Nom étudiant: Samuel Ogulluk
Numéro étudiant: 21501479

LU3PY126 FOAD

TP 1 Intégration des équations différentielles

Table des matières

Introduction

Au cours de ce TP, nous nous intéresserons à la résolution numérique d'équations différentielles.
Ainsi, nous verrons notamment les point suivants :

- ➔ résolutions d'équation différentielles d'ordre 1 par la méthode d'Euler
- ➔ résolution d'un système d'équations différentielles d'ordre 1 par la méthode d'Euler
- ➔ résolution d'une équation différentielle d'ordre 2 (cas du ressort non amorti)
- ➔ Application de la méthode de Runge-Kutta à l'ordre 2 et à l'ordre 4

Les codes présentés sont en C++ et Python pour la visualisation et sont joint à ce compte-rendu ainsi que disponibles dans le dépôt suivant : Dépôt Github

Question 1 Méthode d'Euler

Pour commencer, nous avons déterminé la solution à l'équation de désintégration $X \to Y$ donnant :

$$\frac{dx}{dt} = -kx$$

. Pour ce faire, nous utilisons Euler explicite :

$$y(t_{k+1}) = y(t_k) + f(y(t_k), t_k) \times \Delta t \tag{1}$$

Pour ce faire, nous itérons sur un tableau uniformément réparti temporellement, et on calcule chaque valeur de la fonction à partir des valeurs précédentes selon l'équation 1 ;

TP1/q1.cpp

```cpp
#include "tp1.hpp"

void solve_q1() {
    double k = 1.0;
    double Tmax = 5.0;
    double dt = 0.1;
    double x0 = 10.0;
    int steps = static_cast<int>(Tmax / dt);

    std::vector<double> x(steps);
    x[0] = x0;

    std::ofstream fichier("resultats/q1.csv");
    if (fichier.is_open()) {
        fichier << "t,x_euler,x_analytique" <<
        ↪ std::endl;

        for (int i = 0; i < steps - 1; ++i) {
            double t = i * dt;
            x[i+1] = x[i] - k * x[i] * dt;
            double x_analytique = x0 * std::exp(-k *
            ↪ t);
            fichier << t << "," << x[i] << "," <<
            ↪ x_analytique << std::endl;
        }
        fichier.close();
    }
}
```

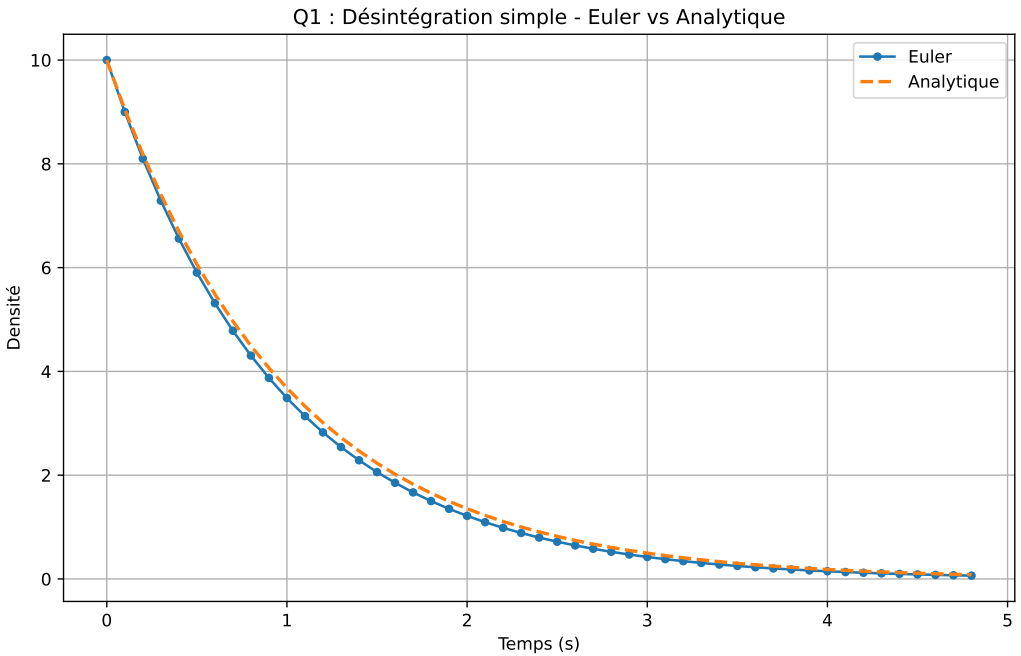

FIGURE 1 – Comparaison de la densité x(t) théorique et numérique

On constate ainsi que la méthode numérique approxime très bien la solution analytique, en présentant cependant toujours un retard, dû à la méthode employé (la méthode implicite aurait une avance).

Question 2 Désintégration X->Y->Z

On simule ensuite un système d'équation, appliquant la méthode d'euler à un vecteur. On obtient ainsi :

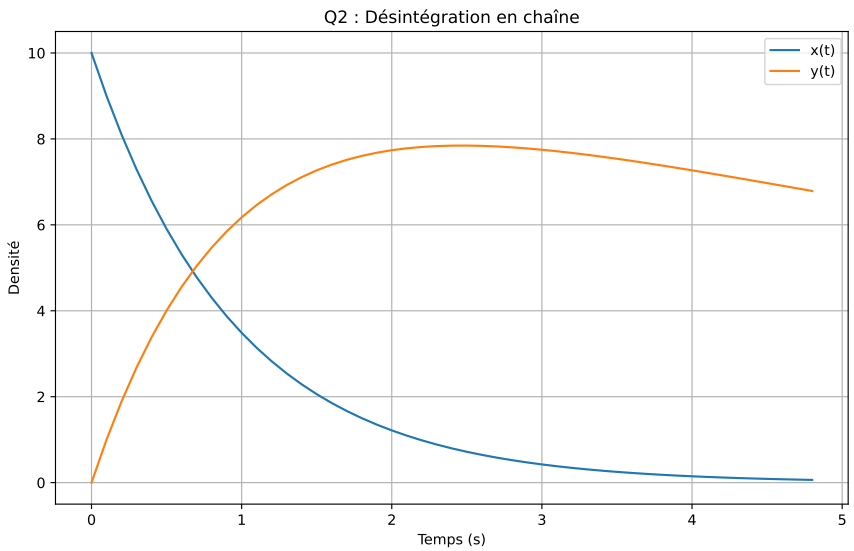

FIGURE 2 – Densité x(t) et y(t) en fonction du temps

On constate ainsi que x(t) n'est pas modifié, tandis que y(t) illustre la transformation Y→ Z, qui est de dérivée proportionnelle à x(t) et y(t).

