TELECOM Nancy *(ESIAL)*
Maths Numériques

TP 2 : interpolation polynomiale (Lagrange)

Dans ce TP on se focalise sur l'interpolation polynomiale de n points (x_i, y_i) avec $i = 0, \dots, n-1$ et $x_i \neq x_j$ pour $i \neq j$. On rappelle qu'il existe un et un seul polynôme de degré inférieur ou égal à $n-1$ qui passe par ces points et que ce polynôme s'exprime dans la base de Lagrange par :

$$p(t) = \sum_{i=0}^{n-1} y_i \mathcal{L}_i(t), \text{ où } \mathcal{L}_i(t) = \prod_{\substack{j=0 \\ j\neq i}}^{n-1} \frac{t - x_j}{x_i - x_j} \tag{1}$$

Une évaluation basique de ce polynôme est en $O(n^2)$ mais si on précalcule les quantités :

$$\omega_i = \prod_{\substack{k=0 \\ k\neq i}}^{n-1} (x_i - x_k) \tag{2}$$

pour un coût de $O(n^2)$ (mais qui ne sera effectué qu'une seule fois), alors la formule barycentrique :

$$p(t) = \frac{\displaystyle\sum_{i=0}^{n-1} \frac{y_i}{(t - x_i)\omega_i}}{\displaystyle\sum_{i=0}^{n-1} \frac{1}{(t - x_i)\omega_i}} \tag{3}$$

permet d'obtenir une évaluation en $O(n)$. Vous allez écrire plusieurs fonctions dans un module qui sera dédié à l'interpolation. Récupérer le squelette de ce module (fichier `interp_skeleton.py`) sur Arche (renommer ce fichier `interp.py`). Pour la suite il faudra aussi récupérer le fichier `test1.py`. **Attention :** dans ce module, numpy est importé avec l'instruction `import numpy as np`. Ainsi toute fonction ou variable définie par ce module doit être appelée avec le préfixe `np.` (`np.sin`, `np.linspace`, `np.pi`, `np.arange`, etc.).

Exercice 1 *Evaluation basique*

1. Compléter la fonction `lagrange(t, x, y)` pour calculer la valeur du polynôme d'interpolation des points (x_i, y_i) en t avec l'algorithme naturel basé sur la formule (1) (cf question 1 exercice 2 feuille 4).
2. On aimerait une fonction qui puisse s'appliquer sur un vecteur t plutôt que sur un simple scalaire. Compléter la fonction `lagrange_v(tv, x, y)` qui fait ce travail (il s'agit d'une simple boucle dont le corps est constitué d'un appel à la fonction précédente), c'est à dire retourne un vecteur `pv` où la k ème composante est égal à la valeur du polynôme d'interpolation sur la k ème composante de `tv`.
3. Tester votre fonction avec le script `test1.py` à récupérer sur Arche. Dans celui-ci on interpole la fonction sinus sur $[0, 2\pi]$ ce qui fonctionne très bien avec des abscisses d'interpolation uniformément réparties sur $[0, 2\pi]$. Expérimenter en modifiant la valeur du nombre de points d'interpolation (le script utilise $n = 7$). Par exemple essayer $n = 14$. On peut remarquer que l'erreur maximale est atteinte vers les extrémités de l'intervalle.

4. Ecrire un nouveau script `test2.py` (si vous copiez et collez `test1.py` il n'y a que très peu de changements à effectuer) pour interpoler la fonction de Runge : $f(x) = \frac{1}{1+x^2}$ sur $[-4, 4]$. Cette fois-ci l'interpolation avec abscisses uniformément réparties ne converge plus. Vous pouvez coder la fonction de Runge dans votre module d'interpolation (2 lignes en utilisant la syntaxe vectorielle).

Exercice 2 *Tchebichev à la rescousse*

Ces abscisses permettent d'obtenir un polynôme d'interpolation plus performant (en particulier la convergence doit avoir lieu pour toute fonction absolument continue, cf cours). Pour n abscisses sur $[-1, 1]$, elles sont données par la formule suivante :

$$X_i = \cos\left(\frac{(2i+1)\pi}{2n}\right), \quad i = 0, \ldots, n-1.$$

et pour les obtenir sur $[a, b]$ on utilise la transformation affine :

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} X_i, \quad i = 0, \ldots, n-1.$$

1. Dans `interp.py` rajouter une fonction `tchebychev(a,b,n)` qui retourne le vecteur de ces abscisses de Tchebichev. Aide : cela peut se faire vectoriellement 1/ en utilisant la fonction `arange` de numpy (`arange(n)` retourne le tableau $[0, 1, \ldots, n-1]$), 2/ en appliquant les deux formules précédentes directement sur des tableaux.
2. Expérimenter ces abscisses dans les deux tests précédents : avec la fonction sinus l'erreur doit être plus petite (pour un même nombre n de points d'interpolation) et avec la fonction de Runge, l'interpolation converge bien vers la fonction lorsque $n \to +\infty$.

Exercice 3 *Place à l'optimisation !*

Pour obtenir une assez bonne précision pour la fonction de Runge, il faut par exemple $n = 130$ (l'erreur maximum est de l'ordre de 2.10^{-14} (on peut faire mieux en utilisant la symétrie de la fonction mais là n'est pas notre propos). Prenez aussi $m = 600$ pour obtenir une courbe d'erreur assez lisse. Là on sent la complexité en $O(n^2)$ de l'évaluation basique de Lagrange (il faut une dizaine de secondes ou plus pour le calcul). Il est temps de tester la formule barycentrique de Lagrange en $O(n)$.

1. Dans `interp.py` rajouter une fonction `lagrange_poids_fb(x)` qui calcule les quantités ω_i de la formule (2).
2. Ecrire une fonction `lagrange_fb(t,x,y,w)` qui calcule le polynôme de Lagrange en t avec la formule barycentrique (3). **Attention** la formule barycentrique ne convient pas pour une abscisse $t = x_i$, et il faut auparavant examiner si t est égal à l'une des abscisses d'interpolation x_i (et dans ce cas retourner y_i). Pour cela on peut faire une simple recherche linéaire. Pour éviter d'écrire cette recherche on pourra utiliser le bout de code suivant :

   ```
   ind = np.nonzero(x == t)
   if len(ind[0]) > 0:
      p = y[ind[0][0]]
   else:
      # t est different des x[i] on utilise la formule barycentrique
   ```

   La fonction numpy `nonzero` retourne un t-uple dont le premier objet est un tableau contenant les indices correspondant à des valeurs non nulles (ou vraies dans le cas d'un tableau de booléens comme ici).
3. De même que pour l'évaluation basique écrire une fonction `lagrange_fb_v(tv,x,y,w)` qui utilise la fonction précédente de sorte à évaluer le polynôme sur toutes les abscisses du vecteur `tv` (procéder en copiant-collant `lagrange_v(tv,x,y)`).
4. Copier-coller `test2.py` sur `test2_bis.py` et modifier le de sorte à utiliser la formule barycentrique. Maintenant ça doit aller beaucoup plus vite ! NB : en fait il est possible de faire mieux en utilisant les possibilités vectorielles de numpy de sorte à moins faire travailler l'interprète python.