Notebook zéros certifiés stage

August 18, 2020

1 Les zéros de la fonctions cosinus

```
In [109]: from ore_algebra import *
          from ore_algebra import DifferentialOperators
          from sage.rings.rational_field import QQ
```

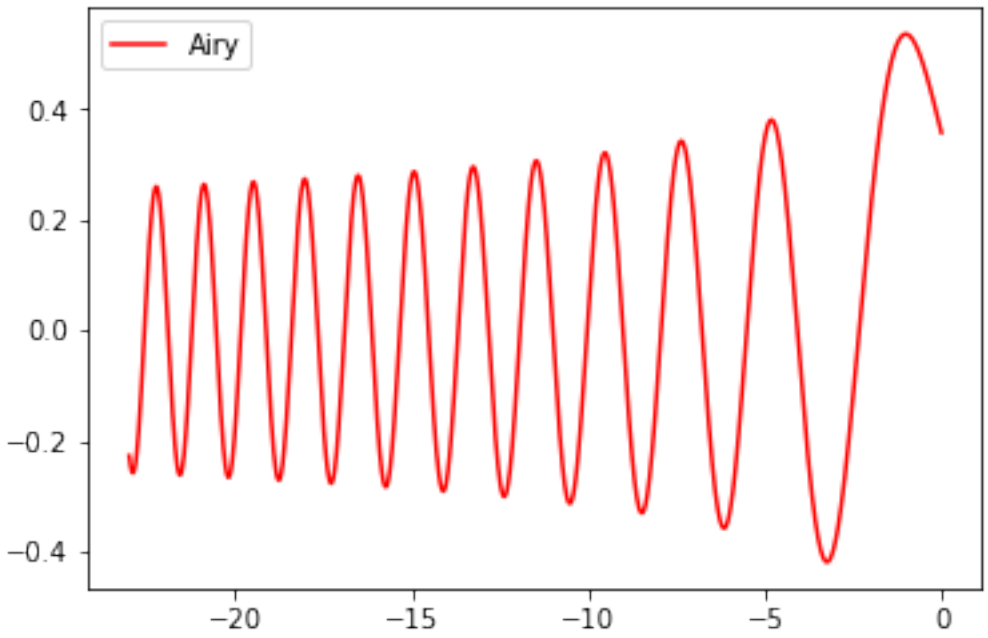
On appelera f_k une fonction avec pour opérateur différentiel dop_k et conditions initiales ini_k.

```
In [110]: Rx.<x> = QQ['x']
          A.<Dx> = OreAlgebra(Rx, 'Dx')
          dop_1=Dx^2+1
          ini_1=[1,0]

In [111]: %time find_certified_zeros(dop1,ini1,[-10,30])

CPU times: user 5.33 s, sys: 15 ms, total: 5.34 s
Wall time: 5.34 s


Out[111]: [[[-7.8539818, -7.8533567],
            [-4.7213772, -4.6810719],
            [-1.5750421, -1.5549629],
            [1.5663275, 1.5864036],
            [4.7047991, 4.7451431],
            [7.8533740, 7.8583805],
            [10.990719, 11.010790],
            [14.135336, 14.155449],
            [17.275146, 17.295234],
            [20.415984, 20.436062],
            [23.561710, 23.564211],
            [26.699682, 26.719767],
            [29.844751, 29.847253]],
           []]
```

2 Les limites de l'algorithme: la fonction d'Airy

On calcule avec les conditions initiales à grande précision

```
In [112]: dop_2=Dx^2-x
          ini_2=[RealBallField(500)((gamma(2/3)*3^(2/3))^(-1)),RealBallField(500)(-(gamma(1/3)*3

In [143]: plot_function_Airy(dop_2,ini_2,"Airy",'red')
```




```
In [113]: %time find_certified_zeros(dop2,ini2,[-4,0])

CPU times: user 78 ms, sys: 0 ns, total: 78 ms
Wall time: 72.9 ms


Out[113]: [[], []]
```

On remarque que l'algorithme de bissection-exclusion exclut tous les zéros de la fonction. C'est lié à la forme de l'équation différentielle Dx^2-x, l'opérateur de récurrence correspondant (n+1)(n+2)Sn^3-1 est d'ordre 3. Voici une piste de réponse:

```
In [114]: dop_2.power_series_solutions(10)

Out[114]: [x + 1/12*x^4 + 1/504*x^7 + O(x^10),
           1 + 1/6*x^3 + 1/180*x^6 + 1/12960*x^9 + O(x^10)]
```

Renvoie une base de solutions de la fonction pour les conditions initiales [0,1] et [1,0]. C'est cette base que nous utilisons dans notre algorithme pour calculer le développement en série de Taylor de notre fonction. Les développement de notre fonction n'ont donc pas de coeffcients de la forme 3n+2, or ce n'est pas le cas pour tous les x différents de zéro. La méthode pour calculer le développement en série de Taylor de notre algorithme est lacunaire et devrait être fixée. Nous nous limiterons aux fonctions dont l'ordre de l'équation récurrence sur les coefficients est égale à l'ordre de l'équation différentielle.

