SOME USEFUL NUMERICAL / COMPUTATIONAL ISSUES AND IDEAS FOR NON-LINEAR DYNAMICS

FINDING FIXED POINTS

Finding fixed points, be it for ODEs or for discrete dynamical systems is all about solving N nonlinear equations in N unknowns.

ODEs

$$\dot{X} = f(X)$$

FPS when $f(X) = 0$ ← N eqns in N unknowns

unknowns $X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}$ $f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} = 0$ ← equations

$$f_1(x_1 \dots x_N) = 0$$
$$f_2(x_1 \dots x_N) = 0$$
$$\dots$$
$$f_N(x_1 \dots x_N) = 0$$

Discrete dynamical Systems

$$X_{i+1} = f(X_i)$$

FPS when $X = f(X)$.

(or $f(X) - X = 0$.)

N equations in N unknowns.

$$f_1(x_1 \dots x_N) - x_1 = 0$$
$$f_2(x_1 \dots x_N) - x_2 = 0$$
$$\vdots$$
$$f_N(x_1 \dots x_N) - x_N = 0$$

How to solve N nonlinear equations in N unknowns?

(1) Sometimes we only have linear equations, so those are relatively easy to solve, either by hand or using a computer. One is then guaranteed knowledge of existence and uniqueness of solutions.

↑ may or not be unique but it is easy to see which.

(2) Simple nonlinear equations can be solved by hand/symbolic toolbox if there are only a few of them.

(3) More generally, one is forced to use numerical methods to find the fixed points. These are iterative methods.

Examples: Newton-Raphson to find the solution of $f(x) = 0$, $x \in \mathbb{R}$.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$ ← start with a guess and repeat the iteration.

If you start x_0 close enough to x^*, we are guaranteed convergence typically.

Newton's method for N eqns in N unknowns (generalization of above)

$$f(x) = 0$$

Start with some guess x_0 and repeat the following iteration:

$$X_{i+1} = X_i - J^{-1}(X_i) \cdot f(X_i)$$

↑ Jacobian of f w.r.t X at X_i.

Both these "basic" algorithms have convergence issues. fsolve in MATLAB is an implementation of algorithms closely related to the Newton's method, but made a bit more robust in their performance.

Exercise: Try using fsolve to solve the following 2 equations in 2 unknowns.

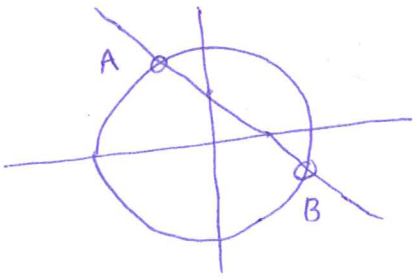
```
→ function f = simplifunction(X)
    x1 = X(1); x2 = X(2);
    f = [x1^2 + x2^2 - 1 ; x1 + x2 - 0.5]
end
```

← defines the 2 equations we want to choose $X = [x1; x2]$ such that $f = [0; 0]$.

```
→ Xresultt = fsolve(@simplifunction, [0,0])   ← does not converge to an answer
→ Xresut   = fsolve(@simplifunction, [1, 0])  ← converges to B
→ Xresultt = fsolve(@simplifunction, [0, 1])  ← converges to A
```

↑ initial guess of the solution

↑ Converges to (0.5, 0.5) not a solution



Note 1) There is no guarantee that any given initial guess will converge to a solution.

2) To increase the likelihood of finding all solutions, run fsolve from a lot of different initial guesses.

3) Sometimes despite trying many initial guesses, fsolve might not find any solutions. This might mean one of 2 things

a) The equations have no solutions

b) The equations do have solutions, but you just have not found them yet.

Note: You can check that the answer is correct by evaluating f and noting that it is very close to zero.

Most numerical methods for the solution of nonlinear equations cannot provide guarantees.

However, there is a class of methods based on interval arithmetic which can provide guarantees related to existence and number of solutions. These methods may be inefficient for large N.

Finding limit cycles using fsolve.

Say P(Y) is a Poincaré map (function) that takes a point Y on a Poincaré section to another point on the Poincare section by following the trajectory of the ODE to the next intersection with the section.

As noted earlier, a fixed point Y^* of the Poincare map, $Y^* = P(Y^*)$, gives a point on a limit cycle.

