Activity 2

Numerical simulations

Goals

The nonlinear dynamical systems cannot be completely treated by analytical calculations. We need then to code programs to simulate these ones.

Theory

Simulations of the discrete time dynamical systems is obvious, since the time propagation consists simply to a loop reproducing the recurrence relation $X_{n+1} = \varphi(X_n)$. This is the same things for the symbolic dynamical systems where the simulation consists to a loop making act iteratively the generating rule onto the initial string/grid. But this is more complicated for the continuous time dynamical systems, where it needs to integrate a differential equation.

Algorithmic stability

We want to integrate an equation $\dot{X} = F(t, X(t))$ onto a time interval $[0, T]$. We consider a time discretisation $[0, T] \to \{t_0, t_1, ..., t_N\}$ with $t_i = i\Delta t$ and $\Delta t = \frac{T}{N}$ the discretisation step. We know the initial condition $X_0 \equiv X(0)$. We search an approximation X_n of $X(t_n)$. We call one step algorithm, a scheme with the following structure:

$$X_{n+1} = X_n + \mathfrak{F}(X_n, t_n)\Delta t$$

where $\mathfrak{F} : \Gamma \to \Gamma$ depends on F and on Δt.

Definition 14 (Consistent algorithm)**.** *An algorithm is said consistent of order $p \in \mathbb{N}$ if there exists a constant $C \in \mathbb{R}^+$ which does not depend of Δt and which is such that*

$$\left\| \frac{X(t_{n+1}) - X(t_n)}{\Delta t} - \mathfrak{F}(X(t_n), t_n) \right\| \leq C\Delta t^p \qquad \forall n$$

In other words, an algorithm is consistent of order p (we say also simply is of order Δt^p) if it generates at each step a bounded error of magnitude Δt^p.

Definition 15 (Algorithmic stability)**.** *An algorithm is said:*

- *unconditionally stable, if $\exists R \in \mathbb{R}^{+*}$ (dependent on T) and $\exists X_* \in \Gamma$ such that $\forall n$, $\forall \Delta t$ we have $\|X_n - X_*\| \leq R$.*
- *(conditionally) stable, if $\exists R \in \mathbb{R}^{+*}$ (dependent on T), $\exists X_* \in \Gamma$, ant $\exists \Delta t_{max} \in \mathbb{R}^{+*}$ such that $\forall n$, $\forall \Delta t \leq \Delta t_{max}$ we have $\|X_n - X_*\| \leq R$.*

- *stable with respect to the errors if for* $(\epsilon_n)_n$, $\exists K \in \mathbb{R}^{+*}$ *and* $\exists \Delta t_{max} \in \mathbb{R}^{+*}$ *such that* $\forall n$, $\forall \Delta t \leq \Delta t_{max}$ *we have* $\|X_n - Y_n\| \leq K\left(\|X_0 - Y_0\| + \sum_{m=0}^{n-1}\|\epsilon_m\|\right)$ *with* $Y_{n+1} = Y_n + \mathfrak{F}(Y_n, t_n)\Delta t + \epsilon_n$.

The unconditional stability is the stronger property since it induces that the algorithm converges for all step Δt (but this does not mean that it converges to a good approximation of $X(t)$!). An algorithm which is conditionally stable, converges only if the step is chosen sufficiently small (anew with a step just small enough, nothing ensures that the algorithm converges to a good approximation). The stability with respect to the errors ensures that the perturbations induced by machine errors do not induce divergence. This criteria can be tested numerically by manually setting errors $(\epsilon_n)_n$ larger than the machine errors.

Proposition 1. *A sufficient condition to an algorithm be stable with respect to the errors is* $\exists \Delta t_{max} \in \mathbb{R}^{+*}$, $\exists K \in \mathbb{R}^{+*}$ *such that* $\forall \Delta t \leq \Delta t_{max}$, $\forall t \in [0, T]$, $\mathfrak{F}$ *be* K*-Lipschitzian, i.e.* $\forall X, Y \in \Gamma$, *we have* $\|\mathfrak{F}(X,t) - \mathfrak{F}(Y,t)\| \leq K\|X - Y\|$.

Theorem 3 (Convergence)**.** *If an algorithm is stable with respect to the errors and consistent of order* p, *the there exists* $K \in \mathbb{R}^{+*}$ *such that* $\forall n$, $\|X_n - X(t_n)\| \leq K\Delta t^p$

In practice, we can test numerically the convergence of an algorithm by plotting the graph of the Cauchy criterion $\ln\left(\frac{1}{N}\sum_n \|X_n^{(q+1)} - X_n^{(q)}\|\right)$ with respect to q, where $X_n^{(q)}$ is the numerical solution computed with a step $\Delta t^{(q)} = \frac{\Delta t^{(0)}}{\lambda^q}$ for some λ (2, 10, or other). With a starting test step $\Delta t^{(0)}$ large, the curve decreases before a stabilisation at a Cauchy error corresponding of the higher precision reachable (this one is limited by the machine precision, the hardware architecture and the chosen algorithm). The algorithm is converged for values of Δt corresponding to this error floor.

Numerical integrators

There exists two kinds of numerical integrators (algorithms to integrate a differential equation):

- Finite difference integrators: these algorithms are based on the finite difference $\frac{X(t+\Delta t) - X(t)}{\Delta t}$ to approximate the derivative $\dot{X}(t)$. This kind of algorithm does not ensure the preservation of the measure (even if the system is conservative), and so it does not ensure that the constants of motion be well constant in the numerical approximation.
- Symplectic integrators: these algorithms can be applied onto systems described by a separable Hamiltonian $\mathscr{H}(q,p) = \mathscr{K}(q) + V(q)$ (not term mixing q and p).

We can note that the Hamilton equations can be rewritten as

$$\begin{aligned} \dot{q}^i &= \{q^i, \mathscr{H}\} \\ \dot{p}^i &= \{p^i, \mathscr{H}\} \end{aligned}$$

where $\{.,.\}$ is the Poisson bracket of Γ : $\{f,g\} = \sum_k \left(\frac{\partial f}{\partial q^k}\frac{\partial g}{\partial p^k} - \frac{\partial f}{\partial p^k}\frac{\partial g}{\partial q^k}\right)$. It follows that the Hamiltonian flow satisfies the following equation

$$\dot{\varphi}^t = \{\varphi^t, \mathscr{H}\} = L_{\mathscr{H}}\varphi^t$$

where $L_{\mathscr{H}} = \{., \mathscr{H}\}$ (partial Poisson bracket with $\mathscr{H}$) is called Poisson-Lie derivative with respect to $\mathscr{H}$. This equation is linear in the space of functions of Γ, we can then write the Hamiltonian flow as

$$\varphi^t = e^{tL_{\mathscr{H}}}$$

in the meaning of the differential operator exponential. Let $X = (q,p)$ we have then

$$\begin{aligned} & X(t_{n+1}) = \varphi^{\Delta t}(X(t_n)) \\ \iff & X_{n+1} = e^{\Delta t L_{\mathscr{H}}} X_n \end{aligned}$$

But due to the separability of the Hamiltonian, we have:

$$L^2_{\mathscr{H}} X = \{\{X, \mathscr{H}\}, \mathscr{H}\} = \{\dot{q}, \mathscr{H}\} = 0$$

and

$$L^2_V X = \{\{X, V\}, V\} = \{\dot{p}, V\} = 0$$

This induces that $\forall \alpha, \beta \in \mathbb{R}$

$$e^{\alpha L_{\mathscr{H}}} X = X + \alpha L_{\mathscr{H}} X \text{ and } e^{\beta L_V} X = X + \beta L_V X$$

So, if we split $e^{\Delta t L_{\mathscr{H}}}$ into operators $e^{\alpha L_{\mathscr{H}}}$ and $e^{\beta L_V}$, the computation becomes easy because $L_{\mathscr{H}} X = (\frac{\partial \mathscr{H}}{\partial p}, 0)$ and $L_V X = (0, -\frac{\partial V}{\partial q})$. But $[L_{\mathscr{H}}, L_V] \neq 0$, and so the splitting must be realised only with an approximation using the Trotter formula:

$$e^{A+B} = \lim_{n \to +\infty} \left(e^{\frac{A}{n}} e^{\frac{B}{n}} \right)^n$$

or the Baker-Campbell-Hausdorff (BCH) formula

$$e^A e^B = e^{A+B+\frac{1}{2}[A,B]+\frac{1}{12}[A,[A,B]]-\frac{1}{12}[B,[A,B]]-\frac{1}{24}[B,[A,[A,B]]]+\ldots}$$

If k is the order of approximation in the splitting of the exponential, the symplectic integrator is of order Δt^k.
In contrast with the finite difference integrators, the symplectic integrators ensure the preservation of the measure (the constant of motion are well constant in the numerical approximation). They are unconditionally stable.

