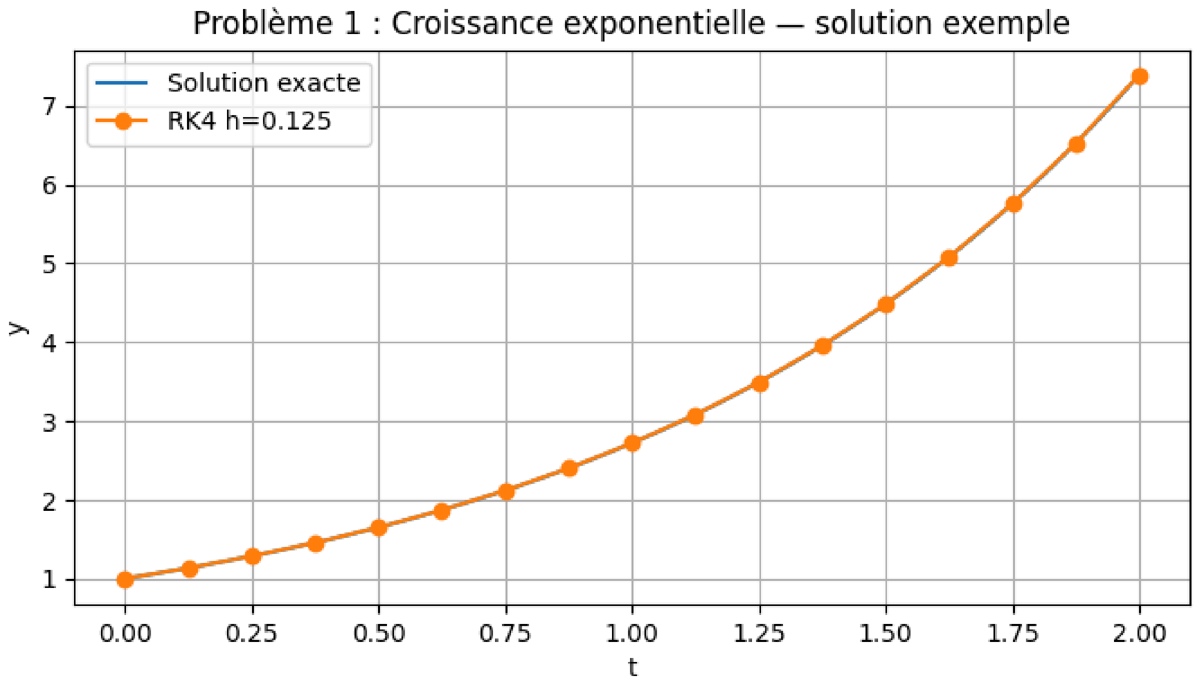
In [1]:

