580.439/639 Project #1

Project Assignment: due October 28, 2013, 5:00 P.M. in 505 Traylor or by email (PDF file) to E. Young or K. Hageman.

The purpose of this project is to study some basic properties of systems of non-linear differential equations like those used to model neurons. Matlab is used to solve the equations numerically. Matlab must be used; there are many other packages, some very nice ones for neural modeling, but the goal is to make sure you understand the basics. In the first part of the project, a simple neuron model, the Morris-Lecar equations (MLE), will be studied as an introduction to non-linear dynamics and simulation methods. The MLE are discussed in chapt. 7 of Koch and Segev, second edition (the chapter by Rinzel and Ermentrout, called R&E below). Note that the model considered in R&E is slightly different in the first and second editions, so make sure you read R&E in the second edition. The second part studies a model of neural oscillation by Manor and colleagues *J Neurophysiol* 77:2736-2752, 1997.

It will be helpful to read the chapter on numerical methods (chapter 14 in Koch and Segev, second edition) by Mascagni and Sherman. This is a readable introduction to the algorithms used to integrate sets of differential equations and will help you to know what happens when you run a simulation.

If you have not yet used the Matlab ordinary differential equation (ODE) solvers, you should prepare for this project by reading the sections of the Matlab doc files on commands beginning with *ode* and running some of the examples described there.

Appendix 1 contains listings of Matlab m-files for the MLE (except for mlec). The files can be downloaded from the course web page. There are nine m-files in this directory:

mlesim.m – simplest possible code to obtain a solution to the MLE. Run this first just to see what happens.

mlec.m – slightly fancier version that gives you control over the initial conditions and allows more displays. If you wish, you can use this as the basis for your own code.

mlode.m – the ODE file for the MLE model, called by both the above.

mlodejac.m – function to compute the Jacobian of the MLEs.

minf.m, winf.m, and tauw.m – functions that compute m_∞, w_∞, and τ_w for the model.

setmleparms, getmleparms – set and get the parameters of the model. Used internally to avoid parameter passing which can be a problem for Matlab routines like numjac(), that may be useful in this project..

Looking at mlesim.m first, the parameters of the model are defined by two column vectors *pml* and *iext* from the workspace. The parameters contained in these are defined in the comments and are set by the first two executable (non-comment) statements in the file, using the same names

as in R&E; the parameters are set to values near those used for Fig. 7.1. The last two parameters in *pml* are not used. *iext* contains parameters of a steady current pulse which can be applied to the model. The pulse has amplitude *iext(1)* $\mu A/cm^2$ (set to 0 in mlesim), starts at time *iext(2)* ms and ends at time *iext(3)* ms. The rest of mlesim.m sets up the minimal conditions for running an ODE solver, runs the solver (ode15s) to obtain a solution (in *t* and *y*), and plots it in a figure window. The state vector *y* is [*V,W*]'. The code in mlesim.m should be clear to you after you have read the introductory Matlab material mentioned above.

You will want to customize mlesim or mlec for your particular computing style and you will have to add code to the files to do the problems below (find equilibrium points, plot nullclines, etc).

The m-file mlode.m is the ODE file for the MLE; it returns dy/dt for a point y. mlodejac.m returns the Jacobian of the MLEs and is needed only to speed up calculations.

For the MLE model, it is relatively easy to work out the Jacobian matrix. However, for more complex models this may be too tedious. It is possible to compute Jacobians numerically with the Matlab numjac() function. The mlodejac.m file is set up to do the direct computation, but contains comments explaining the use of numjac(). Be careful with numjac(); it has a tendency to change slightly from version to version of Matlab.

A cogent write-up summarizing the answers to the following questions should be handed in on the due date listed above. The write-up should contain documentation supporting your answers and illustrations demonstrating the behavior of the models. The writeup must contain a narrative so it reads like a real paper. That is, don't just string together a bunch of Matlab printouts. The grading will be based on explaining how you got the answer, as well as on having the answer. These Matlab files have been rewritten slightly in response to changes in the recent versions of Matlab. If you get errors when you run them, consult with the instructors. Note the value of the parameter phi is different in the files from the value used by R&E. Please use the value (0.2) given in the programs.

Both the computations and the writeup must be your individual effort.

