by Taylor's Theorem,

$$\begin{aligned} x(t_0 + \Delta t) &= x(t_0) + \Delta t\, x'(t_0) + O(\Delta t^2) \\ &= x^0 + \Delta t\, X(x^0, t_0) + O(\Delta t^2) \\ &= \Sigma^E(X, x^0, t_0, \Delta t) + O(\Delta t^2), \end{aligned}$$

so $\big\|\sigma(X, x^0, t_0, t_0 + \Delta t) - \Sigma(X, x^0, t_0, \Delta t)\big\|$, the local truncation error for Euler's Method, does go to zero quadratically in Δt. When we partition $[0, T]$ into N equal parts, $\Delta t = \frac{T - t_0}{N}$, each step in the recursion for computing x^k will contribute a local truncation error that is $O(\Delta t^2) = O(\frac{1}{N^2})$. Since there are N steps in the recursion and at each step we add $O(\frac{1}{N^2})$ to the error, this suggests that the global error will be $O(\frac{1}{N})$, and hence will go to zero as N tends to infinity, and this can be proved rigorously, so Euler's Method is convergent.

One excellent general purpose finite difference method for solving IVPs, and it goes by the name Runge-Kutta—or more properly the fourth order Runge-Kutta Method—since there is a whole family of Runge-Kutta methods. The stepping procedure for fourth order Runge-Kutta is:

Runge-Kutta Step

$\Sigma^{RK^4}(X, x_0, t_0, \Delta t) := x_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$, where:
$k_1 = \Delta t\, X(x_0, t_0)$
$k_2 = \Delta t\, X(x_0 + \frac{1}{2}k_1, t_0 + \frac{\Delta t}{2})$
$k_3 = \Delta t\, X(x_0 + \frac{1}{2}k_2, t_0 + \frac{\Delta t}{2})$
$k_4 = \Delta t\, X(x_0 + k_3, t_0 + \Delta t)$

Runge-Kutta is fourth order, meaning that the local truncation error goes to zero as the fifth power of the step-size, and the global error as the fourth power. So if for a fixed step-size we have attained an accuracy of 0.1, then with one-tenth the step-size (and so ten times the number of steps and ten times the time) we can expect an accuracy of 0.00001, whereas with the Euler method, ten times the time would only increase accuracy from 0.1 to 0.01.

Fourth Matlab Project.

Write a Matlab M-File function `Euler(X,x0,T,n)`that estimates $x(T)$, the solution at time T of the initial value problem $\frac{dx}{dt} = X(x)$, $x(0) = x0$ by applying the Euler stepping method to the interval $[0, T]$ with n time steps. Similarly write such a function `RungeKutta(X,x0,T,n)` that uses the Runge-Kutta stepping method. Make some experiments using the case $V = \mathbf{R}$, $\frac{dx}{dt} = x$ with $x0 = 1$ and $T = 1$, so that $x(T) = e$. Check how large n has to be to get various degrees of accuracy using the two methods.

▷Project 5. Frobenius Theorem and Algorithm F

In this project you will develop a Matlab implementation of what we call Algorithm F. First we recall the mathematics behind the algorithm. We start with two vector fields X^1 and X^2 on $\mathbf{R}^n$ that depend on two real parameters t_1 and t_2. That is, for $i = 1, 2$, X^i is a map (assumed to be C^k with $k \geq 2$) of $\mathbf{R}^n \times \mathbf{R}^2 \to \mathbf{R}^n$,

$$(x_1, \ldots, x_n, t_1, t_2) \mapsto (X^i_1(x_1, \ldots, x_n, t_1, t_2), \ldots, X^i_n(x_1, \ldots, x_n, t_1, t_2))$$

and our goal is to solve a certain initial value problem $(*)$ on a rectangle r in $\mathbf{R}^2$ given by $r := \{(t_1, t_2) \in \mathbf{R}^2 \mid a_1 \leq t_i \leq b_i, \;\; i = 1, 2\}$. This means that we are looking for a function $x(t_1, t_2) \to \mathbf{R}^n$ defined on r that has an assigned "initial value" $x^0 \in \mathbf{R}^n$ at the "bottom left corner" (a_1, a_2) of r and that satisfies the two first order partial differential equations $\frac{\partial x}{\partial t_i} = X^i(x, t_1, t_2)$, for $i = 1, 2$:

$$(*) \qquad \begin{aligned} &1) \quad x(a_1, a_2) = x^0, \\ &2) \quad \frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2), \\ &3) \quad \frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2), \end{aligned}$$

Algorithm F for Constructing Solutions of the System $(*)$.

We next describe the algorithm that we will refer to as Algorithm F for constructing the solution $x(t_1, t_2)$ to $(*)$ in r provided a solution exists. The algorithm will produce a map $(t_1, t_2) \mapsto x(t_1, t_2)$, defined in r and for which the following five statements a) to e) are valid:

a) $x(t_1, t_2)$ satisfies the initial value condition 1) of $(*)$,

b) $x(t_1, t_2)$ satisfies 2) of $(*)$, along the line $t_2 = a_2$ (i.e., the bottom edge of r) .

c) $x(t_1, t_2)$ satisfies 3) of $(*)$ in all of r,

d) $x : r \to V$ is C^k,

e) The properties a), b), c) uniquely determine the function $x(t_1, t_2)$ in r, hence it will be the unique solution of $(*)$ in r if a solution exists.

The strategy behind Algorithm F comes from a change in point of view. Instead of regarding 2) and 3) of $(*)$ as a pair of coupled PDE for $x(t_1, t_2)$ with independent variables t_1 and t_2, we consider them as two independent ODEs, the first with t_1 as independent variable and t_2 as parameter, and the second with these roles reversed.

In more detail, we first we solve the initial value problem $\frac{dy}{dt} = X^1(y, t, t^0_2)$ and $y(a_1) = x^0$ on $[a_1, b_1]$ and define $x(t, a_2) = y(t)$ for $a_1 \leq t \leq b_1$. This makes statements a) and b) true, and moreover, by the uniqueness of solutions of the IVP for ODEs, conversely if a) and b) are to hold then we must define $x(t, a_2)$ this way for t in $[a_1, b_1]$.

Then, for each $t_1 \in [a_1, b_1]$ we define $x(t_1, t)$ for $a_2 \leq t \leq b_2$ as follows. We note that $x(t, a_2)$ has already been defined in the preceding step, so we solve the IVP $\frac{dz}{dt} = X^2(z, t_1, t)$ and $z(a_2) = x(t, a_2)$ and define $x(t_1, t) = z(t)$ for $t \in [a_2, b_2]$. This extends the definition of $x(t_1, t_2)$ to the remainder or r, and it clearly is the unique definition that will make c) valid.

[That completes the formal description of Algorithm F. We now restate it in less formal and more intuitive language. First find $x(t_1, t_2)$ along the line $t_2 = a_2$ by freezing the value of t_2 at a_2 and regarding the partial differential equation $\frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2)$ as an ODE in which t_2 is just a parameter. Then, regard the PDE $\frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2)$ as an ODE in which t_1 is a parameter, and for each parameter value t_1 in $a_1, b_1]$ solve this ODE, taking for initial value at $t_2 = a_2$ the value $x(t_1, a_2)$, found in the first step.]

Remark As we saw in the lecture notes, the function $x(t_1, t_2)$ produced by Algorithm F does not necessarily satisfy 2) of $(*)$ except along the line $t_2 = a_2$ (where it does by construction). On the other hand we saw that by exploiting the "equality of cross-derivatives" principle we could develop a simple condition on the two vector fields X^1 and X^2 (that we called "compatiblity") that turned out to be a necessary and sufficient condition for Algorithm F to always produce a solution of $(*)$. Here is the definition:

0.3.2 Definition. Let $X^1 : \mathbf{R}^n \times \mathbf{R}^2 \to \mathbf{R}^n$ and $X^2 : \mathbf{R}^n \times \mathbf{R}^2 \to \mathbf{R}^n$ be C^2 vector fields on $\mathbf{R}^n$ depending on two real parameters t_1 and t_2. We will call X^1 and X^2 *compatible* if the following n conditions hold identically:

$$\frac{\partial X_i^1}{\partial t_2} + \sum_{j=1}^{n} \frac{\partial X_i^1}{\partial x_j} X_j^2 = \frac{\partial X_i^2}{\partial t_1} + \sum_{j=1}^{n} \frac{\partial X_i^2}{\partial x_j} X_j^1, \qquad 1 \leq i \leq n.$$

The fact that compatibility is necessary and sufficient for the output of Algorithm F to be a solution of the IVP $(*)$ for all choices of initial conditions is the content of the Frobenius Theorem.

0.4 Fifth Matlab Project.

Your assignment for the fifth project is to implement Algorithm F in Matlab. This should consist of an M-File, AlgorithmF.m, defining a Matlab function AlgorithmF(X1,X2,...), together with various auxilliary M-Files that implement certain subroutines required by the algorithm. (As usual, it is a matter of programming taste to what extent you use subfunctions as opposed to functions defined in separate M-Files.)

Let's consider in more detail just what the inputs and output to AlgorithmF should be. First, the output, x, should represent the function $x(t_1, t_2)$ that solves the IVP $(*)$. Since we are going to get this solution by solving some ODEs numerically (using Runge-Kutta), in Matlab x will be a two-dimensional array x(i,j) of vectors of length n,

```
function       x = AlgorithmF(X1,X2,x0,a1,a2,b1,b2,...)
```

The size of the output array x will be given by two positive integers, T1Res and T2Res that specify the number of subintervals into which we divide the intervals [a1,b1] and [a2,b2]. Let's define h1 := (b1 - a1)/T1Res and h2 := (b2 - a2)/T2Res. We take T1Res + 1 subdivision points in [a1,b1], a1 + i * h1, i = 0,1, ..., T1Res, and similarly we take T2Res + 1 subdivision points [a2,b2], a2 + j * h2, j = 0,1, ..., T2Res, It will be convenient to store these in arrays T1 and T2 of length T1Res + 1 and T2Res + 1 respectively. That is, T1(i) = a1 + i * h1 and T2(i) = a2 + i * h2. Then the array x(i,j) will have size T1Res + 1 by T2Res + 1. We will store at x(i,j) the approximate value of the solution $x(t_1, t_2)$ of $(*)$ at the point (T1(i),T2(j)), found by solving the ODEs we mentioned using Runge-Kutta. So now the first line of our M-File has become:

```
function       x = AlgorithmF(X1,X2,x0,a1,a2,b1,b2,T1Res,T2Res,...)
```

We need one more input parameter, namely a real number StepSize to control the accuracy of the Runge-Kutta algorithm. StepSize is not the actual size of the steps used in the Runge-Kutta integration, but rather an upper bound for it. When we propagate the solution of an ODE y(t) from a value t = t0 where we already know it to a next value t = t0 + h where we need it, we will divide h in a number N of equal steps to make h/N less than StepSize and use that many steps in our Runge-Kutta method. (Since we will usually settle for accuracy of about 10^{-8} and Runge-Kutta is fourth order, in practice we usually take StepSize approximately 0.01). So finally the first line of our M-File has become:

```
function       x = AlgorithmF(X1,X2,x0,a1,a2,b1,b2,T1Res,T2Res,StepSize)
```

The first two input parameters X1 and X2 represent the vector fields defining the system of PDE we are dealing with. Each is a function of $n + 2$ variables, x1, x2,...,xn,t1,t2, In practice the actual functions substituted for these parameters will be taken from functions defined either in an M-File or an inline expression.

Writing and Testing the Algorithm F Code

Once you understand the above discussion well you should find it straightforward to actually write the code for AlgorithmF. Start by assigning to x(0,0) the value x0, Then, for i = 0 to T1Res, inductively find x(i+1,0) from x(i,0) by using Runge-Kutta to solve the ODE $\frac{\partial x}{\partial t1} = X1(x, t1, a2)$ on the interval [T1(i),T1(i+1)] with initial value x(i,0) at time t1 = T1(i). Then, in a similar manner, for each i from 0 to T1Res, and each j from 0 to T2Res, inductively find x(i,j+1) from x(i,j) by applying Runge-Kutta to solve the ODE $\frac{\partial x}{\partial t2}$ =X2(x,T1(i),t2) on the interval [T2(j),T2(j+1)] with initial value x(i,j) at time t2 = T2(j).

After the solution array x is constructed, it should be displayed either in wireframe (using meshgrid) or in patch mode (using surf).

Here is an "extra credit" addition to Project 5. Write an M-File defining a function to checks whether the two vector fields X1 and X2 are compatible. I suggest that you do this by checking numerically whether the two sides of the n compatibility conditions are equal at the points (T1(i),T2(j)). Here, to allow for roundoff errors, "equal" should mean that the absolute value of the difference is less than some tolerance. Use centered differences to compute the partial derivatives. See if you can make your test of equality "scale invariant".

This means that if it succeeds or fails for X1 and X2, it should do the same if you multiply both X1 and X2 by the same scalar.

Algorithm F Test Case

As you probably realize by now, in order to have confidence in the correctness of computer programs, it is important to test them carefully—and in order to test them, one needs inputs for which the correct output is known. Since it may not be not entirely obvious how to construct a good test case for Algorithm F, let me suggest one possibility.

Recall that the spherical polar coordinates of a point x in $\mathbf{R}^3$ with cartesian coordinates (x_1, x_2, x_3) are defined by $r := \|x\|$, $\phi := \tan^{-1}(x_2/x_1)$ $\theta := \cos^{-1}(x_3/r)$. In the other direction, $x_1 = r\sin(\theta)\cos(\phi)$, $x_2 = r\sin(\theta)\sin(\phi)$, and $z := r\cos(\theta)$.

Let's use t_1 to denote the colatitude θ and t_2 to denote the longitude ϕ. Then we get a parametrization of the sphere through a point x and centered at the origin, by $(t_1, t_2) \mapsto \|x\|\,(\sin(t_1)\cos(t_2), \sin(t_1)\sin(t_2), \cos(t_1))$, with $0 \le t_1 \le \pi$, and $0 \le t_2 \le 2\pi$.

