python-jupyter-tp1-enonce

May 2, 2017

1 jupyter - activité 3 : usage de `sympy`

Sauvegardez et fermez le calepin précédent et créez en un nouveau, donnez lui un nom.

On commence par importer une bibliothèque de calcul symbolique. La cellule suivante permettra que les dessins soient insérés dans le calepin et non pas créés dans une nouvelle fenêtre.

```
In [ ]: from sympy import *
```

```
In [ ]: %matplotlib inline
```

```
In [ ]: # En décommentant `init_printing()` on affiche les résultats en jsmath :
        # c'est certes plus joli mais c'est moins compréhensible...
        # init_printing()
```

Nous créons deux variables symboliques et en profitons pour réviser une identité remarquable :

```
In [ ]: a,b=Symbol('a'),Symbol('b')
        print((a+b)**2)
        print(expand((a+b)**2))
```

On peut développer par `expand`... et factoriser par `factor` :

```
In [ ]: factor(a**3-8)
```

```
In [ ]: factor(a**2-3)
```

```
In [ ]: factor(a**2-3,extension=sqrt(3))
```

mais ne rêvons pas : on ne sait que rarement factoriser des expressions, et `sympy` ne sait pas plus qu'un humain factoriser les expressions un peu trop complexes...

1.0.1 quelques exercices de la feuille de TD1 : polynômes

- Exo 1 : utiliser `simplify`
- Exo 2 : `pquo`, `prem` pour polynomial quotient, polynomial remainder
- Exo 3 : `factor`

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Pour le dernier calcul, remarquons que le discriminant de $Y^2 - Y + 1$ est $(-1)^2 - 4 \times 1 \times 1 = -3 \ldots$ Voici comment tracer une courbe de fonction :

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]: x=Symbol('x')
        plot(sin(x))
```

```
In [ ]: # exo 1 de la feuille de TD 2
        f = lambda x : x**2+1*x+4
        for i in range(3) :
            print('f(%s) = %s'%(i,f(i)))
        plot(f(x),(x,-2,4))
```

Dans l'exo 2 de la feuille 2, on calcule un polynôme de Newton :

$$n = x \mapsto 4 + 2x + x\,(x-1) + \frac{-10}{6}x\,(x-1)\,(x-2)$$

Quelle est sa forme développée ?

```
In [ ]: n = lambda x : 4 + 2*x + x*(x-1) + -10/6*x*(x-1)*(x-2)
        # n = lambda x : 4 + 2*x + x*(x-1) + Rational(-10,6)*x*(x-1)*(x-2)
        x=Symbol('x')
        print(n(x))
```

C'est mieux avec un rationnel - et non un flottant :

```
In [ ]: n = lambda x : 4 + 2*x + x*(x-1) + Rational(-10,6)*x*(x-1)*(x-2)
        print(n(x))
```

Voici enfin cette forme développée :

```
In [ ]: print(expand(n(x)))
```

Rappelons que cette fonction est connue par ses valeurs en 0, 1, 2 et 3 :

```
In [ ]: for i in range(4):
            print('n(%s) = %s'%(i,n(i)))
```

1.1 polynômes de Lagrange

Nous allons construire une fonction qui renverra un **polynôme de Lagrange**.

Rappelons comment un tel polynôme est construit : * les paramètres sont deux suites de $n+1$ valeurs $x_i : 0 \leq i \leq n$ et $y_i : 0 \leq i \leq n$, les abscisses x_i devant être deux à deux distinctes ; * notre fonction (informatique) renvoie la fonction (mathématique) p qui est le seul polynôme de degré au plus n qui vérifie pour chque i l'équation $p(x_i) = y_i$, ce polynôme étant obtenu par la formule de calcul : * les L_i forment la base de Lagrange :

$$L_i = x \mapsto \prod_{j \in \{0,1...n\}, j \neq i} (x - x_i)$$

* le polynôme interpolateur est obtenu par la combinaison linéaire

$$p = x \mapsto \sum_{i \in \{0,1...n\}} \frac{y_i}{L_i(x_i)} \times L_i(x)$$

La signature de notre fonction pourra être

```
def LagrangePoly(xi,yi) :
""" En entrée deux listes de n+1 nombres
Renvoie de poly de Lagrange associé à ces deux listes
"""
```

Il est prudent de disposer d'un jeu d'essai, par exemple en reprenant l'**exercice 2** de la **feuille de TD 2** :

On construitra d'abord la base de Lagrange : * construire la liste des $n+1$ monomes $x - x_j$; * construire la liste des n monomes $x - x_j$, sauf le numéro i ; * construire le produit de ces monômes - qui est noté $L_i(x)$ dans notre cours.

On conclura en construisant alors le polynôme de Taylor.

In [ ]:

```
In [ ]: dessin=plot(pl1(x),pl2(x),(x,-1,4),ylim=(0,25))
```

Reprendre les calcules de la feuille 2 sur les polynômes de Lagrange.

In [ ]:

2 polynômes de Newton

```
In [ ]: ##exo3 coureur à pied
        A=Matrix(3,3,[1,1,1,4,2,1,9,3,1])
        B=Matrix(3,1,[60,130,250])
        print(A.inv()*B)
```

```
In [ ]: ##exo 3 coureur à pied
        pn1=NewtonPoly(500,500,[60,130,250])
        print(pn1(x))
        print(expand(pn1(x)))
```

```
print(pn1(2000))
######
pn2=NewtonPoly(500,500,[60,130,250,350])
print(pn2(x))
print(expand(pn2(x)))
print(pn2(2000))
######
plot(pn1(x),(x,-500,2500),ylim=(-200,1000))
```

In [ ]:

```
##exo 3 coureur à pied avec changement d'unité
pn1=NewtonPoly(1,1,[60,130,250])
print(pn1(x))
print(expand(pn1(x)))
print(pn1(4))
######
pn2=NewtonPoly(1,1,[60,130,250,350])
print(pn2(x))
print(expand(pn2(x)))
print(pn2(4))
############
pn3=NewtonPoly(0,1,[0,60,130,250,350])
print(pn3(x))
print(expand(pn3(x)))
############
plot(pn1(x),pn2(x),pn3(x),(x,-0.5,4.5),ylim=(-50,500))
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: