Numerical simulations of time-dependent problems applied to epidemiology, artificial intelligence, and finance

Gabriel Turinici

Paris, March 25, 2024

Contents

Chapter 1

Motivations and Examples: Epidemiology, Finance, deep learning

The object of this book is on one hand the presentation of numerical algorithms for finding the solutions to time depending problems and on the other hand the computation of derivatives in a computational graph ; we will see that the two share some important features and in particular we will apply these techniques to the control of evolution equations and also to techniques in statistical (deep) learning. We begin with some examples of applications.

1.1 Ordinary Differential Equations (ODE)

An important model in epidemiological modeling is the SIR model; the initials denote S for the group of 'susceptible' individuals, I for the group of infected individuals, and R for the group of those recovered, see Figure 1.1 for a graphical representation.

After a derivation (which will be presented later in Section

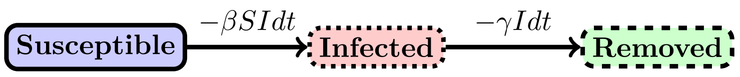


Figure 1.1: Schematic representation of the SIR model in equation (1.1).

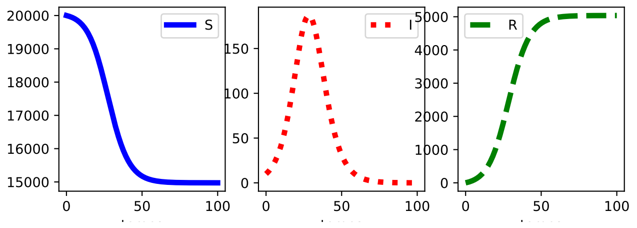


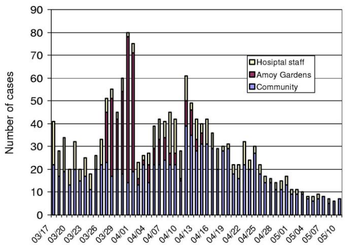
Figure 1.2: Typical evolution of the system in Equation (1.1); data taken from [5].

2.9), we obtain the system of equations, called the SIR model

$$\begin{cases} \frac{dS}{dt} = -\frac{\beta SI}{N} \\ \frac{dI}{dt} = \frac{\beta SI}{N} - \gamma I \\ \frac{dR}{dt} = \gamma I. \end{cases} \tag{1.1}$$

We assume $S(0) = S_0 \neq 0,\ I(0) = I_0 > 0,\ R(0) = R_0 \geq 0$, $S_0 + I_0 + R_0 = N$, N is the total population. Here, β, γ are parameters of the model. A typical evolution is given in Figure 1.2.

In reality, the model needs to be adapted, as real data is not always compatible with simple models, see Figure 1.3. Therefore, we move beyond the domain of models with analytical solutions and must find accurate numerical approximations of their solutions.



$$
\begin{aligned}
dS/dt &= -rS(t)I(t) - r_pS(t)I_p(t)\\
dE/dt &= rS(t)I(t) - bE(t)\\
dI/dt &= bE(t) - aI(t)\\
dR/dt &= aI(t)\\
dI_p/dt &= r_pS(t)I_p(t) - a_pI_p(t)\\
dR_p/dt &= a_pI_p(t).
\end{aligned}
$$

Figure 1.3: Actual evolution of the number of infected individuals; image taken from [5]. To accurately reproduce real data, it is necessary to use a model like the one on the right.

1.2 Stochastic Differential Equations (SDE)

In financial applications (calculations for derivative products in different scenarios) or in physics (path integrals, etc.), there is a need to handle quantities that evolve over time and also contain an element of uncertainty. For instance, the yield $\frac{S_{t+\Delta t}-S_t}{S_t}$ of a financial asset contains a predictable part and another random part, which can be modeled, like in the Black-Scholes model, as a normal variable $\mathcal{N}(\mu\Delta t, \sigma^2\Delta t)$[1]. The following stochastic differential equation (SDE) is obtained (see [7] for details):

$$dS_t = \mu S_t dt + \sigma S_t dW_t. \tag{1.2}$$

An illustration of solution scenarios for (1.2) is provided in Figure 1.4.

Reminder: derivative products are financial instruments whose value depends (according to a pre-established contract) on an underlying asset. Example: a European call option on S_t with a final value of $(S_T - K)_+$. However, the calculation of the value before expiration is unknown. Models need to be imposed, and quantities such as:

$$\mathbb{E}^{\mathbb{Q}}[e^{-r(T-t)}(S_T - K)_+ \,|(S_u)_{u\leq t}], \tag{1.3}$$

[1]We do not discuss the justification of the validity of this model here; for real-life applications, this justification must be carefully validated!!

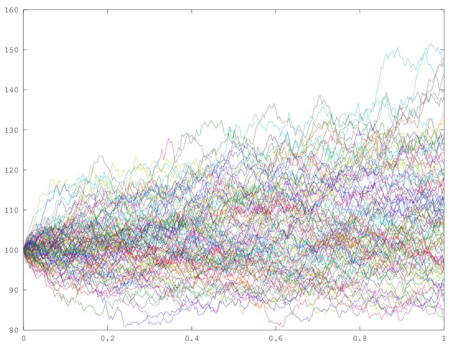


Figure 1.4: Solution scenarios for (1.2).

need to be calculated. As a reminder, S_t follows an SDE; the goal is to calculate solutions, study the accuracy of numerical calculations, determine if precise scenario calculations are desired (strong convergence) or only averages (weak convergence), etc...

1.3 Computating the derivativ in a computational graph and control of evolution equations

The goal is to influence the evolution of a system by acting on various parameters called "controls". The same approach helps us study the sensitivity of a result (obtained from solving an evolution equation) with respect to input parameters; an example can be constructed from section 1.1 if we want to khow how $S(\infty)$ of (1.1) depends on β.

In general, whenever a result is obtained using sequential calculations on a computational graph, the derivative of the result with respect to the inputs can be calculated. This is called backpropagation; in control theory, this gives rise to "adjoint states".

Example (adapted from [4], also see [2, chap 6.5]): $f =$

$5 \cdot (x + y \cdot z)$. The graph has inputs x, y, z, and output $f = f(x, y, z)$. To calculate $\partial_x f$, $\partial_y f$, $\partial_z f$, write it as a computational graph (direct or "forward" calculation):

$u = y \times z$
$v = x + u$
$f = 5 \times v$

Let $x = 1$, $y = 2$, $z = 3$; here are the relations obtained by elementary derivation of each calculation (adjoint or "backward" calculation):

$\partial_v f = 5$
$\partial_x f = \partial_v f \times \partial_x v = \partial_v f = 5$
$\partial_u f = \partial_v f \times \partial_u v = \partial_v f = 5$
$\partial_y f = \partial_u f \times \partial_y u = 5z = 15$
$\partial_z f = \partial_u f \times \partial_z u = 5y = 10.$

We will study the relationship between the derivative and the computational graph and observe the emergence of an auxiliary variable called the adjoint state. It is crucial for the formalization of the calculation and allows the treatment of complex situations (cf. Figure 1.5 for the "Inception" network [6]).

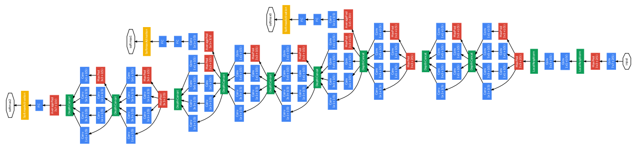


Figure 1.5: "Inception" network architecture [6]. Each cell represents a multi-variable calculation such as matrix-vector multiplication followed by a non-linear operation like taking the positive part on each component.

Chapter 2

Ordinary Differential Equations (ODE)

Let I be an open interval included in $\mathbb{R}_+$. Consider the following ordinary differential equation (ODE):

$$\frac{dX}{dt} = f(t, X(t)), \quad X(t_0) = X_0, \tag{2.1}$$

with the integral form

$$X(t) = X(t_0) + \int_{t_0}^{t} f(s, X(s))ds. \tag{2.2}$$

2.1 Existence and Uniqueness of the Solution

To show the existence and uniqueness of the solution to the previous ODE, we use the following two theorems:

Theorem 2.1 (Local Lipschitz variant of Cauchy-Lipschitz). *Let $f : I \times \mathbb{R} \to \mathbb{R}$ be a locally Lipschitz continuous function at $X_0 \in \mathbb{R}$, $t_0 \in I$. In other words, there exist two balls $B_x(X_0, R_x)$, $B_t(t_0, R_t)$ and a constant $L > 0$ such that $\forall t \in$*

$B_t(t_0, R_t)$, $\forall X_1, X_2 \in B_x(X_0, R_x)$:

$$|f(t, X_1) - f(t, X_2)| \leq L|X_1 - X_2|.$$

Then there exists $\varepsilon > 0$ such that the Cauchy problem (2.1) has a unique local solution: $X(t) : (t_0 - \varepsilon, t_0 + \varepsilon) \subset I \to \mathbb{R}$. Moreover, $X(\cdot)$ is a C^1 function.

Theorem 2.2 (Global Lipschitz variant of Cauchy-Lipschitz)**.** *Under the same assumptions as in Theorem (2.1), if L is the same for all R_x (radius of the ball) and initial condition X_0, then a global solution exists and is unique.*

> **Remark 2.3.** *Global existence also holds if we can find a continuous function $\alpha : \mathbb{R} \to \mathbb{R}_+$ such that*
>
> $$|f(t, X_1) - f(t, X_2)| \leq \alpha(t)|X_1 - X_2|.$$
>
> *This allows L to depend on time.*

Example 2.4 (linear function)**.** *Let $f(t, X) = rX$ with $r \in \mathbb{R}$ constant. Then $|f(t, X_1) - f(t, X_2)| = |r| \cdot |X_1 - X_2|$, so we obtain global existence with $L = |r|$.*

Example 2.5 (non-linear function that blows up)**.** *Let $f(t, X) = \frac{5}{X-3}$. An immediate calculation gives $|f(t, X_1) - f(t, X_2)| = \frac{5}{|(X_1-3)(X_2-3)|} \cdot |X_1 - X_2|$, so we obtain local existence for $L = \sup_{X_1, X_2 \in \mathcal{V}} \frac{5}{|(X_1-3)(X_2-3)|}$ in an open neighbourhood $\mathcal{V}$ of any point $X_0 \neq 3$ (such that $3 \notin \overline{\mathcal{V}}$). However, as $\frac{5}{|(X_1-3)(X_2-3)|}$ is not bounded around $X_0 = 3$, the global existence theorem is not applicable at $X_0 = 3$.*

2.2 Numerical Methods

2.2.1 Important notations

If the solution to the Cauchy problem exists, it is unique (cf. Theorem 2.1). To numerically find the solution, it is approximated using different methods. The approximation is done, for example, on $[0, T]$ with N points. We introduce some important notations used throughout this chapter :

1. the equation to solve is (2.1);
2. $h = T/N$ is called the "time step"; we denote $t_n = n \cdot h$, $\forall n \leq N$;
3. $X_n = X(t_n)$ is the exact solution; in general we do not have a analytic formula so X_n will remain unknown;
4. U_n will be an approximation of X_n; this is the main object that we search
5. $f_n = f(t_n, U_n)$.

2.2.2 One step methods

So the main question is how to calculate the U_n? For example, starting from the following formula:

$$X(t_{n+1}) = X(t_n) + \int_{t_n}^{t_{n+1}} f(s, X(s))ds. \tag{2.3}$$

one could imagine a recurrence that is called **a one-step method** given by :

$$U_{n+1} = U_n + h\phi(t_n, U_n, f_n, h). \tag{2.4}$$

Each function ϕ gives another numerical method. Note that ϕ can also depend on U_{n+1} or f_{n+1}; in this case, we speak of implicit methods. A "numerical scheme" is a procedure to solve the ODE. It is formulated **independent** of the function f that actually defines the ODE. We do not look for numerical schemes that only work for particular ODEs but for schemes that work for a large class of functions f. So in general the regularity of f is not an important issue (see nevertheless the "stiff" numerical schemes latter) so we can suppose f at least C^1.

2.2.3 Definition of 4 classic numerical schemes

We describe below four important numerical schemes. To illustrate their use we take two particular cases for function f namely $f_1(t,X) = rX$ and $f_2(t,X) = rX^2$. We recall that for f_1, the solution $X(t)$ of $\dot{X}(t) = f_1(t, X(t))$ is $X(t) = e^{rt}X_0$, while for f_2, the solution $Y(t)$ of $\dot{Y}(t) = f_2(t, Y(t))$ is $Y(t) = \frac{Y_0}{1-rtY_0}$.

- **Explicit Euler (denoted EE from now on):**

$$\begin{cases} U_{n+1} &= U_n + hf(t_n, U_n) = U_n + hf_n \\ U(0) &= X(0) \end{cases} \tag{2.5}$$

Here, $\phi = f_n$. Examples: for $f_1 : U_{n+1} = U_n + hrU_n = (1+rh)U_n$; for f_2: $U_{n+1} = U_n + hrU_n^2 = (1+rhU_n)U_n$.

- **Implicit Euler (denoted IE from now on):**

$$\begin{cases} U_{n+1} &= U_n + hf_{n+1} \\ U(0) &= X(0) \end{cases} \tag{2.6}$$

Here, $\phi = f_{n+1}$. Examples: for $f_1 : U_{n+1} = U_n + hrU_{n+1}$ so $U_{n+1} = \frac{U_n}{1-rh}$; for f_2: $U_{n+1} = U_n + hrU_{n+1}^2$ so U_{n+1} is a solution of $rhU_{n+1}^2 - U_{n+1} + U_n = 0$.

When f is Lipschitz, for sufficiently small h, the value U_{n+1}, a solution of the implicit IE scheme definition equation, is unique, see exercise 2.5 page 39.

- **Crank-Nicolson (denoted CN from now on, implicit):**

$$\begin{cases} U_{n+1} &= U_n + h\Big[\dfrac{f_n + f_{n+1}}{2}\Big] \\ U(0) &= X(0) \end{cases} \tag{2.7}$$

Examples: for $f_1 : U_{n+1} = U_n + hr\frac{U_n+U_{n+1}}{2}$ so $U_{n+1} = \frac{1+\frac{rh}{2}}{1-\frac{rh}{2}}U_n$; for f_2: $U_{n+1} = U_n + hr\frac{U_n^2+U_{n+1}^2}{2}$ so U_{n+1} is a solution of $\frac{rh}{2}U_{n+1}^2 - U_{n+1} + (1+\frac{rh}{2}U_n)U_n = 0$.

- **Heun (denoted H from now on, explicit):**

$$\begin{cases} U_{n+1} &= U_n + \frac{h}{2}\Big[f_n + f(t_{n+1}, U_n + hf_n)\Big] \\ U(0) &= X(0) \end{cases} \tag{2.8}$$

Examples: for f_1 : $U_{n+1} = U_n + \frac{h}{2}[rU_n + r(U_n + hrU_n)]$; for f_2: $U_{n+1} = U_n + \frac{h}{2}[rU_n^2 + r(U_n + hrU_n^2)^2]$.

Intuition 2.6. *The relation* (2.3) *indicates that we need to find a way to approximately calculate the integral of $f(t, X(t))$ between t_n and $t_{n+1} = t_n + h$. The EE scheme takes an approximation using the method of rectangles by using the value at t_n, the IE scheme uses the value at $t_n + h$, and the CN scheme takes the average of the two, i.e., it uses the trapezoidal rule. As for the Heun scheme, it uses an approximation of X_{n+1} that it reintroduces into a CN-type scheme but with the idea of keeping it explicit.*

2.3 Error, Consistency, and Order

2.3.1 Error

When introducing the exact solution into the formula (2.4) for one-step methods, we obtain "truncation errors" $\tau_{n+1}(h)$:

$$\underbrace{X(t_{n+1}) = X(t_n) + h\phi(t_n, X_n, f(t_n, X_n), h)}_{\text{true for the numerical scheme, i.e., } U_n \text{ instead of } X_n \text{, etc.}} + h\tau_{n+1}(h).$$

or

$$\begin{aligned} \tau_{n+1}(h) &:= \frac{X(t_{n+1}) - X(t_n) - h\phi(t_n, X_n, f(t_n, X_n), h)}{h} \\ &= \frac{X(t_{n+1}) - X(t_n)}{h} - \phi(t_n, X_n, f, h). \end{aligned} \tag{2.9}$$

Definition 2.7. *The remainder that appears when the true solution is placed into the relation defining the numerical* *scheme (similar in form to the initial equation) is called the truncation error. For one-step methods* (2.4)*, it is* $\tau_{n+1}(h)$ *defined in* (2.9)*, which is called the local truncation error at step* $n+1$*. The global truncation error is defined by the relation:* $\tau(h) = \max_{n=1,\dots,N} |\tau_n(h)|$*.*

Remark 2.8. *The truncation error here is the same as the error (divided by h) between* X_{n+1} *and the* U^*_{n+1} *obtained starting from* $U_n = X_n$*.*

Example 2.9. Explicit Euler: *Using the Taylor series formula to the 2nd order:*

$$X(t+h) = X(t) + h\dot{X}(t) + \frac{1}{2}h^2\ddot{X}(\xi),\ \xi \in [t, t+h]$$

For ($t = t_n$ *and* $t_n + h = t_{n+1}$)*, we get:* $\tau_{n+1}(h) = \frac{1}{2}h\ddot{X}(\xi_n)$*.*
Implicit Euler: *: similar computations but also check the technique 2.14 and the exercise 2.5 page 39.*

2.3.2 Consistency and Order

Definition 2.10. *A scheme is said to be consistent if:*

$$\lim_{h\to 0} \tau(h) = 0, \tag{2.10}$$

i.e., for small h, the exact solution satisfies the scheme.
A scheme is of order "p" if: $\tau(h) = O(h^p)$ *for* $h \to 0$*.*

2.4 Stability and Convergence

2.4.1 Zero-Stability

To study stability with respect to perturbations, we check if Z_n^h defined by:

$$\begin{aligned} Z_{n+1}^{(h)} &= Z_n^{(h)} + h[\phi(t_n, Z_n^{(h)}, f(t_n, Z_n^{(h)}), h) + \delta_{n+1}] \\ Z_0^{(h)} &= \delta_0 + X_0, \end{aligned} \tag{2.11}$$

is close to U_{n+1}.

To know more 2.11. *Numerical inaccuracies do not appear during an addition but mostly in the computation, often complex, of the function ϕ; that's why the perturbations δ_n are placed where indicated in the formula (2.11). For example, if $f(t,X) = X^2 - 1$ needs to be calculated at $t = 0, X = \sqrt{2}$, the value $\sqrt{2}^2 - 1 = 1$ is often affected by errors. Python calculation example:*

```
In [2]: numpy.sqrt(2)**2 -1
Out[2]: 1.0000000000000004
```

Definition 2.12. *The scheme given by ϕ is called zero-stable if there exists h_0 and a constant C (independent of ε) such that if $h \leq h_0$ and $|\delta_n| < \varepsilon$ $(\forall n)$: then*

$$|Z_{n+1}^{(h)} - U_{n+1}| \leq C\varepsilon, \;\; \forall n \geq 0. \tag{2.12}$$

Theorem 2.13. *Assuming f and ϕ are Lipschitz with respect to their second variable, meaning that there exist $\Lambda > 0, h_0 > 0$ such that $\forall h < h_0$ such that :*

$$|\phi(t, X, f(t,X), h) - \phi(t, Y, f(t,Y), h)| < \Lambda|X - Y|, \forall X, Y.$$

Then the numerical scheme given by ϕ is zero-stable.

Proof. Let's denote: $W_n = Z_n^{(h)} - U_n$. Then

$$W_{n+1} = Z_n^{(h)} - U_n + h[\phi(t_n, Z_n^{(h)}, f, h) - \phi(t_n, U_n, f, h)] + h\delta_{n+1}$$

so $|W_{n+1}| \leq |W_n| + h\Lambda|W_n| + h|\delta_{n+1}|$ thus by summing these inequalities and simplifying terms:

$$|W_{n+1}| \leq |W_0| + h\Lambda\sum_{s=0}^{n}|W_s| + \sum_{s=1}^{n+1}h|\delta_s|.$$

This allows us to conclude using the discrete Gronwall's lemma (see exercise 2.3 page 38):

$$|W_{n+1}| \leq |W_0| + h\Lambda\exp(h\Lambda n) \leq (1+T)\varepsilon\exp(\Lambda T).$$

□

Important technique 2.14. *Question: which methods among EE, IE, CN, H satisfy the assumptions of theorem 2.13 ?*
- *EE: $\phi = f_n$, Lipschitz when f is.*
- *H: similar techniques*
- *For general implicit methods, see exercise 2.5 page 39.*
- *Intuition for IE: by definition, ϕ has the property: $\phi(t_n, U_n, f_n, h) = f(t_{n+1}, U_{n+1})$ (assuming the existence of a unique solution). Then for two initial points U_n, V_n, we need to bound $f(t_{n+1}, U_{n+1}) - f(t_{n+1}, V_{n+1})$: $|f(t_{n+1}, U_{n+1}) - f(t_{n+1}, V_{n+1})| \leq L|U_{n+1} - V_{n+1}|$ and $|U_{n+1} - V_{n+1}| \leq |U_n - V_n| + h|f(t_{n+1}, U_{n+1}) - f(t_{n+1}, V_{n+1})| \leq |U_n - V_n| + hL|U_{n+1} - V_{n+1}|$ thus $|U_{n+1} - V_{n+1}| \leq |U_n - V_n|/(1 - hL)$...*
- *CN: similar techniques*

2.4.2 Convergence

Definition 2.15. *A scheme is said to be convergent of order p if, with the previous notations, $|U_n - X_n| = O(h^p)$. A scheme convergent of order 1 is simply called "convergent."*

Theorem 2.16. *Under the same assumptions as in Theorem 2.13, we have:*

$$|U_n - X_n| \leq (|U_0 - X_0| + nh\tau(h)) \exp(\lambda nh).$$

In particular, if for $p \geq 1$: $|U_0 - X_0| = O(h^p)$ and $\tau(h) = O(h^p)$, then $|U_n - X_n| = O(h^p)$ (the scheme converges of order p).

Proof. We follow the same steps as in the proof of Theorem 2.13, with $\delta_j = \tau_j(h)$ (using the discrete Gronwall's lemma). Here, the exact solution X_n plays the role of the perturbation $Z_n^{(h)}$. □

To know more 2.17. *The previous theorem can be rewritten as stating that* ***consistency and stability imply convergence****. This is a principle often encountered.*

Corrolary 2.18. *The schemes EE and IE converge of order 1. The schemes CN and H converge of order 2.*

Proof. I will provide detailed reasoning only for the Crank-Nicholson scheme:

$$X_{n+1} = X_n + \frac{h}{2}\Big[f(t_n, X_n) + f(t_{n+1}, X_{n+1})\Big] + h\tau_{n+1}(h) \tag{2.13}$$

so (for now, treating it as if it were explicit, see exercise 2.5 for details):

$$X_{n+1} = X_n + \frac{h}{2}\Big[X'_n + X'_{n+1}\Big] + h\tau_{n+1}(h). \tag{2.14}$$

Also, the Taylor series at order 2 for X' and at order 3 for X provide:

$$X'_{n+1} = X'_n + hX''_n + \frac{h^2}{2}X_n^{(3)}(\eta) \tag{2.15}$$

$$X_{n+1} = X_n + hX'_n + \frac{h^2}{2}X''_n + \frac{h^3}{6}X_n^{(3)}(\xi) \tag{2.16}$$

Replacing (2.15) and (2.16) into (2.14) gives:

$$h\tau_{n+1}(h) = \frac{h^3}{6}X_n^{(3)}(\xi) - \frac{h^3}{4}X_n^{(3)}(\eta) \tag{2.17}$$

which leads to $\tau_{n+1}(h) = O(h^2)$ (after some calculations to transfer the implicit version into an explicit one). □

To know more 2.19. *So, we have several methods, each with its convergence order. Which one to choose then? A naive answer would be to pick the scheme with* *the highest order. However, as we saw with CN, for the order to be effective, we need to use higher derivatives of f (meaning f needs to be smooth), and on the other hand, the higher the order of the scheme, the more intermediate calculations of the f function are required (EE/IE have only one f calculation, while CN uses two), which can be costly. In practice, one rarely goes beyond order* 4 *or* 5 *(and sometimes sticks to order* 1*).*

2.4.3 Absolute Stability

Here, stability is considered from the perspective of the solution over a long time $T = Nh$ (as $T \to \infty$), but for a fixed step size h (so $N \to \infty$). For $\lambda \in \mathbb{C}$, $t \geq 0$, we consider the test problem:

$$\dot{Y}(t) = \lambda Y(t) \tag{2.18}$$
$$Y(0) = 1 \tag{2.19}$$

with the solution $Y(t) = e^{\lambda t}$. For $Re(\lambda) < 0$, we obtain $\lim\limits_{t\to+\infty} Y(t) = 0$. So, any local perturbation in time is "erased" in the long run. This is a very desirable property for numerical schemes that have to combat rounding errors, etc. We want to preserve this property.

