▹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 $\frac{dy}{dt} = X^1(y, t, t_2^0)$ 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:

1.0.1 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:

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

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.

1.1 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.