TD/TP : Interpolation polynomiale

Exercice 1 (Différences divisées : Formule de récurrence.)**.**

Rappel de cours :
Soit $f : [a, b] \to \mathbb{R}$ une fonction quelconque. On se donne des points d'interpolations $a \leq x_0 < x_1 < \ldots < x_n \leq b$ et on note $L_n f$ le polynôme d'interpolation de Lagrange associé (*i.e.* $L_n f(x_j) = f(x_j), j = 0, \ldots, n$). On note, pour $k \geq 1$,

$$\omega_k(X) = (X - x_0)(X - x_1)\ldots(X - x_{k-1}).$$

Le polynôme ω_k est le k^{eme} polynôme nodal aux noeuds $x_0, \ldots, x_n$, et on pose $\omega_0 \equiv 1$. La famille $\{\omega_k, 0 \leq k \leq n\}$ est appelée la *base de Newton.* C'est une base de $\mathbb{P}_n$. En étudiant les racines de la différence $L_k f - L_{k-1} f$, on montre que

$$\forall 1 \leq k \leq n, \exists a_k \in \mathbb{R} : L_k f - L_{k-1} f = a_k \omega_k$$

On note $a_k = f_{[x_0 \ldots f_k]}$ et on appelle ce nombre, k^{eme} *différence divisée aux noeuds* $x_0, \ldots, x_k$. Une récurrence immédiate donne alors :

$$L_n f(x) = \sum_{k=0}^{n} f_{[x_0 \ldots x_k]} \omega_k(x)$$

On peut faire le même raisonnement entre n'importe quels noeuds x_i, x_j ($0 \leq i < j \leq n$) ; on note $f_{[x_i \ldots x_j]}$ la $j - i^{eme}$ différence divisée correspondante. Noter que $f_{[x_i]} = f(x_i)$ [1].

Le but de l'exercice est d'établir le résultat de récurrence suivant sur les différences divisées, et de l'exploiter numériquement pour calculer les différences divisées.

Proposition 1. *Soient f, $a \leq x_0 < x_1 < \ldots < x_n \leq b$ et $L_n f$ comme ci-dessus. Alors, pour tout $i < k \in \{0, \ldots, n\}$, on a*

$$f_{[x_i \ldots x_k]} = \frac{f_{[x_{i+1} \ldots x_k]} - f_{[x_i \ldots x_{k-1}]}}{x_k - x_i}. \tag{1}$$

Sans perdre de généralité, on suppose pour la preuve que $i = 0$ et $k \in \{1, \ldots, n\}$.

1. Quel est le coefficient dominant de $L_k f$ (interpolateur aux noeuds $x_0, \ldots, x_k$) ? (utiliser la décomposition de $L_k f$ dans la base de Newton et la définition de $f_{[x_0 \ldots x_k]}$).
2. On pose
   - $p_{k-1} = L_{x_0, \ldots, x_{k-1}} f$ (le polynôme de Lagrange aux noeuds $x_0, \ldots, x_{k-1}$),
   - $q_{k-1} = L_{x_1, \ldots, x_k} f$ (le polynôme de Lagrange aux noeuds « décalés » $x_1, \ldots, x_k$).
   - $\tilde{p}_k$ le polynôme de « différence divisée »,

   $$\tilde{p}_k(x) = \frac{(x - x_0) q_{k-1}(x) - (x - x_k) p_{k-1}(x)}{x_k - x_0}$$

   Montrer que $\tilde{p}_k = p_k$.
3. Soit a_k le coefficient dominant de p_k, b_k celui de q_k et $\tilde{a}_k$ celui de $\tilde{p}_k$. Montrer que

   $$\tilde{a}_k = \frac{b_{k-1} - a_{k-1}}{x_k - x_0}.$$

   Conclure.