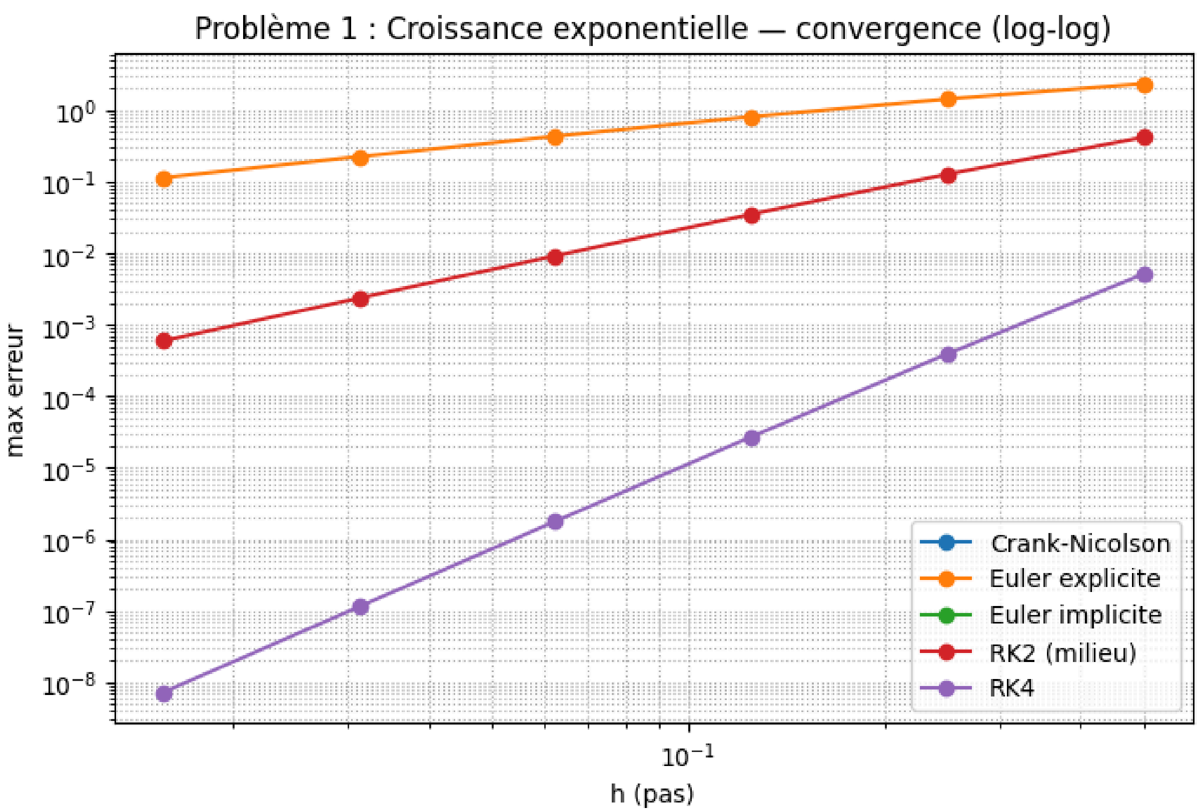
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# --- Utility: Newton solver for scalar or vector functions ---
def newton_solve(F, x0, tol=1e-10, maxiter=50):
    """
    Solve F(x)=0 using Newton's method with numerical Jacobian.
    x0 can be scalar or numpy array.
    """
    x = np.array(x0, dtype=float)
    for k in range(maxiter):
        Fx = np.atleast_1d(F(x))
        if np.linalg.norm(Fx, ord=2) < tol:
            return x
        # numeric Jacobian
        n = x.size
        J = np.zeros((n, n))
        eps = 1e-8
        for i in range(n):
            dx = np.zeros_like(x)
            dx[i] = eps if x.size > 1 else eps
            J[:, i] = (np.atleast_1d(F(x + dx)) - Fx) / dx[i]
        # solve J dx = -F
        try:
            dx = np.linalg.solve(J, -Fx)
        except np.linalg.LinAlgError:
            raise RuntimeError("Jacobian singular in Newton solver")
        x = x + dx
        if np.linalg.norm(dx, ord=2) < tol:
            return x
    raise RuntimeError("Newton did not converge")

# --- ODE solvers ---
def euler_explicit(f, t0, tf, y0, h):
    N = int(np.ceil((tf - t0) / h))
    t = np.linspace(t0, t0 + N * h, N + 1)
    y = np.zeros((N + 1,) + np.shape(y0))
    y[0] = y0
    for n in range(N):
        y[n + 1] = y[n] + h * f(t[n], y[n])
    return t, y

def euler_implicit(f, t0, tf, y0, h):
    N = int(np.ceil((tf - t0) / h))
    t = np.linspace(t0, t0 + N * h, N + 1)
    y = np.zeros((N + 1,) + np.shape(y0))
    y[0] = y0
    for n in range(N):
        tn1 = t[n + 1]
        yn = y[n]
        def G(yp):
            return np.atleast_1d(yp - yn - h * f(tn1, yp))
        yp = newton_solve(G, yn)
```

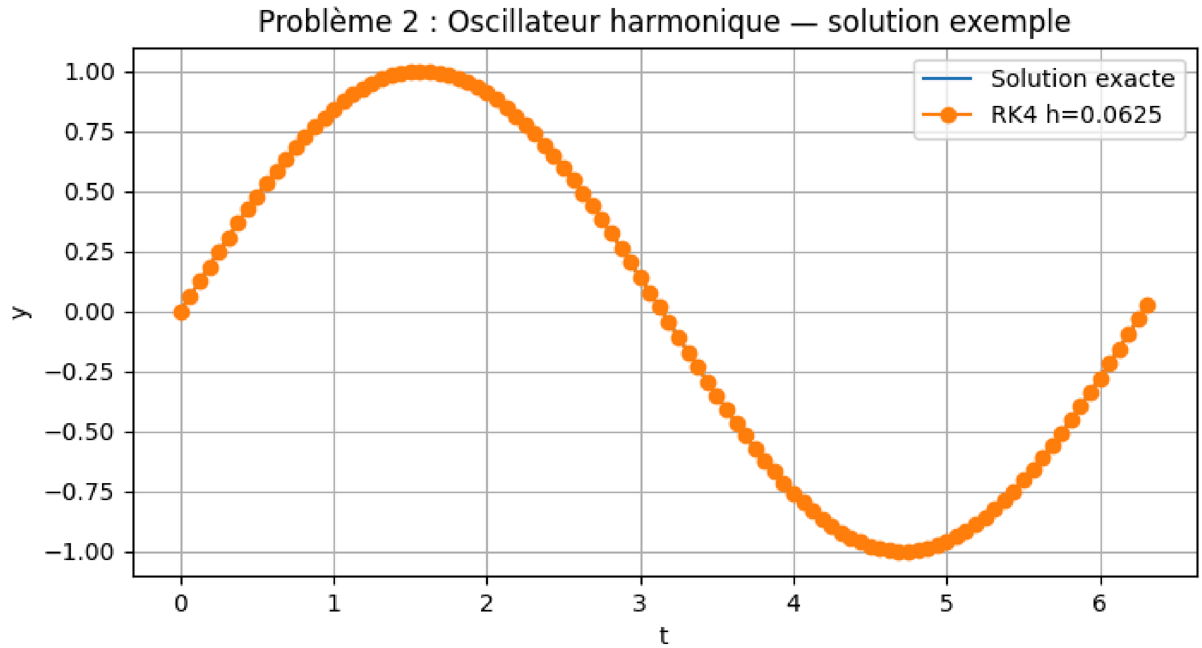
```
        y[n + 1] = yp
    return t, y

def crank_nicolson(f, t0, tf, y0, h):
    N = int(np.ceil((tf - t0) / h))
    t = np.linspace(t0, t0 + N * h, N + 1)
    y = np.zeros((N + 1,) + np.shape(y0))
    y[0] = y0
    for n in range(N):
        tn, tn1 = t[n], t[n + 1]
        yn = y[n]
        def G(yp):
            return np.atleast_1d(yp - yn - h / 2 * (f(tn, yn) + f(tn1, yp)))
        yp = newton_solve(G, yn)
        y[n + 1] = yp
    return t, y

def rk2_midpoint(f, t0, tf, y0, h):
    N = int(np.ceil((tf - t0) / h))
    t = np.linspace(t0, t0 + N * h, N + 1)
    y = np.zeros((N + 1,) + np.shape(y0))
    y[0] = y0
    for n in range(N):
        tn = t[n]
        k1 = f(tn, y[n])
        k2 = f(tn + h / 2, y[n] + h / 2 * k1)
        y[n + 1] = y[n] + h * k2
    return t, y

def rk4(f, t0, tf, y0, h):
    N = int(np.ceil((tf - t0) / h))
    t = np.linspace(t0, t0 + N * h, N + 1)
    y = np.zeros((N + 1,) + np.shape(y0))
    y[0] = y0
    for n in range(N):
        tn = t[n]
        k1 = f(tn, y[n])
        k2 = f(tn + h / 2, y[n] + h / 2 * k1)
        k3 = f(tn + h / 2, y[n] + h / 2 * k2)
        k4 = f(tn + h, y[n] + h * k3)
        y[n + 1] = y[n] + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
    return t, y