Therefore, to find the fixed point, we solve the equations $Y - P(Y) = 0$, which can again be accomplished using fsolve.

NUMERICAL DERIVATIVES, NUMERICAL JACOBIANS

Derivatives are required for various purposes, but here we wish to compute them to study stability of various attractors.

There are various ways to compute or estimate derivatives of functions.

(1) Symbolic differentiation, either by hand or by using some symbolic manipulation toolbox. (eg. MATLAB's symbolic toolbox, MAPLE, MATHEMATICA, etc).

Con - This is practical mainly for small to medium sized problems.

Pro - But the accuracy of the derivative is as good as the accuracy of how well you can evaluate functions.

(2) Numerical differentiation: using forward difference, central difference, etc.

Pro - easy to implement even for complicated functions of many variables.

Con - The accuracy of the derivative is orders of magnitude lower than the accuracy to which you can evaluate the functions.

Can be less efficient than "automatic differentiation".

(3) Automatic differentiation: also called "algorithmic differentiation" is an efficient algorithmic way to obtain derivatives with accuracy as high as the function evaluations themselves and with not as high a computational cost as numerical differentiation (if implemented efficiently).

Automatic differentiation often abbreviated to AD is often confused with normal symbolic differentiation but is way more efficient (for various reasons, eg. because it uses numbers instead of symbols). Ref: See books by Andreas Griewank.

We'll just discuss numerical differentiation below.

Forward difference

Given $f(x)$.

$$\frac{df}{dx} \approx \frac{f(x+h) - f(x)}{h}$$, where h is a small number.

When f can be evaluated to infinite precision,

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = \frac{df}{dx}.$$

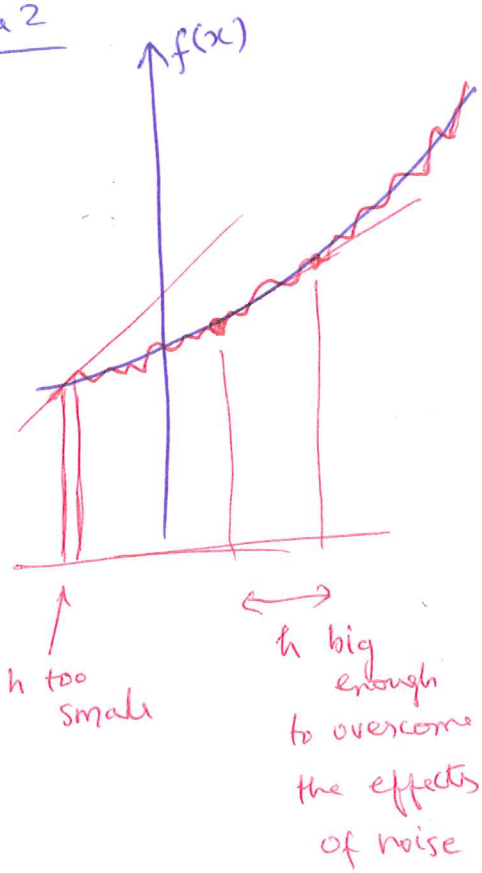
BUT almost always $f(x)$ can be evaluated only to finite accuracy / precision. This means that using a really small h can be bad.

Example 1.

Say $f(x)$ is accurate only to 5 digits and you take h so small that it changes f only in the 10^{th} decimal place

$\Rightarrow \frac{f(x+h) - f(x)}{h}$ is likely to not be very representation of $f'(x)$.

Example 2



Say $f(x)$ is smooth but its numerical approximation is noisy as shown. If h is taken too small, the derivative estimate is affected by the noise. (you are differentiating the noise)

h must be large enough to overcome the effects of noise.

(of course, h must not be so large that the derivative estimate is bad again)

So these examples suggest that h too small or too large is bad, so there might be an optimal range for h where the derivative estimate is most accurate.

