How does the solution to an ODE change with changes in parameters

Consider $\dot{x} = f(t, x, \lambda)$, $x, t, \lambda \in \mathbb{R}$.

Given some initial condition, x_0, the solution is $x(t)$.

How does $x(t)$ change when we change λ?

(λ is a fixed parameter in each simulation)

Method 1

Say $x_{\lambda^*}(t)$ is the solution for λ^*.

And $x_{\lambda^*_{th}}(t)$ is the solution for λ^*_{th}.

Then $\left.\frac{\partial x(t)}{\partial \lambda}\right|_{\lambda=\lambda^*} \approx \frac{x_{\lambda^*_{th}}(t) - x_{\lambda^*}(t)}{h}$. finite difference approximation

"sensitivity" of $x(t)$ to λ changes at $\lambda = \lambda^*$

For now, we keep the initial conditions the same when we examine sensitivity to lambda

This partial derivative is a slight abuse of notation. Here's an explanation: given $x_\lambda(t)$, the solution for some fixed λ, define $Y(t, \lambda) = x_\lambda(t)$. Then, we use the notation:

$$\frac{\partial x_\lambda(t)}{\partial \lambda} = \frac{\partial Y(t, \lambda)}{\partial \lambda}$$

Method 2

"Sensitivity equations" (2)

We want $\frac{dx}{d\lambda}(t)$. We can get an equation for this by differentiating the ODE.

$$\frac{dx}{dt} = f(t, x, \lambda) \quad (1)$$

$$\frac{d}{d\lambda}\left(\frac{dx}{dt}\right) = \frac{d}{d\lambda}\left[f(t, x, \lambda)\right]$$

Switch order — chain rule.

$$\frac{d}{dt}\left(\frac{dx}{d\lambda}\right) = \frac{\partial f}{\partial \lambda} + \frac{\partial f}{\partial x}\frac{dx}{d\lambda}$$

If we call $p = \frac{dx}{d\lambda}$, we have

"variational equations"

$$\frac{dp}{dt} = \frac{\partial f}{\partial \lambda} + \frac{\partial f}{\partial x} p \quad (2)$$

a "sensitivity equations" ODE (2)

to get $\frac{dx(t)}{d\lambda}$, we just integrate this ODE (2)

along with (1). They are coupled ODEs.

Initial conditions $p(0) = \frac{dx(0)}{d\lambda} = 0$, because the initial condition does not depend on λ.

In this chain rule step, the partial derivative with respect to λ has an unconventional interpretation. Here, we differentiate with respect to λ keeping t fixed, but accounting for x changing due to λ changes. i.e., we don't keep x fixed in this partial derivative. This is more apparent if we make the following substitutions:

$$f(t, x_\lambda, \lambda) = f(t, Y(t, \lambda), \lambda) = g(t, \lambda)$$

What if we want to determine sensitivity to initial condition ?

$$\boxed{\frac{dx}{dt} = f(t,x)} \quad (1)$$

We want $\frac{\partial x(t)}{\partial x_0} = q$, say.

Differentiate (1) with respect to x_0

$$\frac{\partial}{\partial x_0}\left(\frac{dx}{dt}\right) = \frac{\partial}{\partial x_0}\left[f(t,x)\right]$$

$$\frac{d}{dt}\left[\frac{\partial x(t)}{\partial x_0}\right] = \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial x_0}$$

$$\boxed{\frac{dq}{dt} = \frac{\partial f}{\partial x} \cdot q} \quad (2)$$

Adjoint equation for sensitivity

Initial condition for q, $\frac{\partial x(0)}{\partial x_0} = 1$. (identity matrix for multivariate case)

Thus, we can determine $\frac{\partial x(t)}{\partial x_0}$ by integrating eqn (2) with (1).

Variational equations for ODEs in higher dimensions

Consider the ODE $\dot{x} = f(t, x, \lambda)$ (1)

where $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}^m$. Say $x(0) = x_0$.

Define Jacobian matrix $D_\lambda x$ ($n \times m$ matrix) $=$

$$\begin{bmatrix} \frac{\partial x_1(t)}{\partial \lambda_1} & \frac{\partial x_1(t)}{\partial \lambda_2} & \cdots & \frac{\partial x_1(t)}{\partial \lambda_m} \\ \frac{\partial x_2(t)}{\partial \lambda_1} & \cdots & & \cdots \\ \vdots & \cdots & & \frac{\partial x_n(t)}{\partial \lambda_m} \end{bmatrix}$$

is the matrix of partial derivatives of the solution $x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix}$ from a given initial condition x_0, with respect to changes in parameter λ.