Definition 2.20. *A scheme is said to be absolutely stable if, for $f(t,x) = \lambda x$ and $\forall h, \lambda$, $U_n \to 0$. Otherwise, its region of absolute stability is:*

$$\{h\lambda \in \mathbb{C} | U_n \to 0\}.$$

To know more 2.21. *Choosing f to be linear is not so surprising because at first order $f(t,x) \simeq f(t,X_0) + \frac{\partial f}{\partial x}(t,X_0)(x - X_0)$ so, apart from a constant, we have a linear function.*

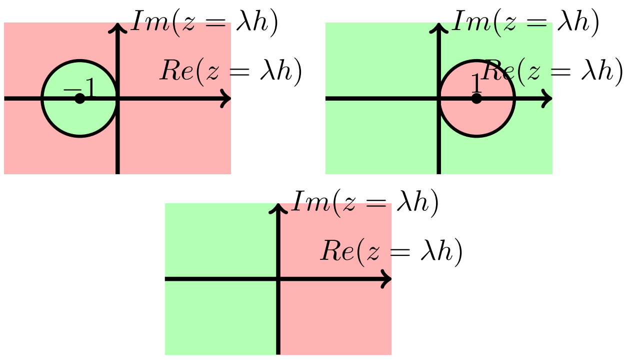


Figure 2.1: The stability of Explicit Euler (top left), Implicit Euler (top right), and Crank-Nicholson (bottom): in green stability region, in red instability region.

Example 2.22 (Explicit Euler). $U_n = (1 + h\lambda)^n U_0$. *We define the stability region by imposing the stability condition:* $|1 + h\lambda| < 1$. *This corresponds to the interior of B((-1,0),1), see figure* (2.1) *for an illustration.*

Example 2.23 (Implicit Euler). $U_{n+1} = \dfrac{U_n}{1 - h\lambda} = \dfrac{U_0}{(1 - h\lambda)^{n+1}}$. *To find the stability region, limit the time step h by imposing the stability condition:*

$$|1 - h\lambda| > 1.$$

So, here it is the exterior of B((1,0),1), see figure 2.1 for an illustration.

Example 2.24 (Crank-Nicholson).

$$U_{n+1} = \left(\frac{1 + \dfrac{h\lambda}{2}}{1 - \dfrac{h\lambda}{2}} \right)^{n+1} U_0. \tag{2.20}$$

The stability region is defined by imposing the stability condition:

$$\left\{ z = h\lambda \in \mathbb{C} : \left| \frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}} \right| < 1 \right\} = \{ z \in \mathbb{C} : Re(z) < 0 \},$$

see figure 2.1 for an illustration.

Example 2.25 (Heun). *The stability region is (after calculations)* $\{z \in \mathbb{C} : |1 + z + \frac{z^2}{2}| < 1\}$.

2.5 Higher-Order Methods: Runge-Kutta

These are methods that evaluate the function at intermediate steps:

$$U_{n+1} = U_n + hF(t_n, U_n; h, f). \tag{2.21}$$

with the function F of the scheme defined by

$$F(t_n, U_n; h, f) = \sum_{i=1}^{s} b_i K_i \tag{2.22}$$

$$K_i = f(t_n + c_i h, U_n + h \sum_{j=1}^{s} a_{ij} K_j), i = 1, 2, ..., s, c_i \geq 0. \tag{2.23}$$

A method of this kind is called a Runge-Kutta (R-K) method. For a simpler presentation, we introduce the **Butcher tableau of the scheme** $\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$ or

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & ... & a_{1s} \\ c_2 & a_{21} & a_{22} & ... & a_{2s} \\ ... & ... & ... & ... & ... \\ c_s & a_{s1} & a_{s2} & ... & a_{ss} \\ \hline & b_1 & b_2 & ... & b_s \end{array} \tag{2.24}$$

We will always assume $\sum_{j=1}^{s} a_{ij} = c_i$.

Definition 2.26. *If A is strictly lower triangular, then the method is called explicit; if A is only lower triangular, then the method is semi-explicit. In all other cases, it is an implicit method.*

Intuition 2.27. • *When A is lower triangular with zero diagonal, the calculation of K_1 is done explicitly. Then this allows the explicit calculation of K_2 and so on. The scheme is thus explicit.*
• *When A is triangular with a non-zero diagonal, the method requires the sequential solution of s equations (not necessarily linear) to find the K_i, $i \leq s$.*
• *When A is full, the method requires the simultaneous solution of s equations (a system of equations) to find the K_i, $i \leq s$.*

Remark 2.28. *For implicit methods, one would also need to show the existence of a solution for the time steps. Assuming f is Lipschitz, this follows from Picard's fixed-point theorem by taking a recurrence as in Exercise 2.5.*

Example 2.29 (4th-order R-K method)**.** *Consider the scheme:*

$$U_{n+1} = U_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \quad (2.25)$$
$$K_1 = f_n = f(t_n, U_n), \quad (2.26)$$
$$K_2 = f(t_n + \frac{h}{2}, U_n + \frac{h}{2}K_1), \quad (2.27)$$
$$K_3 = f(t_n + \frac{h}{2}, U_n + \frac{h}{2}K_2), \quad (2.28)$$
$$K_4 = f(t_{n+1}, U_n + hK_3). \quad (2.29)$$

The associated Butcher tableau is

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array} . \quad (2.30)$$

This scheme is of order 4.

2.5.1 Construction of Second-Order Methods (Explicit)

Proposition 2.30. *Explicit methods with $s = 2$ that are second-order (in terms of convergence) satisfy $b_1 + b_2 = 1$ and $b_2 c_2 = 1/2$.*

Proof. Take $s = 2$. Since the method is explicit, the matrix A in the Butcher tableau is strictly lower triangular, i.e., $A = \begin{pmatrix} 0 & 0 \\ a_{21} & 0 \end{pmatrix}$. Let $a := a_{21}$, then $c_1 = 0 + 0 = 0$ and $c_2 = a + 0 = a$.
The corresponding Butcher tableau is:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \text{a} & \text{a} & 0 \\ \hline & b_1 & b_2 \end{array}$$

Then $U_{n+1} = U_n + hF$ with: $F = b_1K_1 + b_2K_2$, $K_1 = f(t_n, U_n) = f_n$, $K_2 = f(t_n + ah, U_n + haK_1) = f(t_n + ah, U_n + haf_n)$.
So, $U_{n+1} = U_n + h(b_1 f_n + b_2 f(t_n + ah, U_n + ahf_n))$. For it to be of order 2, the truncation error $\tau_{n+1}(h)$ must satisfy $\tau_{n+1}(h) = O(h^2)$.
Recalling that $\tau_{n+1}(h) = \frac{X_{n+1} - U^*_{n+1}}{h}$, where U^*_{n+1} is the scheme starting from the exact solution $X(t_n) = X_n$, here, $U^*_{n+1} = X_n + h[b_1 f(t_n, X_n) + b_2 f(t_n + ah, X_n + ahf(t_n, X_n))]$.
We will need the 2D Taylor formula: let G be a $\mathcal{C}^2$ function, then

$$G(\alpha + h_1, \beta + h_2) = G(\alpha, \beta) + \underbrace{\frac{\partial G}{\partial \alpha}(\alpha, \beta)}_{\text{notation: } G_\alpha} h_1 + \underbrace{\frac{\partial G}{\partial \beta}}_{G_\beta}(\alpha, \beta) h_2$$
$$+ O\left(\left(\sqrt{h_1^2 + h_2^2} \right)^2 \right). \tag{2.31}$$

So, by the 2D Taylor formula,

$$f(t_n + ah, X_n + ahf(t_n, X_n))$$

$$= f(t_n, X_n)+ah[f_t(t_n, X_n)+f_X(t_n, X_n)f(t_n, X_n)]+O(h^2)$$
$$= f(t_n, X_n)+ah[f_t(t_n, X_n)+f_X(t_n, X_n)f(t_n, X_n)]+O(h^2)$$

On the other hand, as $X'(t) = f(t, X(t))$, we also have

$$X''(t) = \frac{\mathrm{d}}{\mathrm{d}t}X'(t) = \frac{\mathrm{d}}{\mathrm{d}t}f(t, X(t))$$
$$= f_t(t, X(t)) + f_X(t, X(t))f(t, X(t)).$$

So,

$$f(t_n + ah, X_n + ahf(t_n, X_n)) = f(t_n, X_n) + ahX''(t_n) + O(h^2)$$
$$= X'(t_n) + ahX''(t_n) + O(h^2) \tag{2.32}$$

The truncation error thus satisfies:

$$\tau_{n+1}(h) = \frac{X_{n+1} - U^*_{n+1}}{h}$$
$$= \frac{\overbrace{X(t_{n+1}) - X(t_n)}^{\text{Taylor-Lagrange}} - hb_1X'(t_n) - hb_2(X'(t_n) + ahX''(t_n)) + O(h^3)}{h}$$
$$= \frac{hX'(t_n) + \frac{h^2}{2}X''(t_n) + O(h^3) - hb_1X'(t_n) - hb_2(X'(t_n) + ahX''(t_n)) + O(h^3)}{h}$$
$$= X'(t_n)(1 - b_1 - b_2) + h\left(\frac{X''(t_n)}{2} - ab_2X''(t_n)\right) + O(h^2)$$
$$= X'(t_n)(1 - b_1 - b_2) + hX''(t_n)\left(\frac{1}{2} - ab_2\right) + O(h^2). \tag{2.33}$$

For the method to be of order 2, it is necessary and sufficient that $b_1 + b_2 = 1$ and $ab_2 = 1/2$, which gives the conclusion. □

Example 2.31 (Heun as a Second-Order RK Method). *In the case where $b_1 = b_2$, then $b_1 = b_2 = \frac{1}{2}$ and $a = 1$:*

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

So: $K_1 = f(t_n, U_n) = f_n$ and $K_2 = f(t_n+h, U_n+hf(t_n, U_n)) = f(t_{n+1}, U_n+hf_n)$, thus $U_{n+1} = U_n+\frac{h}{2}[f_n+f(t_{n+1}, U_n+hf_n)]$, giving us the Heun method.

2.5.2 Consistency

Proposition 2.32. *Let f be a Lipschitz function. Then the Runge-Kutta method (explicit) is consistent if and only if $\sum_{i=1}^{s} b_i = 1$.*

Proof. by Taylor expansions. □

2.6 Advanced considerations: adaptive time steps for Runge-Kutta

Motivation: Sometimes the solution is almost constant, but other times it is highly variable. We would like to take advantage of the "calm" regions and use a large step h, which will be adjusted later in highly oscillatory regions. For this, we need to employ **adaptive step** (i.e., variable and adjusted) methods. To determine how to choose this step, we need **error estimations**.

How to estimate the error in practice? The easiest way would be to double the step. One idea would be to perform a calculation with a step $2h$ and compare it with two consecutive calculations with step h. Let Y_{2h} be the value obtained after a single step of size $2h$ and $Y_{h,h}$ after two steps of size h. Both are assumed to start from X_n or close to it within a tolerance. We know that, if the method is of order $p > 1$:

$$X(t_n + 2h) = Y_{2h} + (2h)^{p+1}\psi_n + O(h^{p+2})$$
$$X(t_n + 2h) = Y_{h,h} + 2h^{p+1}\psi_n + O(h^{p+2}).$$

The quantity $\Delta = Y_{2h} - Y_{h,h} = (2^{p+1} - 2)h^{p+1}\psi_n$ helps us adjust the h.

Remark 2.33. *The formulas provide an approximation of order $p + 1$ for $X(t_n + 2h)$. However, the error would then be unknown.*

Although in principle the method above would be interesting, it uses too many evaluations of f. We will refine it by constructing two methods that use the same evaluations (thus same K_i) but whose linear combinations involving b_i yield different orders. We then talk about **embedded schemes** of Runge-Kutta-Fehlberg (R-K-F). Notation:

$$\begin{array}{c|c} c & A \\ \hline & b^T \\ & \hat{b}^T \\ \hline & E^T \end{array}$$

where c, A, b give a scheme of order p while $c, A, \hat{b}^T$ give a scheme of order $p+1$. The difference $E = b - \hat{b}$ serves to estimate the truncation error $\Delta = h \sum_{i=1}^{s} E_i K_i$.

The most popular are the R-K-F schemes of orders 4-5 or 5-6 or even 2-3. In practice, the algorithm is as follows:

- At the beginning, we specify a tolerance Δ_0.

- If $\Delta > \Delta_0$ then we redo the calculation with the step $\tilde{h} = h \sqrt[p+1]{\frac{\Delta_0}{\Delta}}$.

- If $\Delta \leq \Delta_0$, the step h is kept constant.

2.7 Advanced considerations: systems of ODEs

Let $I \subset \mathbb{R}_+$ be an open interval and $F : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$. The problem is to solve the following Cauchy problem:

$$\begin{cases} Y'(t) = F(t, Y(t)) \\ Y(t=0) = Y_0 \in \mathbb{R}^n \end{cases} \tag{2.34}$$

Example 2.34. *$X'' = f(X)$ is not an ODE, but it can be written as a system, by setting: $Y_1 = X, Y_2 = X'$, and we get:*

$$\begin{cases} Y_1' = Y_2 \\ Y_2' = f(Y_1) \end{cases}$$

Theorem 2.35 (existence and uniqueness). *Let $F :]-\infty, \infty[\times \mathbb{R}^n \to \mathbb{R}^n$ be a continuous function and Lipschitz with respect to the second variable*

$$\|F(t,y) - F(t,\tilde{y})\| \leq L\|y - \tilde{y}\| \quad \forall t \in \mathbb{R},\ \forall y \in \mathbb{R}^n,$$

with an L not depending on $y \in \mathbb{R}^n$. Then the Cauchy problem (2.34) *has a unique global solution (i.e., defined for all $t \geq 0$). If F is Lipschitz only around $(t_0 = 0, Y_0)$, then the solution is only locally defined.*

Particular case: $F(t, y) = Ay$ with A being an $n \times n$ matrix. The problem

$$\begin{cases} Y' = AY \\ Y(0) = Y_0 \end{cases}$$

has the (unique) solution $Y(t) = e^{At}Y_0$ where we recall the definition of $e^{At} = \sum_{n=0}^{\infty} \frac{(At)^k}{k!}$.
If A is diagonalizable, i.e.:

1) $\exists Q$ invertible such that $A = QDQ^{-1}$, D diagonal

2) or equivalently $\exists V_i, \lambda_i$, such that $AV_i = \lambda_i V_i$, $\|V_i\| = 1$, and $\{V_i; i = 1, \ldots, n\}$ is a basis of $\mathbb{R}^n$

then, $Y(t) = \sum_{i=1}^{n} e^{\lambda_i t} V_i < y_0, V_i >$.

2.7.1 System Stability:

By setting $Z = Q^{-1}Y$, we obtain:

$$Y' = AY \Longrightarrow Y' = QDQ^{-1}Y \Longrightarrow Q^{-1}Y' = DQ^{-1}Y \quad (2.35)$$

And, since $Z' = (Q)^{-1}Y'$, we derive the following differential equation: $Z' = DZ$, and the problem:

$$\begin{cases} Z_1' = \lambda_1 Z_1 \\ \quad \vdots \\ Z_n' = \lambda_n Z_n \end{cases}$$

The solutions to this problem are given by: $Z_i(t) = e^{\lambda_i t} Z_i(0)$, and the stability of the system is equivalent to the stability of all the ODEs in the problem.

Example 2.36 (Explicit Euler). *$U_{n+1} = U_n + hf(t_n, U_n)$. Let $f(y) = Dy$.*

Applying Explicit Euler to this example, we have: $U_{n+1} = U_n + hDU_n = (1 + hD)U_n$. Therefore, the scheme is stable if $|1 + h\lambda_i| < 1$ for all $i = 1, \ldots, n$.

Example 2.37 (Implicit Euler). *$U_{n+1} = U_n + hf(t_{n+1}, U_{n+1})$; for the previous example, we have: $U_{n+1} = U_n + hDU_{n+1}$, so $U_{n+1} = (1 - hD)^{-1}U_n$. The scheme is stable if $|1 - h\lambda_i| > 1$ for all $i = 1, \dots, n$.*

To know more 2.38 (Implementation). *We always consider the case $f(t, y) = Ay$.*
- For explicit schemes $U_{n+1} = U_n + hAU_n$, so it's a direct calculation.
- For implicit schemes $U_{n+1} = U_n + hAU_{n+1}$, $U_{n+1} = (I - hA)^{-1}U_n$; a linear system must be solved. For more complicated functions f, methods like Newton's or Picard's approximation are needed (cf. exercise 2.5) etc. ...

2.7.2 Stiff Systems

We have seen that implicit schemes are sometimes challenging to implement; why use them then? Consider the following differential system:

$$\begin{aligned} u' &= 998u + 1998v \text{ with } u(0) = 1 \\ v' &= -999u - 1999v \text{ with } v(0) = 0 \end{aligned}$$

We make the change of variables $u = 2y - z$ and $v = -y + z$, which gives us:

$$\begin{cases} y' = -y \\ z' = -1000z \end{cases} \Rightarrow \begin{cases} y(t) = e^{-t}y_o \\ z(t) = e^{-1000t}z_0 \end{cases}$$

(so $\lambda_1 = -1$, $\lambda_2 = -1000$). Returning to our initial variables, we obtain the desired solution:

$$\begin{aligned} u &= 2e^{-t} - e^{-1000t}, \\ v &= e^{-t} + e^{-1000t}. \end{aligned}$$

For the stability of Explicit Euler, $|1 + h\lambda_1| < 1$ and $|1 + h\lambda_2| < 1$ are required, so $h \leq \frac{2}{1000}$.

For Implicit Euler stability, $|1 - h\lambda_i| > 1$, which is always satisfied ($\forall h > 0$). Assuming we are only interested in the e^{-t} part of the solution (treating e^{-1000t} as a perturbation, which it is), the precision of both schemes could be good for sufficiently large steps h; however, for Explicit Euler, we must use a small h as stability is not guaranteed otherwise. Conclusion: using implicit schemes allows solving with a larger h, hence more quickly.

2.8 Multi-step Methods

The idea behind these schemes is to use previous steps (values) that are available.

Vocabulary: These schemes are also known as predictor-corrector methods.

Definition 2.39. *The linear multi-step scheme of order s (with the notation $a_s = 1$) is given by the recurrence:*

$$\sum_{k=0}^{s} a_k y_{n+k} = h \sum_{k=0}^{s} b_k f(t_{n+k}, y_{n+k}). \tag{2.36}$$

Here y_{n+s} is unknown, and the previous values $y_{n+s-1}, \dots, y_n$ are known.

We notice that if $b_s \neq 0$, then the method is implicit; otherwise, it is explicit. Let's consider some examples.

Example 2.40. Explicit Euler *For $s = 1, a_0 = -1$, $a_1 = b_0 = 1$, $b_1 = 0$,*

$$1 \cdot y_{n+1} + (-1) \cdot y_n = 1 \cdot hf(t_n, y_n) + 0 \cdot hf(t_{n+1}, y_{n+1}).$$

Example 2.41. Adams-Bashforth Two-Step (explicit) *Taking $s = 2, a_0 = 0, a_1 = -1, a_2 = 1, b_0 = -\frac{1}{2}, b_1 = \frac{3}{2}, b_2 = 0$, the scheme is defined by the relation:*

$$y_{n+2} = y_{n+1} + \frac{3}{2} hf(t_{n+1}, y_{n+1}) - \frac{1}{2} hf(t_n, y_n). \tag{2.37}$$

Example 2.42. BDF Two-Step (implicit) *Here* $s = 2, a_0 = \frac{1}{3}, a_1 = -\frac{4}{3}, a_2 = 1, b_0 = 0, b_1 = 0, b_2 = \frac{2}{3}$*, and thus:*

$$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_n = \frac{2}{3}hf(t_{n+2}, y_{n+2}). \tag{2.38}$$

Definition 2.43. *The local truncation error is* $\tau_{n+s}(h)$ *defined by:*

$$\tau_{n+s}(h) := \frac{\sum_{k=0}^{s} a_k X(t_{n+k}) - h\sum_{k=0}^{s} b_k f(t_{n+k}, X_{n+k})}{h}. \tag{2.39}$$

Reminder: The global truncation error is $\tau(h) = \max_n |\tau_{n+s}(h)|$; the multi-step scheme $(a_k, b_k)_{k=0}^{s}$ is consistent if $\lim_{h\to 0}\tau(h) = 0$.

Theorem 2.44. *The multi-step scheme* $(a_k, b_k)_{k=0}^{s}$ *is consistent if and only if:*

$$\sum_{k=0}^{s} a_k = 0, \ \sum_{k=0}^{s} b_k = \sum_{k=0}^{s} k a_k. \tag{2.40}$$

or, equivalently:

$$\sum_{k=0}^{s-1} a_k = -1, \ \sum_{k=0}^{s} b_k = s + \sum_{k=0}^{s-1} k a_k. \tag{2.41}$$

Example 2.45 (EE as a Consistent Multi-step Scheme). *Taking the previous example where* $a_0 = -1, a_1 = 1, b_0 = 1, b_1 = 0$*, the conditions for the theorem are satisfied. Thus, the EE scheme is consistent.*

Example 2.46 (Adam-Bashforth as a Consistent Multi-step Scheme). *The Adam-Bashforth scheme satisfies*

$$\sum_{k=0}^{2} a_k = 0 \ \textit{and} \ \sum_{k=0}^{2} a_k k = \sum_{k=0}^{2} b_k = 1. \tag{2.42}$$

Therefore, this scheme is consistent.

Example 2.47 (BDF as a Consistent Multi-step Scheme). *The BDF scheme ($s = 2$) satisfies* $\sum_{k=0}^{2} a_k = 0$ *and* $\sum_{k=0}^{2} a_k k = \sum_{k=0}^{2} b_k = 2/3$. *Hence, this scheme is also consistent.*

2.9 Application in Epidemiology: SIR Model

We quickly present the modeling leading to the SIR system. We refer to [5, 3] and similar references for a more detailed presentation (just to change, we took a reference prior to the COVID-19 pandemic...). The variables are:

- S = the number of people susceptible to infection (the population not yet affected by the epidemic).

- I = the number of infected people.

- R = the number of people who have had the disease, died, or can no longer transmit it (having acquired immunity or being in quarantine, etc.).

Let $N = S(0) + I(0) + R(0)$ be the initial population (which will be conserved).

Model assumptions:

1. The number of infections (transition from S to I) is proportional to the number of individuals in S, the infection rate $I(t)/N$, and the duration Δt. We denote β as the proportionality factor. Since the number of new infections between t and $t + \Delta t$ is $S(t) - S(t + \Delta t)$, we obtain $S(t) - S(t + \Delta t) \simeq \beta S I \Delta t / N$

2. The transition $I \to R$ is proportional to the number of individuals in I; it depends on the recovery rate γ (here $1/\gamma$ is the average number of days before leaving the I compartment).

By taking the limit $\Delta t \to 0$, we obtain the system of equations, called the SIR model:

$$\frac{dS}{dt} = -\beta SI/N \tag{2.43}$$
$$\frac{dI}{dt} = \beta SI/N - \gamma I \tag{2.44}$$
$$\frac{dR}{dt} = \gamma I \tag{2.45}$$

We assume $S(0) = S_0 \neq 0,\ I(0) = I_0 > 0,\ R(0) = R_0 = 0$.

Remark 2.48. *Since* $\frac{d}{dt}(S+I+R) = 0$, *then* $S(t)+I(t)+R(t) = N = cst$.

A first question to ask is when I will be increasing or decreasing. Fortunately, this can be read directly from the equation for I: $I' = I(\beta S/N - \gamma)$, so it will increase when S/N is greater than $\frac{1}{\mathcal{R}_0}$, where $\mathcal{R}_0 = \beta/\gamma$ is known as the "reproduction rate." Note that $\mathcal{R}_0$ depends only on the characteristics of the disease and transmission and not on the state $S(t), I(t), R(t)$. In particular, if initially $S(0) \simeq N$, there will be no epidemic (in the sense that I will always be decreasing) if $\mathcal{R}_0 < 1$, and conversely, there will be an epidemic with exponential growth if $\mathcal{R}_0 > 1$.

Therefore, $\mathcal{R}_0$ is an important number, and in particular, it is the target of most epidemic containment policies; to make it sub-unitary, you can:

1. Make β small by reducing contacts (lockdown, etc.).

2. Make γ large by isolating the sick (so they are no longer contagious).

3. Otherwise, finally, make $S(0)$ smaller through vaccination if it is **effective and without side effects** (otherwise, it won't work).

In this (constant parameter!) model, the typical evolution of I, illustrated in Figure 1.2 on page 8, is as follows: it grows until a maximum value (corresponding to the "epidemic peak") and then decreases. Of course, in practice, it's more complicated because β will change depending on the preventive measures implemented, which in turn will fluctuate, etc.

The total number of infected individuals is

$$R_\infty := \lim_{t\to\infty} R(t) = I_0 + S_0 - \lim_{t\to\infty} S(t).$$

Note that $\lim_{t\to\infty} S(t)$ exists because $S(t), I(t), R(t) \geq$, $\forall t$ and S is decreasing).

Let ζ be the size of the epidemic. It can be shown (see [3]) that ζ is a solution of $1 - \frac{\zeta}{S(0)} = e^{-\mathcal{R}_0(\zeta + I(0))}$.

Remark 2.49. *Note that $S_\infty := \lim_{t\to\infty} S(t) \neq 0$; therefore, even in the absence of any protective measures, the epidemic will not affect everyone. It is then called the phenomenon of herd immunity in the SIR model. However, 'not everyone' can still include too many people, and in practice, epidemic containment measures must be taken.*

In practice: β and γ are unknown, so we proceed in 2 steps

- Inversion: find β, γ from observations $R(n)$,$n = 1, \ldots,$ N_{max}
- Prediction: calculate $S(t), I(t), t \geq N_{max}$

Remark 2.50. *Sometimes more complicated models are necessary, such as: $S \to E \to I \to R$ or as in [1].*

2.10 Ordinary Differential Equations Exercises

Exercise 2.1. *(Gronwall's Lemma: Integral Variant)*

Let $T > 0$ (it can also be ∞), $a(t)$, $b(t)$, $\lambda(t)$ be continuous functions on $[0, T]$, where $\lambda(t) \geq 0$ for all t. Define $\Lambda(t) = \int_0^t \lambda(\tau)d\tau$.

