Lagrange

February 27, 2019

1 Interpolation de Lagrange

Marc Lorenzi
Février 2019

```
In [1]: from sympy import *
        import matplotlib.pyplot as plt
        import math
        init_printing()
        x = Symbol('x')
```

```
In [2]: plt.rcParams['figure.figsize'] = (10, 6)
```

1.1 1. La théorie

Dans tout ce notebook,

- n désigne un entier naturel.
- I est un intervalle de $\mathbb{R}$.
- $x_0 < x_1 < \ldots < x_n$ sont $n+1$ points distincts de I.
- f est une fonction de I vers $\mathbb{R}$.

1.2 1.1 Le problème de l'interpolation

Problème : trouver un polynôme P de degré inférieur ou égal à n tel que

$$\forall i \in [0, n], P(x_i) = f(x_i)$$

Nous allons voir que ce problème admet une unique solution. Le polynôme correspondant est appelé le polynôme d'interpolation de f aux points x_i.

Nous allons dans ce notebook :

1. Montrer l'existence et l'unicité d'un tel polynôme.
2. Calculer efficacement la valeur du polynôme d'interpolation en un point x.
3. Majorer $|f(x) - P(x)|$ pour $x \in I$.
4. Voir comment rendre ce majorant aussi petit que possible en choisissant astucieusement les x_i.
5. Apprendre à marcher sur les mains.
6. Et bien d'autres choses ...

1.2.1 1.2 Les polynômes élémentaires de Lagrange

Pour $k = 0, \ldots, n$ le $k$ième polynôme de Lagrange est le polynôme

$$L_k = \frac{\prod_{j \neq k}(X - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

C'est l'unique polynôme de degré inférieur ou égal à n qui s'annule en tous les x_j, sauf en x_k où il prend la valeur 1.

Proposition : $\mathcal{B} = (L_0, L_1, \ldots, L_n)$ est une base de l'espace $\mathbb{R}_n[X]$ des polynômes de degré inférieur ou égal à n.

Démonstration : Comme Card $\mathcal{B} = \dim \mathbb{R}_n[X]$ il suffit de montrer que $\mathcal{B}$ est, par exemple, libre. Soient $\lambda_k, k = 0, 1, \ldots, n$ $n + 1$ réels. Supposons que

$$\sum_{k=0}^{n} \lambda_k L_k = 0$$

On a pour tous i, k $L_k(x_i) = \delta_{ki}$ où le symbole de Kronecker δ_{ki} vaut 1 si $k = i$ et 0 sinon. Évaluons l'égalité ci-dessus en x_i. Il vient

$$0 = \sum_{k=0}^{n} \lambda_k L_k(x_i) = \sum_{k=0}^{n} \lambda_k \delta_{ki} = \lambda_i$$

ceci pour tout i. D'où la liberté. Tout polynôme P de degré inférieur ou égal à n est donc combinaison linéaire des L_k, d'où leur nom "élémentaires".

La fonction `lagrange` prend en paramètre un entier k, un réel x et une liste $xs = [x_0, \ldots, x_n]$ de réels distincts. Elle renvoie $L_k(x)$.

```
In [3]: def lagrange(k, x, xs):
            p = 1
            n = len(xs) - 1
            for j in range(n + 1):
                if j != k:
                    p *= (x - xs[j]) / (xs[k] - xs[j])
            return p
```

Ayant défini au début du notebook x comme un symbole, nous pouvons donc obtenir une experssion explicite des L_k.

```
In [6]: expand(lagrange(1, x, [-2, -1, 0, 1, 2]))
```

Out[6]:

$$-\frac{x^4}{6} + \frac{x^3}{6} + \frac{2x^2}{3} - \frac{2x}{3}$$

```
In [7]: expand(lagrange(2, x, [-2, -1, 0, 1, 2]))
```

Out[7]:

$$\frac{x^4}{4} - \frac{5x^2}{4} + 1$$

1.2.2 1.3 Subdivisions régulières

Soient $a, b \in \mathbb{R}$, $a < b$. Une subdivision du segment $[a, b]$ est une suite $x_0 = a < x_1 < \ldots < x_n = b$. Pour tout entier $n \geq 1$, la subdivision régulière à $n + 1$ points de $[a, b]$ est définie par

$$x_k = a + k\frac{b-a}{n}, k = 0, \ldots, n$$

```
In [8]: def subdi(a, b, n):
            d = (b - a) / n
            return [a + k * d for k in range(n + 1)]
```

```
In [10]: xs = subdi(-1, 1, S(10))
         xs
```

Out[10]:

$$\left[-1, \quad -\frac{4}{5}, \quad -\frac{3}{5}, \quad -\frac{2}{5}, \quad -\frac{1}{5}, \quad 0, \quad \frac{1}{5}, \quad \frac{2}{5}, \quad \frac{3}{5}, \quad \frac{4}{5}, \quad 1\right]$$

1.2.3 1.4 Graphes des polynômes élémentaires

Nous pouvons maintenant effectuer quelques petits affichages. Commençons par le graphe des polynômes élémentaires.

```
In [12]: xs = subdi(-1, 1, S(5))
         ps = [expand(lagrange(k, x, xs)) for k in range(len(xs))]
         ps
```

Out[12]:

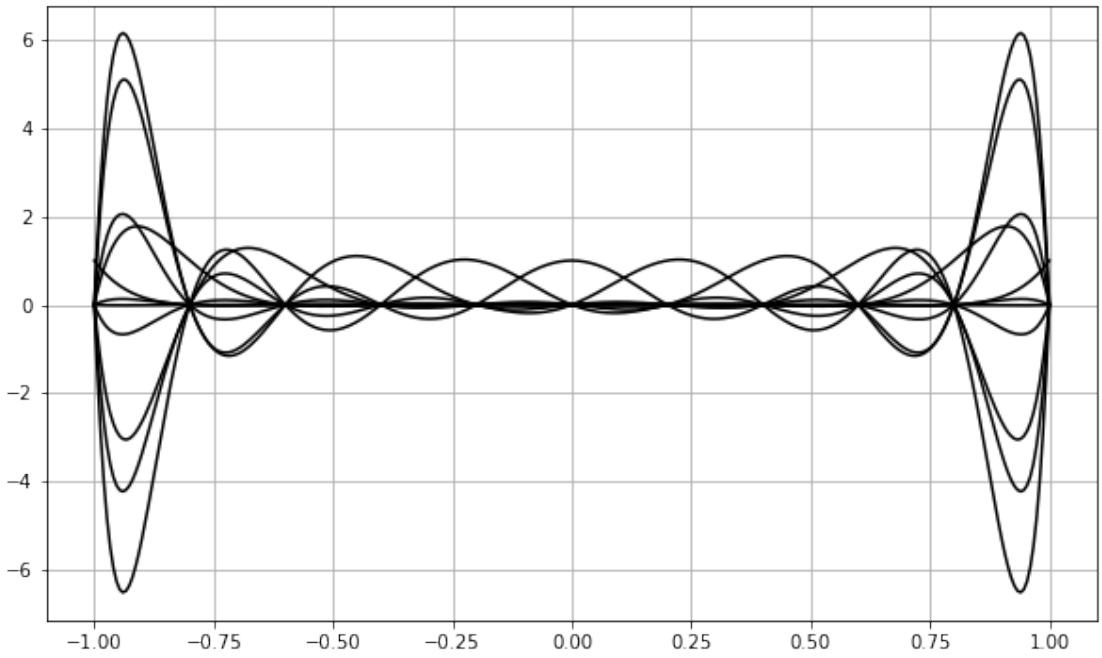
$$\left[-\frac{625x^5}{768} + \frac{625x^4}{768} + \frac{125x^3}{384} - \frac{125x^2}{384} - \frac{3x}{256} + \frac{3}{256}, \quad \frac{3125x^5}{768} - \frac{625x^4}{256} - \frac{1625x^3}{384} + \frac{325x^2}{128} + \frac{125x}{768} - \frac{25}{256}, \quad -\frac{3125x}{384}\right.$$

