Résolution numérique d'équations différentielles du second ordre par la méthode de tir

Camil BENAMEUR
Maxence CONTANT
Lucas JACQUES
Tobias WENDL

Juin 2022

Table des matières

1 Introduction

Ce document traite de la résolution numérique d'équations différentielles du second ordre par la méthode de tir. Nous étudierons les effets d'une perturbation des conditions initiales ou de l'équation sur l'efficacité de cette méthode empirique. Cette méthode s'applique pour des problèmes aux limites d'équations du second ordre.

Nous commencerons par définir quelques termes importants à la bonne compréhension de notre projet. Ensuite, nous définirons la méthode de tir dans son ensemble en décrivant précisément son déroulement.

Pour finir, nous effectuerons plusieurs études de cas de cette méthode. Nous commencerons par l'étude d'un cas linéaire, sans perturbations (celui du cours). Celui-ci nous servira de cas de référence. Nous poursuivrons par une perturbation des conditions initiales et enfin par une perturbation de l'équation différentielle afin d'en observer leur impact sur la méthode de tir.

2 Définitions

1. **Problème aux limites :** Un problème aux limites est un problème constitué d'une équation différentielle dont on recherche une solution en prenant des valeurs imposées en des limites du domaine de définition.

2. **Interpolation :** L'interpolation est une opération mathématique qui consiste à déterminer une courbe ou fonction plus simple, qui coïncide avec la courbe ou fonction de départ en un nombre fini de points ou de valeurs.

   Ainsi, Il existe plusieurs type d'interpolations, comme l'interpolation de Lagrange, l'interpolation linéaire ou l'interpolation polynomiale. (ici, nous ne nous concentrerons que sur l'interpolation de Lagrange qui est un cas particulier d'interpolation polynomiale)

3. **Polynôme d'interpolation de Lagrange :** Le polynôme d'interpolation de Lagrange est un polynôme qui permet d'interpoler une série de points par un polynôme qui passe exactement par ces mêmes points, aussi appelés noeuds.

4. **Problème de Cauchy :** En analyse, un problème de Cauchy est un problème constitué d'une équation différentielle dont on recherche une solution vérifiant une, ou plusieurs, conditions initiales imposées en un point.

   Il est à noter que le théorème de Cauchy-Lipschitz garanti l'unicité d'une solution au problème de Cauchy.

5. **Perturbation de l'équation différentielle :** Une perturbation de l'équation différentielle consiste à ajouter à l'équation une perturbation telle qu'un polynôme.

6. **Perturbation des conditions initiales de l'équation :** Une perturbation des conditions initiales consiste à faire varier celles-ci afin d'étudier l'effet de cette variation sur le résultat.

3 La méthode de tir

3.1 Définition et description mathématique :

On peut définir la méthode de tir comme étant une méthode mathématique permettant de résoudre un problème aux limites en le réduisant à un problème aux conditions initiales.

Cela implique qu'il faut trouver une, ou des solutions, qui satisfassent le problème aux conditions initiales pour plusieurs conditions initiales différentes. On va essayer plusieurs solutions jusqu'à ce que l'une des solutions satisfasse à la fois le problème aux conditions initiales et le problème aux limites.

Ainsi la méthode de tir tient son nom du fait que l'on effectue des "tirs" pour différents angles à partir d'un point initial, jusqu'à que l'un de ces tirs atteigne la "cible" (la condition aux limites).

3.2 La méthode d'Euler :

La méthode d'Euler, ou algorithme d'Euler, permet de résoudre par approximation des équations différentielles du premier ordre de la forme :

$$\forall x \in I, y'(x) = f(x, y(x)), I \in \mathbb{R} \tag{1}$$

On considère un problème de Cauchy correspondant à un système de équations couplées.

$$\begin{cases} y' = g(x, y, z) \\ z' = f(x, y, z) \end{cases} \tag{2}$$

On soumet à ce système les conditions initiales $(y(x_0), z(x_0)) = (y_0, z_0)$ En partant de x_0, y_0 et z_0 on itère par les formules suivantes :

$$\begin{cases} x_{n+1} = x_n + h \\ y_{n+1} = y_n + hg(x_n, y_n, z_n) \\ z_{n+1} = z_n + hf(x_n, y_n, z_n) \end{cases} \tag{3}$$

Si le système d'équations couplées est issu d'une équation du second ordre de la forme :

$$\forall x \in I, y''(x) = f(x, y(x), y'(x)), I \in \mathbb{R} \tag{4}$$

on prendra $z_0 = y'_0$ et $g(x, y, z) = z$.