3 Temps d'exécution sur plusieurs fonctions D-finies

```
In [115]: def time(dop,ini,segm):
              t=cputime()
              find_certified_zeros(dop,ini,segm)
              return cputime(t)
```

On introduit ici quelques fonctions: f_3, f_4 et f_5 définies par leur opérateur dop_i et leurs conditions initiales ini_i.

```
In [116]: dop_3=Dx^3+Dx^2+1
          ini_3=[1,1,1]
          dop_4=((1/10)*x^2-50)*Dx^4-x*Dx+1
          ini_4=[1,-2,3,-4]
          dop_5=Dx^5+10*x^2*Dx^3-1
          ini_5=[1,-10,10,3,-5]

In [117]: X=[1/2+k/2 for k in range(45)]
```

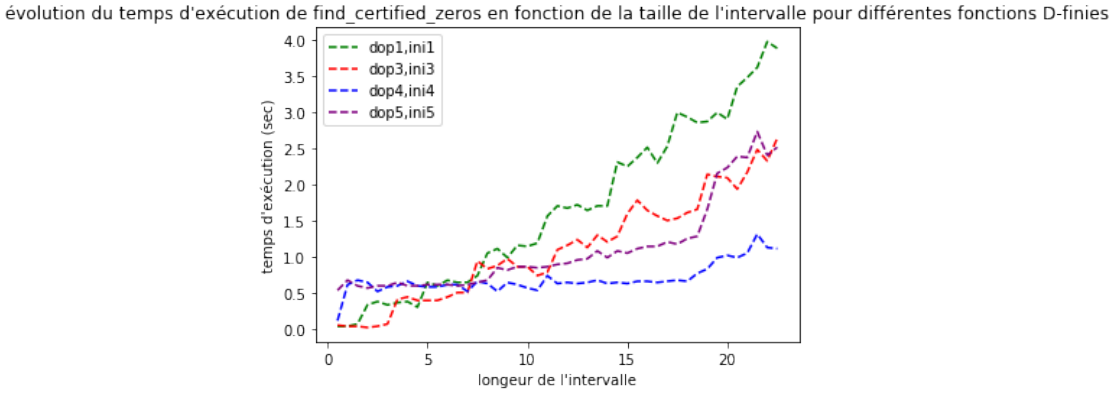
C'est le vecteur correspondant à la taille des segments.

```
In [105]: inc1=[]
          inc2=[]
          inc3=[]
          inc4=[]
          taille=1/2
          for i in range (45):
              inc1.append(time(dop_1,ini_1,[0,1/2+i/2]))
              inc2.append(time(dop_3,ini_3,[0,1/2+i/2]))
              inc3.append(time(dop_4,ini_4,[0,1/2+i/2]))
              inc4.append(time(dop_5,ini_5,[0,1/2+i/2]))
```

