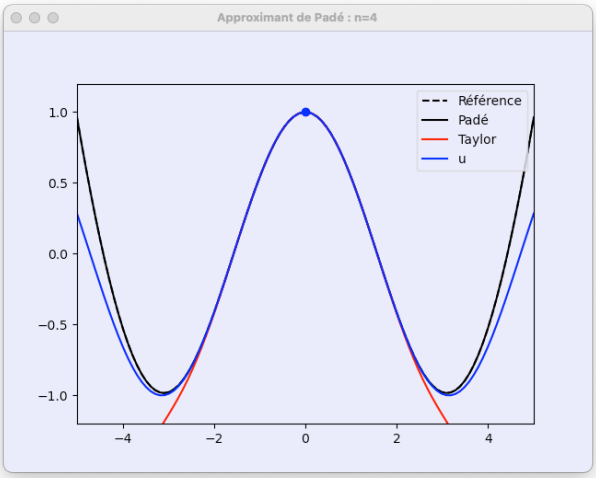
Python 22-23 for dummies : problème 2
Approximants de Padé !

Nous allons maintenant considérer un approximant de Padé $u^h(x)$ d'une fonction quelconque $u(x)$ dont on connaît la valeur U_0 et les $2n$ dérivées successives U_0', U_0'', U_0''' ... à l'origine $x = 0$.



L'approximant de Padé est toujours un quotient de deux polynômes de degré n avec $2n + 1$ coefficients inconnus a_k. A titre d'exemple, notre interpolation pour $n = 2$ s'écrira sous la forme :

$$u(x) \approx u^h(x) = \frac{a_0 + a_1 x + a_2 x^2}{1 + a_3 x + a_4 x^2}$$

Mais, maintenant on choisit les coefficients a_k afin que ce quotient fournisse une approximation de $u(x)$, avec une précision semblable au développement $u^t(x)$ de Taylor d'ordre $2n$ à l'origine. Plus précisément, on exigera que la différence en x entre les deux fonctions soit du même ordre que la précision du développement de Taylor, c'est-à-dire : $\mathcal{O}(x^{2n+1})$.

A nouveau, nous allons considérer le cas $n = 2$ à titre purement illustratif, même si il faudra pouvoir résoudre le problème pour n quelconque. Comme notre approximant est caractérisé par cinq paramètres a_i, il est donc possible de pouvoir approximer le développement de série de Taylor à l'origine d'ordre cinq :

$$u^t(x) = U_0 + U_0'\, x + U_0''\, \frac{x^2}{2} + U_0'''\, \frac{x^3}{6} + U_0''''\, \frac{x^4}{4}$$

On choisit les paramètres afin d'identifier les six premiers termes du polynôme de Taylor à l'origine (appelé aussi dans ce cas polynôme de MacLaurin :-)

$$u^h(x) = u^t(x) + \mathcal{O}(x^5)$$

$$\downarrow$$

$$\frac{a_0 + a_1 x + a_2 x^2}{1 + a_3 x + a_4 x^2} = U_0 + U_0' x + U_0'' \frac{x^2}{2} + U_0''' \frac{x^3}{6} + U_0'''' \frac{x^4}{24} + \mathcal{O}(x^5)$$

$$\downarrow \quad \text{En espérant que le dénominateur ne vaille pas zéro :-)}$$

$$a_0 + a_1 x + a_2 x^2 = \left(1 + a_3 x + a_4 x^2\right)\left(U_0 + U_0' x + U_0'' \frac{x^2}{2} + U_0''' \frac{x^3}{6} + U_0'''' \frac{x^4}{24} + \mathcal{O}(x^5)\right)$$

$$
\begin{aligned}
a_0 + a_1 x + a_2 x^2 &= U_0 + \left(U_0' + a_3 U_0 \right) x \\
&\quad + \left(\tfrac{1}{2} U_0'' + a_3 U_0' + a_4 U_0 \right) x^2 \\
&\quad + \left(\tfrac{1}{6} U_0''' + a_3 \tfrac{1}{2} U_0'' + a_4 U_0' \right) x^3 \\
&\quad + \left(\tfrac{1}{24} U_0'''' + a_3 \tfrac{1}{6} U_0''' + a_4 \tfrac{1}{2} U_0'' \right) x^4 + \mathcal{O}(x^5)
\end{aligned}
$$

Dans ce devoir, il s'agira de calculer les coefficients a_k afin que $u(X_k) = U_k$ pour un nombre n quelconque et une fonction $u(x)$ quelconque. Ensuite, il s'agira d'évaluer l'interpolation de Padé pour un vecteur d'abscisses x_i.

Il faut donc finalement juste résoudre le système suivant !

$$
\begin{bmatrix}
1 & & & & \\
 & 1 & & -U_0 & \\
 & & 1 & -U_0' & -U_0 \\
 & & & -U_0''/2 & -U_0' \\
 & & & -U_0'''/6 & -U_0''/2
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}
=
\begin{bmatrix} U_0 \\ U_0' \\ U_0''/2 \\ U_0'''/6 \\ U_0''''/24 \end{bmatrix}.
$$

L'objectif de ce second devoir est d'appréhender l'utilisation de `numpy`, mais aussi d'être le plus rapide possible : il s'agit donc de construire la matrice en essayant d'obtenir l'implémentation la plus rapide surtout si n est grand !