On part donc de $(x_0, y_0) = y(x_0)$ et $z_0 = y'(x_0)$ et on itère selon les formules suivantes :

$$\begin{cases} x_{n+1} = x_n + h \\ y_{n+1} = y_n + hz_n \\ z_{n+1} = z_n + hf(x_n, y_n, z_n) \end{cases} \tag{5}$$

3.3 Déroulement de la méthode de tir :

De manière générale, on sait que pour un problème de Cauchy donné il existe une solution unique que l'on peut calculer de manière approchée. Si l'on "couple" ce problème de Cauchy avec un problème aux limites, cela revient à imposer à la solution trouvée des conditions de la forme :

$$y(a) = y_a, y(b) = y_b \tag{6}$$

On peut alors poser le problème sous la forme suivante :

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = y_a \end{cases} \tag{7}$$

Une fois le problème posé, on commence par effectuer des "tirs" avec des valeurs $m_0, m_1, ..., m_k$. Chacun de ces tirs correspond à une valeur candidate de la dérivée de y en x_a, on intègre numériquement et on observe les valeurs $\theta(m_k)$ approchées en b.

Afin d'obtenir $\theta(m_k)$, on intègre numériquement en faisant appel à la méthode d'Euler. Pour chaque tir, on observe θ et on effectue le prochain tir. On saura que le bon tir a été effectué lorsque la cible a été dépassée. Par exemple, si notre premier tir est trop "long", on va effectuer des tirs en ajustant l'angle par incrément de sorte à ce que l'un des tirs soit trop "court".

On peut désormais calculer le polynôme d'interpolation de Lagrange $L(X)$ sur les noeuds $m_0, m_1, ..., m_k$ avec les valeurs $\theta(m_0), \theta(m_1), ..., \theta(m_k)$.

On rappelle :

$$L(X) = \sum_{i=0}^{k} \theta(m_i) L_i(X) \tag{8}$$

Avec :

$$L_i(X) = \prod_{j \in [0,n], j \neq i}^{k} \frac{X - m_j}{m_i - m_j} \tag{9}$$

On résout l'équation algébrique $L(X) = y_b$ qui nous fournit au moins une racine m^* correspondant à la valeur candidate de la dérivée de y en x_a donnant le tir le plus précis.

Pour finir, on va vérifier la cohérence de nos calculs en construisant la courbe intégrale solution correspondant à m^*. Pour cela, on intègre le problème de Cauchy suivant :

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = y_a \end{cases} \tag{10}$$

Il est à noter que la méthode de tir est une approche possible et qu'il existe d'autres méthodes plus directes afin de résoudre de tels problèmes.

4 Études de cas

4.1 Application sur un exemple linéaire

Appliquons la méthode de tir à un problème aux limites linéaire, sans aucune perturbation. Soit le problème aux limites suivant :

$$\left\{ y'' + y = 0, y(0) = 1, y(\frac{\pi}{2}) = \frac{1}{2} \right\} \tag{11}$$

Nous savons que ce problème pourrait être intégré directement nous donnant une solution de la forme :

$$y = Acos(x) + Bsin(x) \tag{12}$$

À l'aide de Mapple, nous effectuons différents tirs. Naturellement la première valeur à essayer est le taux d'accroissement de y entre 0 et $\frac{\pi}{2}$:

$$m_0 = \frac{y(\frac{\pi}{2}) - y(0)}{\frac{\pi}{2} - 0} = \frac{\frac{1}{2} - 1}{\frac{\pi}{2} - 0} = -\frac{1}{\pi} \tag{13}$$

On utilise l'algorithme d'Euler implémenté sur Mapple (forward Euler method) en utilisant la fonction dSolve afin d'intégrer numériquement l'équation et de déterminer $y_{m_0}(\frac{\pi}{2})$. On obtient $\theta(m_0) = -0.32$.

Nous allons ensuite procéder à différents tirs en faisant varier m_k de plus ou moins 0.5 jusqu'à dépasser $y(\frac{\pi}{2})$. Essayons $m_1 = m_0 - 0.5$. On trouve $\theta(m_1) = -0.82$. Or, nous cherchons à nous approcher au maximum de $\theta(m_k) = 0.5$. On va alors essayer les tirs $m_2 = m_0 + 0.5$ et $m_3 = m_0 + 1$ pour lesquels on obtient $\theta(m_2) = 0.18$ et $\theta(m_3) = 0.68$.

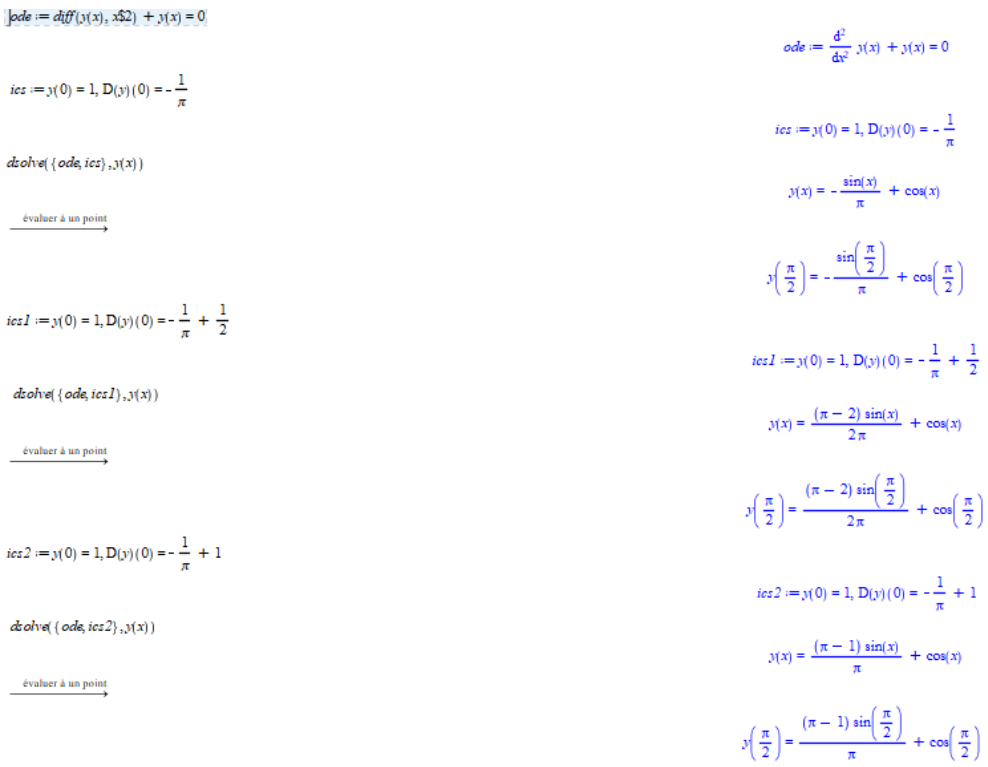

FIGURE 1 – Calcul des valeurs de $\theta(m_k)$ dans Mapple

On remarque que pour le tir m_3 on dépasse la cible. Cela signifie que l'on a tous les noeuds m_k ainsi que toutes les valeurs $\theta(m_k)$ nécessaires à l'interpolation de Lagrange.

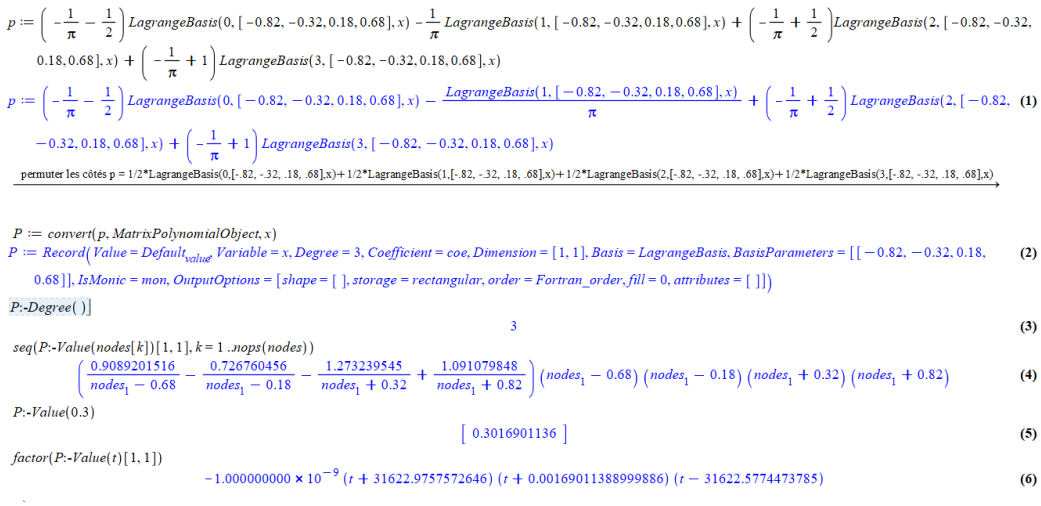

FIGURE 2 – Calcul du polynôme interpolateur de Lagrange avec Mapple

On obtient un polynôme P de degré $Deg(P) = 3$, cela correspond au fait que l'on ai effectué 4 tirs.
De plus $P(0.3) \approx 0.3$, notre polynôme interpolateur est donc proche de l'identité.
Finalement on obtient la forme factorisée de notre polynôme, mettant en évidence ses racines.

4.2 Transition vers python

Plusieurs éléments nous ont convaincu de nous lancer dans la réalisation d'un algorithme sur Python.

- Dans une certaine mesure, Python nous permet d'automatiser la méthode en créant un outil pratique d'application de la méthode de tir.

- Obtention d'une solution $\theta(m_k)$ approchée numériquement sous forme de flottant, et ceci même dans les cas de non linéarité, ce que nous ne sommes pas parvenu à faire avec Mapple. (cf. partie 4.5)

- Utilisation de nos résultats afin d'illustrer le raisonnement à l'aide de certains graphiques clés dans le déroulement de la méthode de tir.

4.3 Construction d'un algorithme sur Python

Notre but étant de construire un algorithme complet qui automatise la méthode de tir, nous avons commencé par construire une fonction Euler qui implémente directement l'algorithme d'Euler. Elle prend en argument une fonction f qui représente $y'' = f(x, y, y')$, ainsi que les conditions initiales aux points de coordonnées (x_a, y_a) et (x_b, y_b), et un entier n. Ce dernier représente le nombre de fois que l'on divise l'intervalle, afin d'obtenir h, la résolution de l'intégration numérique.

En sortie, la fonction Euler renvoie la valeur de $\theta(m_k)$ déterminée par intégration, et deux tableaux x_array et y_array contenant les abcisses et ordonnées des points d'étape de l'algorithme :

```
def Euler(f: Callable, xa: float, xb: float, ya: float, y1a: float, n: int):
    y_list = []
    h = (xb - xa) / float(n) # Pas de l'algorithme d'Euler

    x_list = np.linspace(xa,xb,n,endpoint=True)

    x = x_list[0]
    y = ya
    zn = y1a
    z_temp = zn


    for i in x_list:
        z_temp = zn
        zn += h * f(x, y, zn)
        y += h * z_temp
        x=i
        y_list.append(y)


    return ({"yb": y, "x_array":  x_list, "y_array": np.array(y_list)})
```

Pour la suite, nous commençons par déclarer toutes les variables qui nous seront utiles :

```
def f(x, y, y1): # Équation différentielle, de la forme y''=f(x,y,y')
    return -y

xa= 0 # Position de la première condition initiale xa
xb= pi/2 # Position de la seconde condition initiale xb

ya=3 # Valeur de y(xa)
yb=0 # Valeur de y(xb)


dy1a=abs(ya-yb)/2 # Pas de tir, si la cible est manqué ou est inatteignable
#modifier le pas de tir peut permettre l'obtention de meilleurs résultats

n= 1000 # Permet de définir le pas utilisé dans l'algorithme d'Euler h = (xb-xa)
```

Puis on les utilise dans une fonction *shot*() réalisant des tirs successifs :

```
def shot(f: Callable, xa: float, xb: float, ya: float, yb: float, dy1a: int, n:
    # dy1a = m1-m0 /  n
    tirs = []
    y1a = (yb-ya)/(xb-xa)    # Initialisation de m0 à la valeur
                             # du taux d'accroissement
    y1 = y1a
    yb_shot = Euler(f, xa, xb, ya, y1, n)["yb"]
    list_m = []
    list_yb = []

    preshot1 = Euler(f, xa, xb, ya, y1, n)["yb"]
    preshot2 = Euler(f, xa, xb, ya, y1+1/2, n)["yb"]
    is_growing = preshot2-preshot1>=0


    if yb_shot < yb and is_growing:  # on vérifie si le tir initial est
    #trop court ou trop long ainsi
    #la tendance de la courbe sur [xa,xb]
        while yb_shot < yb:  # on effectue des tirs par incrément tant que
        le tir
            #   est trop court en s'assurant de bien dépassé la cible
```

```
            new_shot = Euler(f, xa, xb, ya, y1, n)
            yb_shot = new_shot["yb"]
            tirs.append(new_shot)
            list_yb.append(yb_shot)
            list_m.append(y1)
            y1 += dy1a  # On incrémente la valeur de mk
    elif yb_shot > yb or not is_growing:
        while yb_shot > yb:
            new_shot = Euler(f, xa, xb, ya, y1, n)
            yb_shot = new_shot["yb"]
            tirs.append(new_shot)
            list_yb.append(yb_shot)
            list_m.append(y1)
            y1 -= dy1a  # On incrémente la valeur de mk

    return ({"tirs": tirs, "list_m": list_m, 'list_yb': list_yb, 'y1a': y1a})

new_shot = shot(f, xa,xb, ya, yb, dy1a, n)
tirs, list_m, list_yb, y1a = new_shot.values()
```

Nous créons ensuite deux graphiques. Dans le premier nous traçons tous les tirs. Nous traçons ensuite la droite reliant la cible et le point de départ dans les deux. Le second sera destiné au tir

```
# Création avec matplotlib.pyplot de deux graphiques sur une même fenêtre
figure, (graph1,graph2) = plt.subplots(2)


# Trace la courbe de chaque tir
for i, tir in enumerate(tirs):
    graph1.plot(tir["x_array"],tir["y_array"], label = 'Tir '+str(i+1))

# Nous traçons la droite passant par le point de départ du tir et la cible
print(len(list_m))
x = np.linspace(list_m[0], list_m[-1], n*5, endpoint=True) # Intervalle de
#définition [m0,...,mk] du polynôme interpolateur de Lagrange
x_droite = np.linspace(xa, xb, n*5, endpoint=True) # on définit
#l'intervalle de définition [xa,xb] de la droite reliant (xa,ya) et (xb,yb)
graph1.plot(x_droite, y1a*(x_droite-xa) + ya, label='Droite reliant les deux
point',c="red")
```

```
graph2.plot(x_droite, y1a*(x_droite-xa) + ya, label='Droite reliant les deux
point',c="red")
```

Ensuite, on détermine notre meilleur tir en utilisant un Polynôme interpolateur de Lagrange, l'égalité sera assumée à 3 décimales près. L'affiche du tir optimal sera ainsi fait sur la deuxième fenêtre :

```
interpolation_m = lagrange(list_m, list_yb) # Polynôme de Lagrange avec noeuds:
best_y1a=None
try:
    for k in x:

        if np.around(interpolation_m(k),2) == np.around(yb,2): # on cherche k te
            best_y1a = k

    if type(best_y1a)==NoneType: raise Exception("Une erreur est survenue. Essay

except Exception as e:
    print(e)
    sys.exit()

best_shot = Euler(f, xa, xb, ya, best_y1a, n)

arounded_y1a = around(best_y1a,3)

# Définition des titres des graphs
graph1.set_title('Graph des tirs effectués')
graph2.set_title('Graph du meilleur tir pour m='+str(arounded_y1a))
graph2.plot(best_shot["x_array"],best_shot["y_array"],label="Meilleur tir
pour m=" + str(arounded_y1a))

# On place les points de départ et d'arrivée des tirs
graph1.scatter(xa,ya, c="red",label="Point de départ du tir en
("+str(np.around(xa,2))+","+str(np.around(ya,2))+")")
graph2.scatter(xa,ya, c="red",label="Point de départ du tir en
("+str(np.around(xa,2))+","+str(np.around(ya,2))+")")

graph1.scatter(xb,yb, c="red",marker="x",label="Cible en
("+str(np.around(xb,2))+","+str(np.around(yb,2))+")")
graph2.scatter(xb,yb, c="red",marker="x",label="Cible en
("+str(np.around(xb,2))+","+str(np.around(yb,2))+")")
```

```
# Affichage des légendes et des graphiques
graph2.legend()
graph1.legend()
print('(m_k):' + str(np.around(list_yb, 2)))
print("La meilleure approximation de y'(a) est " + str(best_y1a))
plt.show()
```

4.4 Perturbation des conditions initiales

Dans cette partie, nous allons appliquer à nouveau la méthode de tir avec une perturbation des conditions initiales.

$$\left\{ y'' + y = 0, y(0) = 5, y(\frac{\pi}{2}) = \frac{1}{2} \right\} \tag{14}$$

Nous allons effectuer à l'aide de Python différents tirs. La première valeur de tir que l'algorithme va effectuer est le taux d'accroissement de y entre 0 et $\frac{\pi}{2}$. On trouve :

$$m_0 = \frac{y(\frac{\pi}{2}) - y(0)}{\frac{\pi}{2} - 0} = \frac{\frac{1}{2} - 5}{\frac{\pi}{2} - 0} = -\frac{9}{\pi} \tag{15}$$

La valeur de $y_{m_0}(\frac{\pi}{2})$ est ensuite déterminée par la fonction $Euler()$ et on obtient alors $\theta(m_0) = -2.89$.

L'opération sera donc répétée en faisant varier m_k de plus ou moins 0.5 jusqu'à trouver le bon $\theta(m_k)$. Le programme effectue donc les tirs suivants :

1. $m_1 = m_0 + 0.5$ qui donne $\theta(m_1) = -2.37$
2. $m_2 = m_0 + 1$ qui donne $\theta(m_2) = -1.87$
3. $m_3 = m_0 + 1.5$ qui donne $\theta(m_3) = -1.37$
4. $m_4 = m_0 + 2$ qui donne $\theta(m_4) = -0.87$
5. $m_5 = m_0 + 2.5$ qui donne $\theta(m_5) = -0.37$
6. $m_6 = m_0 + 3$ qui donne $\theta(m_6) = 0.14$
7. $m_7 = m_0 + 3.5$ qui donne $\theta(m_7) = 0.64$

On remarque que pour le tir m_7 on dépasse la cible et la fonction *shot* termine son exécution puisqu'elle a obtenue tous les noeuds m_k ainsi que toutes les valeurs $\theta(m_k)$ nécessaires à l'interpolation de Lagrange pour déterminer la meilleure valeur $m*$.

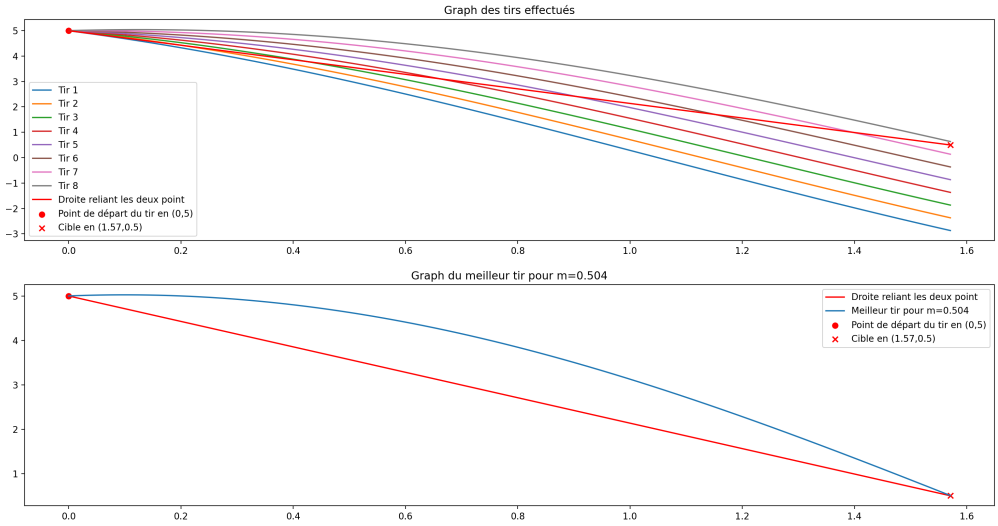

FIGURE 3 – Graph des tirs m_k obtenus avec python

On obtient donc notre meilleure approximation de solution pour $m^* = 0.504$ sur l'intervalle $[x_a, x_b]$.

Il est également possible de perturber les conditions initiales d'une autre manière. Observons ce qu'il se passe si nous perturbons $y(x_b)$.
Prenons par exemple $y(\frac{\pi}{2}) = 2$:

$$\left\{y'' + y = 0, y(0) = 1, y(\frac{\pi}{2}) = 2\right\} \tag{16}$$

Le programme calcule de nouveau le taux d'accroissement de y entre 0 et $\frac{\pi}{2}$.

$$m_0 = \frac{y(\frac{\pi}{2}) - y(0)}{\frac{\pi}{2} - 0} = \frac{2-1}{\frac{\pi}{2} - 0} = \frac{2}{\pi} \tag{17}$$

Les tirs pour des m_k variants de plus ou moins $\frac{1}{5}$ sont alors effectués. Les valeurs $\theta(m_0), ..., \theta(m_k)$ sont obtenues à l'aide de notre algorithme python :

```
θ(m_k):[0.64 0.84 1.04 1.24 1.44 1.64 1.84 2.04]
La meilleure approximation de y'(a) est 2.002452939000908
```

FIGURE 4 – Valeurs des différents $\theta(m_k)$ obtenus avec python

Voici le graphique des tirs réalisés ainsi que le meilleur tir :

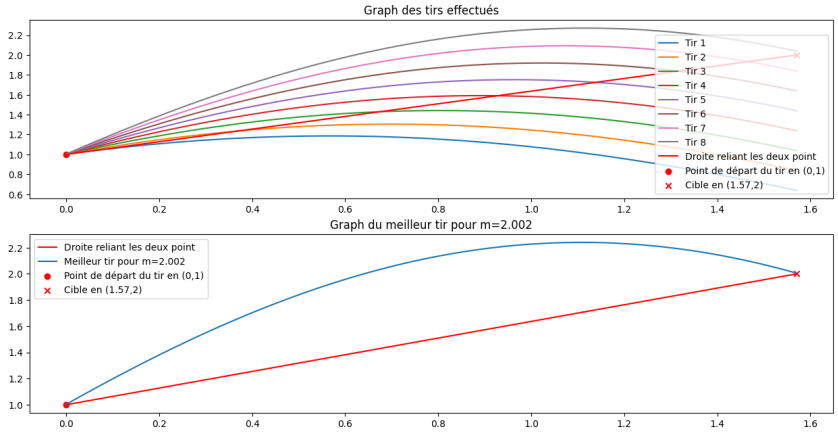

FIGURE 5 – Graph des tirs m_k obtenus avec python pour $y(\frac{\pi}{2}) = 2$

Après avoir perturbé $y(x_a)$ et $y(x_b)$, nous diminuons x_b, ceci revient à rapproché la cible. Prenons par exemple $x_b = \frac{\pi}{3}$

$$\left\{ y'' + y = 0, y(0) = 1, y(\frac{\pi}{3}) = \frac{1}{2} \right\} \tag{18}$$

Le taux d'accroissement m_0 de y entre 0 et $\frac{\pi}{3}$ est calculé :

$$m_0 = \frac{y(\frac{\pi}{3}) - y(0)}{\frac{\pi}{3} - 0} = \frac{\frac{1}{2} - 1}{\frac{\pi}{3} - 0} = -\frac{3}{2\pi} \tag{19}$$

m_k varie afin d'obtenir les tirs suivants :

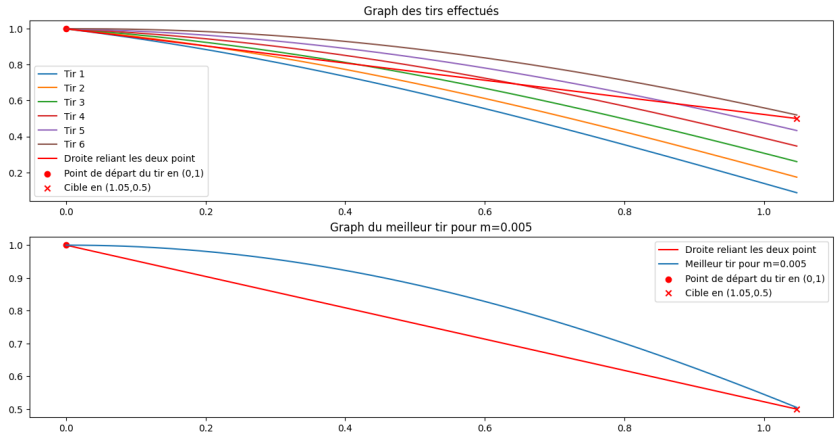

FIGURE 6 – Graph des tirs m_k obtenus avec python pour $y(\frac{\pi}{3}) = \frac{1}{2}$

Exceptions rencontrées lors de la perturbation des conditions initiales

Après avoir utilisé notre algorithme sur différentes perturbations de conditions initiales, nous nous sommes rendu compte que nous ne pouvions pas appliquer la méthode en utilisant x_a et x_b quelconques. Comme nous l'avons vu en (12), la solution à l'équation est 2π périodique ce qui implique que pour certains intervalles, la variation aux abords de x_b est trop faible et nécessite plus de précisions. Nous avions rencontré le même problème en utilisant Mapple. Nous développerons ce problème dans la partie 5.

4.5 Perturbation de l'équation différentielle

Dans cette partie, nous allons appliquer à nouveau la méthode de tir en perturbant l'équation différentielle. On fait cela en ajoutant un membre $\frac{1}{100}y^2$. Le but étant d'appliquer la méthode de tir à une équation que nous ne savons pas résoudre.

$$\left\{ y'' + y + \frac{1}{100}y^2 = 0, y(0) = 1, y(\frac{\pi}{2}) = \frac{1}{2} \right\} \tag{20}$$

Comme vu dans les parties précédentes, nous allons effectuer le premier tir en m_0. Avec $m_0 = -\frac{1}{\pi}$ puis nous ferons varier m_k de plus ou moins 0.5.

On utilise notre algorithme Python afin d'intégrer numériquement l'équation et de déterminer la meilleure solution approchée. Avec notre algorithme nous obtenons les tirs suivants :

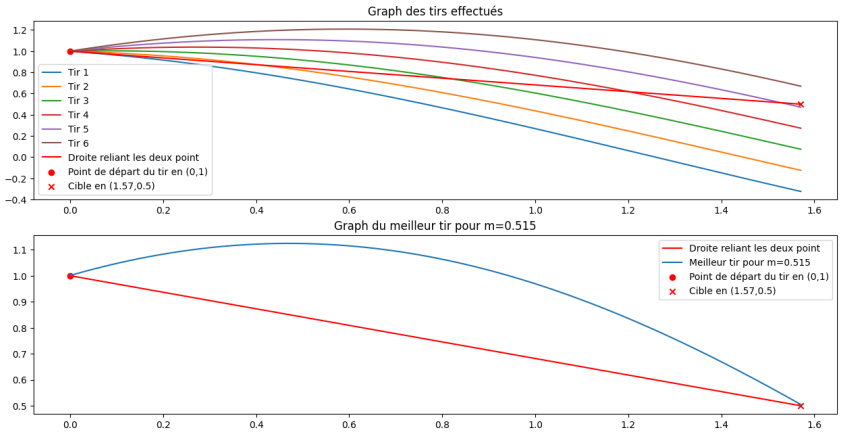

FIGURE 7 – Graph des tirs m_k obtenus avec python pour une perturbation de l'équation différentielle

On remarque que l'on obtient $m* = 0,515$. Dans le cas classique on obtient $m* \approx 0.5$.

4.6 Cas d'une EDL2 avec solution générale de forme exponentielle

On se propose d'étudier le système suivant afin de mettre à l'épreuve notre algorithme :

$$\begin{cases} y'' = y \\ y = Ae^{x} + Be^{-x} \end{cases} \tag{21}$$

Le choix de m_0 reste pertinent dans ce cas là, étant donné de la nature exponentielle de la solution et de notre intervalle, il semble logique que la solution numérique sera proche de la droite reliant la cible et le point de départ.

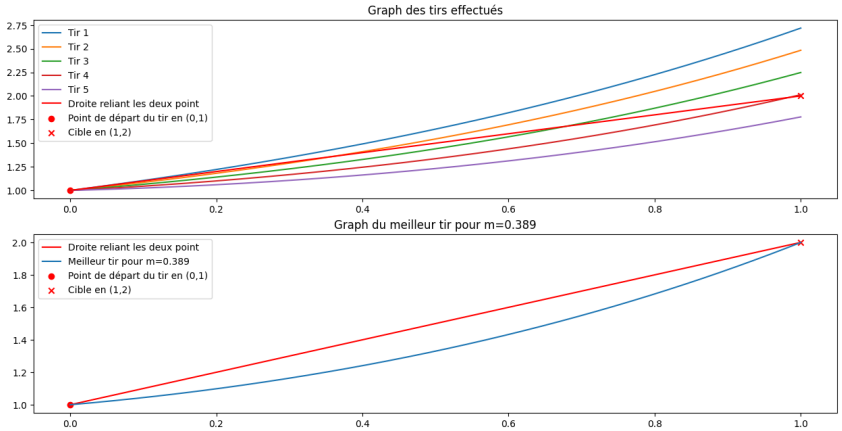

FIGURE 8 – Graph des tirs m_k obtenus avec python pour une solution de forme exponentielle.

5 Cas des solutions périodiques

Lorsque la solution générale est périodique, comme dans notre cas de référence, on se rend compte que l'on peut translater x_b de k période(s).

Effectuons cette translation en prenons x_b tel que :
$x_b = x_b + k * T$ avec $k \in \mathbb{Z}^*$ et $T = 2\pi$
On gardera $x_a = 0$

Dans cet exemple on prendra $k = 5$. On pourra également définir $y_a = 1$ et $y_b = 2$. En utilisant notre algorithme, on obtient le graphique suivant :

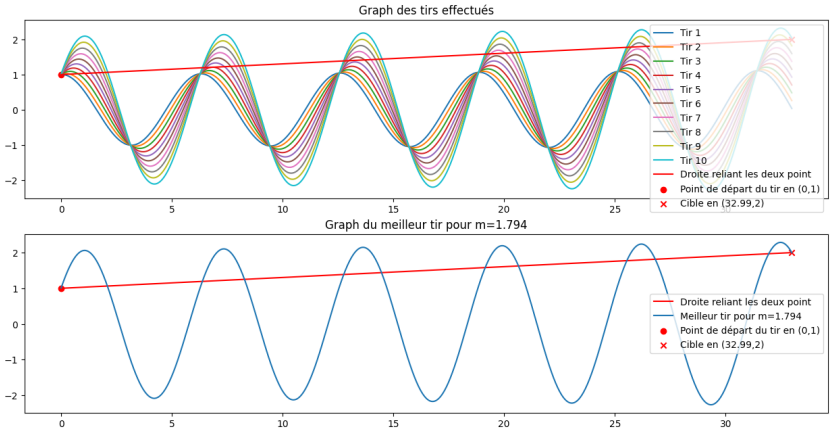

FIGURE 9 – Graph des tirs m_k obtenus avec python

Faire un tir sur $[x_a, x_b]$ revient à faire le même tir que sur $[x_a, x_b - kT]$ mais en observant k pseudo-périodes de notre meilleure approximation entre $[0, x_a + kT]$.
En effet le tir sur $[x_a, x_b]$ est observé presque à l'identique sur $[x_a + kT, x_b]$ hormis une erreur observée sur $y(x_a + kT)$ qui est différent de $y(x_a)$.
Cela est dû à des imprécisions de calcul numérique. C'est pour cela que l'on a du mal a exécuter le programme sur un intervalle $[2k\pi, 2\pi(1+k)]$, car pour ce faire $y(x_a + kT)$ doit être exactement égal à $y(x_a)$, ce qui nécessiterai une méthode d'intégration numérique bien plus précise possédant une résolution suffisante peu importe la taille de l'intervalle $[x_a, x_b]$.

6 Conclusion

Lors de nos travaux et expériences autour de la méthode de tir, nous avons pu nous rendre compte à quel point cette méthode était empirique. Elle permet de déterminer une solution approchée qui ne peut s'appliquer que localement. En partant de notre cas de référence et en lui appliquant des perturbations, nous avons pu noter un certain nombre d'effets.

6.1 Effets de la perturbation des conditions initiales

Nous avons perturbé les conditions initiales de deux manières. En premier lieu, nous avons modifié le $y(x_a)$, c'est à dire la hauteur à laquelle nous tirons. Ensuite nous avons modifié $y(x_b)$, c'est à dire la hauteur de notre cible. Ce type de perturbation ne pose pas de problème dans le cadre de l'application de la méthode de tir, mais elles augmentent, cependant, sensiblement l'erreur (cf. partie 4.1).

Cependant, lorsque nous perturbons l'intervalle $[x_a, x_b]$ et que y s'avère périodique, l'application de la méthode de tir peut nécessiter une très grande précision lors de l'intégration numérique, ce qui rend cette méthode difficilement applicable.

Nous noterons que nous n'avons pas pu effectuer des tests sur tous les cas possibles.

6.2 Effets de la perturbation de l'équation différentielle

Pour finir, nous avons étudié les effets observés après une perturbation de l'équation. C'est dans cette optique que nous avons eu l'idée d'ajouter un membre $\frac{1}{100}y^2$ à notre équation de référence. Cet ajout a eu pour effet de créer une erreur. Notre algorithme nous permet de résoudre l'équation composée du cas linéaire et de ce membre ajouté. Ainsi nous obtenons $m* = 0,515$. Dans le cas de référence (11) on obtenait $m* \approx 0.5$, on a donc une erreur qui résulte de notre perturbation. On a pu vérifier que cette erreur est d'autant plus grande que le coefficient devant le y^2 est grand.

6.3 Axes d'amélioration et pistes de recherches

Afin d'améliorer l'efficacité de notre programme, nous pourrions commencer par améliorer l'algorithme d'intégration numérique. Par exemple nous pourrions remplacer l'algorithme d'Euler par une méthode de Runge-Kutta d'ordre 4.

Par ailleurs, nous pourrions améliorer notre fonction *shot*. Enfin, nous pourrions utiliser d'autres méthodes d'interpolation qui nous permettraient d'obtenir une solution approchée plus proche de la véritable solution.

7 Annexes

1. Lien du dépôt Github pour le programme en Python :
   https://github.com/camilbenameur/shoting_method
2. Feuilles de calcul Mapple en pièce jointe de l'email.