Specific Project Steps

The project has four parts. The first 6 steps are designed to make sure you can use Matlab for systems like this. Parts 7-9 illustrate the properties of MLE. Parts 10-12 illustrate properties of Hopf and saddle-node bifurcations. The remainder of the project studies a second model.

1. An essential step in developing a model is working out a consistent set of units for the variables. Usually it is desirable to make the numerical values of variables of the order 1; that is, it is possible to specify currents in amps, but then the numbers will be very small, which can cause problems with the numerical algorithms (see question 4 below). In neuron models, it is usually better to specify currents in microAmps or nanoAmps or picoamps. The same thing goes for potentials (mV), time (mS), and impedances. You are not free to choose any set of units, however, in that the units must be consistent. Thus volts, amps, siemens (1/ohms), farads, and seconds are a consistent set of units. For example, Ohms law requires that current=conductance x voltage, and the units are designed so that amps=siemens x volts. If you change volts to mV, you must make corresponding changes in the other units so that Ohm's law is still numerically true.

R&E choose the following units: milliseconds, microamps/cm^2, millivolts, microfarads/cm^2, and millisiemens/cm^2. Show that this is a consistent set of units. Typically in neuron models current and admittance values are expressed per unit area of membrane, sometimes using the capacitance of the cell (which is easy to measure) as a stand-in for its area (which is hard to measure), assuming that capacitance = 1 μFd/cm^2 x area

Suppose that it had been desirable to specify conductance as microsiemens/cm^2. Give a set of units consistent with μS/cm^2 for conductance. Is your solution unique?

2. The parameters in the mlesim.m file are set near the values for Fig. 7.1 in R&E. Find the equilibrium point for the system with this set of parameters (and with *iext*=0). You should come up with two different ways of doing this calculation (not counting looking at the caption of Fig. 7.1). Explain your methods.

   Plot a basic phase-plane for the system with *iext* = 0. It will be useful to write a small program to plot the nullclines, find the equilibrium points, and compute the eigenvalues at the equilibrium points. In finding the equilibrium points and other parameters in this work, it is not sufficient to just read an approximate value off the screen. You should compute the values accurately.

   A useful Matlab plotting subroutine for phase plane analysis is quiver() which can be used to make an arrow plot of the local directions of flow (i.e. the vector [*dV/dt, dW/dt*]) in the phase plane. Be warned, however, that Matlab requires that the X and Y values given to this function must have similar magnitudes. If you place *V* (max value about 100) on one axis and *W* (max value about 1) on the other axis, it is necessary to multiply *W* by 100 to get a usable plot (i.e. plot *V* versus 100**W*).

   In the following, when you make phase plane plots, it will usually be very helpful to your understanding to plot both trajectories and nullclines on the same plot.

   In the following, always run the model from its equilibrium point (i.e. set the initial values of *V* and *W* to their values at the equilibrium point) unless instructed to do otherwise. Remember that as you apply a steady-state (D.C.) current *iext*, the equilibrium point changes.

3. Is the equilibrium point found in part 2 stable (with *iext* = 0)? Check this by computing the eigenvalues of the Jacobian of the system at the equilibrium point. Obtain the Jacobian in three ways: 1) The function mlodejac() will return the Jacobian; 2) The Matlab function numjac() will evaluate the Jacobian numerically using calls to mlode(); 3) Derive an expression for the Jacobian of the MLE by hand. Once you have the Jacobian, evaluate its eigenvalues by hand or using the Matlab eig function.

   NOTE: if you use the discussion on P. 259 of R&E to understand the eigenvalue computation, be warned that there is an error in Eqns. 7.13 and 7.14. What is the error?

4. Explain why the default numerical tolerance values built into matlab (*AbsTol*=10^{-6} and *RelTol*=10^{-3}) are reasonable for the MLE. Suppose you set the units for the voltage variable in the MLE to kV (kilovolts). This would be a stupid thing to do, but if you did it, how would you have to change the values of *AbsTol* and *RelTol* to get the same error control? Understanding this issue is essential for accurate use of the ODE solvers.

5. If you run the mlec routine exactly as it is distributed, you will get an action potential similar to the one in R&E Fig. 7.1 (but not exactly the same). The difference between the two is in the parameter ϕ (0.02 here, 0.04 in R&E). Make a phase plane plot of your action potential (like Fig. 7.1B) and explain the difference in the shapes of your plot and the one in Fig. 7.1B, in terms of the effects of the parameter ϕ (this is a place where having the nullclines will be very helpful). Check your answer by making ϕ even smaller, say 0.01. Can the effects of ϕ be explained by changes in the nullclines or equilibrium point?

