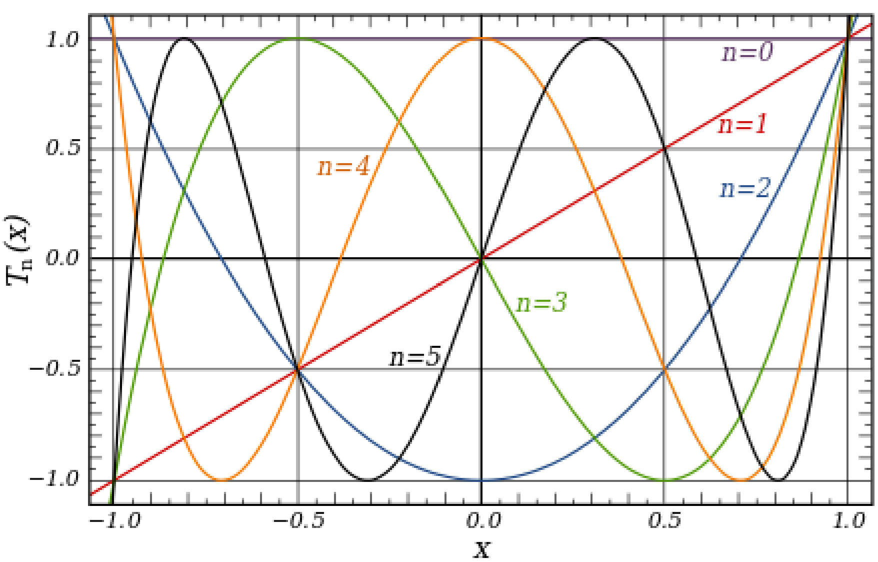
AlgoNum GM3
2022/2023

Evaluation Polynomiale

AlgoNum - TD1 - MMSN



Etudiants : DANTAS Alexandre, DELL'OVA Fabio, GUINES Antoine, LANGOLFF Clément, KESSLER Aymeric

Encadrant : Gleyse. B

Contents

Soit un polynôme $P_n(x) = \sum_{j=0}^{n} a_j x^j$ de degré n à coefficients réels. On cherche à calculer sa valeur en un point α réel.

1 Méthode de Hörner

1.1 Algorithme de Hörner

On sait que $P_n(x) = \sum_{j=0}^{n} a_j x^j$. Si l'on divise P_n par $(x-\alpha)$ alors P_n peut s'écrire tel que :

$$P_n(x) = Q(x)(x-\alpha) + R_n$$

avec Q un polynôme de degré $n-1$. On peut donc écrire $Q(x) = \sum_{j=1}^{n} b_j x^{j-1}$ et on pose $R_n = b_0$. Ainsi il vient que :

$$\sum_{j=0}^{n} a_j x^j = \sum_{j=1}^{n} b_j x^{j-1}(x-\alpha) + b_0$$

$$\Leftrightarrow \sum_{j=0}^{n} a_j x^j = \sum_{j=1}^{n} b_j x^j - \sum_{j=1}^{n} b_j \alpha x^{j-1} + b_0$$

$$\Leftrightarrow \sum_{j=0}^{n} a_j x^j = b_n x^n + (b_{n-1} - \alpha b_n)x^{n-1} + ... + (b_1 - \alpha b_2)x + b_0 - \alpha b_1$$

En identifiant terme à terme, on remarque que :

$$\begin{cases} b_n = a_n \\ a_{k-1} = b_{k-1} - \alpha b_k \ \forall k \in \{1, .., n-1\} \end{cases} \iff \begin{cases} b_n = a_n \\ b_{k-1} = a_{k-1} + \alpha b_k \ \forall k \in \{1, .., n-1\} \end{cases}$$

Cette formule de réccurence nous permet finalement d'obtenir la valeur de b_0, ce qui correspond à la valeur de $P_n(\alpha)$ car $P_n(\alpha) = R_n = b_0$. On en déduit ainsi l'algorithme suivant pour calculer $P_n(\alpha)$:

Algorithm 1 Algorithme de Hörner

```
DEBUT
    lire(n)        /lecture degrès du polynôme/
    POUR i de 0 à n FAIRE
        lire(a[i])        /lecture des coefficients/
    FIN POUR
    b[n] ← a[n]
    POUR i de n-1 à 0 FAIRE
        b[i] ← a[i] + α*b[i+1]        /calcul des bi/
    FIN POUR
    retourner b[0]
FIN
```

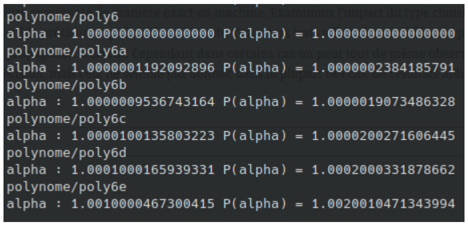
On remarque que lors du calculs des $b[i]$, on a une multiplication et une somme. L'algorithme à donc un coût de n multiplications et n additions. Si l'on considère qu'une multiplication coûte 5 cycles d'horloge et une somme 1 cycle, cela fait 6 cycles par tour de boucle, donc 6n cycles.

Complexité : $O(n)$.

