Portier Adrien

LMAT1151 - Calcul numérique : méthodes et outils logiciels
TP3 : calcul symbolique

```python
import sympy as sympy
from sympy.abc import*
```

1.

```python
def VérificationLebesgue():
    casA = (a**2 + b**2 + c**2 + d**2)**2 - (a**2 + b**2 - c**2 - d**2)**2
    casB = (2*a*c + 2*b*d )**2 + (2*a*d - 2*b*c)**2
    return (sympy.expand(casA-casB) == 0)
```

```python
VérificationLebesgue()
```

```
True
```

> Ici, j'ai dû faire attention à ne pas écrire les "**" comme des "^" car j'étais un peu confus à mettre des exposants sur des lettres, car avant la découverte de Sympy, le seul logiciel où je devais faire ça devait être sur LaTeX et non Python.

2.

```python
def ExpansionTrigonométrique(expression):
    return expression.expand(trig = True)
```

```python
ExpansionTrigonométrique(sympy.tan(5*x))
```

$$\frac{\tan^{5}(x)}{5\tan^{4}(x) - 10\tan^{2}(x) + 1} - \frac{10\tan^{3}(x)}{5\tan^{4}(x) - 10\tan^{2}(x) + 1} + \frac{5\tan(x)}{5\tan^{4}(x) - 10\tan^{2}(x) + 1}$$

Supplémentaire 2.

> On peut vérifier si cette expansion trigonométrique de $\tan(5x)$ en terme de $\tan(x)$ est vérifiée pour Sympy :

```python
def VerifTan():
    Expans = ExpansionTrigonométrique( sympy.tan(5*x) )
    Tan = sympy.tan(5*x)
    return sympy.expand( Expans - Tan , trig = True ) == 0
VerifTan()
```

```
True
```

> Test de l'invariance de l'expansion trigonométrique : On vérifie que la formule de $\tan(5x)$ en terme de $\tan(x)$ équivaut à la formule de $\sin(5x)$ en terme de $\sin(x)$ divisé par la formule de $\cos(5x)$ en terme de $\cos(x)$

```python
def VerifSinCosTan():
    ExpSin = ExpansionTrigonométrique(sympy.sin(5*x))
    ExpCos = ExpansionTrigonométrique(sympy.cos(5*x))
    ExpTan = ExpansionTrigonométrique(sympy.tan(5*x))
    return sympy.simplify( ( ExpSin / ExpCos ) - ExpTan , trig = True) == 0
VerifSinCosTan()
```

```
True
```

> Je ne pense pas qu'il s'agit d'une manière très optimisée de comparer les expressions mais au moins, on comprend bien ce qu'il se passe dans toutes les opérations, je ne pense pas que le calcul symbolique soit une méthode calcul où on s'attend à une complexité faible. Étant donné que Python gère mieux les calculs au sein de fonctions, on procédera de cette manière pour la suite.

3.

```python
def MachinTan():
    a = sympy.Integer(3)/sympy.Integer(2)
    b = sympy.tan( 5*sympy.atan( sympy.Integer(1) / sympy.Integer(5) )
    - sympy.atan( sympy.Integer(1)/sympy.Integer(239) )).evalf()
    #Pour avoir une expression "numérique" en format sympy

    return b.equals(a)
MachinTan()
```

```
True
```

> J'ai dû chercher la commande "evalf" car je n'arrivais pas à comprendre pourquoi la valeur b dans ma fonction ne se transformait pas directement en réel, car je percevais la valeur comme un nombre déjà calculé.

4.

```python
def TaylorTan(ordre):
    return sympy.series(sympy.tan(x), x, 0, ordre)

TaylorTan(10)
```

$$x + \frac{x^{3}}{3} + \frac{2x^{5}}{15} + \frac{17x^{7}}{315} + \frac{62x^{9}}{2835} + O\left(x^{10}\right)$$

Supplémentaire 4.