6. Fig. 7.1 of R&E shows responses to brief current pulses. In fact, these simulations were probably done by setting the initial condition on V to some value more positive than the equilibrium point while keeping the initial condition on W at the equilibrium point. Show (with equations) that resetting the initial condition on V in this way is equivalent to applying an impulse of current at i_{ext}. That is, show that running the model with

$$V(0) = v_1, \quad W(0) = w_0, \quad \text{and} \quad i_{ext}(t) = 0$$

is the same as running it with

$$V(0) = v_0, \quad W(0) = w_0, \quad \text{and} \quad i_{ext}(t) = i_0\,\delta(t)$$

where (v_0, w_0) is the equilibrium point and $\delta(t)$ is the Dirac delta. Derive an expression for v_1 in terms of v_0, i_0 and parameters of the model.

7. Simulate depolarizing current pulses of various amplitudes by setting the voltage initial condition to a succession of values positive to the resting potential while starting W at the equilibrium point. Find depolarizations that are sufficient to produce action potentials. Plot phase-plane trajectories for values of initial depolarization that do and do not produce action potentials. Make sure i_{ext}=0 for this part and make sure that the stopping time is large enough to see the whole action potential (300 ms or so).

   Action potentials are usually thought to have thresholds. Does the MLE with the parameters of Fig. 7.1 have a threshold depolarization? If your answer is yes, define what you mean by threshold. Be careful here, you should investigate values of v_1 between –14.9 and –15 mV carefully, i.e. to several decimal places; it will be useful to make a plot of the maximum amplitude of the action potential versus v_1.

8. Run the model with i_{ext}=86 μA/cm^2 with three sets of initial conditions: 1) Set the initial conditions to the equilibrium point used above, appropriate for i_{ext}=0; 2) Set the initial conditions to the equilibrium point when i_{ext}=86 μA/cm^2; and 3) Set the initial conditions off the equilibrium point for i_{ext}=86 μA/cm^2, say at (-27.9, 0.17). Make sure you set the time span for the simulation to be long enough to see the full response. Plot the three trajectories on a phase plane. Describe the stable states of this system (make sure to characterize the equilibrium point for i_{ext}=86 μA/cm^2 with eigenvalues).

   Explain the difference between trials 1) and 2) above. That is, describe two experiments in which current is applied to a cell to produce the responses you observed. Hint: the difference is in the *waveforms* of the applied current. Make sure you understand the difference.

9. The system with i_{ext}=86 μA/cm^2 apparently has two stable states. Find the unstable periodic orbit (unstable limit cycle) that divides the phase plane into those initial conditions that

converge to the equilibrium point and those that converge to the limit cycle. Do this by running the model backward in time (meaning that instead of solving $d\vec{X}/dt = \vec{F}(\vec{X})$, simulate $d\vec{X}/dt = -\vec{F}(\vec{X})$) to find an unstable periodic orbit like the UPO in R&E Fig. 7.3A. Make sure you understand what happens to 1) nullclines 2) equilibrium points and 3) stable and unstable solutions when the model is run backwards in time.

Show that the UPO is a true threshold in the sense that an infinitesimal change in the initial condition on *V* leads from a subthreshold waveform to an action potential.

10. Analyze the equilibrium points for i_{ext}= 80, 86, and 90 μA/cm^2 and show that a Hopf bifurcation occurs near here. Find the current at the bifurcation point. Consider the oscillating waveform near the bifurcation point. Does it correspond to that predicted by the eigenvalues of the system linearized around the equilibrium point for i_{ext}=86 (do this quantitatively, don't just answer "yes" or "no".)?

    Make a plot of the rate of firing action potentials versus the applied current over the range 80-100 μA/cm^2. In each case, start the system at its equilibrium point for the applied current , so the bifurcation is clear in the plot (i.e. the transition from 0 rate below the bifurcation point to a non-zero rate above the bifurcation).