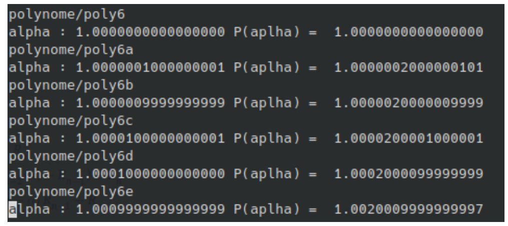
1.2 Stabilité numérique de l'algorithme de Hörner

La représentation des nombres réels en machine étant effectuée grâce aux flottants, certains nombres ne pourront pas être représenté de manière exact en machine. Examinons l'impact du type choisi sur le résultat de l'évaluation de P_n en α. Pour représenter les réels en machine on a donc le choix entre le type simple précision, codé sur 4 octets ou double précision représenté sur 8 octets. Grâce aux tests effectués sur différents polynômes on se rend compte que les 2 types différents donnent des résultats très proches. L'algorithme ne semble donc pas être trop instable quant à la propagation d'erreur. Cependant dans certains cas on peut tout de même observer que le resultat diffère de près d'un centième entre les 2 types pour un même polynôme. Les résultats sont donc légèrement moins précis en simple précision, mais cela ne semble pas donner, dans la plupart des cas, de résultats abberants.

Exemple sur le polynôme $X^2 + 1$



```
polynome/poly6
alpha : 1.0000000000000000 P(alpha) = 1.0000000000000000
polynome/poly6a
alpha : 1.0000001192092896 P(alpha) = 1.0000002384185791
polynome/poly6b
alpha : 1.0000009536743164 P(alpha) = 1.0000019073486328
polynome/poly6c
alpha : 1.0000100135803223 P(alpha) = 1.0000200271606445
polynome/poly6d
alpha : 1.0001000165939331 P(alpha) = 1.0002000331878662
polynome/poly6e
alpha : 1.0010000467300415 P(alpha) = 1.0020010471343994
```



```
polynome/poly6
alpha : 1.0000000000000000 P(aplha) =  1.0000000000000000
polynome/poly6a
alpha : 1.0000001000000001 P(aplha) =  1.0000002000000101
polynome/poly6b
alpha : 1.0000009999999999 P(aplha) =  1.0000020000009999
polynome/poly6c
alpha : 1.0000100000000001 P(aplha) =  1.0000200001000001
polynome/poly6d
alpha : 1.0001000000000000 P(aplha) =  1.0002000099999999
polynome/poly6e
alpha : 1.0009999999999999 P(aplha) =  1.0020009999999997
```

X^2+1	simple	double
1.0	1.0000000000000000	1.0000000000000000
1.0000001	1.0000002384185791	1.0000002000000101
1.000001	1.0000019073486328	1.0000020000009999
1.00001	1.0000200271606445	1.0000200001000001
1.0001	1.0002000331878662	1.0002000099999999
1.001	1.0020010471343994	1.0020009999999997

On remarque que la version double précision est plus précise que la version simple. Cependant l'erreur vient principalement de la façons dont est codé le point d'évaluation en entrée du programme. Par exemple, le point d'évaluation 1.0001 sera codé en simple précision par 1.0001000165939331 tandis que en version double précision, il sera codé par 1.0001000000000000. Cette erreur de codage à l'entrée du programme peut paraître minime pour un polynôme de degrès 2, elle en sera d'autant plus important lorsque le polynôme sera de beaucoup plus haut degrés.

1.3 Calcul des dérivées de $P_n(\alpha)$

Deux méthodes nous permettent de parvenir au résultat.

1.3.1 Méthode 1

On sait que :

$$\begin{aligned} P_n(x) &= (x-\alpha)P_{n-1}(x) + R_n \\ &= (x-\alpha)((x-\alpha)P_{n-2}(x) + R_{n-1}) + R_n \\ &= (x-\alpha)^2 P_{n-2} + (x-\alpha)R_{n-1} + R_n \end{aligned}$$

Par descente infinie, il vient que :

$$P_n(x) = \sum_{j=0}^{n} (x-\alpha)^j R_{n-j} = \sum_{k=0}^{n} (x-\alpha)^{n-k} R_k$$

On peut le vérifier par récurrence. Supposons la propriété vraie pour un rang n, et montrons qu'elle l'est aussi au rang $n+1$. On a alors :

Hérédité :

$$\begin{aligned} P_{n+1}(x) &= (x-\alpha)P_n + R_{n+1} \\ &= (x-\alpha)\sum_{k=0}^{n}(x-\alpha)^{n-k}R_k + R_{n+1} \\ &= \sum_{k=0}^{n+1}(x-\alpha)^{n+1-k}R_k \end{aligned}$$

Ce qui prouve la propriété au rang $n+1$.

Si l'on dérive cette forme, on a :

$$P_n^{(n-j)}(x) = (n-j)!R_j + (n-j)! \sum_{k=n-j+1}^{n} (x-\alpha)^{k-(n-j)} R_{n-k} \prod_{i=k}^{k-(n-j)+1} i$$

En l'évaluant en α on obtient que :

$$\begin{aligned} P_n^{(n-j)}(\alpha) &= (n-j)!R_j + 0 \\ &= (n-j)!R_j \end{aligned}$$

1.3.2 Méthode 2

En faisant un développement de Taylor de P_n à l'ordre n, on a $P_n = \sum_{k=0}^{n} \frac{P_n^{(k)}(\alpha)}{k!}(x-\alpha)^k$.

On a aussi $P_n(x) = \sum_{k=0}^{n}(x-\alpha)^k R_{n-k}$. En faisant une identification terme à terme, on obtient que

$$R_{n-k} = \frac{P_n^{(k)}(\alpha)}{k!}$$

D'où, par un changement de variable en posant $j = n-k$, on obtient que $P_n^{(n-j)}(\alpha) = (n-j)!R_j$.

1.3.3 Méthode de Calcul

Pour calculer la dérivé à l'ordre $(n-j)$ de P_n il nous suffit d'après le résultat précédent de connaitre R_j. Pour obtenir les différents R_j $\forall j \in \{1,..,n\}$, nous allons utiliser l'algorithme de Hörner précédemment trouvé, et nous allons l'appliquer successivement aux $P_{n-j}, j \in \{1,..,n\}$. On a vu précédemment que $P_n(x)$ pouvait s'écrire sous la forme $P_n(x) = P_{n-1}(x-\alpha) + R_n$ avec $P_{n-1} = \sum_{j=1}^{n} b_j x^{j-1}$ et $R_n = b_0$.

De même, on peut écrire P_{n-1} tel que $P_{n-1}(x) = P_{n-2}(x-\alpha) + R_{n-1}$. Puisque l'on vient d'appliquer Hörner à P_{n-1}, nous connaissons l'intégralité de ses coefficients b_j. Ainsi on procède de la même manière que dans la section **1.1**, en écrivant $P_{n-2}(x) = \sum_{j=2}^{n} c_j x^{j-2}$, et $R_{n-1} = c_1$. On a cette fois :

$$\begin{cases} c_n = b_n \\ c_{k-1} = b_{k-1} + \alpha c_k \ \forall k \in \{2,..,n-1\} \end{cases}$$

Ce qui nous permet d'obtenir c_1, qui correspond à R_{n-1}. On procède ainsi pour déterminer tous les R_j, et on applique la formule trouvée dans la section précédente pour calculer la dérivée voulue.

2 Base de Tchebychev

2.1 Algorithme de Calcul de $P_n(\alpha)$

2.1.1 Méthode 1 :

On considère la base des polynômes de Tchebychev T_j pour $0 \leq j \leq n$ définie par

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{m+1} &= 2xT_m(x) - T_{m-1}(x) \end{aligned}$$

avec $1 \leq m \leq n-1$

L'expression de P_n dans cette base est donnée par $P_n(x) = \sum_{j=0}^{n} p_j T_j$. L'objectif est de trouver une expression itérative des coefficients de chaque degrès de P_n.

On a vu au paragraphe d'avant que $P_n = (x-\alpha)Q(x) + r$ avec $Q(x)$ un polynôme degrès inférieur à $n-1$. Ainsi $Q(x) = \sum_{j=0}^{n-1} q_j T_j$ dans la base de Tchebychev.

$$\begin{aligned}
P_n &= (x-\alpha)\sum_{j=0}^{n-1} q_j T_j + rT_0(x) \\
&= \sum_{j=1}^{n-1} q_j x T_j - \sum_{j=1}^{n-1} \alpha q_j T_j + (x-\alpha)q_0 + r \\
&= \sum_{j=1}^{n-1} \frac{q_j}{2}(T_{j+1}(x) + T_{j-1}(x)) - \sum_{j=1}^{n-1} \alpha q_j T_j + (x-\alpha)q_0 + r \\
&= \sum_{j=0}^{n-2} \frac{q_{j+1}}{2} T_j(x) + \sum_{j=2}^{n} \frac{q_{j-1}}{2} T_j(x) - \sum_{j=1}^{n-1} \alpha q_j T_j + (x-\alpha)q_0 + r \\
&= \sum_{j=2}^{n-2} \frac{q_{j+1}}{2} T_j(x) + \frac{q_1}{2} T_0(x) + \frac{q_2}{2} T_1(x) + \sum_{j=2}^{n-2} \frac{q_{j-1}}{2} T_j(x) + \frac{q_{n-1}}{2} T_n(x) + \frac{q_{n-2}}{2} T_{n-1}(x) \\
&- \sum_{j=1}^{n-1} \alpha q_j T_j + (x-\alpha)q_0 + r \\
&= \sum_{j=2}^{n-2} (\frac{q_{j+1} + q_{j-1}}{2} - \alpha q_j) T_j(x) + (\frac{q_2}{2} + q_0 - \alpha q_1) T_1(x) + (\frac{q_1}{2} - \alpha q_0 + r) T_0(x) \\
&+ \frac{q_{n-1}}{2} T_n(x) + (\frac{q_{n-2}}{2} - \alpha q_{n-1}) T_{n-1}(x)
\end{aligned}$$

On peut constater par identification des coefficients que

$$\begin{aligned}
p_n &= \frac{q_{n-1}}{2} \\
p_{n-1} &= \frac{q_{n-2}}{2} - \alpha q_{n-1}
\end{aligned}$$

Puis pour $2 \leq j \leq n-2$

$$p_j = \frac{q_{j+1} + q_{j-1}}{2} - \alpha q_j$$

et enfin

$$p_1 = \frac{q_2}{2} + q_0 - \alpha q_1$$
$$p_0 = \frac{q_1}{2} - \alpha q_0 + r$$

Nous pouvons ainsi définir une méthode de récurrence pour calculer les coefficients de P_n et trouver l'évaluation de $P_n(\alpha)$ à la fin de l'itération.