# --- Problèmes ---
# Problème 1 : croissance exponentielle
def f1(t, y): return y
y0_1 = 1.0
exact1 = lambda t: np.exp(t)
tspan1 = (0.0, 2.0)

# Problème 2 : oscillateur harmonique
def f2(t, u):
    u1, u2 = u
    return np.array([u2, -u1])
y0_2 = np.array([0.0, 1.0])
exact2 = lambda t: np.sin(t)
```

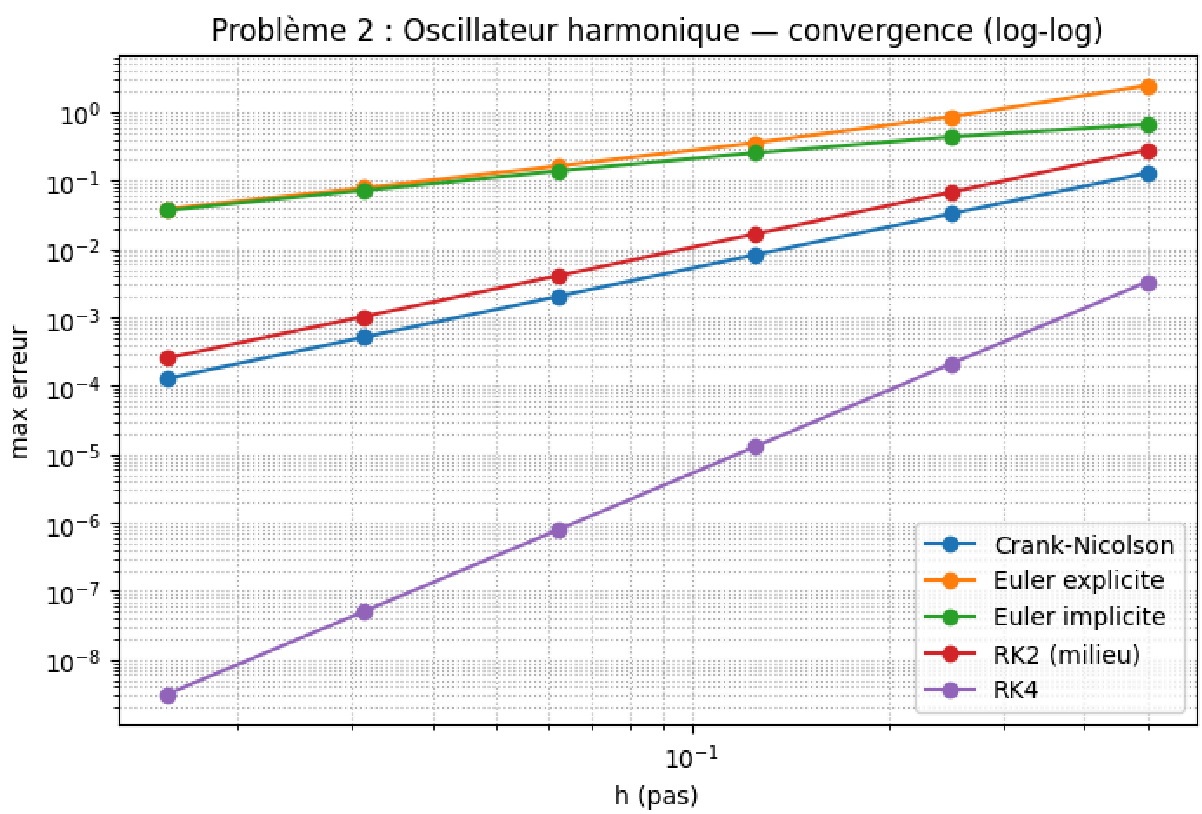
```
tspan2 = (0.0, 2 * np.pi)

# Problème 3 : équation raide
def f3(t, y):
    return -100 * y + 100 * t + 101
y0_3 = 1.0
exact3 = lambda t: np.exp(-100 * t) + t + 1
tspan3 = (0.0, 1.0)

# --- Liste des méthodes ---
methods = [
    ("Euler explicite", euler_explicit),
    ("Euler implicite", euler_implicit),
    ("Crank-Nicolson", crank_nicolson),
    ("RK2 (milieu)", rk2_midpoint),
    ("RK4", rk4),
]

# --- Étude de convergence ---
h_values = np.array([0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625])

def compute_errors(problem_f, y0, tspan, exact_func, is_system=False):
    results = []
    for h in h_values:
        for name, method in methods:
            try:
                t, y = method(problem_f, tspan[0], tspan[1], y0, h)
            except Exception:
                results.append((name, h, np.nan))
                continue
            if is_system:
                y_num = np.array([yi[0] for yi in y])  # on compare u1
            else:
                y_num = np.array(y).flatten()
            y_exact = exact_func(t)
            err = np.max(np.abs(y_num - y_exact))
            results.append((name, h, err))
    df = pd.DataFrame(results, columns=["method", "h", "max_error"])
    return df

# --- Exécution des 3 problèmes ---
df1 = compute_errors(f1, y0_1, tspan1, exact1, is_system=False)
df2 = compute_errors(f2, y0_2, tspan2, exact2, is_system=True)
df3 = compute_errors(f3, y0_3, tspan3, exact3, is_system=False)

# --- Fonction pour tracer ---
def plot_solution_and_errors(df, problem_f, y0, tspan, exact_func, is_system, probl
    # Exemple solution avec RK4
    t_ex, y_ex = rk4(problem_f, tspan[0], tspan[1], y0, example_h)
    if is_system:
        y_ex_plot = np.array([yi[0] for yi in y_ex])
    else:
        y_ex_plot = np.array(y_ex).flatten()
    t_exact = np.linspace(tspan[0], tspan[1], 1000)
    y_exact_dense = exact_func(t_exact)
```

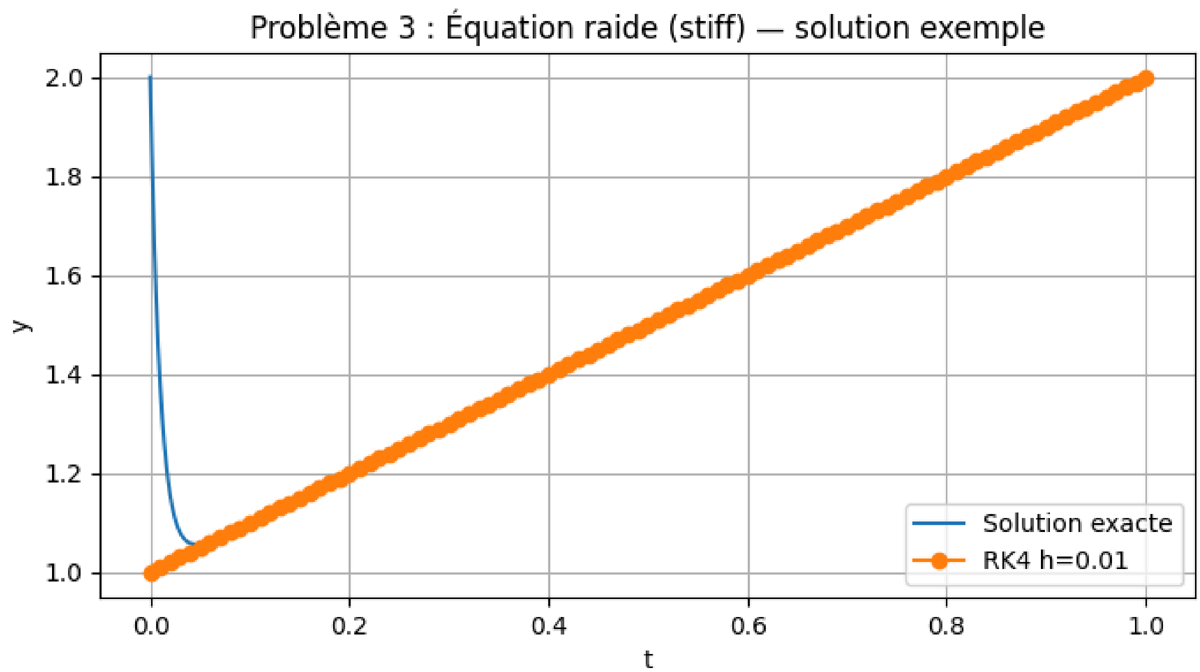
```
    plt.figure(figsize=(8, 4))
    plt.plot(t_exact, y_exact_dense, label="Solution exacte")
    plt.plot(t_ex, y_ex_plot, 'o-', label=f"RK4 h={example_h}")
    plt.xlabel("t")
    plt.ylabel("y")
    plt.title(problem_title + " — solution exemple")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Erreurs log-log
    plt.figure(figsize=(8, 5))
    for name, group in df.groupby("method"):
        hs = group["h"].values
        errs = group["max_error"].values
        plt.loglog(hs, errs, marker='o', label=name)
    plt.xlabel("h (pas)")
    plt.ylabel("max erreur")
    plt.title(problem_title + " — convergence (log-log)")
    plt.legend()
    plt.grid(True, which="both", ls=":")
    plt.show()

# --- Tracés pour les 3 problèmes ---
plot_solution_and_errors(df1, f1, y0_1, tspan1, exact1, False, "Problème 1 : Croiss
plot_solution_and_errors(df2, f2, y0_2, tspan2, exact2, True, "Problème 2 : Oscilla
plot_solution_and_errors(df3, f3, y0_3, tspan3, exact3, False, "Problème 3 : Équati

# --- Ordre empirique (pente log-log) ---
def empirical_order(df):
    rows = []
    for name, group in df.groupby("method"):
        group = group.sort_values("h")
        hs = group["h"].values
        errs = group["max_error"].values
        mask = np.isfinite(errs) & (errs > 0)
        if mask.sum() >= 2:
            p = np.polyfit(np.log(hs[mask]), np.log(errs[mask]), 1)[0]
            order = round(-p, 3)
        else:
            order = np.nan
        rows.append((name, order))
    return pd.DataFrame(rows, columns=["method", "empirical_order"])

print("Ordres empiriques — Problème 1")
print(empirical_order(df1))
print("\nOrdres empiriques — Problème 2")
print(empirical_order(df2))
print("\nOrdres empiriques — Problème 3")
print(empirical_order(df3))
```

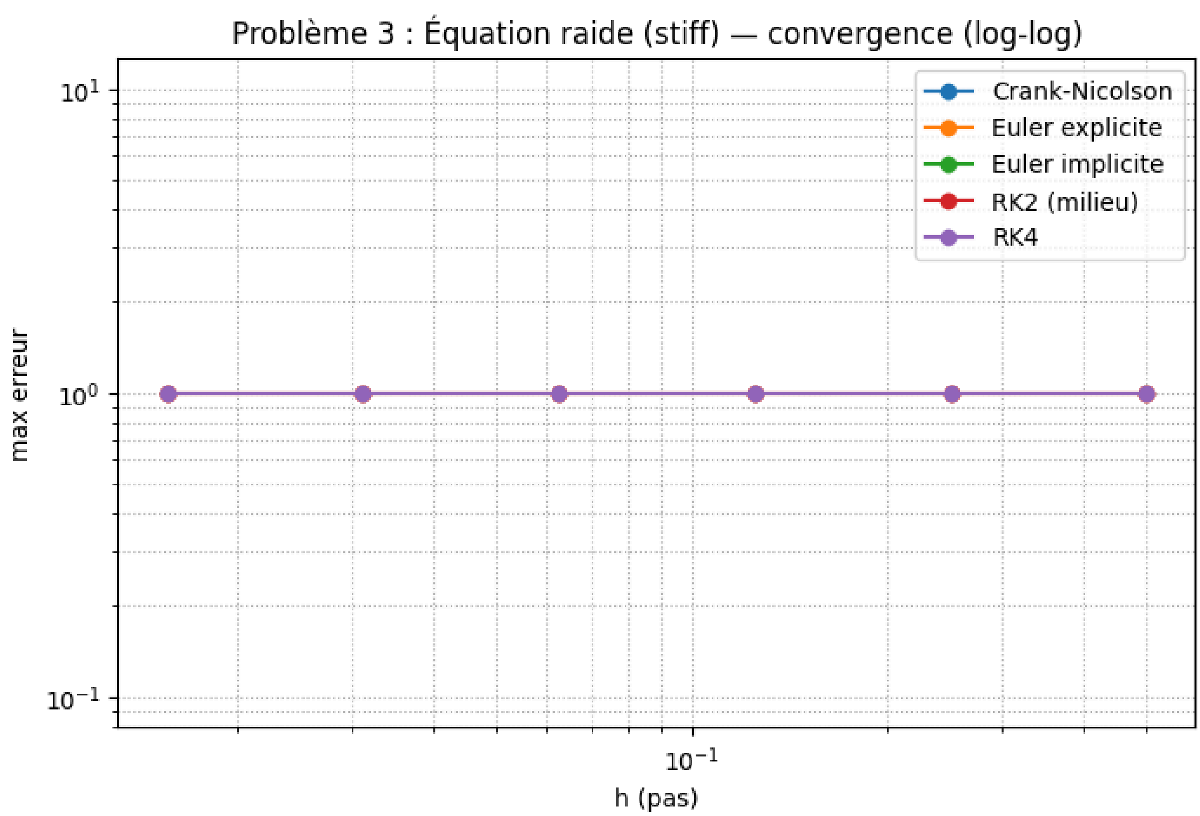

Problème 1 : Croissance exponentielle — solution exemple
Solution exacte
RK4 h=0.125
y
t
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1
2
3
4
5
6
7


Problème 1 : Croissance exponentielle — convergence (log-log)
max erreur
h (pas)
10^{-1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
Crank-Nicolson
Euler explicite
Euler implicite
RK2 (milieu)
RK4


Problème 2 : Oscillateur harmonique — solution exemple
Solution exacte
RK4 h=0.0625
y
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1
2
3
4
5
6


Problème 2 : Oscillateur harmonique — convergence (log-log)
Crank-Nicolson
Euler explicite
Euler implicite
RK2 (milieu)
RK4
max erreur
h (pas)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-1}


Problème 3 : Équation raide (stiff) — solution exemple
y
t
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Solution exacte
RK4 h=0.01


Problème 3 : Équation raide (stiff) — convergence (log-log)
max erreur
h (pas)
10^1
10^0
10^{-1}
10^{-1}
Crank-Nicolson
Euler explicite
Euler implicite
RK2 (milieu)
RK4

```
Ordres empiriques — Problème 1
             method  empirical_order
0   Crank-Nicolson              NaN
1  Euler explicite           -0.878
2  Euler implicite              NaN
3     RK2 (milieu)           -1.899
4              RK4           -3.893

Ordres empiriques — Problème 2
             method  empirical_order
0   Crank-Nicolson           -1.997
1  Euler explicite           -1.184
2  Euler implicite           -0.844
3     RK2 (milieu)           -2.019
4              RK4           -4.006

Ordres empiriques — Problème 3
             method  empirical_order
0   Crank-Nicolson              NaN
1  Euler explicite             -0.0
2  Euler implicite              NaN
3     RK2 (milieu)             -0.0
4              RK4             -0.0
```