TP1/q2.cpp

```cpp
// q2.cpp
#include "tp1.hpp"

void solve_q2() {
    double k = 1.0;
    double k2 = 0.1;
    double dt = 0.1;
    double Tmax = 5.0;
    int steps = static_cast<int>(Tmax / dt);

    std::vector<double> y(steps, 0.0);
    std::vector<double> x(steps, 0.0);
    x[0] = 10.0; // x0

    std::ofstream fichier("resultats/q2.csv");
    if (fichier.is_open()) {
        fichier << "t,x,y" << std::endl;
        //Euler mais sur deux variables (la deuxieme
        ↪  est la dérivée de la premiere)
        for (int i = 0; i < steps - 1; ++i) {
            double t = i * dt;
            double dxdt = -k * x[i];
            double dydt = k * x[i] - k2 * y[i];

            x[i+1] = x[i] + dxdt * dt;
            y[i+1] = y[i] + dydt * dt;
```

Question 3 Oscillateur harmonique

On s'intéresse maintenant à un oscillateur harmonique d'équation : $\ddot{x} + \omega_0^2 x = 0$

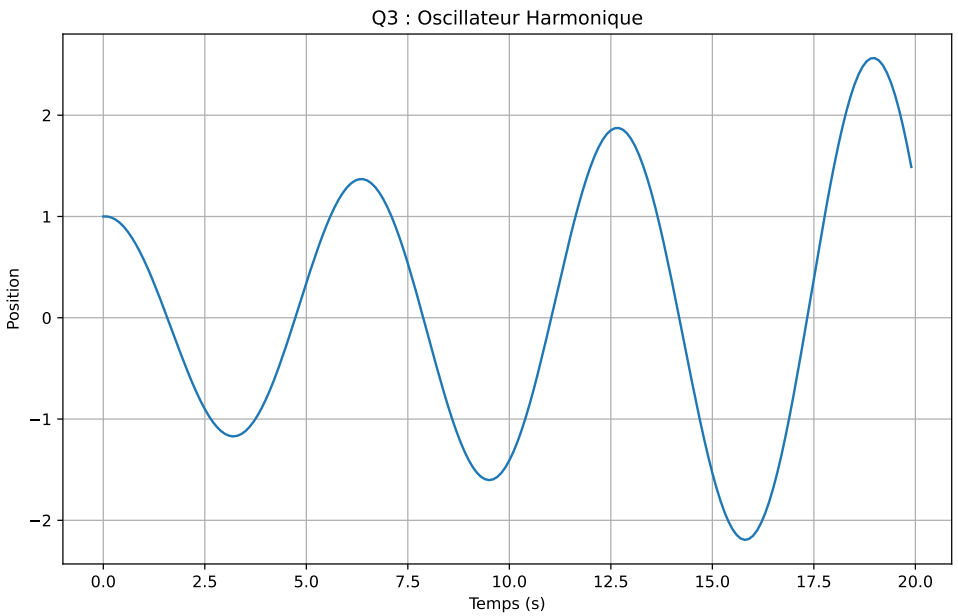

FIGURE 3 – x(t) d'un oscillateur harmonique

On constate ici un problème énergétique, l'amplitude du mouvement est croissante alors que le système est réversible.

TP1/q3.cpp

```cpp
#include "tp1.hpp"

void solve_q3() {
    double dt = 0.1;
    double Tmax = 20.0;
    int steps = static_cast<int>(Tmax / dt);
    // CI
    const int n = 2;
    double state[n];
    state[0] = 1.0;  // x0 = 1.0
    state[1] = 0.0;  // v0 = 0.0

    std::ofstream fichier("resultats/q3.csv");
    if (fichier.is_open()) {
        fichier << "t,x,v" << std::endl;

        for (int i = 0; i < steps; ++i) {
            double t = i * dt;
            fichier << t << "," << state[0] << "," <<
             ↪  state[1] << std::endl;
            euler(n, t, state, dt, deriv3);
        }
        fichier.close();
    }
}
```

Question 4 Fonction Deriv

Afin de résoudre le système étudié en question 2, on vient créer la fonction deriv suivante :

TP1/q4.cpp

```
6 void deriv_q4(int n, double t, double y[], double
   ↪  dy[]) {
7     dy[0] = y[1];
8     dy[1] = -w0_q4 * w0_q4 * y[0];
9 }
```

Cette fonction nous permet ensuite de venir reproduire les résultats vu précedémment, à l'aide d'un code plus général.