If we now differentiate with respect to t_1 and t_2, we find that these parametrizations of the family of spheres centered at the origin are solutions of the first order system of PDE:

$$\frac{\partial x}{\partial t_1} = X^1(x, t_1, t_2),$$
$$\frac{\partial x}{\partial t_2} = X^2(x, t_1, t_2),$$

where X^1 and X^2 are the maps $\mathbf{R}^3 \times \mathbf{R}^2 \to \mathbf{R}^3$ given by:

$$X^1(x, t_1, t_2) := \|x\|\,(\cos(t_1)\cos(t_2), \cos(t_1)\sin(t_2), -\sin(t_1)),$$
$$X^2(x, t_1, t_2) := \|x\|\,(-\sin(t_1)\sin(t_2), \sin(t_1)\cos(t_2), 0).$$

When you have finished defining AlgorithmF and want to test it, try it with this choice of X1 and X2, and use (0,0,r) as an initial condition at time t1 = t2 = 0. If you display the solution x using meshgrid, you should see a sphere of radius r displayed with latitude and longitude gridlines.

▷Project 6. Fundamental Theorem of Plane Curves

This Matlab project is concerned in part with the visualization and animation of curves. Before getting into the details of the project, I would like to make a few general remarks on the subject of mathematical visualization that you should keep in mind while working on this project—or for that matter when you have any programming task that involves visualization and animation of mathematical objects.

1) How should you choose an error tolerance?

First, an important principle concerning the handling of errors in any computer graphics context. Books on numerical analysis tell you how to estimate errors and how to keep them below a certain tolerance, but they cannot tell you what that tolerance should be—that must depend on how the numbers are going to be used. Beginners often assume they should aim for the highest accuracy their programming system can provide—for example fourteen decimal places for Matlab. But that will often be far more than is required for the task at hand, and as you have already seen, certain algorithms may require a very long time to attain that accuracy. The degree of one's patience hardly seems to be the best way to go about choosing an error tolerance.

In fact, there is often is a more rational way to choose appropriate error bounds. For example, in financial calculations it makes no sense to compute values with an error less than half the smallest denomination of the monetary unit involved. And when making physical calculations, it is useless to calculate to an accuracy much greater than can be measured with the most precise measuring instruments available. Similarly, in carpentry there is little point to calculating the length of a board to a tolerance less than the width of the blade that will make the cut.

This same principle governs in mathematical visualization. My approach is to choose a tolerance that is "about half a pixel", since any higher accuracy won't be visible anyway. It is usually fairly easy to estimate the size of a pixel. There are roughly 100 pixels per inch, so for example if you are are graphing in a six inch square window, and the axes go from minus one to one, then six hundred pixels equals two length units, so half a pixel accuracy means a tolerance of $\frac{1}{600}$ or roughly 0.002.

1) How should you represent a curve?

Mathematically a curve in $\mathbf{R}^n$ is given by a map of an interval $[a, b]$ into $\mathbf{R}^n$. We can only represent the curve on a computer screen when $n = 2$ or $n = 3$. Let's consider the case of plane curves ($n = 2$) first. If $\alpha(t) = (x(t), y(t))$ then for any N we can divide the interval $[a, b]$ into N equal subintervals of length $h = \frac{b-a}{N}$, namely $[t_k, t_{k+1}]$, where $t_k = a + kh$ and $k = 0, \ldots, N(-1$. We associate to α and N an approximating "N-gon" α_N (i.e., a polygon with N sides) with vertices $v_k := (x(t_k), y(t_k))$. It is some α_N with N suitably large) that actually gets drawn on the computer screen when we want to display α. This reduces the actual drawing problem to that of drawing a straight line segment, and the latter is of course built into every computer system at a very low level.

In Matlab the code for plotting the curve α, or rather the polygon α_{30} would be:

```
N = 30
h = (b-a)/N;
t = a:h:b ;
plot(x(t),y(t)), axis equal;
```

To plot a curve $\alpha(t) = (x(t), y(t), z(t))$ in $\mathbf{R}^3$ is really no more difficult. In Matlab the only change is that the last line gets replaced by:

```
plot3(x(t),y(t),z(t)), axis equal;
```

only now one has to be more careful about interpreting just what it is that one sees on the screen in this case. The answer is that one again is seeing a certain polygon in the plane, but now it is the projection of the polygon in $\mathbf{R}^3$ with vertices at $v_k := (x(t_k), y(t_k), z(t_k))$. (The projection can be chosen to be either an orthographic projection in some direction or else a perspective projection from some point.)

1) How do you create animations?

Viisualization can be a powerful tool for gaining insight into the nature of complex mathematical objects, and frequently those insights can be further enhanced by careful use of animation. Remember that time is essentially another dimension, so animations allow us to pack a lot more information onto a computer screen in a format that the human brain can easily assimilate. The number of ways that animation can be used are far to numerous to catalog here, but in addition to obvious ones, such as rotating a three dimensional object, one should also mention "morphing". Mathematical objects frequently depend on several parameters (e.g., think of the family of ellipses: $x = a\cos(\theta)$, $y = b\sin(\theta)$). Morphing refers to moving along a curve in the space of parameters and creating frames of an animation as you go.

All animation techniques use the same basic technique—namely showing a succession of "frames" on the screen in rapid succession. If the number of frames per second is fast enough, and the change between frames is small enough, then the phenomenon of "persistence of vision" creates the illusion that one is seeing a continuous process evolve. Computer games have become very popular in recent years, and they depend so heavily on high quality animation that the video hardware in personal computers has improved very rapidly. Still, there are many different methods (and tricks) involved in creating good animations, and rather than try to cover them here we will have some special lectures on various animation techniques, with particular emphasis on how to implement these techniques in Matlab.

Matlab Project # 6.

Your assignment for the sixth project is to implement the Fundamental Theorem of Plane Curves using Matlab. That is, given a curvature function $k : [0, L] \to \mathbf{R}$, construct and plot a plane curve $x : [0, L] \to \mathbf{R}^2$ that has k as its curvature function. To make the solution unique, take the initial point of x to be the origin and its initial tangent direction to be the direction of the positive x-axis. You should also put in an option to plot the evolute of the curve as well as the curve itself. Finally see if you can build an animation that plots the osculating circle at a point that moves along the curve x. For uniformity, name your M-File PlaneCurveFT, and let it start out:

`function x = PlaneCurveFT(k,L,option)`

If option is not given (i.e., nargin = 2) or if option = 0, then just plot the curve x. If option = 1, then plot x and, after a pause, plot its evolute in red. Finally, if option = 2, then plot x and its evolute, and then animate the osculating circle (in blue) along the curve, also drawing the radius from the center of curvature.

[To find the curve x, you first integrate k to get $\overrightarrow{t} = x'$, and then integrate $\overrightarrow{t}$. The curvature, k, will be given as a Matlab function, so you can use the version of Simpson's Rule previously discussed for the first integration. But $\overrightarrow{t}$ will not be in the form of a Matlab function that you can substitute into that version of Simpson's Rule, so you will need to develop a slightly modified version of Simpson's. where the input is a matrix that gives the values of the integrand at the nodes rather than the integrand as a function.]

▷Project 7. Fundamental Theorem of Space Curves

Your assignment for the seventh Matlab project is to implement the Fundamental Theorem of Space Curves. That is, given a (positive) curvature function $k : [0, L] \to \mathbf{R}$, and a torsion function $\tau : [0, L] \to \mathbf{R}$, construct and plot a space curve x that has k as its curvature function and τ as its torsion function. To make the solution unique, take the initial point of x to be the origin and its initial tangent direction to be the direction of the positive x-axis. You should also use plot3 to plot the curve. See if you can create an animation that moves the Frenet Frame along the curve. For uniformity, name your M-File SpaceCurveFT, and let it start out:

```
function       x = SpaceCurveFT(k,tau,L)
```

Note that a problem analagous to the one mentioned in Project # 6 will appear again hear. To get the Frenet frame along the curve you will solve the Frenet equations, and for this you can use the Runge-Kutta algorithm that you developed earlier. Then, to obtain the curve x you will need to integrate its tangent vector, i.e., the first element of the Frenet frame that you just derived, and since the tangent vector is not in the form of a Matlab function, you will need to use the version of Simpson's Rule developed for Project # 6 where the input is a matrix that contains the values of the integrand at the nodes.

▷Project 8. Fundamental Theorem of Surfacess

This final Matlab project asks you to implement the Fundamental Theorem of Surface Theory as a Matlab function. It is a complicated problem and to manage the complexity successfully you will have to organize your work carefully and work slowly and deliberately. When you have completed this project, I think you will have adequate excuse to feel proud of both your programming skill and your comprehension of the mathematics involved.

Review of Notation and Definitions

Before stating the project we need to recall some definitions and notational conventions that will be used. $\mathcal{O}$ will be the rectangle $[a1, b1] \times [a2, b2]$ in $\mathbf{R}^2$. A point in $\mathcal{O}$ will be denoted by p or (t_1, t_2) in a mathematical context or `(t1,t2)` in a Matlab context. We have two 2×2 symmetric matrix-valued functions, g and ℓ defined in $\mathcal{O}$: $g = (g_{ij})$ and $\ell = (\ell_{ij})$. The matrix g should be positive definite, so in particular it is invertible and we denote its inverse by $g^{-1} = (g^{ij})$. By Cramer's Rule:

$$g^{-1} = \frac{1}{\det(g)} \begin{pmatrix} g_{22} & -g_{12} \\ -g_{12} & g_{11} \end{pmatrix},$$

i.e., $g^{11} = g_{22}/\det(g)$, $g^{22} = g_{11}/\det(g)$, and $g^{12} = g^{21} = -g_{12}/\det(g)$, where $\det(g)$ is the determinant of g, given by $\det(g) := g_{11}g_{22} - g_{12}^2$. We also have corresponding positive definite 3×3 symmetric matrices G and G^{-1} defined in $\mathcal{O}$ by:

$$G = \begin{pmatrix} g_{11} & g_{12} & 0 \\ g_{12} & g_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and hence

$$G^{-1} = \begin{pmatrix} g^{11} & g^{12} & 0 \\ g^{12} & g^{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

We also have two 3×3 matrix-valued functions A^1 and A^2 defined in $\mathcal{O}$ by:

$$A^1 = \begin{pmatrix} \frac{1}{2}(g_{11})_{t_1} & \frac{1}{2}(g_{11})_{t_2} & -\ell_{11} \\ (g_{12})_{t_1} - \frac{1}{2}(g_{11})_{t_2} & \frac{1}{2}(g_{22})_{t_1} & -\ell_{12} \\ \ell_{11} & \ell_{12} & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} \frac{1}{2}(g_{11})_{t_2} & (g_{12})_{t_2} - \frac{1}{2}(g_{22})_{t_1} & -\ell_{12} \\ \frac{1}{2}(g_{22})_{t_1} & \frac{1}{2}(g_{22})_{t_2} & -\ell_{22} \\ \ell_{12} & \ell_{22} & 0 \end{pmatrix}$$

and finally, there are two further 3×3 matrix-valued functions in $\mathcal{O}$, $P^k := G^{-1}A^k$.

The Gauss-Codazzi Equations

If g_{ij} and ℓ_{ij} are the coefficients of the First and Second Fundamental Forms of a surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$, then the matrix-valued functions P^1 and P^2 defined in $\mathcal{O}$ as above satisfy the matrix identity

$$P^1_{t_2} - P^2_{t_1} = P^1 P^2 - P^2 P^1$$

called the Gauss-Codazzi Equations.

Statement of the Project

The primary Matlab M-File should be called SurfaceFT.m and should start out:

```
function   F = SurfaceFT(g11,g12,g22,l11,l12,l22,a1,b1,a2,b2,T1Res,T2Res)
```

where $I := \sum_{ij}$ `gij(t1,t2) dti dtj` and $II := \sum_{ij}$ `lij(t1,t2) dti dtj` are quadratic forms in $\mathcal{O}$, and the function $F : \mathcal{O} \to \mathbf{R}^3$ returned is supposed to be the surface having I and II as its First and Second Fundamental Forms. For this surface to exist, we know from the Fundamental Theorem that it is necessary and sufficient that I be positive definite and that the Gauss-Codazzi equations be satisfied.

Of course the heavy lifting of the SurfaceFT will be done by AlgorithmF (i.e., the solution of the Frobenius Problem) which you will apply to integrate the frame equations in order to get the frame field $\mathfrak{f}$—after which you must apply AlgorithmF a second time to get the surface $\mathcal{F}$ from $\mathfrak{f}_1$ and $\mathfrak{f}_2$.

But to carry out the first application of AlgorithmF, you must first compute the matrices $P^1 = G^{-1}A^1$ and $P^2 = G^{-1}A^2$ that define the right hand sides of the two frame equations. Recall that G^{-1} is the inverse of the 3×3 matrix

$$G = \begin{pmatrix} \texttt{g11(t1,t2)} & \texttt{g12(t1,t2)} & 0 \\ \texttt{g12(t1,t2)} & \texttt{g22(t1,t2)} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and so it can be easily computed from the `gij(t1,t2)` by using Cramer's Rule, while the two 3×3 matrices A^1 and A^2 are given explicitly (see above) in terms of the `g11(t1,t2)` **and their partial derivatives with respect to the** t_i . (Of course, once you have both G^{-1} and A^i, you get P^i as their matrix product.)

In order to be able to compute the A^i, you will first need to define some auxilliary functions. Most of them can probably be subfunctions defined in the same file, though some could be separate M-Files. For example you will want to have a function called `g(t1,t2)` that returns a 2×2 matrix $\begin{pmatrix} \texttt{g11(t1,t2)} & \texttt{g12(t1,t2)} \\ \texttt{g12(t1,t2)} & \texttt{g22(t1,t2)} \end{pmatrix}$ and another called `l(t1,t2)` that returns a 2×2 matrix $\begin{pmatrix} \texttt{l11(t1,t2)} & \texttt{l12(t1,t2)} \\ \texttt{l12(t1,t2)} & \texttt{l22(t1,t2)} \end{pmatrix}$. You will then want to create the functions `G(t1,t2)` and `invG(t2,t2)` that return the 3×3 matrices G and G^{-1}.

There is another complication before you can define the Matlab functions `A1` and `A2` that represent A^1 and A^2. You not only need the functions `gij(t1,t2)` but also their first partial derivatives with respect to the variables `t1` and `t2`. I recommend that along with

the function `g(t1,t2)` you also define two more functions `g_t1(t1,t2)` and `g_t2(t1,t2)` that return 2×2 matrices whose entries are the partial derivatives of the `gij(t1,t2)` with respect to `t1` and `t2` respectively. As usual you can compute these partial derivatives using symmetric differencing—you don't need to do it symbolically.

You should define a Matlab function GaussCodazziCheck that will check whether or not the Gauss-Codazzi Equations are satisfied. Once you have defined the two 3×3 matrix-valued functions P^1 and P^2, it will be easy to write GaussCodazziCheck, since the Gauss-Codazzi equations are just $P^1_{t_2} - P^2_{t_1} = P^1P^2 - P^2P^1$. The idea is to check the identities numerically, matrix element by matrix element, at a sufficiently dense set of points, again using symmetric differencing to compute the derivatives.