11. Set your MLE program to the parameters of Figure 7.4 in the R&E chapter:

```
%   gca, gk, gl,vca,  vk,  vl,  phi,    v1,  v2, v3, v4,  v5, v6, C, vic,wic
pml=[4, 8.0, 2, 120, -84, -60, 0.0667, -1.2, 18, 12, 17.4 12,17.4,20, 0, 0]';
iext = [30., -1, 2000]';
```

    Note that the current should be set to a non-zero value. Determine the equilbrium point(s) of this system (there should be 3) and characterize them as to stability. Show, in a phase plane plot, the nullclines, equilibrium points, and manifolds (if there are any) for iext = 30.

12. Do a bifurcation analysis of this system in the current range between 30 and 50 μA/cm^2. Pay special attention to the range between 39 and 40. Show that a bifurcation occurs here and tell what kind it is (in terms of the names introduced in class). It will be necessary to show what happens to the equilibrium points as the current increases.

    Make a plot of the rate of firing action potentials versus the applied current over the range 30-45 μA/cm^2 and show how the behavior of this bifurcation is different from that of the Hopf bifurcation in part 10.

    The squid giant axon behaves like the bifurcation in 10, but most neurons behave more like the bifurcation in 12 (even though the systems are different, more complex in the neurons).

13. The second order system describe by Manor and colleagues (*J Neurophysiol* 77:2736-2752, 1997) has a T-type calcium channel and a leakage channel. It was designed to study oscillations occurring in neurons in the inferior olive. The ultimate goal of this work was to study oscillations in coupled cells. Here, we will only use the model to study the effect of parameter variations in single cells because of limited time.

    The model is written below:

$$C\frac{dV}{dt} = I_{ext} - G_T\, m_\infty^3(V)h\left(V - E_{Ca}\right) - G_L(V - E_L)$$

$$\frac{dh}{dt} = \phi\frac{h_\infty(V) - h}{\tau_h(V)}$$

where

$$m_\infty^3(V) = \left(1 + e^{(-61-V)/4.2}\right)^{-3}$$

$$h_\infty(V) = \left(1 + e^{(V+85.5)/8.6}\right)^{-1}$$

$$\tau_h(V) = 40 + 30\left(1 + e^{(V+84)/7.3}\right)^{-1} e^{(V+160)/30}$$

and

C = 1 μFd/cm^2 ϕ = 1

E_{Caa} = 120 E_L = -63

The remaining parameters, G_T, G_L, and I_{ext} are varied over the range shown in Figs. 2 and 3 of the original paper; the goal of the modeling effort is to explore the range of parameter values over which the model is stable at a fixed resting potential, is bistable at two possible resting potentials, or shows oscillations, either limit cycles or damped oscillations.

For part 13, write a set of Matlab programs to simulate this model. At minimum, there should be an ODE file, a program to construct a phase plane, including nullclines, and to find and characterize equilibrium point.

14. Construct four phase planes with the following parameter values:

1) $G_L = 0.25, G_T = 0.45, I_{ext} = 0$
2) $G_L = 0.175, G_T = 0.45, I_{ext} = 0$
3) $G_L = 0.12, G_T = 0.45, I_{ext} = 0$
4) $G_L = 0.05, G_T = 0.45, I_{ext} = 0$
5) $G_L = 0.05, G_T = 0.45, I_{ext} = -0.3$

In each case, show the nullclines and equilibrium points, characterize the equilibrium points as to stability, and argue that the system does or does not have a limit cycle in this condition. The last argument should be based on the theoretical methods discussed in class if possible. If not, consider trajectories which can either show a limit cycle or show that one is not possible.

15. Construct a bifurcation diagram (using I_{ext} as the bifurcation variable) for case 4) above and explain what is meant by bistability in the Manor et al. paper. Write a simple criterion for detecting bistability in a model and compute the range of currents over which model 4) is bistable.