Note that λ is fixed during the solution of the ODE and we are seeking the variation of the solution to variations in λ.

The Jacobian $D_\lambda x$ satisfies the so-called "sensitivity equations" or "variational equations" obtained by differentiating the ODE (1).

$$\underbrace{\frac{d}{dt}\left(D_\lambda x\right)}_{n\times m \text{ matrix}} = \underbrace{D_x f}_{n\times n \text{ matrix}} \cdot \underbrace{D_\lambda x}_{n\times m \text{ matrix}} + \underbrace{D_\lambda f}_{n\times m \text{ matrix}} \qquad (2)$$

where, as consistent with our notation,

$$D_x f = \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \cdots \\ & \ddots & \\ \cdots & & \partial f_n/\partial x_n \end{bmatrix}.$$

$(i,j)^{th}$ element of $D_x f = \partial f_i/\partial x_j$

$(i,j)^{th}$ element of $D_\lambda f = \partial f_i/\partial \lambda_j$

$(i,j)^{th}$ element of $D_\lambda x = \partial x_i/\partial \lambda_j$

etc.

Thus, differential equation (2) is a MATRIX differential equation. for the $n \times m$ matrix of sensitivities.

Note: In our notation, we have said $X(t)$. But really, it is better to use $X(t, \lambda)$ to illustrate the dependence on λ.

Note 2: Initial conditions for the sensitivities $\Big]$ $D_\lambda X(0) = 0$ (the zero matrix) because x_0 do not depend on λ.

Sensitivity to initial conditions x_0

$D_{x_0} X$ = $n \times n$ matrix of partial derivatives of solution $X(t)$ with respect to initial conditions x_0

Again, we can get an ODE for this matrix by differentiating (1).

$$D_{x_0} \dot{X} = \frac{d}{dt}\left(\underbrace{D_{x_0} X}_{n \times n \text{ matrix}}\right) = D_{x_0} f = \underbrace{D_x f}_{n \times n \text{ matrix}} \cdot \underbrace{D_{x_0} X}_{n \times n \text{ matrix}}.$$

Thus, the ODE governing the matrix of sensitivities is

$$\frac{d}{dt}\left(D_{x_0}x\right) = D_x f \cdot D_{x_0}x \qquad (3)$$

an $n \times n$ matrix ODE. The initial condition for the matrix of sensitivities is the identity matrix.

$$D_{x_0}x(0) = I$$

Example

Lorenz equations

famous for its chaotic attractor (inspired by weather prediction

$$\left.\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned}\right\} \quad (1)$$

state variable $s = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^3$

parameters $\lambda = \begin{bmatrix} \sigma \\ \beta \\ \rho \end{bmatrix} \in \mathbb{R}^3$.

$$\dot{s} = f(s, \lambda) \quad (1)$$

$D_{s_0} s(t)$ = 3×3 matrix of sensitivity to initial conditions

$$= \begin{bmatrix} \frac{\partial x(t)}{\partial x_0} & \frac{\partial x(t)}{\partial y_0} & \frac{\partial z(t)}{\partial z_0} \\ \frac{\partial y(t)}{\partial x_0} & \frac{\partial y(t)}{\partial y_0} & \frac{\partial y(t)}{\partial z_0} \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$\boxed{\frac{d}{dt}\left(D_{s_0} s\right) = D_s f \cdot D_{s_0} s} \quad (2)$$

with initial conditions $\boxed{D_{s_0} s(0) = I}$ identity matrix

(3)

$$D_s f = \begin{bmatrix} -\sigma & \sigma & 0 \\ \rho & -1 & x \\ y & x & -\beta \end{bmatrix} \quad (4)$$

use in (2)

We simulate (1) and (2) simultaneously, with their appropriate initial conditions.

i.e., we can use the same ODE file to encode both ODEs (1) and (2).

(1) is 3 scalar ODEs. (2) is a matrix (3×3) ODE $\equiv$ 9 scalar ODEs.

$\Rightarrow$ we can write an ODE function with $3+9 = 12$ scalar ODEs.