Of course, when you are all done you will need some good test cases on which to try out you algorithm. We will discuss this elsewhere.

GOOD LUCK, AND HAVE FUN!

Appendix II
Homework and Exams

First Assignment

▷ **Problem 1.** When I want to see an initial explanation of something, I have for some time now used Google to do a search and then I read a few of the top-rated items that look promising. What I have found surprising is that this seems to be quite successful even for fairly abstract mathematical subjects.

In the second lecture I plan to explain something called Felix Klein's Erlanger program. Use Google to try to get some idea what this is about, and after I talk about it in class, tell me if you think reading about it in advance this way helped you or not.

You should do the next few exercises before the third lecture. They are designed for me to get some feedback from you on whether you find it easy to learn a topic by carrying out a short series of exercises. Please try to work through the following definitions and exercises and we will discuss in class whether you feel this is a good method for you to learn something.

We start with a definition.

Definition Let X be a set. A real-valued function ρ on $X \times X$ is called a *metric* or *distance function* for X if it satisfies the following three properties:

a) Symmetry: $\rho(x, y) = \rho(y, x)$ for all x and y in X.

b) Positivity: $\rho(x, y) \geq 0$, with equality if and only if $x = y$.

c) Triangle inequality $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$ for all x, y, z in X.

By a *metric space* we mean a set X together with some fixed metric for X. (We will usually denote this metric by ρ_X, but if in a certain context there is no danger of confusion, we will often use just ρ.)

Example 1. Take X to be $\mathbf{R}$, the real numbers, and define $\rho(x, y) = |x - y|$.

Example 2. More generally, take X to be $\mathbf{R}^n$, and define $\rho(x, y) = \|x - y\|$.

Example 3. Take X to be the sphere $\mathbf{S}^2$, and define $\rho(x, y)$ to be the length of the shorter of the segments of great circles joining x to y.

▷ **Problem 2.** Suppose $\{p_n\}$ is a sequence in the metric space X and $p \in X$. Give a definition for what it means for the sequence $\{p_n\}$ to converge to p. (Hint: A seqence $\{x_n\}$ of real numbers converges to a real number x if and only if $\lim_{n\to\infty} |x_n - x| = 0$.) We usually write either $p_n \to p$ or $\lim_{n\to\infty} p_n = p$ to denote that $\{p_n\}$ converges to p. Show that if $p_n \to p$ and $p_n \to q$ then $p = q$, i.e., limits of sequences are unique. (Hint $\rho(p, q) \leq \rho(p_n, p) + \rho(p_n, q)$.)

Now suppose that X and Y are two metric spaces and that $f : X \to Y$ is a function mapping X into Y. We say that f is a *continuous* function if whenever a sequence $\{x_n\}$ in X converges to some limit x in X, it follows that the sequence $\{f(x_n)\}$ in Y converges to $f(x)$.

Definition Let X and Y be metric spaces and $f : X \to Y$ a function from X into Y. If K is a positive real number, we say that f satisfies a Lipschitz condition with constant K

if $\rho_Y(f(x_1), f(x_2)) \leq K\rho_X(x_1, x_2)$ for all x_1 and x_2 in X, and we say that f is a Lipschitz function if it satisfies a Lipschitz condition with some constant K.

▷ **Problem 3.** Show that every Lipschitz function is continuous.

Definition A *contraction mapping* of a metric space X is a function that maps X into itself and that satisfies a Lipschitz condition with constant $K < 1$.

▷ **Problem 4.** Let X be a metric space and $f : X \to X$ a contraction mapping with Lipschitz constant $K < 1$. Prove the "Fundamental Inequality for Contraction Mappings":

$$\rho(x, y) \leq \frac{1}{1 - K}\big(\rho(x, f(x)) + \rho(y, f(y))\big)$$

holds for all x,y in X. (Hint: This is VERY easy if you apply the triangle inequality in the right way. But where does $K < 1$ come in?)

Second Assignment Due Friday, Sept.12, 2003

Some exercises on linear transformations and matrices.

▷ **Problem 1.** Let $v_1, \ldots, v_n$ be a basis for a vector space V and let S and T be two linear operators on V. If the matrices of S and T relative to this basis are respectively S_{ij} and T_{ij}, then show that the matrix elements of the composed linear operator ST are given by $(ST)_{ij} = \sum_{k=1}^{n} S_{ik}T_{kj}$, and that the matrix elements of the sum operator $S+T$ are given by $(S+T)_{ij} = S_{ij} + T_{ij}$.

In what follows, $\mathcal{P}^n$ denotes the space of polynomials functions $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ of degree $\leq n$. Clearly $\mathcal{P}^n$ is a vector space of dimension $n+1$ and $1, x, x^2, \ldots, x^n$ is a basis for $\mathcal{P}^n$ (called the standard basis).

▷ **Problem 2.** Differentiation defines an operator D on $\mathcal{P}^n$, and of course D^k denotes the k-th derivative operator.

a) What is the matrix of D in the standard basis?

b) What is the kernel of D^k?

c) What is the image of D^k?

▷ **Problem 3.** Define an inner-product on $\mathcal{P}^n$ by $\langle P_1, P_2\rangle = \int_{-1}^{1} P_1(x)P_2(x)\,dx$, and note that the standard basis is **not** orthonormal (or even orthogonal). Let us define orthonormal polynomials $L_k(x)$ by applying the Gram-Schmidt Algorithm to the standard basis. (The L_k are usually called the normalized Legendre Polynomials.) Compute L_0, L_1, and L_2.

Third Assignment Due Friday, Sept.26, 2003

Here are a few exercises concerning adjoints of linear maps. We recall that if V and W are inner-product spaces and $T : V \to W$ is a linear map then there is a uniquely determined map $T^* : W \to V$, called the adjoint of T, satisfying $\langle Tv, w\rangle = \langle v, T^*w\rangle$ for all $v \in V$ and $w \in W$.

▷ **Problem 1.** Show that $(T^*)^* = T$.

▷ **Problem 2.** Recall that if T_{ij} is an $m \times n$ matrix (i.e., m rows and n columns) and S_{ji} an $n \times m$ matrix, then S_{ji} is called the *transpose* of T_{ij} if $T_{ij} = S_{ji}$ for $1 \leq i \leq m$, $1 \leq j \leq n$. Show that if we choose orthonormal bases for V and W, then the matrix of T^* relative to these bases is the transpose of the matrix of T relative to the same bases.

▷ **Problem 3.** Show that $\ker(T)$ and $\mathrm{im}(T^*)$ are orthogonal complements in V, and similarly, $\mathrm{im}(T)$ and $\ker(T^*)$ are each other's orthogonal complements in W. (Note that by Exercise 1, you only have to prove one of these.)

▷ **Problem 4.** Show that a linear operator T on V is in the orthogonal group $\boldsymbol{O}(V)$ if and only if $TT^* = I$ (where I denotes the identity map of V) or equivalently, if and only if $T^* = T^{-1}$.

If $T : V \to V$ is a linear operator on V, then T^* is also a linear operator on V, so it makes sense to compare them and in particular ask if they are equal.

Definition. A linear operator on an inner-product space V is called *self-adjoint* if $T^* = T$, i.e., if $\langle Tv_1, v_2\rangle = \langle v_1, Tv_2\rangle$ for all $v_1, v_2 \in V$.

Note that by Exercise 3 above, self-adjoint operators are characterized by the fact that their matrices with respect to an orthonormal basis are symmetric.

▷ **Problem 5.** Show that if W is a linear subspace of the inner-product space V, then the orthogonal projection P of V on W is a self-adjoint operator on V.

Definition. If T is a linear operator on V, then a linear subspace $U \subseteq V$ is called a T-invariant subspace if $T(U) \subseteq U$, i.e., if $u \in U$ implies $Tu \in U$.

Remark. Note that if U is a T-invariant subspace of V, then T can be regarded as a linear operator on U by restriction, and clearly if T is self-adjoint, so is its restriction.

▷ **Problem 6.** Show that if $T : V \to V$ is a self-adjoint operator, and $U \subseteq V$ is a T-invariant subspace of V, the $U^\perp$ is also a T-invariant subspace of V.

We next recall for convenience the definitions relating to eigenvalues and eigenvectors. We assume that T is a linear operator on V.

If λ is a real number, then we define the linear subspace $E_\lambda(T)$ of V to be the set of $v \in V$ such that $Tv = \lambda v$. In other words, if I denotes the identity map of V, then $E_\lambda(T) = \ker(T - \lambda I)$. Of course the zero vector is always in $E_\lambda(T)$. If $E_\lambda(T)$ contains a non-zero vector, then we say that λ is an *eigenvalue* of T and that $E_\lambda(T)$ is the λ-eigenspace of T. A non-zero vector in $E_\lambda(T)$ is called an *eigenvector* of T belonging to the

eigenvector λ. The set of all eigenvalues of T is called the *spectrum* of T (the name comes from quantum mechanics) and it is denoted by $\mathrm{Spec}(T)$.

▷ **Problem 7.** Show that a linear operator T on V has a diagonal matrix in a particular basis for V if and only if each element of the basis is an eienvector of T, and that then $\mathrm{Spec}(T)$ consists of the diagonal elements of the matrix.

▷ **Problem 8.** If T is a self-adjoint linear operator on an inner-product space V and λ_1, λ_2 are distinct real numbers, show that $E_{\lambda_1}(T)$ and $E_{\lambda_2}(T)$ are orthogonal subspaces of V. In other words, eigenvectors of T that belong to different eigenvalues are orthogonal. (Hint: Let $v_i \in E_{\lambda_i}(T), i = 1, 2$. You must show that $\langle v_1, v_2 \rangle = 0$. Start with the fact that $\langle Tv_1, v_2 \rangle = \langle v_1, Tv_2 \rangle$.)

Fourth Assignment Due Tuesday, Oct. 6, 2003

▷ **Problem 1.** Let X be a metric space. Recall that a subset of X is called *open* if whenever it contains a point x it also contains all points "sufficiently close" to x, and that a subset F of X is called closed if whenever a sequence in F converges to a point p of X it follows that $p \in F$. Show that a set is open if and only if its complement is closed.

▷ **Problem 2.** Let X and Y be metric spaces, $f : X \to Y$ continuous, and A an open (resp. closed) subset of Y. Show that $f^{-1}(A)$ is open (resp. closed) in X. Deduce from this that the unit sphere, $S(V)$, in an inner-product space V is a closed and hence compact subset of V.

▷ **Problem 3.** Recall that a subset K of a metric space X is called *compact* if every sequence in K has a subsequence that converges to some point of K (the Bolzano-Weierstrass property), Show that if $f : X \to \mathbf{R}$ is continuous real-valued function on X then f must be bounded on any compact subset K of X and in fact there is a point p of K where f assumes its maximum value. (Hint # 1: If it were not bounded above on K, then there would be a sequence $k_n \in K$ such that $f(k_n) > n$. Hint #2: Choose a sequence $k_n \in K$ so that $f(k_n)$ converges to the least upper bound of the values of f on K.)

▷ **Problem 4.** Prove the so-called Chain Rule.

Chain Rule. *Let U, V, W be inner-product spaces, Ω an open set of U and O and open set of V. Suppose that $G : \Omega \to V$ is differentiable at $\omega \in \Omega$ and that $F : O \to W$ is differentiable at $p = G(\omega) \in O$. Then $F \circ G$ is differentiable at ω and $D(F \circ G)_\omega = DF_p \circ DG_\omega$.*

▷ **Problem 5.** Let $f : \mathbf{R}^n \to \mathbf{R}^m$ be a function from $\mathbf{R}^n$ to $\mathbf{R}^m$. We use the usual conventions, so that if $x = (x_1, \ldots, x_n) \in \mathbf{R}^n$ then its image point $f(x)$ has the m components $(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$. Show that if f is differentiable at a point p of $\mathbf{R}^n$ then the matrix of the differential, Df_p, with respect to the standard bases of $\mathbf{R}^n$ and $\mathbf{R}^m$, is just the so-called "Jacobian matrix" at p, namely the matrix of partial derivatives $\frac{\partial f_i}{\partial x_j}(p)$.

▷ **Problem 6.** Let $\sigma : (a, b) \to V$ and $\gamma : (a, b) \to V$ be differentiable curves in an inner-product space V. Show that $\frac{d}{dt} \langle \sigma(t), \gamma(t) \rangle = \langle \sigma'(t), \gamma(t) \rangle + \langle \sigma(t), \gamma'(t) \rangle$, and in particular $\frac{d}{dt} \|\sigma(t)\|^2 = 2 \langle \sigma(t), \sigma'(t) \rangle$. Deduce that if $\sigma : (a, b) \to V$ has its image in $S(V)$, i.e., if $\|\sigma(t)\|$ is identically one, then $\sigma(t)$ and $\sigma'(t)$ are orthogonal for ll t.

Midterm Exam Due Friday, October 24, 2003

Part I: Successive Approximation

If X is any set and $f : X \to X$ a mapping of X to itself, then for each positive integer n we define a mapping $f^{(n)} X \to X$ by composing f with itself n times. That is, $f^{(1)}(x) = f(x)$, $f^{(2)}(x) = f(f(x))$, $f^{(3)}(x) = f(f(f(x)))$, etc. To be more formal, we define the sequence $f^{(n)}$ inductively by: $f^{(1)} := f$ and $f^{(n+1)} := f \circ f^{(n)}$.

▷ **Problem 1.** Show that $f^{(n)} \circ f^{(k)} = f^{(n+k)}$.

▷ **Problem 2.** Let X be a metric space and suppose that f satisfies a Lipschitz condition with constant K. (Recall this means that $\rho(f(x_1), f(x_2)) \leq K\rho(x_1, x_2)$ for all $x_1, x_2 \in X$.) Show that $f^{(n)}$ satisfies a Lipschitz condition with constant K^n.

In what follows, we suppose that X is a metric space and that $f : X \to X$ is a contraction mapping, i.e., we assume that f satisfies a Lipschitz condition with constant $K < 1$. (We refer to K as a contraction constant for f.) We recall that in an earlier assignment you proved the so-called Fundamental Inequality For Contraction Mappings, namely, for all $x_1, x_2 \in X$,

$$\rho(x_1, x_2) \leq \frac{1}{1-K}\Big(\rho(x_1, f(x_1)) + \rho(x_2, f(x_2)\Big).$$

▷ **Problem 3.** Show that a contraction mapping $f : X \to X$ can have at **most** one fixed point, i.e., there is at most one point $x \in X$ such that $f(x) = x$.

▷ **Problem 4.** Show that if $f : X \to X$ is a contraction mapping with contraction constant K and if x is **any** point of X then

$$\rho(f^{(n)}(x), f^{(m)}(x)) \leq \left(\frac{K^n + K^m}{1-K}\right)\rho(x, f(x)),$$

and deduce that $f^{(n)}(x)$ is a Cauchy sequence.

▷ **Problem 5.** Now prove:

Banach Contraction Principle. *If X is a complete metric space and $f : X \to X$ is a contraction mapping, then f has a unique fixed point p and if x is any point of X then the sequence $\{f^{(n)}(x)\}$ converges to p.*

Important Caveat! In applications, X is frequently a closed subset of a Banach space V (hence complete) and f is some mapping from X into V for which one can prove that f satisfies a Lipschitz condition with constant $K < 1$. **But that is not enough!** One must also prove that f maps X into itself in order to apply the Contraction Principle.

▷ **Problem 6.** Show that if $f : X \to X$ is a contraction mapping and p is the unique fixed point of f, then for any x in X, $\rho(f^{(n)}(x), p) \leq \left(\frac{K^n}{1-K}\right)\rho(x, f(x))$

Remark. The sequence $\{f^{(n)}(x)\}$ is usually referred to as *the sequence of iterates of x under f*, and the process of locating the fixed point p of a contraction mapping f by taking the limit of a sequence of iterates of f goes by the name *"the method of successive approximations"*. To make this into a rigorous algorithm, we must have a "stopping rule". That is, since we cannot keep iterating f forever, we must know when to stop. One rather rough approach is to keep on iterating until successive iterates are "close enough", but a better method is provided by the previous problem. Suppose we decide to be satisfied with the approximation $f^{(n)}(x)$ if we can be sure that $\rho(f^{(n)}(x), p) \leq \epsilon$ where ϵ is some "tolerance" given in advance. We first compute $f(x)$, then $\rho(f(x), x)$, and then solve $\left(\frac{K^n}{1-K}\right)\rho(x, f(x)) = \epsilon$ for n and iterate $n-1$ more times to get our acceptable approximation $f^{(n)}(x)$ to p.

▷ **Problem 7.** Solve $\left(\frac{K^n}{1-K}\right)\rho(x, f(x)) = \epsilon$ for n in terms of ϵ, K, and $\rho(x, f(x))$.

▷ **Problem 8.** Carry out the following (third) Matlab Project. (Make a printout of your version of the M-File and submit it with the exam, but also send an electronic version to Izi and me as an email attachment.)

Third Matlab Project.

Write an Matlab M-file that implements the Successive Approximations algorithm. Name it SuccessiveApprox.m, and use it to define a Matlab function SuccessiveApprox(f, K, x, eps). Assume that $f : \mathbf{R}^n \to \mathbf{R}^n$ is known to be a contraction mapping with contraction constant K, that $x \in \mathbf{R}^n$, and you want to compute iterates of x until you are within eps of the fixed point p of f. Use a subfunction to compute the number of times n you need to iterate f starting from x to get within eps of p, and then use a loop and feval to iterate applying f to x the appropriate number of times.

Part II: Inverse Function Theorems

I don't think you will need any convincing that "solving equations" is an essential mathematical activity. For us, solving an equation will mean that we have normed spaces V and W, a map f from a subset X of V into W, and given $y \in W$ we would like to find (some or all) $x \in X$ such that $f(x) = y$. In practice, what frequently happens is that we start with an x_0 and y_0 satisfying $f(x_0) = y_0$, and given y close to y_0 we would like to find the x close to x_0 that satisfy $f(x) = y$. A so-called "inverse function theorem" is a theorem to the effect that, under certain assumptions on f, for each y in some neighborhood U of y_0, there is a **unique** $x = g(y)$ near x_0 that solves $f(x) = y$. In this case g is called the local inverse for f near x_0. Perhaps the mother of all inverse function theorems is the Lipschitz Inverse Function Theorem, which we state below after reviewing some standard notation.

Notation. In the following, I denotes the identity mapping of the space V, so $F_0 = I - f$ means the mapping $F_0(x) = x - f(x)$ and more generally for any y in V, $F_y := I - f + y$ means the map $F_y(x) := x - f(x) + y$. Also we denote the closed ball of radius ϵ in V centered at v_0 by $\bar{B}_\epsilon(v_0, V) := \{v \in V \mid \|v - v_0\| \leq \epsilon\}$.

Lipschitz Inverse Function Theorem. *Let V be a Banach space and $f : O \to V$ a*

map of a neighborhood of the origin into V such that $f(0) = 0$, and suppose also that $F_0 := I - f$ satisfies a Lipschitz condition with Lipschitz constant $K < 1$. Then:
1) f satisfies a Lipschitz condition with constant $1 + K$.
2) For $r > 0$ small enough that, $\bar{B}_r(0, V) \subseteq O$, the map $F_y := I - f + y$ is a contraction mapping of $\bar{B}_r(0, V)$, provided y in $\bar{B}_{(1-K)r}(0, V)$.
3) For each y in $\bar{B}_{(1-K)r}(0, V)$, there is a unique $x = g(y)$ in $\bar{B}_r(0, V)$ such that $f(x) = y$.
4) This inverse map $g : \bar{B}_{(1-K)r}(0, V) \to V$ satisfies a Lipschitz condition with Lipschitz constant $\frac{1}{1-K}$.

Some general hints for the following problems, The next problems will lead you through the proof of the Lipschitz Inverse Function Theorem, and the following hints may be of some help. But first try to do the problems without looking at the hints. (You quite possibly will think of some or all of them on your own anyway.)

a) Note that (since the "y"s cancel) $F_y(v_1) - F_y(v_2) = F_0(v_1) - F_0(v_2)$, so any Lipschitz condition satisfied by one of the F_y is also satisfied by all the others.
b) If you write out what it means to say that F_0 satisfies a Lipschitz condition with constant K (and rearrange terms a bit) you will find $\|(f(v_1) - f(v_2) - (v_1 - v_2)\| \leq K \|v_1 - v_2\|$.
c) What does it mean for x to be a fixed point of F_y?
d) You are probably used to thinking of the triangle inequality in the form $\|x + y\| \leq \|x\| + \|y\|$, but if you replace x by $x - y$ you end up with $\|x\| \leq \|x - y\| + \|y\|$, and quite often it is this second form of the triangle inequality that is easier to apply.

▷ **Problem 9.** Prove conclusion 1) of the Lipschitz Inverse Function Theorem.

▷ **Problem 10.** Prove conclusion 2) of the Lipschitz Inverse Function Theorem. (Hint: The only slightly tricky part is showing that F_y maps $\bar{B}_r(0, V)$ to itself provided $\|y\| \leq (1 - K)r$, i.e., that for such y, if $\|x\| \leq r$, then also $\|F_y(x)\| \leq r$. See the "Important Caveat!" immediately after the statement of the Banch Contraction Principle.)

▷ **Problem 11.** Prove conclusion 3) of the Lipschitz Inverse Function Theorem. (Hint: You may want to look at general hint c) for this.)

▷ **Problem 12.** Prove conclusion 4) of the Lipschitz Inverse Function Theorem.

Remark. The principle application of the Lipschitz Inverse Function Theorem is as a lemma to prove the (far more important) Differentiable Inverse Function Theorem. We consider this next.

More Notation. If V and W are normed spaces, then we denote by $L(V, W)$ the space of all continuous linear maps $T : V \to W$. (If V and W finite dimensional, then **every** linear map $T : V \to W$ is automatically continuous, so this is consistent with our earlier use of $L(V, W)$.) We saw that there was a natural choice of norm for $L(V, W)$, namely $|||\, T \,||| := \sup_{\|v\|=1} \|Tv\|$, or equivalently, $|||\, T \,||| := \sup_{\|v\|\neq 0} \frac{\|Tv\|}{\|v\|}$. We also saw that $|||\, T \,|||$ was the smallest Lipschitz constant for T. If O is open in V and $F : O \to W$ is differentiable at every point of O, then we have a map $DF : O \to L(V, W)$ called the differential of F, namely $p \mapsto DF_p$, and we say that F is C^1 (or continuously differentiable) in O if this mapping is continuous. We recall also that if O is convex and if $|||\, DF_p \,||| \leq K$ for all $p \in O$ then we showed that K was a Lipschitz constant for F.

▷ **Problem 13.** Assume that $F : O \to W$ is C^1, $p_0 \in O$ and $K > |||\, DF_{p_0} \,|||$. Show that there is a neighborhood U of p_0 such that K is a Lipschitz constant for f restricted to U.

Differentiable Inverse Function Theorem (1st Case). *Let V be a Banach space and $f : O \to V$ a C^1 map of a neighborhood of the origin into V such that $f(0) = 0$ and $Df_0 = I$, the identity map of V. If $\epsilon > 0$ is sufficiently small, then there is an $r > 0$ and a unique "inverse" map $g : B_r(0, V) \to B_\epsilon(0, V)$ such that $f(g(v)) = v$ for all v in $B_r(0, V)$. Moreover g is differentiable at the origin and $Dg_0 = I$.*

▷ **Problem 14.** Prove the 1st case of the Differentiable Inverse Function Theorem. Hint: First show that if ϵ is sufficiently small then $I - f$ satisfies a Lipschitz condition with constant $\frac{1}{2}$ in $B_\epsilon(0, V)$ and apply the Lipschitz Inverse Function Theorem.

Remark. The only hard part of this problem is showing that $Dg_0 = I$. But don't worry if you cannot prove that—it's an "extra credit" problem.

Differentiable Inverse Function Theorem (2nd Case). *Let V and W be Banach spaces and $F : O \to W$ a C^1 map of a neighborhood of the origin of V into W such that $f(0) = 0$ and DF_0 has a continuous linear inverse. If $\epsilon > 0$ is sufficiently small, then there is an $r > 0$ and a unique "inverse" map $G : B_r(0, W) \to B_\epsilon(0, V)$ such that $F(G(w)) = w$ for all w in $B_r(0, W)$. Moreover G is differentiable at the origin of W and $DG_0 = (DF_0)^{-1}$.*

▷ **Problem 15.** Prove the 2nd case of the Differentiable Inverse Function Theorem. Hint: Prove this by reducing it to the 1st case of the Differentiable Inverse Function Theorem. Namely, define $f : O \to V$ by $f := (DF_0)^{-1} \circ F$, and carry on from there.

And finally, the following general case of the Differentiable Inverse Function Theorem follows from the "2nd Case" simply by replacing $F(v)$ by $F(v + v_0) - F(v_0)$!

Differentiable Inverse Function Theorem. *Let V and W be Banach spaces, $v_0 \in V$, and $F : O \to W$ a C^1 map of a neighborhood of v_0 in V into W such that DF_{v_0} has a continuous linear inverse. If $\epsilon > 0$ is sufficiently small, then there is an $r > 0$ and a unique "inverse" map $G : B_r(F(v_0), W) \to B_\epsilon(x_0, V)$ such that $F(G(w)) = w$ for all w in $B_\epsilon(0, W)$. Moreover G is also C^1, and in fact, if $v = G(w)$ then $DG_w = (DF_v)^{-1}$.*

Fifth Assignment Due Friday, Nov. 21, 2003

▷ **Problem 1.** Show that if $\alpha(t) = (x(t), y(t))$ is a plane parameterized curve (not necessarily parameterized by arclength) then its curvature at $\alpha(t)$ is given by the formula:

$$\frac{x'(t)y''(t) - y'(t)x''(t)}{\left(x'(t)^2 + y'(t)^2\right)^{\frac{3}{2}}}.$$

Consider the semicircle of radius r as the graph of $y = \sqrt{r^2 - x^2}$, i.e., parameterized by $x(t) = t$, $y(t) = \sqrt{r^2 - t^2}$ with $-r < t < r$. Use the above formula to show that its curvature is $\frac{1}{r}$.

▷ **Problem 2.** Show that if the torsion function τ of a space curve is identically zero then the curve lies in a plane.

▷ **Problem 3.** Compute the curvature and torsion of the Helix:

$$\alpha(t) := (r\cos(t), r\sin(t), bt)$$

.

▷ **Problem 4.** Show that the "triple vector product" $(u \times v) \times w$ is given by the formula

$$(u \times v) \times w = (u \cdot w)v - (v \cdot w)u.$$

Definition Let V be a real vector space. A real-valued function $B : V \times V \to \mathbf{R}$ is called a *bilinear form* on V if it is linear in each variable separately (i.e., when the other variable is held fixed). The bilinear form B is called *symmetric* (respectively *skew-symmetric*) if $B(v_1, v_2) = B(v_2, v_1)$ (respectively $B(v_1, v_2) = -B(v_2, v_1)$) for all $v_1, v_2 \in V$.

▷ **Problem 5.** Show that every bilinear form on a vector space can be decomposed uniquely into the sum of a symmetric and a skew-symmetric bilinear form.

Definition A real-valued function Q on a vector space V is called a *quadratic form* if it can be written in the form $Q(v) = B(v, v)$ for some symmetric bilinear form B on V. (We say that Q is *determined by* B.)

▷ **Problem 6.** (Polarization Again.) Show that if Q is a quadratic form on V then the bilinear form B on V such that $Q(v) = B(v, v)$ is uniquely determined by the identity $B(v_1, v_2) = \frac{1}{2}(Q(v_1 + v_2) - Q(v_1) - Q(v_2))$.

Remark. Suppose that V is an inner-product space. Then the inner product is of course a bilinear form on V and the quadratic form it determines is just $Q(v) = \|v\|^2$. More generally, if $A : V \to V$ is any linear operator on V, then $B^A(v_1, v_2) = \langle Av_1, v_2\rangle$ is a bilinear form on V and B^A is symmetric (respectively, skew-symmetric) if and only if A is self-adjoint (respectively, skew-adjoint).

▷ **Problem 7.** Show that any bilinear form on a finite dimensional inner-product space is of the form B^A for a unique choice of linear operator A on V. (Hint. Recall the isomorphism of V with its dual space V^* given by the inner-product.)

Final Exam Due Friday, December 5, 2003

▷ **Problem 1.** Recall that if $f : \mathcal{O} \to \mathbf{R}$ is a real-valued function we get a parametric surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$, called the graph of f, by $\mathcal{F}(t_1, t_2) := (t_1, t_2, f(t_1, t_2))$.