1. *If for all $t > 0$:*

$$a(t) \leq b(t) + \int_0^t \lambda(s)a(s)ds, \tag{2.46}$$

   then

$$a(t) \leq b(t) + \int_0^t e^{\Lambda(t)-\Lambda(s)}\lambda(s)b(s)ds. \tag{2.47}$$

   Indication: Estimate the derivative of $A(t) = e^{-\Lambda(t)} \int_0^t \lambda(s)a(s)ds$.

   Alternative: Let ξ be the right-hand side of (2.47). It also satisfies $\xi(t) = b(t) + \int_0^t \lambda_s \xi_s ds$ (direct calculation or use $V(t) = \int_0^t \lambda\xi$), i.e., (2.46) with equality. It is then natural to want to show that $a(t) \leq \xi(t)$ for all t; this is done by bounding $\xi - a$ using (2.46) and the equation for ξ.

2. *If b is differentiable with an integrable derivative on $[0, T]$, then*

$$a(t) \leq e^{\Lambda(t)} \left(b(0) + \int_0^t e^{-\Lambda(s)} b'(s)ds \right). \tag{2.48}$$

3. *If, in addition, b is monotonically increasing, then*

$$a(t) \leq e^{\Lambda(t)} b(t). \tag{2.49}$$

4. *Verify that in the absence of the assumption $\lambda(t) \geq 0$, a counterexample is $\lambda(t) = \lambda < 0$, $b(t) = b + \omega(t)$, supp$(\omega) \subset]0, T[$, $a(t) = be^{\lambda t}$.*

Exercise 2.2. *(Gronwall: Differential Variant* ***without the sign assumption****)*

Let $T > 0$ (it can also be ∞), $g(t)$, $\lambda(t)$ be continuous functions on $[0, T]$, and $a(t)$ a differentiable function with a continuous derivative on $[0, T]$. Define $\Lambda(s) = \int_0^t \lambda(\tau)d\tau$.

If for all $t > 0$:

$$a'(t) \leq g(t) + \lambda(t)a(t) \tag{2.50}$$

then

$$a(t) \leq e^{\Lambda(t)}a(0) + \int_0^t e^{\Lambda(t)-\Lambda(s)}g(s)ds. \tag{2.51}$$

Indication: Estimate the derivative of $A(t) = e^{-\Lambda(t)}a(t)$.

Exercise 2.3. *(Gronwall: Discrete Variant)*

Let k_n be a sequence of positive real numbers and $\phi_n \geq 0$ a sequence such that

$$\phi_0 \leq g_0 \tag{2.52}$$

$$\phi_n \leq g_0 + \sum_{s=0}^{n-1} p_s + \sum_{s=0}^{n-1} k_s\phi_s, \;\; n \geq 1. \tag{2.53}$$

If $g_0 \geq 0$ and $p_n \geq 0$ for all $n \geq 0$, then

$$\phi_n \leq \left(g_0 + \sum_{s=0}^{n-1} p_s\right) exp\left(\sum_{s=0}^{n-1} k_s\right) \tag{2.54}$$

Exercise 2.4. *Consider the Cauchy problem:*

$$x'(t) = 2|x(t)|^{1/2} \tag{2.55}$$

$$x(0) = 0 \tag{2.56}$$

1. *Show that for any constant $\lambda \in [0, \infty]$, this problem has the solution $x_\lambda(t) = (t - \lambda)^2$ if $t \geq \lambda$ and $x_\lambda(t) = 0$ otherwise. Comment on uniqueness.*

2. *Write an explicit/implicit Euler scheme and explain towards which solution $x_\lambda(t)$ it converges numerically.*

Exercise 2.5 (Existence of Implicit Schemes). *Let $\Psi : \mathbb{R}_+ \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be Lipschitz with respect to all arguments, and let $h > 0$.*

1. *Show that the equation*

$$y = x + h\Psi(t, x, y), \tag{2.57}$$

has a unique solution for small enough h and provide a numerical method to compute it. Hint: Picard iterations can be used. Notation: the solution will be denoted by $y = s(t, x, h)$.

2. *Now, let y be a solution of* (2.57), *and ϕ be the function defined by*

$$y = x + h\phi(t, x). \tag{2.58}$$

Provide the formula for ϕ in terms of $s(\cdot)$ and $\Psi(\cdot)$ and show that ϕ is well-defined and Lipschitz for sufficiently small h.

Exercise 2.6 (Stability of Implicit Schemes). *By using possibly Exercise 2.5, show that the Crank-Nicholson scheme satisfies the assumptions of Theorem 2.13 (page 19) for zero-stability.*

Exercise 2.7 (Theoretical Convergence). *Provide a convergence result for the Euler scheme without using the discrete Gronwall's lemma.*

Hints: Start without round-off errors and establish a recurrence formula for the error.

Exercise 2.8 (RK Writing). *Verify that the Heun's method is indeed a two-step Runge-Kutta method and write down the corresponding Butcher tableau.*

Do the same for the modified Euler method:

$$u_{n+1} = u_n + hf(t_n + \frac{h}{2}, u_n + \frac{h}{2} f_n)$$

Exercise 2.9 (θ-Scheme)**.** *Consider the "θ-scheme":*

$$u_{n+1} = u_n + h\left\{(1-\theta)f(t_n, u_n) + \theta f(t_{n+1}, u_{n+1})\right\}.$$

1. *Write down the Butcher tableau of the scheme.*

2. *Show that the stability region of this scheme includes $\{z = h\lambda; Re(z) < 0\}$ if and only if $\theta \geq 1/2$.*

Exercise 2.10 (Multi-Step Methods)**.** 1. *Show that the BDF-2 scheme* (2.38) *satisfies the consistency conditions.*

2. *Show that for BDF-2, the truncation error is indeed of order* 2*.*

3. *Determine the order of the truncation error for the Adam-Bashforth scheme* (2.37)*.*

Exercise 2.11 (SIR Model)**.** *Write one step of the implicit Euler method for the system*

$$\begin{aligned}\frac{dS}{dt} &= -rSI,\\ \frac{dI}{dt} &= rSI - aI,\\ \frac{dR}{dt} &= aI.\end{aligned}$$

Exercise 2.12 ((Identification of ODE Schemes))**.** *With the notations from the lecture, a student intends to numerically solve the Lorenz system: $x'(t) = \sigma(y(t) - x(t))$, $y'(t) = x(t)(\rho - z(t)) - y(t)$, $z'(t) = x(t)y(t) - \beta z(t)$. He has three programs L1, L2, and L3, each implementing a different scheme in the list: Explicit Euler, Implicit Euler, Crank-Nicholson. He performs the following tests: he runs the three programs with $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$. $h = 10^{-2}$, starting from $(x_0, y_0, z_0) = (1, 2, 3)$. He obtains vectors representing the numerical solution after* **TWO time steps** *(to 1×10^{-3} accuracy):*

Program L1: `v_1=[1.228, 2.511, 2.893]`
Program L2: `v_2=[1.213, 2.483, 2.886]`
Program L3: `v_3=[1.244, 2.543, 2.900]`

1. *Recall the definitions of the three schemes.*
2. *Find the values that the Explicit Euler scheme obtains after* **ONE time step**.
3. *Determine which scheme each program uses. Rigorously justify your answer.*

Note: It is possible to solve the problem without excessive calculations or a calculator. If necessary, approximate to 3-4 decimal places. Solution on page 99.

Various other exercises (Additional ODE)

Exercise 2.13. *Consider the Cauchy problem:*

$$x' = 2y, \tag{2.59}$$
$$y' = -2x - 4x^3 - y, \tag{2.60}$$

with initial values $(x(0), y(0)) = (x_0, y_0) \neq (0,0)$. Show that this problem has a maximal solution over the interval $]\alpha, \beta[$ (with $-\infty \leq \alpha < \beta \leq \infty$).

Exercise 2.14. *Consider the Cauchy problem:*

$$x' = 2y(z-1), \tag{2.61}$$
$$y' = -x(z-1), \tag{2.62}$$
$$z' = -xy, \tag{2.63}$$

with initial values $(x(0), y(0), z(0)) = (x_0, y_0, z_0)$. Show that this problem has a maximal solution over the interval $[0, \infty[$.

Exercise 2.15 (Autonomous systems). *Consider the system $x' = f(x)$ with f being C^1 class. The state x is a vector in $\mathbb{R}^d$.*

1/ Let x_1 and x_2 be two solutions of this system. Then if these solutions touch at a point, they are equal.

2/ So, let x be a solution. Then either $t \mapsto x(t)$ is injective, or it is periodic.

2.11 ODE Python lab

Exercise 2.16 (Numerical precision). *Implement exercise 2.4 in order to observe all the behaviours described in the lecture for EE and IE with finite precision or not.*

Exercise 2.17 (SIR model, scheme order). *Write a program that solves the SIR system* (2.43)-(2.45) *using the Euler Explicit, Heun, and Runge-Kutta schemes of order* 4 *with Butcher table* (2.30)*. Take as an example* $S_0 = 10.0^6$, $I_0 = 10$, $R_0 = 0$ *(but work with the proportions of the total population),* $r = 0.5$, $a = 0.33$, $T = 150.0$, $N = 150$ *(*$h = T/N$*).*

1. *Implement it using the "odeint" function in python (without any scheme).*

2. *Study the order of the schemes by varying* h *and comparing it with the solution found by 'odeint' at time* T *(take the error on "S"). For this study, take* $T_0 = 52$, $T = 60$ *(get the initial values at time* T_0 *from the previous calculation) and* $h = 0.05, 0.01, 0.1, 0.5, 1, 2, 4$*. The result should be similar to that in figure 2.2.*

3. *Study the impact of control policies that will change* r *and* a*.*

Exercise 2.18. *Numerically study the stability of Euler Explicit for the case of the system* $x'(t) = \lambda x(t)$ *with* $\lambda = i$, $T = 100 \cdot 2\pi$, $h = 2\pi/100$*. Typical results are in figure 2.3.*

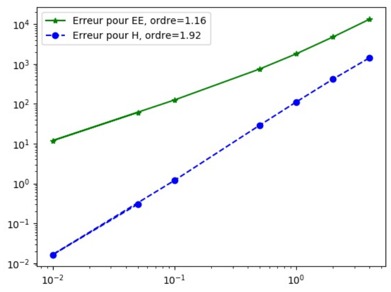


Figure 2.2: Results for exercise 2.17

.

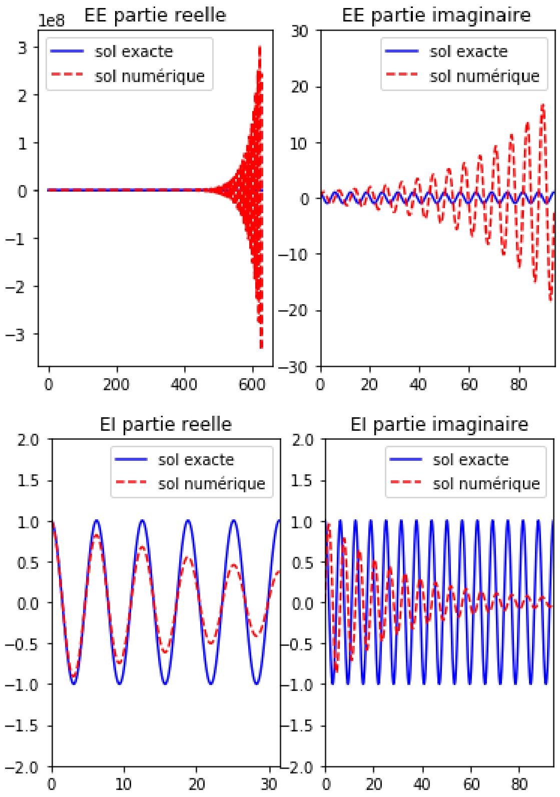


Figure 2.3: Stability of EE and IE schemes.

Chapter 3

Automatic Differentiation, Backpropagation, Optimization, Control

3.1 Introduction

The goal of this chapter is to provide some insights into how to automatically compute gradients (given the code that computes the function). This is known as *automatic differentiation*; one way to implement it is through *backpropagation*, which is used in optimization and control problems. Let's start with some examples.

3.1.1 Example 1: Explicit function

Calculating the gradient of a function with 3 variables $f(x, y, z) = (3x^2 + y)z - x$. This case is quite simple; we can perform the operations manually to calculate $\nabla f = (\partial_x f, \partial_y f, \partial_z f)$. This will serve as a textbook case and verification.

3.1.2 Example 2: Optimization in an epidemiological model

We now consider a current context with the SIR system, seen previously. The system is of the form $X' = f(X(t), u(t))$

$$\begin{cases} \dot{S} = -\beta SI/N_p \\ \dot{I} = \beta SI/N_p - \gamma I \\ \dot{R} = \gamma I \end{cases}$$

with $u(t) = \beta(t) = \beta$ the control and N_p the total population. Thus, the goal is to minimize the cost of containment ($\int_0^T c(\beta(t))dt$) and the number of infected people $(S(0) - S(T))$ over the period [0,T]. In other words, we seek:

$$\min_{\beta} J[\beta] := S(0) - S(T) + \int_0^T c(\beta(t))dt$$

We have initially:

$$\frac{\partial}{\partial \beta}\left[\int_0^T c(\beta(t))dt\right] = c'(\beta(t)).$$

However, the difference $S(0) - S(T)$ is much less obvious to differentiate because β does not appear explicitly, although it does appear in this function.

3.1.3 Example 3: Neural networks (NN)

Suppose we have a database Ω; we then wish to define a neural network, *Neural Network* (NN), that will "learn" based on certain results from this database, $\omega \in \Omega$.

We will then seek to optimize certain parameters X, so that the output of our network is as close as possible to the chosen examples ω. Thus, it is an optimization problem:

$$\arg\min_{X} \{\mathbb{E}_{\omega}\left(\mathcal{L}(X, \omega)\right)\}$$

We will be concerned with neural network of which we give an example below. The principle is as follows: we have an input layer, called the *Input Layer*, a certain number of hidden layers, the *Hidden Layers*, and an output layer, the *Output Layer*, see figure 3.1 for an illustration.

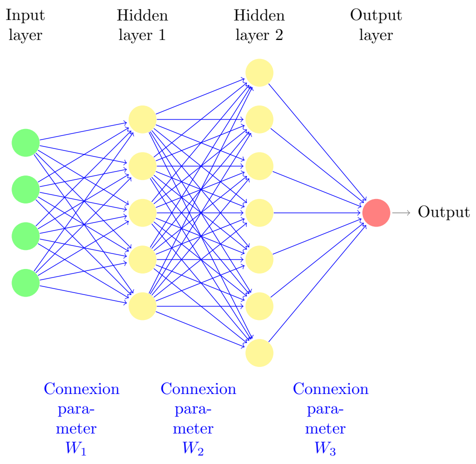

Input layer
Hidden layer 1
Hidden layer 2
Output layer
Output
Connexion parameter W_1
Connexion parameter W_2
Connexion parameter W_3

Figure 3.1: Example of a neural network

Each of the hidden layers $(\tilde{Y}, Y)$ performs the following calculation: We take all the results from the previous layer, we assign them a weight (a vector W, and a bias b). We then obtain $\tilde{Y}$ and finally apply an activation function $\mathcal{A}$ (for example, the ReLU function which is the positive part: $ReLU(x_1, ..x_N) = ((x_1)_+, ..., (x_N)_+)$, we get Y.

Finally, we apply an output function g to the last layer (for example, a sigmoid-type function $x \mapsto \frac{1}{1+e^{-x}}$ which transforms the input into an output of type 'probability', i.e., a number between 0 and 1). If we have to choose between K classes, we can use the softmax function

$$x \in \mathbb{R}^K \mapsto s(x) := \left(\frac{e^{x_k}}{\sum_{\ell=1}^{K} e^{x_\ell}} \right)_{k=1}^{K} \in \mathbb{R}^K, \tag{3.1}$$

which returns a probability distribution, and in particular the component $s(x)_k = \frac{e^{x_k}}{\sum_\ell e^{x_\ell}}$ of the result can be interpreted as the probability that label k is correct one.

In general, we can describe the output Y_{n+1} of a given layer $n+1$, with respect to the output Y_n of the previous layer n, as follows:

$$\begin{cases} \tilde{Y}_{n+1} = W_{n+1} Y_n + b_{n+1} \\ \\ Y_{n+1} = \mathcal{A}_{n+1}\left(\tilde{Y}_{n+1}\right) \end{cases}$$

We then seek to optimize the weights W as well as the biases b, that is, to optimize:

$$X = (W_1, b_1, ..., W_n, b_n)$$

To minimize a loss function: $\mathcal{L}(X, \omega)$ which describes "the difference between the estimation by the neural network with parameters X, and the examples ω". This minimization can then be performed with gradient descent.

3.2 Finite Difference Approach

A first idea would be to use the formula

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h). \tag{3.2}$$

This formula is derived using the Taylor series. It is called the formula of finite differences (non-centered). There is also another more accurate formula

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2). \tag{3.3}$$

It is also derived with the Taylor series (exact to order 2 with remainder of order 3). The approximation (3.3) is called centered finite differences of order 2.

Now let's analyze the cost of such a formula for a general function $G(y_1, ..., y_N)$. It should be noted that in general, the most expensive part is the evaluation of the function G. Thus the cost will be expressed in terms of the number of evaluations of the function G needed to calculate $\nabla_X G$.

For example, if we want to obtain $\partial_{y_k} G$ we can use the approximation (3.3) for the function

$$x \mapsto f(x) := G(y_1, ..., y_{k-1}, x, y_k, ..., y_N). \qquad (3.4)$$

This means that calculating $\partial_{y_k} G$ will cost two evaluations of G. So calculating $\nabla_X G$ as a whole will cost $2N$ evaluations of G which is prohibitive when N is large (for neural networks N can be of the order of 10^9!). Another approach is needed.

To know more 3.1. *However, finite differences are used to independently verify the implementation that will be done with automatic differentiation detailed later on.*

Warning 3.2. *Generally, one also needs to pay attention to numerical precision. If we assume that the numerical precision of the function f calculation is 10^{-16}, then we will have an error of $\frac{10^{-16}}{2h} + h^2$; this can be minimized with respect to h to obtain the optimal error order which will be $O(10^{-10.66})$ (achieved for $h = O(10^{-16/3})$). Note that in particular, we never achieve the order of 10^{-16} for the derivative calculation (this can be slightly improved by using higher-order formulas than 2 ... but which have an even greater cost).*

3.3 Computational Graphs, Notions of *Forward* and *Backward*

3.3.1 Direct Computational Graphs

We now delve into the heart of the matter: computational graphs.

Definition 3.3. ***Computational Graph.*** *A computational graph is a directed, connected, acyclic graph, in which the* *nodes correspond to operations that create new variables using the values of variables from incoming nodes.*

Each variable can thus be used by a target node, and each operation takes as input the result of previous operations. Its own result can then be used by other target operation nodes.

Let's take the example of the function $f : \mathbb{R}^3 \to \mathbb{R}$ defined by:

$$f(x, y, z) = (3x^2 + y)z - x. \tag{3.5}$$

We can associate with this function f the computational graph in figure 3.2.

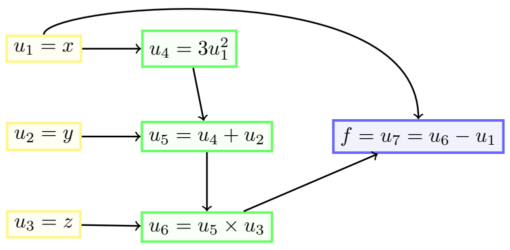


Figure 3.2: Computational graph of f in equation (3.5).

Remark 3.4. *the yellow nodes* (u_1, u_2, u_3) *have no incoming degrees, they are nodes that contain only the input variables to our function* f*. The blue node* u_7 *constitutes the output of the function* f*. This graph is said to be direct: it performs the same operations as those of the function.*

Definition 3.5 (Input, Output). *The Input of a direct computational graph consists of all nodes with zero incoming degree. The Output of a computational graph consists of all nodes with zero outgoing degree.*

Example 3.6. *Taking again figure 3.2, the yellow nodes* (u_1, u_2, u_3) *are Inputs, the blue node* u_7 *is an output.*

	U_4	U_5	U_6	U_7
number of variables	1	2	2	2
function	$3U_1^2$	$U_4 + U_2$	$U_5 \times U_3$	$U_6 - U_1$

Table 3.1: Functions of the nodes of the graph in figure 3.2.

In table 3.1 we specify the functions that define each node of the graph in figure 3.2. In an optimization problem, we will seek to obtain the gradient of the function computed by the graph. This is where the notion of inverse computational graph comes in.

3.3.2 Backward Computational Graphs

We begin with some reminders concerning the computation of derivatives of composite functions. We assume all functions to be sufficiently regular. Let $f : \mathbb{R}^L \to \mathbb{R}$ be a function and $g_1, \dots, g_L$ functions from $\mathbb{R}^N$ to $\mathbb{R}$, and let $\forall X \in \mathbb{R}^N$:

$$F(X) := f(g_1(X), \dots, g_L(X))$$

The derivative of the function F with respect to X_k is given by:

$$\frac{\partial F}{\partial X_k} = \sum_{l=1}^{L} \frac{\partial f}{\partial g_l}(g_1(X), \dots, g_L(X)) \frac{\partial g_l}{\partial X_k}. \tag{3.6}$$

We recall that the gradient of the function F is the vector:

$$\nabla_X F = \left(\frac{\partial F}{\partial X_1}, \dots, \frac{\partial F}{\partial X_L} \right). \tag{3.7}$$

Sometimes the notation "$\nabla_X F$" is also seen as "$J_X F$".

To know more 3.7. *We recall the definition of the first-order derivative (Jacobian):*

Definition 3.8 (Jacobian). *Let* $H : \mathbb{R}^N \to \mathbb{R}^P$ *with* *the notations, for $X = (X_1, ...X_N) \in \mathbb{R}^N$:*

$$H(X) = \begin{pmatrix} H_1(X) \\ ... \\ H_P(X) \end{pmatrix}. \tag{3.8}$$

The Jacobian matrix of H is the matrix of the derivatives of H written as follows

$$J_X H = \begin{pmatrix} \frac{\partial H_1}{\partial X_1} & \dots & \frac{\partial H_1}{\partial X_N} \\ \vdots & \dots & \vdots \\ \frac{\partial H_P}{\partial X_1} & \dots & \frac{\partial H_P}{\partial X_N} \end{pmatrix} \in \mathbb{R}^{P \times N}. \tag{3.9}$$

In particular $(J_X H)_{ij} = \frac{\partial H_i}{\partial X_j}$. Sometimes the transpose is also used $\frac{DH}{DX} = D_X H = (J_X H)^T$.

Remark 3.9. *The function $F : \mathbb{R}^N \to \mathbb{R}$ is in fact the composite function $f \circ g$. We have:*

$$\left(\frac{\partial F}{\partial X_1}, \dots, \frac{\partial F}{\partial X_L}\right) = J_X F = J_X(f \circ g) = J_g f \times J_X g, \tag{3.10}$$

where "$\times$" denotes the usual matrix-vector product. This is a formula for deriving composite functions.

Returning to the graph 3.2; we employ the following notation:

$$\delta U_k = \frac{\partial f}{\partial U_k}. \tag{3.11}$$

Obviously, $\delta U_7 = \delta f = 1$. For the others it is not as immediate but we can see immediately that this calculation seems to be easier to develop **in reverse**, which is why it will be called backward. Let's be more precise.

To use the formula (3.6), the calculation must be put in this form. This is not possible with the initial graph but with a 'layered' version as in figure 3.3. where each *layer* contains all the variables necessary to calculate the values of the next layer: the last layer contains only U_7, its derivative is already calculated. The penultimate layer contains only U_6 and $U_{1,4}$ (which propagates the value of x). We can use the composite derivation to obtain $\delta U_6 = \frac{\partial U_7}{\partial U_6} = 1$, according to the formula of U_7 in table 3.1; similarly $\delta U_{1,4} = \frac{\partial U_7}{\partial U_{1,4}} = -1$.

So we continue backwards; this new layer contains $U_{1,3}$, U_5 and $U_{3,3}$. The idea is to see $U_{1,3}$, U_5 and $U_{3,3}$ as inputs, the next layer U_6 and $U_{1,4}$ as intermediate variables and U_7 as the output. We then apply the composite derivation formula, for example:

$$\delta U_5 = \frac{\partial f}{\partial U_5} = \frac{\partial f}{\partial U_6} \cdot \frac{\partial U_6}{\partial U_5} + \frac{\partial f}{\partial U_{1,4}} \cdot \frac{\partial U_{1,4}}{\partial U_5} = \delta U_6 \cdot U_{3,3} + 0 = U_{3,3}.$$

The calculation can thus continue. Pay attention to the final calculation of δU_1 which will have two non-zero terms.

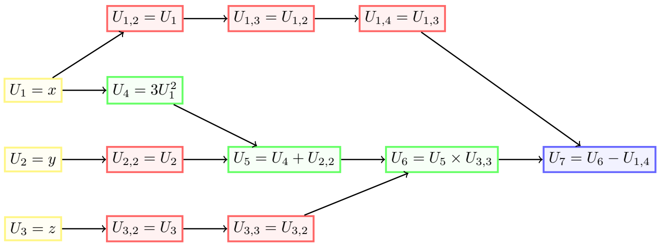


Figure 3.3: Layered computational graph of f in equation (3.5). It was built by adding variables that will be necessary later in the form of new trivial variables, such as $U_{1,2} = U_1$, etc.