- Solving this system will not only give us a solution $\left[x(t), y(t), z(t)\right] = S(t)$ we will also get the 3×3 matrix of sensitivities of $S(t)$ with respect to $S(0)$.

Contents

- parameters for the Lorenz equation
- simulate for some initial conditions
- simulating LOTs of initial conditions starting very close to each other
- simulating the sensitivity equations along with the original ODE
- plot the maximum singular value of the sensitivity matrix as a function of time


```
clear; close all; clc;
```

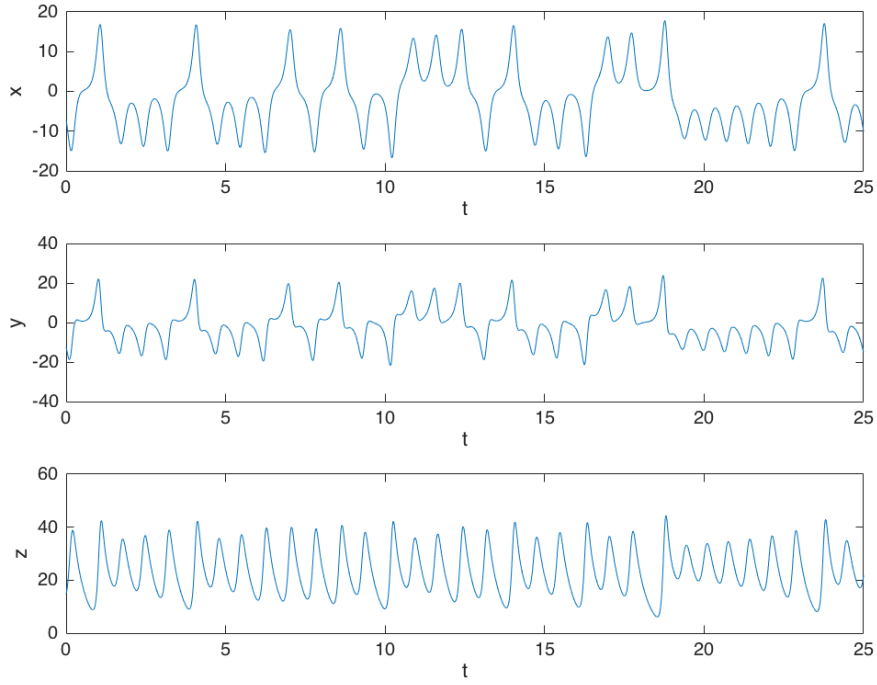
parameters for the Lorenz equation

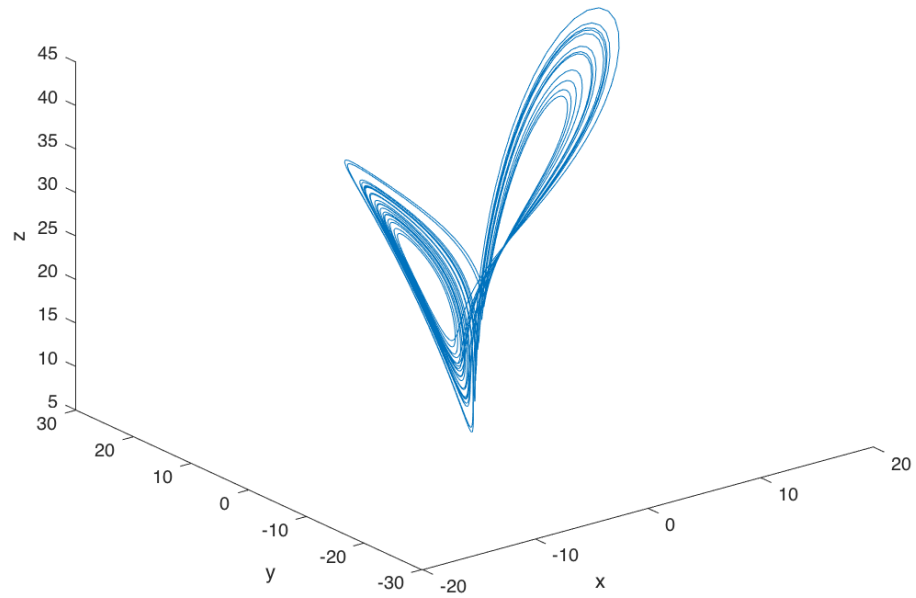
see wikipedia page for more details

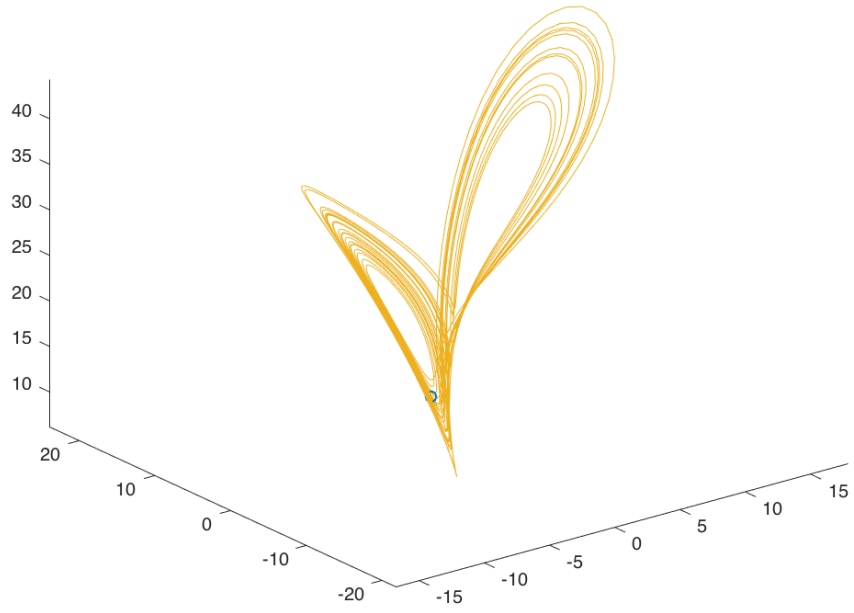
```
param.rho = 28;
param.sigma = 10;
param.beta = 8/3;
```

simulate for some initial conditions

```
tMax = 25;
tSpan = linspace(0,tMax,tMax*100); % time to simulate

stateVar0 = [-7.152697136677542; % some initial conditions close to the chaotic attractor
 -12.240110163291554;
  14.803748615043197];
options = odeset('reltol',1e-8,'abstol',1e-8);

[tList,stateVarList] = ode45(@odeLorenz,tSpan,stateVar0,options,param);

figure(1);
subplot(311); plot(tList,stateVarList(:,1)); xlabel('t'); ylabel('x');
subplot(312); plot(tList,stateVarList(:,2)); xlabel('t'); ylabel('y');
subplot(313); plot(tList,stateVarList(:,3)); xlabel('t'); ylabel('z');

figure(2);
plot3(stateVarList(:,1),stateVarList(:,2),stateVarList(:,3));
xlabel('x'); ylabel('y'); zlabel('z');

figure(3); % xlabel('x'); ylabel('y'); zlabel('z'); hold on;
comet3(stateVarList(:,1),stateVarList(:,2),stateVarList(:,3),0.001);
```

