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.