Remark 3.10. *This 'layered' representation introduces additional variables, here in red, but has the advantage of having a block structure where each block uses only the block that directly precedes it. This solution has the advantage of being rigorous but the disadvantage of complicating the calculations.*

To simplify the calculations, the idea is to keep the initial graph but to follow the following rule: going backward from the output to the inputs, we calculate the derivative corresponding to a variable U_a only when all the derivatives δU_b of the boxes U_b such that an arrow between U_a and U_b exists have been calculated. Put simply: when everyone downstream of U_a in the graph has its derivative calculated, then we can also calculate the derivative δU_a. In our case, for example, this means attempting to calculate δU_1 only once δU_4 and $\delta U_{1,2}$ are known. The calculation results in the figure 3.4.

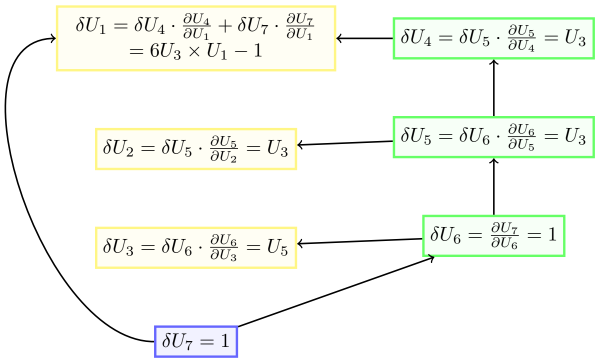

$\delta U_1 = \delta U_4 \cdot \frac{\partial U_4}{\partial U_1} + \delta U_7 \cdot \frac{\partial U_7}{\partial U_1} = 6U_3 \times U_1 - 1$
$\delta U_4 = \delta U_5 \cdot \frac{\partial U_5}{\partial U_4} = U_3$
$\delta U_2 = \delta U_5 \cdot \frac{\partial U_5}{\partial U_2} = U_3$
$\delta U_5 = \delta U_6 \cdot \frac{\partial U_6}{\partial U_5} = U_3$
$\delta U_3 = \delta U_6 \cdot \frac{\partial U_6}{\partial U_3} = U_5$
$\delta U_6 = \frac{\partial U_7}{\partial U_6} = 1$
$\delta U_7 = 1$

Figure 3.4: Backward computational graph of f in equation (3.5).

3.3.3 Summary

The remarks below are in practice the most useful part of the whole chapter. Read them carefully.

Important technique 3.11. *In summary, suppose given a (directed)* ***forward*** *computational graph $G = (V, E)$ which encodes the direct operations e.g., as our function:*

- $V = \{U_a, a = 1, ...n_v\}$ *are all required variables*
- E *are the edges i.e., if* $(a, b) \in E$ *then variable* U_a *is used to compute* U_b.

Then it is possible to construct a ***backward*** *computational graph $G_B = (V, E_B)$ with $E_B = \{(b,a)|(a,b) \in E\}$ which encodes the computation of the derivatives. Then, assuming that G is connected, the last vertex is final (i.e., out-degree null) and denoting $\delta U_a := \frac{\partial U_{n_v}}{\partial U_a}$ then :*

$$\delta U_{n_v} = 1,\ \forall a < n_v\ : \delta U_a = \sum_{(a,b)\in E} \delta U_b \cdot \frac{\partial U_b}{\partial U_a}. \quad (3.12)$$

In particular the computation on the backward graph starts from U_{n_v} and for any other a, one calculates δU_a only after all δU_b with $(a,b) \in E$ have been computed.

3.4 Example 1: Neural Networks

Suppose we have a training dataset Ω, and we want to define a neural network (NN) that will "learn" from this dataset, $\omega \in \Omega$.

We will then seek to optimize certain parameters X, so that the output of our network is as close as possible to the (known) output for the examples ω in the dataset Ω (for all $\omega \in \Omega$ the correct output is known; we are thus in the context of so-called supervised learning). It is therefore an optimization problem:

$$\arg\min_X \{\mathbb{E}_\omega (\mathcal{L}(X, \omega))\} \quad (3.13)$$

We propose to study the neural network defined by the schema illustrated in figure 3.1.

3.4.1 Problem Definition

The network consists of an input layer, called *Input Layer*, a number of hidden layers, the *Hidden Layers*, and an output layer, the *Output Layer*. Each of the hidden layers $(\tilde{Y}, Y)$ performs the following calculation: we take all the results from the previous layer, we assign them a weight (a vector W, and a bias b). We then obtain $\tilde{Y}$ and finally apply an activation function $\mathcal{A}$, to obtain Y. Finally, we apply an output function g to the last layer. In general, we can describe the output Y_{n+1} of a given layer $n+1$, relative to the output Y_n of the previous layer n as follows:

$$\begin{cases} \tilde{Y}_{n+1} = W_{n+1} Y_n + b_{n+1} \\ \\ Y_{n+1} = \mathcal{A}_{n+1}\left(\tilde{Y}_{n+1}\right) \end{cases}$$

We then seek to optimize the weights W as well as the biases b, i.e., to optimize:

$$X = (W_1, b_1, ..., W_n, b_n)$$

To minimize a loss function: $\mathcal{L}(X, \omega)$ which describes "the difference between the estimation by the neural network with the parameters X, for the examples ω". This minimization can then be performed, for example, with a stochastic gradient descent algorithm (SGD).

3.4.2 Computational Graph of the Neural Network

In our example defined by the graph in Figure 3.1, we have:

- An *Input Layer*, $Y_0 \in \mathbb{R}^4$;
- A first *Hidden Layer*, $(\tilde{Y}_1, Y_1) \in \mathbb{R}^5$;
- A second *Hidden Layer*, $(\tilde{Y}_2, Y_2) \in \mathbb{R}^7$;
- An *Output Layer*, $O = g(Y_2) \in \mathbb{R}$.

Therefore, we have the computational graph presented in Figure 3.5.

$$\begin{tikzcd}
 & W_1 \arrow[dr] & & & W_2 \arrow[dr] & & & \\
Y_0 \arrow[rr] & & \tilde{Y}_1 \arrow[r, "\mathcal{A}_1"] & Y_1 \arrow[rr] & & \tilde{Y}_2 \arrow[r, "\mathcal{A}_2"] & Y_2 \arrow[r, "g"] & g(Y_2) \\
 & b_1 \arrow[ur] & & & b_2 \arrow[ur] & & &
\end{tikzcd}$$

Figure 3.5: Forward propagation graph of the neural network

$$\begin{tikzcd}
 & W_1 & & & W_2 & & & \\
Y_0 & & \tilde{Y}_1 \arrow[ll] \arrow[ul, "\frac{\partial \tilde{Y}_1}{\partial W_1}"'] \arrow[dl, "\frac{\partial \tilde{Y}_1}{\partial b_1}"] & Y_1 \arrow[l, "\delta\mathcal{A}_1"'] & & \tilde{Y}_2 \arrow[ll, "\frac{\partial \tilde{Y}_2}{\partial Y_1}"'] \arrow[ul, "\frac{\partial \tilde{Y}_2}{\partial W_2}"'] \arrow[dl, "\frac{\partial \tilde{Y}_2}{\partial b_2}"] & Y_2 \arrow[l, "\delta\mathcal{A}_2"'] & g(Y_2) \arrow[l, "\delta g"'] \\
 & b_1 & & & b_2 & & &
\end{tikzcd}$$

Figure 3.6: Backward propagation graph of the neural network

3.4.3 Gradient Calculation of the Loss Function

We will now focus on the backward graph of this neural network as well as on the gradient calculation:

We then have (note that this is a symbolic representation, for rigor it is necessary to employ the Jacobian matrix, see the sidebar 3.7 page 52):

$$\left\{\begin{array}{l}
\dfrac{\partial g}{\partial W_2} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial W_2} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial \tilde{Y}_2} \cdot \dfrac{\partial \tilde{Y}_2}{\partial W_2} \\[2ex]
\dfrac{\partial g}{\partial b_2} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial \tilde{Y}_2} \cdot \dfrac{\partial \tilde{Y}_2}{\partial b_2} \\[2ex]
\dfrac{\partial g}{\partial W_1} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial W_1} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial \tilde{Y}_2} \cdot \dfrac{\partial \tilde{Y}_2}{\partial W_1} \\
= \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial \tilde{Y}_2} \cdot \dfrac{\partial \tilde{Y}_2}{\partial Y_1} \cdot \dfrac{\partial Y_1}{\partial W_1} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial \tilde{Y}_2} \cdot \dfrac{\partial \tilde{Y}_2}{\partial Y_1} \cdot \dfrac{\partial Y_1}{\partial \tilde{Y}_1} \cdot \dfrac{\partial \tilde{Y}_1}{\partial W_1} \\[2ex]
\dfrac{\partial g}{\partial b_1} = \dfrac{\partial g}{\partial Y_2} \cdot \dfrac{\partial Y_2}{\partial \tilde{Y}_2} \cdot \dfrac{\partial \tilde{Y}_2}{\partial Y_1} \cdot \dfrac{\partial Y_1}{\partial \tilde{Y}_1} \cdot \dfrac{\partial \tilde{Y}_1}{\partial b_1}
\end{array}\right.$$

It is then possible to compute the quadruplet: $\left[\frac{\partial g}{\partial W_1}, \frac{\partial g}{\partial b_1}, \frac{\partial g}{\partial W_2}, \frac{\partial g}{\partial b_2}\right]$

which gives us the gradient of g with respect to $X = (W_1, b_1, W_2, b_2)$. We can then optimize g with respect to X using gradient descent.

Remark 3.12. *Note that in reality, we mainly need the derivative of the loss function $\mathcal{L}$ in* (3.13)*; it directly results from the derivative of g. Moreover, the construction of the loss function takes into account both objectives (reproduction of known results) as well as other considerations, for example generalization power; indeed, overfitting must be prevented. To address this issue, we can introduce a penalty in the weight update at each iteration of gradient descent, or perform dropout, i.e., temporarily remove a neuron to force the algorithm not to overfit.*

Remark 3.13. *The gradient expressed above corresponds to the proposition shown earlier: each of the partial derivatives can be expressed as a sum.*

To know more 3.14 (Stochastic Gradient Descent)**.** *We recall the formula for gradient descent for a multi-dimensional function $\mathcal{F} : \mathbb{R}^p \to \mathbb{R}$, with gradient $\nabla_x \mathcal{F}$:*

$$x_{n+1} = x_n - \gamma_n \nabla_x \mathcal{F}(x_n) .$$

Under certain assumptions (convexity, γ_n, ...), the sequence $(x_n)_{n\geq 1}$ converges to the nearest local minimum.
Here, γ_n is also called the "learning rate" in the case of statistical learning. In the case of high dimensions and when the function $\mathcal{F}$ has the form of an average $\mathcal{F} = \mathbb{E}_\omega F(x, \omega)$ as in equation (3.13)*, computing the gradient at a point becomes very costly (and obtaining the average requires an expensive empirical average to calculate); this is where stochastic gradient descent (SGD) is used. We then take, among the different gradients $\nabla_x F(x, \omega)$, one value (or several, but in a small*

number) randomly selected.

$$x_{n+1} = x_n - \gamma_n \nabla_x F\left(x_n, \omega_n\right).$$

Several stochastic gradient descent algorithms combine different techniques (for learning rate variation as well as stochastic descent).

3.5 Example 2: Control Problem

We now place ourselves in the context of the SIR system, seen previously. However, we will start with a more general presentation by considering that we use the Explicit Euler method to solve $x' = f(t, x(t), u(t))$ with the function $F(x(T))$ to optimize.

3.5.1 Computational Graphs

We solve this problem through the analysis of its computational graphs. Since we use the Explicit Euler method the direct computational graph is the one in the top of the figure 3.7. Of course, this graphs represents only a part of the whole graph. The complete direct graph will include all time steps and is presented in Figure 3.8 for $N = 2$.

Using the usual techniques, we can compute subsequently the backward computational graph as in figure 3.7 (bottom).

3.5.2 Construction of a Discrete Version of an Euler-Lagrange Procedure

The direct and backward computational graphs allow to se the whole procedure as a discrete version of an Euler-Lagrange framwork.

Consider a general situation when $x(t)$ satisfies the ODE: $x'(t) = f(t, x(t), u(t))$ At the discrete level, the EE scheme

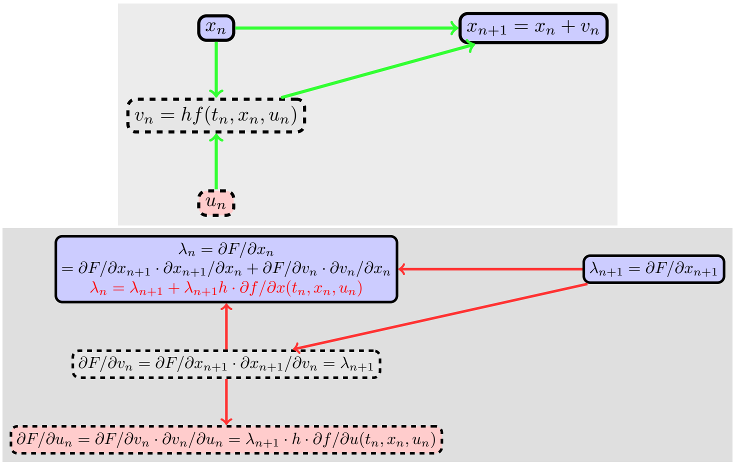

x_n
$x_{n+1} = x_n + v_n$
$v_n = hf(t_n, x_n, u_n)$
u_n
$\lambda_n = \partial F/\partial x_n$
$= \partial F/\partial x_{n+1} \cdot \partial x_{n+1}/\partial x_n + \partial F/\partial v_n \cdot \partial v_n/\partial x_n$
$\lambda_n = \lambda_{n+1} + \lambda_{n+1} h \cdot \partial f/\partial x(t_n, x_n, u_n)$
$\lambda_{n+1} = \partial F/\partial x_{n+1}$
$\partial F/\partial v_n = \partial F/\partial x_{n+1} \cdot \partial x_{n+1}/\partial v_n = \lambda_{n+1}$
$\partial F/\partial u_n = \partial F/\partial v_n \cdot \partial v_n/\partial u_n = \lambda_{n+1} \cdot h \cdot \partial f/\partial u(t_n, x_n, u_n)$

Figure 3.7: Description of the "forward" mode (top image) and "backward" mode (bottom image) for a control problem; the schema obtained for the adjoint state λ_n is an Explicit Euler type discretization of (3.16). Although this seems to correspond to Implicit Euler, it should be remembered that (3.16) is solved in reverse in time, which means that for example λ_{n+1} is known before λ_n.

can be written $x_{n+1} = x_n + hf(t_n, x_n, u_n)$. The computational graph reads, for $\lambda_n = \frac{\partial F}{\partial x_n} =: \delta x_n$:

$$\lambda_n = \lambda_{n+1} + \lambda_{n+1} \frac{\partial f}{\partial x}(t_n, x_n, u_n) h \tag{3.14}$$

$$\lambda_N = \frac{\partial F}{\partial x_N} =: \delta x_N. \tag{3.15}$$

The peculiarity of this formula is that we know the end, namely λ_N, and we must construct the other values λ_n. This leads us to view $\lambda(t)$ as a solution of a **backward-in-time** equation and thus we find ourselves solving, by an Explicit Euler method, the equation:

$$\lambda'(t) = -\frac{\partial f}{\partial x}(t, x(t), u(t)) \lambda(t). \tag{3.16}$$

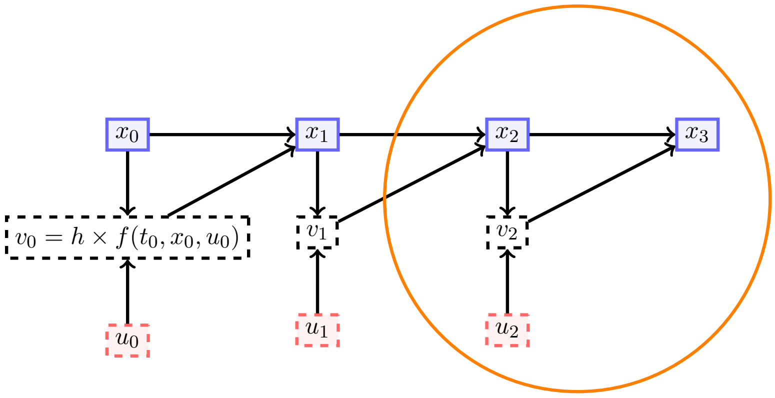


Figure 3.8: Complete direct graph of the SIR system for $n = 2$; the orange circle represents the part that we had previously plotted in 3.7.

We also obtain a formula for the derivative:

$$\frac{\partial F}{\partial u_n} = \lambda_{n+1} h \frac{\partial f}{\partial u_n}(t_n, x_n, u_n). \tag{3.17}$$

In summary, the computational graph shows us that the derivative can be obtained with the help of solving the ODE (3.16) involving $\lambda(t)$, which will be called the adjoint state. This conclusion **is independent** of the numerical scheme used initially to solve the ODE !!

It is also important to note that the cost of this method is just 2 times the cost of calculating F, and thus independent of the number N of u_n values to be optimized (to be compared with $2N$ evaluations that would be necessary if using a finite difference formula like (3.3)).

To know more 3.15. *The relation* (3.17) *is consistent with a formulation of the problem as an Euler-Lagrange type minimization; it is not a proof, rather a verification. The minimization can be written as* $\min_{x'=f(t,x,u)} F(x(T))$. *We apply the Lagrange mul-*

tiplier method to the function $F(x(t))$ and introduce G:

$$G(x,u,\lambda) = F(x(T)) - \int_0^T (x' - f(t,x,u))\lambda(t)dt,$$

where $\lambda(t)$ is the Lagrange multiplier. The Euler Lagrange method prescribes to set to zero the partial derivatives of G with respect to λ, x and u. As expected, the derivative with respect to the multiplier λ allows us to obtain the constraint:

$$\frac{\partial G}{\partial \lambda(t)} = 0 \Leftrightarrow x' = f(t,x,u).$$

The derivative with respect to the Lagrange multiplier $\lambda(t)$ will allow us to link our computational graph to the Euler-Lagrange procedure:

$$\begin{aligned}\frac{\partial G}{\partial x(t)} &= -\frac{\partial}{\partial x(t)}\int_0^T \lambda(x' - f(t,x,u))dt \\ &\overset{IPP}{=} -\frac{\partial}{\partial x(t)}\left[[\lambda x]_0^T - \int_0^T x\lambda' - \lambda f dt\right] \\ &= \lambda' - \lambda\frac{\partial f}{\partial x}(t,x(t),u(t)).\end{aligned}$$

We find equation (3.16) *again. Thus, the computational graph allows us to obtain a discrete version of the Euler-Lagrange procedure for our control variable. The derivative with respect to $x(T)$ leads to find that $\lambda(T)$ should be set equal to $\frac{\partial F(x(T))}{\partial x(T)}$.*
Finally, the derivative of G with respect to the control $u(t)$ allows to find the gradient of the whole functional with respect to the control as in the exercise 3.3 :

$$\frac{\partial F}{\partial u(t)} = \frac{\partial G}{\partial u(t)} = \partial_u f(t,x(t),u(t))\lambda(t). \qquad (3.18)$$

To this should be added any derivative involving directly u : for instance $\frac{\beta}{2}\int_0^T u(t)^2 dt$ would contribute a term $\beta u(t)$.

3.5.3 Problem Reminder

Let's now apply the described procedure to our situation of the SIR model. The problem can be represented by a system, which we call SIR. The system is of the form $x' = f(t, x(t), u(t))$

$$\begin{cases} \dot{S} = -\beta SI/N_p \\ \dot{I} = \beta SI/N_p - \gamma I \\ \dot{R} = \gamma I \end{cases}$$

with $u(t) = \beta(t) = \beta$ the control and $x = (S, I, R)$ the state. The goal is to minimize the cost of the confinement $(\int_0^T c(\beta(t))dt)$ and the number of infected people $(S(0) - S(T))$ during the period [0,T]. In other words, we seek:

$$\min_{\beta} S(0) - S(T) + \int_0^T c(\beta(t))dt.$$

To find the derivative of $\int_0^T c(\beta(t))dt$ with respect to β we use that (assuming $c(\cdot)$) is C^1 class) for any variation $\delta\beta \in L^2[0,T]$:

$$\int_0^T c(\beta(t) + \delta\beta(t))dt = \int_0^T c(\beta(t))dt \\ + \langle c'(\beta(\cdot)), \delta\beta(\cdot)\rangle_{L^2[0,T]} + o(\|\delta\beta\|_{L^2[0,T]}). \quad (3.19)$$

This relation can be proved easily using the Taylor expansion for c; by the definition of the Frechet derivative in $L^2[0,T]$ we obtain thus :

$$\frac{\partial}{\partial\beta}\left[\int_0^T c(\beta(t))dt\right] = c'(\beta(\cdot)). \quad (3.20)$$

On the other hand, the epidemic size $S(0) - S(T)$ is less straightforward to differentiate because β does not appear explicitly, although it is involved in this function.

The methodology presented in 3.15 can be applied ; we introduce the adjoint states λ and μ solution of :

$$\lambda'(t) = \beta(t)I(t)(\lambda(t) - \mu(t))/N_p, \qquad \lambda(T) = 1, \tag{3.21}$$

$$\mu'(t) = \beta(t)S(t)(\lambda(t) - \mu(t))/N_p + \gamma\mu(t), \qquad \mu(T) = 0. \tag{3.22}$$

Then $\frac{\partial}{\partial\beta}S(T) = -SI(\lambda - \mu)/N_p$ (the equality holds as functions of t).

3.6 Theoretical Appendix: Graph Theory

We will introduce some notions of graph theory.

Definition 3.16 (Graph). *A (directed /oriented) graph is a pair $G = (V, E)$ consisting of:*

- *V a set of vertices;*
- *E a set of edges, $E \subseteq V^2$, with $(a, b) \in E$ meaning that there is an edge between a and b. If the graph is considered oriented then the edge is* **from** *a* **to** *b.*

Example 3.17. *See figure 3.9 for an example of a directed graph.*

If the graph is oriented, the edge (a, b) is not the same as (b, a) (if $a \neq b$); note the it is possible for an edge/arc to point to the node from which it originates i.e. (x, x).

Remark 3.18. *An undirected graph is a pair $G = (V, E)$ where $E \subseteq \{(x, y), (x, y) \in V^2, x \neq y\}$ is a set of edges that connect two different nodes together. An arc is thus an edge with a direction. We speak of a path from one node to another to refer to a consecutive sequence of edges connecting the two nodes.*

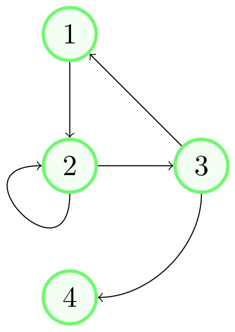


Figure 3.9: Example of a oriented graph

Definition 3.19 (Connected Graph). *A graph is said to be connected if it is in one piece, i.e., for any pair of nodes $v_1, v_2 \in V$ there exists a path from node v_1 to node v_2.*

Example 3.20. *See figures 3.10 and 3.11 for an example of a connected or not connected graph.*

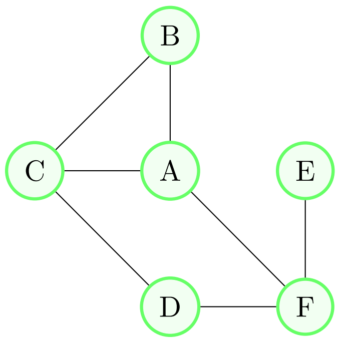


Figure 3.10: Connected graph

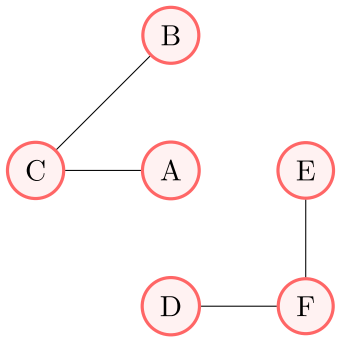


Figure 3.11: Non-connected graph

Definition 3.21. ***Graph without cycles.*** *A graph without cycles is a graph that contains no path that has the same starting and ending node.*

The graph in Figure 3.10 has several cycles (notably $\{A, F, D, C\}$), while the one in Figure 3.11 has none.

Definition 3.22 **(Node Degrees).** *Given an arc* $\{x, y\}$ *connecting nodes* x *and* y*, we call the starting node* x *the* ***source*** *and the ending node* y *the* ***target****. We then define:*

- *the* ***out-degree*** *of a node as the number of arcs that have this node as the source;*
- *the* ***in-degree*** *of a node as the number of arcs that have this node as the target.*

A graph thus has as many in-degrees as out-degrees.

Example 3.23. *In the graph of Figure 3.9:*

- *the out-degree of node* $\{2\}$ *is 2, the out-degree of node* $\{4\}$ *is zero.*
- *the in-degree of node* $\{3\}$ *is 1, the in-degree of node* $\{2\}$ *is 2.*
- *There are a total of 5 in-degrees and 5 out-degrees, which confirms our definition: there are as many in-degrees as out-degrees.*

3.7 Exercises

Exercise 3.1 (**Finite differences of orders 3 and 4**).