La fonction `tracer_poly` trace la courbe de p pour x entre `xmin` et `xmax`. Les paramètres optionnels de la fonction sont passés à `plt.plot`.

```
In [13]: def tracer_poly(p, xmin, xmax, *opt1, **opt2):
             xs = subdi(xmin, xmax, 300)
             ys = [p.subs(x, t) for t in xs]
             plt.plot(xs, ys, *opt1, **opt2)
```

Voici les polynômes élémentaires de Lagrange associés à une subdivision régulière.

```
In [14]: xs = subdi(-1, 1, 10)
         ps = [expand(lagrange(k, x, xs)) for k in range(len(xs))]
         for p in ps:
             tracer_poly(p, -1., 1., 'k')
         plt.grid()
         plt.show()
```



Exercice : Parmi les courbes ci-dessus, trouver la courbe de L_3. Facile, L_3 est le seul à ne pas s'annuler en x_3. Une fois que vous l'avez trouvée, surlignez-la sur l'écran de votre ordinateur avec un feutre rouge :-).

1.2.4 1.5 Le polynôme d'interpolation

Proposition : Il existe un unique polynôme P de degré inférieur ou égal à n tel que $P(x_k) = f(x_k)$ pour $k = 0, \ldots, n$. Le polynôme P est donné par

$$P = \sum_{k=0}^{n} f(x_k) L_k$$

où les L_k sont les polynômes élémentaires de Lagrange.

Démonstration : Soit P le polynôme défini ci-dessus. Tout d'abord, P est de degré inférieur ou égal à n. De plus, pour $i = 0, 1, \ldots, n$,

$$P(x_i) = \sum_{k=0}^{n} f(x_k) L_k(x_i) = \sum_{k=0}^{n} f(x_k) \delta_{ki} = f(x_i)$$

Ce polynôme est donc solution du problème. Par ailleurs, supposons que P et Q sont solutions. Le polynôme $P - Q$ est de degré inférieur ou égal à n et s'annule en les $n + 1$ points $x_0, x_1, \ldots, x_n$. C'est donc le polynôme nul. D'où l'unicité.

La fonction `interpoler` prend en paramètres une fonction f et une liste xs de points. Elle renvoie le polynôme d'interpolation de f aux points de la liste xs.

```
In [15]: def interpoler(f, xs):
             ps = [lagrange(k, x, xs) for k in range(len(xs))]
             p = 0
```

```
    n = len(xs)
    for k in range(n):
        p = p + f(xs[k]) * ps[k]
    return p
```

Testons sur un petit exemple. Prenons $f : x \mapsto \frac{1}{x^2+1}$ et interpolons sur une subdivision de $[-2, 2]$. Nous allons fréquemment utiliser cette fonction dans toute la suite, alors appelons-la *l'exemple*.

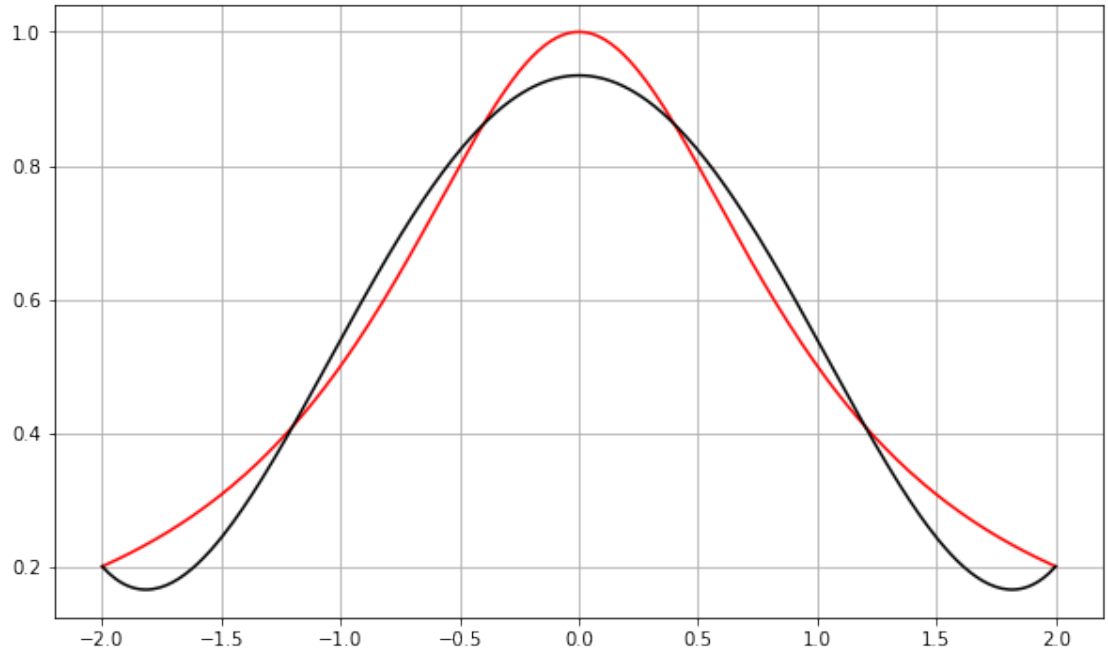
```
In [16]: def exemple(x):
             return 1 / (x ** 2 + 1)
```

```
In [17]: p = expand(interpoler(exemple, subdi(-2, 2, S(5))))
         p
```

Out[17]:

