Rapport : Projet maths

1 Partie Mathématiques

1.1 Dérivée de Gateau $J'(u, \delta u)$

- Déterminer la dérivée de Gâteau de la fonction $x(u)$.

  On a :
  $$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ x(0) = x_0 \end{cases}$$
  Et la formule de la dérivée de Gâteau :
  $$f'(u, v) = \lim_{\lambda \to 0^+} \frac{f(u + \lambda) - f(u)}{\lambda}.$$
  On cherche :
  $$\begin{cases} \dot{x}'(u, \delta u)(t) \\ x'(0) \end{cases}$$
  Or $x'(0) = 0$
  Et :
  $$\begin{aligned} \dot{x}'(u, \delta u)(t) &= \lim_{\lambda \to 0^+} \frac{\dot{x}(u, \delta u)(t) - \dot{x}(u)(t)}{\lambda} \\ &= \lim_{\lambda \to 0^+} \frac{Ax(u, \delta u)(t) + B(u, \delta u)(t) - [Ax(u)(t) + B(u)(t)]}{\lambda} \\ &= \lim_{\lambda \to 0^+} \frac{A[x(u, \delta u)(t) - x(u)(t)] + B(u, \delta u)(t) - B(u)(t)}{\lambda} \\ &= Ax'(u, \delta u)(t) + \lim_{\lambda \to 0^+} \frac{B(u)(t) + B(\lambda \delta u)(t) - B(u)(t)}{\lambda} \\ &= Ax'(u, \delta u)(t) + B(\delta u)(t) \end{aligned}$$
  On a donc la dérivée de Gâteau de la fonction $x(u)$:
  $$\begin{cases} \dot{x}'(u, \delta u)(t) = Ax'(u, \delta u)(t) + B(\delta u)(t) \\ x'(0) = 0 \end{cases}$$

- En déduire la dérivée de Gâteau de J.

  $$\begin{aligned} J'(u, \delta u)(t) &= \lim_{\lambda \to 0^+} \frac{J(u, \lambda \delta u)(t) - J(u)(t)}{\lambda} \\ &= \lim_{\lambda \to 0^+} \int_0^T \frac{\epsilon}{2}(u + \lambda \delta u)^2 \, \mathrm{d}s + \frac{1}{2}x^2(u + \lambda \delta u)(t) - \int_0^T \frac{\epsilon}{2}u^2(t) \, \mathrm{d}s - \frac{1}{2}x^2 u(t) \\ &= \lim_{\lambda \to 0^+} \frac{1}{\lambda}[\int_0^T \frac{\epsilon}{2}(u^2(s) + \lambda^2 \delta u^2(s) \, \mathrm{d}s) - \int_0^T \frac{\epsilon}{2}u^2(s)] \, \mathrm{d}s \\ &= \lim_{\lambda \to 0^+} \frac{1}{\lambda}[\frac{1}{2}x^2(u + \delta u)(t) - \frac{1}{2}x^2(u)(t)] \end{aligned}$$
  On identifie par rapport à la dérivée de Gâteau à la constante près, d'où :
  $$\begin{aligned} J'(u, \delta u)(t) &= \lim_{\lambda \to 0^+} \int_0^T \frac{\epsilon}{2} \frac{\lambda^2 \delta u^2(s) + 2\lambda u(s) \delta u(s)}{\lambda} \, \mathrm{d}s + [\frac{1}{2}x^2 u(t)]' \\ &= \lim_{\lambda \to 0^+} \int_0^T \frac{\epsilon}{2}[\lambda \delta u^2(s) + 2u(s) \delta u(s)] \, \mathrm{d}s + x'(u, \delta u)(t)x(t) \\ &= \int_0^T \epsilon u(s) \delta u(s) \, \mathrm{d}s + x'(u, \delta u)(t)x(t) \end{aligned}$$

- Déterminer la fonction $g(t)$.

  En prenant la formule de la dérivée de Gâteau appliquée pour la fonction J on a :

$$J'(u,\delta u)(t) = \lim_{\lambda \to 0^+} \frac{J(u,\delta u)(t) - J(u)(t)}{\lambda}$$

  D'après la dérivée de Taylor-Young qui fait intervenir le gradient, on a :

$$J(u,`\lambda u)(t) = J(u)(t) + \lambda(\nabla J(u)(t), \delta u) + o(||\lambda \delta u||)$$

$$\Leftrightarrow J(u,`\lambda u)(t) - J(u)(t) = \lambda(\nabla J(u)(t), \delta u)$$

  En remplaçant on obtient :

$$\begin{aligned} J'(u,\delta u)(t) &= \lim_{\lambda \to 0^+} \frac{\lambda(\nabla J(u)(t), \delta u)}{\lambda} \\ &= (\nabla J(u)(t), \delta u) \\ &= \int_0^T \nabla J(u)(s)\delta u(s)\,\mathrm{d}s \end{aligned}$$