Appendix 1: Matlab files for MLE

```
% MLESIM - simulate Morris-Lecar equations
% Needs a column vector pml of params:
%    pml=[gca, gk, gl, vca, vk, vl, phi, v1, v2, v3, v4, v5, v6, C, vic, wic]'
% For meaning of parameters, see Rinzel and Ermentrout.
% and a column vector iext describing the external current
%   iext = [iampl, tstart, tstop]'
% which are the current amplitude and the start and stop times.
% These are set up internally to the values for Fig. 7.1 of R&E if not supplied
% in the workspace.
% This always runs with i.c.s set to 0, just to give a spike.

% Parameters for Fig. 7.1. NOTE initial conditions are not included.
% NOTE VALUE OF phi CHANGED FROM FIG. 7.1 VALUE.
%    gca, gk, gl, vca,  vk,  vl,  phi,  v1,  v2,v3, v4,v5, v6, C,junk,junk
pml=[4.4, 8.0, 2, 120, -84, -60, 0.02, -1.2, 18, 2, 30, 2, 30, 20, 0,  0]';

% External current parameters.
%     iext tstart tstop
iext = [0,   0,    0]';

figure(1); clf   % Shouldn't be necessary but prevents an annoying bug
                 % in odeplot

% Store parameters for mlode:
setmleparms(pml, iext);

% Simulate for 100 ms from 0 initial conditions (should produce an AP)
tspan = [0; 100];
y0 = [0; 0];

% Show the state variables during the simulation, tell solver where to get
% the Jacobian.
options = odeset('OutputFcn', @odeplot, 'Jacobian', @mlodejac, ...
                    'Vectorized', 'on');
%options = odeset('Jacobian', @mlodejac, 'Vectorized', 'on');

% Do the simulation
[t,y] = ode15s(@mlode, tspan, y0, options);

% Print final value of state variables
current = y(end,:); fprintf('Final values: v=%g,  w=%g\n',current);

% Replot the state variables so the W variable can be seen
[ax, h1, h2] = plotyy(t, y(:,1), t, y(:,2));
axes(ax(1)); ylabel('V, mV.')
axes(ax(2)); ylabel('W')
xlabel('Time, ms.');
```

```
function setmleparms(pml, iext)

% Sets current values of MLE parameters. Call as
%       call setmleparms(pml, iext)
% This stores the parameters in an internal store from which they can be
% fetched by getmleparms. This is how mlode gets its parameters.
% pml is a column vector of params
%    pml=[gca, gk, gl, vca, vk, vl, phi, v1, v2, v3, v4, v5, v6 C vic wic]'
% and iext is a column vector describing the external current
%   iext = [iampl, tstart, tstop]'

global PMLXYZ IEXTXYZ

PMLXYZ = pml;
IEXTXYZ = iext;

return



function [pml,iext] = getmleparms

% Returns values of MLE parameters stored in internal store by setmleparms.
% Call as
%        [pml,iext] = getmleparms
% User should set parameters to be used by mlode with setmleparms. This routine
% is used by mlode to read the current parameter vector.
% pml is a column vector of params
%    pml=[gca, gk, gl, vca, vk, vl, phi, v1, v2, v3, v4, v5, v6 C vic wic]'
% and iext is a column vector describing the external current
%   iext = [iampl, tstart, tstop]'

global PMLXYZ IEXTXYZ

pml = PMLXYZ;
iext = IEXTXYZ;

return



function ydot = mlode(t, y)

% MLODE - ODE file for the Morris-Lecar Equations.
% Evaluates the derivative of the state vector for the Morris-Lecar
% equations with parameters pml, where pml is a column vector of params
%    pml=[gca, gk, gl, vca, vk, vl, phi, v1, v2, v3, v4, v5, v6 C vic wic]'
% and iext is a column vector describing the external current
%   iext = [iampl, tstart, tstop]'
% v5=v3 and v6=v4 are for tauw().
% Parameters are set through function SETMLEPARMS() only. mlode reads
% the parameters using GETMLEPARMS.
%       ydot = mlode(t,y);     returns dy/dt eval at t,y
% Note, also available:
%       jac = mlodejac(t,y)    returns the Jacobian at t,y
% Note mlode is vectorized, but mlodejac is not.
```

```
% Get parameters
    [pml,iext] = getmleparms;

% Compute
    ydot = zeros(2,1);
    if t>=iext(2) & t<iext(3)
       ydot(1) = (iext(1) - pml(1)*minf(y(1),pml).*(y(1)-pml(4)) - ...
           pml(2)*y(2).*(y(1)-pml(5)) - pml(3)*(y(1)-pml(6)))/pml(14);
    else
       ydot(1) = (-pml(1)*minf(y(1),pml).*(y(1)-pml(4)) - ...
           pml(2)*y(2).*(y(1)-pml(5)) - pml(3)*(y(1)-pml(6)))/pml(14);
    end
    ydot(2) = pml(7)*(winf(y(1),pml)-y(2))./tauw(y(1),pml);
return



function jac = mlodejac(t,y)

% MLODEJAC - computes Jacobian for MLE system.
% To get the Jacobian, proceed in one of the two ways:
% 1. t = 0; y = [state vector at the eq. pt. where the jacobian is desired];
%    setmleparms(pml, iext)     % Set parameters of MLE model
%    jac = mlodejac(t, y)       % Returns the jacobian
%
% 2. t = 0; y = [state vector]; fac = [];
%    setmleparms(pml, iext)
%    [jacn,fac]=numjac(@mlode, t, y, mlode(t,y), [1e-5;1e-5],fac,0)

% Parameter vector:
%          1    2   3   4    5   6   7    8   9  10  11  12  13  14 15   16
%    pml=[gca, gk, gl, vca, vk, vl, phi, v1, v2, v3, v4, v5, v6, C, vic, wic]'
% Get parameters:
    [pml,iext] = getmleparms;

% Compute Jacobian
    jac = zeros(2,2);
    jac(1,1) = (-pml(1)*0.5/(pml(9)*cosh((y(1)-pml(8))/pml(9))^2)*(y(1)-pml(4))
- ...
           pml(1)*minf(y(1),pml) - pml(2)*y(2) - pml(3))/pml(14);
    jac(1,2) = -pml(2)*(y(1)-pml(5))/pml(14);
    jac(2,1) = pml(7)*(0.5/ ...
         (pml(11)*cosh((y(1)-pml(10))/pml(11))^2*tauw(y(1),pml)) - ...
         (winf(y(1),pml)-y(2))*sinh((y(1)-pml(12))/pml(13))/(2*pml(13)));
    jac(2,2) = -pml(7)/tauw(y(1),pml);
return



function min = minf(v,pml)
% MINF - compute m-infinity for the MLE. m = minf(v,pml)
% where pml is the usual MLE parameter vector. v and m can be vectors.
   min = 0.5*(1 + tanh((v-pml(8))/pml(9)));
return
```

```
function win = winf(v,pml)
% WINF - compute w-infinity in the MLE.
%      w = winf(v,pml);
% where v can be a vector and pml is the usual MLE parameter vector.
   win = 0.5*(1 + tanh((v-pml(10))/pml(11)));
return
```

```
function tw = tauw(v,pml)
% TAUW - compute time constant for dw/dt in the MLE.
%     tau = tauw(v,pml);
% tau and v can be vectors.
   tw = 1./cosh((v-pml(12))/(2*pml(13)));
return
```

Appendix 2: Finding zeros with a Newton method

The problem is to find where $\vec{F}(\vec{X})=0$, where $\vec{F}$ is a vector-valued function of a vector $\vec{X}$ of variables (both N-dimensional). For this project, $\vec{F}$ is the set of functions defining the r.h.s. of the differential equations being simulated (as defined in your ODE file) and $\vec{X}$ is the state vector. Suppose there is a starting condition $\vec{X}_0$ and we want to move to a value $\vec{X}_Z$ where $\vec{F}(\vec{X}_Z)=0$ to a good approximation. This can be done in steps using a Newton method. Consider a first-order approximation of a step from $\vec{X}$ to $\vec{X}+d\vec{X}$:

$$\vec{F}(\vec{X}+d\vec{X}) \approx \vec{F}(\vec{X}) + \mathbf{J}\, d\vec{X}$$

where $\mathbf{J}$ is the Jacobian of $\vec{F}$ with respect to $\vec{X}$. Now if we want $\vec{F}(\vec{X}+d\vec{X})$ to be zero, then

$$\vec{F}(\vec{X}) + \mathbf{J}\, d\vec{X} = 0 \quad \text{or} \quad d\vec{X} = -\mathbf{J}^{-1}\, \vec{F}(\vec{X})$$

and $\vec{X}_1 = \vec{X}_0 + d\vec{X}$ should be closer to the zero of $\vec{F}$. This process can be iterated until $d\vec{X}$ is smaller than a preset tolerance or until $\vec{F}(\vec{X})$ is sufficiently close to zero.

To do the calculations in Matlab, numjac() can be used to compute the Jacobian of your ODE file and the \ function (see "help mldivide") can be used to compute $d\vec{X}$.