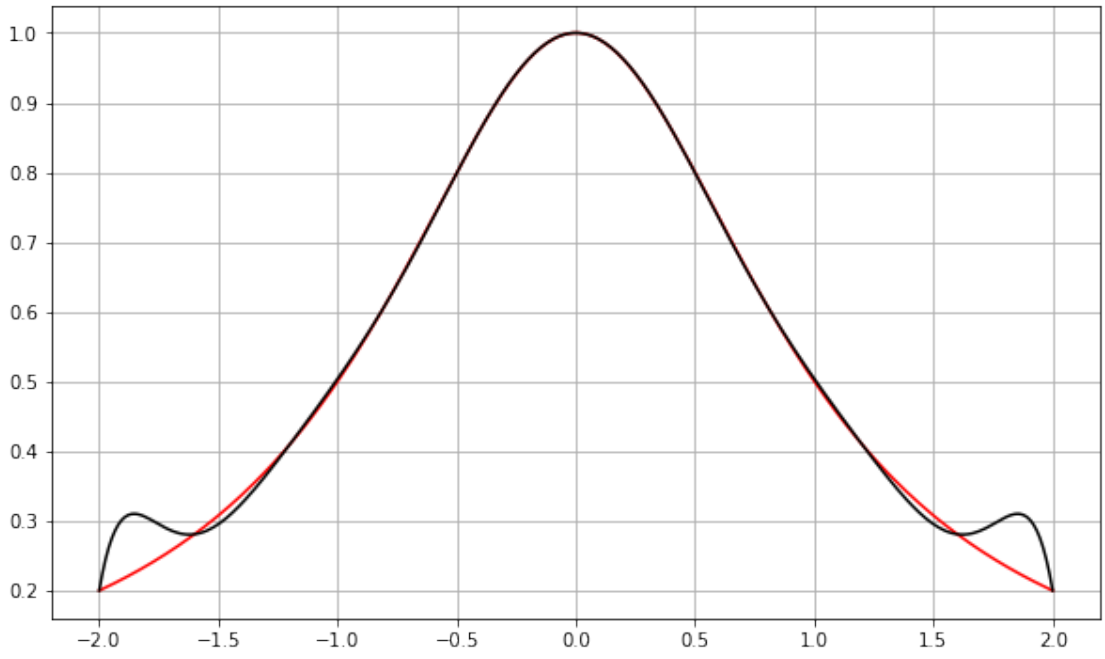
$$\frac{125x^4}{1769} - \frac{825x^2}{1769} + \frac{8269}{8845}$$

```
In [18]: xs = subdi(-2, 2, 300)
         ys = [exemple(t) for t in xs]
         plt.plot(xs, ys, 'r')
         tracer_poly(p, -2, 2, 'k')
         plt.grid()
         plt.show()
```



Que se passe-t-il si on augmente le nombre de points de la subdivision ?

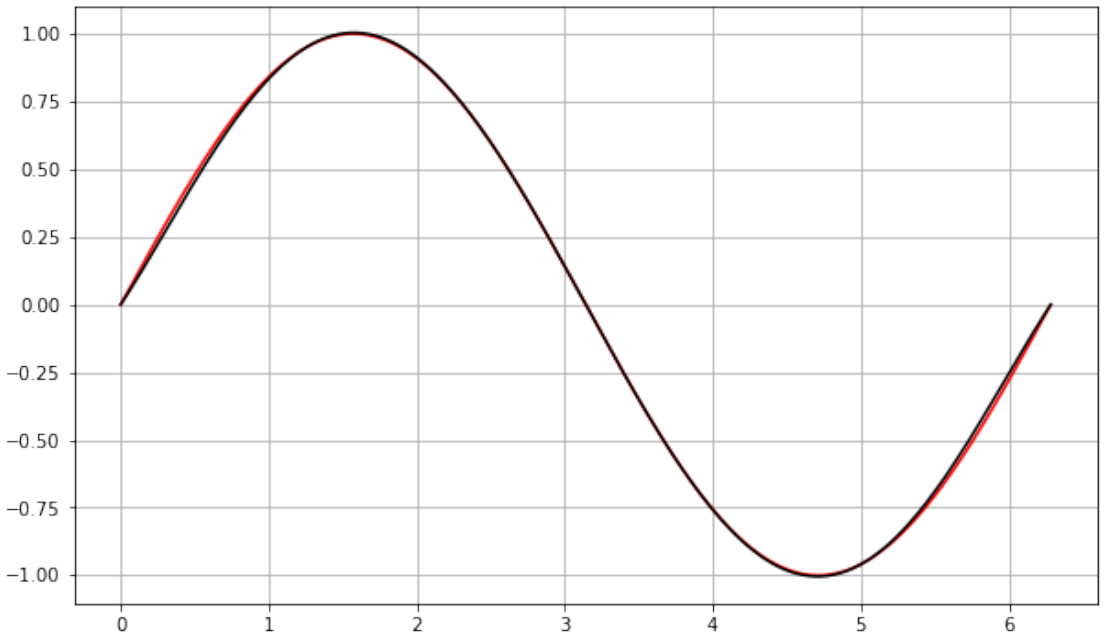
```
In [19]: p = expand(interpoler(exemple, subdi(-2, 2, 10)))
         xs = subdi(-2, 2, 300)
         ys = [exemple(t) for t in xs]
         plt.plot(xs, ys, 'r')
         tracer_poly(p, -2, 2, 'k')
         plt.grid()
         plt.show()
```



Sur cet exemple, augmenter le nombre de points d'interpolation conduit à deux phénomènes : si l'on excepte un voisinage des bords de l'intervalle, on augmente la qualité de l'interpolation. En revanche, au bord de l'intervalle, on assiste à une dégradation de l'interpolation. C'est le phénomène de Runge ...

Un deuxième exemple avant de poursuivre : la fonction sinus sur l'intervalle $[0, 2\pi]$. Nous l'appellerons ... sin :-).

```
In [20]: p = expand(interpoler(math.sin, subdi(0, 2 * math.pi, 5)))
         xs = subdi(0, 2 * math.pi, 300)
         ys = [math.sin(t) for t in xs]
         plt.plot(xs, ys, 'r')
         tracer_poly(p, 0, 2 * math.pi, 'k')
         plt.grid()
         plt.show()
```



Cette fois-ci, la concordance entre la fonction et le polynôme d'interpolation est tout à fait remarquable. Pourquoi ? Nous verrons cela plus loin.

Arrêtons les calculs formels et penchons-nous sur des problèmes numériques. Comment calculer efficacement $P(x)$ pour une valeur donnée de x ?

```
In [21]: init_printing(pretty_print=False)
```

1.3 2. L'algorithme de Neville

Il existe un certain nombre d'algorithmes permettant le calcul de $P(x)$, où P est le polynôme d'interpolation de Lagagrange. Nous allons en examiner un dans cette partie.