  Par identification on voit donc que $g(t) = \nabla J$

1.2 Calcul de ∇J

- Afin de déterminer le gradiant de J, on introduit l'équation différentielle rétrograde

$$\begin{cases} \dot{p}(t) = A^T p(t) \\ p(T) = x(T). \end{cases}$$

  Montrons que :

$$\nabla J(u) = \varepsilon u + B^T p.$$

- On a donc :

$$\int_0^T \dot{p}(t)x'(u,\delta u)(t)dt = \int_0^T -A^T p(t)x'(u,\delta u)dt \tag{1}$$

  Avec $x'(u,\delta u) = Ax'(u,\delta u)(t) + B(\delta u)(t)$
  On fait donc une Intégration par parties sur (1) (partie droite) :

$$\begin{array}{cc} u(x) = x'(u,\delta u) & v'(x) = \dot{p}(t)) \\ u'(x) = \dot{x}'(u,\delta u) & v(x) = p(t) \end{array}$$

  Ainsi :

$$[x'(u,\delta u)(t)p(t)]_0^T - \int_0^T \dot{x}'(u,\delta u)p(t)$$

$$(x'(u,\delta u)(T)p(T) \underbrace{-x'(u,\delta u)(0)p(0))}_{0} - \int_0^T \dot{x}'(u,\delta u)p(t)$$

  Avec l'égalité d'avant, on a :

$$x'(u,\delta u)(T)p(T) - \int_0^T \dot{x}'(u,\delta u)p(t) = \int_0^T -A^T p(t)x'(u,\delta u)dt$$

$$x'(u,\delta u)(T)p(T) = \int_0^T -A^T p(t)x'(u,\delta u) + \int_0^T \dot{x}'(u,\delta u)p(t)$$

$$x'(u,\delta u)(T)p(T) = \int_0^T -A^T p(t)x'(u,\delta u) + \dot{x}'(u,\delta u)p(t)\,\mathrm{d}t$$

$$x'(u,\delta u)(T)p(T) = \int_0^T -A^T p(t)x'(u,\delta u) + p(t)[Ax'(u,\delta u)(t) + B(\delta u)(t)]\,\mathrm{d}t$$

$$x'(u,\delta u)(T)p(T) = \int_0^T -p(t)Ax'(u,\delta u) + p(t)Ax'(u,\delta u)(t) + p(t)B(\delta u)(t)\,\mathrm{d}t$$

$$x'(u,\delta u)(T)p(T) = \int_0^T p(t)B(\delta u)(t)\,\mathrm{d}t$$

Or :

$$J'(u,\delta u)(t) = \int_0^T \epsilon u(s)\delta u(s) + x'(u,\delta u)(t)x(t)\,\mathrm{d}t$$

Et donc :

$$\int_0^T B^T p(t)\delta u\,\mathrm{d}t = J'(u,\delta u) - \int_0^T \varepsilon u\delta u\,\mathrm{d}t$$

$$\int_0^T \delta u(B^T p(t) + \varepsilon u)\,\mathrm{d}t = J'(u,\delta u)(t)$$

$$= \int_0^T \nabla J\delta u\,\mathrm{d}t$$

- On peut donc en conclure que :

$$\nabla J = \varepsilon u + B^T p(t)$$

2 Partie Informatique - Résolution numérique

Afin de répondre à la problématique, modéliser les données et pouvoir répondre à la question, nous avons choisi d'utiliser le langage python.

Nous utilisons les conditions initiales de l'énoncé

```
######################
# Initialize constants #
######################

print "Initialize constants"

T = 1.0
w = 2*pi/T
epsilon=10**-3
NT = 500
dT = T/NT
ro = 0.03

x = np.transpose(np.atleast_2d([0.1, 0, 0, 0]))
u = np.transpose(np.atleast_2d([0, 0]))

A = np.matrix([[0, 1, 0, 0],
               [3*(w**2), 0, 0, 2*w],
               [0, 0, 0, 1],
               [0, -2*w, 0, 0]])

B = np.matrix([[0, 0],
               [1, 0],
               [0, 0],
               [0, 1]])
```

Le programme présente plusieurs parties. Le main itère un nombre donné de fois pour converger vers une solution de déplacement de plus en plus précise et minimale. La solution finale de u est utilisée comme valeur de départ de la prochaine itération.

```
# Iterates and call the other functions
def main():
    # Uses the global u vector
    global u
    # Iterates and call the functions to get closer and closer to the best solution
    print "Starts iterating"
    for i in range(0, NT):
        if (i%50 == 0):
            print "Iteration number " + str(i)
        xTab = firstEquation(x, u)
        pTab = secondEquation(xTab[NT])
        grad = gradient(pTab[NT-i])
        # We initialize the next u
        u = u-ro*grad
```

