Estimation a posteriori

rapport scéance 1

Victor Baleux

14 septembre 2025

Table des matières

Chapitre 1

Équation différentielle ordinaire — Euler explicite

1.1 Code

Listing 1.1 – Script de génération des figures #1–#3 (Euler explicite, étude d'erreurs).

```
"""
Euler_ODE_Errors.py
-----------------------------------------------
EDO: u'(t) = -λ u(t), u(0)=u0, λ=1.

Figures générées :
1) Comparaison_visuelle.png
   -> 2×2 : haut Δt=1 s, bas Δt=0.001 s ; (gauche) solutions, (droite) erreur
2) Erreur_vs_delta_temps.png
   -> Erreurs L2 (u et u') en fonction de Δt (log-log)
3) Erreur_vs_derivé.png
   -> Scatter de l'erreur ponctuelle |e(t_n)| en fonction de la norme
      de la dérivée exacte |u'_{ex}(t_n)|, pour Δt=1 s et Δt=0.001 s.
      Sous-figure gauche: axes linéaires ; droite: log-log.
"""

import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from typing import Tuple, List


@dataclass
class Problem:
    lam: float = 1.0    # λ
    u0: float = 1.0     # condition initiale
    T: float = 60.0     # horizon temporel (1 minute)


def euler_explicite(pb: Problem, dt: float) -> Tuple[np.ndarray, np.ndarray, np.
    ndarray]:
    """
    Intègre u' = -λ u par Euler explicite avec pas nominal dt jusqu'à T.
    Le dernier pas est ajusté (si nécessaire) pour tomber exactement à T.
```

```python
    Retourne:
        t : instants (taille N+1)
        u : solution numérique aux instants t
        dts: taille de chaque intervalle (longueur N), avec dernier dt possiblement ajusté
    """
    T = pb.T
    lam = pb.lam
    u0 = pb.u0

    N_full = int(np.floor(T / dt))
    t_list = [0.0]
    u_list = [u0]

    assert dt < 2.0/lam, f"Instable pour Euler explicite: λΔt={lam*dt:.3g} (attendu < 2)."

    # Pas réguliers de taille dt
    for _ in range(N_full):
        un = u_list[-1]
        un1 = un + dt * (-lam * un)
        u_list.append(un1)
        t_list.append(t_list[-1] + dt)

    # Dernier pas ajusté pour atteindre exactement T (si besoin)
    t_current = t_list[-1]
    dt_last = T - t_current
    dts = [dt] * N_full  # liste des pas
    if dt_last > 1e-14:
        un = u_list[-1]
        un1 = un + dt_last * (-lam * un)
        u_list.append(un1)
        t_list.append(T)
        dts.append(dt_last)

    t = np.array(t_list, dtype=float)
    u = np.array(u_list, dtype=float)
    dts = np.array(dts, dtype=float)
    return t, u, dts


def u_exact(t: np.ndarray, pb: Problem) -> np.ndarray:
    return pb.u0 * np.exp(-pb.lam * t)


def l2_error_function(t: np.ndarray, u_num: np.ndarray, dts: np.ndarray, pb: Problem) -> float:
    ue = u_exact(t, pb)
    e_left = u_num[:-1] - ue[:-1]      # erreur évaluée au bord gauche de chaque intervalle
    return float(np.sqrt(np.sum((e_left**2) * dts)))


def l2_error_derivative(t: np.ndarray, u_num: np.ndarray, dts: np.ndarray, pb: Problem) -> float:
    # dérivée numérique par intervalle
    du = np.diff(u_num)
    uprime_num = du / dts  # taille N
```

```
    # points milieux de chaque intervalle
    t_mid = t[:-1] + 0.5 * dts
    # dérivée exacte aux milieux
    uprime_ex = -pb.lam * u_exact(t_mid, pb)
    eprime = uprime_num - uprime_ex
    return float(np.sqrt(np.sum((eprime**2) * dts)))


def convergence_slope(dts: np.ndarray, errs: np.ndarray) -> float:
    x = np.log(dts)
    y = np.log(errs)
    A = np.vstack([x, np.ones_like(x)]).T
    slope, _ = np.linalg.lstsq(A, y, rcond=None)[0]
    return float(slope)


def plot_part1_two_rows(pb: Problem,
                        dt_top: float = 1.0,
                        dt_bottom: float = 1e-3,
                        savepath: str = "Visual_comparison.png") -> None:
    """
    Figure 2x2 : top = Δt=1 s ; bottom = Δt=0.001 s.
    Colonnes: (gauche) solutions ; (droite) erreur ponctuelle.
    """
    # TOP (Δt = 1 s)
    t1, u1, dts1 = euler_explicite(pb, dt_top)
    ue1 = u_exact(t1, pb)
    err1 = np.abs(u1 - ue1)

    # BOTTOM (Δt = 0.001 s)
    t2, u2, dts2 = euler_explicite(pb, dt_bottom)
    ue2 = u_exact(t2, pb)
    err2 = np.abs(u2 - ue2)

    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    (ax00, ax01), (ax10, ax11) = axes

    # Top-left: solutions Δt=1 s
    ax00.plot(t1, ue1, label="Solution exacte $u_{ex}(t)$")
    ax00.plot(t1, u1, marker="o", linestyle="--", label=r"Euler $\Delta t=1$ s")
    ax00.set_xlabel("Temps t (s)")
    ax00.set_ylabel("Amplitude u(t)")
    ax00.set_title("Solutions -- Δt = 1 s")
    ax00.grid(True, alpha=0.3)
    ax00.legend()

    # Top-right: erreur Δt=1 s
    ax01.plot(t1, err1, marker="o", linestyle="-", label=r"$|e(t_n)|$")
    ax01.set_xlabel("Temps t (s)")
    ax01.set_ylabel("Erreur ponctuelle")
    ax01.set_title("Erreur -- Δt = 1 s")
    ax01.grid(True, alpha=0.3)
    ax01.legend()

    # Bottom-left: solutions Δt=0.001 s
    ax10.plot(t2, ue2, label="Solution exacte $u_{ex}(t)$")
    ax10.plot(t2, u2, linestyle="-", linewidth=1.0, label=r"Euler $\Delta t=0.001$
     s")
```

```python
    ax10.set_xlabel("Temps t (s)")
    ax10.set_ylabel("Amplitude u(t)")
    ax10.set_title("Solutions -- Δt = 0.001 s")
    ax10.grid(True, alpha=0.3)
    ax10.legend()

    # Bottom-right: erreur Δt=0.001 s
    ax11.plot(t2, err2, linestyle="-", linewidth=1.0, label=r"$|e(t_n)|$")
    ax11.set_xlabel("Temps t (s)")
    ax11.set_ylabel("Erreur ponctuelle")
    ax11.set_title("Erreur -- Δt = 0.001 s")
    ax11.grid(True, alpha=0.3)
    ax11.legend()

    fig.suptitle("u'(t) = -λ u, u(0)=1 ; λ=1 -- Comparaison visuelle Δt", y=0.98)
    fig.tight_layout(rect=[0, 0, 1, 0.96])
    fig.savefig(savepath, dpi=150)


def plot_part2(pb: Problem, n_steps: int = 20, dt_min: float = 1e-3, dt_max: float = 1.0,
               savepath: str = "L2_error_vs_deta_time.png") -> None:
    """
    Δt décroissants de 1 s à 0.001 s (échelle logarithmique), 20 valeurs.
    Trace ||e||_L2 et ||e'||_L2 en fonction de Δt (log-log).
    """
    dts_list = np.logspace(np.log10(dt_max), np.log10(dt_min), n_steps)
    errL2_u: List[float] = []
    errL2_du: List[float] = []

    for dt in dts_list:
        t, u_num, dts = euler_explicite(pb, dt)
        errL2_u.append(l2_error_function(t, u_num, dts, pb))
        errL2_du.append(l2_error_derivative(t, u_num, dts, pb))

    dts_arr = np.array(dts_list, dtype=float)
    errL2_u = np.array(errL2_u, dtype=float)
    errL2_du = np.array(errL2_du, dtype=float)

    slope_u = convergence_slope(dts_arr, errL2_u)
    slope_du = convergence_slope(dts_arr, errL2_du)

    fig, ax = plt.subplots(1, 1, figsize=(6.5, 4.5))
    ax.loglog(dts_arr, errL2_u, marker="o", linestyle="-", label=r"$\|u_h-u_{ex}\|_{L^2(0,T)}$")
    ax.loglog(dts_arr, errL2_du, marker="s", linestyle="--", label=r"$\|u'_h-u'_{ex}\|_{L^2(0,T)}$")
    ax.set_xlabel("Pas de temps Δt (s)")
    ax.set_ylabel("Erreur L2 sur [0, T]")
    ax.set_title(f"Erreurs L2 vs Δt -- pentes ≈ {slope_u:.2f} (u), {slope_du:.2f} (u')")
    ax.grid(True, which="both", alpha=0.3)
    ax.legend()
    fig.tight_layout()
    fig.savefig(savepath, dpi=150)

    # Impression console (utile pour le rapport)
```

```python
    print(f"Pente de convergence (log-log) pour ||u_h - u_ex||_L2 : {slope_u:.4f}")
    print(f"Pente de convergence (log-log) pour ||u'_h - u'_ex||_L2 : {slope_du:.4f}")


def plot_error_vs_exact_derivative(pb: Problem,
                                   dts_to_show: List[float] = [1.0, 1e-3],
                                   savepath: str = "Error_vs_exact_derivative.png") -> None:
    """
    Scatter: erreur ponctuelle |e(t_n)| en fonction de |u'_{ex}(t_n)|,
    pour plusieurs pas de temps (par défaut: Δt=1 s et Δt=0.001 s).
    Deux sous-graphes: (gauche) axes linéaires, (droite) log-log.
    """
    fig, (ax_lin, ax_log) = plt.subplots(1, 2, figsize=(12, 4.5))

    for dt in dts_to_show:
        t, u_num, dts = euler_explicite(pb, dt)
        ue = u_exact(t, pb)
        err = np.abs(u_num - ue)
        deriv_norm = np.abs(-pb.lam * ue)  # = pb.lam * |ue|

        ax_lin.scatter(deriv_norm, err, s=10, alpha=0.6, label=fr"$\Delta t={dt:g}$ s")
        ax_log.loglog(deriv_norm + 1e-16, err + 1e-16, marker="o", linestyle="", markersize=3, alpha=0.6, label=fr"$\Delta t={dt:g}$ s")

    ax_lin.set_xlabel(r"$|u'_{ex}(t_n)|$")
    ax_lin.set_ylabel(r"$|e(t_n)|$")
    ax_lin.set_title("Erreur vs norme de la dérivée exacte (linéaire)")
    ax_lin.grid(True, alpha=0.3)
    ax_lin.legend()

    ax_log.set_xlabel(r"$|u'_{ex}(t_n)|$")
    ax_log.set_ylabel(r"$|e(t_n)|$")
    ax_log.set_title("Erreur vs norme de la dérivée exacte (log-log)")
    ax_log.grid(True, which="both", alpha=0.3)
    ax_log.legend()

    fig.tight_layout()
    fig.savefig(savepath, dpi=150)


def main():
    pb = Problem(lam=1.0, u0=1.0, T=60.0)

    # Partie 1 : deux rangées: Δt = 1 s (haut) et Δt = 0.001 s (bas)
    plot_part1_two_rows(pb, dt_top=1.0, dt_bottom=1e-3,
                        savepath="Comparaison_visuelle.png")

    # Partie 2 : erreur L2 en fonction de Δt (20 valeurs entre 1 et 1e-3)
    plot_part2(pb, n_steps=20, dt_min=1e-3, dt_max=1.0,
               savepath="Erreur_vs_delta_temps.png")

    # Partie 3 : Erreur ponctuelle vs norme de la dérivée exacte
    plot_error_vs_exact_derivative(pb, dts_to_show=[1.0, 1e-3],
                                   savepath="Erreur_vs_derivé.png")
```

```python


if __name__ == "__main__":
    main()
```