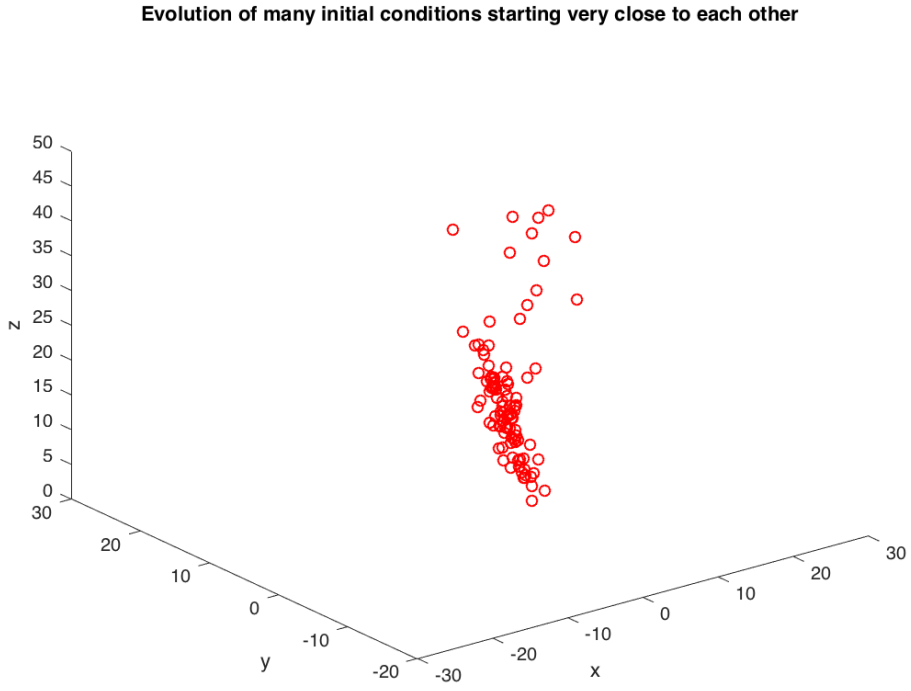

x
t
y
t
z
t


z
y
x



simulating LOTs of initial conditions starting very close to each other

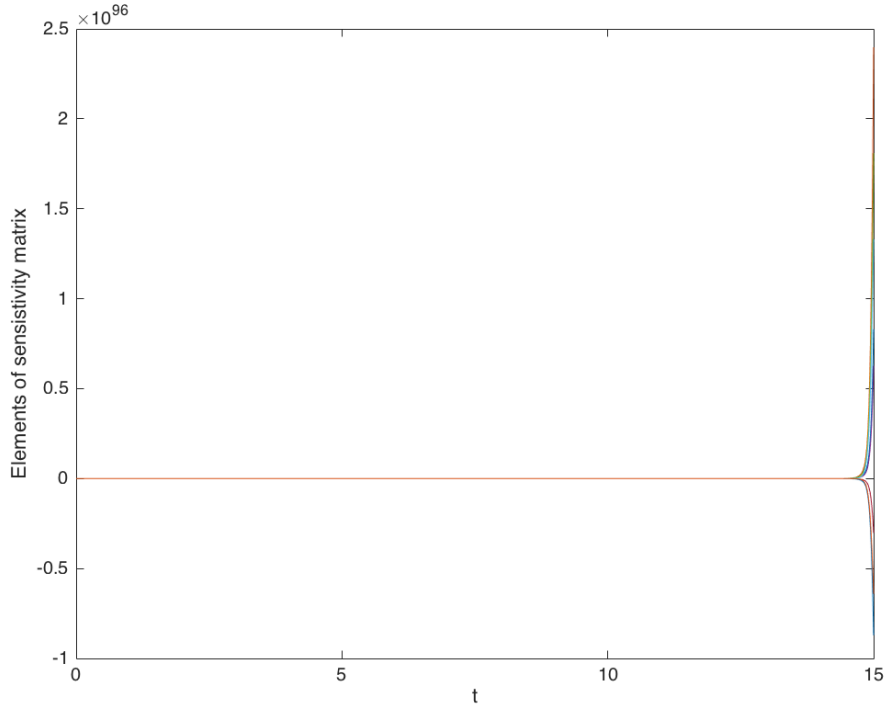
```
tMax = 15;
tSpan = linspace(0,tMax,tMax*25); % time to simulate

stateVar0 = [-7.152697136677542; % some initial conditions close to the chaotic attractor
 -12.240110163291554;
  14.803748615043197];
numInitialConditions = 100;
stateVar0Matrix = repmat(stateVar0,1,numInitialConditions) + ...
    1e-2*randn(3,numInitialConditions); % a whole bunch of initial conditions VERY close to each other
stateVarStore = cell(numInitialConditions,1);
for i = 1:numInitialConditions
    stateVar0 = stateVar0Matrix(:,i);
    [tList,stateVarStore{i}] = ode45(@odeLorenz,tSpan,stateVar0,options,param);
end
figure(4);
for j = 1:length(tList)
    a = [];
    for i = 1:numInitialConditions
        a = [a; stateVarStore{i}(j,1),stateVarStore{i}(j,2),stateVarStore{i}(j,3)];
    end
     plot3(a(:,1),a(:,2),a(:,3),'ro'); hold on;
     xlabel('x'); ylabel('y'); zlabel('z');
     title('Evolution of many initial conditions starting very close to each other');
     xlim([-30 30]); ylim([-20 30]); zlim([0 50]);
%      pause
    pause(0.01);
    hold off;
end
```