1.3.1 2.1 Relations de récurrence sur les polynômes d'interpolation

Proposition : Soit $n \geq 1$. Soit P le polynôme d'interpolation de f. Pour $i = 0, 1, \ldots, n$ soit P_i le polynôme d'interpolation de f aux points $x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. Soient $0 \leq i, j \leq n$, $i \neq j$. Soit $x \in I$. Alors

$$P(x) = \frac{(x - x_j)P_j(x) - (x - x_i)P_i(x)}{x_i - x_j}$$

Démonstration : Posons

$$Q = \frac{(X - x_j)P_j - (X - x_i)P_i}{x_i - x_j}$$

Soit $k \in [0, n]$. Trois cas se présentent :

- $k \neq i, j$. Alors

$$Q(x_k) = \frac{(x_k - x_j)P_j(x_k) - (x_k - x_i)P_i(x_k)}{x_i - x_j} = \frac{(x_k - x_j)f(x_k) - (x_k - x_i)f(x_k)}{x_i - x_j} = f(x_k)$$

- $k = i$. On a

$$Q(x_i) = \frac{(x_i - x_j)P_j(x_i) - (x_i - x_i)P_i(x_i)}{x_i - x_j} = P_j(x_i) = f(x_i)$$

- $k = j$. Identique au cas précédent.

Remarquons enfin que le degré de Q est inférieur ou égal à n. Par l'unicité du polynôme d'interpolation, on a $Q = P$.

1.3.2 2.2 L'algorithme

Changeons un peu nos notations par rapport au paragraphe précédent et notons bien lourdement $P_{i_1 \ldots i_k}$ le polynôme d'interpolation de f aux points $x_{i_1}, \ldots, x_{i_k}$. Ce qu nous voulons, c'est calculer $P_{01\ldots n}$.

On connaît évidemment P_j, polynôme d'interpolation de f en l'unique point x_j, qui est constant égal à $f(x_j)$. Dans la fonction `neville` ci-dessous, Q est une liste de $n+1$ flottants. Pour $j = 0, \ldots, n$, on initialise $Q[j]$ à la valeur $f(x_j)$. Suit une boucle indexée par j, allant de 1 à n :

- Première itération : pour $k = n, n-1, \ldots, 1$, on affecte à $Q[k]$ la valeur

$$\frac{(x - x_k)Q[k-1] - (x - x_{k-1})Q[k]}{x_{k-1} - x_k}$$

qui n'est autre que $P_{(k-1)k}(x)$.

- Deuxième itération : pour $k = n, n-1, \ldots, 2$, on affecte à $Q[k]$ la valeur

$$\frac{(x - x_k)Q[k-1] - (x - x_{k-2})Q[k]}{x_{k-2} - x_k}$$

c'est à dire

$$\frac{(x - x_k)P_{(k-2)(k-1)}(x) - (x - x_{k-2})P_{(k-1)k}(x)}{x_{k-2} - x_k}$$

qui n'est autre que $P_{(k-2)(k-1)k}(x)$.

- Troisième itération : pour $k = n, n-1, \ldots, 3$, on affecte à $Q[k]$ la valeur $P_{(k-3)(k-2)(k-1)k}(x)$.

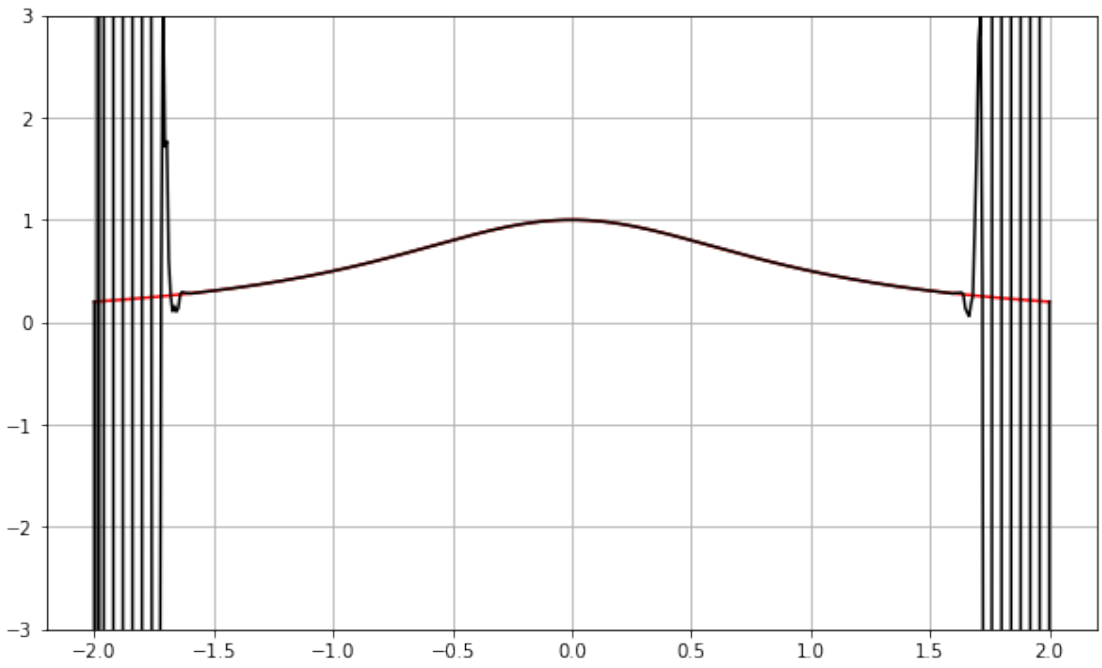
- Et ainsi de suite.

À la sortie de la boucle, $Q[n]$ contient $P_{0\ldots n}(x) = P(x)$. On le renvoie.
Remarque : La complexité de la fonction `neville` est clairement en $O(n^2)$.

```
In [22]: def neville(f, xs, x):
             n = len(xs) - 1
             Q = (n + 1) * [0]
             for j in range(n + 1): Q[j] = f(xs[j])
             for j in range(1, n + 1):
                 for k in range(n, j - 1, -1):
                     Q[k] = ((x - xs[k]) * Q[k - 1] - (x - xs[k - j]) * Q[k]) / (xs[k - j] - xs[
             return Q[n]
```

Nous pouvons maintenant tenter d'interpoler en un nombre de points plus important qu'au paragraphe précédent. Soyons toutefois raisonnables, une complexité en $O(n^2)$ nous invite à la modestie :-).

```
In [23]: ts = subdi(-2, 2, 100)
         xs = subdi(-2, 2, 500)
         ys = [exemple(t) for t in xs]
         zs = [neville(exemple, ts, x) for x in xs]
         plt.ylim(-3, 3)
         plt.plot(xs, ys, 'r')
         plt.plot(xs, zs,'k')
         plt.grid()
         plt.show()
```



Aux bords de l'intervalle, les problèmes s'aggravent lorsqu'on augmente le nombre de points. Si vous n'êtes pas convaincus, enlevez dans la cellule ci-dessus la ligne `plt.ylim(-3, 3)`.

1.4 3. L'erreur commise

Nous supposons dans cette partie que l'intervalle I de définition de f est un **segment**, et que f est de classe $\mathcal{C}^{n+1}$ sur I. La fonction $|f^{(n+1)}|$ est continue sur le segment I, elle y possède donc un maximum que nous noterons M_{n+1}.