Show that the First Fundamental Form coefficients are:

$$g_{11} = 1 + f_{t_1}^2, \quad g_{12} = f_{t_1} f_{t_2}, \quad g_{22} = 1 + f_{t_2}^2,$$

and that the Second Fundamental Form coefficients are:

$$\ell_{11} = \frac{f_{t_1 t_1}}{\sqrt{1 + f_{t_1}^2 + f_{t_2}^2}}, \quad \ell_{12} = \frac{f_{t_1 t_2}}{\sqrt{1 + f_{t_1}^2 + f_{t_2}^2}}, \quad \ell_{22} = \frac{f_{t_2 t_2}}{\sqrt{1 + f_{t_1}^2 + f_{t_2}^2}}.$$

▷ **Problem 2.** Let $t \mapsto \alpha(t) = (x(s), z(s))$ be a curve parametrized by arclength lying in the x, z-plane and not meeting the z-axis—i.e., $x(s) > 0$ for all s in the domain (a, b) of α, and let $\mathcal{O} = (0, 2\pi) \times (a, b)$. The surface of revolution defined by α is the parametrized surface $\mathcal{F} : \mathcal{O} \to \mathbf{R}^3$, defined by $\mathcal{F}(t_1, t_2) := (x(t_2)\cos(t_1), x(t_2)\sin(t_1), z(t_2))$. Show that the First Fundamental Form coefficients are:

$$g_{11} = x(t_2)^2, \quad g_{12} = 0, \quad g_{22} = 1,$$

and that the Second Fundamental Form coefficients are:

$$\ell_{11} = -x(t_2)z'(t_2), \quad \ell_{12} = 0, \quad \ell_{22} = x''(t_2)z'(t_2) - x'(t_2)z''(t_2).$$

Show also that the principle curvatures are

$$x''(t_2)z'(t_2) - x'(t_2)z''(t_2) \text{ and } -\frac{z'(t_2)}{x(t_2)},$$

(so the Gaussian curvature is $K = -\frac{x''(t_2)z'(t_2)^2 - x'(t_2)z'(t_2)z''(t_2)}{x(t_2)} = -\frac{x''(t_2)}{x(t_2)}$).

▷ **Problem 3.** Let $q : \mathbf{R}^2 \to \mathbf{R}$ be a C^2 function and define symmetric 2×2 matrix -valued functions g and ℓ on $\mathbf{R}^2$ by:

$$g_{11} = \cos^2(q), \quad g_{12} = 0, \quad g_{22} = \sin^2(q),$$

and:

$$\ell_{11} = -\ell_{22} = \sin(q)\cos(q), \quad \ell_{12} = 0.$$

a) Derive explicit expressions in terms of q for the 3×3 matrices P^1, P^2, and also for their commutator $[P^1, P^2] := P^1P^2 - P^2P^1$.

b) Prove that the Gauss-Codazzi equation

$$(P^1)_{t_2} - (P^2)_{t_1} = [P^1, P^2]$$

is satisfied if and only if q satisfies the so-called sine-Gordon equation (SGE)

$$q_{t_1t_1} - q_{t_2t_2} = \sin q \cos q.$$

(Hint: Check the the Gauss-Codazzi equation entry by entry. This gives 9 equations, most of which are automatically satisfied just because g_{12} and ℓ_{12} vanish, and the rest reduce to SGE.

c) Now we know that if q satisfies the SGE, then, by the Fundamental Theorem of Surfaces, there exists a surface in $\mathbf{R}^3$, unique up to rigid motion with First and Second Fundamental Forms respectively:

$$\cos^2 q\ dt_1^2 + \sin^2 q\ dt_2^2, \quad \text{and} \sin q \cos q\ (dt_1^2 - dt_2^2).$$

Use the formula $K = \frac{\det(\ell_{ij})}{\det(g_{ij})}$ to prove that the Gaussian curvature of this surface has the constant value -1.

Note that if we find solutions q of the SGE, then g_{ij}, ℓ_{ij} given in Problem 3 satisfy the Gauss-Codazzi equation, so your program can draw the corresponding $K = -1$ surfaces. The next two Exercises give two family of examples of solutions of the SGE.

▷ **Problem 4.** Let a be a non-zero constant, and

$$q(t_1, t_2) = 2 \arctan C(t_1, t_2),$$

where

$$C(t_1, t_2) = \exp\left(\frac{1}{2}(a + a^{-1})t_1 + \frac{1}{2}(a - a^{-1})t_2\right).$$

(i) Prove that $\cos q = \frac{1-C^2}{1+C^2}$ and $\sin q = \frac{2C}{1+C^2}$. (Hint: by definition of q, $\tan \frac{q}{2} = C$).

(ii) Prove that q satisfies the SGE.

(iii) Use your program to prove that for the case when $a = 0.2, 0.4, 0.6, 0.8$ and 1

$$g_{11} = \cos^2 q = \left(\frac{1 - C^2}{1 + C^2}\right)^2, \quad g_{22} = \sin^2 q = \left(\frac{2C}{1 + C^2}\right)^2, \quad g_{12} = 0,$$

and

$$\ell_{11} = -\ell_{22} = \sin q \cos q = \frac{2C(1 - C^2)}{(1 + C^2)^2}, \quad \ell_{12} = 0$$

satisfy the Gauss-Codazzi equation, and draw the corresponding surfaces. (Note that all these surfaces are of curvature -1, and when $a = 1$ you should get the pseudosphere and for $a \neq 1$ you get a family of Dini surfaces),

▷ **Problem 5.** Let $0 < \alpha < 1$, and let

$$B(t_1, t_2) = \frac{\sqrt{1-\alpha^2}\sin(\alpha\ t_2)}{\alpha\cosh(\sqrt{1-\alpha^2}\ t_1)},$$

$$q(t_1, t_2) = \ 2\arctan B(t_1, t_2),$$

and

$$g_{11} = \cos^2 q = \left(\frac{1-B^2}{1+B^2}\right)^2, \quad g_{22} = \sin^2 q = \left(\frac{2B}{1+B^2}\right)^2, \quad g_{12} = 0,$$

$$\ell_{11} = -\ell_{22} = \sin q \cos q = \frac{2B(1-B^2)}{(1+B^2)^2}, \quad \ell_{12} = 0.$$

By direct calculation you can check that,

$$q_{t_1t_1} - q_{t_2t_2} = \frac{2B(1-B^2)}{(1+B^2)^2} = \sin q \cos q,$$

i.e., q satisfies the SGE. So g_{ij} and ℓ_{ij} should give a surface with $K = -1$ for each constant $0 < \alpha < 1$, and in this exercise, we ask you to use your program to check the Gauss-Codazzi equation numerically and then draw the corresponding surfaces: Use your program to check that the above g_{ij}, ℓ_{ij} satisfy the Gauss-Codazzi equation when the constant $\alpha = 0.4, 0.6$ and 0.8, and draw the corresponding surface. (Note that $B(t_1, t_2)$ is periodic in t_2, so to get beautiful picture you should draw your surface with domain $t_2 \in [0, \frac{2\pi}{\alpha}]$ and $t_1 \in [-2, 2]$. Also you can do some experiments to see what happen if you change α to other rational numbers between 0 and 1).