TP1/q4.cpp

```
11 void solve_q4() {
12     double dt = 0.1;
13     double Tmax = 50.0;
14     int steps = static_cast<int>(Tmax / dt);
15
16     // CI
17     const int n = 2;
18     double state[n];
19     state[0] = 1.0;  // x0 = 1.0
20     state[1] = 0.0;  // v0 = 0.0
21
22     std::ofstream fichier("resultats/q4.csv");
23     if (fichier.is_open()) {
24         fichier << "t,x,v" << std::endl;
25
26         for (int i = 0; i < steps; ++i) {
27             double t = i * dt;
28             fichier << t << "," << state[0] << "," <<
                ↪  state[1] << std::endl;
29
30             // Intégration par Euler
31             euler(n, t, state, dt, deriv_q4);
32         }
33
34         fichier.close();
35     }
36 }
```

Question 5 Fonction Euler

On souhaite maintenant créer une fonction générale permettant de résoudre numériquement des systèmes d'équations différentielles. Pour ce faire, on utilise l'équation 1 pour obtenir la fonction suivante :

TP1/q5.cpp

```cpp
void euler_q5(int n, double t, double y[], double dy[]) {

    deriv_q5(n, t, y, dy);

    for (int i = 0; i < n; ++i) {
        y[i] += dy[i] * dt;
    }
}

void deriv_q5(int n, double t, double y[], double dy[]) {
    dy[0] = -k_q5 * y[0];             // dx/dt
    dy[1] = k_q5 * y[0] - k2_q5 * y[1]; // dy/dt
}
```

Ainsi, on peut calculer à chaque itération, un point supplémentaire de la fonction recherchée et traiter la question 2 comme un cas particulier.

TP1/q5.cpp

```cpp
void solve_q5() {
    double dt = 0.1;
    double Tmax = 5.0;
    int steps = static_cast<int>(Tmax / dt);

    // CI
    const int n = 2;
    double state[n];
    state[0] = 10.0; // x0 = 10.0
    state[1] = 0.0;  // y0 = 0.0

    std::ofstream fichier("resultats/q5.csv");
    if (fichier.is_open()) {
        fichier << "t,x,y" << std::endl;

        for (int i = 0; i < steps; ++i) {
            double t = i * dt;

            fichier << t << "," << state[0] << "," << state[1] << std::endl;

            euler_q5(n, t, state, dt);
        }

        fichier.close();
    }
}
```

Question 6 Fonction Deriv3

On souhaite maintenant faire de même pour l'équation d'un oscillateur harmonique. On crée ainsi la fonction Deriv3, qui prend en entrée un vecteur constitué de x et de sa dérivée afin de résoudre le système 2 :

$$\ddot{x} = -\omega_0^2 x \iff \begin{cases} u &= \frac{dx}{dt} \\ \frac{du}{dt} &= -\omega_0^2 x \end{cases}$$

$$\iff \begin{cases} x(t_{k+1}) &= x(t_k) + u(t_k) \times \Delta t \\ u(t_{k+1}) &= u(t_k) - \omega_0^2 x(t_k) \times \Delta t \end{cases} \quad (2)$$

TP1/q6.cpp

```cpp
void deriv3(int n, double t, double y[], double dy[]) {
    dy[0] = y[1];
    dy[1] = -w0_global * w0_global * y[0];
}

void euler_q6(int n, double t, double y[], double dy[]) {
    deriv3(n, t, y, dy);
    for (int i = 0; i < n; ++i) {
        y[i] += dy[i] * dt;
    }
}
```

TP1/q6.cpp

```cpp
void solve_q6() {
    double dt = 0.1;
    double Tmax = 20.0;
    int steps = static_cast<int>(Tmax / dt);
    const int n = 2;
    double state[n];
    state[0] = 1.0;  // x0 = 1.0
    state[1] = 0.0;  // v0 = 0.0
    std::ofstream fichier("resultats/q6.csv");
    if (fichier.is_open()) {
        fichier << "t,x,v" << std::endl;
        for (int i = 0; i < steps; ++i) {
            double t = i * dt;
            fichier << t << "," << state[0] << "," << state[1] << std::endl;
            euler_q6(n, t, state, dt);
        }
        fichier.close();
    }
}
```

Question 7 Fonction Euler générale

On vient ensuite modifier le prototype de la fonction euler pour qu'elle prenne une fonction de type derivée en entrée. Ainsi, pour résoudre une équation différentielle ordinaire, il suffit de fournir les conditions initiales et l'équation différentielle sous forme de fonctionnelle.

TP1/q7.cpp

```cpp
void euler(int n, double t, double y[], double dt,
           void deriv(int, double, double[],
            ↪ double[])) {
    double dy[n];

    deriv(n, t, y, dy);

    // Euler explicite ici
    for (int i = 0; i < n; ++i) {
        y[i] = y[i] + dy[i] * dt;
    }
}
```

On peut ainsi résoudre les précédents exemples.

TP1/q7.cpp

```cpp
void solve_q7() { // exemple de résolution de q6
    double dt = 0.1;
    double Tmax = 20.0;
    int steps = static_cast<int>(Tmax / dt);

    const int n = 2;
    double state[n];
    state[0] = 1.0;  // x0 = 1.0
    state[1] = 0.0;  // v0 = 0.0

    std::ofstream fichier("resultats/q7.csv");
    if (fichier.is_open()) {
        fichier << "t,x,v" << std::endl;

        for (int i = 0; i < steps; ++i) {
            double t = i * dt;

            fichier << t << "," << state[0] << "," <<
             ↪ state[1] << std::endl;

            euler(n, t, state, dt,deriv3);
        }

        fichier.close();
    }
}
```

Question 8 Simulation de particules chargées

On vient enfin simuler une particule chargée dans un champ électromagnétique uniforme et constant.