A chaque itération, le main doit appeler une fonction qui renvoie une solution à une équation d'Euler permettant d'obtenir X. Pour obtenir cette solution, il est necessaire d'itérer pour converger vers celle-ci. Nous avons fait une fonction qui renvoie la solution de f(x, u) necessaire pour calculer Euler au rang k.

```
############################
# Solve the first equation #
############################

# Returns xk : the solution of the current step of the euler equation
def calculateF(xk, uk):
    return np.dot(A, xk) + np.dot(B, uk)

# Returns xn : the array of the solutions of the euler equation
def firstEquation(xk, uk):
    xTab=[xk]
    for k in range(0, NT):
        res=calculateF(xk,uk)
        xk = xk+dT*res
        xTab.append(xk)
    return xTab
```

Cette solution sert de condition initiale pour résoudre l'équation d'Euler rétrograde. Euler rétrograde, à l'instar d'euler, itère également en convergeant vers la solution. A chaque itération une fonction qui calcule g(x, u) est appellée.

```
############################
# Solve the second equation #
############################

# Returns pk : the solution of the current step of the backward euler equation
def calculateG(xk):
    return np.dot(-1*A.T, xk)

# Returns pn : the array of the solutions of the backward euler equation
def secondEquation(xk):
    pTab=[xk]
    for k in range(0, NT):
        res = calculateG(xk)
        xk = xk-dT*res
        pTab.append(xk)
    return pTab
```

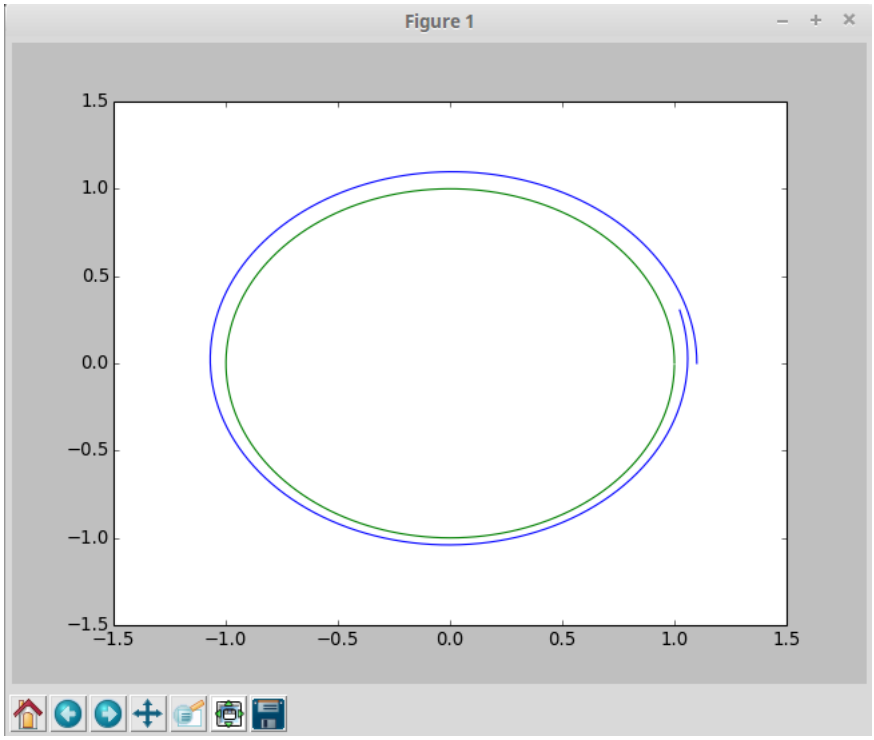
Enfin cette solution est utilisée dans une fonction qui récupère le gradient à la kième itération du *main*.

```
############
# Gratient #
############

# Returns uk : the current gradient value
def gradient(p):
    return epsilon*u + np.dot(B.T, p)
```

Au bout des 500 itérations du main, nous récupérons donc un tableau de positions X qui est converti dans le repère circulaire de l'ISS.

```
# Put the results in tabs and modify the values to put them in the circular base
print "Retrieving values"
x1, y1, x2, y2 = [], [], [], []
for i in range(0, NT):
    # These arrays represent the mouvement of the rocket
    x1.append(newCoordX(float(xTab[i][0][0]), float(xTab[i][2, 0]), i))
    y1.append(newCoordY(float(xTab[i][0][0]), float(xTab[i][2, 0]), i))
    # Second array is to print the circular mouvement of the ISS Station
    x2.append(cos(w*i*dT))
    y2.append(sin(w*i*dT))

# Prints the results
print "Printing values"
plt.plot(x1, y1)
plt.plot(x2,y2)
plt.show()
print "End of program"
```

Puis ce tableau est affiché dans un graphique (en bleu). Nous avons également affiché la trajectoire de l'ISS (en vert).



Nous pouvons observer que la courbe converge légèrement. Après de nombreux tests, nous avons remarqué que la courbe est très sensible aux valeurs initiales. La courbe semble au mieux avec une matrice A qui a pour première ligne 0, 0, 1, 0. Il serait également intéressant de faire partir le satellite du centre du cercle vert mais nous ne sommes pas parvenu à avoir une courbe propre., d'autant plus qu'il faudrait tenir compte d'autres paramètres tels que la force de gravité terrestre, la constante de gravitation, etc.