Algorithm 2 Algorithme d'évaluation de P en α

```
VAR    q0 , q1 , r , qtemp : Réel; j : Entier; p[0..n] tableau de Réel
DEBUT
    q1 ← 2 * p[n]
    q0 ← 2 * p[n-1]+α * q1
    POUR j de n-2 à 2 pas de -1 FAIRE
        qtemp ← 2 * ( p[j] + α * q0 ) - q1
        q1 ← q0
        q0 ← qtemp
    FIN POUR
    qtemp ← p[1] - q1 * 0.5 + α * q0
    q1 ← q0
    q0 ← qtemp
    r ← p[0] + α * q1 - q0 * 0.5
    écrire r
FIN
```

On remarque que cet algorithme effectue 2 multiplications et 2 additions à chaque boucle. Il effectue également 3 multiplications et une addition pour l'initialisation, ainsi que 4 additions et multiplications après la fin de la boucle. Ainsi :

- Coût en multiplication : $3 + 2 * (n - 1) + 4 = 2n + 5$
- Coût en addition : $1 + 2 * (n - 1) + 4 = 2n + 3$

2.1.2 Méthode 2 :

On sait que l'on a les expressions suivantes :
$$\begin{cases} P_n(x) = \sum_{j=0}^{n} \lambda_j T_j(x) \\ T_j(x) - 2xT_{j-1}(x) + T_{j-2}(x) = 0 \end{cases}$$
Ainsi on a :

$$\sum_{j=2}^{n} C_j(T_j(x) - 2xT_{j-1}(x) + T_{j-2}(x)) = 0$$

$$\iff C_n(T_n(x) - 2xT_{n-1}(x) + T_{n-2}(x)) + C_{n-1}(T_{n-1}(x) - 2xT_{n-2}(x) + T_{n-3}(x)) + ...$$
$$... + C_3(T_3(x) - 2xT_2(x) + T_1(x)) + C_2(T_2(x) - 2xT_1(x) + T_0(x)) = 0$$

$$\iff C_nT_n(x) + (C_{n-1} - 2xC_n)T_{n-1}(x) + (C_{n-2} - 2xC_{n-1} + C_n)T_{n-2}(x) + ...$$
$$... + (C_2 - 2xC_3 + C_4)T_2(x) + (C_3 - 2xC_2)T_1(x) + C_2T_0(x) = 0$$

$$\iff \underbrace{C_nT_n(x) + (C_{n-1} - 2xC_n)T_{n-1}(x) + (\sum_{j=2}^{n-2}(C_j - 2xC_{j+1} + C_{j+2})T_j(x))}_{P_n(x) - \lambda_1T_1 - \lambda_0T_0} \quad (*)$$
$$+ (C_3 - 2xC_2)T_1(x) + C_2T_0(x) = 0$$

Nous avons alors :

$$P_n(x) - \lambda_1T_1 - \lambda_0T_0 + (C_3 - 2xC_2)T_1(x) + C_2T_0(x) = 0$$

Ce qui revient à écrire :

$$P_n(x) = (\lambda_1 - C_3 + 2xC_2)T_1(x) + (\lambda_0 - C_2)T_0(x)$$

Ainsi $P_n(x)$ peut s'écrire à l'aide des uniques polynômes T_0 et T_1 (les seules polynômes de Tchebychev dont on connait l'expression direct).

De plus, $(*)$ nous permet d'identifier les $n-2$ derniers coefficients du polynôme. Nous obtenons alors :

$$\begin{cases} C_n = \lambda_n \\ C_{n-1} - 2xC_n = \lambda_{n-1} \\ C_j - 2xC_{j+1} + C_{j+2} = \lambda_j \quad \forall j \in \{2, ..., n-2\} \end{cases}$$

On peut se poser la question si le système a toujours une solution. Pour cela, écrivons le système sous forme matricielle.

$$\underbrace{\begin{bmatrix} 1 & 0 & \dots & & \dots & 0 \\ -2x & 1 & 0 & \dots & \dots & 0 \\ 1 & -2x & 1 & 0 & & \vdots \\ 0 & 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 1 & -2x & 1 \end{bmatrix}}_{T} \begin{bmatrix} C_n \\ C_{n-1} \\ \vdots \\ \vdots \\ \vdots \\ C_2 \end{bmatrix} = \begin{bmatrix} \lambda_n \\ \lambda_{n-1} \\ \vdots \\ \vdots \\ \vdots \\ \lambda_2 \end{bmatrix}$$

On remarque que T est une matrice triangulaire inférieur à diagonale unité. Son déterminant vaut 1, elle est donc inversible, ce qui signifie que le sysème ci-dessus possède une unique solution, et nous permet donc d'affirmer que les C_i existent bien pour $i \in \{2, \dots, n-2\}$.

A l'aide de cette partie théorique nous en déduisons un algorithme pour calculer $P_n(\alpha)$ quelque soit $\alpha \in \mathbb{R}$. Cet algorithme se nomme l'algorithme de Clenshaw. Il consiste à déterminer les C_i pour $i \in \{2, \dots, n-2\}$.

$$\begin{cases} C_n = \lambda_n \\ C_{n-1} = \lambda_{n-1} + 2xC_n \\ C_j = \lambda_j + 2xC_{j+1} - C_{j+2} \quad \forall j \in \{2, ..., n-2\} \end{cases}$$

Enfin, nous évaluons le polynôme en x avec

$$P_n(x) = (\lambda_1 - C_3 + 2xC_2)T_1(x) + (\lambda_0 - C_2)T_0(x)$$

Algorithme 3 Algorithme d'évaluation de P en α

VAR C_1 , C_2 , C_3 , r ,x: **Réel;** j **:** **Entier;** p[0..n] tableau de Réel
DEBUT
  $C_3 \leftarrow$ p[n]
  x = 2 * α
  $C_2 \leftarrow$ p[n-1] + x * C_3
  POUR j allant de n-2 à 2 pas de -1 **FAIRE**
    $C_1 \leftarrow$ p[j] + x * C_2 - C_3
    $C_3 \leftarrow C_2$
    $C_2 \leftarrow C_1$
  FIN POUR
  r $\leftarrow$ (p[1] - C_3 + x * C_2) * α + (p[0] - C_2)
  écrire r
FIN

Coût de l'algorithme :

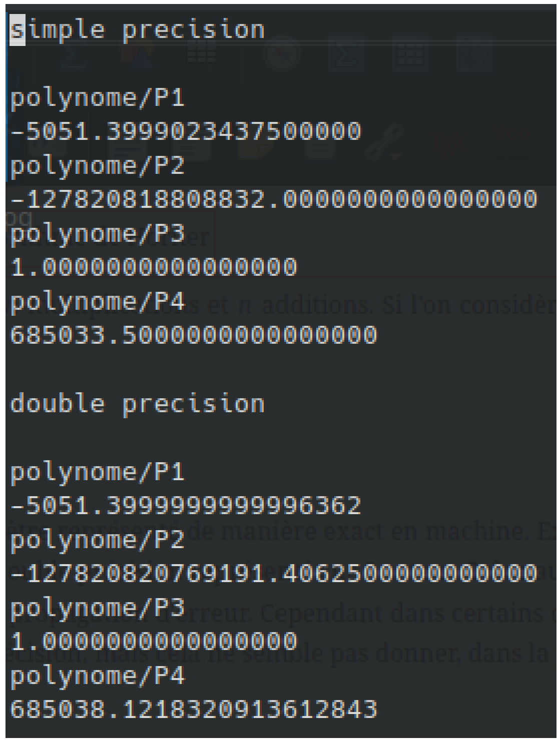
- n multiplications
- $2(n-4)+5 = 2n-3$ additions

Ce qui donne un algorithme en $O(3n)$

2.2 Stabilité numérique

En plus de l'exemple proposé sur la fiche de TD, nous avons testé l'évaluation d'un polynôme de degré 100 pour vérifier la stabilité de l'algorithme de Clenshaw.

Exemple : évaluation de X^{100}



```
simple precision

polynome/P1
-5051.3999023437500000
polynome/P2
-127820818808832.0000000000000000
polynome/P3
1.0000000000000000
polynome/P4
685033.5000000000000000

double precision

polynome/P1
-5051.3999999999996362
polynome/P2
-127820820769191.4062500000000000
polynome/P3
1.0000000000000000
polynome/P4
685038.1218320913612843
```

Pour un point d'évaluation en entrée égale à 1.0, l'algorithme nous donne effectivement le bon résultat que ça soit en simple précision ou en double précision. Cependant pour un légère variation aux environs de 1.0, nous avons remarqué de grosses variations. La version simple donne pour une évaluation en 1,01 le résultat 685033.5000000000000000 et la version double 685038.1218320913612843. Alors que le résultat de $(1.01)^{100}$est 2.70481382942.

3 Applications I

On condidère dans cette partie le résultat suivant :

$$\forall |a| < 1 \; ln(1 - 2acos(\theta) + a^2) = -2\sum_{n=1}^{\infty} \frac{a^n}{n} cos(n\theta)$$

3.1 Calcul de ln(10)

On cherche à calculer $ln(10)$ avec une précision supérieure à 10^{-4}. On a :

$$\begin{aligned}
ln(10) &= ln((1+\frac{1}{9})\times 9)\\
&= ln(9) + ln((1+\frac{1}{9})\\
&= ln(9) + ln(1 - 2\times\frac{1}{3}\times cos(\frac{\pi}{2}) + (\frac{1}{3})^2)\\
&= ln(9) - 2\times\sum_{n=1}^{\infty}\frac{1}{n}\times(\frac{1}{3})^n\times cos(n\times\frac{\pi}{2})\\
&= ln(9) - 2\times\sum_{p=1}^{\infty}\frac{1}{2p}\times(\frac{1}{3})^{2p}\times cos(2p\times\frac{\pi}{2})\\
&= ln(9) - \sum_{p=1}^{\infty}\frac{1}{p}\times(\frac{1}{9})^p\times cos(p\pi)\\
ln(10) &= ln(9) - \sum_{p=1}^{\infty}\frac{1}{9^p p}\times(-1)^p
\end{aligned}$$

Notre but étant d'approcher $ln(10)$ à 10^{-4} près cela revient à chercher le plus petit n vérifiant

$$|ln(10) - ln(9) + \sum_{p=1}^{n}\frac{1}{9^p p}\times(-1)^p| < 10^{-4}$$

$$\iff |ln(9) - \sum_{p=1}^{\infty}\frac{1}{9^p p}\times(-1)^p - ln(9) + \sum_{p=1}^{n}\frac{1}{9^p p}\times(-1)^p| < 10^{-4}$$

$$\iff |\sum_{p=n+1}^{\infty}\frac{1}{9^p p}\times(-1)^p| < 10^{-4}$$

On pose $\forall n\in\mathbb{N}^*$ $a_n = \frac{1}{9^n n}$, alors comme $(a_n)_n$ est décroissante positive on a par le critère des séries alternées que :

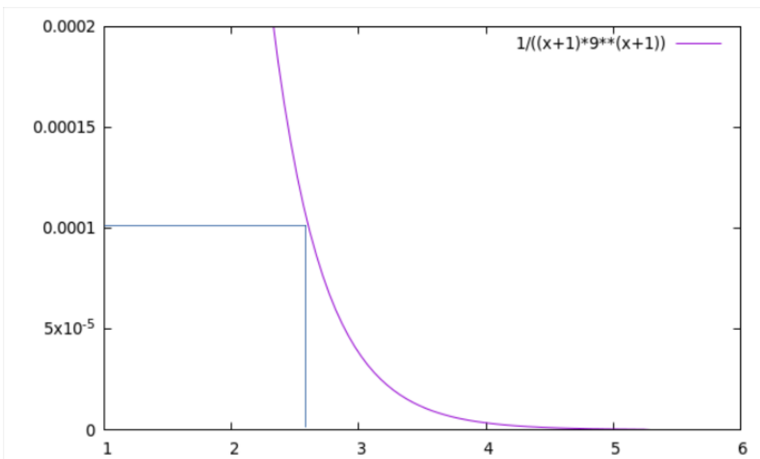
$$|R_n| < a_{n+1}$$

avec $R_n = \sum_{p=n+1}^{\infty} \frac{1}{9^p p} \times (-1)^p$
Ainsi, on obtient l'inégalitée suivante :

$$|\sum_{p=n+1}^{\infty} \frac{1}{9^p p} \times (-1)^p| < \frac{1}{9^{n+1}(n+1)}$$

On cherche donc n tel que : $\frac{1}{9^{n+1}(n+1)} < 10^{-4}$. On remarque que $n = 3$ est le premier entier permettant d'approcher $ln(10)$ avec une précision supérieure à 10^{-4}. En effet on obtient pour $n = 3$:

$$\frac{1}{9^4 \times 4} \approx 3,81 \times 10^{-5} < 10^{-4}$$



graphe de la fonction $\frac{1}{(n+1)9^{n+1}}$

Ainsi en prenant $n = 3$, on s'assure d'avoir une précision supérieure à 10^{-4} pour le calcul de $ln(10)$.

Calcul à la main de ln(10)

On calcule ln(10) avec la formule $ln(10) = ln(9) - \sum_{p=1}^{\infty} \frac{1}{9^p p} \times (-1)^p$ en prenant p variant de 1 à $n = 3$.

$$
\begin{aligned}
ln(10) &= ln(9) + \frac{1}{9} - \frac{1}{9^2 \times 2} + \frac{1}{9^3 \times 3} \\
&= 2,1972 + \frac{1}{9} - \frac{1}{162} + \frac{1}{2187} \\
&= 2,302595
\end{aligned}
$$

Calcul de ln(10) par l'ordinateur

```
Calcul de ln(10) avec une précision de 10e-4

2.302557
```

Les deux calculs approchent bien la valeur de ln(10) à 10^{-4} près; on peut remarquer une légère différence au niveau des chiffres significatifs après 10^{-4}. Cette différence s'explique par le codage des fractions par l'ordinateur qui peut provoquer de légères variations.

3.2 Développement de $ln(10 + 6x)$

À partir de l'expression donnée on développe, on a :

$$
\begin{aligned}
ln(10 + 6x) &= ln(9 + 2 \times 3x + 1) \\
&= ln(9 - 2 \times 3cos(\theta) + 1),\ \theta \in [0; \pi] \implies cos(\theta) \in [1; -1] \\
&= ln(1 + 2 \times (-\frac{1}{3}) \times cos(\theta) + (-\frac{1}{3})^2) + ln(9) \\
&= ln(9) - 2 \times \sum_{n=1}^{\infty} \frac{1}{(-3)^n n} cos(n\theta)
\end{aligned}
$$

On veut pouvoir écrire que $ln(10 + 6x) = \sum_{n=0}^{\infty} b_n T_n$.
Montrons par récurrence la propriété P_n qui est la suivante : $\forall n \in \mathbb{N}^*$, le couple (T_n, T_{n-1}) vérifie $T_n(x) = cos(n\theta)$ et $T_{n-1}(x) = cos((n-1)\theta)$, avec $x = cos(\theta)$.
Initialisation :
Pour $n = 0$, on a $T_0(x) = 1 = cos(0 \times \theta)$

Pour $n = 1$, on a $T_1(x) = cos(\theta) = cos(arccos(x)) = x$
Donc le couple définit par (T_0, T_1) vérifie bien l'hypothèse de récurrence, ainsi P_1 est vraie.
Hérédité : On suppose maintenant la propriété P_n vraie au rang n, c'est à dire que le couple (T_n, T_{n-1}) vérifie l'hypothèse de récurrence. Montrons qu'alors P_{n+1} est aussi vraie, c'est à dire que le couple (T_{n+1}, T_n) vérifie :

$$\begin{cases} T_n(x) = cos(n\theta), \; x = cos(\theta) \\ T_{n+1}(x) = cos((n+1)\theta), \; x = cos(\theta) \end{cases}$$

Par hypothèse de récurrence il est évident que $T_n(x) = cos(n\theta)$. Montrons maintenant l'égalité pour T_{n+1}.
On a :

$$\begin{aligned} T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \\ &= 2cos(\theta)T_n(x) - T_{n-1}(x) \\ &= 2cos(\theta)cos(n\theta) - cos((n-1)\theta) \\ &= 2cos(\theta)cos(n\theta) - cos(\theta)cos(n\theta) - sin(n\theta)sin(\theta) \\ &= cos(\theta)cos(n\theta) - sin(n\theta)sin(\theta) \\ &= cos(n\theta + \theta) \\ &= cos((n+1)\theta) \end{aligned}$$

Finalement on obtient bien que $T_{n+1}(x) = cos((n+1)\theta)$ donc P_{n+1} est bien vérifiée.
Ainsi comme $ln(10+6x) = ln(9) - 2 \times \sum_{n=1}^{\infty} \frac{1}{(-3)^n n} cos(n\theta)$, on peut identifier les b_n. On obtient :

$$\begin{cases} b_0 = ln(9) \\ b_n = \frac{-2}{(-3)^n n}, \; \forall n \in \mathbb{N}^* \end{cases}$$

On cherche mainenant le $k \in \mathbb{N}$ tel que :

$$|ln(10+6x) - ln(9) + \sum_{n=1}^{k} \frac{2}{3^n n} \times (-1)^n \times cos(n\theta)| < 10^{-4}$$

$$\iff |ln(9) - 2 \times \sum_{n=1}^{\infty} \frac{1}{(-3)^n n} cos(n\theta) - ln(9) + \sum_{n=1}^{k} \frac{2}{3^n n} \times (-1)^n \times cos(n\theta)| < 10^{-4}$$

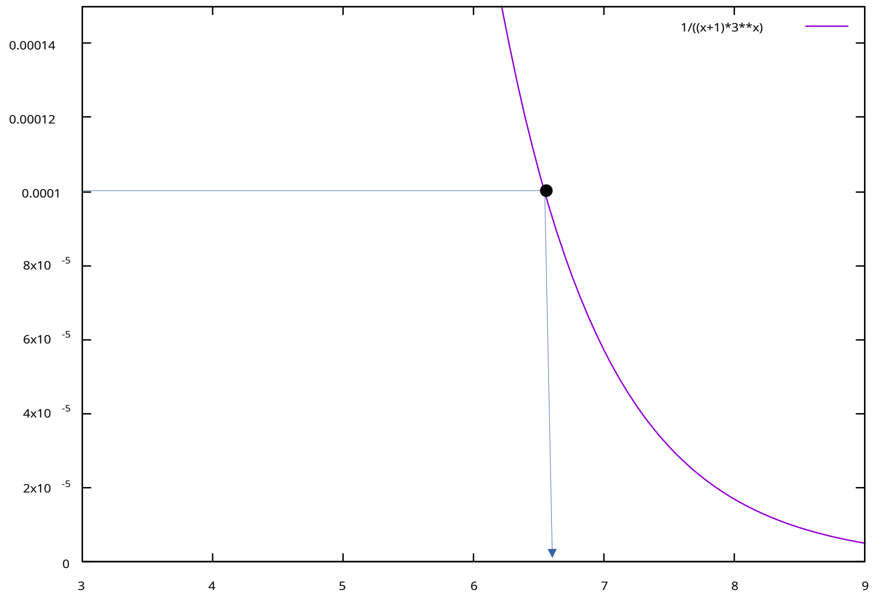
$$\iff |\sum_{n=k+1}^{\infty} \frac{2 \times (-1)^n}{3^n n} cos(n\theta)| < 10^{-4}$$

Or on a :

$$\begin{aligned}
|\sum_{n=k+1}^{\infty} \frac{2 \times (-1)^n}{3^n n} cos(n\theta)| &\leq \sum_{n=k+1}^{\infty} |\frac{2 \times (-1)^n}{3^n n}||cos(n\theta)| \\
&\leq \sum_{n=k+1}^{\infty} |\frac{2}{3^n n}| \\
&\leq 2 \times \frac{1}{k+1} \times \sum_{n=k+1}^{\infty} (\frac{1}{3})^n \\
&= \frac{2}{k+1} \times (\frac{1}{3})^{k+1} \times \frac{1}{1-\frac{1}{3}} \\
&= \frac{1}{(k+1) \times 3^k}
\end{aligned}$$

On remarque ensuite que $k = 7$ est le premier entier tel que $\frac{1}{(k+1) \times 3^k} < 10^{-4}$. En effet on obtient pour $k = 7$:

$$\frac{1}{8 \times 3^7} \approx 5,72 \times 10^{-5} < 10^{-4}$$



graphe de la fonction $\frac{1}{(k+1)3^k}$

Ainsi en prenant $k = 7$, on s'assure d'avoir une précision supérieure à 10^{-4} pour le calcul de $ln(10 + 6x)$.

Exemple

Pour tester le programme, nous avons choisit comme point d'évaluation 0,5 car celui-ci est codé sans erreur par l'ordinateur.

$$
\begin{aligned}
ln(10+6*\frac{1}{2}) = ln(13) \\
&= ln(9) - 2\times\sum_{n=1}^{7}\frac{1}{(-3)^n n}cos(n\theta) \\
&= ln(9) - 2\times(\frac{-1}{3}\times 0,5 + \frac{1}{(-3)^2\times 2}\times cos(2arccos(0.5)) + ... \\
&... + \frac{1}{(-3)^7\times 7}\times cos(7arccos(0.5))) \\
&= 2,56491507
\end{aligned}
$$

Le programme obtient pour une évaluation de $ln(10+6x)$ en 0,5 :

```
Calcul de ln(10 + 6x) avec une précision de 10e-4

PntEvaluation/point1
evalué en
0.5
2.564935
```

3.3 Algorithme de calcul de $ln(10+6x)$

On a précédemment démontré que $\forall x \in [-1;1],\ \ ln(10{+}6x) \approx \sum_{n=0}^{7} b_n T_n(x)$, avec :

$$
\begin{cases} b_0 = ln(9) \\ b_n = \frac{-2}{(-3)^n n},\ \ \forall n \in \mathbb{N}^* \end{cases}
$$

On reprend donc exactement l'algorithme de Clenshaw pour le calcul de $P_n(\alpha)$ développé en section 2.1.2. Simplement on rajoute à chaque tour de boucle le calcul des b_n.

4 Application II

4.1 Développement de $\frac{10+x}{101+20x}$

$$\frac{10+x}{101+20x} = \frac{10\times(1+0,1x)}{10\times(10,1+2x)}$$

On pose $x = cos(\theta) \in [-1,1]$

$$\begin{aligned}
\frac{10+x}{101+20x} &= \frac{1+0,1cos(\theta)}{10,1+2cos(\theta)} \\
&= \frac{1-(-0,1cos(\theta))}{10\times(1,01-2(-0,1)cos(\theta))} \\
&= \frac{1}{10}\times\frac{1-(-0,1cos(\theta))}{1-2(-0,1)cos(\theta)+(-0,1)^2} \\
&= \frac{1}{10}\times\sum_{n=0}^{\infty}(-0,1)^n cos(n\theta)
\end{aligned}$$

Comme $cos(n\theta) = T_n(x)$, on a $\frac{10+x}{101+20x} = \sum_{n=0}^{\infty} b_n T_n(x)$ avec $b_n(x) = \frac{(-1)^n}{10^{n+1}}$
On cherche ensuite à avoir une approximation de $\frac{10+x}{101+20x}$ à 10^{-5}près.
Pour cela, appliquons le critère de Cauchy à la série $S = \sum_{n=0}^{\infty} \frac{(-1)^n}{10^{n+1}} cos(n\theta)$
:

$$\begin{aligned}
|S_{k+1}-S_k| &= \frac{1}{10}|\sum_{n=0}^{k+1}(-0,1)^n cos(n\theta) - \sum_{n=0}^{k}(-0,1)^n cos(n\theta)| \\
&= \frac{1}{10}|(-0,1)^{k+1}cos((k+1)\theta)| < 10^{-5} \\
&\iff |(-0,1)^{k+1}cos((k+1)\theta)| < 10^{-4} \\
&\iff |(-1)^{k+1}\times(0,1)^{k+1}| < 10^{-4} \\
&\iff (0,1)^{k+1} < 10^{-4} \\
&\iff (10^{-1})^{k+1} < 10^{-4} \\
&\iff 10^{-k-1} < 10^{-4} \\
&\iff k > 3
\end{aligned}$$

Ainsi en prenant $k = 4$, nous pourrons obtenir une approche de $\frac{10+x}{101+20x}$ à 10^{-5}près.

4.2 Algorithme d'évaluation de $\frac{10+x}{101+20x}$

Dans cette partie nous reporduisons le raisonnement précédent en écrivant $\frac{10+x}{101+20x}$ comme $\sum_{n=0}^{4} b_n T_n(x)$ *pour* $\forall x \in [-1;1]$ et :

$$\left\{ b_n = \frac{(-1)^n}{10^{n+1}}, \ \forall n \in \mathbb{N} \right.$$

Dans l'algorithme d'évalution en α il nous faut donc calculer les b_n adaptés au nouveau problème. Ainsi à l'aide de l'algorithme de Clenshaw et par le calcul de seulement 5 termes nous réussissons à obtenir le résultat avec une précision de 5 décimales.

Si nous nous intéressons aux résultats informatiques, nous observons que le résultat n'est pas exact avec une précision de 10^{-5} mais de 10^{-4}. En effet, dans cette partie, les $b_n$sont égaux à $0,1 \times (\frac{-1}{10})^n$, cependant le chiffre $(\frac{-1}{10})$ n'est pas représentable exactement en machine; il est impossible de l'écrire comme $2^e(1+m)$, *où* $m \in [0;1[$. Par conséquent, à chaque $b_n$calculé dans la somme et à la fin du calcul le produit par $\frac{1}{10}$ génère au total beaucoup d'approximations et d'erreurs. En allant un peu plus loin, si nous réalisons plus de boucles pour avoir une supposée meilleure précision, nous observons que le résultat se dégrade. Ceci s'explique encore une fois par le fait que le nombre $\frac{1}{10}$ apporte à chaque itération plus d'erreurs et ne permet pas d'obtenir la solution à ce niveau de précision. On peut donc considérer le problème comme instable numériquement.

4.3 Exemple

Dans cette exemple, nous évaluons $\frac{10+x}{101+20x}$ en 1.0 Tout d'abord à la main puis avec le programme. On prend $k = 4$ pour avoir une approximation à 10^{-5} près.

$$\begin{aligned}
\frac{10+x}{101+20x} &= \frac{1}{10} \times \sum_{n=0}^{4} (-0,1)^n cos(n\theta) \\
&= \frac{1}{10} \times (1.0 - 0,1 + (-1,0)^2 \times cos(2arccos(1,0)) + \\
&+ (-1,0)^3 \times cos(3arccos(1,0)) + (-1,0)^4 \times cos(4arccos(1,0))) \\
&= 0,09091
\end{aligned}$$

Le programme donne quant à lui :

```
Calcul de (10 + x)/(101 + 20x) avec une précision de 10e-5

Evalué en :
1.
0.0909100000000000
```

Ce qui est bien fidèle au résultat obtenu à la main.