On obtient ainsi l'évolution dans le temps des coordonnées de la particule, ainsi que la trajectoire qu'elle prend :

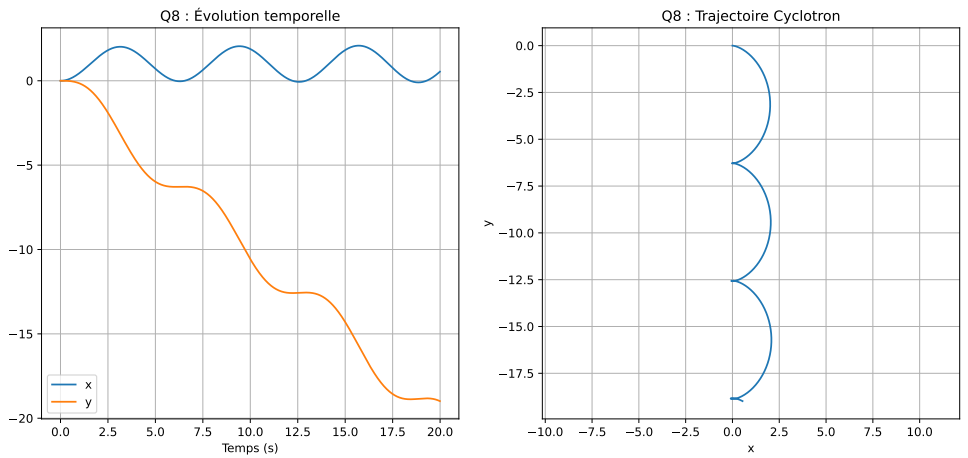

FIGURE 4 – Evolution des coorodnnées de la particule chargée

TP1/q8.cpp

```cpp
void deriv_particle(int n, double t, double y[],
    double dy[]) {
    dy[0] = y[2];
    dy[1] = y[3];
    dy[2] = 1.0 + y[3];
    dy[3] = -y[2];
}

void solve_q8() {
    double dt = 0.01;
    double Tmax = 20.0;
    int steps = static_cast<int>(Tmax / dt);
    const int n = 4;
    double state[n] = {0.0, 0.0, 0.0, 0.0};
    std::ofstream fichier("resultats/q8.csv");
    if (fichier.is_open()) {
        fichier << "t,x,y" << std::endl;
        for (int i = 0; i < steps; ++i) {
            fichier << i * dt << "," << state[0] <<
                "," << state[1] << std::endl;
            euler(n, i * dt, state, dt,
                deriv_particle);
        }
        fichier.close();
    }
}
```

Question 9 Oscillateur harmonique et erreurs

On s'intéresse maintenant à l'erreur commise par la méthode d'Euler explicite sur le cas de l'oscillateur harmonique, dont on connaît la solution analytique :

TP1/q9.cpp

```
7  void deriv_q9(int n, double t, double y[],
   ↪  double dy[]) {
8      dy[0] = y[1];
9      dy[1] = -w0_q9 * w0_q9 * y[0];
10 }
11
12 double solution_analytique(double t, double
   ↪  x0, double v0, double w0) {
13     return x0 * std::cos(w0 * t) + (v0 /
       ↪  w0) * std::sin(w0 * t);
14 }
```

On constate encore la divergence de l'enveloppe montrant les difficulté de cette méthode à traiter l'énergie. On constate de plus que diminuer Δt permet de diminuer l'erreur, mais ne change rien à la divergence de l'amplitude.