1.2 Commentaire rapide du code

Le script 1.1 :

- résout $u'(t) = -\lambda u$ avec $u(0) = 1$ par Euler explicite pour différents pas Δt;
- calcule les erreurs L^2 sur u et u' par rapport à la solution exacte;
- produit trois figures: comparaison visuelle des trajectoires (grille 2×2), convergence en log-log, et erreur sur la dérivée exacte.

1.3 Images et commentaires

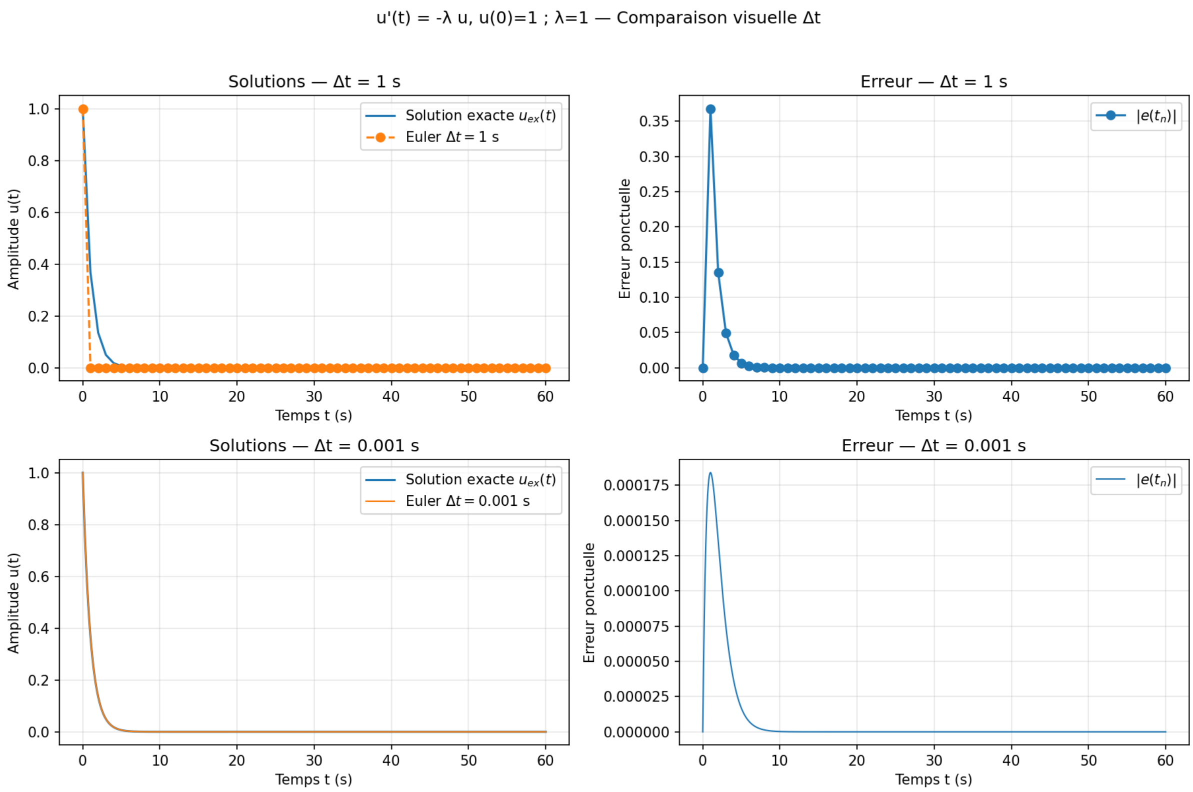

FIGURE 1.1 – Comparaison visuelle (grille 2×2) — haut: $\Delta t = 1$ s, bas: $\Delta t = 10^{-3}$ s ; à gauche les solutions (exacte vs. Euler), à droite l'erreur $|u - u_{\text{exact}}|$.

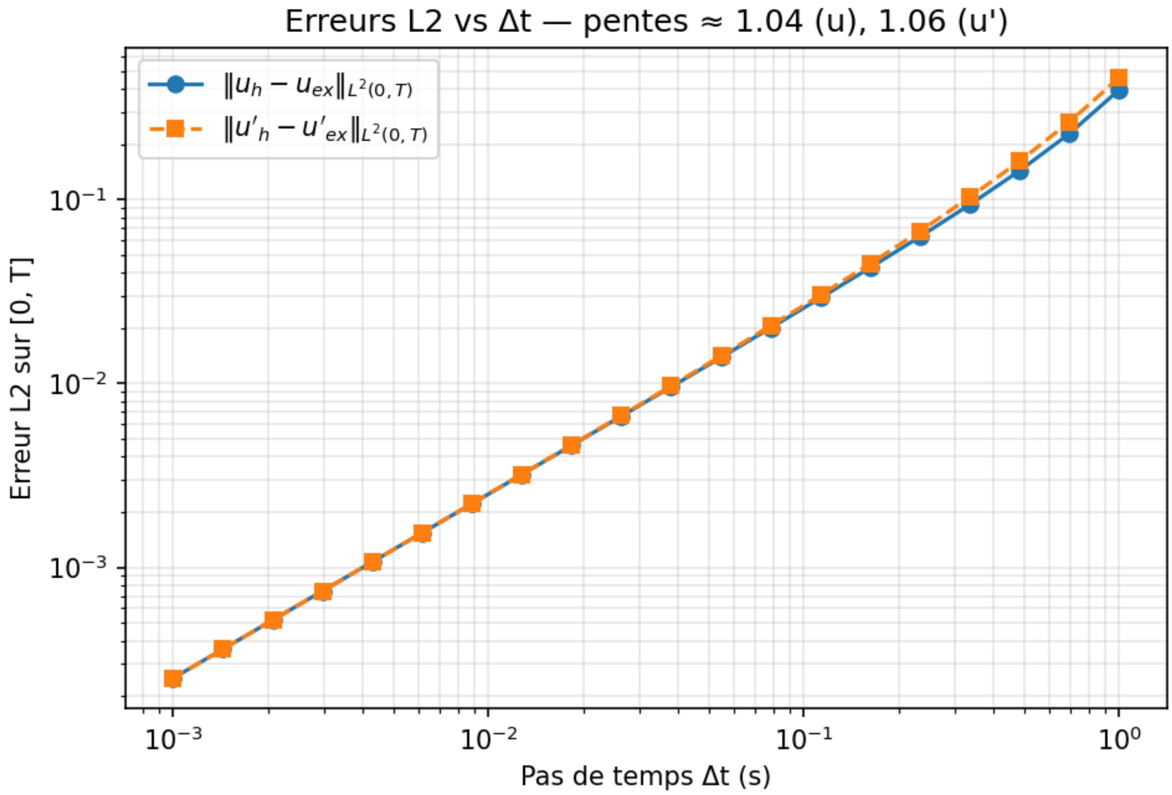

FIGURE 1.2 – Erreurs L^2 en fonction de Δt (échelle log-log) — les pentes numériques confirment l'ordre attendu d'Euler explicite.

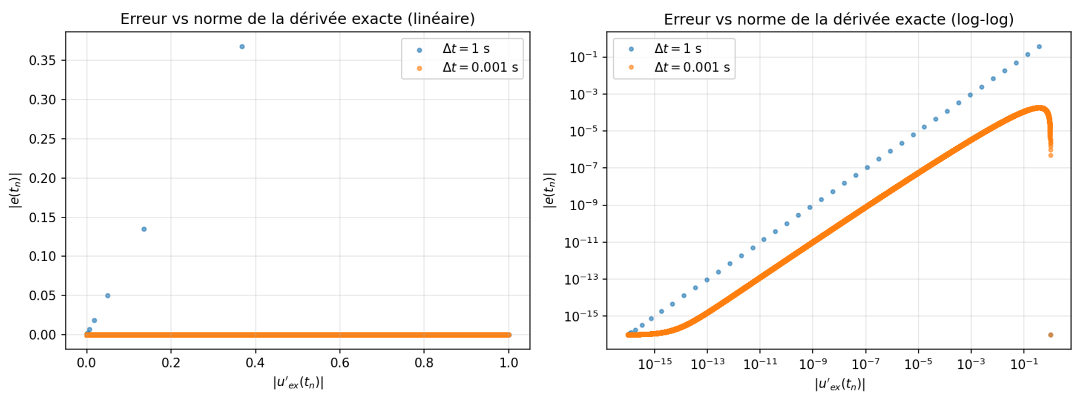

FIGURE 1.3 – Erreur par rapport à la dérivée exacte $u'(t)$ — cohérente avec l'analyse : plus Δt est petit, plus l'approximation s'améliore.

Notes méthodologiques (définitions et références)

- **Définition de la norme L^2 (temps)** : pour une suite de valeurs e_n sur des intervalles $[t_n, t_{n+1})$ de longueur Δt_n, on utilise $\|e\|_{L^2(0,T)} \approx \left(\sum_n e_n^2 \, \Delta t_n\right)^{1/2}$.
- **Définition de la norme L^2 (espace)** : pour une grille régulière (x_i, y_j) de pas $(\Delta x, \Delta y)$ et un champ scalaire $e_{i,j}$, on prend $\|e\|_{L^2(\Omega)} \approx \left(\sum_{i,j} e_{i,j}^2 \, \Delta x \, \Delta y\right)^{1/2}$.

— **Erreur sur la dérivée en EDO** : la dérivée numérique u'_h est approchée sur chaque intervalle par différence avant $(u_{n+1} - u_n)/\Delta t_n$, comparée à $u'_{\text{ex}}(t_n) = -\lambda u_{\text{ex}}(t_n)$.
— **Référence pour la 2D** : en l'absence de solution analytique, on construit u_{ref} sur une grille deux fois plus fine, puis on restreint u_{ref} sur la grille grossière (échantillonnage 2 : 1) avant de calculer les erreurs L^2 sur u et sur $\|\nabla u\|$.
— **Stabilité Euler explicite** : la condition $\lambda\,\Delta t < 2$ est rappelée et respectée dans les expériences (ici $\lambda = 1$).