1. *Show that*

$$f'(x) = \frac{2f(x+3h) - 9f(x+2h) + 18f(x+h) - 11f(x)}{6h} + O(h^3). \tag{3.23}$$

2. *Find a third-order formula of the form:*

$$f'(x) = \frac{\alpha f(x+2h) + \beta f(x+h) + \gamma f(x) + \delta f(x-h)}{h} + O(h^3). \tag{3.24}$$

3. *Show that*

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4). \tag{3.25}$$

Exercise 3.2 (**Reminder : Euler-Lagrange multipliers method**).
Find the minimum of the function $x + y$ under the constraint $x^2 + y^2 = 1$ using the method of Euler-Lagrange multipliers.

Exercise 3.3 (**Euler-Lagrange multipliers: finding the derivative**).
Let $f, g : \mathbb{R}^2 \to \mathbb{R}$ be C^2 functions. Suppose that the equation $g(x, y) = 0$ has a unique solution $y = \mathcal{Y}(x)$ for each given x, with $\mathcal{Y}$ being C^1. Also suppose that for all x: $\nabla_y g(x, \mathcal{Y}(x)) \neq 0$. Let $F : \mathbb{R} \to \mathbb{R}$ where $F(x) = f(x, \mathcal{Y}(x))$. We assume $\mathcal{Y}(\cdot)$ is difficult to obtain and we want to compute the gradient $\nabla_x F(x)$ without using $\mathcal{Y}(\cdot)$ too many times. Let $L(\lambda, x, y) = f(x, y) + \lambda g(x, y)$.

1. *Show that for every x there exists λ_x such that :*

$$\nabla_y L(\lambda_x, x, \mathcal{Y}(x)) = \nabla_\lambda L(\lambda_x, x, \mathcal{Y}(x)) = 0. \tag{3.26}$$

2. *Show that : $\nabla_x F(x) = \nabla_x L(\lambda_x, x, \mathcal{Y}(x))$.*

3. *Generalize for functions $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$.*

Exercise 3.4 (example of computational graph)**.**

1. *Write the computational graph for the following computation: $z = x + \sin(x \cdot y) + x^2$. Then write the backward graph for computing the derivatives of z with respect to x and y. Explicitly calculate the values of these two graphs for $x = 2$, $y = \pi/6$.*
2. *Same for the function $(x, y, z) \mapsto xyze^{x^2+yz}$.*

Exercise 3.5 (Backward mode IE, cf figure 3.7)**.**

1. *Find the formulas for forward and backward mode for a control problem as in figure 3.7 but with an Implicit Euler scheme.*
2. *Similarly, if the forward graph implements a general RK scheme, find the "adjoint" scheme, i.e., the one used by the backward propagation of the adjoint state. What do you observe?*

Exercise 3.6 (computational graph of projection)**.** *Let $P = \{(x, y, z)^T \in \mathbb{R}^3$ such that $x + y + z = 1\} \subset \mathbb{R}^3$ and $\Pi : \mathbb{R}^3 \to P$ defined for any vector $v = (v_1, v_2, v_3)^T \in \mathbb{R}^3$ by $\Pi(v) = v - \lambda(v) \cdot (1, 1, 1)^T$ where $\lambda(v) \in \mathbb{R}$ is the only real number such that $\Pi(v) \in P$. Let $g : \mathbb{R}^3 \to \mathbb{R}$ be a C^1 function.*

1. *Find the formula for λ_v as a function of v;*
2. *Write the direct computational graph that computes $v \mapsto g(\Pi(v))$;*
3. *Write the backward computational graph that computes the derivatives of $g(\Pi(v))$ with respect to the inputs (components of v).*

4. *Compute the derivatives for* $v = (1, 2, 3)^T$ *and* $g(x, y, z) = x^2 + yz - 2$.

Reminder: for any vector a, a^T *denotes its transpose.*

Exercise 3.7 (computational graph of softmax/cross entropy)**.** *Compute the loss function for the case of cross-entropy loss after a final "softmax" layer (output in* $\mathbb{R}^3$*) and architecture FC/ReLU* $4 - 5 - 7 - 3$*; compute its derivatives. A graphical description in figure 3.12.*

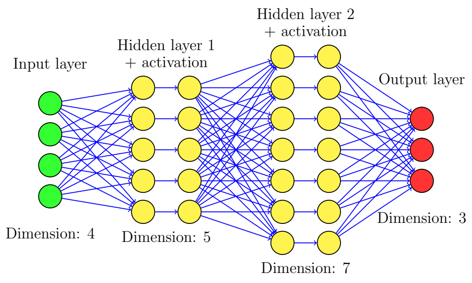


Figure 3.12: The neural network in exercise 3.7 page 69.

Exercise 3.8 (("Layer Normalization"))**.** *A part of the work of the so-called "Layer Normalization" layer is the computation of the standard deviation of a data sample* $x = (x_1, ..., x_n)$*. It is done as follows: first, calculate the empirical mean* $\bar{\mu} = (\sum_{k=1}^n x_k)/n$*, then the empirical variance* $v = \sum_{k=1}^n (x_k - \bar{\mu})^2/(n-1)$ *and finally the estimate of the standard deviation* $\bar{\sigma} = \sqrt{v}$*.*

1. *Draw the direct computational graph corresponding to the above calculations; for each node in the graph, explicitly specify, as in the lecture, the inputs/outputs and the operation performed by the node.*

2. *Similarly, draw the backward computational graph for the derivatives of the final output with respect to the inputs;*

3. *Compute the derivatives for* $n = 3$, $x = (1, 2, 4)$ **with the help of the direct and backward computational graphs**.

3.8 Automatic differentiation lab

Exercise 3.9. *Implement exercise 3.5 for the case of the SIR model in section 3.1.2. Take the values* $T = 150$,$N = 150$, $h = T/N$ $S(0) = 10^6$, $I(0) = 10$, $R(0) = 0$, $N_{total} = S(0) + I(0) + R(0)$ $\beta = 0.5/N_{total}$, $\gamma = 1/3$, $c(\beta) = c_0/\beta$ *with* $c_0 = 10^{-2}$.

Exercise 3.10. *Build and train a neural network with dense layers as described in section 3.1.3.*

Chapter 4

Stochastic Differential Equations (SDEs)

4.1 Background on Brownian motion, martingales, integrals and stochastic processes, Itô's formula

For a review of stochastic calculus, see the notes from [7] from which these results and exercises 4.3, 4.1 are extracted.

4.1.1 Brownian motion: definition

Definition 4.1 (real Brownian motion). *Let $B = B_t, t \geq 0$ be a process defined on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$ equipped with the natural (completed) filtration of the process B, denoted $\mathbb{F}$, such that:*

1. *B is a process with continuous trajectories;*
2. *B is a process with independent increments, meaning that for all $0 \leq s \leq t$, the random variable $B_t - B_s$ is independent of $\mathcal{F}_s$;*
3. *for all $0 \leq s \leq t$, the random variable $B_t - B_s$ follows the normal distribution $\mathcal{N}(0, t-s)$.*

If additionally $B_0 = 0$ (or $B_0 \neq 0$), we say that B is a standard (or non-standard) Brownian motion.

When the filtration $\mathbb{F}$ is given a priori, an $\mathbb{F}$-adapted process that satisfies the above conditions is called an $\mathbb{F}$-Brownian motion.

Throughout, unless explicitly stated otherwise, we assume $B_0 = 0$.

Remark 4.2. *Let $B = B_t,\ t \geq 0$ be an $\mathbb{F}$-Brownian motion on the filtered probability space $(\Omega, \mathcal{F}, \mathbb{P}; \mathbb{F})$. Then B is a Gaussian process with mean function $e_B(t) = 0$ and covariance operator $K_B(s,t) := cov(B_s, B_t) = min(s,t)$.*

We denote by B or W the Brownian motions.

4.1.2 Quadratic variation

Definition 4.3. *Let $t > 0$ be a given real number and $\Delta := t_0 = 0 \leq t_1 \leq \cdots \leq t_n = t$ be a subdivision of the interval $[0,t]$. The module of the subdivision Δ is denoted by $|\Delta|$ and defined as $|\Delta| := \sup_i |t_i - t_{i-1}|$. For a process X_t, we denote*

$$V^{(2)}t(X, \Delta) := \sum_{i=1}^{n} |X(t_i) - X(t_{i-1})|^2 . \tag{4.1}$$

The function $t \mapsto V_t^{(2)}(X) := \overline{\lim}_{|\Delta|\to 0} V_t^{(2)}(X, \Delta) < \infty$ is called the quadratic variation of X.

Proposition 4.4 (Quadratic variation of Brownian motion)**.** *Let B be a Brownian motion on the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Then, for all $t \geq 0$,*

$$\lim_{|\Delta|\to 0} \mathbb{E}\left[|V_t^{(2)}(B, \Delta) - t|^2 \right] \quad = \quad 0. \tag{4.2}$$

This can also be written as $V_t^{(2)}(B, \Delta) \xrightarrow[|\Delta|\to 0]{L^2} t$. We say that the quadratic variation over $[0,t]$ of the Brownian motion exists in L^2 and $V_t^{(2)}(B) = t$.

Intuition 4.5. *The quadratic variation sums the squares of the oscillations of the Brownian motion. As the time interval becomes smaller and smaller, we obtain a sum of independent increments which will converge to its average. This can be formalized by studying the χ^2 variables representing the quadratic variation of the Brownian motion.*

4.1.3 Integration of processes in $\mathcal{L}([0,T])$

We consider the sets $\mathcal{L}(\Omega,\mathcal{F},\mathbb{P};\mathbb{F};[0,T])$ (simply denoted as $\mathcal{L}([0,T])$) and $\mathcal{L}^2(\Omega,\mathcal{F},\mathbb{P};\mathbb{F};[0,T])$ (simply denoted as $\mathcal{L}^2([0,T])$) defined by :

$$\mathcal{L}([0,T]) := \left\{(H_t)_{0\le t\le T},\ \mathbb{F}\text{-adapted process } \Big| \left[\int_0^T H_u^2 du\right] < \infty\ \mathbb{P}-\text{p.s.}\right\}.$$

$$\mathcal{L}^2([0,T]) = \left\{\{H_t,\ 0\le t\le T\},\ \mathbb{F}\text{-adapted process } \Big| \mathbb{E}\left[\int_0^T H_u^2 du\right] < \infty\right\}.$$

Theorem 4.6 (Stochastic integral)**.** *There exists a unique linear mapping $\mathcal{I}$ that maps any process $H \in \mathcal{L}([0,T])$ to a process $\mathcal{I}[H]$ with continuous trajectories on $[0,T]$, such that if H is continuous and locally bounded and Δ^n is a sequence of subdivisions of $[0,t]$ with $|\Delta^n| \to 0$, then (Riemann-Itô sum property):*

$$\mathbb{P} - \lim_{n\to\infty} \sum_{t_k\in\Delta^n} H_{t_k}(B_{t_{k+1}} - B_{t_k}) = \int_0^t H_s dB_s. \qquad (4.3)$$

If additionally $H \in \mathcal{L}^2([0,T])$, then $\mathcal{I}[H]$ is a martingale.

Definition 4.7. *For $H \in \mathcal{L}([0,T])$, the process $\mathcal{I}[H]$ is called the stochastic integral or Itô integral of H with respect to the Brownian motion B. We denote $\int_0^t H_s dB_s := \mathcal{I}[H]_t$.*

Example: exercise 4.1.

4.1.4 Itô Process

Definition 4.8 (Itô Process)**.** *A stochastic process X is called an Itô process if it can be written in the form*

$$X_t = X_0 + \int_0^t \alpha_u du + \int_0^t H_u dB_u \tag{4.4}$$

where X_0 is $\mathcal{F}_0$-measurable, $\alpha_t, t \in \mathbb{R}_+$ and $H_t, t \in \mathbb{R}_+$ are two $\mathbb{F}$-adapted processes satisfying the integrability conditions

$$\int_0^T |\alpha_u| du < \infty \ \mathbb{P} - a.s. \text{ and } \int_0^T |H_u|^2 du < \infty \ \mathbb{P} - a.s. \tag{4.5}$$

It is also denoted differentially as: $dX_t = \alpha_t dt + H_t dB_t$.

Warning 4.9. *The differential form is* **just a notation** *resulting from the identity $\int_a^b dX_t = X_b - X_a$. The Ito process X is in general not differentiable as suggested by the particular case $\alpha = 0$, $H = 1$ where we recover the Brownian motion.*

From now on, we will denote $\mathfrak{I}$ as the set of Itô processes and $\mathfrak{I}^2$ as the subset of Itô processes, $X_t = X_0 + \int_0^t \alpha_u du + \int_0^t H_u dB_u \in \mathfrak{I}$ such that:

$$\mathbb{E}\left[\int_0^T |\alpha_u|^2 du\right] < \infty \text{ and } \mathbb{E}\left[\int_0^T |H_u|^2 du\right] < \infty;. \tag{4.6}$$

4.1.5 Itô Formula

Intuition 4.10. *The exercise 4.1 page 92 shows the following equality: $\int_0^T 2B_s dB_s = B_T^2 - T$, which can also be written differentially as: $dB_s^2 = 2B_s dB_s + ds$. Therefore, the usual composite derivative formula $df(x) = f'(x)dx$ does not apply to $f(x) = x^2$ and $x = B_s$ because the term $-T$ appears. Let's take a closer look: the infinitesimal increment between t and $t + \Delta t$ of an Itô process is of the order $\alpha_t \Delta t$ for the continuous part and $H_t \sqrt{\Delta t}\mathcal{N}(0,1)$ for the Brownian part. For small Δt, it is clearly the Brownian part that dominates. Let's make a* **formal** *calculation for a function $f(x,y)$ with $x = t$, $y = \sqrt{t}$ in the limit of small t:*

$$f(t,\sqrt{t}) = f(0,0) + \frac{\partial f}{\partial y}(0,0)\sqrt{t} + \frac{\partial f}{\partial x}(0,0)t$$
$$+\frac{1}{2}\frac{\partial^2 f}{\partial x^2}t^2 + \frac{\partial^2 f}{\partial x \partial y}t\sqrt{t} + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}\sqrt{t}^2 + o(t^2 + \sqrt{t}^2).$$

To take into account all terms of order less than or equal to t, it is necessary to include the term $\frac{1}{2}\frac{\partial^2 f}{\partial y^2}\sqrt{t}^2$, that is to say, to write:

$$f(t,\sqrt{t}) = f(0,0) + \frac{\partial f}{\partial y}(0,0)\sqrt{t} + \frac{\partial f}{\partial x}(0,0)t$$
$$+\frac{1}{2}\frac{\partial^2 f}{\partial y^2}t + o(t). \qquad (4.7)$$

It is this additional term that constitutes the novelty and must be taken into account. It comes from the non-differentiability of the Brownian motion, which also leads to the non-differentiability of $t \mapsto f(t,\sqrt{t})$ at zero, and therefore the presence of a term of order $\sqrt{t}$ and terms $\sqrt{t}^2$.

Theorem 4.11 (Itô's formula)**.** *Let $f : \mathbb{R}_+ \times \mathbb{R} \to \mathbb{R}$ be* *a $C^{1,2}$ function (i.e., differentiable once in time with continuous derivative and twice in space with continuous second derivative). Let $X \in \mathfrak{I}$ be an Itô process $X_t = X_0 + \int_0^t \alpha_u du + \int_0^t H_u dB_u$. Then, the process $Y_t = f(t, X_t) \in \mathfrak{I}$ and*

$$\begin{aligned} f(t, X_t) =& f(0, X_0) + \int_0^t \frac{\partial f}{\partial t}(u, X_u)du + \int_0^t \frac{\partial f}{\partial x}(u, X_u)dX_u \\ &+ \frac{1}{2}\int_0^t \frac{\partial^2 f}{\partial x^2}(u, X_u)H_u^2 du \end{aligned} \tag{4.8}$$

or, in differential form,

$$df(t, X_t) = \left\{ \frac{\partial f}{\partial t} + \alpha_t \frac{\partial f}{\partial x} + H_t^2 \frac{1}{2}\frac{\partial^2 f}{\partial x^2} \right\}(t, X_t)dt + H_t \frac{\partial f}{\partial x}(t, X_t)dB_t. \tag{4.9}$$

Example: $Y_t = X_t^2$.

4.1.6 Stochastic differential equations

Ǵiven two applications a and b we wonder if there exists an Itô process $X = \{X_t, t \leq 0\}$ whose (unique) decomposition satisfies in differential notation :

$$dX_t = a(t, X_t)dt + b(t, X_t)dB_t.$$

See exercise 4.3 page 93 for examples of such equations appearing in mathematical finance models.

Theorem 4.12. *Let $T > 0$ and $a(\cdot,\cdot), b(\cdot,\cdot) : [0, T] \times \mathbb{R} \to \mathbb{R}$ be measurable functions such that there exist constants $C, L > 0$ satisfying for all $x, y \in \mathbb{R}$, $t \in [0, T]$:*

$$|a(t,x)| + |b(t,x)| \leq C(1 + |x|) \tag{4.10}$$

$$|a(t,x) - a(t,y)| + |b(t,x) - b(t,y)| \leq L|x - y|. \tag{4.11}$$

Let Z be a random variable independent of $\mathcal{F}^\infty = \sigma\{B_s, s \geq 0\}$ and such that

$$\mathbb{E}[Z^2] < \infty.$$

Then the stochastic differential equation

$$X_t = Z + \int_0^t a(s, X_s)ds + \int_0^t b(s, X_s)dB_s, \ t \in [0, T], \quad (4.12)$$

or in differential notation

$$dX_t = a(t, X_t)dt + b(t, X_t)dB_t, \quad (4.13)$$

admits a unique solution $X = \{X_t, t \leq 0\}$ *such that:*

1. X *is continuous with respect to* t*;*
2. X *is adapted to the filtration* $\mathcal{F}_t^Z$ *generated by* $\mathcal{F}_t$ *and* Z*;*
3. $X \in \mathcal{L}^2([0, T])$ *i.e.,:*

$$\mathbb{E}\left[\int_0^T X_t^2 \, dt\right] < \infty.$$

4.1.7 Framework summary

We place ourselves in a framework of a filtered probability space $(\Omega, \mathbb{P}, \mathcal{F}, (\mathcal{F}_t))$ and $(W_t)_{t\geq 0}$ is a Brownian movement which generates the filtration $(\mathcal{F}_t)$.

We call Itô process, a stochastical process $(X_t)_{0\leq t\leq T}$ with values in $\mathbb{R}$ such that:

$$\mathbb{P} - ps \ \ \forall t \leq T : X_t = X_0 + \int_0^t K_s ds + \int_0^t H_s dW_s, \quad (4.14)$$

or equivalently:

$$dX_t = K_t dt + H_t dW_t, \quad (4.15)$$

with X_0 given $\mathcal{F}_0-$measurable, (H_t) and (K_t) adapted to $(\mathcal{F}_t)$, $\int_0^T |K_s| \, ds < \infty$ and $\int_0^T H_s^2 ds < \infty, \mathbb{P} - ps$.

When K and H depend on X we obtain Stochastic Differential Equations (SDE) ; we recalled above the hypothesis under which we can ensure a solution to the SDE exists. The approximation of the solution is the subject of this chapter.

4.2 Euler-Maruyama and Milshtein scheme

The methods that we are going to present concern the resolution of the following type of equations:

$$dX_t = a(t, X_t)dt + b(t, X_t)dW_t, \tag{4.16}$$

or, in full form:

$$X_t = X_0 + \int_0^t a(s, X_s)ds + \int_0^t b(s, X_s)dW_s. \tag{4.17}$$

As in chapter 2 we keep the notations $0 = t_0 < t_1 < \cdots < t_n < \cdots < t_N = T$. Unless otherwise stated, we will assume that $t_{n+1} - t_n$ is constant equal to h.
Euler-Maruyama (E-M) scheme: This scheme proposes to estimate $X(t_n)$ by Y_n verifying:

$$Y_{n+1} = Y_n + a(t_n, Y_n)(t_{n+1} - t_n) + b(t_n, Y_n)(W_{t_{n+1}} - W_{t_n}).$$

Remark 4.13. *When $t_{k+1} - t_k = h$ for all k and if we note $\Delta W_n := W_{t_{n+1}} - W_{t_n}$, $a_n := a(t_n, Y_n)$, $b_n := b(t_n, Y_n)$ then the E-M scheme is written:*

$$Y_{n+1} = Y_n + a_n h + b_n \Delta W_n, \quad Y_0 = X(0).$$

Remark 4.14. *Note that a_n and b_n are $\mathcal{A}_{t_n}$-measurable random variables, where $(\mathcal{A}_t)_{t\geq 0}$ is the filtration generated by $(W_t)_{t\geq 0}$.*

Milshtein scheme: It is given by the relation:

$$Y_{n+1} = Y_n + a_n h + b_n \Delta W_n + \frac{1}{2} b_n b_n' [(\Delta W_n)^2 - h], \quad Y_0 = X(0).$$

To know more 4.15. *Suppose $a(t, X) = \alpha X$ and $b(t, X) = \beta X$, are there any implicit schemes possible ? A suggestion could be: $Y_{n+1} = Y_n + \alpha Y_{n+1} h + \beta Y_{n+1} \Delta W_n$ which means that $Y_{n+1} = \frac{Y_n}{1 - \alpha h - \beta \Delta W_n}$. Now, this is unbounded since if $\xi \sim \mathcal{N}(0, 1)$ then*

$\mathbb{E}(\frac{1}{|\xi|}) = +\infty$, which leads to numerical problems and instabilities. So this will not work this way. It would therefore be better to use: $Y_{n+1} = Y_n + \alpha h Y_{n+1} + \beta Y_n \Delta W_n$; For Milshtein we could use at most an implicit term while keeping this term $\frac{b_n b'_n}{2}(\Delta W_n^2 - h)$ explicit.

4.3 Weak and Strong Consistency

As a reminder, for an ODE, consistency could be written in the form: truncation error is null as $h \to 0$, or $\tau(h) = o(1)$. This motivates the following definition:

Definition 4.16 (Weak Consistency)**.** *A scheme that yields $(Y_n)n \geq 1$ is said to be weakly consistent if:*

$$(W1) \quad \lim_{h\to 0} \mathbb{E}\left[\left(\mathbb{E}\left[\frac{Y_{n+1}-Y_n}{h}\middle|\mathcal{A}_{t_n}\right] - a_n\right)^2\right] = 0.$$

$$(W2) \quad \lim_{h\to 0} \mathbb{E}\left[\left(\mathbb{E}\left[\frac{(Y_{n+1}-Y_n)^2}{h}\middle|\mathcal{A}_{t_n}\right] - b_n^2\right)^2\right] = 0.$$

Theorem 4.17. *Let $T > 0$ and $a(.,.), b(.,.), b'(.,.) : [0,T] \times \mathbf{R} \to \mathbf{R}$ be bounded functions. Then the Euler-Maruyama (E-M) and Milstein (M) schemes defined earlier are weakly consistent.*

Proof. We start with the Euler-Maruyama Scheme (EM) and check first the (W1) condition.

$$
\begin{aligned}
&\mathbb{E}\left[\frac{Y_{n+1}-Y_n}{h}|\mathcal{A}_{t_n}\right]-a_n\\
&=\mathbb{E}\left[\frac{Y_n+a_nh+b_n\Delta W_n-Y_n}{h}|\mathcal{A}_{t_n}\right]-a_n\\
&=\mathbb{E}\left[a_n+b_n\frac{\Delta W_n}{h}|\mathcal{A}_{t_n}\right]-a_n\\
&=a_n+b_n\mathbb{E}\left[\frac{\Delta W_n}{h}|\mathcal{A}_{t_n}\right]-a_n\\
&\ \text{(since } a_n, b_n \text{ are measurable w.r.t. } \mathcal{A}_{t_n})\\
&=b_n\mathbb{E}\frac{[\Delta W_n]}{h}(\text{ since } \Delta W_n\perp\mathcal{A}_{t_n})=0. \qquad (4.18)
\end{aligned}
$$

We now verify (W2):

$$
\begin{aligned}
&\mathbb{E}\left[\frac{(Y_{n+1}-Y_n)^2}{h}|\mathcal{A}_{t_n}\right]-b_n^2=\mathbb{E}\left[\frac{(a_nh+b_n\Delta W_n)^2)}{h}|\mathcal{A}_{t_n}\right]-b_n^2\\
&=\mathbb{E}\left[ha_n^2+2a_n\Delta W_nb_n+b_n^2\frac{(\Delta W_n)^2}{h}|\mathcal{A}_{t_n}\right]-b_n^2\\
&=ha_n^2+2a_nb_n\mathbb{E}[\Delta W_n]+b_n^2\frac{\mathbb{E}[\Delta W_n^2]}{h}-b_n^2\\
&=ha_n^2(\text{ since } \Delta W_n\sim\mathcal{N}(0,h))\xrightarrow{L2}0. \qquad (4.19)
\end{aligned}
$$

We now consider the Milstein Scheme (M) and start with checking (W1). Recall that $\Delta W_n\sim\mathcal{N}(0,h)$. We can write :

$$
\begin{aligned}
&\mathbb{E}\left[\frac{Y_{n+1}-Y_n}{h}|\mathcal{A}_{t_n}\right]-a_n\\
&=\frac{1}{h}\mathbb{E}\left[a_nh+b_n\Delta W_n+\frac{b_nb_n'}{2}(\Delta W_n^2-h)|\mathcal{A}_{t_n}\right]-a_n\\
&=\frac{a_nh}{h}+b_n\frac{1}{h}\underbrace{\mathbb{E}[\Delta W_n]}_{=0}+\frac{1}{2h}b_nb_n'\underbrace{\mathbb{E}[\Delta W_n^2-h]}_{=0}-a_n\\
&=0 \qquad (4.20)
\end{aligned}
$$