simulating the sensitivity equations along with the original ODE

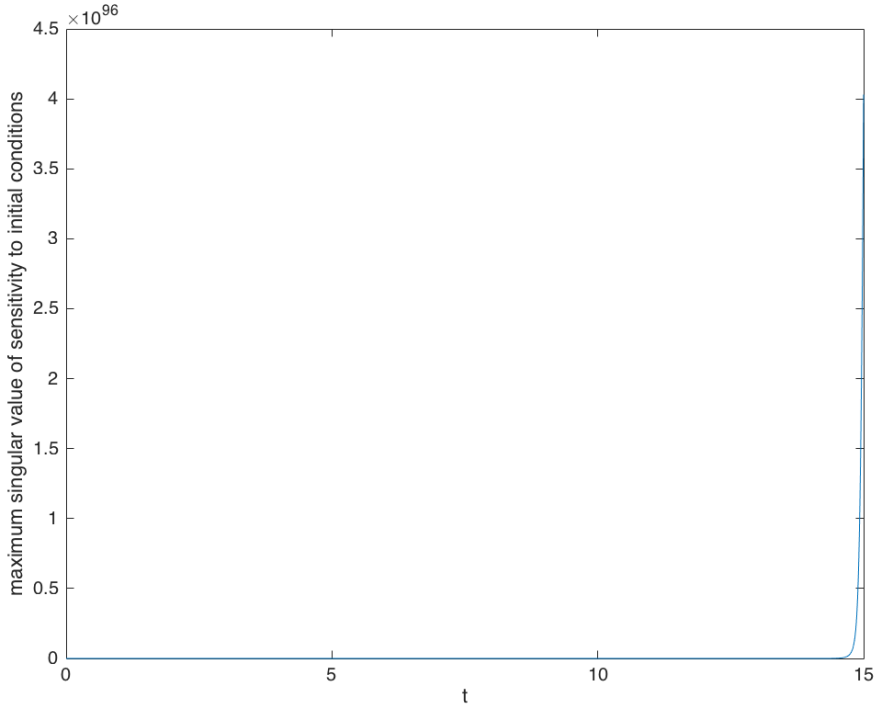
thus, we are simulating 12 ODEs instead of 3 ODEs

```
tMax = 15;
tSpan = linspace(0,tMax,tMax*100); % time to simulate

DsDs0_0 = eye(3); % how initial conditions change wrt initial conditions

% append state to sensitivity matrix, so they can be simultaneously simulated
% of course, the 3x3 matrix needs to be reshaped into a 9x1 matrix, so that
% ode45 can handle it
stateAndSensitivity0 = [stateVar0; reshape(DsDs0_0,numel(DsDs0_0),1)];

[tList,stateAndSensitivityList] = ode45(@odeLorenzWithSensitivity,tSpan,stateAndSensitivity0,options,param);

% plot the individual sensitivity matrix components
figure(5);
plot(tList,stateAndSensitivityList(:,4:12)');
xlabel('t'); ylabel('Elements of sensistivity matrix');

% note that the sensitivity equations characterize the expansion for
% initial conditions that are arbitrarily close to each other. This
% expansion grows exponentially. But the expansion slows down when the
% initial conditions are further apart, for instance, manifested in
% boundedness of the trajectories.
```



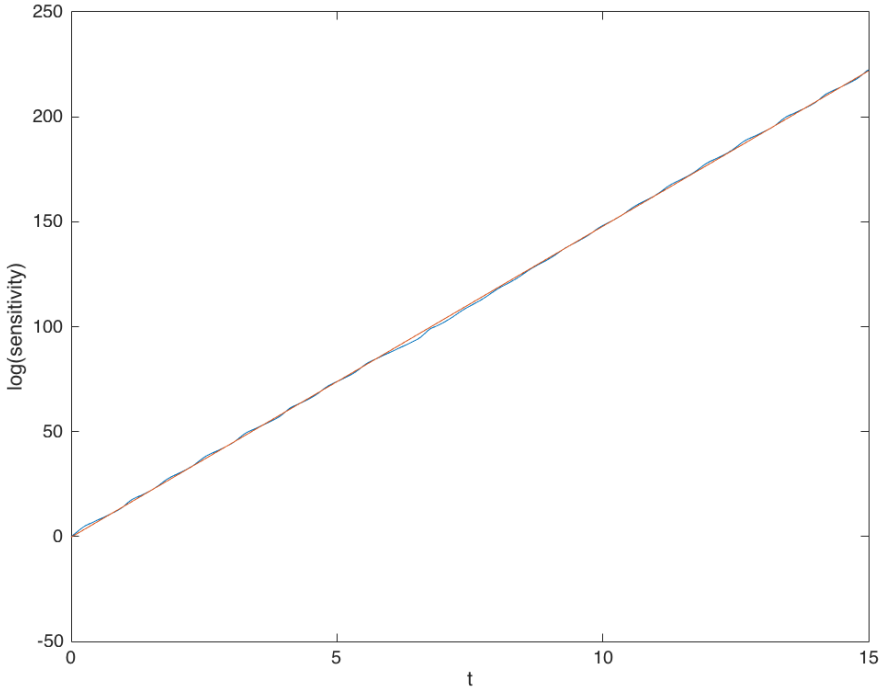
plot the maximum singular value of the sensitivity matrix as a function of time

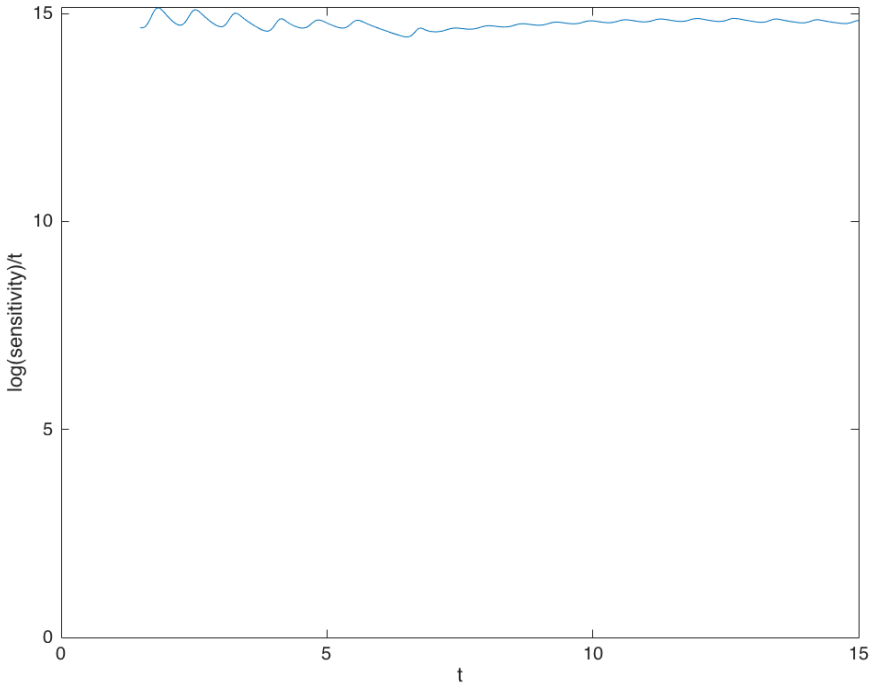
```
maxSingularValueList = [];
for i = 1:length(tList)
   DsDs0 = stateAndSensitivityList(i,4:12);
   DsDs0 = reshape(DsDs0,3,3);
   maxSingularValueList = [maxSingularValueList; max(svd(DsDs0))];
end

figure(6);
plot(tList,maxSingularValueList);
xlabel('t'); ylabel('maximum singular value of sensitivity to initial conditions');
```




```
figure(7);
logSensitivity = log(maxSingularValueList);
plot(tList,logSensitivity);
xlabel('t'); ylabel('log(sensitivity)');

% time averaged maximal Lyapunov exponent is roughly the slope of the linear fit to the log of the
% max singular value of the sensitivity matrix (eventually)
coeffsRegression = regress(logSensitivity,[ones(size(tList)) tList]); % doing a linear fit
LyapunovExponentEstimate1 = coeffsRegression(2)
hold on; plot(tList,coeffsRegression(1)+coeffsRegression(2)*tList)

% maximal Lyapunov exponent is the limit of the 1/t times log of the
% max singular value of the sensitivity matrix
figure(8);
temp = logSensitivity(0.1*end:end)./tList(0.1*end:end);
plot(tList(0.1*end:end),temp);
xlabel('t'); ylabel('log(sensitivity)/t');
ylim([0 max(temp)]);
LyapunovExponentEstimate2 = temp(end)
```

```
LyapunovExponentEstimate1 =

  14.813254568940248


LyapunovExponentEstimate2 =

  14.829463308501168
```