Formally

Deriving an optimal h that (approximately) minimizes error in derivative estimate

Taylor series

$$f(x+h) = f(x) + hf'(x) + O(h^2).$$

To simplify analysis, let us write this as

$$f(x+h) \sim f(x) + hf'(x) + Ch^2 \quad \text{where } C \sim f''(x).$$

So, rearranging

$$hf'(x) \sim f(x+h) - f(x) + C_1 h^2. \qquad C_1 = -C.$$

$$f'(x) \sim \frac{f(x+h) - f(x)}{h} + C_1 h.$$

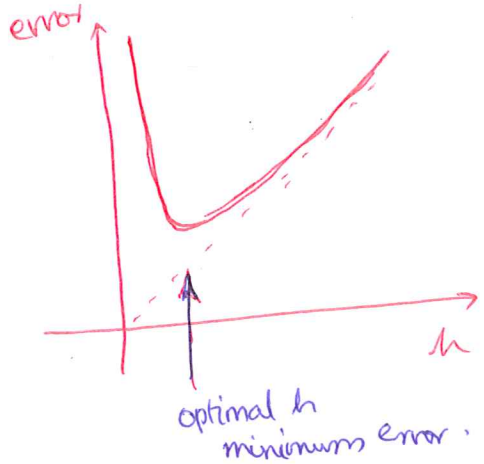
Say the error in evaluating $f(x)$ (or) $f(x+h)$ is approximately Δ in magnitude $(\pm\Delta)$. Then, the error in the evaluation of $\frac{f(x+h) - f(x)}{h}$ would be

of order $\frac{\Delta}{h}$.

So error in $f'(x)$ by approximating it with $\frac{f(x+h) - f(x)}{h}$

is:

$$E = \text{error in } f'(x) = \frac{\Delta}{h} + C_1 h.$$



To minimize E and find h_{opt}, we set $\frac{dE}{dh} = 0$

$$\frac{dE}{dh} = 0 = -\frac{\Delta}{h^2} + C_1$$

$$\Rightarrow \boxed{h_{opt} = \sqrt{\Delta / C_1}}$$

That is, the greater the error in function evaluation Δ, the greater should h be.

On the other hand, the greater C_1 – the rate of change of slope – the smaller should h be so that one does not average the slope (derivative) over too large a distance.

If one assumes $C_1 \sim 1$.

function accuracy		
$\Delta \sim 10^{-16}$ (machine precision)	$\Rightarrow$	$h_{opt} \sim 10^{-8}$
$\Delta \sim 10^{-8}$	$\Rightarrow$	$h_{opt} \sim 10^{-4}$
$\Delta \sim 10^{-4}$	$\Rightarrow$	$h_{opt} \sim 10^{-2}$

Again; if h is taken to be too small or too large, you may get bad derivative and Jacobian estimates, thereby (sometimes) resulting in incorrect conclusions about the system behavior (eg. stable or unstable).

Note also that $E(h_{opt}) = \Delta \sqrt{\frac{C_1}{\Delta}} + C_1 \sqrt{\frac{\Delta}{C_1}} = \sqrt{\Delta C_1}$

If $C_1 \sim 1$, $E(h_{opt})$ error in derivative estimate $\sim \sqrt{\Delta}$.

Thus the error in the derivative $\sqrt{\Delta}$ is much larger than the error in the function evaluations Δ. This is one of the big disadvantages of numerical differentiation

---

Numerical Jacobians.

Given $f = \begin{bmatrix} f_1(x_1, x_2 \cdots x_n) \\ f_2(x_1, x_2 \cdots x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots x_n) \end{bmatrix}$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_n}{\partial x_1} & \cdots & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Each partial derivative is approximated by a separate forward difference

$$\frac{df_i}{dx_j} \sim \frac{f_i(x_1, x_2, \cdots, x_i + h, \cdots x_n) - f(x_1 \cdots x_n)}{h}$$

change only x_i by h.

Finally, note that if you are finding the fixed point numerically (by using nonlinear equation solver) and the the Jacobian also numerically (by using forward or central difference), you should keep in mind that the corresponding eigenvalues are correspondingly approximate (but still pretty accurate).

If one or more of the eigenvalues are close to some stability boundary, the stability conclusions may change when the accuracy of the eigenvalue estimation is improved. (But this is rare).

Note 1: Finding the Jacobian of a Poincaré map is no different from finding the Jacobian of any other vector function.

Note 2: For all these calculations, trying to evaluate your function with the highest accuracy helps. For instance, if you are solving ODEs in MATLAB, you could set 'reltol' and 'abstol' to 10^{-10} or thereabouts.