> Ici, on peut vérifier que le polynôme de Taylor d'ordre 9 de la fonction $\tan$ n'admet pas de différence absolue supérieure à 1 avec les successives évalutations de la fonction $\tan$ entre $-\frac{\pi}{4}$ et $\frac{\pi}{4}$. Bien évidemment, le paquet Numpy ne prend pas la vraie fonction $\tan$ en compte, mais certainement un polynôme de Taylor d'ordre suffisamment élevé pour qu'on puisse faire confiance à ces valeurs.

```python
import numpy as np
```

```python
def VerifTaylorTan():
    SerieTaylor = TaylorTan(10).removeO()
    for k in np.arange(-np.pi/4,np.pi/4,0.01):
    #Qui revient à prendre les valeurs partant de -np.pi/4 et à incrémenter de 0.01 jusqu'à np.
        if np.abs(SerieTaylor.subs(x,k) - np.tan(k)) > 1:
            return False
    return True
VerifTaylorTan()
```

```
True
```

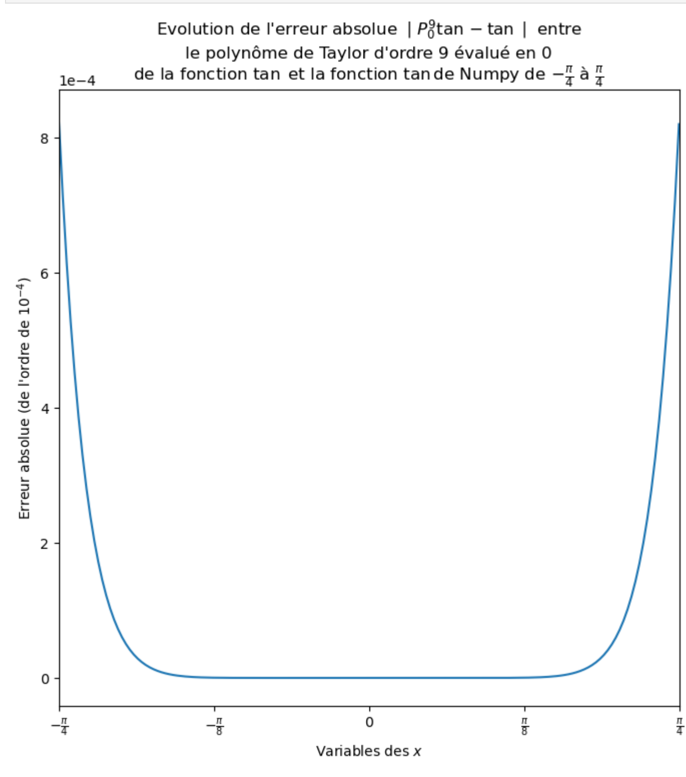
> Représentons graphiquement l'erreur absolue entre $P_0^9 \tan$ qui est le polynôme de Taylor d'ordre 9 évalué en 0 de la fonction $\tan$ et la fonction $\tan$ de Numpy de $-\frac{\pi}{4}$ à $\frac{\pi}{4}$.

```python
import matplotlib.pyplot as plt
```

```python
def ErreurTan():
    fig, axes = plt.subplots(figsize = (8,8))
    axes.set_xlim(-np.pi/4,np.pi/4)
    axes.set_xticks(np.arange(-np.pi/4,np.pi/4+np.pi/8,np.pi/8),
    labels = [r"$-\frac{\pi}{4}$", r"$-\frac{\pi}{8}$"
    , r"$0$",r"$\frac{\pi}{8}$", r"$\frac{\pi}{4}$"] )

    axes.set_xlabel(r"Variables des $x$")
    axes.set_ylabel(r"Erreur absolue (de l'ordre de $10^{-4}$)")
    axes.set_title(r"Evolution de l'erreur absolue $\mid P^{9}_0 \tan - \tan \mid$ entre"
    + "\n"+  r"le polynôme de Taylor d'ordre 9 évalué en 0" +
    "\n"+ r"de la fonction $\tan$ et la fonction $\tan$"+
    r"de Numpy de $-\frac{\pi}{4}$ à $\frac{\pi}{4}$")
    SerieTaylor = TaylorTan(10).removeO()
    xPi = np.arange(-np.pi/4,np.pi/4,0.01)
    xTan = np.tan(xPi)
    xTaylor = [SerieTaylor.subs("x",k) for k in xPi]
    Erreur = np.abs(xTan-xTaylor)
    axes.ticklabel_format(style = "sci", scilimits = (0,0), axis = "y")
    axes.plot(xPi,Erreur)

ErreurTan()
```

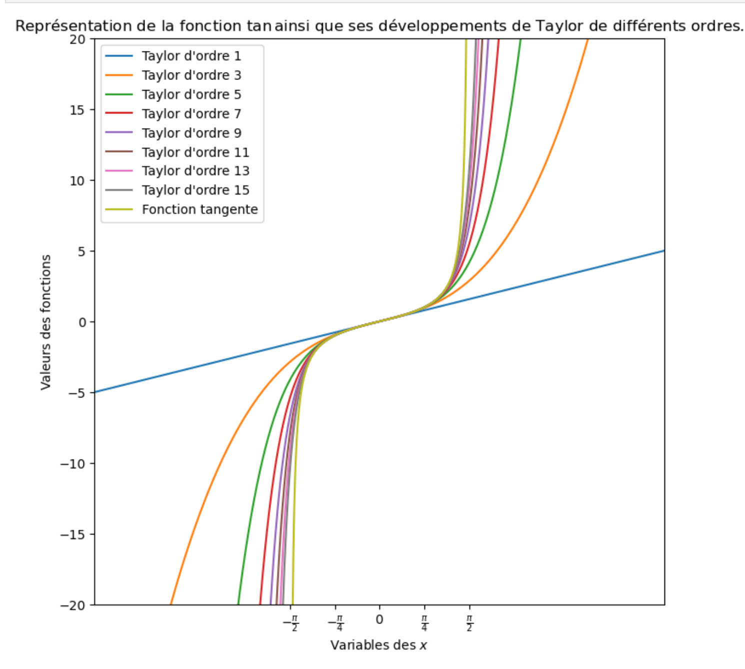


> Et une autre curiosité, regardons l'évolution graphique entre la fonction $\tan$ et ses développements de Taylor :

```python
def GraphTaylorTan():
    fig, axes = plt.subplots(figsize = (8,8))
    axes.set_xlim(-5,5)
    axes.set_ylim(-20,20)
    axes.set_xticks(np.arange(-np.pi/2,np.pi/2+np.pi/4,np.pi/4),
    labels = [r"$-\frac{\pi}{2}$", r"$-\frac{\pi}{4}$",
    r"$0$",r"$\frac{\pi}{4}$", r"$\frac{\pi}{2}$"] )
    axes.set_xlabel(r"Variables des $x$")
    axes.set_ylabel(r"Valeurs des fonctions")
    axes.set_title(r"Représentation de la fonction $\tan$"+
    "ainsi que ses développements de Taylor de différents ordres.")

    xPi = np.arange(-np.pi/2+10**(-6),np.pi/2-10**(-6),0.01)
    xTaylor = np.arange(-5,5,0.01)
    for k in np.arange(1,17,2):
        SerieTaylor = TaylorTan(k+1).removeO()
        yTaylor = [SerieTaylor.subs("x",k) for k in xTaylor]
        axes.plot(xTaylor,yTaylor, label = "Taylor d'ordre {}".format(k))
    axes.plot(xPi,np.tan(xPi), label = "Fonction tangente")
    axes.legend()

GraphTaylorTan()
```



5.

```python
def FactorisationRationnelle(puissance):
    return sympy.factor(x**(puissance) - 1, fraction = False)
FactorisationRationnelle(12)
```

$$(x-1)(x+1)\left(x^{2}+1\right)\left(x^{2}-x+1\right)\left(x^{2}+x+1\right)\left(x^{4}-x^{2}+1\right)$$

Supplémentaire 5.