Algorithms

Finite difference integrators

Explicit Euler algorithm

$$\dot{X} = \lim_{h \to 0} \frac{X(t+h) - X(t)}{h} \simeq \frac{X_{n+1} - X_n}{\Delta t}$$

We have then the following scheme:

$$X_{n+1} = X_n + F(t_n, X_n)\Delta t$$

The error is of magnitude Δt, the algorithm is conditionally stable.

Runge-Kutta of order 2 (RK2) algorithm

$$\dot{X} = \lim_{h \to 0} \frac{X(t+h) - X(t+\frac{h}{2})}{\frac{h}{2}} \simeq 2\frac{X_{n+1} - X_{n+\frac{1}{2}}}{\Delta t} \Rightarrow X_{n+1} = X_{n+\frac{1}{2}} + \frac{\Delta t}{2} F(t_n + \frac{\Delta t}{2}, X_{n+\frac{1}{2}})$$

$$\dot{X} = \lim_{h \to 0} \frac{X(t+\frac{h}{2}) - X(t)}{\frac{h}{2}} \simeq 2\frac{X_{n+\frac{1}{2}} - X_n}{\Delta t} \Rightarrow X_{n+\frac{1}{2}} = X_n + \frac{\Delta t}{2} F(t_n + \frac{\Delta t}{2}, X_{n+\frac{1}{2}})$$

and then

$$X_{n+1} = X_n + F(t_n + \frac{\Delta t}{2}, X_{n+\frac{1}{2}})\Delta t$$

We obtain an estimation of $X_{n+\frac{1}{2}}$ by the Euler method. We have the the following scheme:

$$X_{n+1} = X_n + F\left(t_n + \frac{\Delta t}{2}, X_n + F(t_n, X_n)\frac{\Delta t}{2}\right)\Delta t$$

The error is of magnitude Δt^2, the algorithm is conditionally stable.

Runge-Kutta of order 4 (RK4) algorithm

$$X_{n+1} = X_n + (K_1 + 2K_2 + 2K_3 + K_4)\frac{\Delta t}{6}$$

with

$$\begin{aligned} K_1 &= F(X_n, t_n) \\ K_2 &= F\left(X_n + K_1\frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right) \\ K_3 &= F\left(X_n + K_2\frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right) \\ K_4 &= F\left(X_n + K_3\Delta t, t_n + \Delta t\right) \end{aligned}$$

The error is of magnitude Δt^4, the algorithm is conditionally stable.

Symplectic integrators

Symplectic Euler algorithm

BCH or Trotter formula at the first order of approximation:

$$e^{\Delta t L_{\mathscr{H}}} \simeq e^{\Delta t L_{\mathscr{K}}} e^{\Delta t L_V}$$

We have then the following scheme:

$$p^i_{n+1} = p^i_n - \left.\frac{\partial V}{\partial q^i}\right|_{q=q_n} \Delta t$$

$$q^i_{n+1} = q^i_n + \left.\frac{\partial \mathscr{K}}{\partial p^i}\right|_{p=p_{n+1}} \Delta t$$

When we can write $\frac{\partial \mathscr{K}}{\partial p^i} = \frac{p^i}{m} = v^i$ and $-\frac{\partial V}{\partial q^i} = \frac{a^i}{m}$ (with v^i and a^i the speed and the acceleration), we have

$$v^i_{n+1} = v^i_n + a^i_n \Delta t$$
$$q^i_{n+1} = q^i_n + v^i_{n+1} \Delta t$$