1. d'après la définition du polynôme nodal à 0 noeuds ($\omega_0 \equiv 1$) et le fait que pour un seul noeud $\{x_i\}$, l'interpolation de Lagrange est la constante $L_0 f(x) \equiv f(x_i)$

4. Implémentation : La proposition établie ci-dessus suggère de calculer les différences divisées en calculant successivement, de gauche à droite, les colonnes du tableau triangulaire T suivant :

$$T = \begin{pmatrix} f_{[x_0]} & & & & \\ f_{[x_1]} & f_{[x_0,x_1]} & & & \\ f_{[x_2]} & f_{[x_1,x_2]} & f_{[x_0,x_1,x_2]} & & \\ \vdots & \vdots & \vdots & \ddots & \\ f_{[x_n]} & \cdots & & & f_{[x_0,x_1,\ldots,x_n]} \end{pmatrix}$$

Remarquez que chaque élément $T_{i,j}$ pour $i \geq j$ est une fonction des $(x_k)_{k\leq n}$ et des deux éléments $T_{i,j-1}$ et $T_{i-1,j-1}$.

(a) pour $i \geq j$, déterminer ℓ, k tels que $T_{i,j} = f_{[x_\ell,\ldots,x_k]}$

(b) En déduire l'expression de $T_{i,j}$ en fonction de $T_{i,j-1}$, $T_{i-1,j-1}$ et des $(x_k)_{k\leq n}$.

(c) Ecriture vectorielle : En déduire pour $j \geq 2$, l'expression du vecteur correspondant aux éléments sous la diagonale, $(T_{i,j}, i \in \{j, \ldots, n+1\})$ en fonction de sous-vecteurs de la colonne $T_{\cdot,j-1}$ et de sous-vecteurs du vecteur des noeuds $x_i, i \in \{0, \ldots, n\}$.

(d) Écrire une fonction `dividif` prenant en argument un vecteur $X = (x_0, \ldots, x_n)$ (les noeuds d'interpolation), un vecteur de données Y (les valeurs de f aux noeuds x_i) et qui renvoie le vecteur des différences divisées $d = (f_{[x_0]}, f_{[x_0\, x_1]}, \ldots f_{[x_0 \ldots x_k]})$.
Attention : en `R` la numérotation des vecteurs commence à 1, prendre garde au décalage d'indices concernant le vecteur de noeuds.
On partira du squelette de code suivant :

```
dividif=function(x,y){
    ## Computes the divided differences (coefficients on the Newton basis) for
    ##  Lagrange interpolation.
    ##
    ## @title dividif: Newton's Divided differences
    ## @param x a vector containing the interpolation nodes
    ## @param y a vector of same size as x: values of the interpolated function at
    ##          the nodes
    ## @return a vector of same size as x: the divided differences
    ##        \eqn{f_[x_0, ... x_k]} of order 'length(x) -1'.

    n = length(x) -1 ## n: degree of Lagrange polynomial.
    Tmat = matrix(ncol = n+1, nrow = n+1)
    Tmat[,1] = y ## initialisation of the vector of divided differences:
    if(n ==0) {return(diag(Tmat))}
    for (j in 2:(n+1) ) {
        Tmat[j : (n+1), j ] = ## Complete the code
    }
    return(diag(Tmat))
}
```

5. Testez votre code :

```
nodes = c(0, 1, 2, 3)
om0 = function(x){1}
om1 = function(x){x- nodes[1]}
om2 = function(x){(x- nodes[1])*(x- nodes[2])}
om3 = function(x){(x- nodes[1])*(x- nodes[2])*(x- nodes[3])}
A <- c(0.1,0.2, -0.1 , -0.3)

divtest <- function(x) {
  return( A[1] * om0(x) + A[2]*om1(x) + A[3]*om2(x)+ A[4]*om3(x) )
```

```
}
divA <- dividif(x= nodes, y = sapply(nodes, divtest) )

divA
A
```