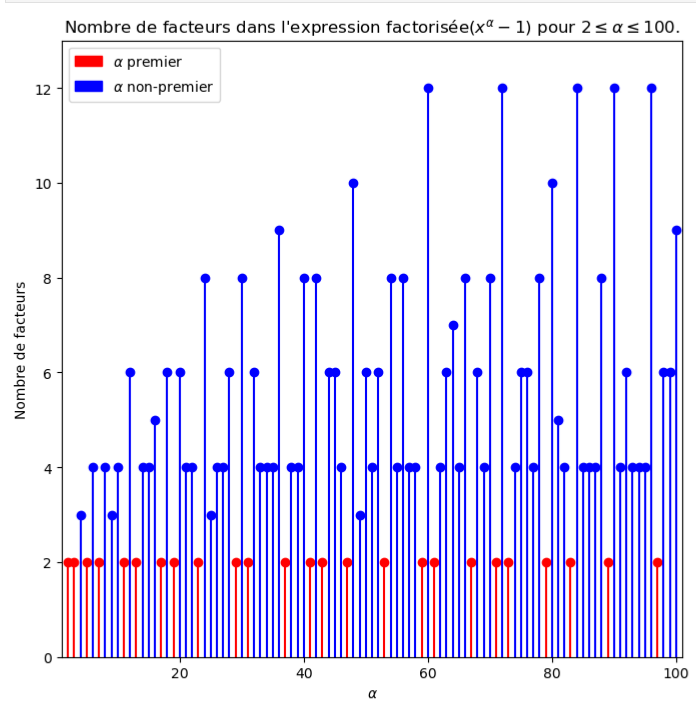
> Il serait intéressant de voir s'il y a un lien entre la puissance du x et le nombre de facteurs dans cette factorisation rationnelle. Je pense qu'il serait difficile de l'estimer mathématiquement et que la puissance de l'ordinateur pourrait être utilisée. On va donc estimer l'évolution du nombre de facteurs dans une expression de type $(x^{\alpha}-1)$ factorisée pour $2 \leq \alpha \leq 100$.

```python
import matplotlib.patches as mpatches
```

```python
def EvolutionFacteursFactorisation():
    fig, axes = plt.subplots(figsize = (8,8))
    Puissance = np.array(range(2,101))
    axes.set_xlim(1,101)
    axes.set_ylim(0,13)
    axes.set_title(r"Nombre de facteurs dans l'expression factorisée"
    + r"$(x^{\alpha}-1)$ pour $2\leq \alpha \leq 100$.")
    axes.set_xlabel(r"$\alpha$", loc = "center")
    axes.set_ylabel("Nombre de facteurs", loc = "center")
    for k in range(len(Puissance)):
        NbFacteurs = len(sympy.factor_list(FactorisationRationnelle(Puissance[k]))[1])
        if sympy.isprime(Puissance[k]):
            axes.stem(Puissance[k], NbFacteurs, "red")
        else:
            axes.stem(Puissance[k], NbFacteurs,"blue", label = "Nombre non-premier")
    rouge = mpatches.Patch(color = 'red', label = r"$\alpha$ premier")
    bleu = mpatches.Patch(color = 'blue', label = r"$\alpha$ non-premier")
    axes.legend(handles=[rouge,bleu])



EvolutionFacteursFactorisation()
```



> J'aurais bien aimé essayer de trouver d'autres liens, mais ils étaient soit trop compliqués à représenter graphiquement, soit pas évidents.

6.

```python
def DecompositionEnFractionSimple(f):
    return sympy.apart(f)

DecompositionEnFractionSimple( x/(x**12-1) )
```

$$\frac{x\left(x^{2}-2\right)}{6\left(x^{4}-x^{2}+1\right)} - \frac{x}{6\left(x^{2}+1\right)} - \frac{x-1}{12\left(x^{2}+x+1\right)} - \frac{x+1}{12\left(x^{2}-x+1\right)} + \frac{1}{12(x+1)} + \frac{1}{12(x-1)}$$

> La décomposition semble plausible étant donné la factorisation obtenue au point (5.)

Supplémentaire 6.

> Décomposition pour un exposant non-naturel

```python
DecompositionEnFractionSimple( 1/(x**(sympy.Integer(13)/2) - 1))
```

$$-\frac{x^{\frac{11}{2}} + 3x^{\frac{9}{2}} + 5x^{\frac{7}{2}} + 7x^{\frac{5}{2}} + 9x^{\frac{3}{2}} + 11\sqrt{x} + 2x^{5} + 4x^{4} + 6x^{3} + 8x^{2} + 10x + 12}{13\left(x^{\frac{11}{2}} + x^{\frac{9}{2}} + x^{\frac{7}{2}} + x^{\frac{5}{2}} + x^{\frac{3}{2}} + \sqrt{x} + x^{6} + x^{5} + x^{4} + x^{3} + x^{2} + x + 1\right)} + \frac{1}{13\left(\sqrt{x}-1\right)}$$

> Décomposition pour un exposant premier

```python
DecompositionEnFractionSimple( 1/(x**7-1) )
```

$$-\frac{x^{5} + 2x^{4} + 3x^{3} + 4x^{2} + 5x + 6}{7\left(x^{6} + x^{5} + x^{4} + x^{3} + x^{2} + x + 1\right)} + \frac{1}{7(x-1)}$$

7.

```python
def Integration(f,a,b):
    return sympy.integrate(f, (x,a,b))
Integration(-x / (x**12-1), 0, sympy.Integer(1)/2)
```

$$-\frac{\sqrt{3}\operatorname{atan}\left(\frac{\sqrt{3}}{6}\right)}{12} - \frac{\log\left(\frac{13}{16}\right)}{24} + \frac{\log\left(\frac{21}{16}\right)}{24} + \frac{\log\left(\frac{5}{4}\right)}{12} - \frac{\log\left(\frac{3}{4}\right)}{12} + \frac{\sqrt{3}\operatorname{atan}\left(\frac{\sqrt{3}}{2}\right)}{12}$$

> Et on peut vérifier que la valeur est bien proche que celle indiquée dans l'énoncé.

```python
Integration(-x / (x**12-1), 0, sympy.Integer(1)/2).evalf()
```

$$0.125004360227235$$

> $0.125004360227235 \approx 0.125004$

Supplémentaire 7.

> Calculer $\int_{-\infty}^{\infty} e^{-x^2}\,\mathrm{d}x$

```python
Integration( sympy.exp(-x**2), -np.inf, np.inf)
```

$$\sqrt{\pi}$$

> qui est une valeur connue.

> En revanche, calculer $\int_0^{\pi} \frac{\sin(x^2)}{x^2+1}\,\mathrm{d}x$ semble compliqué.

```python
Integration( ( sympy.sin(x**2) )/ (x**2+1) , 0, sympy.pi)
```

$$\int\limits_{0}^{\pi} \frac{\sin\left(x^{2}\right)}{x^{2}+1}\, dx$$

> ne donne pas de valeur facilement calculable, mais on peut avoir une estimation :

```python
Integration( ( sympy.sin(x**2) )/ (x**2+1) , 0, sympy.pi).doit()
```

$$\int\limits_{0}^{\pi} \frac{\sin\left(x^{2}\right)}{x^{2}+1}\, dx$$

```python
Integration( ( sympy.sin(x**2) )/ (x**2+1) , 0, sympy.pi).evalf()
```

$$0.375443417414074$$

> Qui semble être une valeur plausible, au vu de la représentation graphique :

```python
def graph():
    fig, axes = plt.subplots(figsize = (8,8))
    axes.set_title(r"Représentation de la fonction" +
    r" qui associe à $x \in \mathbb{R}$, $\frac{\sin(x^2)}{x^2 + 1}$, entre $0$ et $\pi$.")
    axes.set_xlim(0,np.pi)
    axes.set_xlabel(r"$x$")
    axes.set_ylabel("Valeurs de la fonction")
    xcas = np.linspace(0,np.pi,1000)
    xsqr = np.square(xcas)
    fx = np.sin(xsqr)/(xsqr+1)
    axes.fill_between(xcas, 0, fx, where=(fx) >= 0 , color='green', alpha = 0.75)
    axes.fill_between(xcas, 0, fx, where=(fx) < 0 , color='red', alpha = 0.75)
    axes.plot(xcas,fx, label = r"$\frac{\sin(x^2)}{x^2 + 1}$")
    axes.plot(xcas,np.zeros(1000),c="black")
    plt.show()




graph()
```