Chapitre 2

Convection–diffusion 2D : schémas, erreurs et cartes

2.1 Code

Listing 2.1 – Script de génération du triptyque (solution, erreur sur u, erreur sur $\|\nabla u\|$).

```
"""
Convection-Diffusion(-Réaction) 2D (Dirichlet uniquement aux bords entrants).
- Génère **uniquement** un triptyque (collage) des 3 vues :
  solution, erreur sur u, erreur sur la norme du gradient.
- Sauvegarde **dans le même dossier que ce script** et affiche le triptyque.

Équation : u_t + V·∇u − ν Δu = −λ u + f(x,y),
           f(x,y) = Tc · exp(−k · ‖(x,y) − s_c‖²).
"""

from io import BytesIO
from pathlib import Path
import numpy as np
import scipy.sparse as sp
import scipy.sparse.linalg as spla
import matplotlib.pyplot as plt

def bords_entrants(v1, v2):
    """Côtés entrants (V·n < 0) pour V=(v1,v2)."""
    return (v1 > 0), (v1 < 0), (v2 > 0), (v2 < 0)  # gauche, droite, bas, haut

def masque_dirichlet(Nx, Ny, inflow):
    in_left, in_right, in_bot, in_top = inflow
    m = np.zeros((Ny, Nx), dtype=bool)
    if in_left:  m[:, 0]  = True
    if in_right: m[:, -1] = True
    if in_bot:   m[0, :]  = True
    if in_top:   m[-1, :] = True
    return m

def assemble_operateur(Nx, Ny, dx, dy, dt, nu, lam, mD):
    alpha = 1.0/dt + lam
    N = Nx*Ny
    rows, cols, vals = [], [], []
```

```python

    def idg(i,j): return i + Nx*j

    for j in range(Ny):
        for i in range(Nx):
            p = idg(i,j)
            if mD[j,i]:
                rows.append(p); cols.append(p); vals.append(1.0)
                continue

            diag = alpha
            # x
            if i == 0:
                diag += nu*(1.0/dx**2)
                rows.append(p); cols.append(idg(i+1,j)); vals.append(-nu*(1.0/dx**2))
            else:
                rows.append(p); cols.append(idg(i-1,j)); vals.append(-nu*(1.0/dx**2))
                diag += nu*(1.0/dx**2)
            if i == Nx-1:
                diag += nu*(1.0/dx**2)
                rows.append(p); cols.append(idg(i-1,j)); vals.append(-nu*(1.0/dx**2))
            else:
                rows.append(p); cols.append(idg(i+1,j)); vals.append(-nu*(1.0/dx**2))
                diag += nu*(1.0/dx**2)
            # y
            if j == 0:
                diag += nu*(1.0/dy**2)
                rows.append(p); cols.append(idg(i,j+1)); vals.append(-nu*(1.0/dy**2))
            else:
                rows.append(p); cols.append(idg(i,j-1)); vals.append(-nu*(1.0/dy**2))
                diag += nu*(1.0/dy**2)
            if j == Ny-1:
                diag += nu*(1.0/dy**2)
                rows.append(p); cols.append(idg(i,j-1)); vals.append(-nu*(1.0/dy**2))
            else:
                rows.append(p); cols.append(idg(i,j+1)); vals.append(-nu*(1.0/dy**2))
                diag += nu*(1.0/dy**2)

            rows.append(p); cols.append(p); vals.append(diag)

    return sp.csr_matrix((vals, (rows, cols)), shape=(N, N))

def advection_amont(u, v1, v2, dx, dy, uL, uR, uB, uT):
    Ny, Nx = u.shape
    dudx = np.zeros_like(u)
    dudy = np.zeros_like(u)
    # x
    if v1 >= 0:
        dudx[:, 1:] = (u[:, 1:] - u[:, :-1]) / dx
        dudx[:, 0]  = (u[:, 0]  - uL[:, 0]) / dx
```

```
    else:
        dudx[:, :-1] = (u[:, 1:] - u[:, :-1]) / dx
        dudx[:, -1]  = (uR[:, -1] - u[:, -1]) / dx
    # y
    if v2 >= 0:
        dudy[1:, :] = (u[1:, :] - u[:-1, :]) / dy
        dudy[0,  :] = (u[0,  :] - uB[0, :]) / dy
    else:
        dudy[:-1, :] = (u[1:, :] - u[:-1, :]) / dy
        dudy[-1,  :] = (uT[-1, :] - u[-1, :]) / dy
    return -(v1*dudx + v2*dudy)

def source_gauss(x, y, Tc, k, sc):
    X, Y = np.meshgrid(x, y, indexing='xy')
    return Tc * np.exp(-k * ((X - sc[0])**2 + (Y - sc[1])**2))

def norme_grad(u, dx, dy):
    Ny, Nx = u.shape
    dudx = np.zeros_like(u); dudy = np.zeros_like(u)
    dudx[:, 1:-1] = (u[:, 2:] - u[:, :-2]) / (2*dx)
    dudy[1:-1, :] = (u[2:, :] - u[:-2, :]) / (2*dy)
    dudx[:, 0]  = (u[:, 1] - u[:, 0]) / dx
    dudx[:, -1] = (u[:, -1] - u[:, -2]) / dx
    dudy[0, :]  = (u[1, :] - u[0, :]) / dy
    dudy[-1, :] = (u[-1, :] - u[-2, :]) / dy
    return np.sqrt(dudx**2 + dudy**2)

def erreur_L2(champ, ref, dx, dy):
    diff = champ - ref
    return np.sqrt(np.sum(diff**2) * dx * dy)

def resout_imex(ax,bx,ay,by,Nx,Ny,T,v1,v2,nu,lam,u_in,Tc,k,sc,cfl):
    x = np.linspace(ax, bx, Nx)
    y = np.linspace(ay, by, Ny)
    dx = (bx-ax)/(Nx-1); dy = (by-ay)/(Ny-1)

    inflow = bords_entrants(v1, v2)
    mD = masque_dirichlet(Nx, Ny, inflow)

    limites = []
    if abs(v1)>0: limites.append(dx/abs(v1))
    if abs(v2)>0: limites.append(dy/abs(v2))
    dt = cfl*min(limites) if limites else 0.02*min(dx,dy)
    nsteps = int(np.ceil(T/dt)); dt = T/nsteps

    M = assemble_operateur(Nx, Ny, dx, dy, dt, nu, lam, mD)
    lu = spla.splu(M.tocsc())

    f = source_gauss(x, y, Tc, k, sc)
    u = np.zeros((Ny, Nx))

    uL = np.full((Ny, 1), u_in)
    uR = np.full((Ny, 1), u_in)
    uB = np.full((1, Nx), u_in)
    uT = np.full((1, Nx), u_in)

    for _ in range(nsteps):
        adv = advection_amont(u, v1, v2, dx, dy,
```

```
                              np.repeat(uL, Nx, axis=1)[:, :1],
                              np.repeat(uR, Nx, axis=1)[:, -1:],
                              np.repeat(uB, Ny, axis=0)[:1, :],
                              np.repeat(uT, Ny, axis=0)[-1:, :])
        u_star = u + dt*(adv + f)
        rhs = (u_star/dt).ravel()
        rhs[mD.ravel()] = u_in
        u = lu.solve(rhs).reshape(Ny, Nx)

    return x, y, u, {"dt": dt, "nsteps": nsteps}

def figure_as_image(plotter, figsize=(6,5), dpi=160):
    """Crée une figure Matplotlib via la fonction plotter(ax) et renvoie son image
     PIL en mémoire."""
    import PIL.Image as Image
    fig, ax = plt.subplots(figsize=figsize)
    plotter(ax)
    buf = BytesIO()
    fig.tight_layout()
    fig.savefig(buf, format="png", dpi=dpi)
    plt.close(fig)
    buf.seek(0)
    return Image.open(buf).convert("RGB")

def collage_horizontal(images, outpath):
    """Colle des images PIL horizontalement et enregistre le résultat."""
    from PIL import Image
    h = max(im.size[1] for im in images)
    imgs = [im.resize((int(im.size[0]*h/im.size[1]), h)) for im in images]
    W = sum(im.size[0] for im in imgs)
    canvas = Image.new("RGB", (W, h), (255,255,255))
    xoff = 0
    for im in imgs:
        canvas.paste(im, (xoff, 0)); xoff += im.size[0]
    canvas.save(outpath)
    return canvas

def run():
    # ----- Paramètres -----
    ax, bx, ay, by = 0.0, 1.0, 0.0, 1.0
    Nx, Ny = 61, 61
    T = 1.0
    v1, v2 = 1.0, 0.3
    nu, lam = 0.01, 0.0
    u_in = 0.0
    Tc, k = 1.0, 80.0
    sc = (0.35, 0.55)
    cfl = 0.45

    # >>> Enregistrer dans le **même dossier que le script** <<<
    outdir = Path(__file__).parent
    outdir.mkdir(parents=True, exist_ok=True)
    outfile = outdir / "triple_panel.png"

    # Solve coarse
    x, y, uC, infoC = resout_imex(ax,bx,ay,by,Nx,Ny,T,v1,v2,nu,lam,u_in,Tc,k,sc,
    cfl)
    dxC, dyC = (bx-ax)/(Nx-1), (by-ay)/(Ny-1)
```

```

    # Reference fine
    NxF, NyF = 2*(Nx-1)+1, 2*(Ny-1)+1
    xF, yF, uF, infoF = resout_imex(ax,bx,ay,by,NxF,NyF,T,v1,v2,nu,lam,u_in,Tc,k,
    sc,cfl)
    uF_on_C = uF[::2, ::2]

    # Errors
    e_u_L2 = erreur_L2(uC, uF_on_C, dxC, dyC)
    gC = norme_grad(uC, dxC, dyC)
    dxF, dyF = (bx-ax)/(NxF-1), (by-ay)/(NyF-1)
    gF = norme_grad(uF, dxF, dyF)
    gF_on_C = gF[::2, ::2]
    e_g_L2 = erreur_L2(gC, gF_on_C, dxC, dyC)

    e_u_pw = np.abs(uC - uF_on_C)
    e_g_pw = np.abs(gC - gF_on_C)
    extent = [ax, bx, ay, by]

    # Figures en mémoire
    def plot_solution(axp):
        im = axp.imshow(uC, extent=extent, origin='lower', aspect='auto')
        axp.set_title(f"Solution u(x,y, T={T:.2f})\nV=({v1},{v2}), ν={nu}, λ={lam}
    ")
        axp.set_xlabel("x"); axp.set_ylabel("y")
        plt.colorbar(im, ax=axp, fraction=0.046, pad=0.04)

    def plot_err_u(axp):
        im = axp.imshow(e_u_pw, extent=extent, origin='lower', aspect='auto')
        axp.set_title(f"Erreur ponctuelle |u - u_ref| (‖e‖={e_u_L2:.2e})")
        axp.set_xlabel("x"); axp.set_ylabel("y")
        plt.colorbar(im, ax=axp, fraction=0.046, pad=0.04)

    def plot_err_grad(axp):
        im = axp.imshow(e_g_pw, extent=extent, origin='lower', aspect='auto')
        axp.set_title(f"Erreur ponctuelle |‖∇u‖ - ‖∇u_ref‖| (‖e‖={e_g_L2:.2e})")
        axp.set_xlabel("x"); axp.set_ylabel("y")
        plt.colorbar(im, ax=axp, fraction=0.046, pad=0.04)

    img1 = figure_as_image(plot_solution)
    img2 = figure_as_image(plot_err_u)
    img3 = figure_as_image(plot_err_grad)

    # Collage (unique fichier écrit)
    collage = collage_horizontal([img1, img2, img3], outfile)

    # Affichage
    plt.figure(figsize=(14,5))
    plt.imshow(collage)
    plt.axis("off")
    plt.title("Triptyque : Solution -- Erreur u -- Erreur ‖∇u‖")
    plt.show()

    print("Triptyque enregistré dans :", outfile)

if __name__ == "__main__":
    run()
```

2.2 Commentaire rapide du code

Le script 2.1 :

- pose l'EDP $u_t + \mathbf{V} \cdot \nabla u - \nu \Delta u = -\lambda u + f(x, y)$ avec source gaussienne;
- utilise un schéma explicite stabilisé (upwind pour la convection, diffusion centrée) sur une grille régulière;
- compare la solution numérique à une référence fine et en extrait deux cartes d'erreur (u et $\|\nabla u\|$).

2.3 Image et commentaire

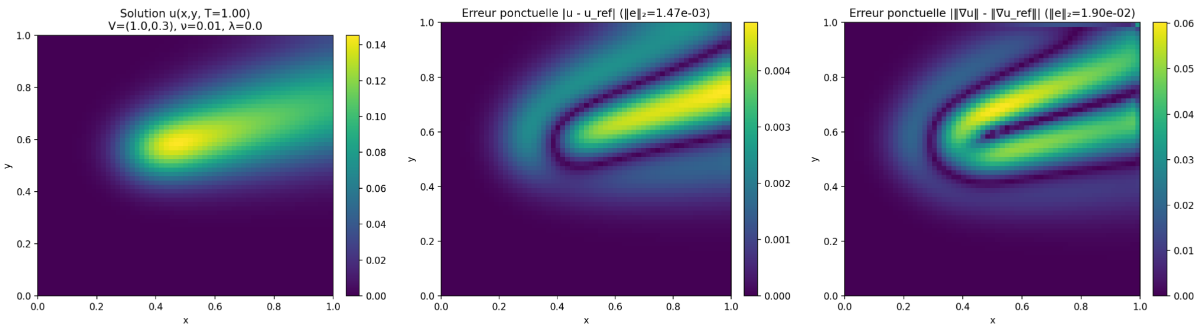

FIGURE 2.1 – Triptyque : (gauche) solution $u(x, y)$, (centre) erreur ponctuelle $|u - u_{\text{ref}}|$, (droite) erreur sur la norme du gradient $|\|\nabla u\| - \|\nabla u_{\text{ref}}\||$.