To check (W2) we compute :

$$\begin{aligned}
&\mathbb{E}\left[\frac{(Y_{n+1}-Y_n)^2}{h}|\mathcal{A}_{t_n}\right] - b_n^2\\
&= \frac{1}{h}\mathbb{E}\left[(a_n h + b_n\Delta W_n + \frac{b_n b_n'}{2}(\Delta W_n^2 - h))^2|\mathcal{A}_{t_n}\right] - b_n^2\\
&= \mathbb{E}\left[\left(\underbrace{a_n\sqrt{h}}_{O(\sqrt{h})} + b_n\frac{\Delta W_n}{\sqrt{h}} + \frac{b_n b_n'}{2}\left(\frac{\Delta W_n^2 - h}{\sqrt{h}}\right)\right)^2|\mathcal{A}_{t_n}\right] - b_n^2\\
&= \mathbb{E}\left[a_n^2 h + b_n^2\frac{\Delta W_n^2}{h} + \frac{b_n^2 b_n'^2}{4}\frac{(\Delta W_n^2 - h)^2}{h} + 2a_n b_n\sqrt{h}\frac{\Delta W_n}{\sqrt{h}} + \right.\\
&\left. 2a_n\sqrt{h}\frac{b_n b_n'}{2}\frac{\Delta W_n^2 - h}{\sqrt{h}} + \frac{b_n^2 b_n'}{2\sqrt{h}}\Delta W_n\frac{\Delta W_n^2 - h}{\sqrt{h}}|\mathcal{A}_{t_n}\right] - b_n^2\\
&= a_n^2 h + \frac{b_n^2 {b'}_n^2}{4}\mathbb{E}[\frac{(\Delta W_n^2 - h)^2}{h}] + \frac{b_n^2 {b'}_n}{2\sqrt{h}}\underbrace{\mathbb{E}\left[\Delta W_n\frac{\Delta W_n^2 - h}{\sqrt{h}}\right]}_{=0}\\
&= a_n^2 h + \frac{b_n^2 {b'}_n^2}{4}h\underbrace{\mathbb{E}[(N(0,1)^2 - 1)^2]}_{\textit{bounded in terms of } h} = O(h) \xrightarrow[h\to 0]{} 0. \qquad (4.21)
\end{aligned}$$

□

Remark 4.18. *The exact solution satisfies (W1) and (W2), see exercise 4.5 page 94.*

Remark 4.19. *The weak consistency concerns only certain functions depending on the solution, particularly the moments.*

Definition 4.20 (Strong Consistency)**.** *A scheme $(Y_n)_{n\geq 1}$ is said to be strongly consistent if:*
(F1) condition (W1) is satisfied

(F2): $\lim_{h\to 0}\mathbb{E}\left[\frac{1}{h}|Y_{n+1}-Y_n-\mathbb{E}[Y_{n+1}-Y_n|\mathcal{A}_{t_n}]-b_n\Delta W_n|^2\right] = 0$

Theorem 4.21. *Assume a, b, a' and b' are bounded. Then the Euler-Maruyama (E-M) and Milstein (M) schemes are strongly consistent.*

Proof. a) <u>Euler-Maruyama Scheme (E-M)</u>
(F1) is satisfied by the previous theorem.
Verify (F2):
$Y_{n+1} - Y_n - \mathbb{E}[Y_{n+1} - Y_n|\mathcal{A}_{t_n}] - b_n\Delta W_n = a_n h + b_n \Delta W_n - \underbrace{a_n h}_{\mathbb{E}[Y_{n+1}-Y_n|\mathcal{A}_{t_n}]} - b_n\Delta W_n = 0$

b) <u>Milstein Scheme (M)</u>
(F1) is satisfied by the previous theorem.
Verify (F2):
$Y_{n+1} - Y_n - \mathbb{E}[Y_{n+1} - Y_n|\mathcal{A}_{t_n}] - b_n\Delta W_n = a_n h + b_n \Delta W_n + \frac{b_n b_n'}{2}[\Delta W_n^2 - h] - a_n h - b_n\Delta W_n = \frac{b_n b_n'}{2}[\Delta W_n^2 - h]$. We need to show that $\lim_{h\to 0} \mathbb{E}\left[\frac{1}{h}\left(\frac{b_n b_n'}{2}[\Delta W_n^2 - h]\right)^2\right] = 0$. But since b, b' are bounded, we obtain that

$$\lim_{h\to 0} \mathbb{E}\left[\frac{1}{h}\left(\frac{b_n b_n'}{2}[\Delta W_n^2 - h]\right)^2\right] \le \lim_{h\to 0} C_0 h \mathbb{E}\left[\left(\bigg(\underbrace{\frac{\Delta W_n}{\sqrt{h}}}_{\mathcal{N}(0,1)}\bigg)^2 - 1\right)^2\right] = 0 \quad (4.22)$$

where C_0 is a constant and we used the fact that $\mathbb{E}\left[\left(\left(\frac{\Delta W_n}{\sqrt{h}}\right)^2 - 1\right)^2\right]$ is a constant independent of h. □

Remark 4.22. *Strong consistency implies weak consistency, see exercise 4.5 page 94.*

4.4 Weak and Strong Convergence

We use the same notations as before, and in particular in the following two definitions, the time step h satisfies $h = T/N$, where N is the number of time steps to reach a given T (N is an integer). We will write N_h to better represent this dependence.

Definition 4.23 (Strong Convergence)**.** *A scheme* $(Y_n)_{n\geq 1}$ *is said to be strongly convergent of order* $\gamma > 0$ *at time* T *if there exists* $h_0 > 0$ *and* $C > 0$ *independent of* h *such that for all* $h \leq h_0$*:*

$$\mathbb{E}[|X(hN_h) - Y_{N_h}|] \leq Ch^{\gamma} \quad (\text{ with } N_h = [T/h] \in \mathbb{N}). \tag{4.23}$$

Definition 4.24 (Weak Convergence)**.** *The scheme* $(Y_n)_{n\geq 1}$ *is said to be weakly convergent of order* $\beta > 0$ *if there exists* $h_0 > 0$ *and* $C > 0$ *independent of* h *such that for all* $h \leq h_0$*, and for all* $g \in C_p^{2\beta+2}$*:*

$$|\mathbb{E}[g(X(hN_h))] - \mathbb{E}[g(Y_{N_h})]| \leq Ch^{\beta},$$

where $C_p^{2\beta+2}$ *is the set of functions with at most polynomial growth*[1] *of class* $2\beta + 2$*.*

Example 4.25. *The functions* $log(\cdot)$ *and* $sin(\cdot)$ *have at most polynomial growth. The exponential function does not.*

Example 4.26. *Let* $W = (W_t)$ *and* $B = (B_t)$ *be two independent Brownian motions and* X_t *be the exact solution of the equation* $dX_t = 0 \cdot dt + 1 dW_t$*, i.e.,* $X_t = W_t$*. If we propose the numerical scheme* $Y_{n+1} = Y_n + \Delta B_n$*, we obtain* $Y_n = B_{t_n} = B_{nh}$*. Although the processes* W *and* B *are totally independent,* Y_n *perfectly respects the distribution of the variable* W_{t_n}*, so in particular this scheme is weakly convergent to any order* $\gamma \geq 0$*. However, it is not strongly convergent to any order.*

Theorem 4.27 (Convergence of Euler-Maruyama and Milstein Schemes)**.** *The Euler-Maruyama scheme converges strongly of order* $1/2$ *and weakly of order* 1*. The Milstein scheme converges strongly and weakly of order* 1*.*

Proof. This theorem is admitted. □

[1] A function f is said to have at most polynomial growth if there exist $s \in \mathbb{N}$ and $C_s > 0$ such that : $|f(x)| \leq C_s(1 + |x|^s), \ \forall x \in \mathbb{R}$.

4.5 Integral Form Taylor for ODE

In this part, we consider an ordinary differential equation, $X'_t = a(t, X_t)$, whose integral form is $X_t = X_0 + \int_0^t a(s, X_s)ds$. We write t as an index in the previous equation. For any sufficiently regular function f, using the equation for X we obtain:

$$\frac{d}{dt} f(X_t) = a(t, X_t) \frac{\partial}{\partial x} f(X_t) \qquad (4.24)$$

Equation (4.24) can also be written as $f'(X_t) = a \cdot f_x$. This definition of f allows us to write it in integral form

$$f(X_t) = f(X_0) + \int_0^t Lf(X_s)ds, \qquad (4.25)$$

where we have introduced the operator L which, for a function f, acts by $L(f) = a\frac{\partial}{\partial x}(f)$. Let $f(x) = x$; equation (4.25) allows us to write $X_t = X_0 + \int_0^t a(X_s)ds$ because here $L(x) = a\frac{\partial}{\partial x}(x) = a$.

To simplify our calculations, we set $a = a(x)$, so there is no explicit dependence on t. Similarly, it is important to note that (4.25) is valid for any function f, in particular for $f = a$. In this case,

$$a(X_s) = a(X_0) + \int_0^s La(X_{s_2})ds_2$$

By reintroducing (4.5) into the definition of X_t, we obtain

$$\begin{aligned} X_t &= X_0 + \int_0^t \left[a(X_0) + \int_0^s La(X_{s_2})ds_2 \right] ds \\ &= X_0 + ta(X_0) + \int_0^t \int_0^s La(X_{s_2})ds_2 ds. \end{aligned}$$

This process can be repeated as many times as desired, which allows us to define the Taylor formula in integral form:

$$\begin{aligned} f(X_t) &= f(X_0) + \int_0^t Lf(X_s)ds \\ &= ... \\ &= f(X_0) + \sum_{k=1}^{n} \frac{t^k}{k!} L^k f(X_0) \\ &+ \int_0^t \int_0^s \int_0^{s_2} ... \int_0^{s_n} L^{n+1} f(X_{s_n}) ds_n ... ds. \end{aligned} \tag{4.26}$$

By using (4.26) for $a = 1$, $f(x) = x$, we obtain the usual Taylor formula.

4.6 Itô-Taylor Formulas

In this section, we consider a stochastic differential equation $dX_t = a(t, X_t)dt + b(t, X_t)dW_t$. The Itô formula,

$$\begin{aligned} f(X_t) &= f(X_0) \\ &+ \int_0^t \left\{ a(X_s) \frac{\partial}{\partial x} f(X_s) + \frac{1}{2} b(X_s)^2 \frac{\partial^2}{\partial x^2} f(X_s) \right\} ds \\ &+ \int_0^t b(X_s) \frac{\partial}{\partial x} f(X_s) dW_s \end{aligned} \tag{4.27}$$

leads us to introduce two operators:

$$L^0 = a(X_s) \frac{\partial}{\partial x} + \frac{1}{2} b(X_s)^2 \frac{\partial^2}{\partial x^2}, \; L^1 = b(X_s) \frac{\partial}{\partial x}$$

Equation (4.27) can then be written as:

$$f(X_t) = f(X_0) + \int_0^t (L^0 f)(X_s) ds + \int_0^t (L^1 f)(X_s) dW_s \tag{4.28}$$

We want to find an expression for X_t of order 1. Equation (4.28) remains true for any function f, especially for

$f(x) = x$. We thus have the stochastic differential equation in integral form:

$$X_t = X_0 + \int_0^t a(X_s)ds + \int_0^t b(X_s)dW_s$$

By applying (4.28) to the functions a and b, we obtain:

$$\begin{aligned} X_t &= X_0 \\ &+ \int_0^t \left\{ a(X_0) + \int_0^s (L^0 a)(X_\sigma)d\sigma + \int_0^s (L^1 a)(X_\sigma)dW_\sigma \right\} ds + \\ &+ \int_0^t \left\{ b(X_0) + \int_0^s (L^0 b)(X_\sigma)d\sigma + \int_0^s (L^1 b)(X_\sigma)dW_\sigma \right\} dW_s. \end{aligned}$$

Finally, we have

$$X_t = X_0 + \underbrace{a(X_0)\int_0^t ds}_{O(t)} + \underbrace{b(X_0)\int_0^t dW_s}_{O(t^{1/2})} + \mathrm{R}$$

We study the remainder R, in order to estimate the order of its terms.

$$\begin{aligned} R = &\underbrace{\int_0^t \int_0^s (L^0 a)(X_\sigma)d\sigma ds}_{O(t^2)} + \underbrace{\int_0^t \int_0^s (L^1 a)(X_\sigma)dW_\sigma ds}_{O(t^{3/2})} + \\ &+ \underbrace{\int_0^t \int_0^s (L^0 b)(X_\sigma)d\sigma dW_s}_{O(t^{3/2})} + \underbrace{\int_0^t \int_0^s (L^1 b)(X_\sigma)dW_\sigma dW_s}_{O(t)} \end{aligned}$$

We notice that the last term is of order 1. As a reminder, we have

$$\begin{aligned} (L^1 b)(X_\sigma) = (L^1 b)(X_0) &+ \int_0^\sigma (L^0 L^1 b)(X_u)du \\ &+ \int_0^\sigma (L^1 L^1 b)(X_u)dW_u \end{aligned} \tag{4.29}$$

We will keep in (4.29) only the first term, $L^1 b(X_0)$ as we are only interested in terms of order 1. Finally, we obtain the **Itò-Taylor formula of order 1 with remainder** $O(t^{3/2})$:

$$\begin{aligned} X_t &= X_0 + a(X_0)\int_0^t ds + b(X_0)\int_0^t dW_s \\ &\quad + L^1 b(X_0)\int_0^t\int_0^s dW_\sigma dW_s + R_2 \end{aligned} \tag{4.30}$$

Here R_2 is a term of order $t^{3/2}$. We can verify that the remainder R_2 contains only terms of order greater than or equal to 3/2:

$$\begin{aligned} R_2 = &\underbrace{\int_0^t\int_0^s L^0 a(X_\sigma) d\sigma ds}_{O(t^2)} + \underbrace{\int_0^t\int_0^s L^1 a(X_\sigma) dW_\sigma ds}_{O(t^{3/2})} + \\ &+ \underbrace{\int_0^t\int_0^s L^0 b(X_\sigma) d\sigma dW_s}_{O(t^{3/2})} \\ &+ \underbrace{\int_0^t\int_0^s\int_0^\sigma L^0 L^1 b(X_u) du dW_\sigma dW_s}_{O(t^2)} + \\ &+ \underbrace{\int_0^t\int_0^s\int_0^\sigma L^1 L^1 b(X_u) dW_u dW_\sigma dW_s}_{O(t^{3/2})} \end{aligned}$$

We can see that all terms of R_2 are of order greater than 1, so we have indeed found an expression for X_t of order 1.

4.7 Application of Itò-Taylor to the Construction of Numerical Schemes

In this section, we can attempt to approximate the solution of a differential equation through numerical calculation using the Euler-Maruyama method. We will compute the terms appearing in the first-order Ito-Taylor formula in (4.30) $[t_n, t_{n+1}]$:

$$X_{t_{n+1}} = X_{t_n} + a(X_{t_n})\int_{t_n}^{t_{n+1}} ds + b(X_{t_n})\int_{t_n}^{t_{n+1}} dW_s \\ + L^1 b(X_{t_n})\int_{t_n}^{t_{n+1}}\int_{t_n}^{t_{n+1}} dW_\sigma dW_s + R_2$$

By computing the integrals, we obtain:

$$X_{t_{n+1}} = X_{t_n} + a(X_{t_n})(t_{n+1}-t_n) + b(X_{t_n})(W_{t_{n+1}} - W_{t_n}) + O(h), \tag{4.31}$$

which allows us to obtain the Euler-Maruyama scheme that uses the Ito-Taylor expansion with inclusion of all terms of order $\frac{1}{2}$ but only some of the terms of order 1.

Continuing in the Ito-Taylor formula (4.30) with the calculation of $L^1 b(X_{t_n})$; for simplification, we denote $b_n = b(X_{t_n})$. Given the definition of L^1, we then obtain

$$L^1 b(X_{t_n}) = b(X_{t_n})\frac{\partial}{\partial x} b(X_{t_n}), \tag{4.32}$$

and therefore

$$L^1 b(X_{t_n})\int_{t_n}^{t_{n+1}}\int_{t_n}^{s} dW_\sigma dW_s = b_n b_n' \int_{t_n}^{t_{n+1}} (W_s - W_{t_n}) dW_s. \tag{4.33}$$

Note that $\int_0^T W_s dWs = \frac{W_T^2 - T}{2}$. Integrating the previous expression, we get:

$$L^1 b(X_{t_n}) = b_n b_n' \left[\frac{(W_s - W_{t_n})^2 - (s - t_n)}{2}\right]_{t_n}^{t_{n+1}} \tag{4.34}$$

$$= \frac{b_n b_n'}{2}\left[(Wt_{n+1} - W_{t_n})^2 - (t_{n+1} - t_n)\right] \tag{4.35}$$

$$= \frac{b_n b_n'}{2}(\Delta W_n{}^2 - h), \tag{4.36}$$

where $\Delta W_n = W_{t_{n+1}} - W_{t_n}$. We then have the following expression:

$$X_{t_{n+1}} = X_{t_n} + a(X_{t_n})h + b(X_{t_n})(W_{t_{n+1}} - W_{t_n})$$

$$+\frac{1}{2}b_n b_n'(\Delta W_n{}^2 - h) + O(h^{3/2}),$$

which justifies the expression of the Milstein scheme.

4.8 Application to the Evaluation and Delta-Hedging of Derivative Products Options, Black & Scholes Formulas

We will briefly indicate how numerical schemes are used for the evaluation and hedging (delta-hedging) of derivative products. This presentation is provided for completeness, and interested readers will find a much more detailed discussion in [7].

We maintain the previous notations, namely the usual probabilistic framework, (W_t) a Brownian motion, and S_t an underlying asset that satisfies, by assumption, the equation $\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$; we say that we are operating within the framework of a Black-Scholes model.

To know more 4.28. *We do not comment here on the question of whether or not such a model is actually verified in reality; more precisely, reality never exactly matches any model, but some give errors that we are willing to manage while others do not. This is a more general discussion that does not currently fall within the scope of this course but should concern every practitioner.*

Let C_t be the price at time t of a derivative product whose payoff at maturity $T \geq t$ is given by a measurable random variable $\mathcal{F}_T$. For technical convenience, we also assume that $\mathbb{E}[G^2] < \infty$. For example, $G = (S_T - K)_+$ for a European call, while $G = \left(\frac{1}{T}\int_0^T S_t dt - K\right)_+$ for an Asian call. It is known that there exists a probability Q^*, called the *risk-neutral* probability, and $\tilde{W}$ a Q^*-Brownian motion such that

$$\frac{dS_t}{S_t} = rdt + \sigma d\tilde{W}_t, \quad S(0) = S_0 \tag{4.37}$$

and

$$C_t = \mathbb{E}^{Q^*}\left[e^{-r(T-t)}G\,|\mathcal{F}_t\right]. \tag{4.38}$$

Of course, this can also be written as $e^{-rt}C_t = \mathbb{E}^{Q^*}\left[e^{-rT}C_T\,|\mathcal{F}_t\right]$, which recalls a martingale property for the discounting $\tilde{C}_t = e^{-rt}C_t$ of C_t.

To understand why (again, see [7] for a more rigorous presentation), let's consider a self-financing portfolio Π_t that **finances** G, meaning $\Pi_T = G$. It will consist of Δ_t units of the underlying asset and the rest in risk-free product. Of course, due to absence of arbitrage opportunities, it follows that $C_t = \Pi_t$ for all t. By the self-financing condition, we can deduce the evolution of Π_t: $d\Pi_t = \Delta_t dS_t + (\Pi_t - \Delta_t S_t)rdt$ or, recalling the definition of dS_t:

$$d\Pi_t = r\Pi_t dt + \Delta_t S_t[(\mu - r)dt + \sigma dW_t].$$

Now, let's denote $\tilde{\Pi}_t = e^{-rt}\Pi_t$ (discounting of Π_t). Then, by Ito's formula applied to the function $x \mapsto e^{-rt}x$, we can write $d\tilde{\Pi}_t = \Delta_t S_t d[(\mu - r)dt + \sigma dW_t]$. Let's now change the measure and find Q^* equivalent to $\mathbb{P}$ such that $\tilde{W} = \frac{\mu - r}{\sigma}t + W_t$ is a Q^*-Brownian motion. Then we will obtain $d\tilde{\Pi}_t = \Delta_t S_t \sigma d\tilde{W}_t$; thus under the measure Q^* the process $\tilde{\Pi}_t$ is an Ito process without a dt term, hence a martingale, which justifies formula (4.38).

Of course, this reasoning was under the assumption that we can find a portfolio Π_t that finances G, meaning that $\Pi_t = G$. In fact, we can even show this, we will just outline the argument. We will keep the same definition of Q^* (which has nothing to do with the derivative product, it's just a transformation involving the Brownian motion (W_t)).

Let M_t be the martingale defined by $M_t = \mathbb{E}^{Q^*}\left[e^{-rT}G\,|\mathcal{F}_t\right]$ (the fact that (M_t) is a martingale follows from the properties of conditional expectation and the fact that G has a finite second moment).

By the martingale representation theorem (which roughly says that any martingale is an Ito process without a dt term), there exists H_t such that $M_t = \int_0^t H_s d\tilde{W}_s$ or equivalently in integral form $dM_t = H_t d\tilde{W}_t = H_t\left[\frac{\mu-r}{\sigma}dt + dW_t\right]$.

Now we just need to reverse the operations and find that the portfolio Π_t containing $\frac{H_t}{\Delta_t\sigma}$ units of the underlying asset is self-financed and finances G.

To know more 4.29. *Note that the existence of such a portfolio tells us that the risk contained in the derivative S_t can be offset (we say "hedged") by having a variable number Δ_t of units of the underlying asset (often $\Delta_t = \frac{\partial}{\partial S}C_t$). Or, in other words, a portfolio that contains -1 derivative (seller) and $+\Delta_t$ units of the underlying asset S_t will be neutral (we say "delta neutral"), meaning its first-order evolution is deterministic. In particular, if such a portfolio is self-financed, it will start from 0 and end at 0 regardless of the path taken by the underlying asset. This leads us into the realm of "delta-hedging", a procedure used by option sellers who do not want to make directional bets on S_t but just pocket their commission and hedge against all other risks.*

From a numerical perspective, to apply (4.38) and compute the price of a derivative product (let's take a European call option with strike K as an example, i.e., $G = (S_T - K)_+$ and $t = 0$):

1. simulate a large number M of realizations S_t^m, $m = 1, ..., M$ of S_t (which follows the equation (4.37)): this

can be done either by a direct formula after simulating $\tilde{W}$ or by numerical schemes such as Euler-Maruyama or Milstein;

2. calculate the empirical mean $\frac{1}{M}e^{-rT}\sum_{m=1}^{M}(S_T^m - K)_+$ which will be the option price at time 0.

4.9 Exercises

Throughout the following, we consider a stochastic differential equation (SDE)

$$dX_t = a(t, X_t)dt + b(t, X_t)dW_t \tag{4.39}$$

The coefficients a and b that satisfy, at a minimum, the conditions of the existence theorem of an Itô process, namely:
- a, b are adapted to the filtration $\mathcal{A}_t$ of the process X_t
- $\int_0^T |a_s|ds \leq \infty$ a.s., $\int_0^T |b_s|^2 ds \leq \infty$ a.s.

Reminder: the time step here is equal to h and we denote $t_n = nh$. Numerical schemes will provide approximations Y_n of X_{t_n}.
When there is no ambiguity, we denote $a_n = a(t_n, Y_n)$, $b_n = b(t_n, Y_n)$.

Exercise 4.1 (Riemann sums, cf. theorem 4.6, page 73)**.** *Let $\Delta : t_0 = 0 < t_1 < ... < t_N = T$ be a partition of* $[0, T]$.

1. *Calculate the L^2 limit (if it exists) of the Riemann-type sum*

$$S_1 = \sum_{k=0}^{N-1} B_{t_k}\left(B_{t_{k+1}} - B_{t_k}\right); \tag{4.40}$$

as $|\Delta| \to 0$.

2. *Calculate the L^2 limit (if it exists) of the Stratonovich-type sum*

$$S_2 = \sum_{k=0}^{N-1} B_{\frac{t_k+t_{k+1}}{2}}\left(B_{t_{k+1}} - B_{t_k}\right); \tag{4.41}$$

as $|\Delta| \to 0$.

Exercise 4.2 (Itô's formula)**.** *For this exercise,* $(X_t)_{t\geq 0} \in \mathcal{I}^2$ *and we will also admit that* $(Y_t)_{t\geq 0}$ *(defined below) is also in* $\mathcal{I}^2$.

1. *Let* $Y_t = \log(X_t)$. *Calculate* dY_t *knowing that* $dX_t = \mu X_t dt + \sigma X_t dB_t$.
2. *Let* $dX_t = a \cdot dt + b \cdot dB_t$. *Calculate* dY_t *where* $Y_t = e^{X_t}$.

Exercise 4.3 (Stochastic Differential Equations for Some Financial Models)**.** *Let* $W = \{W_t, t \geq 0\}$ *be a standard Brownian motion.*

1. *log-normal (Black-Merton-Scholes) model: let* $\mu, \sigma \in \mathbb{R}$. *Prove, using a result from the course, that the following SDE:*

   $$dS_t = \mu S_t dt + \sigma S_t dW_t,\ S(0) = S_0 \in \mathbb{R}, \tag{4.42}$$

   has a unique solution. Show that this solution is

   $$S_t = e^{(\mu - \sigma^2/2)t + \sigma W_t} S_0. \tag{4.43}$$

2. *Vasicek model: let* $\alpha, \beta, \sigma \in \mathbb{R}$, $\alpha \neq 0$, $\sigma > 0$. *Prove, using a result from the course, that the following SDE:*

   $$dr_t = \alpha(\beta - r_t)dt + \sigma dW_t, r(0) = r_0 \in \mathbb{R}, \tag{4.44}$$

   has a unique solution. Show that this solution is

   $$r_t = r_0 e^{-\alpha t} + \beta(1 - e^{-\alpha t}) + \sigma e^{-\alpha t} \int_0^t e^{\alpha s} dW_s. \tag{4.45}$$

3. *Cox-Ingersoll-Ross (CIR) model: let* $\alpha, \beta, \sigma \in \mathbb{R}$, $\alpha \neq 0$, $\sigma > 0$. *Determine whether the result from the course can be used to prove that the following SDE has a unique solution:*

   $$dr_t = \alpha(\beta - r_t)dt + \sigma\sqrt{|r_t|}dW_t,\ r(0) = r_0 \geq 0. \tag{4.46}$$

Exercise 4.4. *(weak consistency) Suppose a is bounded: $|a(t,x)| \leq M \ \forall t, x$.*

1/ Show that the following scheme, known as "weak Euler Maruyama":

$$Y_{n+1} = Y_n + a(t_n, Y_n)h + b(t_n, Y_n)\xi_n \sqrt{h} \tag{4.47}$$

is weakly consistent. Here ξ_n are independent random variables and independent of $\mathcal{A}_{t_n}$ such that $P(\xi_n = \pm 1) = \frac{1}{2}$.

2/ Generalize for other variables ξ_n.

3/ Is the scheme strongly consistent?

Exercise 4.5. *(Strong and Weak Consistency)*

1. *Show that for any scheme, strong consistency implies weak consistency.*

2. *Show that the exact solution satisfies the conditions of strong consistency (therefore also weak consistency).*

Exercise 4.6. *(Heun's Scheme for SDEs) In this exercise, we consider that in* (4.39) *the coefficients a and b are independent of time, C^2 functions, and a, b, and their first and second derivatives (a', a'', b', b'') are also bounded. We study a formal generalization of Heun's scheme*

$$Y_{n+1} = Y_n + \frac{1}{2}\Big\{a(Y_n) + a\Big(Y_n + a(Y_n)h + b(Y_n)\Delta W_n\Big)\Big\}h$$
$$+\frac{1}{2}\Big\{b(Y_n) + b\Big(Y_n + a(Y_n)h + b(Y_n)\Delta W_n\Big)\Big\}\Delta W_n \tag{4.48}$$

Show that this scheme is not strongly consistent for all choices of a and b, and find for which types of coefficients the scheme is (sufficient conditions).

Exercise 4.7 (Consistency: Definitions)**.** *Show that for equation* (4.39)*, the definitions of consistency as SDE and as ODE coincide if $b = 0$ (a and b will be assumed as smooth as necessary).*

Exercise 4.8 (Order of Strong Convergence for Euler-Maruyama Limited to 1/2). *Provide an example of coefficients a and b such that the Euler-Maruyama scheme applied to equation* (4.39) *has a strong convergence order strictly less than* 1.0.

Exercise 4.9. *In equation* (4.39)*, we assume a, b Lipschitz, with growth at most quadratic in X. Show that a strongly consistent scheme starting from $X(0)$ converges strongly. Apply it to Euler-Maruyama and Milstein schemes and show that in these cases, the convergence order γ is greater than* 0.5.

Exercise 4.10 (Ito-Taylor Expansion for Milstein Scheme). *Explain how the additional term in Milstein compared to Euler-Maruyama is related to the Ito-Taylor expansion.*

Exercise 4.11 (Multi-step SDE Scheme). *With the notations from the course, consider the SDE scheme defined by $Y_{n+1} = Y_n + \frac{3}{2}a_n h - \frac{1}{2}a_{n-1}h + b_n\sqrt{h}\xi_n$ where ξ_n are i.i.d variables with mean m and finite variance σ^2, and each ξ_n independent of the filtration $\mathcal{A}_{t_n}$. We suppose that a, b are functions independent of time, and a, b, a', b', a'', b'' bounded.*

1. *Find m and σ^2 such that the scheme is weakly consistent.*
2. *Is the scheme strongly consistent? Justify.*

Exercise 4.12 (SDE Scheme). *With the notations from the course, consider the SDE scheme defined by $Y_{n+1} = Y_n + ha(Y_{n+1}) + b_n\Delta W_n$. We suppose that a, b are functions independent of time, and a, b, a', b', a'', b'' bounded.*

1. *Show that the scheme is well-posed by demonstrating that the equation $Z = Y_n + ha(Z) + b_n\Delta W_n$ admits a unique solution for sufficiently small h.*
2. *Calculate $\mathbb{E}\left[Y_{n+1} - Y_n \middle| \mathcal{A}_{t_n}\right]$.*
3. *Does the scheme satisfy the first condition of strong consistency? Justify.*
4. *Is the scheme strongly consistent? Justify.*

4.10 SDE Python lab

Exercise 4.13 (Brownian Motion). *1. Write a program that calculates a realization of a Brownian motion W_t over a horizon $T = 1$ with $N = 500$, $h = T/N$. Plot this realization.*

2. Modify the previous program, without adding a "for" loop, to calculate M realizations of the Brownian motion W_t (same parameters). Plot $M = 50$ such realizations on the same plot.

3. Using Riemann-Itô sums (4.3), *calculate $\int_0^T W_t dW_t$ and compare it with the exact formula for various values of h and by averaging over the realizations.*

Exercise 4.14 (EM and M Schemes). *In this exercise, we consider the SDE $dX_t = \theta(\mu - X_t)dt + \sigma dW_t$; X_t is called the Ornstein–Uhlenbeck process. We choose $\theta = 1.0$, $\mu = 10.0$, $\sigma = 3.0$, $X_0 = 0$, $T = 1$, $N = 100$.*

1. *Write a program that simulates X_t using a weak Euler-Maruyama scheme (see exercise 4.4), and plot the result for $M = 100$ scenarios.*

2. *Modify the previous program to implement the Milstein scheme (+ plot).*

3. *Show numerically, by varying h and adjusting M, that EM converges strongly to order 0.5, M to order 1, and EM weakly to order 1 (a specific test function should be chosen).*

Exercise 4.15 (European Option Pricing). *In this exercise, we consider the SDE $dS_t = \mu S_t dt + \sigma S_t dW_t$ (Black-Scholes model). Here, $T = 1.0$, $N = 255$, $M = 100$ (number of variations), $S_0 = 100.$, $\mu = 0.1$, $\sigma = 0.25$, $r = 0.05$ (risk-free interest rate), $K = 110$ (strike price).*

1. *Write a program that simulates W_t, S_t (with an exact formula); plot S_t. Calculate the option price and a confidence interval around it.*

2. *Repeat the above steps, but with an Euler-Maruyama scheme for calculating S_t.*

In all cases, plot the results.
Hint: for the confidence interval, you can use the following function:

```
import numpy as np

def bootstrap_mean_confidence_interval(data, num_iterations=10000,
    alpha=0.05):
    """
    A function to compute the average and a confidence interval
          around it.

    Use example
    data = np.array([0.2, 0.5, 0.7, 0.8, 1.1, 1.3, 1.5, 1.8, 2.0,
          2.2])
    print(bootstrap_mean_confidence_interval(data))


    Parameters
    ----------
    data : 1D array of data
    num_iterations : number of bootstrap iterations, default is
          10000.
    alpha : confidence level, the default is 0.05.

    Returns : the average and a confidence interval around it as a
            tuple
    -------
    """

    n = len(data)
    means = np.zeros(num_iterations)

    for i in range(num_iterations):
        means[i] = np.mean(np.random.choice(data, size=n, replace=
              True))

    means.sort()
    lower_bound = means[int(num_iterations * (alpha / 2))]
    upper_bound = means[int(num_iterations * (1 - alpha / 2))]
    mean_estimate = np.mean(means)

    return mean_estimate, (lower_bound, upper_bound)
```

Chapter 5

Some solutions

5.1 Solution for exercises section 2.10

Exercise 5.1. *Solution of exercise 2.12.*
After ONE time step: EE scheme gives $[1.1, 2.23, 2.94]$ *(direct calculations). For your information (calculations not easily feasible), the IE scheme gives* $[1.114, 2.256, 2.946]$ *and the CN scheme gives* $[1.107, 2.242, 2.943]$.
After TWO TIME STEPS the EE scheme gives $[1.213, 2.483, 2.886]$ *(direct calculations); this allows us to identify one of the schemes. For your information, the IE scheme gives* $[1.244, 2.543, 2.9]$ *and the CN scheme gives* $[1.228, 2.511, 2.893]$. *To identify, one must use IE, going backwards!*

5.2 ODE lab solutions, section 2.11

Exercise 5.2. *Solution of exercise 2.16.*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  8 12:12:15 2021

@author: turinici
"""

import numpy as np
import matplotlib.pyplot as plt
#%matplotlib inline
%matplotlib auto

# np.sqrt(2.)*np.sqrt(2.) == 2.
# False !!!!
```

```
#parameters
T=10.0 # final time
N=100 # number of time steps
h = T/N #
U0 = 0.0

def ftxt(t,x):
# define the function entering the ODE x'(t) = f(t,x(t))
    return(2.0*np.sqrt(np.abs(x)))

def Explicit_Euler(h,N,ode_function,initial_value):
    """
    Parameters
    ----------
    h : time step
    N : number of time steps (integer)
    input_function : function to integrate
    initial_value : initial value of the ODE

    Returns
    -------
    List of approximate solution obtained by Explicit Euler scheme
        of step h.

    """

    U= [0.0]*(N+1) # creates a list on N+1 elements, filled with
        value 0
    U[0]=initial_value

    for ii in range(N):
        U[ii+1]=U[ii]+h*ode_function(ii*h,U[ii])
    return(U)

# define the time grid
trange = np.linspace(0,T,num=N+1,endpoint=True)
# solve by EE the ODE : use EE at any time step and put in a numpy
    array : solution
solution = Explicit_Euler(h,N,ftxt,U0)
solution2 = Explicit_Euler(h,N,ftxt,np.sqrt(2.)*np.sqrt(2.)-2.)

# plot the results
plt.figure(1)
#plt.plot(solution)
plt.plot(trange,solution,'*r',trange,solution2,'ob')
plt.xlabel('time (t)')
plt.ylabel('x(t)')
plt.legend(['infinite precision solution x(t)','finite precision
    solution x(t)'])
```

Exercise 5.3. *Solution of exercise 2.17.*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  8 14:30:03 2021

@author: turinici
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

%matplotlib inline
#%matplotlib auto

#parameters
T=100 # final time
N=250 # number of time steps
```

```
h = T/N #
S0 = 20000
I0=10.
R0=0.

y0 = [S0,I0,R0]
#%%

beta=1.15
gamma=1./1.
print('reproduction number=',beta/gamma)

def sir_list(y,t,betaSIR,gammaSIR):
    """
    define the SIR function
    Parameters
    ----------
    t : time
    x : list of components of dimension d

    Returns
    -------
    a list, value of the function

    """
    S,I,R=y
    ntotal=S+I+R
    return [-betaSIR*S*I/ntotal,betaSIR*S*I/ntotal-gammaSIR*I,
        gammaSIR*I]

def sir_array(y,t,betaSIR,gammaSIR):
    """ like sir_list but return an array """
    return np.array(sir(y,t,betaSIR,gammaSIR))

#sir=sir_array
sir=sir_list


# define the time grid
trange = np.linspace(0,T,num=N+1,endpoint=True)
solution = odeint(sir, y0, trange, args=(beta,gamma))

Ssol=solution[:,0]
Isol=solution[:,1]
Rsol=solution[:,2]


plt.figure(2,figsize=(4,2))
plt.xlabel('temps')
plt.plot(trange,Ssol,'-b',trange,Isol,':r',trange,Rsol,'--g')
plt.legend(['S','I','R'])
plt.tight_layout()
plt.savefig('sir.pdf')

plt.figure(4,figsize=(7.5,2.5))
plt.subplot(1,3,1)
plt.xlabel('temps')
plt.plot(trange,Ssol,'-b',linewidth=4)
plt.legend(['S','I','R'])
plt.subplot(1,3,2)
plt.xlabel('temps')
plt.plot(trange,Isol,':r',linewidth=4)
plt.legend(['I'],loc='upper right')
plt.subplot(1,3,3)
plt.xlabel('temps')
plt.plot(trange,Rsol,'--g',linewidth=4)
plt.legend(['R'])
plt.savefig('sir3.pdf')
```

```
#%%

#in this second part we explore the order of error of several
    schemes

# TODO

# 1/ obtain the precise values of X(t) for t1=T0=52 and t2=60
# if times values are already in the trange just use:
#X52=[Ssol[52],Isol[52],Rsol[52]

#otherwise compute again with odeint
t1=52.0
T0=t1
t2=60.0
new_trange = [0.0,t1,t2]
new_solution = odeint(sir, y0, new_trange, args=(beta,gamma),rtol
    =1.e-10)

Xinit=new_solution[1]
Xend=new_solution[2]

def ftxt(t,y):
    """ define the function used by the ODE"""
    return sir_array(y,t,beta,gamma)

#this is the function appearing in the formula U_{n+1}= U_n + h \
    phi(Un,...)
#Explicit Euler
def phi_function_EE_scheme(Un,ftxt,h,tn):
    return ftxt(tn,Un)
#Heun
def phi_function_Heun_scheme(Un,ftxt,h,tn):
    return 0.5*( ftxt(tn,Un) +ftxt(tn+h,Un + h*ftxt(tn,Un)) )
#RK4
def phi_function_RK4_scheme(Un,ftxt,h,tn):
    K1 = ftxt(tn,Un)
    K2 = ftxt(tn+h/2.,Un+h/2.*K1)
    K3 = ftxt(tn+h/2.,Un+h/2.*K2)
    K4 = ftxt(tn+h,Un+h*K3)
    return (K1+2.*K2+2.*K3+K4)/6.


# starting from value at t1 solve numerically with EE, Heun,
    compare at time T2
# the numerical and the precise values for different values of h

error_list_RK4=[]
error_list_Heun=[]
error_list_EE=[]
hlist=[0.05, 0.01, 0.1, 0.5, 1., 2., 4.]

for h in hlist:
    current_N=np.int64((t2-t1)/h)
    #test if t2-t1 is really a multiple of h: assert())
    #assert(current_N*h == t2-t1)
    #use Xinit as initial values
    current_X_RK4=Xinit
    current_X_EE=Xinit
    current_X_Heun=Xinit
    for jj in range(current_N):
        current_X_RK4=current_X_RK4 + \
            h*phi_function_RK4_scheme(current_X_RK4,ftxt,h, t1+jj*
                h)
        current_X_Heun=current_X_Heun + \
            h*phi_function_Heun_scheme(current_X_Heun,ftxt,h, t1+
                jj*h)
        current_X_EE=current_X_EE + \
            h*phi_function_EE_scheme(current_X_EE,ftxt,h, t1+jj*h)

    #error_list_Heun.append(np.abs(current_X[0]-Xend[0]))
    error_list_Heun.append(np.sum(np.abs(current_X_Heun-Xend)))
    error_list_EE.append(np.sum(np.abs(current_X_EE-Xend)))
    error_list_RK4.append(np.sum(np.abs(current_X_RK4-Xend)))

plt.figure(3,figsize=(16,8))
plt.loglog(hlist,error_list_EE,'o-',hlist,error_list_Heun,'o-',
           hlist,error_list_RK4,'o-',)
plt.legend(['error_EE','error_Heun','error_RK4'])
```

Exercise 5.4. *Solution of exercise 2.18*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  8 12:12:15 2021

@author: turinici : TP no. 1 :ex 2.17: stability of Explicit Euler

for z'(t) = L * z(t)  with L = i  = sqrt(-1)
With notation z = x+i*y :
x,y=z
x' = Re(z') = Re( i * z) = Re(i*(x+iy))= - y
y' = Im(z') = Im(i*z) = Im(i*(x+iy)) = x
ODE equation is : [x,y]' = [-y,x]

"""

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#%matplotlib auto

#parameters
T=100.*2*np.pi # final time = 100*2*pi
N=5000 # number of time steps
h = T/N #
U0 = [1.,0.]

def ftxt(t,z):
    """
    define the function entering the ODE x'(t) = f(t,x(t))
    Parameters
    ----------
    t : time
    z : list of components of dimension d

    Returns
    -------
    a list, value of the function

    """
    x,y=z
    return [-y,x]

def Explicit_Euler(h,N,ode_function,initial_value):
    """
    Parameters
    ----------
    h : time step
    N : number of time steps (integer)
    input_function : function to integrate
    initial_value : initial value of the ODE

    Returns
    -------
    List of approximate solution obtained by Explicit Euler scheme
        of step h.
    """

    U= [initial_value]*(N+1) # creates a list on N+1 elements,
        filled with value 0
    for ii in range(N):
        U[ii+1]=[uiik+h*xk for xk,uiik in zip( ode_function(ii*h,U
            [ii]), U[ii])]
#           U[ii+1]=list(np.array(U[ii])+h*np.array(ode_function(ii*h
     ,U[ii])))

    return(U)

# define the time grid
trange = np.linspace(0,T,num=N+1,endpoint=True)
# solve by EE the ODE : use EE at any time step and put in a numpy
       array : solution
solution = Explicit_Euler(h,N,ftxt,U0)

solutionx = [z[0] for z in solution]
solutiony = [z[1] for z in solution]


# plot the results
plt.figure(1)
plt.subplot(2,1,1)
plt.plot(trange,solutionx,'-r')
plt.xlabel('time_(t)')
plt.ylabel('z')
plt.legend(['Real(z(t))'])
plt.subplot(2,1,2)
#plt.plot(solution)
plt.plot(trange,solutiony,'-b')
plt.xlabel('time_(t)')
plt.ylabel('z')
plt.legend(['Im(z(t))'])
```

5.3 Backward lab exercises corrections, section 3.8

Exercise 5.5. *Solution of exercise 3.9*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  8 14:30:03 2021

@author: turinici
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from scipy.interpolate import interp1d

%matplotlib inline
#%matplotlib auto

#parameters
T=150 # final time
N=150 # number of time steps
h = T/N #
S0 = 1.e+6
I0=10.
R0=0.
ntotal0=S0+I0+R0


# constant appearing in function c(beta) = c0/beta
c0=1.0
```

```
def cost(t,beta_function):
    """implements the cost function: c(beta(t)) = c0/beta(t)

    Inputs: beta is a function, t is a number

    """
    return c0/beta_function(t)


def dcost(t,beta_function):
    """implements the derivative of the cost function: c'(beta(t))
         = -c0/beta(t)**2

    Inputs: beta is a function, t is a number

    """
    return -c0/beta_function(t)**2

# test: dcost(1.,lambda t : 3.)

y0 = [S0,I0,R0]

beta=0.5
gamma=1./3.
print('reproduction_number=',beta/gamma)

def sir_list(y,t,betaSIR,gammaSIR):
    """
    define the SIR function
    Parameters
    ----------
    t : time
    x : list of components of dimension d

    Returns
    -------
    a list, value of the function

    """
    S,I,R=y
    ntotal=S+I+R
    return [-betaSIR*S*I/ntotal,betaSIR*S*I/ntotal-gammaSIR*I,
        gammaSIR*I]

def sir_array(y,t,betaSIR,gammaSIR):
    """ like sir_list but return an array """
    return np.array(sir(y,t,betaSIR,gammaSIR))

#sir=sir_array
sir=sir_list


# define the time grid
trange = np.linspace(0,T,num=N+1,endpoint=True)
solution = odeint(sir, y0, trange, args=(beta,gamma))

Ssol=solution[:,0]
Isol=solution[:,1]
Rsol=solution[:,2]

#construct functions S,I,R
Sfun = interp1d(trange,Ssol,fill_value="extrapolate")
Ifun = interp1d(trange,Isol,fill_value="extrapolate")
Rfun = interp1d(trange,Rsol,fill_value="extrapolate")


plt.figure(2)
plt.plot(trange,Ssol,trange,Isol,trange,Rsol)
plt.legend(['S','I','R'])
```

```
def sir_list_beta_function(y,t,beta_function,gammaSIR):
    """
    define the SIR function
    Parameters
    ----------
    t : time
    x : list of components of dimension d

    Returns
    -------
    a list , value of the function

    """
    S,I,R=y
    ntotal=S+I+R
    betat=beta_function(t)
    return [-betat*S*I/ntotal,betat*S*I/ntotal-gammaSIR*I,gammaSIR
        *I]



def adjoint_list_beta_function(lambdamu,t,beta_function,gammaSIR,
     Sfunction,Ifunction,Rfunction):
    """
    define the SIR function
    Parameters
    ----------
    t : time
    x : list of components of dimension d

    Returns
    -------
    a list , value of the function

    """
    lambda_t,mu_t=lambdamu
    betat=beta_function(t)
    It=Ifunction(t)
    St=Sfunction(t)
    Rt=Rfunction(t)
    ntotal=St+It+Rt
    return [betat*It*(lambda_t-mu_t)/ntotal,betat*St*(lambda_t-
        mu_t)/ntotal+gammaSIR*mu_t]


#see how can we use ODEINT to solve backwards:
# example exp(2*t) : x' = 2*x
# tmp=odeint(lambda x,t : 2.*x, 10.0, [0, 0.5, 1., 1.5, 2.])
# tmp = array([[ 10.],[ 27.18281891],[
    73.89056376],[200.85537821],[545.98153731]])
# we want to solve backward: give value at time 2 =
    545.9815003314424
# tmp2=odeint(lambda x,t : 2.*x, 545.98153731, [0, 0.5, 1., 1.5,
    2.][::-1])

betafunction =lambda t : beta

adjoint_solution = odeint(adjoint_list_beta_function, [-1.,0.],
    trange[::-1],
                                args=(betafunction,gamma,Sfun,Ifun,Rfun)
                                    )


lambdasol=adjoint_solution[:,0][::-1]#in order to correspond to
    trange not to trange[::-1]
musol=adjoint_solution[:,1][::-1]

#compute the gradient

gradientST = Ssol*Isol*(lambdasol-musol)
```

```
dcostarray=[dcost(t,betafunction) for t in trange]
gradient = dcostarray-gradientST/ntotal0
plt.figure(6)
plt.subplot(3,1,1)
plt.plot(trange,gradientST)
plt.subplot(3,1,2)
plt.plot(trange,gradient)
plt.subplot(3,1,3)
plt.plot(trange,dcostarray)


plt.figure(7)
plt.plot(trange,gradient)
plt.figure(8)
plt.plot(trange,-gradientST)
```

5.4 SDE exercises solutions, section 4.9

Reminder: in the calculation of conditional expectations, the following properties are useful ($\mathcal{A}$ is a tribe):

a/ $E(XZ|\mathcal{A}) = ZE(X|\mathcal{A})$ if Z is measurable with respect to $\mathcal{A}$

b/ $E(X|\mathcal{A}) = EX$ if X is independent of $\mathcal{A}$

c/ upper bound: $E(X|\mathcal{A}) \leq E(|X||\mathcal{A})$

Exercises 4.1 to 4.3 concern stochastic calculus. They are corrected in the book [7, Chapter 2], a PDF version of which is available online (PDF address: see "Bibliography" section on page 122).

Exo. 4.4 We verify the two properties in the definition of weak consistency.

$$E\left(\frac{Y_{n+1}-Y_n}{h}\middle|\mathcal{A}_{t_n}\right) - a_n = \mathbb{E}\left(\frac{a_n h + b_n\sqrt{h}\xi_n}{h}\middle|\mathcal{A}_{t_n}\right) - a_n$$

$$= a_n + \frac{b_n}{\sqrt{h}}\mathbb{E}\xi_n - a_n = 0,$$

where we used the fact that a_n and b_n are measurable with respect to $\mathcal{A}_{t_n}$, also the fact that ξ_n is independent of $\mathcal{A}_{t_n}$, and $\mathbb{E}\xi_n = 0$. Thus,

$$\mathbb{E}\left|E\left(\frac{Y_{n+1}-Y_n}{h}\middle|\mathcal{A}_{t_n}\right) - a_n\right|^2 = \mathbb{E}0 = 0, \qquad (5.1)$$

which gives the first bound in the definition of consistency, with $c(h) = 0$. For the second condition:

$$\mathbb{E}\left(\frac{(Y_{n+1}-Y_n)^2}{h}\middle|\mathcal{A}_{t_n}\right) = \mathbb{E}\left(\frac{(a_n h + b_n\sqrt{h}\xi_n)^2}{h}\middle|\mathcal{A}_{t_n}\right)$$
$$= ha_n^2 + 2a_n b_n\sqrt{h}\mathbb{E}\xi_n + b_n^2\mathbb{E}\xi_n^2 = ha_n^2 + b_n^2.$$

We again used $\mathbb{E}\xi_n = 0$ but also $\mathbb{E}\xi_n^2 = 1$ (and of course, the independence of ξ_n and the measurability of a and b with respect to $\mathcal{A}_{t_n}$). We conclude that

$$\mathbb{E}\left|\mathbb{E}\left(\frac{(Y_{n+1}-Y_n)^2}{h}\middle|\mathcal{A}_{t_n}\right) - b_n^2\right|^2 = \mathbb{E}(ha_n^2)^2 \leq h^2 M. \quad (5.2)$$

But $Mh^2 \to 0$ for $h \to 0$, which completes the demonstration of weak consistency (with $c(h) = Mh^2$ in both estimates).

2/ We notice that any sequence of independent random variables ξ_n with mean 0 and variance 1, independent of $\mathcal{A}_{t_n}$, yields the same result.

