Exercice 1 :

(1) Rappeler une équation polynomiale qui admet comme solution $\sqrt{2}$

(2) Écrire une fonction **dicho(n,a,b)** qui retourne le couple $(a_n, b_n)$ obtenue par l'algorithme de dichotomie avec $a_0 = a$ et $b_0 = b$.

(3) Écrire une fonction **newton(n,x)** qui calcule $x_n$ avec l'algorithme de Newton en prenant $x_0 = 2$.

(4) Tester (sans les implémenter, mais on pourra s'aider d'une calculatrice) ces algorithmes avec $n = 3$, $a = 1$, $b = 2$ et $x = 2$. Lequel est le plus rapide ?

Exercice 2 :

La méthode d'Héron est une méthode efficace d'extraction de racine carrée.
Elle porte le nom d'Héron (1 siècle après JC) mais certains calculs antérieurs, notamment Égyptien, semblent montrer que la méthode est plus ancienne.

L'idée est que la suite récurrente définie pour tout entier $n$ par $u_{n+1} = \frac{1}{2}\left(u_n + \frac{K}{u_n}\right)$ et $u_0 > 0$ converge vers $\sqrt{K}$ où $K$ est un réel positif.

(1) Expliquer en quoi la méthode de Héron est un cas particulier de la méthode de Newton.

(2) Donner une suite qui converge vers $\sqrt[n]{K}$, la racine nième de $K$.

Exercice 3 :

Écrire un algorithme qui calcule une approximation de $\Pi$ à partir de la fonction sin.
Programmer cet algorithme.

Exercice 4 :

Si on applique la méthode de Newton à la fonction $x \mapsto ax - 1$ (avec $a > 0$), on peut espérer approcher l'inverse de $a$.

(1) Expliciter la relation de récurrence mise en place par la relation de Newton... et constater qu'elle est inutilisable.

(2) Montrer qu'en utilisant la méthode de Newton avec $x \mapsto \frac{1}{x} - a$, on corrige ce problème.

(3) En étudiant la courbe de la fonction précédente, conjecturer des valeurs $x_0$ pour lesquelles cette méthode fonctionne.

(4) Programmer cette méthode et déterminer les 5 premiers termes de la suite, lorsque $a = 3$ et $x_0 \in \{-0.1; 0; 0.1; 0.3, 0.7, 1.2\}$.

Exercice 5 :

Dans cet exercice, on utilisera les fonctions (**Horner** et **derivee**) du dernier TP sur les polynômes.

(1) Programmer la fonction **dicho(P,a,b,e)** qui prend en entrée un polynôme $P$, deux entiers $a$ et $b$ et une erreur $e$ qui renvoie un encadrement $[a_n, b_n]$ d'une des solutions de $P(x) = 0$ tels que $|b_n - a_n| < e$.

(2) Programmer la fonction **newton(P,a,e)** qui prend en entrée un polynôme $P$, un entier $a$ et une erreur $e$ qui renvoie une valeur approchée à $e$ près d'une des solutions de $P(x) = 0$.

(3) Tester les deux fonctions (on pourra reprendre $x^3 + 3x^2 + 2x - 1$ avec $x_0 = 1$ et d'autres polynômes), vérifier les solutions avec la résolution polynomiale de **numpy** (Dans **numpy** les polynômes ne sont pas dans le même ordre, il faut inverser la liste avec $L.reverse()$).

(4) Dans chacun des algorithmes, mettre des compteurs pour compter le nombre de boucles à faire pour exécuter l'algorithme. Le résultat-il cohérent ?

(5) Programmer la fonction **tangente(P,a,e)** qui prend en entrée un polynôme $P$, un entier $a$ et une erreur $e$ qui renvoie une valeur approchée à $e$ près d'une des solutions de $P(x) = 0$ puis refaire les tests pour le comparer aux autres algorithmes

Exercice 6 :

**Dans cet exercice, on reprend les algorithmes du TP sur les matrices.** On suppose maintenant que $A$ est une matrice et on considère une suite de matrice $(X_n)_{n\in\mathbb{N}}$. Pour inverser la matrice, on utilise la méthode de Newton appliquée à la fonction $f(X) = A - X^{-1}$. On considère ainsi une suite de matrice $(X_n)_{n\in\mathbb{N}}$ vérifiant la relation $X_{n+1} = X_n(2In - AX_n)$.

(1) Expliquer la relation entre cette suite et la méthode de Newton.

(2) Un bon choix de valeur de départ est $X_0 = \alpha_0 A^t$, avec $\alpha_0 = \dfrac{1}{\|A\|_1 \|A\|_\infty}$, où $\|A\|_1 = \max_j \sum_i |a_{i,j}|$ et $\|A|_\infty = \max_i \sum_j |ai, j|$

a. Écrire la fonction **somme_colonne(A,j)** qui fait la somme des valeurs absolues des termes de la colonne $j$.

b. En déduire la fonction **norme1(A)** qui renvoie le maximum de ces sommes.

c. Écrire de même la fonction **normeinfini(A)** qui renvoie le maximum des sommes des lignes.

(3) Programmer la fonction **inversion_matrice_Newton(A)** qui inverse une matrice $A$ à l'aide de cet algorithme.