Rappelons que $n \in \mathbb{N}$, $x_0 < x_1 < \ldots < x_n$ sont $n+1$ points de I, et P est le polynôme d'interpolation de f aux points x_i.

Notons enfin $Q = (X - x_0)(X - x_1)\ldots(X - x_n)$. Le polynôme Q est unitaire, de degré $n+1$ et s'ennule en tous les x_i.

1.4.1 3.1 Une majoration de l'erreur

Soit $x \in I$ différent des x_i. Considérons la fonction $\varphi : I \to \mathbb{R}$, de classe $\mathcal{C}^{n+1}$, définie pour $t \in I$ par

$$\varphi(t) = f(t) - P(t) - \frac{f(x) - P(x)}{Q(x)} Q(t)$$

On a pour tout $i \in [0, n]$, $\varphi(x_i) = 0$, et également $\varphi(x) = 0$. La fonction φ s'annule donc au moins $n+2$ fois sur I. De plus, φ est dérivable sur I. Par le théorème de Rolle φ' s'annule au moins $n+1$ fois sur I. On recommence : φ'' s'annule au moins n fois sur I, …, $\varphi^{(n+1)}$ s'annule au moins 1 fois sur I en un point que nous noterons ξ.

Comme le polynôme P est de degré inférieur ou égal à n, on a $P^{(n+1)} = 0$. De là,

$$0 = \varphi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \frac{f(x) - P(x)}{Q(x)} Q^{(n+1)}(\xi) = f^{(n+1)}(\xi) - (n+1)! \frac{f(x) - P(x)}{Q(x)}$$

En effet, le polynôme Q est unitaire, de degré $n+1$. Sa dérivée d'ordre $n+1$ est donc constante, égale à $(n+1)!$.

On déduit de l'égalité précédente

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} Q(x)$$

Proposition : Pour tout $x \in I$,

$$|f(x) - P(x)| \leq \frac{M_{n+1}}{(n+1)!} |Q(x)|$$

Démonstration : On vient de le faire si x est différent des x_i, et c'est évident si x est l'un des x_i puisque dans ce cas les deux côtés de l'inégalité sont nuls.

La majoration que nous venons d'obtenir est intéressante. Le facteur $(n+1)!$ au dénominateur est prometteur, puisqu'il tend très vite vers $+\infty$ lorsque n augmente. Le facteur M_{n+1} ne dépend que de f et pas des x_i. Et le facteur $|Q(x)|$ ne dépend que des x_i et de x, et pas de f.

1.4.2 3.2 La valeur de M_{n+1} pour notre exemple préféré

Revenons à $f : x \mapsto \frac{1}{x^2+1}$, définie sur le segment $[-2, 2]$. Pour tout $x \in I$,

$$f(x) = \frac{1}{2i} \left(\frac{1}{x - i} - \frac{1}{x + i} \right)$$

On vérifie facilement par récurrence sur n que pour tout entier naturel n,

$$f^{(n)}(x) = \frac{(-1)^n n!}{2i}\left(\frac{1}{(x-i)^{n+1}} - \frac{1}{(x+i)^{n+1}}\right)$$

On a $|x-i| = \sqrt{x^2+1} \geq 1$ et de même $|x+i| \geq 1$. On en déduit que le module de la grosse parenthèse est au plus égal à 2, et donc

$$|f^{(n)}(x)| \leq n!$$

Conclusion peu optimiste : $M_{n+1} \leq (n+1)!$.
Que vaut $f^{(n)}(0)$? Eh bien,

$$f^{(n)}(0) = \frac{(-1)^n n!}{2i}\left(\frac{1}{(-i)^{n+1}} - \frac{1}{i^{n+1}}\right)$$

Lorsque $n = 2p$ est impair, cela donne

$$f^{(n)}(0) = (-1)^{p+1} n!$$

et donc $M_n \geq n!$.
Conclusion pessimiste : Si n est impair, $M_{n+1} = (n+1)!$.
Cette valeur de M_{n+1} annihile évidemment tous les bienfaits du facteur $(n+1)!$ au dénominateur dans le majorant de l'erreur commise. Pour la fonction f de notre exemple, nous avons $|f(x) - P(x)| \leq |Q(x)|$, et pas mieux. En tout cas pas mieux de la façon dont nous avons réalisé notre majoration !

1.4.3 3.3 Une majoration de $|Q(x)|$

Comment majorer $|Q(x)|$? Soit h le *pas* de la subdivision $(x_0, \ldots, x_n)$, c'est à dire la plus grande des différences $x_{i+1} - x_i$, $i = 0, \ldots, n-1$. Soit j tel que $x_j \leq x \leq x_{j+1}$. On a

$$Q(x) = (x-x_0)\ldots(x-x_{j-1})(x-x_j)(x_{j+1}-x)(x_{j+2}-x)\ldots(x_n-x)$$

Maintenant, faites un petit dessin ... On a $x - x_0 \leq (j+1)h$, $x - x_1 \leq jh$, ..., $x - x_{j-1} \leq 2h$. De même $x_{j+2} - x \leq 2h$, ..., $x_n - x \leq (n-j)h$. De là,

$$Q(x) \leq h^{n-1}(x-x_j)(x_{j+1}-x)(j+1)!(n-j)!$$

Petite astuce maintenant : $\binom{n+1}{j+1} \geq n+1$, donc $(j+1)!(n-j)! \leq n!$. Enfin, une petite étude de fonction montre que

$$(x-x_j)(x_{j+1}-x) \leq \frac{1}{4}(x_{j+1}-x_j)^2 \leq \frac{1}{4}h^2$$

Ainsi,

$$|Q(x)| \leq \frac{h^{n+1}n!}{4}$$

Corollaire :

$$|f(x) - P(x)| \leq \frac{h^{n+1}M_{n+1}}{4(n+1)}$$

1.4.4 3.4 Subdivisions régulières

En prenant une subdivision régulière du segment $I = [a, b]$, on a $h = \frac{b-a}{n}$, et donc

$$|f(x) - P(x)| \leq \frac{(b-a)^{n+1} M_{n+1}}{4n^{n+1}(n+1)}$$

Pour notre fonction d'exemple $x \mapsto \frac{1}{x^2+1}$ sur le segment $[-2, 2]$, avec n impair, nous avons $b - a = 4$, $M_{n+1} = (n+1)!$, et donc

$$|f(x) - P(x)| \leq \frac{4^n n!}{n^{n+1}}$$

Une petite application de la formule de Stirling permet de trouver un équivalent de notre majorant :

$$\frac{4^n n!}{n^{n+1}} \sim \left(\frac{4}{e}\right)^n \sqrt{\frac{2\pi}{n}}$$

Lorsque n tend vers l'infini, comme $e < 4$, le majorant tend vers ... $+\infty$:-(.