Appendix III
Matlab Notes

```
% Math 32a   Fall 2003  Richard Palais  Brandeis Univ.
%
%     First Steps---An Introduction to Matlab Programming.

% In the following I assume that you already know how
% to start up Matlab and that you know how to add
% directories (= folders) to the Matlab search path.
% The idea is to get a feeling for the Matlab syntax,
% and a feeling for how to create certain data
% structures and manipulate them. What I suggest is that
% you go through the topics one by one, first reading the
% notes, and then entering the commands in the Matlab
% Command Window and make sure you understand why you
% get the responses that you do.


% HELP and TUTORIALS

help  % gives topics and functions for which help is available
help <topic>   e.g., help plot
helpwin  % opens a help browser window
helpdesk  % A hypertext browser for Matlab documentation
lookfor <topic>   e.g. lookfor cos
% which <function-name>  % gives full path to function M-File.
demo  % stsrts a Matlab demo

% CALCULATOR

2 + 2
2 + 2;
ans*ans
```

```
a=2;
b=3;
c=a+b

% COMPLEX NUMBERS

a = 1 + 2i
b = 3 - 1i
c = a + b

% VECTORS AND THE COLON AND LINSPACE NOTATION

a= [1 1 2 3.5 pi]
b = 1:5
c = a + b
1:0.1:2 % Creates a row vector with entries 1  1.1  1.2 ... 1.9  2

% ANOTHER WAY IS linspace(FIRST,LAST, NUMBER)
linspace(1,2,10)   % Gives a row vector with 10 equi-spaces elements from
1 to 2.
% WHOOPS!
linspace(1,2,11)   % Gives a row vector with 11 equi-spaces elements from
1 to 2.

% MATRICES
% Explicit manual entering. [a b c...;d e f...; ...]
% OR  [a,b,c...;d,e,f...;...]
% each element entered should either be a constant or an initialized variable.
% entering a semi-colon starts a new row.
% elements in each row can be separated either by spaces or commas.
A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
size(A)  % returns (r,c) where r is # of rows of A and c = # of columns
size(A,1)  % returns r
size(A,2)  % returns c
```

```
  % BRANCHING
  % if <condition> <statement>, <statement>, ... end
  % or (better formatting)
  % if <condition>
  %    <statement>
  %    <statement>
  %     ...
  %  end

  % Notice that if you enter several Matlab commands on a single line,
  % then they should be separated by either commas or (to supress output)
  % semi-colons.

  if (j ~= 0)
    k = 1/j
   end

   % the following defines the signum of x, i.e., +1, 0, or -1 depending on
whether
   % x is positive, zero, or negative.
  if  (x > 0)
     sgnum = 1
   elseif (x == 0)  % NOTE!  == tests for equality. A single = is assignment.
     signum = 0
   else
     signum = -1
  end

  % ITERATION USING FOR LOOPS

  for i = 1:5 i*i, end
  %
  sum = 0;
```

```
for i = 1:5  sum = sum + i*i, end
%
sum = 0;
for i = 1:5  sum = sum + i*i; end
sum
%
t=0;
t=cos(t)

% The UP-ARROW KEY takes you back to the preceding command
% This would make it easy here to execute t = cos(t) many times.
% But even more convenient here is to use a "for loop"
t = 0;
for i = 1:5, t = cos(t), end
% Better still make this more general as follows:
t = 0;
N = 5;
for i = 1:N  t = cos(t); end,  t
% For more convenience still we could make this into a Script
% Create a file named IterateCosScript.m (in your Matlab path)
% with the following lines.
t = 0;
for i = 1:N
   t = cos(t);
 end
 t
% Then---
N = 8;
IterateCosScript
% The next step is to make a function M-File  of this!
% Create a file named IterateCos.m
function IterateCos(N)
t=0;
```

```
for i = 1:N
   t = cos(t);
end;
t
%
% Now call IterateCos(5), IterateCos(10),
% use format long to see more precision
format long
IterateCos(30)
IterateCos(60)
IterateCos(80)
IterateCos(100)

% Matlab keeps track of all the variables that you have
% defined in the current session and their current values.
% The command
%  whos
%  gives list of variables in the workspace.
whos
% clear    clears the current workspace
clear
whos
% MORE COLON NOTATION
horizontal=[1:6]
horizontal=[horizontal, 7, 8]
horizontal =  (1/8)*horizontal
horizontal = horizontal(2:7)
vertical = horizontal'

% THE PLOT COMMAND
% We will look carefully at the plot command later,
% but here is how to plot a list of (x,y) values and
% join successive ones by straight lines.
```

```
N=10;
x = (1/N)*2*pi*[0:N];  y = sin(x); plot(x,y);

N=20;
x = (1/N)*2*pi*[0:N];  y = sin(x); plot(x,y);

N=20;
x = (1/N)*2*pi*[0:N];
plot(sin(x),cos(x));
% ??? why don't we get a circle?
plot(sin(x),cos(x)); axis equal;
a=3; b= 2;

% Math 32a   Fall 2003  Richard Palais  Brandeis Univ.

%       Second Steps---More on Matlab Programming.
%
%              Topics for Today
%
% 1)  Subscripting or Indexing
% 2)  Vectorizing code
% 3)  Functions
% 4)  Relational Operators and Boolean Expressions
% 5) Flow Control
%   a)   if  else
%   b)   for loops
%   c)   while loops
%   d)   switch statement
% 6)  2D and 3D Graphing: the plot, plot3, mesh, and surf commands
% 7)   M-Files
%   a) Script M-Files
%   b) Function M-Files
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  To practice with Matlab on your own, choose
%  a topic and enter the lines, one by one, into
%  the Matlab Command Window, and make sure you
%  understand all the output you get back before
%  going on to the next topic.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Subscripting
%
a = [0.0: 0.1: 1]
a(1)
a(5)
a(end)
b = [0:5 ; 1:6; 2:7 ; 3:8 ; 4:9]
b(2:4, 4)
b(3, 2:end)
b(4,4) = 0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Stopwatch Functions
%
%  If you execute the command tic, Matlab starts
% a computation time stopwatch. The next time you
% execute the command toc, Matlab reports how much
% computing time in seconds it has used.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Vectorizing Code
%
%  A matlab command is said to be "vectorized" if it can act
```

```
%  not only on scalars, but also on vectors and matrices
%  (Often this just means it does to each element of an array
%  what it would do to a scalar). But as the next two examples
%  show that's not always the case.
%
% Calcuate Factorial 150 two ways, with a loop and using vectorized product.
tic, factorial = 1; for i = 2:150, factorial = i*factorial; end, toc
tic, factorial = prod(1:150); toc
%
% A similar pair of computations for summing an arithmetic progression.
tic, SUM = 0; for i = 1:10000, SUM = i + SUM; end, toc
tic, SUM = sum(1:10000); toc
% Notice that most built-in Matlab functions are vectorized.
X = [1:10]
sin(X)
X^2
% Why the error message?
% The problem here is that for a matrix,  Matlab uses ^2 to mean
% matrix squaring, which only makes sense for sauare matrices.
% For example:
X = [1,2;3,4]
% Vectorized squaring (i.e., squaring each element of an array
% is denoted by .^2.
X.^2
%  Similarly to multiply corresponding elements of two matrices
% with the same number of rows and columns, use .*
Y = [4,3;2,1]
X.*Y
% Similarly for vecorized division, use ./
X./Y
% X/Y (without the dot) means matrix product of X with Y inverse.
X/Y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%            Inline  Functions
%
%  Suppose we want to create the polynomial function
%  f(x) = 3*x^2 + 5*x + 2
%  Create a string S that defines the polynomial and
%  assign inline(S) to the name of the function:
f = inline('3*x^2 + 5*x + 2')
f(1)
f(2)
v = [0:5]
f(v)
%  We got an error because we didn't vectorize properly.
f = inline('3*x.^2 + 5*x + 2')
x = [-3: 0.1 : 1.5];
plot(x,f(x));
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Relational Operators and Boolean Expressions
%
%  The basic relational operators are
%  > ,  < , >= , <=  with the obvious meanings, and
%  == which means "equals" and ~= which means "not equal"
% If R is any one of these operators and x and y are
% constants or variables then xRy is a boolean
% expession, i.e., it eiher has the value 1 meaning
% xRy is true, or it has the value o meaning xRy is false.
% DON'T CONFUSE (A == B) with A = B !! The latter assigns the
% value of B to A, while (A == B) does not change the value
% of A and rather is an expression that has the value 1 if
% A is equal to B and otherwise has the value zero.
% You can build up more complex boolean expressions by
% combining basic one with & (which means AND) and | (which
% means OR) and ~ (which as we have already seen means NOT).
```

```
% First check that the relational operators are vectorized:
A = 0:8
B = 8-A
bool1 = A>4,
bool2 = (A==B)
bool3 = bool1 & bool2
bool3 = bool1 | bool2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Flow Control
%
%       if, else, and elseif
%
%  The general form of an if statement is:
%
%   if booleanexpression
%       command 1
%       command 2
%       ...
%   else
%       COMMAND1
%       COMMAND2
%        ...
%    end
%
%  If booleanexpression evaluates to 1(true) the first
%  group of commands is executed, otherwise the second
%  set of commands is executed. For example here is a
%  way to compute the maximum of a and b. To create a
%  multi-path branch with more conditions, replace all
%  but the final else by elseif.

max = 0;
```

```
a = 5;
b = 3;
if (a>b)
  max = a;
else
  max = b;
end
max

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       for loops
%
%  We have already seen several examples.
%  However ther is more than meets the eye.
%  The genral form is:
%
%  for x = Array
%    command 1
%    command 2
%    ...
%    end
%
%  First x is set equal to the first column of
%  Array and all the commands executed, then x is
%  set equal to the second column of Array and
%  the commands executed again and so on until
%  the last column of Array is reached.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       while loops
%
%  Here the general form is:
%
```

```
%   while booleanexpression
%    command 1
%    command 2
%    ...
%    end
%
%  First, booleanexpression is evaluated, and if
%  it evaluates to 1 (true) the commands are
%  executed, then booleanexpression is re-evaluated
%  and if it is true then the commands are evaluated
%  again, and so on until booleanexpression evaluates
%  to 0 (false) at which point contol shifts to
%  the first command after the end statement.
%  Of course something better happen durring execution
%  of the commands that will eventually make
%  booleanexpression false or we wre in an infinite loop.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        switch statement
%
%  The switch statement allows a program to switch
%  its execution path to any one of a number of possible
%  different branches depending on the value of an expression
%  (the switch-expression).
%  It has the general form:
%
% switch <switch-expression>
%   case <case-expression> statement,statement,statement, ...;
%   case <case-expression> statement,statement,statement, ...;
%   ...
%  otherwise statement,statement,statement, ...;
%  end
%
```

```
% First <case-expression> is evaluated. It should evaluate to
% either a scalar or a string. Control then jumps to the
% statements following the first case-expression that
% matches the value of switch-expression, or if there is no
% match to the statements following otherwise (or if there is
% no otherwise statement, to the statement following end.
% Example:
%
CurveType = 'Cubic';
switch CurveType
  case 'Linear'
      f = inline('a + b*x');
  case 'Parabola'
     f = inline('a + b*x + c*x^2');
  case 'Cubic'
     f = inline('a + b*x + c*x^2 + d* x^3');
  end
  f
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             Graphing
%   2D curves
%   A circle centerd at (a,b) with radius r is given parametrically by
%   x = a + r*cos(t)  y =b +  r * sin(t)  with t in [0, 2*pi]
a = 1;
b = 2;
r = 2;
t = [0: 0.1: 2*pi];
x = a + r* cos(t);
y = b + r* sin(t);
plot(x,y); axis equal
%%%%%%%%%%%%%%%
% 3D  Curves
% Helix  x = cos(t); y = sin(t);   z = 0.2*t;
```

```
t = [0: 0.1: 6*pi];
x = cos(t);
y = sin(t);
z = 0.2*t;
plot3(x,y,z);
rotate3d;      %This permits you to rotate the 3D object with the mouse.
%%%%%%%%%%%%%
%       Parametric Surfaces
%
%  In studying so-called polar spherical coordinates, you
%  probably learned that the point on the unit sphere
%  with longitude u and co-latitude v has the (x,y,z)
%  coordinates x = cos(u), y = sin(v), and z = cos(v),
%  Here is the way to render the sphere in Matlab.
%  First, this is the command to create a two-dimensional grid of
%  points (u,v) that will represent the longitude and co-latitude
%  of points of the sphere. Let's divide both intervals [0, 2*pi]
%  and [0,pi] into 50 sub-intervals:
[u,v] = meshgrid([0 : 2*pi/50 : 2*pi],[0 : pi/50 : pi]);
%  Then we create the arrays of values of the components of the
%  3D points (x,y,z) that have longitude u and co-latitude v:
x = cos(u).*sin(v);
y = sin(u).*sin(v);
z = cos(v);
%  Finally, the command mesh(x,y,z) takes the 3D grid that we have
%  created and maps it into 3-space using "wireframe rendering".
%  This just means that each 2D gridlines is mapped to the
%  3D segment joining the images of its endpoints.
mesh(x,y,z), axis equal
%  On the other hand, surf(x,y,z) renders the grid in "patch mode".
%  This means that each rectangle of the grid is filled in with a
%  color that is determined by an algorithm we will discuss later.
surf(x,y,z), axis equal
```

```
%
% Exercise: Render the ellipsoid with semi-major axes a,b, and c.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Script M-Files
%
%  The first kind of M-File, called a Script File, is easy to
%  explain. It is just a list of Matlab commands written
%  just as you would write them into the Matlab command window.
%  When you tyoe the name of the Script File (without the .m)
%  into the command window, the effect is exactly the same as if
%  you typed all the commands in the script File, one after another,
%  into the command window at the place. End of story.
%  (Well, almost. you do have to make sure that the script file is in
%  a directorythat is in matlab's search path in order for Matlab to
%  be able to find it.)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        Function M-Files
%
% The second kind of M-file, called a function M-File is a lot more
% sophisticated and more flexible, and it is the real key to what it
% takes to write powerful Matlab programs. They are much like subroutines
% in other programming languages that you may be familiar with. We will
% probably have a lot more to say about them in the weeks to come.
% For an example, let's use the above switch statement to create a
% function file called PlotCurve.m that will plot a line, parabola,
% or cubic, depending on a string parameter the user enters.

 function PlotCurve(CurveType)

%  This function plots either a line, a parabola or a cubic curve depending
%  on whether it is called with parameter 'Linear', 'Parabola', or 'Cubic'.
```

```
t = [-1.5: 0.05: 1.5];
switch CurveType
  case 'Linear'
      plot(t, 1 + 2*t); axis equal
  case 'Parabola'
      plot(t, 1 - 0.5*t + 2*t.^2); axis equal
  case 'Cubic'
      plot(t, 1 - 1.5*t + 2*t.^2 + 2.5* t.^3); axis equal
  end
```

```
% Math 32a   Fall 2003  Richard Palais  Brandeis Univ.

%       Discussion of Project 1

% Here is how I would go about developing the solution
% to Project 1. I would not start with an M-File, but
% first try to develop commands to implement the various
% steps of the algorithm in a text file such as this,
% and test them out in the mMatlab Command Window, and
% then when the pieces of my code seemed to be working
% properly, I would put them in a function M-file and
% add error checking. (It might even pay to have an
% intermediate step consisting of a script M-File, but
% for a simple project such as Project 1, that is not
% necessary.

% Recall, the goal is to apply Gram-Schmidt algorithm to
% the rows of a matrix M.
% Here again is a statement of the algorithm:
%  1) First normalize the initial row.
%  2) Then, in order, project each row (the current_row)
%     onto the (already orthonormalized) earlier rows.
%  3) Subtract this projection from the current row to
%     get the projection normal to earlier rows, and then
%  4) normalize the result.
%  OK, here goes:

% To normalize the initial row of M, namely  M[1,:].
% Compute its norm
norm1 = sqrt(M(1,:) * M(1,:)');
% We could also have written this as
% norm1 = sqrt( dot(M(1,:) , M(1,:)));
```

```
% or as:
% norm1 = norm(M(1,:));
% using Matlabs built-in functions dot and norm.

% Ignore possibility that norm1 might be zero for now
% and divide the first row by its norm
 M(1,:) = M(1,:)/norm1;
 % This completes step 1) above.

% Now start the loop that will orthonormalize later rows.
% Recall that the number of rows of M is size(M,1)
% so we want to iterate over the list of rows from
% row number 2 to row number size(M,1). In matlab-speak this is:
for row_num = 2:size(M,1)
    current_row = M(row_num,:);
% project current_row of M onto each earlier row and add the projections
% to get the projection of current_row on span of earlier rows
   proj = 0;
   for j = 1:row_num - 1
    proj = proj + dot(current_row,M(j,:)) * M(j,:);
    end
% Now subtract proj from the current row to get the projection of the
% current row orthogonal to earlier rows.
    u = current_row - proj;
norm1 = sqrt(u*u');
M(row_num,:) = u/norm1;
 end;
 %
 %
 %  Set up a test matrix.
 tst = [1 2 3 4;2 3 4 1;3 4 1 2; 4 1 2 3]
 % set M equal to tst
 M = tst
```

```
 % Then copy the above code and test it on M
 %
 %