Published with MATLAB® R2015b

SENSITIVE DEPENDENCE ON INITIAL CONDITIONS

Simple example:

$$\dot{x} = kx .$$

Given $x(0) = x_0$,

$$\boxed{x(t) = x_0 e^{kt}} \quad -(1)$$

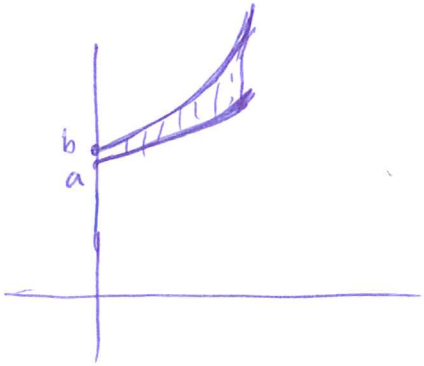
Say we know $x_0 > 0$, but we do not know exactly what x_0 is.

Say we know that $a \leq x_0 \leq b$, equivalently $x_0 \in [a, b]$.

If $x_0 = a$, $x(t) = a e^{kt}$.

If $x_0 = b$, $x(t) = b e^{kt}$.

If $k > 0$



i.e., $\boxed{a e^{kt} \leq x(t) \leq b e^{kt}}$ -(2)

The size of the interval containing $x(t)$ is $(b-a)e^{kt}$ -(3)

If $k > 0$, the size of the interval containing $x(t)$ grows exponentially in time: $(b-a)e^{kt}$.

If $k < 0$, the size of the interval containing $x(t)$ decays exponentially in time.

Another way to look at these equations is:

$(b-a)$ is the initial uncertainty / error in x_0.

$(b-a)\,e^{kt}$ is the uncertainty / error in $x(t)$ at time t.

e^{kt} is the amplification factor of the initial error.

The amplification factor grows or shrinks depending on k.

Let us go back to Eq. ①.

$$x(t) = x_0 e^{kt} \quad \text{--- ①}$$

$\frac{\partial x(t)}{\partial x_0}$ gives the sensitivity of $x(t)$ for small changes in x_0.

$\frac{\partial x(t)}{\partial x_0} = e^{kt}$. Indeed this sensitivity is the same as the amplification factor above (because it is a linear system).

If one had a general nonlinear system,

$\dot{x} = f(x)$, say,

we can define the sensitivity to initial conditions as the Jacobian of $X(t)$ with respect to the initial condition X_0 at some initial conditions X_0.

$$J = D_{X_0} X(t).$$

This Jacobian describes how small errors ΔX_0 in the initial condition X_0 grow or decay over time t.

$$\Delta x(t) = J(t) \cdot \Delta X_0$$

$\Delta x(t)$: error in prediction of $X(t)$.

$J(t)$: Jacobian, sensitivity to initial conditions

ΔX_0: error in initial condition X_0.

Note: J is an $N \times N$ matrix if X is an N-vector.

How to compute $J(t)$?

(Simplest method, but not the most efficient).

- Write a function that takes as input X_0 and has the input $X(t)$. [for some time t].
- Find the numerical Jacobian of this function : $X(t)$ wrt X_0. This is $J(t)$.
- Do this over and over for different values of t.

Because J is a matrix, we would like a scalar quantity that measures the sensitivity to initial conditions.

→ How much J stretches or contracts a typical error in initial conditions can be quantified by using the singular values of J.

(Singular values are similar to, but not quite, eigenvalues of J. The singular values are usually represented by σ_i. and are defined for non-square matrices. The squares of the non-zero singular values of J are the non zero eigenvalues of JJ^T or J^TJ. Singular value function in MATLAB is SVD)

In any case, the maximum singular value measures the worst case amplification of an initial error.

For the system $\dot{x} = kx$; the Jacobian is $J = e^{kt}$.

and its maximum is itself e^{kt}.

Thus, here when $k>0$, the maximum singular value of J grows exponentially in time

If we consider the maximal singular value of the sensitivity jacobian J to grow exponentially, the exponent k is called the "maximal Lyapunov exponent". This can be obtained by plotting log(maximal singular value) vs t and measuring the slope of this line (asymptotically). If slope k>0, we have sensitive dependence on initial conditions -- as in "chaos". Initial conditions initially close together diverge exponentially