Travaux dirigés Polynômes n°6

Mathématiques pour l'informatique

—IMAC 2—

▸ **Exercice 1.** ***Racines réelles d'un polynôme***

Un polynôme peut être représenté numériquement par un vecteur $\mathbf{p}$ *constitué de l'ensemble de ses coefficients. Ainsi* $\mathbf{p}_i$ *correspond au coefficient associé à* x^i*. Vous trouverez sur la page de l'enseignant une classe* `Polynôme` *permettant de définir et d'afficher un polynôme. Cette classe inclut également les opérateurs arithmétiques élémentaires sur les polynômes* $(+,-,\times)$ *nécessaires pour la suite de cet exercice.*

1. *Récupérez le code sur le site de l'enseignant, compilez, testez et lisez le code pour le comprendre.*

2. *Dans le fichier* `Polynomial.cpp`*, implantez l'opérateur* `operator()(const double &x)` *qui permet d'évaluer un polynôme* $p(x)$ *en un point* x_0*, avec par exemple la commande* `double val = p(3);`
   Pour cet exercice, vous pouvez coder la méthode de Horner vue en cours ou n'importe quelle autre méthode. Testez votre méthode en l'appelant dans le fichier `main.cpp`*.*

3. *Dans le fichier* `Polynomial.cpp`*, implantez une fonction* `polynomialFromRoot(const Eigen::VectorXd &roots)` *qui génère un polynôme dont les racines sont passées sous forme de paramètre dans un vecteur. Vérifiez qu'en évaluant votre polynôme sur ses racines, vous trouvez bien zéro.*

4. *Implantez une fonction* `findRoots(const unsigned int nbIter)` *qui calcule les racines réelles d'un polynôme en utilisant la décomposition RQ (ou LU, au choix). Pour rappel, étant donné un polynôme* $p(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$*, le polynôme unitaire* $p_u(x)$ *correspondant est*

$$p_u(x) = x^n + \cdots + \frac{a_2}{a_n}x^2 + \frac{a_1}{a_n}x + \frac{a_0}{a_n}$$

   Par ailleurs, la matrice compagnon C *du polynôme* $p(x)$ *est*

$$\mathrm{C} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -b_0 \\ 1 & 0 & 0 & \cdots & 0 & -b_1 \\ 0 & 1 & 0 & \cdots & 0 & -b_2 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -b_{n-2} \\ 0 & 0 & 0 & \cdots & 1 & -b_{n-1} \end{bmatrix}$$

   où les b_i *sont les coefficients du polynôme unitaire* $p_u(x)$ *associé à* $p(x)$*. Pour créer la matrice* C*, vous pouvez avoir besoin des fonctions* `Eigen` *suivantes :*

- *bottomLeftCorner*
- *setIdentity()*
- *rightCols*
- *head()*

Pour trouver les racines réelles de votre polynôme, vous devez ensuite itérer sur le processus suivant :

```
Repeat
  Q,R = decompositionQR(C)
  C = R*Q
Roots = C.diagonal()
```

La matrice C *convergera vers une matrice diagonale dont les éléments seront les racines réelles de votre polynôme. Pour coder cette fonction, vous pourrez avoir besoin des fonctions de décomposition QR suivantes :*

- `Eigen::HouseholderQR<Eigen::MatrixXd> qr(C);`
- `Eigen::MatrixXd Q = qr.householderQ();`
- `Eigen::MatrixXd R = qr.matrixQR().triangularView<Eigen::Upper>();`

Testez votre programme sur un polynôme dont vous avez choisi les racines avec la fonction `polynomialFromRoot` *de l'exercice précédent.*

5. *Que se passe-t-il pour les racines doubles?*

6. *Dans le fichier* `Polynomial.cpp`*, implantez une fonction* `derivative()` *qui renvoie le polynôme dérivée du polynôme appelant. Testez votre méthode dans le fichier* `main.cpp`*.*

7. *Améliorez la fonction* `findRoots(const unsigned int nbIter)` *en y ajoutant un raffinement non linéaire avec la méthode de Newton sur quelques itérations. Pour rappel, la méthode de Newton permet de résoudre $f(x) = 0$ en partant d'une bonne estimation x_0 de la solution. Plus précisément, la méthode de Newton consiste à itérer la formule suivante :*

$$x^{n+1} = x^n - \frac{f(x_n)}{f'(x_n)}$$