Exercice 2 (Méthode de Horner)**.**
Le but de la méthode de Horner est d'évaluer la valeur d'un polynôme dont on connait les coefficients, en un point donné x. Le point de départ est de remarquer que l'on peut factoriser l'écriture habituelle d'un polynôme ainsi :

$$a_0 + a_1 x + a_2 x^2 + \dots a_n x^n = a_0 + x\left(a_1 + x\left(\dots\dots + x\left(a_{n-2} + x\left(a_{n-1} + x a_n\right)\right)\dots\right)\right),$$

Autrement dit, l'algorithme d'évaluation est le suivant :

1. Initialisation : $p = a_n$
2. Pour $k \in \{1 \dots, n\}$:
   - $p \leftarrow a_{n-k} + p * x$
3. renvoyer p.

Sa complexité est de $O(n)$ multiplications et $O(n)$ additions : c'est bien mieux que la méthode naïve consistant à calculer toutes les puissances de x puis à les ajouter ($O(n^2)$).

1. Sur le même principe, écrire un algorithme permettant d'évaluer rapidement un polynôme p dont on connaît les coefficients $(c_0, \dots, c_n)$ sur la base de Newton aux noeuds $x_0, \dots, x_n$ (voir exercice 3), c'est à dire

$$p(x) = \sum_{k=0}^{n} c_k \omega_k(x)$$

Implémenter cet algorithme selon le squelette de code suivant

```
hornerNewton = function(a,x,z){
    ## Horner's method: Evaluates a polynom P at points z, given
    ## nodes x and the coefficients a of P in Newton's basis
    ##
    ## @param a : vector: the coefficients of the polynomial in
    ##          Newton's basis
    ## @param x : the interpolation nodes.
    ## @param z : vector of points where the polynom needs to be
    ##           evaluated.
    ## @return : a vector of same size as z: the value of the
    ##           polynomial at points z.
    ##
    n <- length(x) - 1 ## degree of the Lagrange poynomial
    if( (n < 0) || (length(a) != (n+1)) )
    {
        stop('at least one interpolating point is needed,
             a and x should have same length')
    }
    f <- a[n+1]*rep(1,length(z))
    if(n >= 1){
        for( k in 1:n){
           f <-  ## complete the code
        }
    }
    return(f)
}
```

2. Testez votre évaluateur en utilisant les mêmes objets que ceux définis pour tester la fonction dividif :

```
hornerNewton(a = A, x = nodes[1:3], z = c(0,0.5,2,3))
sapply( c(0,0.5,2,3), divtest)
```

Exercice 3 (Méthode de Lagrange avec différences divisées de Newton)**.** En utilisant les deux exercices précédents (méthode de Horner et récurrence sur les différences divisées, implémenter un interpolateur de Lagrange aux noeuds x, étant donné un vecteur de valeurs y, renvoyant la valeur du polynôme de Lagrange évalué aux points du vecteur z.

```
interpolDividif=function(x,y,z){
    ## Efficient Lagrange interpolation using Horner's method with
    ## Newton basis for evaluation
    ## @param x : vector containing the interpolation nodes
    ## @param y : vector of same size as x: values of the interpolated
    ##           function at the nodes
    ## @param z : vector of points where the interpolating polynomial
    ##           needs to be evaluated.
    ## @return : vector of same size as z: the value of the
    ##           interpolating polynomial at points z.

## Complete the code
}
```

Testez votre interpolateur avec des points équidistants sur $[-1, 1]$
(`x = seq(-1,1,length.out=n)`), sur la fonction suivante, dont vous ferez varier les paramètres. Affichez les résultats (sur une même figure, f et ses interpolées pour différents nombre de points d'interpolation).

```
myfun=function(x){cos(5*x/(2 + x))* 1/(1+(1*x)^2)}
```

Exercice 4 (Phénomène de Runge, points équidistants ou de Tchebychev)**.**
Le contre-exemple de Runge est fournit par la fonction suivante,

```
funRunge=function(x){ 1/(1+(5*x)^2)}
```

lorsqu'on cherche à l'interpoler sur $[-1, 1]$.

1. Tracer sur la même figure cette fonction et ces interpolations pour des points équidistants, avec différentes valeurs du nombre de noeuds.
2. Utiliser maintenant les points de Tchebychev d'ordre n, $(x_0, \ldots, x_n)$ à la place des points équidistants. Comparer avec les résultats obtenus avec des points équi-distants en affichant les deux résultats et la fonctions sur le même graphique.

Exercice 5 (Interpolateur de Lagrange avec choix des noeuds)**.** Le but de cet exercice est d'implémenter un interpolateur de Lagrange générique, utilisant au choix des noeuds équidistants ou de Tchebychev, sur un intervalle quelconque.

1. Complétez la fonction suivante, qui renvoie l'interpolation de Lagrange d'une fonction sur une grille régulière de taille `neval`, avec des noeuds choisis en fonction du paramètres `nodes`. Si `nodes = 'cheby'`, la fonction doit utiliser les noeuds de Tchebychev. Si `nodes = 'equi'`, la fonction doit utiliser des noeuds équidistants. Ni la position des noeuds ni celle des points d'interpolation n'ont besoin d'être spécifiées, seulement leur nombre.

```
  interpolLagrange =function(n, a, b, neval, nodes = 'equi', FUN, Plot){
## Generic Lagrange interpolation, with equidistant or Chebyshev nodes.
## @param n : the degree of the interpolating polynomial on each
## subinterval
## @param a : left end-point of the interval
## @param b : right end-point of the interval
```

```
## @param neval :number of evaluation points (a regular grid will be
## used on [a,b]
## @param nodes :string, either "equi" (default) for equidistant
## Lagrange interpolation (on each subinterval) or "cheby" for
## using Chebyshev nodes.
## @param FUN: the function to be interpolated
## @param Plot : logical. Setting 'Plot' to TRUE produces a plot
## showing the graph of
## the true functions and its interpolation.
## @return : vector of size 'neval': the values of the Lagrange
## polynomial on an equi-distant grid.

    if (nodes == "equi"){
        x =  ## Complete
    }
    else if (nodes == "cheby"){
        x = ## Complete
    }
    else{stop("the nodes must be either 'equi' or 'cheby'") }

    ##
    ## Complete the code: compute a vector 'f' containing
    ## the interpolated values on an equidistant
    ## evaluation grid 'z'.
    ##
    ##

    if( Plot ){
        if (nodes == "equi"){ methodName = " equidistant "}
        else {  methodName = " Chebyshev "}

        plot(z, sapply(z,FUN), type="l", ylim=range(c(y,f)) )
        title(main = paste("Lagrange interpolation with ",
                        toString(n+1), methodName,
                        " nodes", sep=""))
        lines(z,f, col = 'blue')

        legend('topright', legend=c('function','interpolation'),
              col = c('black','red'), lwd=1)

     }
     return(f)
}
```

2. Testez votre fonction `interpolLagrange` en utilisant les deux fonctions test proposées en introduction, en faisant varier ses différents paramètres. Affichez le résultat (`Plot = TRUE`).

Exercice 6 (Interpolation polynomiale par morceau)**.**
Rappelons que l'erreur (en norme infinie) de la méthode d'interpolation est contrôlée par $\|f^{(n+1)}\|.\|\omega_{n+1}\| \leq \|f^{(n+1)}\|.(b-a)^{n+1}/(n+1)!$. D'où l'idée d'utiliser des polynômes de degré modéré sur des intervalles de longueur h petite. L'interpolation polynomiale par morceau consiste à partitionner $[a,b]$ en sous intervalles $I_1, \ldots, I_K$, par exemple de type $I_k = [a_k, a_{k+1}[$. Soit $h_k = |I_k|$ et $h = \max_{k=1}^{K} h_k$. Pour simplifier, on pourra prendre les h_k égaux, $h_k = \frac{b-a}{K}$. L'interpolation par morceau de degré n sur les K sous intervalles approche f par des fonctions 'polynomiales par morceaux', c'est à dire par une fonction de type