Verlet algorithm

BCH formula at the second order of approximation:

$$e^{\Delta t L_{\mathscr{H}}} \simeq e^{\frac{\Delta t}{2} L_V} e^{\Delta t L_{\mathscr{K}}} e^{\frac{\Delta t}{2} L_V}$$

because $e^A e^B e^A = e^A e^{A+B+\frac{1}{2}[B,A]+\dots} = e^{2A+B+\frac{1}{2}[B,A]+\frac{1}{2}[A,A+B]+\dots} = e^{2A+B+\dots}$. We have then the following scheme:

$$q^i_{n+1} = q^i_n + \left.\frac{\partial \mathscr{K}}{\partial p^i}\right|_{p=p_n} \Delta t - \sum_k \left.\frac{\partial^2 \mathscr{K}}{\partial p^i \partial p^k}\frac{\partial V}{\partial q^k}\right|_{(q,p)=q_n,p_n} \frac{\Delta t^2}{2}$$

$$p^i_{n+1} = p^i_n - \frac{1}{2}\left(\left.\frac{\partial V}{\partial q^i}\right|_{q=q_n} + \left.\frac{\partial V}{\partial q^i}\right|_{q=q_{n+1}}\right)\Delta t$$

where we have used a first order Taylor expansion to compute $\left.\frac{\partial \mathscr{K}}{\partial p^i}\right|_{p=p_n-\frac{\partial V}{\partial q}(q_n)\frac{\Delta t}{2}}$.

When we can write $\frac{\partial \mathscr{K}}{\partial p^i}=\frac{p^i}{m}=v^i$ and $-\frac{\partial V}{\partial q^i}=\frac{a^i}{m}$ (with v^i and a^i the speed and the acceleration), we have

$$q^i_{n+1}=q^i_n+v^i_n\Delta t+a^i_n\frac{\Delta t^2}{2}$$

$$v^i_{n+1}=v^i_n+\frac{a^i_n+a^i_{n+1}}{2}\Delta t$$

Leapfrog algorithm

Same as the Verlet algorithm but where we use the following Taylor expansions $p^i(t_n\pm\frac{\Delta t}{2})=p^i(t_n)\pm\dot{p}^i(t_n)\frac{\Delta t}{2}+\mathcal{O}(\Delta t^2)$ to compute $p^i_{n\pm1/2}=p^i_n\mp\left.\frac{\partial V}{\partial q^i}\right|_{q=q_n}\frac{\Delta t}{2}$ and to obtain $\left.\frac{\partial \mathscr{K}}{\partial p^i}\right|_{p=p_n-\frac{\partial V}{\partial q}(q_n)\frac{\Delta t}{2}}=\left.\frac{\partial \mathscr{K}}{\partial p^i}\right|_{p=p_{n+1/2}}$. We have then the following scheme:

$$q^i_{n+1}=q^i_n+\left.\frac{\partial \mathscr{K}}{\partial p^i}\right|_{p=p_{n+1/2}}\Delta t$$

$$p^i_{n+1/2}=p^i_{n-1/2}-\left.\frac{\partial V}{\partial q^i}\right|_{q=q_n}\Delta t$$

When we can write $\frac{\partial \mathscr{K}}{\partial p^i}=\frac{p^i}{m}=v^i$ and $-\frac{\partial V}{\partial q^i}=\frac{a^i}{m}$ (with v^i and a^i the speed and the acceleration), we have

$$q^i_{n+1}=q^i_n+v^i_{n+1/2}\Delta t$$

$$v^i_{n+1/2}=v^i_{n-1/2}+a^i_n\Delta t$$

To initialise the leapfrog, we start with $p^i_{1/2}=p^i_0-\left.\frac{\partial V}{\partial q^i}\right|_{q=q_0}\frac{\Delta t}{2}$.

Work to be done

1.
   - If your chosen system is discrete in time or symbolic, code a program to simulate the iterations of your system for any initial condition.
   - If your chosen system is continuous in time:
     a. Find if your differential equation can be reformulated as Hamilton equations for some Hamiltonian $\mathscr{H}(q,p)$.
     b. If you have found an Hamiltonian, find if this one (maybe up to a change of variables) is separable.
     c. If your system can be represented by a separable Hamiltonian, choose a symplectic algorithm, else choose a finite difference algorithm, and code this one to simulate the time evolution for any initial condition.
     d. If you use a finite difference algorithm, study (by computing the Cauchy criterion) the convergence by changing the time step Δt. Find if your system presents some constants of motion and compute these ones at each time.

2. Code functions to draw orbits of your system in the phase space Γ (or to draw the tree of growth of the word for 1D symbolic systems, or to draw the evolution of the grid for 2D symbolic systems).

3. If your chosen system is discrete in time, code functions to draw (in a plane of the phase space Γ), on the same graph, several orbits (with different colours) with random initial conditions (the plot of the phase portrait of your system).