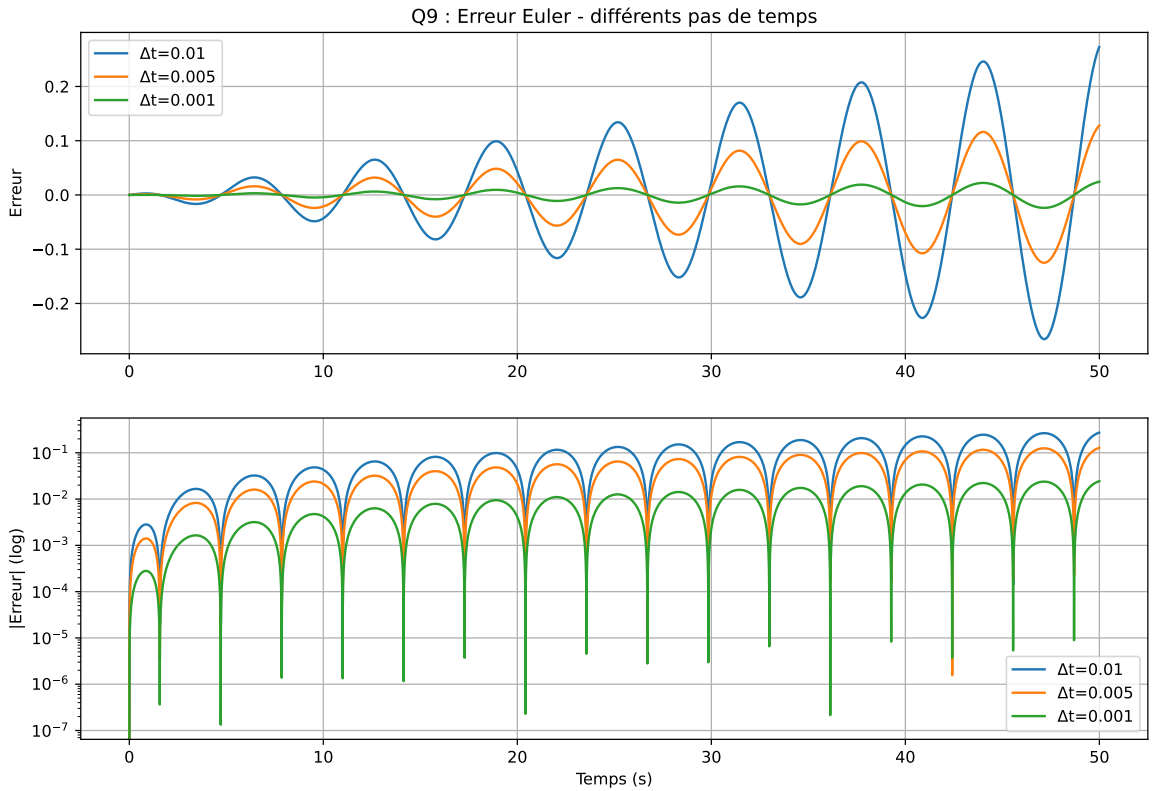

FIGURE 1 – Erreur en fonction du temps et du pas de temps

TP1/q9.cpp

```cpp
void solve_q9_with_dt(double dt, const std::string& filename) {
    double Tmax = 50.0;
    int steps = static_cast<int>(Tmax / dt);
    const int n = 2;
    double state[n];
    state[0] = 1.0;  // x0 = 1.0
    state[1] = 0.0;  // v0 = 0.0
    double x0 = state[0];
    double v0 = state[1];
    std::ofstream fichier(filename);
    if (fichier.is_open()) {
        fichier << "t,x_numerique,x_analytique,erreur" << std::endl;
        for (int i = 0; i < steps; ++i) {
            double t = i * dt;
            double x_th = solution_analytique(t, x0, v0, w0_q9);
            double erreur = state[0] - x_th;
            fichier << t << "," << state[0] << "," << x_th << "," << erreur << std::endl;
            euler(n, t, state, dt, deriv_q9);
        }
        fichier.close();
    }
}

void solve_q9() {
    // Test avec différents pas de temps
    solve_q9_with_dt(0.01, "resultats/q9_dt001.csv");
    solve_q9_with_dt(0.005, "resultats/q9_dt0005.csv");
    solve_q9_with_dt(0.001, "resultats/q9_dt0001.csv");
}
```

Question 10 Runge-Kutta d'ordre 4 pour l'oscillateur harmonique

On vient maintenant reproduire la même expérience, en utilisant cette fois-ci la méthode de Runge-Kutta d'ordre 4. On constate ainsi que l'erreur est inférieure à l'erreur par la méthode d'Euler, de plusieurs ordres de grandeur. En effet, si on constate que le problème de divergence est également présent, la méthode atténue fortement ces effets sur le court à moyen terme.

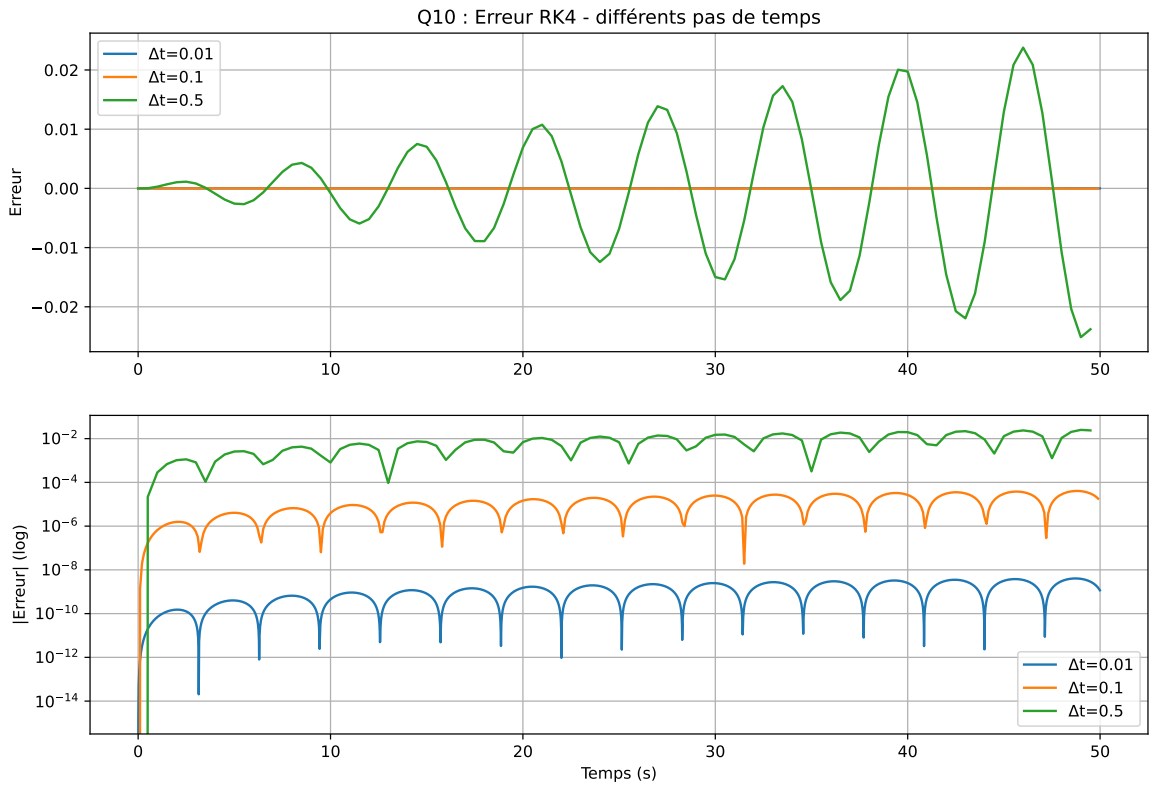

FIGURE 2 – Erreur en fonction du temps et du pas de temps pour RK4

TP1/q10.cpp

```cpp
void solve_q10_with_dt(double dt, const std::string& filename) {
    double Tmax = 50.0;
    int steps = static_cast<int>(Tmax / dt);
    const int n = 2;
    double state[n];
    state[0] = 1.0;
    state[1] = 0.0;
    double x0 = state[0];
    double v0 = state[1];
    std::ofstream fichier(filename);
    if (fichier.is_open()) {
        fichier << "t,x_numerique,x_analytique,erreur" << std::endl;
        for (int i = 0; i < steps; ++i) {
            double t = i * dt;
            double x_th = solution_analytique_q10(t, x0, v0, w0_q10);
            double erreur = state[0] - x_th;
            fichier << t << "," << state[0] << "," << x_th << "," << erreur << std::endl;
            rk4(n, t, state, dt, deriv_q10);
        }
        fichier.close();
    }
}

void solve_q10() {
    solve_q10_with_dt(0.01, "resultats/q10_dt001.csv");
    solve_q10_with_dt(0.1, "resultats/q10_dt01.csv");
    solve_q10_with_dt(0.5, "resultats/q10_dt05.csv");
}
```

Question 11 Runge-Kutta d'ordre 2

On vient maintenant créer la fonction rk2, marchant avec un prototype identique à "euler", en appliquant la méthode de Runge-Kutta d'ordre 2 :

TP1/q11.cpp

```
4  void rk2(int n, double x, double y[],
   ↪  double dx, void deriv(int, double,
   ↪  double[], double[])){
5      int i;
6      double ddx;
7      double k1[n], k2[n], yp[n];
8      ddx = dx / 2;
9
10     deriv(n, x, y, k1);
11     for (i = 0; i < n; i++) {
12         yp[i] = y[i] + k1[i] * ddx;
13     }
14
15     deriv(n, x + ddx, yp, k2);
16     for (i = 0; i < n; i++) {
17         y[i] = y[i] + k2[i] * dx;
18     }
19 }
```

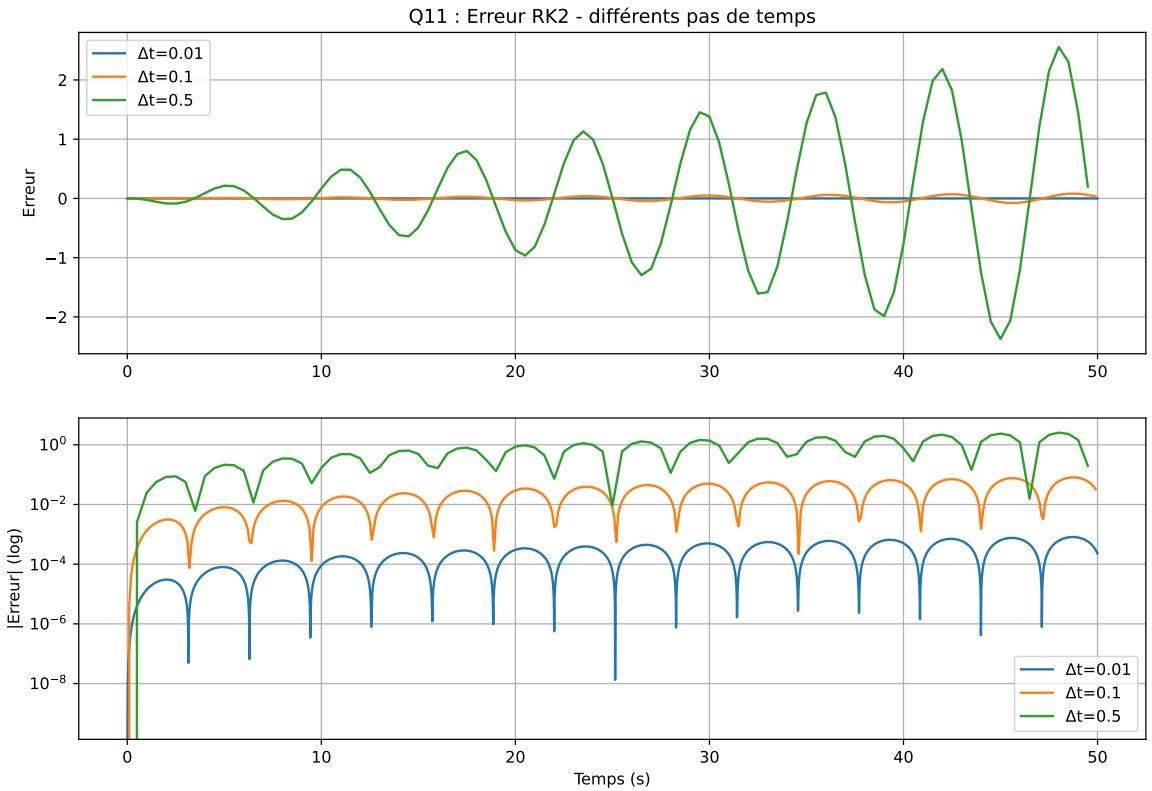

FIGURE 3 – Erreur en fonction du temps et du pas de temps pour RK2

TP1/q11.cpp

```cpp
void solve_q11_with_dt(double dt, const std::string& filename) {
    double Tmax = 50.0;
    int steps = static_cast<int>(Tmax / dt);
    // CI
    const int n = 2;
    double state[n];
    state[0] = 1.0;  // x0 = 1.0
    state[1] = 0.0;  // v0 = 0.0
    double x0 = state[0];
    double v0 = state[1];
    std::ofstream fichier(filename);
    if (fichier.is_open()) {
        fichier << "t,x_numerique,x_analytique,erreur" << std::endl;
        for (int i = 0; i < steps; ++i) {
            double t = i * dt;
            // Solution analytique
            double x_th = solution_analytique_q11(t, x0, v0, w0_q11);
            double erreur = state[0] - x_th;
            fichier << t << "," << state[0] << "," << x_th << "," << erreur << std::endl;
            rk2(n, t, state, dt, deriv_q11);
        }
        fichier.close();
    }
}

void solve_q11() {
    // Test avec différents pas de temps
    solve_q11_with_dt(0.01, "resultats/q11_dt001.csv");
    solve_q11_with_dt(0.1, "resultats/q11_dt01.csv");
    solve_q11_with_dt(0.5, "resultats/q11_dt05.csv");
}
```

Question 12 Comparaison des méthodes numériques

Enfin, nous pouvons comparer les 3 méthodes sur le cas de l'oscillateur harmonique. On constate ainsi que :

— Euler présente une erreur plus importante mais est de complexité algorithmique moindre.

— RK2 se place au milieu tant en terme d'erreur que de complexité algorithmique.

— RK4 est le meilleur en terme d'erreur mais est le plus complexe algorithmiquement parlant.

— Aucun des algorithmes ne prend pas en compte l'énergie et sa conservation.

Ainsi, des arbitrages sont à effectué entre une efficacité accrue (notamment pour la simulation temps réel) et une précision plus importante (pour les simulation physiques de pointe).