$$v : [a,b] \to \mathbb{R}, \qquad \text{avec } v|_{I_k} \in \mathbb{P}_n$$

On se donne une méthode d'interpolation, c'est à dire le degré n du polynôme interpolateur et un type

noeuds (par exemple, des noeuds équidistants ou de Tchebychev). On applique cette méthode sur chaque intervalle : on obtient les noeuds $x_0^k, \dots, x_n^k$ sur chaque sous-intervalle I_k. Le polynôme d'interpolation de Lagrange par morceaux d'une fonction f est la fonction (polynomiale par morceaux) $LM_n^h f$: telle

$$LM_n^h f|_{I_k} \in \mathbb{P}_n \text{ et } \forall i \in \{0, \dots, n\}, \forall k \in \{1, \dots K\}, LM_n^h f(x_i^k) = f(x_i^k).$$

Étant donnés une subdivision $I_1, \dots, I_k$, un degré n et une méthode de construction de noeuds, l'interpolé par morceau existe et est unique. Attention : rien n'assure sa continuité aux extrémités a_k des sous-intervalles.

1. Donner une majoration de l'erreur $\|E_n^M\|_\infty = \|LM_n^h f - f\|_\infty$, en fonction de h et de $f^{(n+1)}$.
2. Écrire une fonction `piecewiseInterpol` en complétant le squelette de code suivant. La fonction doit renvoyer la liste $[z, f]$ où z est une grille d'interpolation et f est la valeur de l'interpolation par morceau de `fun` évaluée sur `z`.

```
piecewiseInterpol=function(n,nInt,a,b,neval, nodes = "equi", FUN, Plot){
    ## @param n : the degree of the interpolating polynomial on each
    ## subinterval
    ## @param nInt : the number of sub-intervals
    ## @param a, b : endpoints of the interval
    ## @param neval : the number of points on the interpolating grid (on
    ## each subinterval)
    ## @param nodes : string, either "equi" (default) for equidistant
    ## Lagrange interpolation (on each subinterval) or "cheby" for
    ## chebyshev nodes.
    ## @param FUN the function to be interpolated
    ## @param Plot : logical. Should the result be plotted ?
    ## @return : a matrix with 2 rows and neval * nInt -neval + 1:
    ## values of the interpolated funtion on a regular grid (first row)
    ## and the corresponding abscissas (second row).

    intEndPoints = seq(a,b,length.out = nInt+1)
    f = c()
    z = c()
    for (m in 1:nInt){
        A = intEndPoints[m]; B = intEndPoints[m+1]
        fm = ## complete the code
        zm = ## complete the code

        if( m >= 2){
          ## remove first element of zm, fm to avoid
          ## duplicate values of the interpolating vector

          ## Complete the code
        }
        z = c(z,zm)
        f = c(f,fm)
    }

    if (Plot == 1){
        if (nodes == "equi") {methodName = " equidistant "}
        else {methodName = " Chebyshev "}

        plot(z, sapply(z,FUN),type="l")
        title(main = paste("Piecewise Lagrange interpolation with ",
                           toString(n+1), methodName, " nodes on ",
                           toString(nInt), " Intervals", sep=""))
        lines(z,f, col='red', lwd=2)
```

```
        legend('topright', legend = c('function','interpolation'),
               lwd=c(1,2), col=c('black','red'))
    }
    return(rbind(f,z) )
}
```

3. Testez votre fonction sur les deux exemples habituels. Comparez les résultats obtenus avec
   - `piecewiseInterpol`, $n = 4$, $nInt = 15$ (une soixantaine d'évaluation de f)
   - `interpolLagrange`, $n = 60$ (c'est à dire, en utilisant à peu près autant d'information sur f : 61 points).