3/ It cannot be, because the second condition in the definition of strong consistency would not be satisfied (indeed, the variables ξ_n have no relation with ΔW_n, thus in particular cannot compensate them during the calculation, and we are left with a term that does not tend towards zero as $h \to 0$).

Exo. 4.6

The calculations of this exercise are not quite similar to those of the previous application for the following reason: $a\Big(Y_n + a_n h + b_n\Delta W_n\Big)$ is neither independent of $\mathcal{A}_{t_n}$ nor measurable with respect to $\mathcal{A}_{t_n}$; indeed, the function a mixes ΔW_n on one hand and Y_n,a_n, and b_n on the other hand, so, $a\Big(Y_n + a_n h + b_n\Delta W_n\Big)$ is not independent of $\mathcal{A}_{t_n}$ due to the presence of Y_n,a_n, and b_n and is not measurable with respect to $\mathcal{A}_{t_n}$ due to the presence of ΔW_n. We need then to replace the exact calculation of the conditional expectation, which we could do before, by an approximate calculation using Taylor formulas.

We first note that, with the notations $a_n = a(Y_n)$, $b_n = b(Y_n)$, $a'_n = a'(Y_n)$, $b'_n = b'(Y_n)$, a Taylor formula to order 2 gives us:

$a\Big(Y_n + a_n h + b_n \Delta W_n\Big) = a_n + a'_n \cdot \Big(a_n h + b_n \Delta W_n\Big) + \frac{a''(\alpha_y^n)}{2} \cdot \Big(a_n h + b_n \Delta W_n\Big)^2$ for some point α_y^n.

Similarly:

$b\Big(Y_n + a_n h + b_n \Delta W_n\Big) = b_n + b'_n \cdot \Big(a_n h + b_n \Delta W_n\Big) + \frac{b''(\beta_y^n)}{2} \cdot \Big(a_n h + b_n \Delta W_n\Big)^2$ for some point β_y^n.

Remark: e.g., a'_n is measurable with respect to $\mathcal{A}_{t_n}$ because it is a function, i.e., $a'(\cdot)$ applied to a variable Y_n which is measurable with respect to $\mathcal{A}_{t_n}$.

Now, we just need to perform the calculations in the same way as before. We **only** omit the initial immediate calculation which uses independence and measurability with respect to $\mathcal{A}_{t_n}$; the reader is invited to redo it if necessary.

$$E\left(\frac{Y_{n+1} - Y_n}{h}\middle| \mathcal{A}_{t_n}\right) - a_n$$
$$= a_n + \frac{a_n a'_n h + \mathbb{E}\Big(\frac{a''(\alpha_y^n)}{2}\Big(a_n h + b_n \Delta W_n\Big)^2 | \mathcal{A}_{t_n}\Big)}{2}$$
$$+ \frac{b_n b'_n}{2h} \mathbb{E}\Delta_n^2 + \frac{\mathbb{E}\Big(\frac{b''(\beta_y^n)}{2} \cdot \Big(a_n h + b_n \Delta W_n\Big)^2 \Delta W_n | \mathcal{A}_{t_n}\Big)}{2h} - a_n.$$

At this point, under the assumptions of the exercise, we can estimate

$$E\left(\frac{Y_{n+1} - Y_n}{h}\middle| \mathcal{A}_{t_n}\right) - a_n = \frac{b'_n b_n}{2} + O(\sqrt{h}) \qquad (5.3)$$

As an example, let's detail the treatment of the term

$$\frac{\mathbb{E}\Big(b''(\beta_y^n) \cdot \Big(a_n h + b_n \Delta W_n\Big)^2 \Delta | \mathcal{A}_{t_n}\Big)}{4h}.$$

First, we need to remember that β_y^n depends on ΔW_n as well, so $b''(\beta_y^n)$ might not be measurable with respect to $\mathcal{A}_{t_n}$ (nor necessarily independent of $\mathcal{A}_{t_n}$). So we will only have upper bounds:

$$\begin{aligned}
&\frac{\mathbb{E}\Big(b''(\beta_y^n)\cdot\Big(a_nh+b_n\Delta W_n\Big)^2\Delta W_n|\mathcal{A}_{t_n}\Big)}{2h}\\
&\leq M_2\frac{\mathbb{E}\Big(\Big(a_nh+b_n\Delta W_n\Big)^2|\Delta W_n|\Big|\mathcal{A}_{t_n}\Big)}{2h}\\
&\leq C(h\sqrt{h}+\sqrt{h}^2+\sqrt{h}^3)\leq C'h^{1/2}
\end{aligned}$$

with constants C, C' independent of h. Returning to (5.3), since $b_nb'_n$ doesn't necessarily tend to be small for $h\to 0$ (in fact, it doesn't even depend on h), the scheme is not generally consistent. A similar calculation shows that

$$\mathbb{E}\Big(\frac{1}{h}\Big|Y_{n+1}-Y_n-\mathbb{E}(Y_{n+1}-Y_n|\mathcal{A}_{t_n})-b_n\Delta W_n\Big|^2\Big)=O(h). \tag{5.4}$$

In conclusion, the scheme is strongly (thus weakly) consistent if and only if $bb'=0$, i.e., b is constant.

Exo. 4.7 We assume a to be as regular as desired, for example, C^∞ bounded with all its derivatives bounded. In the case where $b=0$, (4.39) is completely deterministic and becomes an ODE. The Itô process X_t solution of (4.39) no longer depends on the randomness ω. It is therefore relevant to compare the notion of consistency for ODEs with the two notions of consistency for SDEs. Thus, if we take a scheme $(Y_n)_{n\in\mathbb{N}}$ that approximates the solution to (4.39) and start from a deterministic initial condition $Y_0=X_0\in\mathbb{R}$, it is not necessary to make it dependent on ΔW_n and we can write it in the form $Y_{n+1}=Y_n+h\Phi(t_n,Y_n,a_n,h)$, $a_n=a(t_n,Y_n)$, $\Delta W_n=W_{t_{n+1}}-W_{t_n}$, with Φ as regular as desired (for example, C^∞ bounded with all its derivatives bounded). Since everything is deterministic here, the quantity $\Phi(t_n,Y_n,a_n,h)$ is independent of $\mathcal{A}_{t_n}$ and we obtain

$$\begin{aligned}
\mathbb{E}\left(Y_{n+1}-Y_n|\mathcal{A}_{t_n}\right)&=h\mathbb{E}\left(\Phi(t_n,Y_n,a_n,h)|\mathcal{A}_{t_n}\right)\\
&=h\mathbb{E}(\Phi(t_n,Y_n,a_n,h))\\
&=h\Phi(t_n,Y_n,a_n,h).
\end{aligned}$$

So, the first weak consistency condition becomes

$$\lim_{h\to 0^+} \mathbb{E}\left(\left|\frac{h\Phi(t_n,Y_n,a_n,h)}{h} - a_n\right|^2\right) = 0,$$

i.e., since everything is deterministic,

$$\lim_{h\to 0^+} |\Phi(t_n,Y_n,a_n,h) - a_n|^2 = 0,$$

i.e.

$$\lim_{h\to 0^+} a_n - \Phi(t_n,Y_n,a_n,h) = 0.$$

Now

$$a_n = a(t_n,Y_n) = \tilde{X}'(t_n),$$

where $\tilde{X}'$ is the unique solution to (4.39) satisfying $\tilde{X}(t_n) = Y_n$. Furthermore, as $\tilde{X}'$ is C^∞ and $\tilde{X}(t_n) = Y_n$, we have in particular that

$$\frac{\tilde{X}(t_{n+1}) - Y_n}{h} \to \tilde{X}'(t_n) = a_n \text{ as } h \to 0^+.$$

Thus,

$$\begin{aligned}
&\lim_{h\to 0^+} a_n - \Phi(t_n,Y_n,a_n,h) = 0\\
&\quad\Leftrightarrow \lim_{h\to 0^+}\left(a_n - \frac{\tilde{X}(t_{n+1}) - Y_n}{h}\right)\\
&\quad + \left(\frac{\tilde{X}(t_{n+1}) - Y_n}{h} - \Phi(t_n,Y_n,a_n,h)\right) = 0\\
&\quad\Leftrightarrow \lim_{h\to 0^+} \frac{\tilde{X}(t_{n+1}) - Y_n}{h} - \Phi(t_n,Y_n,a_n,h) = 0,
\end{aligned}$$

which indeed means that the local truncation error given in (2.9) tends to 0 as $h \to 0^+$, given Remark 2.8. Similarly,

$$\begin{aligned}
\mathbb{E}\left(\frac{(Y_{n+1} - Y_n)^2}{h}|\mathcal{A}_{t_n}\right) &= h^2\mathbb{E}\left((\Phi(t_n,Y_n,a_n,h))^2|\mathcal{A}_{t_n}\right)\\
&= h^2\mathbb{E}(\Phi(t_n,Y_n,a_n,h)^2)\\
&= h^2\Phi(t_n,Y_n,a_n,h)^2.
\end{aligned}$$

Furthermore, $b_n = 0$ here. Thus, the weak consistency condition (W_2) becomes

$$\lim_{h\to 0^+} \mathbb{E}\left(\left|\frac{h^2\Phi(t_n, Y_n, a_n, h)^2}{h}\right|^2\right) = 0,$$

i.e., since everything is deterministic,

$$\lim_{h\to 0^+} \left|h\Phi(t_n, Y_n, a_n, h)^2\right|^2 = 0,$$

i.e.

$$\lim_{h\to 0^+} h^2\Phi(t_n, Y_n, a_n, h)^4 = 0,$$

which is automatically satisfied since we assumed Φ to be bounded.

For strong consistency, the first condition is identical to the first weak consistency condition. As for the second condition of strong consistency, we notice that here, by a calculation already done,

$$Y_{n+1} - Y_n - \mathbb{E}\left(Y_{n+1} - Y_n|\mathcal{A}_{t_n}\right) = h\mathbb{E}(\Phi(t_n, Y_n, a_n, h)).$$

Thus, since $b_n = 0$, condition F_2 is rewritten here as

$$\lim_{h\to 0^+} \mathbb{E}\left(\frac{1}{h}\left|h\Phi(t_n, Y_n, a_n, h)\right|^2\right) = 0,$$

i.e., since everything is deterministic,

$$\lim_{h\to 0^+} \frac{1}{h}\left|h\Phi(t_n, Y_n, a_n, h)\right|^2 = 0,$$

i.e.

$$\lim_{h\to 0^+} h\Phi(t_n, Y_n, a_n, h)^2 = 0,$$

which is automatically satisfied since we assumed Φ to be bounded.

Thus, in this case, the weak and strong consistency conditions are identical to each other and identical to the consistency condition for ODEs, which is that the local truncation error given in (2.9) tends to 0.

Exo. 4.8 If we were in the case of an ODE (*i.e.* $b = 0$), we know that the Euler-Maruyama scheme is equivalent to the usual explicit Euler scheme and that the convergence order is necessarily 1. Thus, to find a suitable example, it is necessary to take $b \neq 0$. The simplest case would be to take $b = 1$ and $a = 0$, but in this case, the approximate solution would unfortunately be equal to the exact solution (check it). So, we need to look for something a bit more complicated. Let's take another simple case (there are surely infinitely many others that work, perhaps in a more elementary way). Let's set

$$f(t, x) = tx.$$

f is C^∞ in (t, x). Then, by the Itô formula,

$$df(t, W_t) = \frac{\partial f}{\partial t}(t, W_t)dt + \frac{\partial f}{\partial x}(t, W_t)dW_t$$

$$+\frac{1}{2}\frac{\partial^2 f}{\partial x^2}(t, W_t)dt = W_t dt + t dW_t,$$

which can be rewritten as

$$d(tW_t - \int_0^s W_s ds) = t dW_t.$$

Thus, if we define $X_t = tW_t - \int_0^s W_s ds$, W_t is a solution of

$$dX_t = t dW_t,\ X_0 = 0.$$

Let's see what happens when applying the Euler-Maruyama scheme to this SDE, starting from $Y_0 = 0$, over the time interval $[0, 1]$ (*i.e.* $T = 1$ here, and thus $N = \frac{1}{h}$):

$$Y_{n+1} = Y_n + t_n \left(W_{t_{n+1}} - W_{t_n}\right) = Y_n + nh \left(W_{t_{n+1}} - W_{t_n}\right).$$

It's very easy to solve this recurrence and deduce that

$$Y_n = Y_0 + h \sum_{i=1}^{n} i \left(W_{t_i} - W_{t_{i-1}}\right) = h \sum_{i=1}^{n} i \left(W_{t_i} - W_{t_{i-1}}\right).$$

Especially,

$$Y_N = h\sum_{i=1}^{N} i\left(W_{t_i} - W_{t_{i-1}}\right) = h\sum_{i=1}^{N-1} i\left(W_{t_i} - W_{t_{i-1}}\right) + W_1 - W_{1-h}.$$

Since we are interested in strong convergence, we look at the error at N given by the formula

$$e_N(h) = \mathbb{E}(|Y_N - X_1|)$$

$$= \mathbb{E}(|W_{1-h} - \int_0^1 W_s ds - h\sum_{i=1}^{N-1} i\left(W_{t_i} - W_{t_{i-1}}\right)|).$$

Now, it remains to estimate this quantity. We know that $W_{t_i} - W_{t_{i-1}}$ are all pairwise independent and follow a centered Gaussian law with variance h. Thus, the random variables $i\left(W_{t_i} - W_{t_{i-1}}\right)$ are also all pairwise independent and follow a centered Gaussian law with variance $i^2 h$. Their sum

$$\sum_{i=1}^{N-1} i\left(W_{t_i} - W_{t_{i-1}}\right)$$

is therefore again a centered Gaussian with variance

$$\sigma^2 = h\sum_{i=1}^{N-1} i^2 = h\frac{(N-N)(N+1)(2N)-11)}{6} = \frac{(1-h)(2-h)}{6h^2}.$$

Thus, the random variable

$$|\sum_{i=1}^{N-1} i\left(W_{t_i} - W_{t_{i-1}}\right)|$$

is a "folded normal law" whose mean is given by

$$\sigma\sqrt{\frac{2}{\pi}} = \sqrt{\frac{(1-h)(2-h)}{3\pi h^2}}.$$

Using the triangle inequality, we get

$$e_N(h) \geqslant \mathbb{E}(|W_{1-h} - \int_0^1 W_s ds|) - \sqrt{\frac{(1+h)(2+h)}{3}\pi h^2}.$$

A Taylor expansion gives

$$\sqrt{\frac{(1-h)(2-h)}{3\pi h^2}} = \frac{\sqrt{\frac{2}{3\pi}}}{h} - \frac{1}{2}\sqrt{\frac{3}{2\pi}} + O(h).$$

We still need to estimate $\mathbb{E}(|W_{1-h} - \int_0^1 W_s ds|)$.

Thus, we deduce that

$$e_N(h) \geqslant C'(T)\sqrt{h},$$

for a constant $C'(T) > 0$ depending on T whose value doesn't matter. Obviously, this prohibits having $e_N(h) \leqslant C''h$ for some $C'' > 0$ (simply because $C'(t)\sqrt{h} \leqslant C''h$ is necessarily false for h sufficiently small by dividing each side by $\sqrt{h}$), and thus the scheme is not of order 1 in this case. In fact, the scheme is at best of order 1/2. Since it was admitted in class that the Euler-Maruyama scheme was of strong convergence order at least 1/2, we cannot improve the $\sqrt{h}$ in the previous inequality.

Exo. 4.10

With the notations from the course, we are interested in the term

$$I = L^1 b(X_n) \int_{t_n}^{t_n+h} \int_{t_n}^{s} dW_\sigma dW_s$$

of the Ito-Taylor expansion, which dominates all the remainder terms placed in R_2 as shown in class. We recall that in the case where b does not depend on t, L^1 is given by

$$L^1 : b \mapsto b\, b'.$$

Moreover, the additional term in the Milstein scheme compared to the Euler-Maruyama scheme is given by

$$J = \frac{1}{2} b(X_n) b'(X_n) \left((\Delta W_n)^2 - h\right).$$

Thus, if we want to demonstrate that $I = J$, it suffices to demonstrate that

$$\int_{t_n}^{t_n+h} \int_{t_n}^{s} dW_\sigma dW_s = \frac{1}{2}\left((\Delta W_n)^2 - h\right).$$

To do this, we start by noticing that

$$\int_{t_n}^{s} dW_\sigma = W_s - W_{t_n},$$

so that

$$\begin{aligned}\int_{t_n}^{t_n+h}\int_{t_n}^{s} dW_\sigma dW_s &= \int_{t_n}^{t_n+h} W_s dW_s - W_{t_n}\int_{t_n}^{t_n+h} dW_s \\ &= \int_{t_n}^{t_n+h} W_s dW_s - W_{t_n}\left(W_{t_n+h} - W_{t_n}\right).\end{aligned}$$

Now, we use the Ito formula to compute the quantity $d(W_t^2)$. We set $g(t,x) = x^2$, so that $d(W_t^2) = d(g(t,B_t))$. g is C^∞ and

$$\frac{\partial g}{\partial t}(t,x) = 0,\ \frac{\partial g}{\partial x}(t,x) = 2x,\ \frac{\partial^2 g}{\partial x^2}(t,x) = 2.$$

We then obtain

$$d(W_t^2) = \frac{\partial g}{\partial t}(W_t)dt + \frac{\partial g}{\partial x}(W_t)dW_t + \frac{1}{2}\frac{\partial^2 g}{\partial x^2}(W_t)dt = 2W_t dW_t + dt.$$

By integrating between t_n and $t_n + h$, we get

$$W_{t_n+h}^2 - W_{t_n}^2 = 2\int_{t_n}^{t_n+h} W_s dW_s + (t_n + h - t_n).$$

Taking into account that $t_n + h - t_n = h$, we deduce that

$$\int_{t_n}^{t_n+h} W_s dW_s = \frac{1}{2}\left(-h + W_s^2 - W_{t_n}^2\right).$$

Thus, putting together the previous calculations, we deduce that

$$\begin{aligned}\int_{t_n}^{t_n+h}\int_{t_n}^{s} dW_\sigma dW_s &= \frac{1}{2}\left(-h + W_{t_n+h}^2 - W_{t_n}^2\right) - W_{t_n}\left(W_{t_n+h} - W_{t_n}\right) \\ &= \frac{1}{2}\left(h + W_{t_n+h}^2 - W_{t_n}^2 - 2W_{t_n}W_{t_n+h} + 2W_{t_n}^2\right) \\ &= \frac{1}{2}\left(-h + W_{t_n+h}^2 + W_{t_n}^2 - 2W_{t_n}W_{t_n+h}\right) \\ &= \frac{1}{2}\left(-h + (W_{t_n+h} - W_{t_n})^2\right),\end{aligned}$$

which was the desired result since by definition, $\Delta W_n = W_{t_n+h} - W_{t_n}$.

5.5 Solution of SDE exercises section 4.10

Exercise 5.6. *Solution of exercise 4.13*

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 12 14:51:49 2021

@author: turinici
"""
import numpy as np
import matplotlib.pyplot as plt


#TODO implement a brownian motion

#first idea: use the properties of the Wt:
T=1.0
N=255
M=10#number of scenarios
dt= T/N
W0=0#standard brownian motion
trange=np.linspace(0,T,N+1,endpoint=True)
# We know that Wt is a normal value of variance t
W1=np.sqrt(trange)*np.random.randn(N,1)
plt.figure(1)
plt.subplot(1,2,1)
plt.plot(trange[1:],W1)
#not working because the covariance is always zero ... and not min
    (s,t)
#good implementation: with increments

#another idea: use the cummulative increments property of B.M.
dW=np.sqrt(dt)*np.random.randn(N,M)
W=np.zeros((N+1,M))
W[0,:]=W0
W[1:,:]=W0+np.cumsum(dW,0)

plt.figure(2)
plt.plot(trange,W)

#compute \int_0^T W_t d W_t : using the Riemann-Ito sums
# in fact we compute sum_n W(t_n) * (increment between $t_n$ and
    $t_n+h$)
    # also compute integral minus W_T^2/2 and plot for all
        scenarios

int_WdW=np.zeros_like(W)
int_WdW[0,:]=0.0

for ii in range(N):
    int_WdW[ii+1,:] =int_WdW[ii,:]+ W[ii,:]*dW[ii,:]

plt.figure(3,figsize=(15,5))
plt.subplot(1,3,1)
plt.plot(trange,int_WdW)
plt.title('$t \mapsto \int_0^t W_u d W_u$')
plt.xlabel('t')
plt.subplot(1,3,2)
plt.plot(trange,W**2/2-int_WdW)
plt.title('$t \mapsto W_t^2/2-\int_0^t W_u d W_u$')
plt.xlabel('t')
plt.subplot(1,3,3)
plt.plot(trange,W**2/2-int_WdW- trange[:,None]**2/2)
plt.title('$t \mapsto W_t^2/2-t/2-\int_0^t W_u d W_u$')
plt.xlabel('t')
plt.tight_layout()
plt.show()
```

Exercise 5.7. *Solution of exercises 4.14 and 4.15*

```
"""
@author: Gabrel Turinici
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm


#implementation of the Black-Scholes formula
def blsprice(Price, Strike, Rate, TimeToMaturity, Volatility,
     DividendRate=0):
    """
    Computes price of option with analytic formula.
    input:
       S:Price - Current price of the underlying asset.

       Strike:Strike - Strike (i.e., exercise) price of the option
           .

       Rate: Rate - Annualized continuously compounded risk-free
           rate of return over
         the life of the option, expressed as a positive decimal
             number.

       TimeToMaturity:Time - Time to expiration of the option,
           expressed in years.

      Volatility: volatility
      DividendRate = continuous dividend rate

    output: price of a call and of a put (tuple)
    """

    if TimeToMaturity <= 1e-6: # the option already expired
        call = np.max(Price-Strike,0)
        put = np.max(Strike-Price,0)
        return call,put


    d1 = np.log(Price/Strike)+(Rate-DividendRate + Volatility
        **2/2.0)*TimeToMaturity;
    d1 = d1/(Volatility* np.sqrt(TimeToMaturity))
    d2 = d1-(Volatility*np.sqrt(TimeToMaturity))

    call = Price * np.exp(-DividendRate*TimeToMaturity) * \
        norm.cdf(d1)-Strike* np.exp(-Rate*TimeToMaturity) * norm.
            cdf(d2)
    put  = Strike* np.exp(-Rate*TimeToMaturity) * norm.cdf(-d2)\
        -Price* np.exp(-DividendRate*TimeToMaturity) * norm.cdf(-
            d1)
    return call,put




T=1.0
N=255
M=300#number of scenarios
dt= T/N
W0=0#standard brownian motion
trange=np.linspace(0,T,N+1,endpoint=True)

dW=np.sqrt(dt)*np.random.randn(N,M)
W=np.zeros((N+1,M))
W[0,:]=W0
W[1:,:]=W0+np.cumsum(dW,0)

plt.figure(2)
plt.plot(trange,W)
```

```
S0=100.
mu=0.1
sigma=0.25
taux_r=0.05

#compute S_t with dS_t = mu S_t dt + sigma S_t d W_t
St=np.zeros_like(W)
St[0,:]=S0

for ii in range(N):
    St[ii+1,:] =St[ii,:]+ mu*St[ii,:]*dt + sigma*St[ii,:]*dW[ii,:]

plt.figure(3)
plt.plot(trange,St)
plt.title('$S_t$')





#compute the Monte Carlo price of an option
# solve St in risk-neutral probability, denote rn_St

#compute rn_St with
# d rn_St = r rn_St dt + sigma rn_St d W_t
rn_St=np.zeros_like(W)
rn_St[0,:]=S0

for ii in range(N):
    rn_St[ii+1,:] =rn_St[ii,:]+ taux_r*rn_St[ii,:]*dt \
        + sigma*rn_St[ii,:]*dW[ii,:]

#compute the price of the call
K=110
prixcall,_ = blsprice(S0,K,taux_r,T,sigma)
prixMC=np.exp(-taux_r*T)*np.mean(np.maximum(rn_St[-1,:]-K,0))


print("prix Monte Carlo=",prixMC)

plt.subplot(2,2,4)
plt.hist(np.exp(-taux_r*T) *
           np.maximum(rn_St[-1,:]-K,0)-prixcall ,50)
plt.title('hist du prix Monte Carlo')

erreur_MC =prixMC-prixcall
#plt.savefig("euler_maruyama_monte_carlo.jpg")
```

Bibliography

[1] Antoine Danchin and Gabriel Turinici. Immunity after COVID-19: Protection or sensitization? *Mathematical Biosciences*, 331:108499, 2021.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, Cambridge, Massachusetts, November 2016.

[3] Laetitia Laguzet and Gabriel Turinici. Global optimal vaccination in the SIR model: properties of the value function and application to cost-effectiveness analysis. *Mathematical biosciences*, 263:180–197, 2015.

[4] Andrew Ng. Computation graph - Neural Networks Basics. Coursera, Neural Networks and deep learning course by Andrew Ng, https://www.andrewng.org/courses/, video: https://www.coursera.org/lecture/neural-networks-deep-learning/computation-graph-4WdOY.

[5] Tuen Wai Ng, Gabriel Turinici, and Antoine Danchin. A double epidemic model for the SARS propagation. *BMC Infectious Diseases*, 3(1):19, September 2003.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions, 2014.

[7] Imen Ben Tahar, José Trashorras, and Gabriel Turinici. *Élements de Calcul Stochastique pour l'Évaluation et la*

Couverture des Actifs Dérivés avec Exercices Corrigés Travaux Pratiques et Études de Cas. Ellipses Marketing, Paris, March 2016. version pdf disponible au: `https://turinici.com`.

[8] Gabriel Turinici. Personal web site. https://turinici.com, retrieved March 2024.