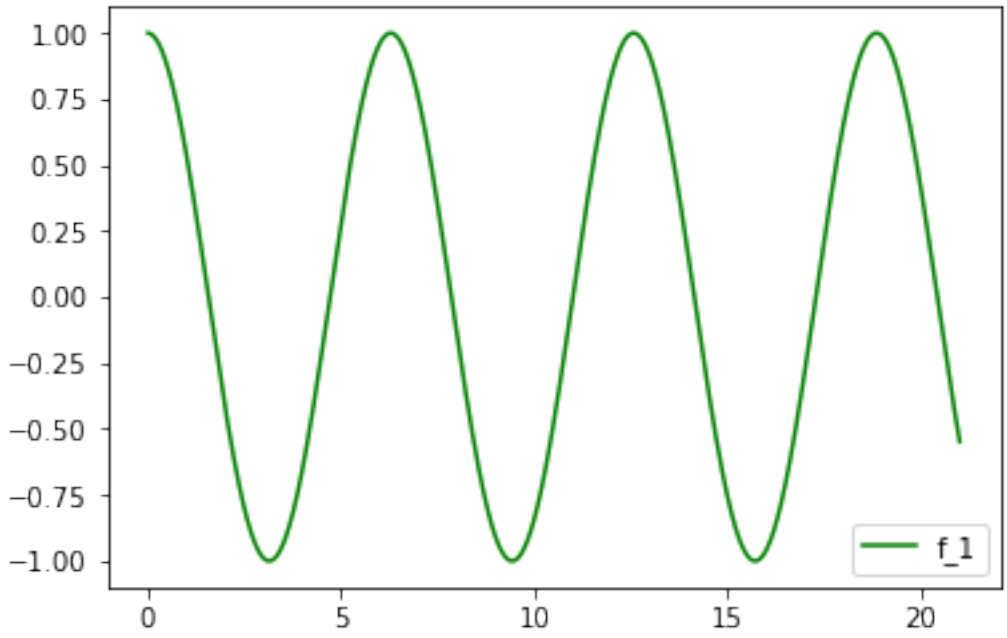
On crée ici les listes des temps de calcul de la fonction find_certified_zeros pour les différentes fonctions f_i

```
In [82]: %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt

In [106]: plt.xlabel("longeur de l'intervalle")
          plt.ylabel("temps d'exécution (sec)")
```

```
plt.title("évolution du temps d'exécution de find_certified_zeros en fonction de la ta
plt.plot(X,inc1,color='green', ls="--",label="f_1")
plt.plot(X,inc2,color='red',ls="--",label="f_3")
plt.plot(X,inc3,color='blue',ls="--",label="f_4")
plt.plot(X,inc4,color='purple',ls="--",label="f_5")
plt.legend()
plt.show()
```



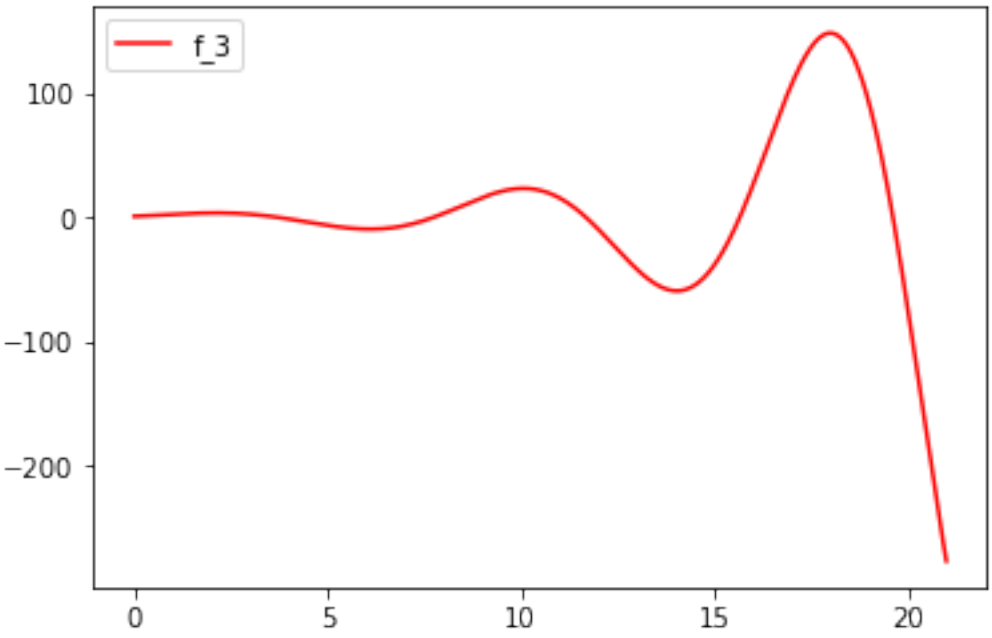
In [144]: `plot_function(dop_1,ini_1,"f_1",'green')`



In [148]: `find_certified_zeros(dop_1,ini_1,[0,21])`

```
Out[148]: [[[1.5707952, 1.5715013],
            [4.7105252, 4.7210451],
            [7.8535723, 7.8561986],
            [10.995190, 11.000447],
            [14.136867, 14.138180],
            [17.278759, 17.278762],
            [20.419766, 20.422392]],
           []]
```

```
In [142]: plot_function(dop_3,ini_3,"f_3",'red')
```

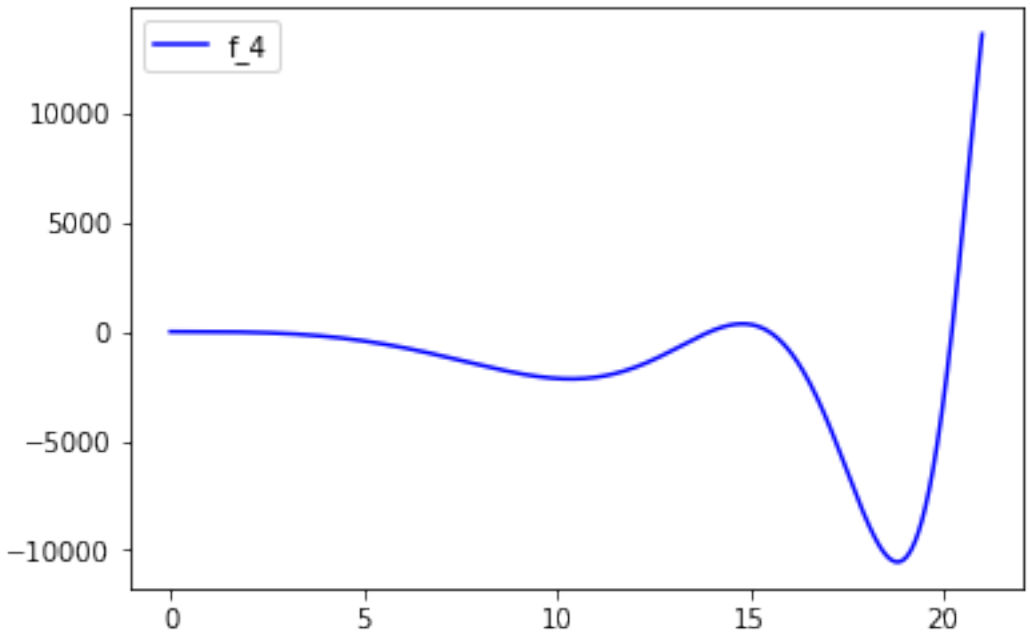



```
In [147]: find_certified_zeros(dop_3,ini_3,[0,21])
```

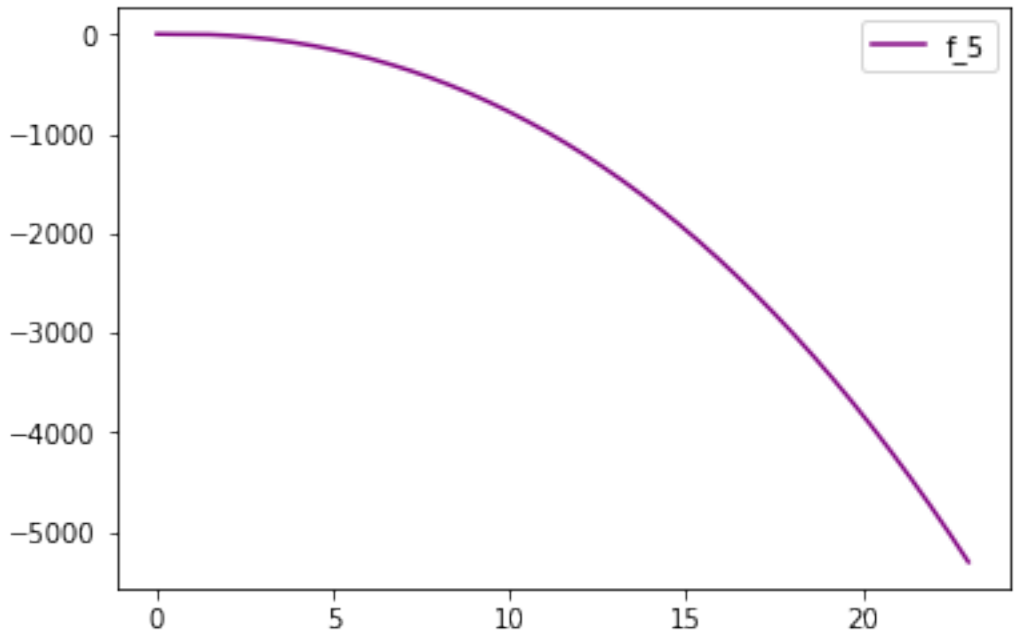
```
Out[147]: [[[3.0728583, 3.0734731],
            [7.0359181, 7.0411731],
            [14.963465, 14.968719],
            [18.927936, 18.930562]],
           []]
```

Le résultat renvoyé semble vraiment étonnant.

```
In [146]: plot_function(dop_4,ini_4,"f_4",'blue')
```




```
In [149]: find_certified_zeros(dop_4,ini_4,[0,21])

Out[149]: [[[0.499626148223877, 0.504876148223877],
             [14.329523456120423,14.3421569024250961],
             [16.112990881048431,16.1147260235699233],
             [20.432932134920335,20.4799323052715472]],
             []]

In [139]: plot_function(dop_5,ini_5,"f_5",'purple')
```




```
In [150]: find_certified_zeros(dop_5,ini_5,[0,21])
```

```
Out[150]: [[[0.099623703, 0.10146659]], []]
```