Comment peut-on espérer faire mieux ? La fonction f étant ce qu'elle est, nous ne pouvons rien faire pour la valeur de M_{n+1}. En revanche, nous pouvons agir sur le polynôme Q. Pouvons nous trouver un polynôme Q unitaire de degré $n+1$ tel que $\max_{[a,b]} |Q|$ soit minimal parmi tous les polynômes unitaires de degré $n+1$? Oui, et c'est ce que nous allons maintenant étudier.

1.5 4. Tchebychev

1.5.1 4.1 Les polynômes de Tchebychev

Pour tout $n \in \mathbb{N}$, considérons la fonction $T_n : [-1, 1] \to \mathbb{R}$ définie par $T_n(x) = \cos(n \arccos x)$. Clairement, pour tout $x \in [-1, 1]$, $|T_n(x)| \leq 1$.

Exercice : Montrer que pour tout $x \in [-1, 1]$, $T_0(x) = 1$, $T_1(x) = x$, et pour tout $n \geq 0$, $T_{n+2}(x) = 2xT_{n+1}(x) - T_n(x)$. Indication : poser $x = \cos t$. Puis remarquer que $\cos(n+2)t + \cos(nt) = 2\cos t \cos(n+1)t$.

Conséquence : T_n est la restriction à $[-1, 1]$ d'une fonction polynomiale que nous continuerons à appeler T_n. Les polynômes T_n sont donc définis par la récurrence :

- $T_0 = 0$, $T_1 = X$.
- Pour tout $n \in \mathbb{N}$, $T_{n+2} = 2XT_{n+1} - T_n$.

On les appelle les **polynômes de Tchebychev de première espèce**.
Exercice : N'y a-t-il pas une question "d'unicité" de T_n à se poser ?
Exercice : Montrer :

- Pour tout $n \geq 0$, T_n est un polynôme de degré n.
- Pour tout $n \geq 1$, le coefficient dominant de T_n est 2^{n-1}.

La fonction `tchebychev` renvoie le $n$ième polynôme de Tchebychev.

```
In [27]: def tchebychev(n, x):
             a = 1
             b = x
```

```
            for k in range(n):
                a, b = b, 2 * x * b - a
            return a
```

In [28]:
```
init_printing(pretty_print=True)
```

In [29]:
```
tchebs = [expand(tchebychev(k, x)) for k in range(8)]
tchebs
```

Out[29]:

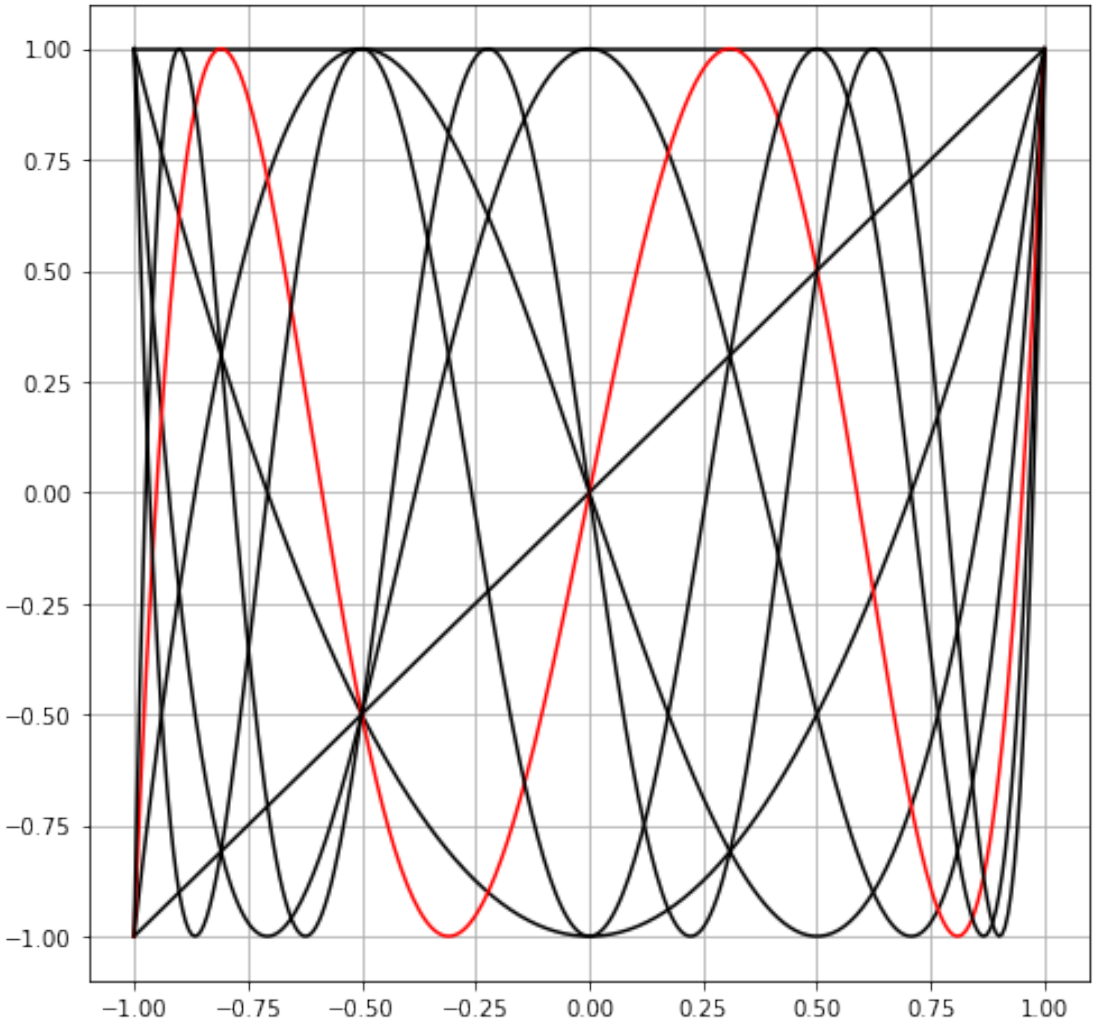
$$\left[1, \quad x, \quad 2x^2 - 1, \quad 4x^3 - 3x, \quad 8x^4 - 8x^2 + 1, \quad 16x^5 - 20x^3 + 5x, \quad 32x^6 - 48x^4 + 18x^2 - 1, \quad 64x^7 - 112x^5 + 56\right.$$

In [30]:
```
init_printing(pretty_print=False)
```

Traçons quelques courbes ...

In [31]:
```
plt.rcParams['figure.figsize'] = (8, 8)
```

In [32]:
```
for k in range(len(tchebs)):
    if k != 5: tracer_poly(tchebs[k], -1., 1., 'k')
    else: tracer_poly(tchebs[k], -1., 1., 'r')
plt.grid()
plt.show()
```




```
In [33]: plt.rcParams['figure.figsize'] = (10, 6)
```

Le polynôme T_5 est tracé en rouge sur la figure. Combien a-t-il de racines dans l'intervalle $[-1, 1]$? Apparemment 5. Plus généralement, quelles sont les racines de T_n ?