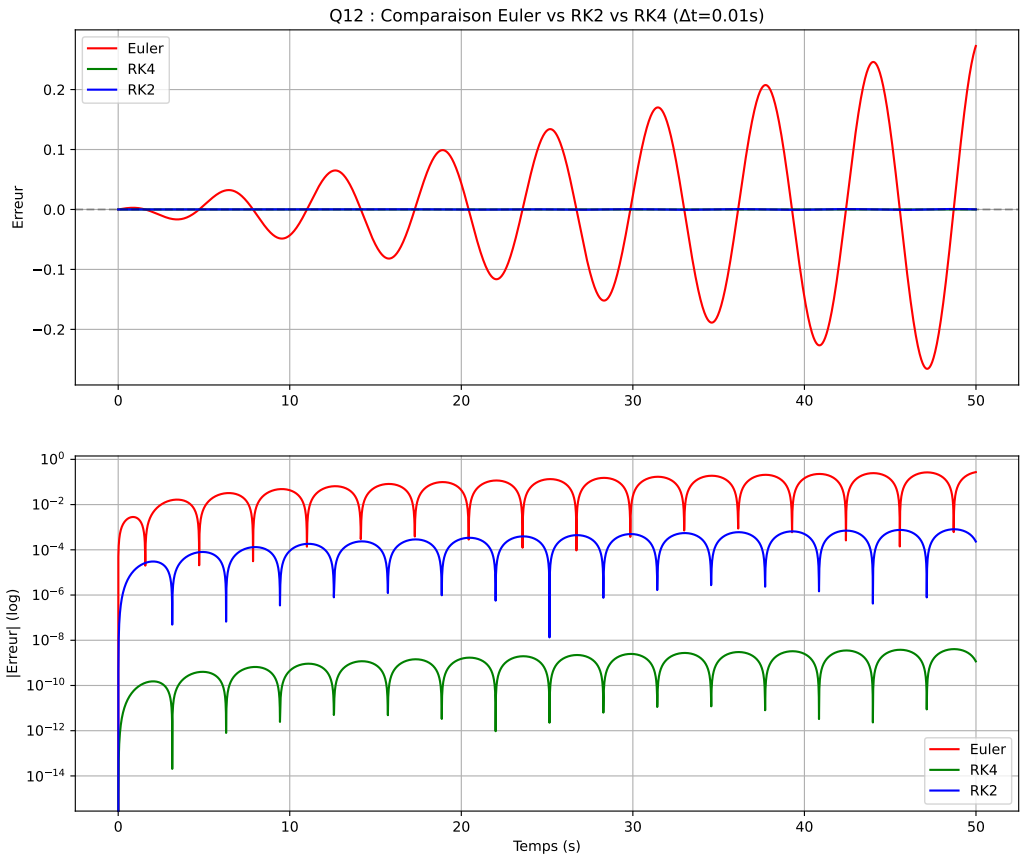

FIGURE 4 – Comparaison des algorithmies : Euler, RK2 et Rk4

Annexe

TP1/visualisation.py

```
18 def q1():
19     try:
20         data = np.loadtxt(os.path.join(RES_DIR,
           ↪  "q1.csv"), delimiter=',', skiprows=1)
21         plt.figure()
22         plt.plot(data[:, 0], data[:, 1], 'o-',
           ↪  label='Euler', markersize=4)
23         plt.plot(data[:, 0], data[:, 2], '--',
           ↪  label='Analytique', linewidth=2)
24         plt.title('Q1 : Désintégration simple - Euler
           ↪  vs Analytique')
25         plt.xlabel('Temps (s)')
26         plt.ylabel('Densité')
27         plt.legend()
28         plt.grid(True, alpha=0.3)
29         plt.savefig(os.path.join(FIG_DIR, "q1.pdf"),
           ↪  dpi=150, bbox_inches='tight')
30         plt.close()
31     except Exception as e:
32         print(f"Erreur Q1: {e}")
```

TP1/visualisation.py

```
34 def q2():
35     try:
36         data = np.loadtxt(os.path.join(RES_DIR,
           ↪  "q2.csv"), delimiter=',', skiprows=1)
37         plt.figure()
38         plt.plot(data[:, 0], data[:, 1], label='x(t)')
39         plt.plot(data[:, 0], data[:, 2], label='y(t)')
40         plt.title('Q2 : Désintégration en chaîne')
41         plt.xlabel('Temps (s)')
42         plt.ylabel('Densité')
43         plt.legend()
44         plt.grid(True, alpha=0.3)
45         plt.savefig(os.path.join(FIG_DIR, "q2.pdf"),
           ↪  dpi=150, bbox_inches='tight')
46         plt.close()
47     except Exception as e:
48         print(f"Erreur Q2: {e}")
```

TP1/visualisation.py

```
50 def q3():
51     try:
52         data = np.loadtxt(os.path.join(RES_DIR,
           ↪  "q3.csv"), delimiter=',', skiprows=1)
53         plt.figure()
54         plt.plot(data[:, 0], data[:, 1])
55         plt.title('Q3 : Oscillateur Harmonique')
56         plt.xlabel('Temps (s)')
57         plt.ylabel('Position')
58         plt.grid(True, alpha=0.3)
59         plt.savefig(os.path.join(FIG_DIR, "q3.pdf"),
           ↪  dpi=150, bbox_inches='tight')
60         plt.close()
61     except Exception as e:
62         print(f"Erreur Q3: {e}")
```

TP1/visualisation.py

```
64 def q8():
65     try:
66         data = np.loadtxt(os.path.join(RES_DIR,
           ↪  "q8.csv"), delimiter=',', skiprows=1)
67         fig, (ax1, ax2) = plt.subplots(1, 2,
           ↪  figsize=(14, 6))
68         ax1.plot(data[:, 0], data[:, 1], label='x')
69         ax1.plot(data[:, 0], data[:, 2], label='y')
70         ax1.set_title('Q8 : Évolution temporelle')
71         ax1.set_xlabel('Temps (s)')
72         ax1.legend()
73         ax1.grid(True, alpha=0.3)
74         ax2.plot(data[:, 1], data[:, 2])
75         ax2.set_title('Q8 : Trajectoire Cyclotron')
76         ax2.set_xlabel('x')
77         ax2.set_ylabel('y')
78         ax2.axis('equal')
79         ax2.grid(True, alpha=0.3)
80         plt.savefig(os.path.join(FIG_DIR, "q8.pdf"),
           ↪  dpi=150, bbox_inches='tight')
81         plt.close()
82     except Exception as e:
83         print(f"Erreur Q8: {e}")
```