Approximant de Padé et fonctions continues

Il est possible d'écrire un approximant de Padé sous la forme d'une fonction continue. En d'autres mots, un approximant de Padé, c'est une approximation sous la forme d'une fonction continue....

$$
u^h(x) = \frac{a_0 + a_1 x + a_2 x^2 + a_3 x^3}{1 + a_4 x + a_5 x^2 + a_6 x^6} = c_0 + \cfrac{c_1}{c_2 + x + \cfrac{c_3}{c_4 + x + \cfrac{c_5}{c_6 + x}}}
$$

Et dans l'exemple du cosinus, cela revient à écrire l'approximant de Padé sous la forme :

$$
u^h(x) = \frac{15120 - 6900\ x^2 + 313\ x^4}{15120 - 660\ x^2 + 13\ x^4} = c_0 + \cfrac{c_1}{c_2 + x^2 + \cfrac{c_3}{c_4 + x^2}}
$$

Cette manière d'écrire permet de réduire le nombre d'opérations à effectuer pour évaluer l'approximant de Padé et cela permet de conclure que calculer l'approximant de Padé est plus efficace que d'utiliser le développement en série de Taylor à l'origine qu'on a utilisé pour en déduire ce fameux approximant de Padé. En d'autres mots, Padé est plus efficace que Taylor pour estimer le cosinus. Aujourd'hui, les approximants de Padé sont utilisés dans de nombreux domaines où on doit calculer des valeurs approchées de fonctions dans les calculatrices et les ordinateurs...

Plus précisément, on vous demande de :

1. Ecrire une fonction

   ```
   a = padeApproximationCompute(n,dU)
   ```

   qui calcule les coefficients de l'approximant de Padé composé d'un quotient de deux polynôme de degré n à partir de la valeur à l'origine et des dérivées successives d'une fonction à l'origine fournie dans le tableau unidimensionnel de taille $(2n+1)$.

   L'argument `dU` sont des tableaux unidimensionnels `numpy` de dimension $2n+1$. On peut supposer que le tableau a toujours une dimension égale à $2n+1$ avec n qui est le premier argument de la fonction. Il n'est ni demandé, ni utile de tester que le tableau fourni est conforme aux spécifications de la fonction.

2. Ensuite écrire une fonction

   ```
   uh = padeEval(a,x)
   ```

   qui évalue l'interpolation de Padé caractérisée par les coefficients a_k pour des abscisses quelconques x_i.

3. Comme on est vraiment super gentil, vous avez le droit d'utiliser la fonction `solve` de `numpy.linalg`, qui résout un système linéaire $Ax = b$. Tout autre import que `numpy` ou `nump.linalg` est par contre rigoureusement interdit et sera retiré à l'exécution de votre programme. Vous avez aussi droit d'utiliser la fonction `factorial` de la librairie mathématique, même si ce n'est pas la manière la plus efficace de procéder.

4. Pour tester votre programme, on vous a fourni un tout petit programme simple `padeApproximationTest.py` qui devrait vous permettre d'écrire et de tester votre code avec votre ordinateur

```
import numpy as np
from sympy import *

  x = symbols('x')
  u = cos(x)
  n = 8
  X  = 1.5
  U = u.subs(x,X)


  [ut,Ut,dU]  = macLaurinCompute(u,x,n,X)

  print("  MacLaurin expansion of %s : %s" % (u,str(ut)))
  print(" ======== Derivatives of %s for x = 0 : " % u, end=''); print(dU)
  print(" ======== Value of %s for x = %4.2f : %.16f" % (u,X,U))
  print(" ======== MacLaurin expansion of order %d  for x = %4.2f : %.16f" % (n,X,Ut))
  print("  Approximation error : %14.7e" % (U-Ut))

  #
  # -2- Calcul de l'approximation de Pade
  #

  n = 4
  a = padeApproximationCompute(n,dU)
  Up = padeEval(a,X)

  print(" ======== Coefficients of Pade approximation of order %d for %s : " % (n,u))
  print("  ",a[:n+1]);
  print("  ",a[n+1:]);
  print(" ======== Pade approximation for x = %4.2f : %.16f" % (X,Up))
  print("  Approximation error : %14.7e" % (U-Up))


  from sympy.utilities.lambdify import lambdify
  from matplotlib import pyplot as plt
```

```
u  = lambdify(x,u,'numpy')
ut = lambdify(x,ut,'numpy')

m = 100;
x = np.linspace(-5,5,m)

#
#   Solution de référence :-)
#

uReference = (15120 - 6900*x*x + 313*x**4)/(15120 +660*x**2 + 13*x**4)

plt.figure("Approximant de Padé : n=%d" % n)
plt.plot(x,uReference,'--k',label='Référence')
plt.plot(x,padeEval(a,x),'-k',label='Padé')
plt.plot(x,ut(x),'-r',label='Taylor')
plt.plot(x,u(x),'-b',label='u')
plt.xlim((-5,5)); plt.ylim((-1.2,1.2))
plt.plot([0],[1],'ob')

plt.legend(loc='upper right')
plt.show()
```

5. Pour les étudiants curieux, observer comment on calcule l'expression symbolique du développement en série de MacLaurin d'une fonction quelconque dans le script de test fourni !

6. Comme nous sommes vraiment trop gentils, on vous a même fourni l'expression analytique de l'approximant de Padé pour le cosinus : c'est donc inacceptable de ne pas avoir la bonne solution. Attention : le code sera évidemment testé pour une autre fonction que le cosinus et une autre valeur de n que quatre : pas si bête quand-même !

7. Votre fonction (avec les éventuelles sous-fonctions que vous auriez créées) sera soumise via le site web du cours.

8. Attention : il ne faut pas recopier le programme de test `main` dans votre soumission : uniquement les deux fonctions que vous avez écrites. Vérifier bien que votre programme fonctionne correctement sur le serveur et pas uniquement sur votre ordinateur : aucun recours ne sera valable si le devoir n'est pas exécuté correctement sur le serveur.