Proposition : Pour tout $n \geq 1$, les racines de T_n sont les réels $x_k = \cos(\frac{\pi}{2n} + \frac{k\pi}{n})$, $k = 0, \ldots, n-1$.

Démonstration : Il est évident que ces réels sont des racines distinctes de T_n. Comme T_n est de degré n, ce sont les seules racines.

1.5.2 4.2 Une propriété de minimalité

Soit $Q = \frac{1}{2^n} T_{n+1}$. Le polynôme Q est unitaire, de degré $n+1$, et on a

$$\forall x \in [-1, 1], |Q(x)| \leq \frac{1}{2^n}$$

De plus, $Q(1) = \frac{1}{2^n} T_{n+1}(1) = \frac{1}{2^n} \cos(n \arccos 1) = \frac{1}{2^n}$. Ainsi,

$$\max_{[-1,1]} |Q| = \frac{1}{2^n}$$

Proposition : Soit P un polynôme unitaire de degré $n+1$. Supposons

$$\max_{[-1,1]} |P| \leq \frac{1}{2^n}$$

Alors $P = Q$.

Démonstration : Soit P un polynôme unitaire de degré $n+1$. Supposons que pour tout $x \in [-1,1]$ on ait $|P(x)| \leq \frac{1}{2^n}$. Posons $R = P - Q$. Le polynôme R est de degré inférieur ou égal à n, car P et Q sont unitaires et de degré $n+1$.

Pour $k = 0, 1, \ldots, n+1$, soit $x_k = \cos \frac{k\pi}{n+1}$. On a $T_{n+1}(x_k) = \cos(k\pi) = (-1)^k$. De là, $R(x_k) = P(x_k) - \frac{(-1)^k}{2^n}$. Comme $|P(x_k)| \leq \frac{1}{2^n}$, il s'ensuit que les réels $R(x_k)$ sont positifs si k est impair et négatifs si k est pair. Par le théorème des valeurs intermédiaires, le polynôme R s'annule au moins $n+1$ fois. Mais le degré de R est inférieur ou égal à n, donc $R = 0$ et ainsi $P = Q$.

Exercice : J'ai un peu triché. Il se pourrait que deux racines successives de R trouvées par le TVI soient égales. Comment rétablir la démonstration ? Indication : racine multiple.

Remarque : ce résultat montre que le polynôme Q est, parmi tous les polynômes unitaires de degré $n+1$, celui dont le maximum de la valeur absolue est minimal (si j'ose dire :-)).

1.5.3 4.3 Subdivisions de Tchebychev

Plaçons-nous sur un segment $I = [a, b]$. Soient $y_0, \ldots, y_n$ les racines de T_{n+1}.

Pour $i = 0, \ldots, n$, posons $x_i = m + \delta y_i$ où $m = \frac{a+b}{2}$ et $\delta = \frac{b-a}{2}$. On vérifie facilement que les x_i forment une subdivision de $[a, b]$.

Posons $Q = \frac{\delta^{n+1}}{2^n} T_{n+1}(\frac{X-m}{\delta})$. Le polynôme Q est unitaire, et ses racines sont les x_i. Ainsi, $Q = (x - x_0) \ldots (x - x_n)$. De plus, pour tout $x \in [a, b]$,

$$|Q(x)| \leq \frac{\delta^{n+1}}{2^n} = \frac{(b-a)^{n+1}}{2^{2n+1}}$$

et ainsi

$$|f(x) - P(x)| \leq \frac{(b-a)^{n+1} M_{n+1}}{2^{2n+1}(n+1)!}$$

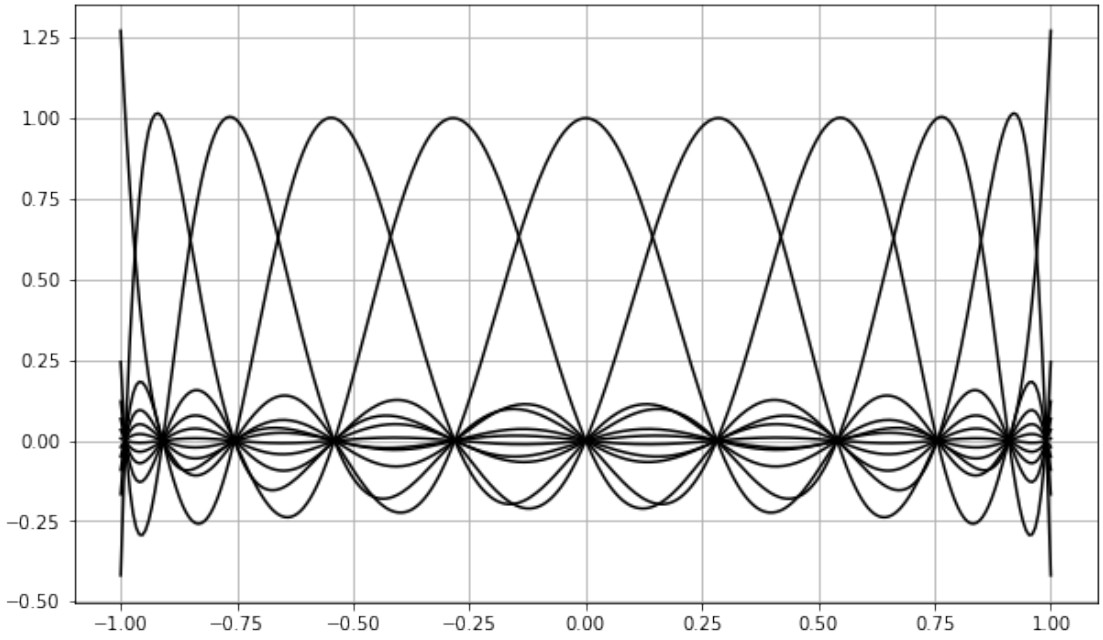
```
In [34]: def subdi_tchebychev(a, b, n):
             m = (a + b) / 2
             d = (b - a) / 2
             return [m - d * math.cos(math.pi / (n + 1) * (k + 1 / 2)) for k in range(n + 1)]

In [35]: subdi_tchebychev(-2, 2, 10)

Out[35]: [-1.9796428837618654, -1.8192639907090369, -1.5114991487085165, -1.0812816349111953, -0
```

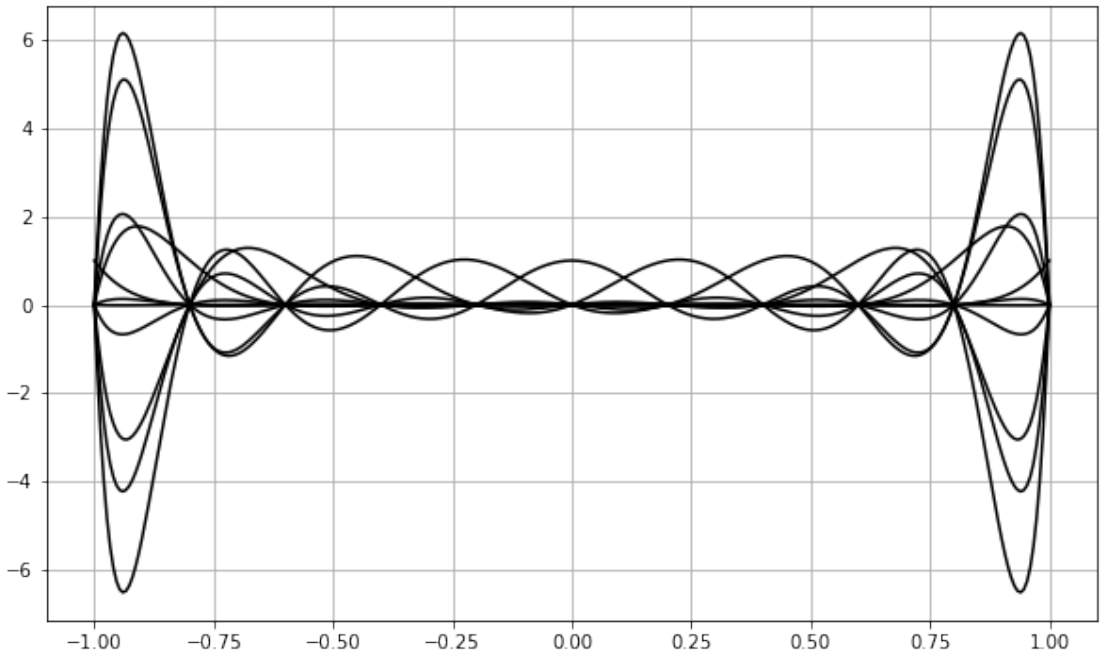
Voici les polynômes élémentaires de Lagrange pour une subdivision de Tchebychev.

```
In [36]: xs = subdi_tchebychev(-1, 1, 10)
         ps = [expand(lagrange(k, x, xs)) for k in range(len(xs))]
         for p in ps:
             tracer_poly(p, -1., 1., 'k')
         plt.grid()
         plt.show()
```



Vous souvenez-vous de ce que nous avions obtenu avec une subdivision régulière ? Le revoici.

```
In [37]: xs = subdi(-1, 1, 10)
         ps = [expand(lagrange(k, x, xs)) for k in range(len(xs))]
         for p in ps:
             tracer_poly(p, -1., 1., 'k')
         plt.grid()
         plt.show()
```



Je ne commenterai pas plus avant mais les polynômes associés à une subdivision de Tchebychev ont l'air "mieux" que ceux associés à une subdivision régulière, surtout aux bords de l'intervalle.

1.5.4 3.5 Retour à l'exemple

Que devient la majoration sur notre exemple ?

$$|f(x) - P(x)| \leq \frac{4^{n+1}(n+1)!}{2^{2n+1}(n+1)!} = 2$$

Notre majorant, égal à 2, ne tend toujours pas vers 0 lorsque n tend vers l'infini, mais il est au moins borné.

Remarque : Depuis le début de ce notebook, aucune majoration n'a été efficace pour notre fonction d'exemple :-).

Remarquons toutefois que si nous nous plaçons sur un segment $I = [a, b]$ quelconque, on a

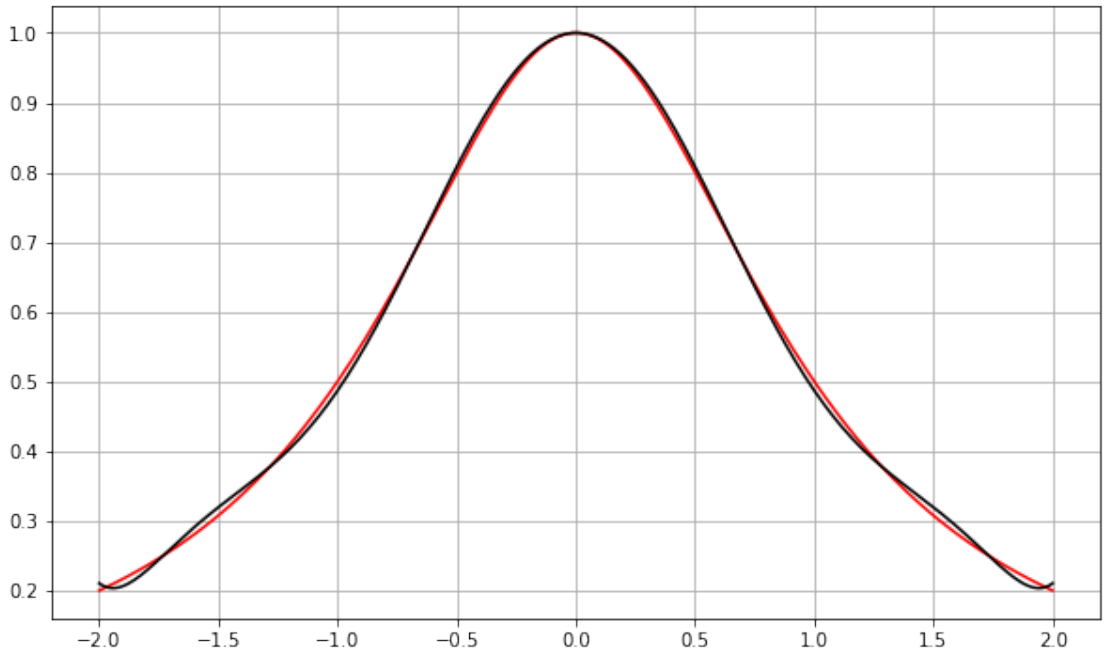
$$|f(x) - P(x)| \leq 2 \left(\frac{b-a}{4} \right)^{n+1}$$

Ce majorant tend vers 0 dès que $b - a < 4$. Nous avons donc montré la convergence uniforme des polynômes d'interpolation de notre exemple f vers la fonction f lorsque l'intervalle est de longueur strictement inférieure à 4. Par exemple, sur $[-1.9999, 1.9999]$. Mais pas sur $[-2, 2]$.

Peut-on faire mieux ? Oui, en fait il y a convergence uniforme sur $[-2, 2]$, mais ceci est une autre histoire.

Petite illustration pour terminer ce notebook, notre fonction `exemple` et son polynôme d'interpolation en des points de Tchebychev ...

In [38]:
```
ts = subdi_tchebychev(-2, 2, 8)
xs = subdi(-2, 2, 300)
ys = [exemple(t) for t in xs]
zs = [neville(exemple, ts, x) for x in xs]
plt.plot(xs, ys, 'r')
plt.plot(xs, zs, 'k')
plt.grid()
plt.show()
```



In [ ]: