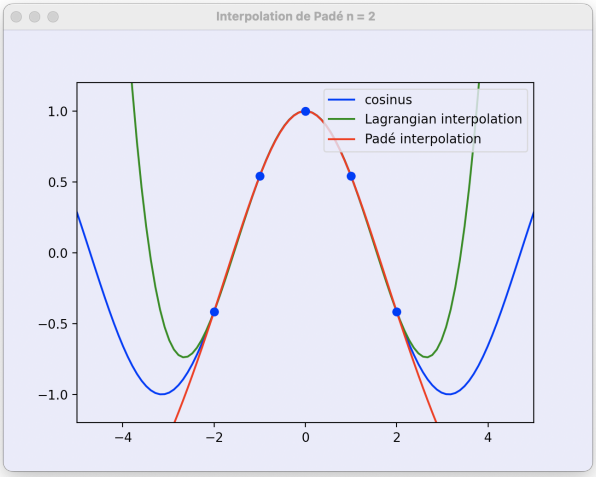
## Python 22-23 for dummies : problème 1
## Interpolation de Padé !

Nous allons construite une interpolation de Padé $u^h(x)$ d'une fonction quelconque $u(x)$ dont on connaît $2n+1$ valeurs $U_k$ pour les $2n+1$ abscisses.




L'interpolation de Padé est définie comme 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 :

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

Sur la figure, on observe l'interpolation polynomiale et l'interpolation de Padé de la fonction $u(x) = cos(x)$ en choisissant cinq points d'interpolation distincts. Dans un second devoir, nous allons découvrir que les approximants de Padé permettent d'évaluer la fonction cosinus à partir d'un quotient de polynômes et que cette approche est plus efficace que l'usage d'un développement en série de Taylor.

Dans ce premier devoir, il s'agira tout d'abord 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 quelconques $x_i$.

Plus précisément, il s'agira de calculer les $2n+1$ paramètres $a_i$ afin de faire passer la fonction par $2n+1$ points $(X_i, u(X_i))$. Pour illustrer notre propos, considérons le cas particulier pour $n = 2$ avec cinq points d'interpolation.

$$\underbrace{U_i}_{u(X_i)} = \underbrace{\frac{a_0 + a_1 X_i + a_2 X_i^2}{1 + a_3 X_i + a_4 X_i^2}}_{u^h(X_i)}$$

En espérant que le dénominateur ne vaille pas zéro :-)

$$\begin{aligned} U_i + a_3 U_i X_i + a_4 U_i X_i^2 &= a_0 + a_1 X_i + a_2 X_i^2 \\ a_0 + a_1 X_i + a_2 X_i^2 - a_3 U_i X_i - a_4 U_i X_i^2 &= U_i \end{aligned}$$

Il faut donc finalement juste résoudre le système suivant !

$$\begin{bmatrix} 1 & X_0 & X_0^2 & -U_0 X_0 & -U_0 X_0^2 \\ 1 & X_1 & X_1^2 & -U_1 X_1 & -U_1 X_1^2 \\ 1 & X_1 & X_2^2 & -U_2 X_2 & -U_2 X_2^2 \\ 1 & X_3 & X_3^2 & -U_3 X_3 & -U_3 X_3^2 \\ 1 & X_4 & X_4^2 & -U_4 X_4 & -U_4 X_4^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}.$$

L'objectif de ce premier devoir est d'appréhender l'utilisation de `numpy`, mais sans rechercher d'être le plus rapide possible : il s'agit donc de construire la matrice, mais on peut le faire avec des boucles sans vectoriser, même si les plus habiles et futés pourront faire une implémentation plus rapide.

Plus précisément, on vous demande de :

1. Ecrire une fonction

   ```
   a = padeInterpolation(X,U)
   ```

   qui calcule les coefficients de l'interpolation de Padé des points $(X_k, U_k)$.

   Les arguments `X` et `U` sont des tableaux unidimensionnels `numpy` de dimension $2n+1$. On peut supposer que les deux tableaux sont toujours de la même dimension et que cette dimension est toujours un nombre impair ! La fonction renverra un tableau `numpy` de même dimension que les arguments fournis : il s'agira donc d'un tableau unidimensionnel de dimension $2n+1$.

2. Ensuite écrire une fonction

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

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

   Les arguments `a` et `x` sont des tableaux unidimensionnels `numpy` de dimension $2n+1$ et $m$ respectivement. La fonction renverra un tableau `numpy` de même dimension que $x$ : il s'agira donc d'un tableau unidimensionnel de dimension $m$. Le second argument peut aussi être un simple réel et alors la fonction doit renvoyer un réel qui correspond à l'ordonnée qui correspond à l'abscisse fournie :-)

3. Attention : même si on présente un exemple avec $n = 2$ dans l'énoncé, il faut écrire un programme pour un nombre $n$ quelconque !
   Et il ne faut pas non plus me fournir une interpolation polynomiale : hein :-)

4. Comme on est vraiment super gentil, vous avez le droit d'utiliser la fonction `solve` de `numpy.linalg`, qui résout un système linéaire $Ax = b$. Par contre, tout autre import que `numpy` ou `nump.linalg` est rigoureusement interdit et sera retiré à l'exécution de votre programme...

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

```
  n = 2
  u =  lambda x : cos(x)
  X = linspace(-2,2,(2*n+1))
  U = u(X)
#
# -1- Calcul des coefficients de l'interpolation
#
  print("==== Computing the Padé approximation :-)")
  a = padeInterpolationCompute(X,U)
  print(" a = ",list(a))
#
# -2- Evaluation l'interpolation de Padé
#     et de l'interpolation polynomiale de Lagrange
#
  x = linspace(-5,5,100)
  upade = padeEval(a,x)
  uh = polyval(polyfit(X,U,len(X)-1),x)
#
# -3- Et un joli plot :-)
#
  from matplotlib import pyplot as plt
  plt.figure('Interpolation de Padé n = %d ' % n)
```

```
plt.plot(x,u(x),'-b',label='cosinus')
plt.plot(x,uh,'-g',label='Lagrangian interpolation')
plt.plot(x,upade,'-r',label='Padé interpolation')
plt.plot(X,U,'ob')
plt.xlim((-5,5)); plt.ylim((-1.2,1.2))
plt.legend(loc='upper right')
plt.show()
```

Au passage, c'est une bonne idée de changer la valeur de $n$ lors de vos tests !

6. Vos deux fonctions (avec les éventuelles sous-fonctions que vous auriez créées) seront soumises via le site web du cours.

7. 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. Pour rappel, toutes vos soumissions seront systématiquement analysées par un logiciel anti-plagiat. Faites vraiment votre programme seul... en vous inspirant uniquement des programmes fournis par l'enseignant :-)
Michel, Ange et Nathan veillent au grain et à la machine pour le moudre !