TP1/visualisation.py

```
85 def q9():
86     try:
87         fig, axes = plt.subplots(2, 1, figsize=(12,
           ↪  8))
88         for dt, f in [(0.01, 'q9_dt001.csv'), (0.005,
           ↪  'q9_dt0005.csv'), (0.001,
           ↪  'q9_dt0001.csv')]:
89             try:
90                 data =
                   ↪  np.loadtxt(os.path.join(RES_DIR,
                   ↪  f), delimiter=',', skiprows=1)
91                 axes[0].plot(data[:, 0], data[:, 3],
                   ↪  label=f't={dt}')
92                 axes[1].semilogy(data[:, 0],
                   ↪  np.abs(data[:, 3]),
                   ↪  label=f't={dt}')
93             except:
94                 pass
95         axes[0].set_title("Q9 : Erreur Euler -
           ↪  différents pas de temps")
96         axes[0].set_ylabel('Erreur')
97         axes[0].legend()
98         axes[0].grid(True, alpha=0.3)
99         axes[1].set_xlabel('Temps (s)')
100        axes[1].set_ylabel('|Erreur| (log)')
101        axes[1].legend()
102        axes[1].grid(True, alpha=0.3)
103        plt.savefig(os.path.join(FIG_DIR, "q9.pdf"),
           ↪  dpi=150, bbox_inches='tight')
104        plt.close()
```

TP1/visualisation.py

```
108 def q10():
109     try:
110         fig, axes = plt.subplots(2, 1, figsize=(12,
            ↪  8))
111         for dt, f in [(0.01, 'q10_dt001.csv'), (0.1,
            ↪  'q10_dt01.csv'), (0.5, 'q10_dt05.csv')]:
112             try:
113                 data =
                    ↪  np.loadtxt(os.path.join(RES_DIR,
                    ↪  f), delimiter=',', skiprows=1)
114                 axes[0].plot(data[:, 0], data[:, 3],
                    ↪  label=f't={dt}')
115                 axes[1].semilogy(data[:, 0],
                    ↪  np.abs(data[:, 3]),
                    ↪  label=f't={dt}')
116             except:
117                 pass
118         axes[0].set_title("Q10 : Erreur RK4 -
            ↪  différents pas de temps")
119         axes[0].set_ylabel('Erreur')
120         axes[0].legend()
121         axes[0].grid(True, alpha=0.3)
122         axes[1].set_xlabel('Temps (s)')
123         axes[1].set_ylabel('|Erreur| (log)')
124         axes[1].legend()
125         axes[1].grid(True, alpha=0.3)
126         plt.savefig(os.path.join(FIG_DIR, "q10.pdf"),
            ↪  dpi=150, bbox_inches='tight')
127         plt.close()
128     except Exception as e:
129         print(f"Erreur Q10: {e}")
```

TP1/visualisation.py

```
131 def q11():
132     try:
133         fig, axes = plt.subplots(2, 1, figsize=(12, 8))
134         for dt, f in [(0.01, 'q11_dt001.csv'), (0.1, 'q11_dt01.csv'), (0.5, 'q11_dt05.csv')]:
135             try:
136                 data = np.loadtxt(os.path.join(RES_DIR, f), delimiter=',', skiprows=1)
137                 axes[0].plot(data[:, 0], data[:, 3], label=f't={dt}')
138                 axes[1].semilogy(data[:, 0], np.abs(data[:, 3]), label=f't={dt}')
139             except:
140                 pass
141         axes[0].set_title("Q11 : Erreur RK2 - différents pas de temps")
142         axes[0].set_ylabel('Erreur')
143         axes[0].legend()
144         axes[0].grid(True, alpha=0.3)
145         axes[1].set_xlabel('Temps (s)')
146         axes[1].set_ylabel('|Erreur| (log)')
147         axes[1].legend()
148         axes[1].grid(True, alpha=0.3)
149         plt.savefig(os.path.join(FIG_DIR, "q11.pdf"), dpi=150, bbox_inches='tight')
150         plt.close()
151     except Exception as e:
152         print(f"Erreur Q11: {e}")
```

TP1/visualisation.py

```
154 def q12():
155     try:
156         fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))
157         colors = {'q9_dt001.csv': 'red', 'q10_dt001.csv': 'green', 'q11_dt001.csv': 'blue'}
158         labels = {'q9_dt001.csv': 'Euler', 'q10_dt001.csv': 'RK4', 'q11_dt001.csv': 'RK2'}
159         for f, c in colors.items():
160             try:
161                 data = np.loadtxt(os.path.join(RES_DIR, f), delimiter=',', skiprows=1)
162                 ax1.plot(data[:, 0], data[:, 3], label=labels[f], color=c)
163                 ax2.semilogy(data[:, 0], np.abs(data[:, 3]), label=labels[f], color=c)
164             except:
165                 pass
166         ax1.set_title('Q12 : Comparaison Euler vs RK2 vs RK4 (deltat=0.01s)')
167         ax1.set_ylabel('Erreur')
168         ax1.legend()
169         ax1.grid(True, alpha=0.3)
170         ax1.axhline(0, color='k', linestyle='--', alpha=0.3)
171         ax2.set_xlabel('Temps (s)')
172         ax2.set_ylabel('|Erreur| (log)')
173         ax2.legend()
174         ax2.grid(True, alpha=0.3)
175         plt.savefig(os.path.join(FIG_DIR, "q12.pdf"), dpi=150, bbox_inches='tight')
176         plt.close()
177     except Exception as e:
178         print(f"Erreur Q12: {e}")
```