 % next we add the boilerplate necessary to make our early
 % code into an M-File, and also add error checking.
 %
 % First, here is the Help Comment:

%  The function GramSchmidt takes as its single input
%  argument and m by n matrix M of real numbers and returns
%  an error message if the rows of M are linearly dependent,
%  but if they are independent then it returns a matrix
%  with orthonormal rows having the same size as M  and such
%  that the first k rows of Q have the same span as th first
%  k rows of M.

%  Programmer: Richard S. Palais

 function    X = GramSchmidt(M)

norm1 = sqrt(M(1,:) * M(1,:)');  % norm of the first row;
if (norm1 <0.000000001)
    Q = 'First row is zero. Cannot continue.';
disp(Q);
return
else
    M(1,:) = M(1,:)/norm1;
end  % if

% Start loop to orthonormalize later rows.
for row_num = 2:size(M,1)
    current_row = M(row_num,:);
% project current_row of M onto each earlier row and add the projections
```

```
  % to get the projection of current_row on span of earlier rows
      proj = 0;
      for j = 1:row_num - 1
          proj = proj + dot(current_row,M(j,:)) * M(j,:);
      end % for j = 1:row_num - 1
  % Now subtract proj from the current row to get the projection of the
  % current row orthogonal to earlier rows.
      u = current_row - proj;
  norm1 = sqrt(u*u');
  if (norm1 <0.000000001)
         Q = 'Rows are linearly dependent, cannot continue.';
     disp(Q);
     return
      else
     M(row_num,:) = u/norm1;
  end  %if (norm1 <0.000000001)
   end;  % for row_num = 2:size(M,1)

    X = M;


   %  END OF GramSchmidt

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Here is a somewhat selection of a few more solutions developed by
   % various class members.

   % First, David Diamondstone's version

   % Call as N = GramSchmidt(M).
  % M should be a rectangular
  % matrix with linearly independent rows,
```

```
% and then N will be a matrix of the same size with orthonormal rows
% and the linear spans of the first k rows of M and N are the same.

% Programmer: David Diamondstone

function N=GramSchmidt(M)

r=size(M,1); c=size(M,2);
for i=2:r,
    for j=1:i-1,
         M(i:r:r*c)=M(i:r:r*c)-(M(i:r:r*c)*M(j:r:r*c)')/(M(j:r:r*c)*M(j:r:r*c)')*M(j
 if M(i:r:r*c)*2==M(i:r:r*c), 'The input matrix was not linearly independant.',
    Failure_Row=i,
         end;
end;
end;
for i=1:r,
   M(i:r:r*c)=M(i:r:r*c)/sqrt(M(i:r:r*c)*M(i:r:r*c)');
end;
 N=M;

 % Comments on David's M-File.
 % Note that David first orthogonalizes
 % all the rows, and only  then normalizes them.
 % This has a number of advantages! Can you see them?
 % Pros: It is VERY short and succinct.
 % Cons: It is a little hard to understand.
 % The error handling is less than prfect, and
 % in particular, it fails to handle
 % the case that the firt row is zero.

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
   % Here is Lacra Bintu's M-File:


   % GramSchmidt takes a matrix with linearly independent rows and returns another
matrix
   % whose rows are orthonormal to each other.
   %If the rows of the input matrix are not linearly independent, the GramSchmidt
will give
   % an error message.


   % Programmer: Lacramioara Bintu


  function F=GramSchmidt(M)
  n=size(M,1); m=size(M,2); %matrix dimensions
  if M(1,:)*M(1,:)'==0
  F='The rows of your matrix are not linearly independent! Orthonormalization
is impossible!';
  else
  F(1,:)=M(1,:)/((M(1,:)*M(1,:)')^(1/2)); %normalize the first row
  for i=2:n
  vi=0;
  for k=1:i-1
  vi=vi+(M(i,:)*F(k,:)')*F(k,:); %project the k'th row onto the space of the
first k-1 orthonormalized vectors
  end
  ui=M(i,:)-vi; %subtract the projection from the original i'th row vector
  u=(ui*ui')^(1/2); %find the norm of the difference
  if (u<10^(-10)*mean(mean(abs(M))))
  % if the norm is zero give error message
  F='The rows of your matrix are not linearly independent! Orthonormalization
is impossible!';
  break %stop going through the loop
  else
  F(i,:)=ui/u; %normalize the difference obtained above
  end
  end
```

```
end

 % Comments on Lacra's M-File
 % Pros: Quite short, but well-commented and so reasonably easy to understand.
 % Appears to work correctly for matrices with linearly independent rows,
 % and error handling is good.
 % The scaling in the line  if (u<10^(-10)*mean(mean(abs(M))))
 % is a very nice touch, since it correctly makes the test depend
 % on the relative round_off error rather than absolute round-off error.

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   %Chandni Rajan Valiathan's  M-File

 %This function takes as input, a rectangular matrix M of size m x n whose
%rows are linearly independant. It transforms the matrix to one whose rows
%are orthonormal and whose first K rows span the same subspace as that
%spanned by the first K rows of M. If it gets, as input, a matrix that has
%linearly dependant rows, it gives an error and no matrix. You can call the
%function as follows >> GramSchmidt (A) (where A is the matrix whose rows
%you want to be orthonormalized.)

%Programmer: Chandni Rajan
             %Valiathan
%Math 32a Fall 2003
%Project 1

function F= GramSchmidt(M)

m= size (M, 1); %no. of rows
v= M(1,:);
normv= sqrt(v*v');
if (normv == 0) %if the first row is the zero vector, then the matrix is
```

```
                %linearly dependant so display an error and return to the
prompt
    F= 'Input matrix has linearly dependant rows, so impossible to orthonormalize';
        return;
  else
     F(1,:)= (1/normv)*v; %otherwise normalize it and replace the first row
of M with
                          %the new orthonormalized vector.
  end;
  for i=2:m,
     w=0; %reset w which stores the orthogonal projection
     vi= M(i, :); %vi= ith row of vectors
     for j=1:(i-1),
         w= w + (vi*(F(j, :))')*F(j, :);
     end; %orthogonal projection of v onto the subspace W of V
     u= vi - w;
     normu= sqrt(u*u'); %n=norm of u
     if (normu < 1e-5)
         F= 'Input matrix has linearly dependant rows, so impossible to orthonormaliz
         return;
         %vectors not linearly dependant so error displayed
     else
         F(i,:)= (1/normu)* u;
     end;
  end;


   % Comments: Clear and well-documented. Works correctly and has good error
handling.


      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Anny Jones M-File is very carefully commentes and
```

```
   % works correctly with both "good" and "bad" matrices


   % GramSchmidt
  % This program takes as input a rectangular matrix (M) of arbitrary size m
x n,
  % and assuming that m rows are linearly independent, transforms (M) into another
  % m x n matrix (Q) whose rows are orthonormal, vis-a-vis the Gram-Schmidt
  % Algorithm. The subspace spanned by the first k rows of (Q) is equal
  % to the subspace spanned by the first k rows of (M).

  % Programmer: M. Anny Jones

  % M = input('matrix M = ');
  function GramSchmidt(M)
  % First I will determine the size of input matrix (M):
  m = size(M,1); n = size(M,2);

  % I will duplicate matrix (M) and overlay each row as it is calculated
  Q = M;

  % Next, I will define Q1, the first element of output matrix (Q):
  w = M( 1, 1:n );
  % Since we know that the norm of vector w is the square root of the inner
product
  % <w,w>, which can be written w*w' when w is a row vector:
  normw = (w*w')^0.5;
  if normw == 0
      disp('The first vector of (M) is zero. Please correct.')
      return
  end
  Q( 1, 1:n ) = w./normw;
  % Here I have overlaid my first vector (Q1, if you will) over (M)
  % Once m rows have been replaced, matrix Q will be the solution.
```

```
  % Now, using Q1, we can iterate the process to find the final elements Qi:
  for i = 2:m,
      w = M( i, 1:n );
      s = 0;
      for j = 1:i-1,
          v = (w*Q( j, 1:n)')*Q( j, 1:n);
          s = s + v;
      end
      % Here, s is the projection of Mi onto Wi-1, ie. the projection of the
      % ith row of M onto the subspace spanned by the first (i-1) rows of M
(or Q).
      % I have used a second 'for' to represent the summation over j=1 to j=i-1
      u = w - s;
      if (u*u' < 0.000001)
          disp('Program cannot be completed! Please check that the')
          disp('rows of the entered matrix are linearly independent.')
          return
      else
          normu = (u*u')^0.5;
          Q( i, 1:n ) = u./normu;
      end
  end
  % Now I have found each subsequent row of Q and the result is displayed.
  Q


  % Math 32a   Fall 2003  Richard Palais  Brandeis Univ.

  %               Discussion of Project 2

  % Let's first write the code for the trapezoidal approximation
  % to the integral of a function f over [a,b] without subdivision.

   trapezoid = 0.5*(b-a)*(feval(f,a) + feval(f,b));
```

```
% We recall that a bound for the error is given by

 ErrT = C1*(b-a)^3/12;

% where C1 is the maximum of the absolute value of f'' on [a,b] .

% Similarly, the Simpson's Rule approximation to the integral
% of f over the same interval, without subdividing, is

Simpson = ((b-a)/6)*(feval(f,a) + 4 * feval(f,(a+b)/2) + feval(f,b));

% and in this case, a bound for the error is given by

 ErrS = C2*(b-a)^5/90;

% where C1 is the maximum of the absolute value of f'''' on [a,b] .

% Lets try this out on a simple function, say sin:

 f = 'sin';

% on the interval [0,pi/2]

 a = 0;
 b = pi/2;

 % Since -cos is an antiderivative for sin, the exact value
 % of the integral is

  integral = (-cos(pi/2) - ( - cos(0)))

% Also, clearly C1 and C2 = 1;
```

```
C1 = 1;
C2 = 1;

%  Now lets design the M-Files.

%     First       TrapezoidalRule1.m
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 function    ApproxInt = TrapezoidalRule1(f,a,b,n)

%  This computes the approximation to the integral
%  of the function f from a to b by dividing the
%  interval into n equal parts and applying the
%  Trapezoidal Rule approximation to each part and
%  then summing.

 %  Richard Palais

 h = (b-a)/n;
 ApproxInt = 0;
 for k = [0:n-1]
   a1 = a + k*h;
   b1 = a1 + h;
   ApproxInt = ApproxInt + trapezoid(f,a1,b1);
 end;


        %%%%%%% Auxiliary Function  %%%%%%%

 function      term = trapezoid(f,a,b)

     term  = 0.5*(b-a)*(feval(f,a) + feval(f,b));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Next    SimpsonsRule1.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  function    ApproxInt = SimpsonsRule1(f,a,b,n)

%  This computes the approximation to the integral
%  of the function f from a to b by dividing the
%  interval into n equal parts and applying the
%  Simpson's Rule approximation to each part and
%  then summing.

 %  Richard Palais

 h = (b-a)/n;
 ApproxInt = 0;
 for k = [0:n-1]
   a1 = a + k*h;
   b1 = a1 + h;
   term = Simpson(f,a1,b1);
   ApproxInt = ApproxInt + term;
 end;

      %%%%%%% Auxiliary Function  %%%%%%%

 function      term = Simpson(f,a,b)

   term = ((b-a)/6)*(feval(f,a) + 4 * feval(f,(a+b)/2) + feval(f,b));

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The second versions below avoid he auxiliary
% function calls, AND more importantly, they
```

```
% avoid re-evaluating the integrand twice for
% each interior endpoint of an interval
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% TrapezoidalRule2.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function    ApproxInt = TrapezoidalRule2(f,a,b,n)

 %  This computes the approximation to the integral
 %  of the function f from a to b by dividing the
 %  interval into n equal parts and applying the
 %  Trapezoidal Rule approximation to each part and
 %  then summing.

 %  Richard Palais

 h = (b-a)/n;
 sum = (feval(f,a) + feval(f,b))/2 ;
 x = a;
 for k = [1:n-1]
     x = x + h;
     sum = sum + feval(f,x);
 end;
 ApproxInt = h * sum;

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%    SimpsonsRule2.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function    ApproxInt = SimpsonsRule2(f,a,b,n)
```

```
  %  This computes the approximation to the integral
  %  of the function f from a to b by dividing the
  %  interval into n equal parts and applying the
  %  Simpson's Rule approximation to each part and
  %  then summing. It gains efficiency by grouping
  %  terms so that the function is not evaluated
  %  more than once at the same argument.

   %  Richard Palais

  h = (b-a)/n;
  x = a;
  z = a + h/2;
  sum = (feval(f,a) + feval(f,b)) + 4 * feval(f,z);
     for k = [1:n-1]
         x = x + h;
         z = z + h;
         sum = sum + 2 * feval(f,x) + 4 * feval(f,z);
     end;
  ApproxInt = (h/6) * sum;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % The third versions below avoid the loop by
  % vectorizing the sum, and they also avoid
  % evaluating the integrand more than once for
  % a given value of the argument.  The code is by
  %   Nick Dufresne. (slightly modified).
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % TrapezoidalRule3.m
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%nick dufresne
%
%usage:
%      result = TrapezoidalRule3(f,a,b,n)
%
% f is an inline function that maps a real input to a vector
% a is the starting point
% b is the ending point
% n is the number of sub-intervals into which [a,b] is divided
% result is the estimated value of the integral
%

function thesum = TrapezoidalRule (f,a,b,n)
delta = (b-a)/n;
arguments = [a:delta:b];

%calculate all the values of the function
%in summing all n parts of the trapezoidal method we count internal points
%twice so we will multiply by 2

values = 2*feval(f,arguments);

%since we have calculated the value of the endpoints twice we need to
%subtract the values and we need to multiply by 0.5*delta  = 1/2*(b-a)/n

thesum = 0.5*delta*(sum(values)-feval(f,a)-feval(f,b));

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%    SimpsonsRule3.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%nick dufresne
%
```

```
%usage:
%      result = SimpsonsRule3(f,a,b,n)
%
% f is an inline function that maps a real input to a vector
% a is the starting point
% b is the ending point
% n is the number of sub-intervals into which [a,b] is divided
% result is the estimated value of the integral
%

function thesum = SimpsonsRule (f,a,b,n)
delta = (b-a)/n;
arguments = [a:delta:b];
midpointArgs = [a+delta/2:delta:b-delta/2];
%calculate all the values of the function.
%in summing all n parts of the trapezoidal method,
%we need to count internal pointstwice so we multiply by 2.

values = 2*feval(f,arguments);

%in the formula all midpoint values are multiplied by 4

midpointValues = 4*feval(f,midpointArgs);

%since we have calculated the value of the endpoints twice we need to
%subtract the values and we need to multiply by (1/6)*delta  = 1/6*(b-a)/n

thesum = (1/6)*delta*(sum(midpointValues)+sum(values)-feval(f,a)-feval(f,b));
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 format long

  f = inline('4./(1+ x.^2)','x');  % must be vectorized for third versions!
```

```
    a = 0;

    b = 1;

    C1 = 12;

    C2 = 96;

  %%%%%%%

    n=10000;

  tic, A = TrapezoidalRule1(f,a,b,n); toc  %elapsed_time = 9.210232

  tic, A = TrapezoidalRule2(f,0,1,n); toc  %elapsed_time = 4.533945

  tic, A = TrapezoidalRule3(f,0,1,n); toc  %elapsed_time = 0.009738  !!!!!

   A

   ErrT = C1*(b-a)^2/(12 * n^2)

   Error  = abs(A-pi)


  tic, A = SimpsonsRule1(f,a,b,n); toc

  tic, A = SimpsonsRule2(f,a,b,n); toc

  tic, A = SimpsonsRule3(f,a,b,n); toc

    A
```

```
 ErrT = C1*(b-a)^2/(12 * n^2)

 Error  = abs(A-pi)

%%%%%

n = 3;

tic, A = SimpsonsRule1(f,a,b,n); toc   %elapsed_time = 0.005829

tic, A = SimpsonsRule2(f,a,b,n); toc   %elapsed_time = 0.004387

tic, A = SimpsonsRule3(f,a,b,n); toc   %elapsed_time = 0.002445n


 A

 ErrS = C2*(b-a)^4/(90 * n^4)

 Error = abs(A-pi)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%   Let's check how the errors vary as a function
%%%   of the  number n of subdivisions.
%%%   First Trapezoidal:
%
n = 5;
A = TrapezoidalRule1(f,a,b,n);
Error5 = abs(A-pi)          %   0.00666653977880
n=10;
A = TrapezoidalRule1(f,a,b,n);
```

```
Error10 = abs(A-pi)          %   0.00166666468263
%
%  Since Trapezoidal is a quadratic method, the ratio
%  of Error5 to Error 10 should be about (10/5)^2 = 4
%
 ratio = Error5/Error10     %  3.99992862887449
%
%%%%   Now Simpson's
%
n = 5;
A = SimpsonsRule1(f,a,b,n);
Error5 = abs(A-pi)          % 3.965057793209326e-08
n=10;
A = SimpsonsRule1(f,a,b,n);
Error10 = abs(A-pi)         %  6.200080449048073e-10
%
%  Since SimpsonsRule1 is a fourth order method, the ratio
%  of Error5 to Error 10 should be about (10/5)^4 = 16
%  BUT,IN FACT:
 ratio = Error5/Error10      %  63.95171523650308  ~ (10/5)^6
%
n = 20;
A = SimpsonsRule1(f,a,b,n);
Error20 = abs(A-pi)          %   9.687362023669266e-12
%
 ratio = Error10/Error20     %  64.00174200055011 ~  (20/10)^6

%  So, the conclusion seems nearly inescapable that for
%  the present function, Simpson's is behaving as a
%  sixth order method !!!!

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Shootout:   Comparison of Nick Dufresne's vectorized Trapezoidal and
```

```
%              Simpson's with non-vectorized versions.

  f = inline('4./(1+ x.^2)','x');   % must be vectorized for third versions!

  a = 0;

  b = 1;

n=100000;

tic, A = TrapezoidalRule3(f,a,b,n); toc    %   elapsed_time = 0.07617

Error  = abs(A-pi)                         %  1.664046678229170e-11

n=14;

tic, A = SimpsonsRule3(f,a,b,n); toc       %  elapsed_time = 0.002523

Error  = abs(A-pi)                         %  8.234657400407741e-11

n = 100000;

tic, A = TrapezoidalRule2(f,a,b,n); toc    %  elapsed_time = 49.330224

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Math 32a   Fall 2003  Richard Palais  Brandeis Univ.

%       Topic for Today:   Programming with M-Files.
%
% 1)  Review of Script M-Files
% 2)  Review of Function M-Files
% 3)  More Advanced Features of Function M-Files
```

```
  %       Variable number of input and output parameters (nargin and nargout)
  %       Global Variables---Workspaces and their Relation to each other
  %       Sub-functions
  %       Functions as parameters (using feval)
  %       Introduction to De-bugging
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %- Script M-Files (review)
  % -They are just a shorthand way to enter commands
  % - should be a list of commands (and comments)
  % - Variables introduced in a script file continue
  %    to live in the Matlab basic Workspace after the
  %    script finishes executing
  %
  %    - Example of a script M-file (IterateCosScript.m )
  %
   t=0;
  for i = 1:N
     t = cos(t);
   end
   t

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % - Function M-Files (review)
  % - must begin with the keyword function,
  % - followed by the list of output variables and an = (if there are any output
variables)
  % - followed by the name of the function (which must be the same as the name
  %      of the M-file without the .m),
  % - followed by list of input arguments (in parentheses),
  % -  followed by the list of commands to be executed.
  % - So here is the general schema for a function M-File with the name FcnName.m,
  %    output variable x, y,..., and input variables a,b,...
  %
```

```
  %  function     [x,y,...] = FcnName(a,b,...)
  %  <command>
  %  <command>
  %  ...
  %
  % EXAMPLE: filename   PolarToRect.m

  function     [x,y] = PolarToRect(r,theta)
  % This function takes two real input parameters, r and theta, that
  % represent the polar coordinates of a point on the plane, and
  % returns two real output parameters, x and y, representing the
  % cartesian coordinates of the same point.
    x = r*cos(theta);
    y = r*sin(theta);
  %
  %
  % - Variables introduced in function M-files are "local" to the function---that
is
  %   they are not part of the basic Workspace, but rather they belong to a
special
  %   Workspace belonging to the M-File, and they disappear after the script
finishes
  %   executing. Workspaces, and their inter-relations are discussed in more
detail
  %   below.
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  %- More Advanced Features of Function M-Files
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %
  %  Variable number of input and output parameters (nargin and nargout)
  %
  %- The variable nargin:   It is possible to define a Matlab functions that
  %  take a variable number of input arguments. When the user calls the function
```

```
  %  with a particular mumber of arguments, say n, then a standard Matlab variable
  %  called nargin (for "number of arguments in" gets the value n which then
can
  %  be used in the program code to do different things depending on n. Here
is the
  %  way it works in a rather artificial example. Suppose we want to be able
to add
  %  a variable number of arguments, say up to four. We create a file MyAdd.m
containing
  %  the following:
  %
  function    theSum = MyAdd(x,y,z,w)
   if nargin == 0
     theSum = 0;
  elseif  nargin == 1
     theSum = x ;
  elseif  nargin == 2;
     theSum = x + y
  elseif  nargin == 3 ;
     theSum = x + y + z ;
  elseif  nargin == 4;
     theSum = x + y + z + w ;
  end
   theSum


  % - There is a similar variable nargout that gives the number of output variables
actually
  %  used in a given call of a function with several output parameters.
  %
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % -  Global Variables---Workspaces and their Relation to each other
  %
  %  A Matlab program that does something reasonably complex will often use
  %  many different functions that are either built in Matlab functions or
  %  else user defined functions defined in some M-File. It is important to
```

```
  %  be able to communicate information (e.g., the current values of variables)
  %  between these functions. Now remember that each M-File has its own private
  %  Workspace which is different from the base Workspaceof the command window.
  %  Suppose you declare a variable named X in the command window and also declare
  %  variables named X in two different function M-Files. If you now call these
  %  functions in the command window, or if you call one of these functions
from
  %  the other, then the different variables named X are unrelated, and changing
  %  the value of one of them does not effect the others. So how can you communicate
  %  information back and forth between M-Files and the command window. You
can
  %  probably guess the answer---the best way to do this is to use the input
and
  %  output variables of the functions as we have been doing. However, this
is
  %  sometime inconvenient and Matlab does have anothr mechanism (which should
  %  be used sparingly) and this is the so-called global Workspace. You can
declare
  %  a variable, say Z to be global as follows:
    global Z
  %  and then it belongs to the global Workspace. If you declare a variable
to
  %  be global in the command window and again in several M-Files, then they
  %  are all the same variable, and if you change the value in any of these
places
  %  it will change it in the others.
  %  NOTE: It is conventional to use all capital letters for global variables!
  %
  %  Example: Create an M-file called TestGlobal.m containing the following:


  function     outVar = TestGlobal(inVar)
  %  This function demonstrates passing values with parameters and
  %  using global variables
   outVar = inVar;
   inVar = 2;
```

```
X = 0;
global Y
Y = 0;

%  Now in the command window, suppose we enter the following:
%
a = 1;
b = 0;
X = 1;
global Y
Y = 1;
a = TestGlobal(b)
b
X
Y
%
% Can you guess what the output will be?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% -  Sub-functions
%
%  It is a good idea in programming to break up a complicated
%  algorithm into a sequence of smaller sub-algorithms. In Matlab,
%  this means that we will often create a function M-File by calling a
%  lot of other functions, either built in or defined in M-Files.
%  However, if you are ONLY going to use some simple function as a
%  sub-routine in defining some other "main" function, then Matlab
%  permits you to define the simple function in the same M-File
%  as the main function. These so-called sub-functions are defined
%  just as if they were a normal function, except they are placed at
%  the end of the M-File in which they will be used (and cannot be used
%  outside that file).
%
```

```
% Example  Create an M-file called TestSubFunction.m containing the following:

function      y = TestSubFunction(x)

    y = Double(Square(x));   % so y = 2 * x * x

%%%%%%%%%%%%%  Sub Functions  %%%%%%%%%%%%%%%%%%%
function   y = Double(x)
   y = 2*x;

function    y = Square(x)
   y = x * x;

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% -  Functions as Parameters   (using feval)

% Very frequently we want to create functions whose arguments are
% other functions---integration is a typical example, as are its
% approximating quadrature rules. Usually we will want to substitute
% for such a function argument a function that is either defined in an
% M-File or is a "built-in" function like sin.  Matlab has a function
% called feval to make this possible. If MyFcn is the name of a function
% of n variables, then  feval('MyFcn',a1,a2,...,an) is the same as
% MyFcn(a1,a2,...,an).   Here is an example of how this gets used.
% Suppose we want to define a Matlab function called Trapezoidal with
% parameters a function f, and two real numbers a, and b, that we want
% to return the trapezoidal estimate of the integral of f from a to b,
% namely 0.5*(f(a) + f(b)). In Matlab version 5 and earlier it is NOT
% possible to have a function as the parameter of a Matlab function
% BUT it is possible to have a string as a parameter, so we define
% Trapezoidal in a Matlab M-file called Trapezoidal.m as follows:
%
```

```
 function        A = Trapezoidal(f,a,b)
     A = 0.5*(b-a)*(feval(f,a) + feval(f,b));
%
%  Notice that this assumes that the parameter f is a string that will
%  hold the NAME of some function.
%   Now suppose that in an M-File  afcn.m we define a function afcn
%   as follows:
%
function   z = afcn(x)
  z = x^2 + 2*x + 4;
%  Then to estimate the integral of afcn from 0 to 1  and put the result
%  in a variable z, we would use the function call:
%
   z = Trapezoidal('afcn',0,1);
%
%   Note carefully the quotes around afcn.
%   The same thing works for a built-in function like sin
%
   z = Trapezoidal('sin',0,1);
   %
%   We can also use Trapezoidal on a function F defined using inline.
%   For example if we define the same unction as above using inline
%   and call it afcni:
%
   afcni = inline('x^2 + 2*x + 4', 'x')
%
%  then the Trapezoidal approximation to the integral is given by
%
   z = Trapezoidal(afcni,0,1);
%
%  THAT'S RIGHT---without the quotes! That is because the inline
%  function actually returns a string. In fact we can do the above
%  evaluation more simply in one step, without defining MyFcnI as follows:
```

```
%
   z = Trapezoidal(inline('x^2 + 2*x + 4', 'x'),0,1);
%
%  Of course this is NOT very elegant---it is both inconsistent and
%  non-intuitive and so leads to lots of programming errors. In the
%  most recent version of Matlab (version 6) this problem has been
%  fixed. It now IS possible to have as a parameter of a Matlab
%  function another Matlab function (rather than the string that is
%  its name, so in Matlab version 6 we could define the Trapezoidal
%  function by
%
 function        A = Trapezoidal(f,a,b)
     A = 0.5*(b-a)*(f(a) + f(b));
%
%  avoiding the use of feval. Then we would call it consistently
%  as
%
  z = Trapezoidal(afcn,0,1);

%  or
%
   z = Trapezoidal(sin,0,1);
%
%  or
%
  z = Trapezoidal(inline('x^2 + 2*x + 4', 'x'),0,1);
%
%  However for this course, I would pefer for you to
%  stick with the Matlab version 5 conventions.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% - Introduction to De-bugging
%
```

```
%  No matter how experience you become or how careful
%  you are, you will always make errors, i.e., bugs
%  are a fact of programming life, and you will have
%  to learn to live with that fact. That does not mean
%  that you have to live with the bugs---it means that
%  you must learn how to test for them and how to
%  eliminate them when you find them. This is a very
%  big topic and we can only scratch the surface here.
%
%  There is one basic, time-tested strategy for debugging,
%  which is to stop the program at various points (so-called)
%  breakpoints) and then test the values of variables to
%  see if they have the values you expect them to. In Matlab
%  if you put the word "keyboard" into an M-File, then any
%  time the programm reaches that point, the program stops
%  running and you can type various commands to check the
%  values of variables (or change them). To restart the execution
%  of the M-File, type the six letters "RETURN" and then
%  the return key
%
```

```
% A Short Introduction to Movie Making in Matlab
% Lacra Bintu


% Matlab stores movies as information in matrices.
% The information for each frame is stored as to one column in the movie matrix.
% Somewhere near the beginning of your file you must initialize your movie
matrix.
% Suppose you want to name your movie MyFilm, and you want it to have N frames.

N=100;                                          % say how many frames you want
MyFilm=moviein(N);                              % initialize the movie matrix;
for i=1:N
    x=cos(2*pi*i/N);    y=sin(2*pi*i/N);    % The simplest example I could
think of:
                                            % a point moving in a circle
figure(1)
plot(x,y, 'o')
    axis([-1.5 1.5 -1.5 1.5])               % fixing the axis, so that they
will
                                            % be the same for all frames
                % axis([XMIN XMAX YMIN YMAX])
MyFilm(:,i)=getframe;                           % takes the information from the
current picture
                                            % and puts it in the ith column
of the movie matrix
    % (this will be the ith frame of MyFilm)
end                                             % It's a wrap!  That's all there
is to it.

movie(MyFilm)                                   % Plays Myfilm
                                        % Say you want to save the movie
                                    % in a file called MyBest (you don't need
to):
```

```
  Save  MyFilm  MyBest
                                                  % To play the saved movie:
  load  MyBest                                    % This loads the information stored
in the movie file
  Movie(MyFilm)                                   % This plays the movie that had
been stored in
                                                  % MyBest (meaning the movie MyFilm)

  % So to recapitulate:

  % <Moviename> = moviein(<number of frames>);    % moviein initializes the
movie matrix
  % <Moviename>(:,i)                              % saves contents of current
figure in
                                                  % i-th column of the movie
matrix.
  % Save <Moviename> <Filename>                   % Saves movie matrix in a
data file
  % load <Filename>                               % Reads movie data stored
in <Filename>
  % Movie <Moviename>                             % Plays <Moviename> as an
animation

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % Sometimes, when you are drawing more than one object
  % in each frame you might want to use hold on and hold off.
  % If you put hold on after plotting the first object,
  % when you plot another object (in the same frame)
  % the first object does not disappear from view.
  % If you put hold off after the second plot, the next
  % plot after that will cover everything already in your figure.
  % Below is a quick example, using the same code as before,
  % but now making use of hold on and hold off.
  % (If you didn't understand what I just said above about hold
```

```
  % on and hold off, just try experimenting
  % with the code below, by deleting first
  % hold off and then hold on and see what happens.
  % That should make it it clear.)

  N=100; MyFilm=moviein(N);
  for i=1:N
    x=cos(2*pi*i/N);  y=sin(2*pi*i/N);
    figure(1)
    plot(x,y, 'o')
    hold on
    x1=[0 x];  y1=[0 y];
    plot(x1,y1)            %draws the radius of the circle
    hold off
    axis([-1.5 1.5 -1.5 1.5])
    MyFilm(:,i)=getframe;
  end
  movie(MyFilm)

  % If you use an earlier version of Matlab,
  % you can turn your movie into an avi file
  % using movie2avi.

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  % Here is a more sophisticated and interesting example.
  % It is the contents of a script M-File called ellipse.m



  % This file creates and plays a movie showing the osculating circles to an
ellipse
  % Lacra Bintu


  clear all
```

```
  close all
  n=100;                % the number of frames
  Mellipse=moviein(n);  % initializing the matrix that will contain the information
for the movie


  a=1.6;                % the value of the big axis of the ellipse
  b=0.9;                % the value of the big axis of the ellipse
  t=[0:2*pi/n:2*pi];    % setting the values of t where we want to see the circles
  x=a*cos(t);           % finding the x coordinate from the parametric equation
of an ellipse
  y=b*sin(t);           % finding the y coordinate from the parametric equation
of an ellipse


  k=a*b./(a^2.*sin(t).^2+b^2*cos(t).^2).^(3/2);   % the curvature of the ellipse
at the chosen t values
  r=1./k;               % the radii of the osculating circles at each chosen
value of t


  xo=x-b*r.*cos(t)./((b^2)*cos(t).^2+a^2*sin(t).^2).^(1/2); % the x cordinate
of the centers of the circles
  yo=y-a*r.*sin(t)./((b^2)*cos(t).^2+a^2*sin(t).^2).^(1/2); % the y cordinate
of the centers of the circles


  m=2*max(r)-min([a,b]); % calculating about how big we want the axes of our
figures to be
  for i=1:n+1


  xc=xo(i)+r(i)*cos(t);  % finding all the x coordinates necessary to draw the
osculating
     % circle at the point i*2pi/n
  yc=yo(i)+r(i)*sin(t);  % finding all the x coordinates necessary to draw the
osculating
     % circle at the point i*2pi/n


  figure(1)
  plot(x,y, 'k')            % draw the ellipse
```

```
  hold on                    % keep the figure from being erased
  plot(xc,yc,'r')            % draw the osculating circle at at the point i*2pi/n
  plot(xo(1:i),yo(1:i))      % draw all the centers of the circles up to that
point
  plot([xo(i),x(i)], [yo(i),y(i)], 'r--')   % draw the normal to the ellipse
  hold off                  % allow the figure to be erased
  axis([-m m -m m])         % set the axis to some appropriate value
  Mellipse(:,i)=getframe;  % put the information from the resulting figure
       % as a column in the movie matrix
  end
  movie(Mellipse)              % play the movie
```