Mathématiques

IUT INFO 2 / 2012-2013

MAJ: 18 mars 2013

Mathématiques générales Module S4

Guillaume CONNAN

UNIVERSITÉ DE NANTES

IUT NANTES

TABLE DES MATIÈRES

TD N° 1

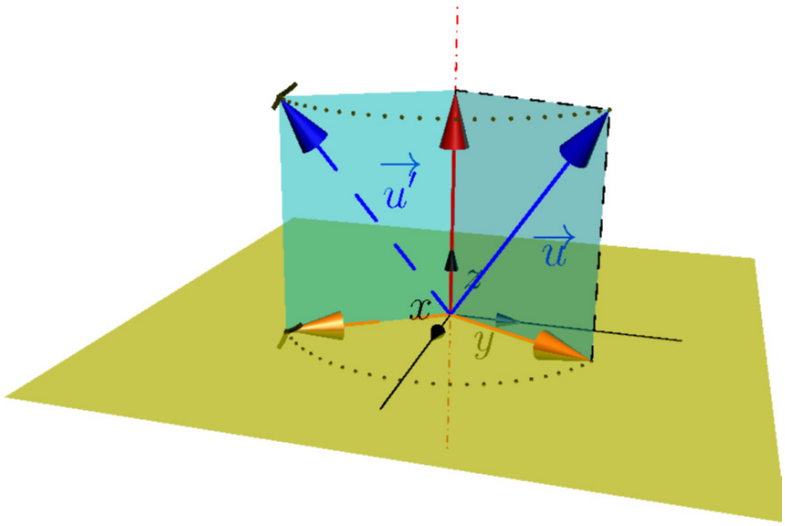
Algèbre : niveau 2

L'étude des rotations vectorielles prolonge notre travail sur le calcul matriciel et les espaces vectoriels. C'est un outil indispensable pour l'informatique et les mathématiques qui permettent de modéliser le mouvement. Les projections sont elles utilisées de manière beaucoup plus générale, par exemple dans le traitement des images qui sera notre champ d'action. Nous étudierons un outil algébrique de compression d'image qui nous fera rentrer dans le monde merveilleux de la réduction d'endomorphismes...

1 Rotations

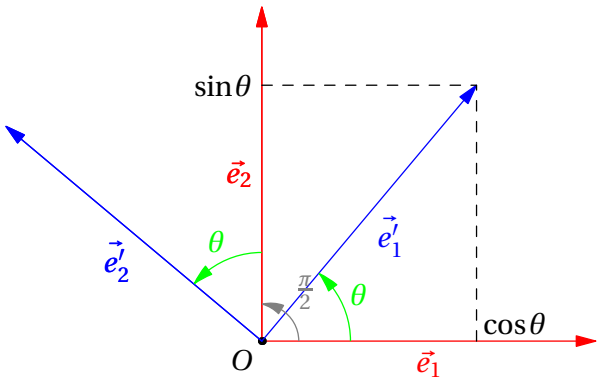
1.1 Approche intuitive

Une rotation vectorielle est une application linéaire particulière. Elle sert à décrire le mouvement qui correspond à ce que vous avez étudié dans le secondaire dans le plan affine mais le généralise en dimension quelconque et surtout, les objets « déplacés » sont des vecteurs, ce qui est un peu plus abstrait que les points ou les solides...
Plus étrange encore, nous ne définirons les rotations vectorielles qu'à la fin de ce chapitre. Nous partirons d'abord d'une approche intuitive.
Nous travaillerons la plupart du temps dans $\mathbb{R}^3$, l'espace usuel de dimension 3. Une rotation sera alors caractérisée par un axe et un angle « autour » duquel les vecteurs « tournent »...
Dans un premier temps, nous étudierons une rotation d'axe $\vec{e_3}$.



Rotation d'un vecteur

Nous avons vu au module précédent que pour caractériser une application linéaire (ici une rotation), il suffit de déterminer les images de chaque vecteur de base.
On observe que e_3 est invariant car il a la même direction que l'axe de rotation.
Il suffit donc de se placer dans le plan (e_1, e_2) et de déterminer les images des deux vecteurs de base :



Projection orthogonale de la rotation d'un vecteur

Si vous avez quelques souvenirs trigonométriques de collège ou de lycée, vous ne serez pas surpris par les résultats suivants :

$$\vec{e'_1} = \cos\theta\vec{e_1} + \sin\theta\vec{e_2}$$
$$\vec{e'_2} = \cos\left(\theta + \tfrac{\pi}{2}\right)\vec{e_1} + \sin\left(\theta + \tfrac{\pi}{2}\right)\vec{e_2} = -\sin\theta\vec{e_1} + \cos\theta\vec{e_2}$$

Ce qui donne pour notre matrice, en se souvenant que $\overrightarrow{e'_3} = e_3$:

$$\begin{matrix} r(\overrightarrow{e_1}) & r(\overrightarrow{e_2}) & r(\overrightarrow{e_3}) & \\ \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} & & & \begin{matrix} \overrightarrow{e_1} \\ \overrightarrow{e_2} \\ \overrightarrow{e_3} \end{matrix} \end{matrix}$$

1 2 Détermination de l'image d'un vecteur

Soit $\overrightarrow{u}$ un vecteur de coordonnées (x, y, z). Cela signifie que $\overrightarrow{u} = x\overrightarrow{e_1} + y\overrightarrow{e_2} + z\overrightarrow{e_3}$.
Soit r la rotation d'axe $\overrightarrow{e_3}$ et d'angle θ.
On obtient alors que :

$$\begin{aligned} r(\overrightarrow{u}) &= xr(\overrightarrow{e_1}) + yr(\overrightarrow{e_2}) + zr(\overrightarrow{e_3}) \\ &= x\big(\cos(\theta)\overrightarrow{e_1} + \sin(\theta)\overrightarrow{e_2}\big) + y\big(-\sin(\theta)\overrightarrow{e_1} + \cos(\theta)\overrightarrow{e_2}\big) + z\overrightarrow{e_3} \\ &= (x\cos(\theta) - y\sin(\theta))\overrightarrow{e_1} + (x\sin(\theta) + y\cos(\theta))\overrightarrow{e_2} + z\overrightarrow{e_3} \end{aligned}$$

Notons R la matrice de r dans la base canonique et U la matrice colonne des coordonnées de $\overrightarrow{u}$ dans la base canonique. Calculons R × U :

$$\begin{matrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} x \\ y \\ z \end{pmatrix} & = & \begin{pmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ z \end{pmatrix} \\ \boxed{\text{R}} & \boxed{\text{U}} & & \boxed{\text{R} \times \text{U}} \end{matrix}$$

Miracle ! R × U est en fait la matrice colonne des coordonnées de $r(\overrightarrow{u})$. Un problème de transformation géométrique se règle par de simples multiplications : la puissance des mathématiques en marche grâce à une notion fondamentale, l'*isomorphie*. On trouve un « corps mathématiques » tout à fait similaire à celui étudié mais dans lequel il est plus aisé de travailler...

Remarque

Espaces isomorphes
Ainsi, il existe de telles *isomorphismes* entre l'Espace géométrique de dimension 3, l'ensemble $\mathbb{R}^3$ de leurs coordonnées et $\mathfrak{M}_{3,1}(\mathbb{R})$, l'ensemble des matrices à coefficients réels ayant trois lignes et une colonne. On parlera souvent de l'un pour l'autre.

1 3 Généralisation

On peut alors admettre la généralisation suivante :

Propriété 1 - 1

Expression matricielle de l'image d'un vecteur
Soit M la matrice d'un déplacement dans une certaine base $\mathcal{B}$ de $\mathbb{R}^3$.
Soit U la matrice colonne des coordonnées d'un certain vecteur $\overrightarrow{u}$ dans cette base.
Alors la matrice colonne des coordonnées de $f(\overrightarrow{u})$ dans cette base vaut :

$$\text{mat}_{\mathcal{B}} f(\overrightarrow{u}) = \text{M} \times \text{U}$$

1 4 Matrice d'une composée de déplacements

On est souvent amené à décomposer le mouvement d'un système en mouvements simples. Pour obtenir les coordonnées finales à partir des initiales, il faut ensuite recomposer les mouvements.
Étudions le cas de la composée de deux déplacements.

Remarque

composée d'applications
On note $\varphi \circ g$ l'application composée de g suivie de f (attention à l'ordre !).
Ainsi, $\varphi \circ g(x) = f\big(g(x)\big)$: on calcule $g(x)$ puis on calcule l'image par f du vecteur obtenu.

Considérons deux déplacements f et g. Notons $\mathrm{M} = \mathrm{mat}_{\mathcal{B}} f$, $\mathrm{N} = \mathrm{mat}_{\mathcal{B}} g$ et $\mathrm{U} = \mathrm{mat}_{\mathcal{B}} \vec{u}$. Or

$$f \circ g(\vec{u}) = f\Big(g(\vec{u})\Big)$$

donc, point de vue matrice :

$$\mathrm{mat}_{\mathcal{B}} f \circ g(\vec{u}) = \mathrm{M} \times (\mathrm{N} \times \mathrm{U}) = (\mathrm{M} \times \mathrm{N}) \times \mathrm{U}$$

On en déduit la propriété suivante :

Propriété 1 - 2

Expression matricielle de la composée de deux applications linéaires
Soit M la matrice d'un déplacement dans une certaine base $\mathcal{B}$ de $\mathbb{R}^3$.
Soit N la matrice d'un déplacement dans la même base.
Alors :

$$\mathrm{mat}_{\mathcal{B}} f \circ g = \mathrm{M} \times \mathrm{N}$$

En particulier, avec les notations habituelles,

$$\mathrm{mat}_{\mathcal{B}} f \circ g(\vec{u}) = (\mathrm{M} \times \mathrm{N}) \times \mathrm{U}$$

2 Produit scalaire - Matrices orthogonales

2 1 Produit scalaire de deux vecteurs

Nous rappellerons juste la définition vue en Terminale :

Définition 1 - 1

produit scalaire
Dans un repère orthonormé de $\mathbb{R}^3$, deux vecteurs $\vec{v}$ et $\vec{v'}$ ont pour coordonnées respectives (x, y, z) et (x', y', z').
On appelle *produit scalaire* de $\vec{v}$ et $\vec{v'}$ et on note $\left\langle \vec{v}, \vec{v'} \right\rangle$ le nombre réel :

$$\left\langle \vec{v}, \vec{v'} \right\rangle = xx' + yy' + zz'$$

2 2 Représentation matricielle

Nous aurons besoin pour le reste de notre étude la *transposée* d'une matrice.

Définition 1 - 2

matrice transposée
On appelle transposée de la matrice A et on note ${}^{\mathrm{t}}\mathrm{A}$ la matrice obtenue en échangeant lignes et colonnes.

Par exemple, si $\mathrm{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, alors ${}^{\mathrm{t}}\mathrm{A} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$
Soit $\vec{v}$ et $\vec{v'}$ deux vecteurs de représentations matricielles respectives V et V′ dans une certaine base orthonormée.
Pour obtenir matriciellement le produit scalaire des deux vecteurs, on effectue le produit $\mathrm{V} \times {}^{\mathrm{t}}\mathrm{V}'$.
Nous aurons besoin enfin de cette importante définition :

Définition 1 - 3

vecteurs orthogonaux
Deux vecteurs sont orthogonaux si, et seulement si, leur produit scalaire est nul.

2 3 Norme d'un vecteur

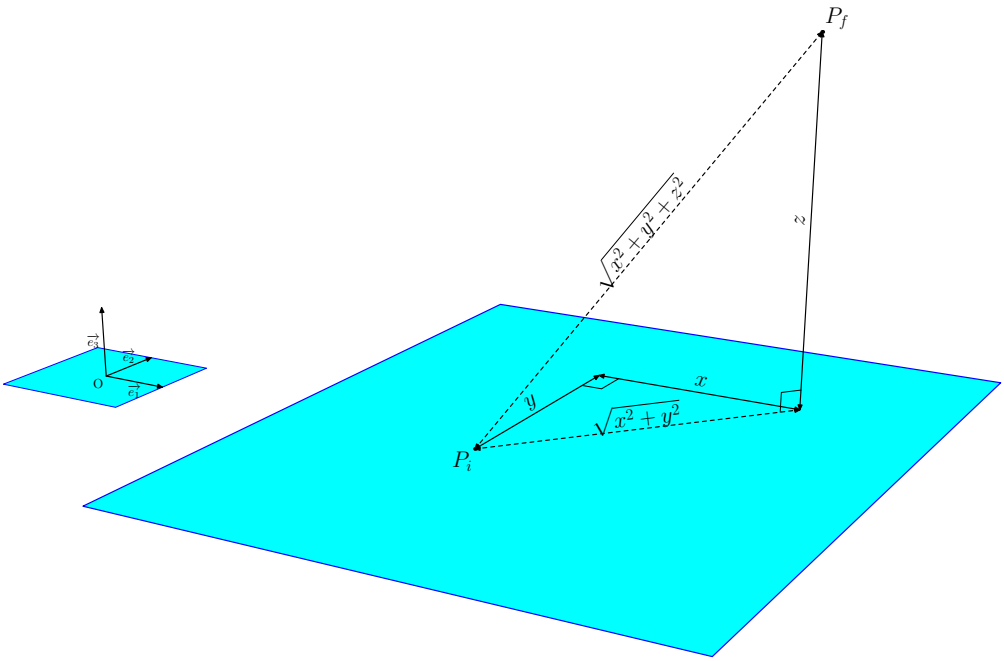
Il s'agit tout simplement de la racine carrée du produit scalaire d'un vecteur par lui-même.

Définition 1 - 4

norme d'un vecteur
Soit $\vec{v}$ un vecteur. On appelle norme de $\vec{v}$ le nombre noté $\left\| \vec{v} \right\|$ et défini par :

$$\left\| \vec{v} \right\| = \sqrt{\left\langle \vec{v}, \vec{v} \right\rangle}$$

Ainsi, si $\vec{v}$ a pour coordonnées (x, y, z) dans un repère orthonormé, alors $\|\vec{v}\| = \sqrt{x^2+y^2+z^2}$. Cela vous rappelle sûrement un vieux théorème de collège...



norme d'un vecteur

2 4 Matrices orthogonales

Nous allons enfin pouvoir revenir à nos transformations de l'espace. Il nous faut d'abord donner une nouvelle définition...

Définition 1 - 5

matrice orthogonale
Une matrice de $\mathfrak{M}_3(\mathbb{R})$ est orthogonale si, et seulement si, elle est inversible et $A^{-1} = {}^tA$

Voici qui est bien pratique : pour inverser ce genre de matrices, il suffit de les transposer. Il nous reste à découvrir qui sont vraiment ces matrices...

Théorème 1 - 1

caractérisation d'une matrice orthogonale
Une matrice de $\mathfrak{M}_3(\mathbb{R})$ est orthogonale si, et seulement si, ses colonnes forment une base orthonormée de $\mathbb{R}^3$.

Une nouvelle fois, nous ne démontrerons pas ce théorème (car nous ne sommes pas vraiment mathématiciens...).
Utilisons-le cependant dès maintenant. Reprenons la matrice d'une rotation d'angle θ et d'axe (Oz) et appelons C_1, C_2 et C_3 ses colonnes :

$$R = \begin{array}{c} \begin{matrix} C_1 & C_2 & C_3 \end{matrix} \\ \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Alors la norme du vecteur associé à C_1 vaut $\sqrt{\cos^2\theta + \sin^2\theta + 0^2} = 1$. Il en va de même pour C_2 et C_3.
Formons à présent les produits scalaires :

$$C_1 \times {}^tC_2 = -\cos\theta\sin\theta + \sin\theta\cos\theta + 0 = 0$$

Il en va de même pour C_1 et C_3 ainsi que pour C_2 et C_3.
La matrice de cette rotation est donc orthogonale.

2 5 Quelques propriétés des matrices orthogonales

Nous allons d'abord donner, une nouvelle fois sans preuve, deux propriétés importantes des déterminants :

Propriété 1 - 3

déterminant d'une transposée et d'un produit

$$\det {}^t A = \det A$$

$$\det A \times B = \det A \cdot \det B$$

cela va nous permettre d'établir le théorème suivant :

Théorème 1 - 2

déterminant d'une matrice orthogonale
Une matrice orthogonale a pour déterminant 1 ou -1

Allez, pour le plaisir, au moins une preuve dans ce cours....
On a d'une part $\det(A \times {}^t A) = \det A \cdot \det {}^t A = (\det A)^2$.
D'autre part $\det(A \times {}^t A) = \det(I_3) = 1$.
Finalement, $(\det A)^2 = 1$, c'est-à-dire $\det A = \pm 1$.
Bon, ce n'était pas méchant...

Théorème 1 - 3

Détermination de l'axe d'une rotation
Si A est une matrice orthogonale de $\mathfrak{M}_3(\mathbb{R})$ différente de l'identité et dont le déterminant vaut 1, alors A est la matrice d'une rotation autour d'un axe.
La direction de l'axe est donnée en déterminant un vecteur invariant.

Voilà pour l'axe. En ce qui concerne l'angle, nous aurons besoin...d'une nouvelle définition !

Définition 1 - 6

trace d'une matrice carrée
La trace d'une matrice, notée tr(A) est la somme des éléments de sa diagonale.

Par exemple, la trace de la matrice $\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$ est $2\cos\theta + 1$.

Or il existe une belle propriété qui dit que la trace ne change pas lorsqu'on effectue des changements de base et donc...

Théorème 1 - 4

détermination du cosinus de l'angle d'une rotation
Soit r une rotation de matrice A dans une certaine base. Alors l'angle θ de la rotation vérifie :

$$\cos\theta = \frac{\mathrm{tr}(A) - 1}{2}$$

ce qui est « magique », c'est qu'on va ainsi pouvoir déterminer la rotation quelque soit la base.

2 6 Produit vectoriel

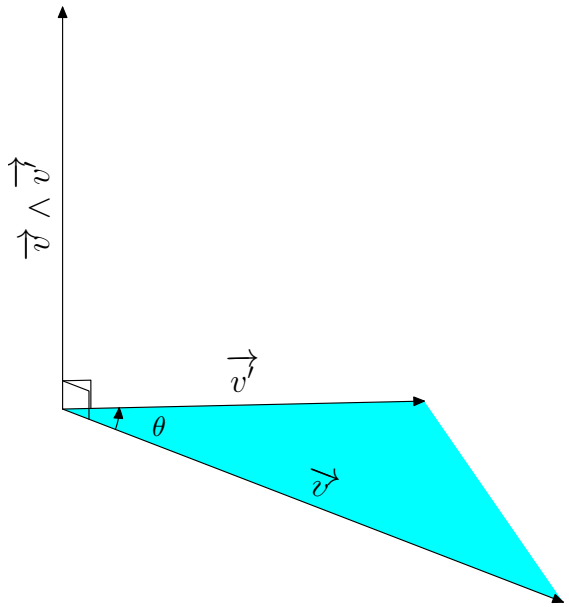
Nous avons parlé du produit scalaire qui à deux vecteurs associe un nombre réel (un scalaire). Nous allons maintenant introduire un nouveau produit qui à deux vecteurs associe un troisième vecteur.

Définition 1 - 7

produit vectoriel
Soit $\vec{v}$ et $\vec{v'}$ deux vecteurs de coordonnées (x, y, z) et (x', y', z') dans une base orthonormée directe.
On appelle produit vectoriel de $\vec{v}$ et $\vec{v'}$ et on note $\vec{v} \wedge \vec{v'}$ le vecteur dont les coordonnées sont :

$$\left(\begin{vmatrix} y & z \\ y' & z' \end{vmatrix}, -\begin{vmatrix} x & z \\ x' & z' \end{vmatrix}, \begin{vmatrix} x & y \\ x' & y' \end{vmatrix} \right)$$

On peut vérifier assez facilement que le *vecteur* $v \wedge v'$ est orthogonal à la fois à $\vec{v}$ et à $\vec{v'}$.



produit vectoriel

On montre aussi le résultat important suivant :

$$\left\|\vec{v} \wedge \vec{v'}\right\| = \|\vec{v}\| \times \left\|\vec{v'}\right\| \times \left|\sin\left(\vec{v}, \vec{v'}\right)\right|$$

Remarque

disposition pratique
Une méthode peu orthodoxe mais pratique de retenir la formule de calcul du produit scalaire est de le disposer comme un déterminant d'une matrice de taille 3 :

$$\vec{v} \wedge \vec{v'} = \begin{vmatrix} \vec{e_1} & \vec{e_2} & \vec{e_3} \\ x & y & z \\ x' & y' & z' \end{vmatrix}$$

puis de développer par rapport à la première ligne :

$$\vec{v} \wedge \vec{v'} = \begin{vmatrix} y & z \\ y' & z' \end{vmatrix} \vec{e_1} - \begin{vmatrix} x & z \\ x' & z' \end{vmatrix} \vec{e_2} + \begin{vmatrix} x & y \\ x' & y' \end{vmatrix} \vec{e_3}$$

On peut donc montrer que $\vec{v} \wedge \vec{v'} = -\vec{v'} \wedge \vec{v}$

Remarque

règle du tire-bouchon
Très grossièrement, quand on « tourne » de $\vec{v}$ vers $\vec{v'}$, on « monte le long » de $\vec{v} \wedge \vec{v'}$ et ~~vis~~ vice-versa

2 7 Exemple de détermination d'une rotation

On connaît la matrice d'une certaine application dans une certaine base :

$$A = \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ \frac{2}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & -\frac{2}{3} & \frac{2}{3} \end{pmatrix}$$

1. Vérifiez que la matrice est orthogonale.
2. Vérifiez que $\det(A) = 1$: il s'agit donc d'une rotation.
3. Pour déterminer l'axe, on va chercher un vecteur de représentation matricielle $V = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ invariant par la rotation donc tel que $A \times V - V$ soit égal à la matrice colonne nulle. On obtient après simplification (faites le calcul...) :

$$\begin{pmatrix} \frac{-x-y-2z}{3} \\ \frac{2x-y+z}{3} \\ \frac{x-2y-z}{3} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

cela revient à résoudre le système :

$$\begin{cases} \frac{-x-y-2z}{3} = 0 \\ \frac{2x-y+z}{3} = 0 \\ \frac{x-2y-z}{3} = 0 \end{cases}$$

Ce système admet une infinité de solutions car en additionnant les deux premières lignes on obtient la troisième. On choisit une des inconnues comme paramètre (par exemple z) et on résout le système résultant ayant maintenant deux équations et deux inconnues.

On obtient $x = -z$ et $y = -z$.

L'axe de la rotation est donc la droite vectorielle dirigée par le vecteur $\vec{v}$ de coordonnées $(1,1,-1)$.

4. Pour l'angle, on sait que $\cos\theta = \frac{\text{tr(A)}-1}{2} = \frac{2-1}{2} = \frac{1}{2}$.

On obtient donc que $\theta = \pm\frac{\pi}{3}$ à un multiple de 2π près.

Pour déterminer le signe, il nous faut un autre renseignement : nous allons utiliser le produit vectoriel. Nous allons déterminer un vecteur $\vec{n}$ orthogonal à $\vec{v}$. Il faut donc s'arranger pour que le produit scalaire de $\vec{n}$ et $\vec{v}$ soit nul.

Si nous posons (x,y,z) les coordonnées de $\vec{n}$ alors $\langle \vec{v}, \vec{n} \rangle = x + y - z$.

Nous pouvons donc choisir $(1,0,1)$ pour les coordonnées de $\vec{n}$.

Déterminons à présent l'image de $\vec{n}$ par la rotation :

$$\underbrace{\begin{pmatrix} \frac{2}{3} & \frac{-1}{3} & \frac{-2}{3} \\ \frac{2}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{-2}{3} & \frac{2}{3} \end{pmatrix}}_{\boxed{\text{A}}} \quad \overbrace{\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}}^{\boxed{\text{N}}} \quad \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

Donc $r(\vec{n})$ a pour coordonnées $(0,1,1)$.

Calculons à présent le produit vectoriel de $\vec{n}$ par $r(\vec{n})$:

$$\vec{n} \wedge r(\vec{n}) = \begin{vmatrix} \vec{e_1} & \vec{e_2} & \vec{e_3} \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} = -\vec{e_1} - \vec{e_2} + \vec{e_3} = -\vec{v}$$

Si nous en avions le loisir, nous pourrions montrer le résultat suivant :

Propriété 1 - 4

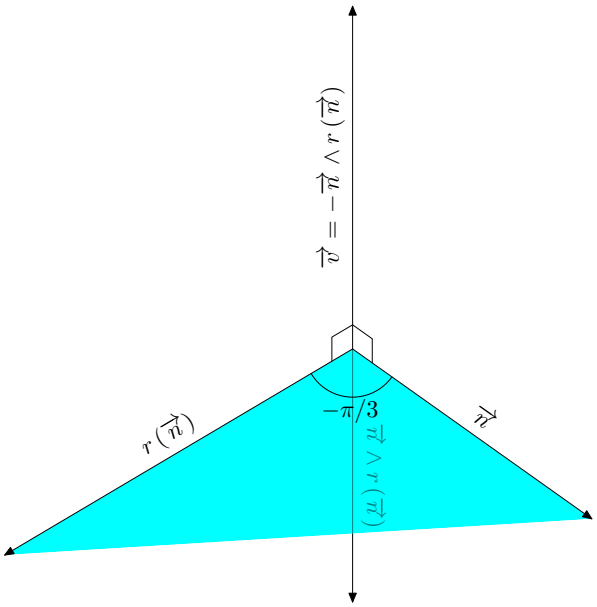
rotation et produit vectoriel

Si l'axe d'une rotation r est dirigé par un vecteur $\vec{v}$ et si $\vec{n}$ est un vecteur du plan orthogonal à $\vec{v}$ alors

$$\vec{n} \wedge r(\vec{n}) = \frac{\|\vec{n}\| \times \|r(\vec{n})\|}{\|\vec{v}\|} \times \sin(\theta) \cdot \vec{v}$$

Ainsi, $-\vec{v} = \frac{\sqrt{2}\times\sqrt{2}}{\sqrt{3}} \times \sin(\theta) \cdot \vec{v}$. Nous en déduisons que $\sin(\theta) = -\frac{\sqrt{3}}{2}$ or $\cos(\theta) = \frac{1}{2}$.

Finalement, A est la matrice dans la base canonique de la rotation d'axe dirigé par $\vec{v}$ de coordonnées $(1,1,-1)$ et d'angle $-\frac{\pi}{3}$.



rotation de matrice A

3 Changement de base

3 1 Le problème

On connaît la matrice A d'une rotation relativement à une certaine base, généralement une base orthonormée dont un des vecteurs dirige l'axe de la rotation.
On voudrait alors connaître la matrice de cette rotation mais relativement à une autre base...

3 2 Matrice de passage

Définition 1 - 8

matrice de passage
Soit $\mathcal{B}$ et $\mathcal{B}'$ deux bases de $\mathbb{R}^3$. On appelle matrice de passage de $\mathcal{B}$ à $\mathcal{B}'$ et on note $\text{mat}_{\mathcal{B}}(\mathcal{B}')$ ou encore $\mathcal{P}_{\mathcal{B},\mathcal{B}'}$ la matrice de $\mathfrak{M}_3(\mathbb{R})$ dont les colonnes sont les vecteurs de $\mathcal{B}'$ exprimés dans $\mathcal{B}$:

$$\begin{array}{c} \begin{array}{ccc} \vec{e}'_1 & \vec{e}'_2 & \vec{e}'_3 \end{array} \\ \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix} \end{array} \begin{array}{c} \vec{e}_1 \\ \vec{e}_2 \\ \vec{e}_3 \end{array}$$

À noter : si les bases sont *orthonormées* alors les matrices de passages seront *orthogonales*.

Nous retiendrons alors la propriété importante suivante :

Propriété 1 - 5

matrice de passage inverse

$$\mathcal{P}_{\mathcal{B}',\mathcal{B}} = \mathcal{P}^{-1}_{\mathcal{B},\mathcal{B}'}$$

En particulier, si les bases sont *orthonormées* :

$$\mathcal{P}_{\mathcal{B}',\mathcal{B}} = {}^{t}\mathcal{P}_{\mathcal{B},\mathcal{B}'}$$

3 3 Changement de coordonnées d'un vecteur

Soit $\vec{v}$ un vecteur dont les coordonnées dans une certaine base $\mathcal{B}$ sont (x, y, z).

On peut donc associer une matrice colonne V à ce vecteur : $V = \text{mat}_{\mathcal{B}}\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

ATTENTION ! À un même vecteur correspond une infinité de vecteurs colonnes différents selon la base choisie !

Notons maintenant $V' = \text{mat}_{\mathcal{B}'} \vec{v}$ alors :

Propriété 1 - 6

changement de coordonnées d'un vecteur

$$V = P_{\mathcal{B},\mathcal{B}'} \times V'$$

Notez bien que cette formule donne les anciennes coordonnées en fonction des nouvelles !

Remarque

On peut en fait retenir que les « primes » sont du côté des « primes ».

Cette formule n'est donc pas pratique sous cette forme mais sachant que $V = \mathcal{P}_{\mathcal{B},\mathcal{B}'} V'$, alors

$$V' = \mathcal{P}^{-1}_{\mathcal{B},\mathcal{B}'} \times V$$

et comme nous travaillerons dans des bases orthonormées, nous utiliserons le plus souvent :

$$V' = {}^t\mathcal{P}_{\mathcal{B},\mathcal{B}'} \times V$$

3 4 Relation de Chasles

Considérons 3 bases. Avec des notations standard on a

$$V'' = \mathcal{P}_{\mathcal{B}'',\mathcal{B}} \times V$$

Mais on a aussi $V' = \mathcal{P}_{\mathcal{B}',\mathcal{B}} \times V$ et $V'' = \mathcal{P}_{\mathcal{B}'',\mathcal{B}'} \times V'$ donc

$$V'' = \mathcal{P}_{\mathcal{B}'',\mathcal{B}'} \times \mathcal{P}_{\mathcal{B}',\mathcal{B}} \times V$$

Finalement :

Propriété 1 - 7

relation de Chasles

$$\mathcal{P}_{\mathcal{B}'',\mathcal{B}} = \mathcal{P}_{\mathcal{B}'',\mathcal{B}'} \times \mathcal{P}_{\mathcal{B}',\mathcal{B}}$$

3 5 Rotation et changement de base

Soit A la matrice d'une rotation r dans une certaine base. Notons toujours $V = \text{mat}_{\mathcal{B}} \vec{v}$ et $W = \text{mat}_{\mathcal{B}} r(\vec{v})$ et leurs pendants « primés ».
Nous avons vu que $W = A \times V$. D'après ce que nous avons vu au paragraphe précédent, cela donne :

$$\mathcal{P}_{\mathcal{B},\mathcal{B}'} \times W' = A \times \mathcal{P}_{\mathcal{B},\mathcal{B}'} \times V'$$

et donc

$$W' = \left({}^t\mathcal{P}_{\mathcal{B},\mathcal{B}'} \times A \times \mathcal{P}_{\mathcal{B},\mathcal{B}'}\right) \times V'$$

On en déduit que $A' = {}^t\mathcal{P}_{\mathcal{B},\mathcal{B}'} \times A \times \mathcal{P}_{\mathcal{B},\mathcal{B}'}$

Théorème 1 - 5

transformation orthogonale et changement de base

$$\text{mat}_{\mathcal{B}'} f = {}^t\mathcal{P}_{\mathcal{B},\mathcal{B}'} \times \text{mat}_{\mathcal{B}} f \times \mathcal{P}_{\mathcal{B},\mathcal{B}'}$$

Il sera pratique de retenir le schéma suivant (attention au sens des flèches) :

Remarque

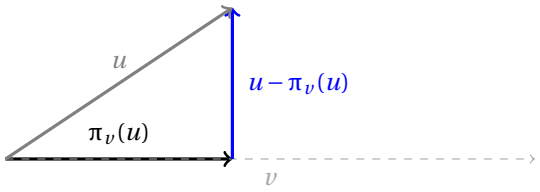
schéma du changement de base

$$\begin{array}{ccc} \mathcal{B}\ \mathbb{R}^3 & \xrightarrow[A]{f} & \mathbb{R}^3\ \mathcal{B} \\ \mathcal{P}_{\mathcal{B},\mathcal{B}'} \downarrow & & \downarrow \mathcal{P}_{\mathcal{B},\mathcal{B}'} \\ \mathcal{B}'\ \mathbb{R}^3 & \xrightarrow[A']{f} & \mathbb{R}^3\ \mathcal{B}' \end{array}$$

4 Projection orthogonale

4 1 Projection orthogonale sur une droite orthogonale

La figure clé qui guidera notre intuition est celle-ci :



Elle correspond à la définition suivante :

Définition 1 - 9

projection orthogonale sur une droite vectorielle
La projection orthogonale de u sur v est le vecteur noté $\pi_v(u)$ défini par :

$$\pi_v(u) = \frac{\langle u, v\rangle}{||v||^2} v$$

Nous démontrerons en exercice que :

Propriété 1 - 8

Le vecteur $u - \pi_v(u)$ est orthogonal à v.

et également :

Propriété 1 - 9

Inégalité de Cauchy-Schwarz

$$|\langle u, v\rangle| \leqslant ||u|| \cdot ||v||$$

ce qui permet d'obtenir la confirmation de ce qui se voit sur le dessin :

Propriété 1 - 10

$$||\pi_v(u) \leqslant ||u||$$

4 2 Projection orthogonales sur un sev

On peut montrer que les composantes d'un vecteur dans une base orthogonale $(e_1, e_2, ..., e_n)$ est :

$$\begin{aligned} u &= \pi_{e_1} e_1 + \pi_{e_2} e_2 + \cdots + \pi_{e_n} e_n \\ &= \frac{\langle u, e_1\rangle}{||e_1||^2} e_1 + \frac{\langle u, e_2\rangle}{||e_2||^2} e_2 + \cdots \frac{\langle u, e_n\rangle}{||e_n||^2} e_n \end{aligned}$$

et si la base est orthonormale :

$$u = \rangle u, e_1 \langle e_1 + \rangle u, e_2 \langle e_2 + \cdots \rangle u, e_n \langle e_n$$

On admettra alors le théorème suivant :

Théorème 1 - 6

Soit W un sev de V et $(e_1, e_2, \cdots, e_n)$ une base orthonormale de W. Alors tout vecteur u de V peut s'écrire sous la forme $u = w_1 + w_2$ avec w_1 un vecteur de W et w_2 un vecteur orthogonal à W (i.e. orthogonal à tous les vecteurs de base de W). Alors :

$$w_1 = \rangle u, e_1 \langle e_1 + \rangle u, e_2 \langle e_2 + \cdots \rangle u, e_n \langle e_n$$

On appelle w_1 la projection orthogonale de u sur W.

4.3 Procédé d'orthonormalisation de Gram-Schmidt

Nous aurons besoin pour compresser nos images en niveaux de gris d'orthonormaliser une base donnée. Nous admettrons que le procédé suivant répond à notre problème.
On part d'une base $(e_1,...,e_n)$ quelconque et on veut en déduire une base $(u_1,...,u_n)$ orthonormale. Il suffit de prendre :

$$\begin{aligned}
u_1 &= \frac{w_1}{||w_1||}, \text{ où } w_1 = e_1 \\
u_2 &= \frac{w_2}{||w_2||}, \text{ où } w_2 = e_2 - \frac{\langle e_2, w_1\rangle}{||w_1||^2} w_1 \\
&\vdots \\
u_n &= \frac{w_n}{||w_n||}, \text{ où } w_n = e_n - \frac{\langle e_n, w_1\rangle}{||w_1||^2} w_1 - \frac{\langle e_n, w_2\rangle}{||w_1||^2} w_2 - \cdots - \frac{\langle e_n, w_{n-1}\rangle}{||w_{n-1}||^2} w_{n-1}
\end{aligned}$$

Les utilisations de ce procédé sont très nombreuses dans des domaines très divers.

5 Décomposition en valeurs singulières (SVD)

Nous allons travailler avec des images qui sont des matrices de niveaux de gris. Notre belle amie Léna sera représentée par une matrice carrée de taille 2^9 ce qui permet de reproduire Léna à l'aide de $2^{18} = 262\,144$ pixels. Léna prend alors beaucoup de place. Nous allons tenter de compresser la pauvre Léna sans pour cela qu'elle ne perde sa qualité graphique. Une des méthodes les plus abordables est d'utiliser la décomposition d'une matrice en valeurs singulières. Nous passerons outre la plupart des démonstrations mais, en bon(ne) informaticien(ne) nous essaierons de comprendre le mécanisme général pour mieux appréhender son utilisation « dans la vraie vie ». C'est également une nouvelle preuve des liens étroits existants entre informatique et mathématique...
C'est un sujet extrêmement riche qui a de nombreuses applications. L'algorithme que nous utiliserons (mais que nous ne détaillerons pas) a été mis au point par deux très éminents chercheurs en 1965 (Gene GOLUB, états-unien et William KAHAN, canadien, père de la norme IEEE-754).

5.1 Valeurs et vecteurs propres

5.1.1 Généralités

Un *vecteur propre* v (*eigenvector* en anglais de spécialité ;-)) d'un endomorphisme f d'un $\mathbb{K}$-espace vectoriel E est un vecteur non nul satisfaisant l'équation :

$$f(v) = \lambda v$$

avec λ un élément de $\mathbb{K}$ appelé *valeur propre* (*eigenvalue*) associée au vecteur propre v.
Cela se traduit matriciellement dans une certaine base $\mathcal{B}$. Si A est la matrice de f dans $\mathcal{B}$ et V le vecteur colonne des coordonnées de v dans $\mathcal{B}$, on a :

$$\mathrm{A} \times \mathrm{V} = \lambda \cdot \mathrm{V}$$

Comme f est un endomorphisme, A est une matrice carrée.

5.1.2 Un exemple

Considérons par exemple une matrice $\mathrm{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$.

Posons $\mathrm{V} = \begin{pmatrix} x \\ y \end{pmatrix}$.
On recherche donc x et y ainsi qu'un scalaire λ vérifiant :

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

C'est un simple petit système linéaire :

$$(\mathrm{S}_\lambda) : \begin{cases} 2x + y = \lambda x \\ x + 2y = \lambda y \end{cases}$$

qui est équivalent au système :

$$(S_\lambda) \Leftrightarrow \begin{cases} (2-\lambda)x + y = 0 \\ x + (2-\lambda)y = 0 \end{cases}$$

Ce système admet au moins une solution (laquelle ?...) mais cela ne nous arrange pas. Pour que v soit un vecteur propre, il faut et il suffit que le déterminant de ce système soit nul (pourquoi ?). Or :

$$\begin{vmatrix} (2-\lambda) & 1 \\ 1 & (2-\lambda) \end{vmatrix} = 0 \Leftrightarrow \lambda^2 - 4\lambda + 3 = 0$$

Cette équation admet exactement deux solutions, 3 et 1 qui sont les valeurs propres de A.
Il reste à déterminer des vecteurs propres associés (sont-ils uniques ?).
Remplaçons pour cela λ par 3 dans l'équation.
Le système devient :

$$(S_3) \Leftrightarrow \begin{cases} -x + y = 0 \\ x - y = 0 \end{cases} \Leftrightarrow x = y$$

Un vecteur propre de A associé à 3 est donc par exemple $(1,1)$ mais également tous ses multiples.
Le *sous-espace propre* E_3 associé à 3 (c'est-à-dire l'ensemble des vecteurs propres associé à λ) est donc $\text{Vect}\{(1,1)\}$.
On trouve de même que $E_1 = \text{Vect}\{(1,-1)\}$.
Si la somme des dimensions des sous-espaces propres est égale à la dimension de l'espace E, alors on dit que A est DIAGONALISABLE et peut s'écrire sous la forme :

$$A = P \times \Delta \times P^{-1}$$

avec P la matrice de passage de $\mathcal{B}$ vers la base de vecteurs propres et Δ une matrice diagonales dont les éléments diagonaux sont les valeurs propres de A.
Il est à noter que toute matrice n'est pas diagonalisable. C'est un problème important qui rend de nombreux services dans des domaines très variés quand il est résolu.

5 2 Décomposition en valeurs singulières

5 2 1 Valeurs et vecteurs singuliers

Nous venons d'étudier les éléments propres d'une matrice *carrée*. Les éléments singuliers en sont une généralisation aux matrices de taille quelconque.
Soit A la matrice d'une application linéaire d'un espace E de dimension m dans un espace F de dimension n. Alors A est rectangulaire.
Un réel *positif* σ est une valeur singulière associée à A si, et seulement si, il existe un vecteur orthonormé u dans E et un vecteur orthonormé v dans F tels que (avec les notations usuelles) :

$$A \times V = \sigma \cdot U \qquad \text{et} \qquad {}^tA \times U = \sigma \cdot V$$

On dit que u (v) est un vecteur singulier à gauche (à droite) pour σ.

5 2 2 La SVD

Un beau théorème affirme alors que toute matrice rectangulaire A se décompose sous la forme :

$$A = U \times S \times {}^tV$$

avec U et V des matrices orthogonales et S une matrice nulle partout sauf sur sa diagonale principale qui contient les valeurs singulières de A rangées dans l'ordre décroissant.
Ce qui est remarquable, c'est que n'importe quelle matrice admet une telle décomposition, alors que la décomposition en valeurs propres (la diagonalisation d'une matrice) n'est pas toujours possible.
Cependant, la décomposition en éléments singuliers utilise la décomposition en éléments propres comme nous le verrons en exercice
Notons r le rang de A et U_i et V_i les vecteurs colonnes de U et V. La décomposition s'écrit :

$$A = \begin{pmatrix} U_1 & U_2 & \cdots & U_r & \cdots & U_m \end{pmatrix} \times \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & \ddots & & \\ & & & & 0 & \end{pmatrix} \times \begin{pmatrix} {}^tV_1 \\ \vdots \\ {}^tV_r \\ \vdots \\ {}^tV_n \end{pmatrix}$$

ou formulé autrement :

$$\begin{aligned} A &= \sigma_1 \cdot U_1 \times {}^tV_1 + \sigma_2 \cdot U_2 \times {}^tV_2 + \cdots + \sigma_r \cdot U_r \times {}^tV_r + 0 \cdot U_{r+1} \times {}^tV_{r+1} + \cdots \\ &= \sigma_1 \cdot U_1 \times {}^tV_1 + \sigma_2 \cdot U_2 \times {}^tV_2 + \cdots + \sigma_r \cdot U_r \times {}^tV_r \end{aligned}$$

Il faut ensuite se souvenir que les σ_i sont classés dans l'ordre décroissant ce qui va avoir une application importante en informatique comme nous le verrons en exercice...

6 EXERCICES

6 1 Rotations

Exercice 1 - 1 reconnaître une application linéaire

Est-ce que l'application : $\begin{array}{ccc} \mathbb{R}^3 & \longrightarrow & \mathbb{R}^3 \\ \vec{x} & \longmapsto & \vec{x}+\vec{a} \end{array}$ avec $\vec{a}$ un vecteur fixe est linéaire ?

Exercice 1 - 2

$f \in \mathcal{L}\left(\mathbb{R}^3\right)$, $\mathcal{B}=(e_1,e_2,e_3)$ est une base de $\mathbb{R}^3$, $\mathrm{A}=\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}=\mathbf{Mat}_{\mathcal{B}}\left(f\right)$.

1. f est-il un endomorphisme de $\mathbb{R}^3$?
2. Quelles sont les coordonnées de e_1 dans la base $\mathcal{B}$?
3. Quelles sont les coordonnées de $f(e_2)$ dans la base $\mathcal{B}$?
4. Quelles sont les coordonnées de $f(x)$ dans la base $\mathcal{B}$ si $x=e_1+e_2-2e_3$?
5. Qu'est ker f ?

Exercice 1 - 3

$f \in \mathcal{L}(\mathbb{R}^p,\mathbb{R}^n)$, démontrer :

$$\ker f=\{0_{\mathbb{R}^p}\} \Leftrightarrow f \text{ est injective}$$

Exercice 1 - 4 détermination d'une rotation à partir d'une matrice

On donne la matrice $\mathrm{A}=\frac{1}{3}\begin{pmatrix} 1 & 2 & 2 \\ -2 & 2 & -1 \\ -2 & -1 & 2 \end{pmatrix}$ d'une certaine transformation f de $\mathbb{R}^3$ dans lui-même. Décrivez f le plus précisément possible.

Faites de même avec la matrice $\mathrm{B}=\frac{1}{3}\begin{pmatrix} 2 & 1 & 2 \\ -2 & 2 & 1 \\ 1 & 2 & -2 \end{pmatrix}$

Exercice 1 - 5 matrice d'une rotation

On considère la rotation u d'angle $\frac{\pi}{4}$ autour de l'axe dirigé par le vecteur $\overrightarrow{v_1}$ de coordonnées $\left(\frac{1}{3},\frac{2}{3},\frac{2}{3}\right)$ dans la base canonique $\mathcal{B}_0$.

1. Montrez que $\overrightarrow{v_2}$ de coordonnées $\left(\frac{2}{3},-\frac{2}{3},\frac{1}{3}\right)$ dans $\mathcal{B}_0$ est orthogonal à $\overrightarrow{v_1}$.
2. Déterminez un vecteur $\overrightarrow{v_3}$ tel que $(\overrightarrow{v_1},\overrightarrow{v_2},\overrightarrow{v_3})$ soit une base orthonormée directe.
3. Déterminez $\mathrm{mat}_{\mathcal{B}_0} u$.

Exercice 1 - 6 composée de rotations

Dans $\mathbb{R}^3$, soit R_x la rotation autour de l'axe dirigé par $\overrightarrow{e_1}$ et d'angle $\frac{\pi}{2}$, soit R_y la rotation autour de l'axe dirigé par $\overrightarrow{e_2}$ et d'angle $\frac{\pi}{2}$ et soit R_z la rotation autour de l'axe dirigé par $\overrightarrow{e_3}$ et d'angle $\frac{\pi}{2}$.
Décrivez $\mathrm{R}_x \circ \mathrm{R}_z$.

Exercice 1 - 7

On travaille dans $\mathbb{R}^3$ et $\mathcal{B}=\left(i,j,k\right)$ est une BOND. Dans ce qui suit, pour les représentations, on choisira une mesure d'angle peu différente de $\frac{\pi}{6}$ ou $\frac{\pi}{3}$.

1. On considère la rotation $\mathrm{Rot}_1(i,\alpha)$ qui transforme $\mathcal{B}$ en $\mathcal{B}_1=\left(i_1,j_1,k_1\right)$, représenter $\mathcal{B}$ et $\mathcal{B}_1$ et déterminer la matrice R_1 de $\mathrm{Rot}_1(i,\alpha)$ dans la base $\mathcal{B}$.
2. On considère la rotation $\mathrm{Rot}_2\left(j,\beta\right)$ qui transforme $\mathcal{B}$ en $\mathcal{B}_2=\left(i_2,j_2,k_2\right)$, représenter $\mathcal{B}$ et $\mathcal{B}_2$ et déterminer la matrice R_2 de $\mathrm{Rot}_2\left(j,\beta\right)$ dans la base $\mathcal{B}$.
3. On considère la rotation $\mathrm{Rot}_3\left(k,\gamma\right)$ qui transforme $\mathcal{B}$ en $\mathcal{B}_3=\left(i_3,j_3,k_3\right)$, représenter $\mathcal{B}$ et $\mathcal{B}_3$ et déterminer la matrice R_3 de $\mathrm{Rot}_3\left(k,\gamma\right)$ dans la base $\mathcal{B}$.

4. Déterminer la matrice de $\text{Rot}_3(k,\gamma)$ dans la base $\mathcal{B}_3$.
5. Déterminer la matrice de $\text{Rot}_3(k,\gamma)$ dans la base $\mathcal{B}_2$.
6. Déterminer la matrice de $\text{Rot}_2(j,\beta)$ dans la base $\mathcal{B}_2$.
7. Déterminer la matrice de $\text{Rot}_2(j,\beta)$ dans la base $\mathcal{B}_3$.
8. On considère la rotation $\text{Rot}_4(j_3,\varphi)$ qui transforme la base $\mathcal{B}_3 = (i_3, j_3, k_3)$ en la base $\mathcal{B}_4 = (i_4, j_4, k_4)$.
   (a) Déterminer la matrice de $\text{Rot}_4(j_3,\varphi)$ dans la base $\mathcal{B}_3$.
   (b) Déterminer la matrice de $\text{Rot}_4(j_3,\varphi)$ dans la base $\mathcal{B}$.
9. Déterminer la matrice $\text{Rot}_1 \circ \text{Rot}_2 \circ \text{Rot}_3$ dans la base $\mathcal{B}_2$.
10. Détertminer la matrice de $\text{Rot}_2 \circ \text{Rot}_4 \circ \text{Rot}_2 \circ \text{Rot}_3 \circ \text{Rot}_2$ dans la base $\mathcal{B}_4$.
11. Déterminer la matrice de passage de $\mathcal{B}_2$ à $\mathcal{B}_4$.
12. Déterminer la matrice de passage de $\mathcal{B}_4$ à $\mathcal{B}_3$.

Exercice 1 - 8

$\mathcal{B} = (i, j, k)$ est une **BOND** de $\mathbb{R}^3$, r_1 est la rotation d'axe j et d'angle de mesure $\alpha \in [0, \frac{\pi}{2}]$ qui transforme la BOND $\mathcal{B}$ en la BOND $\mathcal{B}_1 = (i_1, j_1, k_1)$.

1. Faire un dessin clair représentant $\mathcal{B}$ et $\mathcal{B}_1$.
2. Donner R la matrice de r_1 dans la base $\mathcal{B}$.
3. r_2 est la rotation d'axe i_1 qui transforme la BOND $\mathcal{B}_1$ en la BOND $\mathcal{B}_2$ et r_3 est la rotation d'axe k qui transforme la BOND $\mathcal{B}$ en la BOND $\mathcal{B}_3$. On note S la matrice de r_2 dans la base $\mathcal{B}_1$ et T la matrice de r_3 dans la base $\mathcal{B}$. Donner la matrice de $r_2 \circ r_1 \circ r_3 \circ r_1 \circ r_2$ dans la base B_3.

Exercice 1 - 9

$\mathcal{B} = (i, j, k)$ est une **BOND** de $\mathbb{R}^3$, on considère les vecteurs u et v :

$$u\begin{pmatrix}1\\2\\2\end{pmatrix}_{\mathcal{B}} \quad v\begin{pmatrix}-2\\0\\1\end{pmatrix}_{\mathcal{B}}$$

et il est manifeste que u et v sont orthogonaux. Déterminer des vecteurs a, b et c vérifiant : a colinéaire à u, c colinéaire à v et (a, b, c) est une BOND.

Exercice 1 - 10 Théorème de Pythagore

Démontrez que u et v sont orthogonaux si, et seulement si,

$$||u + v||^2 = ||u||^2 + ||v||^2$$

6 2 Projections

Exercice 1 - 11

1. Démontrez la propriété 1 - 8 page 14.
2. Démontrez la propriété 1 - 9 page 14. Pour cela, vous commencerez par développer $||u + xv||^2$ avec x un réel quelconque. Vous obtenez alors un polynôme du second degré en x : quel est son signe ? Qu'en déduit-on concernant son discriminant ? Concluez.
3. Démontrez la propriété 1 - 10 page 14

Exercice 1 - 12 Niveaux de gris

Un pixel, en général, est représenté par un vecteur (R,V,B), chacune des composante pouvant prendre 256 valeurs entre 0 et 255. Cela donne un échantillon de 2^{24} couleurs...
L'espace des couleurs est alors l'intérieur d'un cube de côté de longueur 256.
Lors de la conversion en niveau de gris, on veut obtenir un nouveau vecteur gris (i.e. avec trois composantes égales) le plus « proche » possible du pixel en couleur.

1. Donnez une base de « l'espace des gris ».
2. Expliquez intuitivement quelle est la plus petite distance entre le point de coordonnées (R,V,B) et la droite des gris dans un repère bien choisi.
3. Déduisez-en une formule permettant de convertir une image en niveau de gris connaissant sa matrice « en couleur ».

6 3 Diagonalisation et SVD

Exercice 1 - 13 Calcul manuel de la SVD

On suppose que la SVD d'une matrice A de taille $m \times n$ s'écrit $U \times S \times {}^tV$.

1. Quelles sont les dimensions de U et V ?
2. Montrez que $A \times {}^tA$ est diagonalisable et que les colonnes de U en forment une base orthonormée de vecteurs propres.
   Montrez de même que les colonnes de V forment une base orthonormée de vecteurs propres de ${}^tA \times A$.
3. Calculez (à la main...) la SVD de $A = \begin{pmatrix} 2 & -2 \\ 1 & 1 \end{pmatrix}$.
   On aura besoin d'orthogonaliser la base de vecteurs propres obtenue...
4. Shoot again avec $\begin{pmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{pmatrix}$

6 4 Lena...

La petite histoire dit que des chercheurs américains de l'University of Southern California étaient pressés de trouver une image de taille 512×512 pour leur conférence. Passe alors un de leurs collègues avec le dernier Playboy sous le bras. Ils décidèrent alors d'utiliser le poster de la Playmate comme support...
La photo originale est ici : http ://www.lenna.org/full/len_full.html mais nous n'utiliserons que la partie scannée par les chercheurs, de taille 5.12in × 5.12in...

6 4 1 Python

Nous aurions pu utiliser Caml qui a un module graphique puissant. Cependant, il code les couleurs sur 24 bits ce qui compliquerait notre approche naïve.
Nous allons donc utiliser Python qui est très simple d'utilisation, possède des bibliothèques très utiles pour illustrer notre propos et permet de retrouver des outils fonctionnels comme `map` et `lambda` qui correspond au `fun` de Caml.
Nous utiliserons iPython pour dialoguer avec Python qu'on lancera avec l'option `pylab` qui charge tout ce dont nous aurons besoin.

Shell

```
$ ipython --pylab
```

Nous pouvons travailler dans emacs avec un buffer contenant notre fichier `lena.py` et un buffer shell dans lequel nous avons ouvert iPython qui a quelques commandes magiques (c'est le terme officiel) comme `%paste` et `%run`. Découvrons d'abord Lena :

Python

```
from scipy import misc

l = misc.lena()
imshow(l,cmap=cm.gray)
```

Python

```
In [8]: type(l)
Out[8]: numpy.ndarray

In [9]: shape(l)
Out[9]: (512, 512)
```

Ceci indique que `t` est un tableau de tableaux de taille 512.
Commentez la fonction suivante et le résultat de son utilisation :

Python
```
def tourne(m):
    n = len(m)-1
    return(array([[m[j,i] for j in range(n)] for i in range(n)]))

imshow(tourne(l),cmap=cm.gray)
```

Dans la suite, nous aurons peut-être besoin de la fonction `map` :

Python
```
In [10]: map(lambda x: x**2, array([1,2,3]))
Out[10]: [1, 4, 9]
```

de la fonction `dot` qui multiplie deux tableaux au sens du produit de matrices :

Python
```
In [12]: dot(array([[1,2,3]]) , array([[4],[5],[6]]))
Out[12]: array([[32]])
```

du produit d'un scalaire par une matrice :

Python
```
In [13]: 5 * array([[1,2,3]])
Out[13]: array([[ 5, 10, 15]])
```

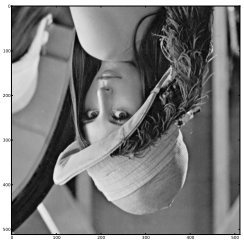
d'une matrice de zéros de taille quelconque :

Python
```
In [14]: zeros((2,3))
Out[14]:
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

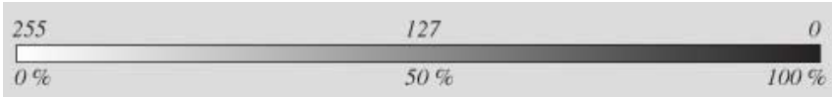
6 4 2 Manipulations de Lena

Comment obtenir les images suivantes :



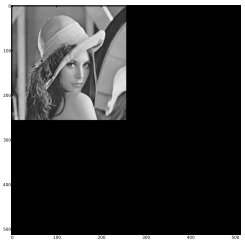


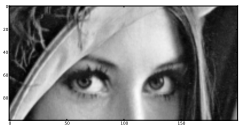
Sachant que la palette de niveaux de gris en 8 bits part du 0 (noir) vers 255 (blanc) :



On peut modifier le contraste en « mappant » par une fonction croissante ou bien une fonction décroissante sur [0 ;255].
On peut même imaginer d'autres manipulations si l'on dispose d'une fonction qui applique une fonction sur un tableau, if you see what I mean... Comment obtenir ces images :


On peut modifier la taille, dupliquer. Comment obtenir ceci :





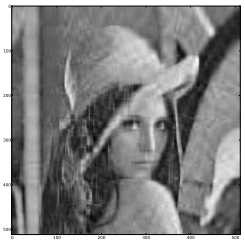
Tout ceci prend un peu de place en mémoire. Il est temps de compresser. Une première idée est de réduire le nombre de niveaux de gris disponibles.
Par exemple, on peut décider que les gris de 0 à 63 vaudront 0, les gris de 63 à 127 vaudront 1, etc.
Créez une fonction qui permet d'obtenir les images suivantes (ayant respectivement 4 et 32 niveaux de gris) :

On peut être beaucoup plus efficace en utilisant la SVD. En effet, la matrice image peut être décomposée sous la forme : $A = \sigma_1 \cdot U_1 \times {}^tV_1 + \sigma_2 \cdot U_2 \times {}^tV_2 + \cdots + \sigma_r \cdot U_r \times {}^tV_r$
avec les σ_i de plus en plus petits. On peut donc approximer A en ne gardant que les premiers termes de la somme sachant que les derniers sont multipliés par des « petits » σ_i.
Le problème est d'obtenir cette décomposition. Python heureusement s'en charge :

Python

```
In [17]: U,s,tV = linalg.svd(l)
```

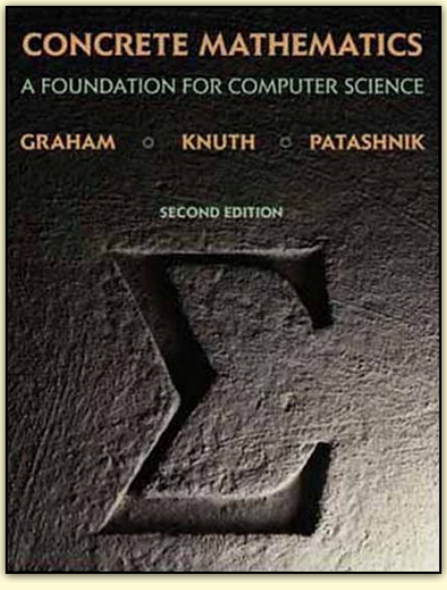
La somme obtenue n'est pas forcément entière. On utilisera alors la fonction `floor` qui peut s'appliquer à un tableau entier. Voici ce qu'on obtient avec 20, 50 et 100 termes dans la suite. Quelles sont les économies réalisées ?



TD N°

2

Mathématiques concrètes



On distingue les mathématiques CONtinues des mathématiques disCRÈTES. Les premières correspondent à celles que vous avez découvertes au lycée avec le calcul différentiel (dérivées, intégrales) alors que les secondes correspondent à celles que vous avez étudiées jusqu'à maintenant à l'IUT (suites, arithmétiques, dénombrement, graphes, automates,...). Nous allons dans cette partie les lier plus étroitement : l'informaticien doit en effet faire le lien entre l'univers physique qui l'entoure qui est continu et la mémoire de son ordinateur qui est discrète. C'est la même distinction que l'on fait entre un système analogique (une montre à gousset) et un système digital (une montre à affichage par cristaux liquides).

1 Complexité d'algorithmes

Nous avons déjà parlé de la complexité du calcul du déterminant avec la définition et avec la méthode du pivot de GAUSS : pour une matrice de taille 30 on peut effecteur le calcul en quelques secondes dans un cas et en quelques milliards d'années dans l'autre : ce n'est pas un détail de s'occuper de la complexité d'un algorithme...

Cela demande souvent une certaine agilité en calcul !...
Vous avez étudié le tri fusion l'an passé. En voici une version pythonesque :

Python

```
def divise(liste):
    n = len(liste)
    if n == 0:
        return [],[]
    return liste[: n//2], liste[n//2:]

def fusionner(L1,L2):
    if len(L1) == 0:
        return L2
    if len(L2) == 0:
        return L1
    if L1[0] < L2[0]:
        return ([L1[0]] + fusionner(L1[1:],L2))
    else:
        return ([L2[0]] + fusionner(L2[1:],L1))

def tri_f(L):
    if len(L) == 1:
        return L
    return fusionner(tri_f(divise(L)[0]),tri_f(divise(L)[1]))
```

Nous allons étudier la complexité temporelle de cet algorithme. La complexité spatiale est en effet tombée en désuétude quand une modeste clé USB à cinq euros contient en mémoire ce qui tenait dans des armoires occupant plusieurs étages d'un immeuble dans les années 1970 et coûtait le prix d'un château...
Soit n la longueur de la liste à trier. Le coût de la division est constant et celui de la fusion est proportionnel à n. Il faut ensuite ajouter le coût de la fusion des deux sous-listes issues des appels récursifs.
Appelons K la fonction coût associé au tri par fusion, alors :

$$\begin{cases} \mathrm{K}(0) = \mathrm{K}(1) = 0 \\ \mathrm{K}(n) = \mathrm{K}\left(\lceil \frac{n}{2} \rceil\right) + \mathrm{K}\left(\lfloor \frac{n}{2} \rfloor\right) + cn \end{cases}$$

Posons $u_n = \mathrm{K}(n)$. La récurrence définissant la suite (u_n) est compliquée mais, comme pour la dichotomie, nous allons l'étudier dans le cas particulier où n est une puissance de 2. On peut toutefois montrer rapidement par récurrence que la suite (u_n) est croissante.
Posons maintenant $n = 2^k$ et $u_n = u_{2^k} = x_k$, alors $x_0 = 0$ et, pour $k \in \mathbb{N}^*$, $x_k = 2x_{k-1} + 2^k c$. On obtient successivement :

$$x_k = 2\left(x_{k-1} + c2^{k-1}\right) = 2\left(2\left(x_{k-2} + c2^{k-2}\right) + c2^{k-1}\right)$$

On montre alors par récurrence que

$$x_k = 2^k x_0 + c \cdot k \cdot 2^k = c \cdot k \cdot 2^k$$

Or, pour tout entier n non nul,

$$2^{\lfloor \log_2(n) \rfloor} \leqslant n \leqslant 2^{\lfloor \log_2(n) \rfloor + 1}$$

donc, comme $(\mathrm{K}(n))_{n \in \mathbb{N}}$ est croissante

$$2^{\lfloor \log_2(n) \rfloor}(c\lfloor \log_2(n) \rfloor) \leqslant \mathrm{K}(n) \leqslant 2^{\lfloor \log_2(n) \rfloor + 1} c\left(\lfloor \log_2(n) \rfloor + 1\right)$$

c'est-à-dire

$$2nc\lfloor \log_2(n) \rfloor \leqslant \mathrm{K}(n) \leqslant 4nc\left(\lfloor \log_2(n) \rfloor + 1\right)$$

Par exemple, si on prend $c = 1$ et qu'on travaille sur une liste de taille 10 000, alors :

$$265\,600 = 20\,000 \times 13,28 \leqslant \mathrm{K}(10\,000) \leqslant 40000(13,28+1) = 571\,200$$

Pour avoir un ordre de comparaison, si l'on prend le tri par insertion, dans le pire des cas, il faut effectuer $n+(n-1)+(n-2)+\cdots+2+1=\frac{n(n+1)}{2}$ comparaisons, ce qui nous donne pour une liste de taille 10 000, au pire une complexité de l'ordre de 50 000 000.
Si l'on effectue 10^5 opérations élémentaires par seconde, il faudra donc moins de cinq secondes pour trier dans un cas et presque neuf minutes dans l'autre dans le pire des cas : il n'y a pas photo...
On voit bien cependant qu'on ne peut prévoir au millième de seconde près ce qui va se passer : on se contente d'un ordre de grandeur. Il nous faut donc un critère pour comparer les ordres de grandeurs de fonctions.

2 Relations de domination

> *Brooks's Law [prov.]*
> *«Adding manpower to a late software project makes it later» – a result of the fact that the expected advantage from splitting work among N programmers is* O(N)*, but the complexity and communications cost associated with coordinating and then merging their work is* O(N^2)
>
> in « The New Hacker's Dictionnary »
> http ://outpost9.com/reference/jargon/jargon_17.html#SEC24

Les notations de LANDAU(1877-1938) ont en fait été crées par Paul BACHMANN(1837-1920) en 1894, mais bon, ce sont tous deux des mathématiciens allemands.
Par exemple, si l'on considère l'expression :

$$f(n) = n+1+\frac{1}{n}+\frac{75}{n^2}-\frac{765}{n^3}+\frac{\cos(12)}{n^{37}}-\frac{\sqrt{765\,481}}{n^{412}}$$

Quand n est « grand », disons 10 000, alors on obtient :

$$f(10\,000) = 10\,000+1+0,0001+0,00000000075-0,0000000000000765+ \text{ peanuts}$$

Tous les termes après n comptent pour du beurre quand n est « grand ». Donnons une définition pour plus de clarté :

Définition 2 - 1

« Grand O »
Soit f et g deux fonctions de $\mathbb{N}$ dans $\mathbb{R}$. On dit que f est un « grand O » de g et on note $f=\mathrm{O}(g)$ ou $f(n)=\mathrm{O}(g(n))$ si, et seulement si, il existe une constante strictement positive C telle que $|f(n)| \leqslant \mathrm{C}|g(n)|$ pour tout $n \in \mathbb{N}$.

Dans l'exemple précédent, $\frac{1}{n} \leqslant \frac{1}{1} \times 1$ pour tout entier n supérieur à 1 donc $\frac{1}{n}=\mathrm{O}(1)$.
De même, $\frac{75}{n^2} \leqslant \frac{75}{1^2} \times 1$ donc $\frac{75}{n^2}=\mathrm{O}(1)$ mais on peut dire mieux : $\frac{75}{n^2} \leqslant \frac{75}{1} \times \frac{1}{n}$ et ainsi on prouve que $\frac{75}{n^2}=\mathrm{O}\left(\frac{1}{n}\right)$.
En fait, un grand O de g est une fonction qui est au maximum majorée par un multiple de g.
Point de vue algorithmique, cela nous donne un renseignement sur la complexité au pire.
Pour le tri fusion, on obtient pour $n>1$:

$$\mathrm{K}(n) \leqslant 4nc\left(\lfloor \log_2(n) \rfloor + 1\right) \leqslant 4nc\log_2(n)\left(1+\frac{1}{\log_2(n)}\right) \leqslant 8cn\log_2(n)$$

On en déduit que $\mathrm{K}(n)=\mathrm{O}(n\log_2(n))$: la complexité est au pire en $n\log_2(n)$, c'est ce qui nous importe. Le rôle des constantes est accessoire car chercher trop de précisions serait illusoire : l'« oubli » de ces constantes correspond en fait au différences entre langages, processeurs, etc.
On peut cependant faire mieux avec le tri fusion car on a aussi une minoration.
C'est le moment d'introduire une nouvelle définition :

Définition 2 - 2

« Grand Oméga »
Soit f et g deux fonctions de $\mathbb{R}$ dans lui-même. On dit que f est un « grand Oméga » de g et on note $f=\Omega(g)$ ou $f(n)=\Omega(g(n))$ si, et seulement si, il existe une constante strictement positive C telle que $|f(n)| \geqslant \mathrm{C}|g(n)|$ pour tout $n \in \mathbb{N}^*$.

Remarque

Comme Ω est une lettre grecque, on peut, par esprit d'unification, parler de « grand omicron » au lieu de « grand O »...

Remarque

$f = \Omega(g) \iff g = O(f)$...

On montre donc facilement pour le tri fusion que $K(n) = \Omega(n\log_2(n))$ grâce à l'inégalité $2cn\lfloor\log_2(n)\rfloor \leqslant K(n)$. Ainsi on a dans ce cas en même temps $f = O(n\log_2(n))$ et $f = \Omega(n\log_2(n))$: c'est encore plus précis et nous incite à introduire une nouvelle définition :

Définition 2 - 3

« Grand Théta »

$$f = \Theta(g) \iff \begin{cases} f = O(g) \\ f = \Omega(g) \end{cases}$$

Le coût de l'algorithme se trouve donc coincé entre deux valeurs de même ordre. On peut ainsi dire que la complexité du tri fusion est en $n\log_2(n)$ ce qui est *souvent* mieux que le tri par insertion dont la complexité *au pire* est en n^2. Mais attention ! *Au pire* ne signifie pas *toujours* : si la liste est déjà triée, le tri par insertion est plus économique.
Voici maintenant une petite table pour illustrer les différentes classes de complexité rencontrées habituellement :

coût \ n	100	1000	10^6	10^9
$\log_2(n)$	≈ 7	≈ 10	≈ 20	≈ 30
$n\log_2(n)$	≈ 665	$\approx 10\,000$	$\approx 2\cdot 10^7$	$\approx 3\cdot 10^{10}$
n^2	10^4	10^6	10^{12}	10^{18}
n^3	10^6	10^9	10^{18}	10^{27}
2^n	$\approx 10^{30}$	$> 10^{300}$	$> 10^{10^5}$	$> 10^{10^8}$

Gardez en tête que l'âge de l'Univers est environ de 10^{18} secondes...
Il existe également deux autres « comparateurs » que l'on utilise peu en algorithmique mais qui peuvent s'avérer utiles dans d'autres domaines.

Définition 2 - 4

« Petit o »
$f = o(g)$ si, et seulement si, pour toute constante positive ε, il existe un entier n_0 tel que, pour tout $n \geqslant n_0$, $|f(n)| \leqslant \varepsilon |g(n)|$

On dit alors souvent que f est *négligeable* devant g.
Contrairement au grand O, la majoration doit se faire quelque soit la constante ε et non pas seulement pour une constante arbitraire.

Définition 2 - 5

Fonctions équivalentes
$f(n) \sim g(n) \iff f(n) = g(n) + o\big(g(n)\big)$

Voici quelques propriétés fort utiles que nous démontrerons à l'occasion :

Propriétés 2 - 1

Manipulation des O

- $O(f) + O(g) = O(f+g)$;
- $f = O(f)$;
- $k \cdot O(f) = O(f)$ si k est une constante ;
- $O\big(O(f)\big) = O(f)$;
- $O(f) \cdot O(g) = O(f \cdot g)$;
- $O(f \cdot g) = f \cdot O(g)$.

Par exemple, $2n^3 + 3n^2 + 5n = O(n^3) + O(n^3) + O(n^3) = O(n^3)$.
Voici quelques résultats que nous ne démontrerons pas mais qui s'avère utile de connaître :

Approximations asymptotiques quand *n* tend vers l'infini ou quand *x* tend vers 0

$$n! = \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)\left(1+\frac{1}{12n}+\frac{1}{288n^2}+\mathrm{O}\frac{1}{n^3}\right)$$
$$\mathrm{e}^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\mathrm{O}(x^5)$$
$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\mathrm{O}(x^5)$$
$$\frac{1}{1-x} = 1+x+x^2+x^3+x^4+\mathrm{O}(x^5)$$
$$(1+x)^\alpha = 1+\alpha x+\binom{\alpha}{2}x^2+\binom{\alpha}{3}x^3+\binom{\alpha}{4}x^4+\mathrm{O}(x^5)$$

3 L'infini à portée de main

3 1 Achille et la tortue

Le paradoxe suivant a été imaginé par ZÉNON D'ÉLÉE (490-430 Avant JC). Achille fait une course avec la tortue. Il part 100 mètres derrière la tortue, mais il va dix fois plus vite qu'elle. Quand Achille arrive au point de départ de la tortue, la tortue a parcouru 10 mètres. Pendant qu'Achille parcourt ces 10 mètres, la tortue a avancé d'un mètre. Pendant qu'Achille parcourt ce mètre, la tortue a avancé de 10cm... Puisqu'on peut réitérer ce raisonnement à l'infini, Zénon conclut qu'Achille ne peut pas dépasser la tortue...
Pour étudier ce problème, nous aurons besoin d'un résultat intermédiaire.

3 2 Inégalité de Bernoulli

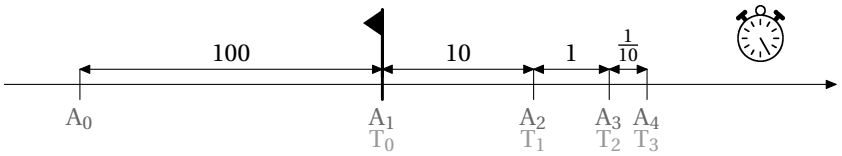
Il est assez simple de démontrer par récurrence, par exemple, que pour tout entier $n > 1$ et tout réel $x > -1$, on a :

$$(1+x)^n > 1+nx$$

Qu'en déduisez-vous au sujet de la limite des suites de terme général q^n ?

3 3 Le retour d'Achille

Voici la situation :



Intéressons-nous d'abord à la distance Achille-Tortue. Notons d_n la distance :

$$d_n = \mathrm{T}_n - \mathrm{A}_n = \mathrm{T}_n - \mathrm{T}_{n-1} = \frac{1}{10}(\mathrm{T}_{n-1} - \mathrm{A}_{n-1}) = \frac{1}{10}d_{n-1}$$

La suite (d_n) est donc géométrique de raison $1/10$ et de premier terme 100. On en déduit que $d_n = 100 \times \left(\frac{1}{10}\right)^{n-1}$.
Or $\left|1/10\right| < 1$, donc $\lim\limits_{n\to+\infty}\left(1/10\right)^{n-1} = 0 = \lim\limits_{n\to+\infty} d_n$.
Ainsi Achille va rattrapper la tortue, mais au bout d'une infinité de trajets : pour le Grec Zénon, la notion d'infini étant « au-delà du réel » ; pour lui Achille ne rattrapera jamais la tortue, ce qui est absurde.
Intéressons-nous plutôt à la durée du trajet d'Achille pour atteindre la tortue, en supposant sa vitesse constante et égale à v.
Notons $t_1 = \frac{d_1}{v} = \frac{100}{v}$, $t_2 = \frac{d_2}{v} = \frac{1}{10}\frac{d_1}{v} = \frac{1}{10}t_1$, etc. La suite (t_n) est donc géométrique de raison $\frac{1}{10}$ de premier terme $t_1 = 100/v$, d'où $t_n = \frac{100}{v}\left(\frac{1}{10}\right)^{n-1}$. La durée du trajet est alors :

$$\begin{aligned}\tau_n &= t_1 + t_2 + \cdots + t_n \\ &= \frac{100}{v}\left(1+\frac{1}{10}+\left(\frac{1}{10}\right)^2+\cdots+\left(\frac{1}{10}\right)^{n-1}\right) \\ &= \frac{1000}{9v}\left(1-\left(\frac{1}{10}\right)^n\right)\end{aligned}$$

Nous venons de voir qu'Achille atteindra la tortue quand n tend vers $+\infty$. Or nous savons calculer $\lim\limits_{n\to+\infty}\tau_n = \frac{1000}{9v}$ qui est un nombre fini. Achille va donc effectuer cette infinité de trajets en un temps fini égal à $1000/9v$.

Zénon pensait qu'une somme infinie de termes strictement positifs était nécessairement infinie, d'où le paradoxe à ses yeux. Il a fallu des siècles à l'esprit humain pour dépasser cette *limite*.
Vous pouvez donc prouver le petit théorème suivant :

Théorème 2 - 1

Série géométrique
La suite de terme général

$$S_n = \sum_{k=0}^{n} q^n$$

converge si, et seulement si, $|q| < 1$. Dans ce cas :

$$S = \lim_{n\to+\infty} S_n = \frac{1}{1-q}$$

4 Calcul différentiel et informatique

4 1 Le problème

Rappelons la définition que vous connaissez bien :

Définition 2 - 6

Dérivabilité
Soit f une fonction numérique définie sur un intervalle I de $\mathbb{R}$.
Soit $a \in$ I. On dit que f est dérivable en a si, et seulement si, le rapport :

$$\tau_a(h) = \frac{f(a+h)-f(a)}{h}$$

admet une limite finie lorsque a tend vers 0.
Dans ce cas on note $f'(a)$ cette limite.

Rappelons que $f'(a)$ est aussi le coefficient directeur de la tangente à la courbe représentative de f au point d'abscisse a.

Théorème 2 - 2

Si f est dérivable en a avec $f'(a) \neq 0$, alors

$$f(x) - f(a) \underset{a}{\sim} f'(a) \times (x-a)$$

Vous pourrez le démontrer en TD.

Définition 2 - 7

Dérivée $k^{ème}$
Si f' est dérivable sur I on note f'' ou $f^{(2)}$ la dérivée de f' et on l'appelle la dérivée seconde de f. De même si f'' est dérivable sur I, f''' ou $f^{(3)}$, la dérivée de f'', est la dérivée troisième de f. Plus généralement on note $f^{(k)}$ la dérivée $k^{ième}$ de f avec la convention $f^{(0)} = f$, $f^{(k+1)} = \left(f^{(k)}\right)'$.

Définition 2 - 8

Fonctions de classe $\mathcal{C}^n$, de classe $\mathcal{C}^\infty$
On dit que f est de classe $\mathcal{C}^n$, $n \in \mathbb{N}$, sur I ssi f est n fois dérivable et ses n dérivées sont continues sur I. On remarquera que si $f^{(n+1)}$ existe sur I alors f est de classe $\mathcal{C}^n$ sur I qui s'écrit $f \in \mathcal{C}^n(\text{I})$. On dit que f est de classe $\mathcal{C}^\infty$ sur I pour exprimer que f est indéfiniment dérivable et, a fortiori, que toutes ses dérivées sont continues.

On peut alors chercher à obtenir une approximation numérique de la dérivée de certaines fonctions avec un ordinateur :

Python
```
def diff(f,h):
    def fp(x):
        return (f(x+h) - f(x))/h
    return fp
```

Notez bien qu'il s'agit ici de programmation qui à une fonction et un réel associe une fonction.
Si on veut connaître le nombre dérivé de $x \mapsto x^2$ en 1, on obtient :

Python

```
>>> def g(x):
    return x**2

>>> diff(g,1e-10)(1)
2.000000165480742
```

Notez bien le double parenthésage !
Nous allons maintenant essayer d'enchaîner les dérivations :

Python

```
def diffn(f,h,n):
    if n == 0:
        return f
    else:
        return diff(diffn(f,h,n-1),h)
```

Essayons avec l'exponentielle dont la dérivée est assez simple puisque c'est elle-même. On ne devrait pas noter de différences au fur et à mesure des dérivations :

Python

```
>>> from math import exp
>>> diff(exp,1e-5)(0)
1.000005000006965
>>> diffn(exp,1e-5,2)(0)
1.0000111849706173
>>> diffn(exp,1e-5,3)(0)
0.8881784197001251
>>> diffn(exp,1e-5,4)(0)
22204.46049250314
>>> diffn(exp,1e-5,5)(0)
-6661338147.75094
```

Ouh la la !... La naïveté ne paie pas en informatique ! Il va falloir mettre le nez dans le processeur et faire un peu de mathématiques pour éviter d'écrire d'énormes bêtises en seulement quatre tours de récursion.

Exercices

Exercice 2 - 1 Domination

Prouvez les propriétés 1-1 du cours.

Exercice 2 - 2 Complexité

Voici deux algorithmes :

```
Fonction algo1(a,n : entiers naturels) : entier naturel
Début
    Si n=0 Alors
        Retourner 1
    Sinon
        Retourner a*algo1(a,n-1)
    FinSi
Fin
```

```
Fonction algo2(a,n : entiers naturels) : entier naturel
Variable
i,r :
Début
    r ← 1
    Pour i variantDe 1 Jusque n Faire
        r ← a*r
    FinPour
    Retourner r
Fin
```

Que calculent-ils ? Quelle est leur complexité ?
En voici un troisième :

```
Fonction algo3(a,n : entiers naturels) : entier naturel
Début
    Si n=0 Alors
        Retourner 1
    Sinon
        Si n=1 Alors
            Retourner a
        Sinon
            Si n est pair Alors
                Retourner algo3(a*a,n/2)
            Sinon
                Retourner a*algo3(a*a,(n-1)/2)
            FinSi
        FinSi
    FinSi
Fin
```

Que calcule-t-il ? Quelle est sa complexité ?
Comparez le temps de calcul de algo3(1 000 000 000) et de algo1(1 000 000 000).

Exercice 2 - 3 Tapis de Sierpinski

Monsieur SIERPINSKI avait ramené d'un voyage en Orient un tapis carré de 1 mètre de côté dont il était très content. Jusqu'au jour où les mites s'introduisirent chez lui.
En 24 heures, elles dévorèrent dans le tapis un carré de côté trois fois plus petit, situé exactement au centre du tapis. En constatant les dégats, Monsieur Sierpinski entra dans une colère noire ! Puis il se consola en se disant qu'il lui restait huit petits carrés de tapis, chacun de la taille du carré disparu. Malheureusement, dans les 12 heures qui suivirent, les mites avaient attaqué les huit petits carrés restants : dans chacun, elles avaient mangé un carré central encore trois fois plus petit. Et dans les 6 heures suivantes elles grignotèrent encore le carré central de chacun des tout petits carrés restants. Et l'histoire se répéta, encore et encore ; à chaque étape, qui se déroulait dans un intervalle de temps deux fois plus petit que l'étape précédente, les mites faisaient des trous de taille trois fois plus petite...

1. Faire des dessins pour bien comprendre la géométrie du tapis troué. Calculer le nombre total de trous dans le tapis de Monsieur Sierpinski après n étapes. Calculer la surface S_n de tapis qui n'a pas encore été mangée après n étapes. Trouver la limite de la suite $(S_n)_{n\geq 0}$. Que reste-t-il du tapis à la fin de l'histoire ?

2. Calculer la durée totale du festin « mitique »...

3. Proposez une animation graphique avec le langage que vous voulez pour illustrer le grignotage.

Exercice 2 - 4 La bille qui rebondit

Vous aurez besoin d'utiliser quelques lois physiques du programme de Terminale qui font partie de la culture générale d'un scientifique :

- En chute libre verticale, l'altitude z suit la loi $z(t) = -\frac{1}{2}gt^2 + v_0 t + z_0$
- la vitesse suit la loi $v(t) = -gt + v_0$ en coordonnées algébriques
- Théorème de l'énergie cinétique et de la conservation de l'énergie :

$$\mathcal{E}_C(B) - \mathcal{E}_C(A) = \frac{1}{2}mv_B^2 - \frac{1}{2}mv_A^2 = -mg(z_B - z_A)$$

1. Une balle part d'une certaine hauteur h_0 au dessus du sol (sans vitesse initiale). Combien de temps met-elle pour arriver sur le sol (négliger les frottements) ? Quelle est son énergie cinétique lorsqu'elle arrive au niveau du sol ?

2. On modélise le rebond de la façon suivante : lorsque la balle rebondit elle perd une certaine proportion p de son énergie cinétique (par exemple $p = 10\%$). Étant partie de la hauteur h_0, à quelle hauteur h_1 va-t-elle remonter ? Quelle est la durée t_0 entre les deux premiers rebonds ?

3. Combien de fois la balle rebondit-elle ? Pendant combien de temps rebondit-elle ?

4. Question subsidiaire : vous connaissez le bruit d'une bille qui rebondit, avec des rebonds de plus en plus rapprochés. Imaginez maintenant une balle qui rebondit, non plus selon le modèle ci-dessus, mais selon un autre loi. Par exemple la durée du n-ième rebond est donné par $1/n$. Que va-t-on entendre ?

5. Proposez un programme dans le langage que vous voulez qui permette de voir la bille rebondir et un autre qui permette d'entendre une série rebondir...

Exercice 2 - 5 Dérivées successives

Calculez les dérivées d'ordre n des fonctions suivantes :

1. $x \mapsto e^x$ **2.** $x \mapsto \frac{1}{1+x}$ **3.** $x \mapsto \frac{1}{1-x}$ **4.** $x \mapsto \ln(1+x)$ **5.** $x \mapsto \cos(x)$

Exercice 2 - 6 Dérivée d'une composée

Calculez la dérivée de la fonction $x \mapsto \ln(\ln(\ln(x)))$; Faites de même avec $x \mapsto \sin\left(\sqrt{\sqrt{\ln(\cos(x))}}\right)$.

TD N°

3

Approximations polynomiales

Dans les années 1960, Pierre BÉZIER, ingénieur chez Renault, définit une méthode permettant de modéliser des carrosseries de voitures : avec quelques données *discrètes* il arrive à représenter une courbe *continue* en utilisant des polynômes qui définissent de nouvelles courbes faciles à paramétrer et qui coïncident avec les courbes initiales en un nombre *discret* de points. Cette méthode a été reprise en informatique par exemple pour la construction de fontes de caractères, pour le dessin vectoriel sur des logiciels comme Blender, The Gimp, etc.

1 Formules de Taylor

Nous ne chercherons pas à démontrer les formules suivantes. ${\mathcal{P}_{\mathcal{B},\mathcal{B}_2}}^{-1}$

1 1 Formule de Taylor-Lagrange

Théorème 3 - 1

Si f est de classe $\mathcal{C}^n$ sur $[a,b]$ et si $f^{(n+1)}$ existe sur $]a,b[$ alors il existe au moins un réel $c \in]a,b[$ tel que :

$$f(b) = \sum_{k=0}^{n} \frac{(b-a)^k}{k!} f^{(k)}(a) + \frac{(b-a)^{n+1}}{(n+1)!} f^{(n+1)}(c)$$

On dit que l'on a écrit la formule de Taylor Lagrange à l'ordre n.

Il est souvent pratique de réécrire cette formule en posant $b = a+h$. Le nombre c appartient donc à l'intervalle $]a, a+h[$. Il existe donc un réel $\theta \in]0,1[$ tel que :

$$f(a+h) = \sum_{k=0}^{n} \frac{h^k}{k!} f^{(k)}(a) + \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(a+\theta h)$$

1 2 Formule de Taylor-Mac Laurin

Dans la formule de Taylor-Lagrange on remplace b par x et a par 0. On obtient :

Théorème 3 - 2

$$f(x) = \sum_{k=0}^{n} \frac{x^k}{k!} f^{(k)}(0) + \frac{x^{n+1}}{(n+1)!} f^{(n+1)}(\theta x)$$

1 3 Formule de Taylor avec reste intégral

Si f est de classe $\mathcal{C}^{n+1}$ sur I contenant a :

Théorème 3 - 3

$$\forall x \in \mathrm{I}, f(x) = \sum_{k=0}^{n} \frac{(x-a)^k}{k!} f^{(k)}(a) + \int_a^x \frac{(x-t)^n}{n!} f^{(n+1)}(t)\,dt$$

1 4 Formule de Taylor-Young.

Si f est de classe $\mathcal{C}^n$ sur un intervalle I contenant a alors :

Théorème 3 - 4

$$\forall x \in \mathrm{I}, f(x) = \sum_{k=0}^{n} \frac{(x-a)^k}{k!} f^{(k)}(a) + o\left((x-a)^n\right)$$

2 Développements limités

2 1 Voisinage

$a \in \mathbb{R}$, nous appellerons voisinage de a toute partie de $\mathbb{R}$ contenant un intervalle ouvert contenant a. L'intervalle $]a-\alpha, a+\alpha[$, avec $\alpha \in \mathbb{R}^{+*}$, est un voisinage de a, nous le noterons $\mathcal{V}_a$ et $\mathcal{V}_a - \{a\}$ sera noté $\mathcal{V}_a^*$. Un voisinage à droite de a est un intervalle du type $]a, a+\alpha[$ et $]a-\alpha, a[$ est un voisinage à gauche de a. Dans ce qui suit nous ne distinguerons pas ces types de voisinages, en conséquence $\mathcal{V}_a^*$ pourra tout aussi bien aussi désigner un voisinage à droite ou à gauche, lors des applications, le contexte permettra de les distinguer. Un voisinage de $+\infty$ est un voisinage du type $[A,+\infty[$ avec A réel positif aussi grand que l'on veut que nous pourrons noter $\mathcal{V}_{+\infty}^*$; $]-\infty;-A]$ est un voisinage de $-\infty$.

2 2 Définition

Définition 3 - 1

On dit que f, définie sur $\mathcal{V}_a$, admet un développement limité d'ordre $n \in \mathbb{N}$ au voisinage de a (on écrit en abrégé : « f admet un $DL_n V(a)$ ») s'il existe un polynôme $P_n \in \mathbb{R}_n[X]$ (ensemble des polynômes de degré inférieur ou égal à n) et une fonction ε vérifiant :

$$\begin{cases} \forall x \in \mathcal{V}_a, f(x) = P_n(x-a) + (x-a)^n \varepsilon(x) \\ \varepsilon(a) = 0 \\ \lim\limits_{x \to a} \varepsilon(x) = 0 \end{cases}$$

2 3 Visualisation de l'approximation avec Sage

Nous allons créer une procédure Sage qui donnera la partie polynomiale d'un DL en utilisant l'algorithme suivant :

Fonction `DL(f(x),n` : *une expression, un entier*`)` : *une expression polynomiale*
Début
  `Der` ← `f (sous forme de fonction!)`
  `Pol` ← `0`
  Pour `k` **variantDe** `0` **Jusque** `n` **Faire**
    `Pol` ← `Pol+Der(0)`$\times \frac{x^k}{k!}$
    `Der` ← `fonction dérivée de Der`
  FinPour
  Retourner `Pol`
Fin

Ce qui donne en Sage :

Sage

```
def DL(F,n):
    var('x')
    f(x) = F
    Der = f
    Pol = 0
    for k in range(n+1):
        Pol += Der(0)*x**k/factorial(k)
        Der = Der.derivative()
    return Pol
```

Par exemple :

Sage

```
sage: DL(exp(x),5)
1/120*x^5 + 1/24*x^4 + 1/6*x^3 + 1/2*x^2 + x + 1
sage: DL(log(1+x),5)
1/5*x^5 - 1/4*x^4 + 1/3*x^3 - 1/2*x^2 + x
```

On peut utiliser la fonction **DL** pour avoir un ordre de grandeur de l'approximation pour différentes fonctions et différentes valeurs de x. On utilisera à bon escient la méthode **subs** :

Sage

```
sage: DL(exp(x),5).subs(x == 0.00001)
1.00001000005000
sage: [DL(log(1+x),5).subs(x==10.**(-k)) - log(1+10.**(-k)) for k in range(5) ]
[0.0901861527733880, 1.53529008409259e-7, 1.65241084704171e-13, 1.10154940724527e-16, 1.10182045778839e-17]
```

Remarque

Il existe une commande Sage qui donne directement le développement limité d'une fonction :

Sage

```
sage: taylor(log(1+x),x,0,5)
1/5*x^5 - 1/4*x^4 + 1/3*x^3 - 1/2*x^2 + x
```

mais c'est moins drôle...

Nous allons maintenant créer des petits dessins animés qui permettent de visualiser la convergence progressive des DL vers la fonction initiale.

Sage

```
def anim_DL(f,p,xm,xM,ym,yM,d):
    P = plot(f,rgbcolor='red',xmin=xm,xmax=xM,ymin=ym,ymax=yM,thickness=3,
                                                        linestyle='--')
    a = animate([[P,DL(f,k)] for k in range(1,p)],xmin=xm,xmax=xM,ymin=ym,ymax=yM)
    a.show(delay=d) # délai en 100e de seconde entre deux images.
```

Tentez :

Sage

```
sage: anim_DL(log(1+x),20,-1,2,-2,2,50)
sage: anim_DL(cos(x),30,-4*pi,4*pi,-1.1,1.1,50)
```

Exercices

Exercice 3 - 1

Écrire la formule de Taylor-Mac Laurin et la formule de Taylor-Young à l'ordre n pour les fonctions suivantes et dans un voisinage de 0 :

1. $f(x) = e^x$
2. $f(x) = \dfrac{1}{1+x}$
3. $f(x) = \dfrac{1}{1-x}$
4. $f(x) = \ln(1+x)$
5. $f(x) = \sin(x)$
6. $f(x) = \sqrt{1+x}$, $n = 4$

Exercice 3 - 2

Donner un $DL_3V(0)$ pour les fonctions suivantes :

1. $f_1(x) = \ln(1+x), g(x) = f_1(x^2)$
2. $f_2(x) = e^x, h(x) = f_2(-2x)$
3. $f_3(x) = f_1(x)f_2(x)$
4. $f_4(x) = f_1(x) + f_2(x)$

Exercice 3 - 3 Polynôme interpolateurs de Lagrange

D'un point de vue abstrait, on pourrait se contenter de dire que si on ne connaît que quelques points de la courbe représentative d'une fonction inconnue, on cherche à approcher cette fonction par une fonction polynomiale passant par ces points. Intuitivement, on peut penser que plus on aura de points de contrôles, meilleures sera l'approximation mais les mathématiques vont venir une nouvelle fois contredire notre intuition.
D'un point de vue plus concret, on voudrait reconstituer un organe humain à partir de quelques mesures prises en imagerie médicale ou reconstituer une couche géologique à partir de mesures sismiques : on est alors amené à reconstituer une formes à partir de nuages de points, en 3D cette fois.
Ces problèmes sont très courants en informatique. Nous nous contenterons bien sûr ici d'un survol de quelques méthodes très simples dans des cas théoriques eux aussi assez triviaux.

On considère n points de coordonnées $(x_1, y_1)\dots(x_n, y_n)$ On veut trouver un polynôme P tel que $P(x_i) = y_i$ pour tout $i \in [\![1\,;\,n]\!]$.
Une petite activité mathématique consiste à prouver que l'unique polynôme de degré $n-1$, solution du problème, est défini par

$$P(x) = \sum_{k=1}^{n} y_k \mathcal{L}_k(x) \quad \text{avec} \quad \mathcal{L}_k(x) = \frac{\prod_{i=1, i\neq k}^{i=n} (x - x_i)}{\prod_{i=1, i\neq k}^{i=n} (x_k - x_i)}$$

Bon, ça fait peur comme ça, mais ce n'est pas si terrible...
Déterminez par exemple « à la main » le polynôme de Lagrange passant par les points $A(1,2)$, $B(2,-1)$ *et* $(3,2)$. *Vérifiez qu'on obtient bien* $P(1) = 2$, $P(2) = -1$ *et* $P(3) = 2$.
Avec Sage, c'est évidemment plus facile...

Sage

```
R = PolynomialRing(QQ, 'x') # on travaille dans Q[X]
f = R.lagrange_polynomial([(1,2),(2,-1),(3,2)])
```

On peut même visualiser la courbe avec les points de contrôle :

Sage

```
def lagrange_plot(L):
    R = PolynomialRing(QQ, 'x')
    f = R.lagrange_polynomial(L)
    G = Graphics()
    G = plot(f(x),xmin=L[0][0]-1,xmax=L[-1][0]+1)
    for k in range(len(L)):
        G += point(L[k])
    G.show()
```

On veut à présent visualiser l'approximation du graphe d'une fonction par des polynômes de Lagrange successifs en augmentant les points de contrôle.
On va par exemple prendre une subdivision régulière de l'intervalle et on augmentera le nombre de points de contrôles pour observer si l'approximation s'en trouve améliorée.
Construisez une animation de l'approximation de la fonction $x \mapsto \frac{1}{1+x^2}$ *sur* $[-5\,;5]$ *avec comme séries successives d'abscisses de points de contrôle des subdivisions régulières de l'intervalle* $[-5\,;5]$.
Faites de même pour $x \mapsto ln(1+x)$ *en adaptant les intervalles.*
On a bien sûr envie de généraliser : créez une fonction Sage idoine **`lagrange_fonc((F,n,xm,xM,ym,yM))`**.
Observez ce que cela donne :

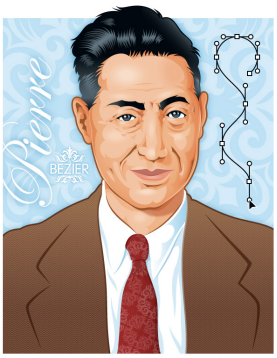
Sage

```
sage : lagrange_fonc(1/(1+t^2),40,-5,5,-0.2,1.2)
sage : lagrange_fonc(ln(1+t),20,0,5,-2,3)
```

Bref, ce n'est pas aussi magique qu'il n'y paraissait : plus on augmente le nombre de points de contrôle, plus les « effets de bord » deviennent gênants.
Vous prouverez peut-être un jour que si f est définie sur un intervalle I, si la suite des dérivées successives de f est uniformément bornée et si l'on fait tendre le nombre de points de contrôle vers $+\infty$, alors la suite des interpolateurs de Lagrange de f converge uniformément vers f sur tout segment $[a;b]$ inclus dans I lorsque n tend vers $+\infty$.
Ça n'a malgré tout que peu d'intérêt car ces conditions sont trop restrictives et de toute façon, la formule de Taylor-Lagrange nous donne ce qu'il faut pour de telles fonctions.
Il y a bien sûr des approximations plus intéressantes

- par les polynômes de Bernstein définis par $B_n(x) = \sum_{k=0}^{n} \binom{n}{k} f\left(\frac{k}{n}\right) x^k (1-x)^{n-k}$ pour des fonctions continues sur un segment
- par des polynômes trigonométriques dans des cas encore plus généraux en étudiant les séries de Fourier...

Exercice 3 - 4 Courbes de Bézier



Dans les années 60, les ingénieurs Pierre BÉZIER et Paul DE CASTELJAU travaillant respectivement chez Renault et Citroën, réfléchissent au moyen de définir de manière la plus concise possible la forme d'une carrosserie.
Le principe a été énoncé par BÉZIER mais l'algorithme de construction par son collègue de la marque aux chevrons qui n'a d'ailleurs été dévoilé que bien plus tard, la loi du secret industriel ayant primé sur le développement scientifique...
Pour la petite histoire, alors que Pierre BÉZIER (diplômé de l'ENSAM et de SUPÉLEC), à l'origine des premières machines à commandes numériques et de la CAO ce qui n'empêcha pas sa direction de le mettre à l'écart : il se consacra alors presque exclusivement aux mathématiques et à la modélisation des surfaces et obtint même un doctorat en 1977.

Paul DE CASTELJAU était lui un mathématicien d'origine, ancien élève de la Rue d'ULM, qui a un temps été employé par l'industriel automobile.
Aujourd'hui, les courbes de Bézier sont très utilisées en informatique.
Une Courbe de Bézier est une courbe paramétrique aux extrémités imposées avec des points de contrôle qui définissent les tangentes à cette courbe à des instants donnés.

Algorithme de Casteljau

Soit t un paramètre de l'intervalle $[0,1]$ et P_1, P_2 et P_3 les trois points de contrôle.
On construit le point M_1 barycentre du système $\{(P_1, 1-t), (P_2, t)\}$ et M_2 celui du système $\{(P_2, 1-t), (P_3, t)\}$.
On construit ensuite le point M, barycentre du système $\{(M_1, 1-t), (M_2, t)\}$.
Exprimez M comme barycentre des trois points P_1, P_2 et P_3.
Faites la construction à la main avec $t = 1/3$ par exemple.
Voici ce que cela donne en Python :

Python

```
from pygal import *

def bezier1(P):
    bezier = XY()
    bezier.add('Vi', [tuple(P[0]),tuple(P[1])])
    bezier.add('Vf', [tuple(P[1]),tuple(P[2])])
    G = []
    for k in range(100):
        t = 0.01*k
        M = (1-t)**2*array(P[0]) + 2*t*(1-t)*array(P[1]) + t**2*array(P[2])
        G += [(M[0],M[1])]
    bezier.add('M(t)', G)
    return bezier
```

Nous allons assimiler les points M_1, M_2 et M à des courbes paramétrées.
Ainsi $M_1(t) = (1-t)P_1 + tP_2$, $M_2(t) = (1-t)P_2 + tP_3$ puis

$$M(t) = (1-t)M_1(t) + tM_2(t) = (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3$$

Vérifiez que $M'(t) = 2(M_2(t) - M_1(t))$. Comment l'interpréter ?

Avec 4 points de contrôle

Faites une étude similaire (« à la main ») avec 4 points de contrôle.

On pourra introduire les polynômes de Bernstein définis par :

$$\mathrm{B}_j^n(t) = \binom{n}{j} t^j (1-t)^{n-j}$$

et utiliser une représentation similaire aux arbres de probabilité.
Créez une procédure **`bern(j,n,t)`** qui calcule $\mathrm{B}_j^n(t)$.
Commentez le script suivant :

Python

```
def bezier3(P):
    bezier = XY()
    bezier.add('Vi', [tuple(P[0]),tuple(P[1])])
    bezier.add('Vf', [tuple(P[2]),tuple(P[3])])
    G = []
    for k in range(1,100):
        t = 0.01*k
        M = (1-t)**3*array(P[0]) + 3*(1-t)**2*t*array(P[1]) + 3*(1-t)*t**2*array(P[2]) + t**3*array(P[3])
        G += [(M[0],M[1])]
    bezier.add('M(t)', G)
    return [bezier,G]
```

et testez :

Python

```
bez = bezier3([[0,0],[-1,1],[0,3],[-1,2]])[0]
bez.render_to_file('./bezier3.svg')
```

Quel est le rôle des points de contrôle 2 et 3 ?

Courbe de Bézier du 3e degré avec un nombre quelconque de points de contrôle
Il est pratique de travailler avec des polynômes de degré trois pour avoir droit à des points d'inflexion.
Augmenter le nombre de points de contrôle implique a priori une augmentation du degré de la fonction polynomiale.
Pour remédier à ce problème, on découpe une liste quelconque en liste de listes de 4 points.
Cependant, cela est insuffisant pour obtenir un raccordement de classe $\mathcal{C}^1$ (pourquoi ? pourquoi est-ce important d'avoir un raccordement de classe $\mathcal{C}^1$?)
Pour assurer la continuité tout court, il faut que le premier point d'un paquet soit le dernier du paquet précédent.
Le dernier « vecteur vitesse » de la liste $[\mathrm{P}_1,\mathrm{P}_2,\mathrm{P}_3,\mathrm{P}_4]$ est $\overrightarrow{\mathrm{P}_3\mathrm{P}_4}$. Il faut donc que ce soit le premier vecteur vitesse du paquet suivant pour assurer la continuité de la dérivée.
Appelons provisoirement le paquet suivant $[\mathrm{P}'_1,\mathrm{P}'_2,\mathrm{P}'_3,\mathrm{P}'_4]$. On a d'une part $\mathrm{P}'_1 = \mathrm{P}_4$ et d'autre part $\overrightarrow{\mathrm{P}_3\mathrm{P}_4} = \overrightarrow{\mathrm{P}'_1\mathrm{P}'_2}$, i.e. $\mathrm{P}'_2 = \mathrm{P}_4 + \overrightarrow{\mathrm{P}_3\mathrm{P}_4}$.
On a donc $\mathrm{P}'_3 = \mathrm{P}_5$ et $\mathrm{P}'_4 = \mathrm{P}_6$.
Connaissant **`bezier3`**, construire une procédure qui trace une courbe de Bézier cubique avec un nombre quelconque de points de contrôle (on prendra un nombre pair de points pour se simplifier la vie).
Testez avec $[[1,1],[2,3],[5,5],[6,2],[7,7],[10,5],[10,2],[8,0],[4,0],[5,1]]$.
C'est bientôt la Saint-Valentin alors dessinez un cœur....

B-splines uniformes
Tout ceci est très beau mais il y a un hic : en changeant un point de contrôle, on modifie grandement la figure.
On considère m nœuds $t_0, t_1 n, \cdots, t_m$ de l'intervalle $[0,1]$.
Introduisons une nouvelle fonction :

$$\mathbf{S}(t) = \sum_{i=0}^{m-n-1} \mathbf{P}_i b_{i,n}(t)\ ,\ t \in [0,1]$$

les P_i étant les points de contrôle et les fonctions $b_{j,n}$ étant définies récursivement par

$$b_{j,0}(t) = \begin{cases} 1 & \text{si} \quad t_j \leqslant t < t_{j+1} \\ 0 & \text{sinon} \end{cases}$$

et pour $n \geqslant 1$

$$b_{j,n}(t) = \frac{t-t_j}{t_{j+n}-t_j} b_{j,n-1}(t) + \frac{t_{j+n+1}-t}{t_{j+n+1}-t_{j+1}} b_{j+1,n-1}(t).$$

On ne considérera par la suite que des nœuds équidistants : ainsi on aura $t_k = \frac{k}{m}$.
On parle de B-splines uniformes et on peut simplifier la formule précédente en remarquant également des invariances par translation.
À l'aide des formules précédentes, on peut prouver que dans le cas de 4 points de contrôles on obtient :

$$S(t) = \frac{1}{6}\Big((1-t)^3 P_0 + (3t^3 - 6t^2 + 4)P_1 + (-3t^3 + 3t^2 + 3t + 1)P_2 + t^3 P_3\Big)$$

Calculez $S(0)$, $S(1)$ puis $S'(0)$ et $S'(1)$: que peut-on en conclure ?
Reprenez l'étude faite avec les courbes de Bézier et comparez les résultats : obtiendrez-vous un plus joli cœur ?

TD N°

4

Intégration numérique et erreurs d'arrondis

À travers la recherche des décimales de π, nous plongerons au cœur du processeur pour comprendre comment il calcule et comment ne pas être surpris par quelques erreurs d'arrondis qui pourraient s'avérer catastrophiques...

1 Approximation de π et calcul approché d'intégrale au petit bonheur

On s'intéresse à l'intégrale sur $[0,1]$ de la fonction $f : t \mapsto \sqrt{1-t^2}$: quel est son rapport avec π ?
Nous utiliserons Python pour rester basique et mieux voir les problèmes. Nous pourrons cependant aller du côté de Sage pour avoir un beau graphique ou faire un calcul en précision infinie.

```
sage: P = plot(sqrt(1-x^2),xmin=0,xmax=1)
sage: P.show(aspect_ratio=1)
```

1.1 Méthode des rectangles

C'est a priori la plus grossière mais la plus simple à écrire. Petit rappel :

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{k=0}^{n} \frac{b-a}{n} f\left(a + k\frac{b-a}{n}\right) + \mathrm{E}_n$$

avec E_n l'erreur commise.

On pourrait avoir cette idée pour calculer cette approximation sur Python :

Python

```
def int_rec_droite(f,a,b,N):
    S = 0
    dt = (b-a)/N
    for k in range(N):
        S += f(a + k*dt)*dt
    return S
```

Écrivez la fonction jumelle **`int_rec_gauche(f,a,b,N)`**.
Il *semble* falloir dix millions d'itérations pour obtenir six bonnes décimales de π...

Python

```
In [1]: from math import sqrt,pi

In [4]: for k in range(8): print(pi-4*int_rec_gauche(f,0.,1.,10**k))
3.141592653589793
0.2370743273414746
0.021175621810747725
0.0020371866787654014
0.00020117597846525115
2.0037187828503278e-05
2.001175991139803e-06
2.0003720369032862e-07
```

1.1.1 Méthode des trapèzes

C'est déjà un peu plus précis, à vue d'œil. Créer une fonction **`int_trap(f,a,b,N)`**. et faites les mêmes tests.

Python

```
In [6]: for k in range(8): print(pi-4*int_trap(f,0.,1.,10**k))
-0.32250896154796127
-0.01081877967185152
-0.00034420431021464637
-1.0891323113604301e-05
-3.444348353198734e-07
-1.0892040602783482e-08
-3.4457015019029313e-10
-1.162758778150419e-11
```

On multiplie notre fonction par 4 et donc son erreur avec. Soyons plus malins...

Python

```
In [7]: for k in range(8): print(0.25*pi-int_trap(f,0.,1.,10**k))
-0.08062724038699032
-0.00270469491796288
-8.605107755366159e-05
-2.7228307784010752e-06
-8.610870882996835e-08
-2.7230101506958704e-09
-8.614253754757328e-11
-2.9068969453760474e-12
```

Pour plus de précision, on pourrait être tenté de prendre un plus petit arc de cercle, disons entre $\frac{\pi}{3}$ et $\frac{\pi}{2}$, en utilisant encore f. En effet, la fonction n'étant pas dérivable en 1, on a un peu peur que cela crée des perturbations. Et intuitivement, on se dit qu'en limitant l'intervalle d'intégration, on devrait limiter l'ampleur de l'approximation.
Avec la méthode `for` :

Python

```
In [9]: for k in range(8): print(pi/12-(int_trap(f,0,1/2,10**k)-sqrt(3)/8))
-0.005817179530668071
-6.0117319471197916e-05
-6.014041918356305e-07
-6.014064968251631e-09
-6.013917142055902e-11
-6.002975894148221e-13
-8.43769498715119e-15
-4.418687638008123e-14
```

Cela semble mieux fonctionner. Bizarre tout de même ce dernier résultat....
On a utilisé des segments de droite horizontaux, des segments de droite obliques...Et si l'on utilisait des segments de parabole : intuitivement, cela colle plus à la courbe, cela devrait être plus précis.

1 1 2 Méthode de Simpson

On interpole la courbe par un arc de parabole. On veut que cet arc passe par les points extrêmes de la courbe et le point d'abscisse le milieu de l'intervalle. Pour cela, on va déterminer les c_i tels que :

$$\int_a^b f(x)\mathrm{d}x = c_0 f(a) + c_1 f\left(\frac{a+b}{2}\right) + c_2 f(b)$$

soit exacte pour $f(x)$ successivement égale à 1, x et x^2.
Posons $h = b - a$ et ramenons-nous au cas $a = 0$. On obtient le système suivant :

$$\begin{cases} c_0 + c_1 + c_2 = h \\ c_1 + 2c_2 = h \\ c_1 + 4c_2 = \frac{4}{3}h \end{cases}$$

Résolvez-le et expliquez le résultat :

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Créez **`int_simps(f,a,b,N)`**.
On subdivise l'intervalle d'intégration et on utilise la méthode de SIMPSON sur chaque subdivision avec notre petit arc de cercle :

Python

```
In [17]: for k in range(8): print(pi/12-(int_simps(f,0,1/2,10**k)-sqrt(3)/8))
5.500957927112582e-05
6.664729046423901e-09
6.684097719755755e-13
3.885780586188048e-16
-1.7208456881689926e-15
-4.907185768843192e-14
4.030109579389318e-14
-6.844913524872709e-12
```

Étrange cette précision fluctuante....
On s'aperçoit qu'on est un peu comme un spécialiste de la tectonique des plaques prenant des photos d'une plage tous les mois à la même heure. Il peut tout à fait se produire que le montage des photos mette en évidence que le niveau de la mer descend et il pourra dire à la radio que les gaz à effet de serre n'ont aucune influence sur la fonte des glaces...

Finalement, il va falloir faire un peu plus de mathématiques et d'informatique et ne pas se contenter d'observer des résultats au petit bonheur.
Avant d'attaquer les mathématiques, il faut avoir à l'esprit que le processeur compte en base 2 et nous en base 10 et que son « zéro » vaut environ $2,2 \times 10^{-16}$: cela vient du fait que la mantisse a 53 bits ce qui correspond en gros à 16 chiffres décimaux. Il faut bien noter que ce n'est pas zéro mais une valeur nommée communément (pas seulement sur Python) **`epsilon`**.
Comme c'est l'ordinateur qui va compter, il faudrait plutôt chercher à le ménager et en tenir compte.

Il nous faut donc quand même regarder sous le capot pour comprendre la panne.
Le problème, c'est que nous ne travaillons pas en précision infinie. En chargeant le module **`sys`**, on a accès à la commande **`float_info.epsilon`** qui donne la différence entre **`1.0`** et le nombre en notation scientifique suivant le plus proche :

Python
```
In [18]: from sys import*
In [19]: float_info.epsilon
Out[19]: 2.220446049250313e-16
```

Eh oui, la largeur des pipe-lines du microprocesseur n'est pas infinie. Ses « réels » admettent des successeurs.
Mais attention, entre 0 et 1, les choses sont différentes :

Python
```
In [20]: e = float_info.epsilon
In [21]: e
Out[21]: 2.220446049250313e-16
In [22]: e/2
Out[22]: 1.1102230246251565e-16
In [23]: 1 + e/2
Out[23]: 1.0
In [24]: 1 + e/2 == 1
Out[24]: True
In [25]: 0 + e/2
Out[25]: 1.1102230246251565e-16
In [26]: e * 1e16
Out[26]: 2.220446049250313
In [27]: -1 - e/2
Out[27]: -1.0
In [28]: 1 - e/2
Out[28]: 0.9999999999999999
```

Cet **`epsilon`** n'est pas zéro, rappelons-le, mais détermine à partir de quand deux flottants seront considérés comme égaux par le système.
Utilisons à nouveau le test `pseudo_egal_float` introduit à la section précédente.

Python
```
def pseudo_egal_float(a, b):
    return abs(a - b) <= (float_info.epsilon * min(abs(a), abs(b)))
```

Si on n'y prête pas attention, on peut arriver à des résultats surprenants :

Python
```
In [30]: pseudo_egal_float(0.1 + 0.1 + 0.1 , 0.3)
Out[30]: True
In [31]: 0.1 + 0.1 + 0.1 == 0.3
Out[31]: False
In [32]: 3 * 0.1 == 0.3
Out[32]: False
In [33]: 4 * 0.1 == 0.4
Out[33]: True
In [34]: 10 + float_info.epsilon - 10
Out[34]: 0.0
In [35]: 1 + float_info.epsilon - 1
Out[35]: 2.220446049250313e-16
```

```
In [36]: 10 + 10*float_info.epsilon - 10
Out[36]: 1.7763568394002505e-15
In [37]: x = 0.1
In [38]: 3*x - 0.3
Out[38]: 5.551115123125783e-17
In [39]: 4*x - 0.4
Out[39]: 0.0
```

Rappelons également que le processeur est plus à l'aise avec les puissances de 2 car une multiplication par 2 ou une de ses puissances revient à un décalage dans son écriture binaire.
Ici, $0,25 = \frac{1}{4}$ en base 10 donc $\frac{1}{100} = 0,01$ en base 2.
Pour $0,1 = \frac{1}{10}$ en base 10, on obtient $\frac{1}{1010}$ en base 2 ; posons la division :

```
 1       | 1010
        ------
 10      | 0,00011
 100     |
 1000    |
 10000   |
- 1010   |
------   |
   1100  |
-  1010  |
-------  |
     10  |
```

On retrouve alors un reste précédent donc $\frac{1}{10}$ en base 10 admet en base 2 un développement périodique : **0,0 0011 0011 0011...**.
On peut alors savoir comment ces nombres sont codés sur un ordinateur disposant de 53 bits pour la mantisse, faire de même pour 0,3 et $3 \times 0,1$ et découvrir pourquoi **0.3-3*0.1** est non nul.
Pour un exposé brillant et complet sur le sujet, étudier les nombreux documents que William KAHAN, fondateur de la norme IEEE 754, a mis en ligne sur sa page web : .
Reprenons l'affinement successif de notre subdivision mais avec des nombres de subdivisions égaux à des puissances de 2 :

Python

```
In [60]: for k in range(13): print(pi/12 - (int_simps(f,0,1/2,2**k) - sqrt(3)/8))
5.500957927112582e-05
3.935118889741851e-06
2.5687273891294993e-07
1.624742479444663e-08
1.0185862153733183e-09
6.371086991308061e-11
3.9825920339353615e-12
2.490230244234226e-13
1.5376588891058418e-14
6.661338147750939e-16
-5.551115123125783e-17
-1.1657341758564144e-15
-7.771561172376096e-16
```

Tout a l'air de bien fonctionner jusqu'à 2^{10} mais ensuite, on arrive aux alentours de **epsilon** et cela commence à se détraquer informatiquement.
En affinant grossièrement à coups de puissances de 10, nous étions passés à côté du problème.
Comme quoi, agir intuitivement en mathématiques ou en informatique (ici : « plus on subdivise petit, meilleure sera la précision ») peut entraîner de graves erreurs...
La fonction est aussi à prendre en considération, puisqu'elle demande de soustraire à 1 un tout petit nombre :

Python

```
In [62]: f(1e-9)
Out[62]: 1.0
```

De plus, cela explique aussi les différences entre la « méthode **while** » et la « méthode **for** ».
Le test **while t+dt<=b** peut s'arrêter pour de mauvaises raisons. Par exemple :

Python

```
In [63]: 1 + 0.1 + 0.1 + 0.1 - 0.3 <= 1
Out[63]: False
```

La boucle peut ainsi s'arrêter inopinément, alors qu'on est loin de la précision demandée.
Précédemment, avec la méthode « gros sabots », nous n'avions pas vu de différence entre la méthode des trapèzes et la méthode de SIMPSON ce qui contredisait notre intuition. Observons à pas plus feutrés ce qui se passe et incluons les rectangles :

Python

```
In [66]:
print('-'*75)
print('{:22s} | {:22s} | {:22s}'.format('rectangle gauche', 'trapeze', 'simpson'))
print('-'*75)

for k in range(10):
    print("{:1.16e} | {:1.16e} | {:1.16e}".format(\
        pi/12 - (int_rec_gauche(f, 0, 1/2, 2**k)-sqrt(3)/8), \
        pi/12 - (int_trap(f, 0, 1/2, 2**k)-sqrt(3)/8), \
        pi/12 - (int_simps(f, 0, 1/2, 2**k)-sqrt(3)/8)))
```

```
---------------------------------------------------------------------------
rectangle gauche       | trapeze                | simpson
---------------------------------------------------------------------------
4.5293036853039759e-02 | -5.8171795306680707e-03 | 5.5009579271125819e-05
1.9737928661185844e-02 | -1.4896493887858187e-03 | 3.9351188897418510e-06
9.1241396362000127e-03 | -3.7497837725530836e-04 | 2.5687273891294993e-07
4.3745806294723244e-03 | -9.3912877730750743e-05 | 1.6247424794446630e-08
2.1403338758707591e-03 | -2.3488877122057605e-05 | 1.0185862153733183e-09
1.0584224993743230e-03 | -5.8728876539682062e-06 | 6.3710869913080614e-11
5.2627480586020514e-04 | -1.4682637225482686e-06 | 3.9825920339353615e-12
2.6240327106874517e-04 | -3.6706854433798952e-07 | 2.4902302442342261e-13
1.3101810126225910e-04 | -9.1767299481571030e-08 | 1.5376588891058418e-14
6.5463166981361010e-05 | -2.2941835209344674e-08 | 6.6613381477509392e-16
```

On distingue mieux cette fois la hiérarchie des méthodes selon leur efficacité.
Nous n'avons pas le choix : il est temps à présent d'invoquer les mathématiques pour préciser un peu ces observations, même si cela peut paraître violent de se lancer dans un raisonnement mathématique avec des calculs, des théorèmes, sans machine.
En fait, ces trois méthodes sont des cas particuliers d'une même méthode : on utilise une interpolation polynomiale de la fonction f de degré 0 pour les rectangles, de degré 1 pour les trapèzes et de degré 2 pour SIMPSON.

1 2 Quelques mots sur la manipulations des flottants

En 1985, une norme de représentations des nombres a été proposée afin, entre autre, de permettre de faire des programmes portables : il s'agit de la norme IEEE-754 (Standard for Binary Floating-Point Arithmetic).
En simple précision, un nombre est représenté sur 32 bits (en fait 33...) :

- le premier donne le signe ;
- un 1 implicite puis 23 bits pour la partie fractionnaire ;
- 8 bits pour l'exposant.

Les 24 bits qui ne sont ni le signe, ni l'exposant représentent la mantisse qui appartient à $[1,2[$.
Notons s_x le bit de signe, e_x les bits d'exposant, $m_x = 1 + f_x$ la mantisse avec f_x la partie fractionnaire.
Il y a deux zéros, -0 et $+0$ car tout nombre est signé. Par exemple :

- $s_{-0} = 1$;
- $b_{-0} = 00000000$
- $f_{-0} = 00000000000000000000000$

Il y a aussi deux infinis. Par exemple $s_{-\infty} = 1$, $e_{-\infty} = 11111111$ et $f_{-\infty} = 0$....
Il y a enfin le *Not a Number*, noté NaN, qui est codé comme $+\infty$ mais avec une partie fractionnaire non nulle.

La majorité des résultats troublants observés vient des erreurs d'arrondis, les deux principales étant l'*élimination* et l'*absorption*.
La première intervient lors de la soustraction de deux nombres très proches. Par exemple :

```
 1.10010010000111111011011
-1.10010010000110000000000
--------------------------
=0.00000000000111111011011
```

La mantisse est ensuite renormalisée pour devenir :

```
1.11111011011000000000000
```

Les zéros ajoutés à droite sont faux.

L'absorption intervient lorsqu'on additionne deux nombres d'ordre de grandeur très différents : les informations concernant le plus petit sont perdues.

```
 1.100100100001111100 01011
+                    1.00000001111
--------------------------------
=1.10010010000111110100101101111
```

Mais après normalisation, on a :

```
1.10010010000111110100101
```

Voici un exemple classique : le calcul de $\sum_{k=1}^{2^{15}} \frac{1}{k^2}$ dans un sens puis dans l'autre.

Python

```
def som_croissante(N):
    S=0
    for k in range(1,N+1):
        S+=1/k**2
    return S
```

Python

```
def som_decroissante(N):
    S=0
    for k in range(N,0,-1):
        S+=1/k**2
    return S
```

Alors :

Python

```
In [68]: som_croissante(2**15)
Out[68]: 1.6449035497357558
In [69]: som_decroissante(2**15)
Out[69]: 1.644903549735758
```

En fait, le résultat correct est :

Sage

```
sage: k=var('k')
sage: s=sum((1/k**2),k,1,2**15)
sage: s.n(digits=20)
1.6449035497357579868
```

Les conséquences de telles erreurs non prises en compte peuvent être plus graves qu'une séance de TP ratée.

Le 4 juin 1996 par exemple, la fusée Ariane 5 a explosé en vol, trente secondes après son décollage. Après enquête, il s'est avéré que la vitesse horizontale de la fusée par rapport au sol était calculée sur des flottants 64 bits puis convertie en entier signé 16 bits. Cette méthode avait été appliquée sur Ariane 4 avec succès car sa vitesse était plus faible donc tenait sur un entier 16 bits, mais ce n'était plus le cas pour Ariane 5... [a]

Un problème d'élimination dans l'horloge des missiles *Patriot* a causé également la mort de dizaines de personnes pendant la première guerre du Golfe [b].

a. Voir le rapport de la commission présidée par J.L. LIONS disponible à cette adresse : http ://www.ima.umn.edu/ arnold/disasters/ariane5rep.html ainsi que la page 22 d'un article de William KAHANE : http ://www.eecs.berkeley.edu/ wkahan/JAVA-hurt.pdf

b. Voir http ://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html

Remarque

Les erreurs présentées ici ne sont pas dues à Pythonet on les retrouve sur d'autres langages :

– avec CAML :

CAML

```
# 0.3-.3.*.0.1;;
- : float = -5.5511151231257827e-17
```

– avec SCILAB :

Scilab

```
-->0.3-3*0.1
 ans = - 5.551D-17
```

– avec giac/XCAS :

giac/XCAS

```
1>> 0.3-3*0.1
1.42108547152e-14
// Time 0
```

– avec MAXIMA :

Maxima

```
(%i1) 0.3-3*0.1;
(%o1) - 5.5511151231257827E-17
```

Recherche

On considère la suite u définie par $u_{n+1} = 4u_n - 1$, $u_0 = \frac{1}{3}$.
Calculez à la main les premiers termes de la suite.
Utilisez ensuite Python pour obtenir une approximation des 100 premiers termes.
Faites de même avec la suite v définie par $v_{n+1} = 3v_n - 1$, $v_0 = 0,5$.

2 Méthodes de Newton-Cotes et programmation objet

Voici une présentation plus professionnelle, aussi bien au niveau mathématique qu'informatique mais il aurait été abrupt de commencer par là...
Pour donner une valeur approchée de l'intégrale I, une première idée est de remplacer la fonction f par un polynôme P qui interpole f en plusieurs points. Cependant, dès qu'on augmente le nombre de pivots, on risque fort de se retrouver confronté au phénomène de RUNGE.
On modifie alors l'idée de départ ainsi : on commence par subdiviser régulièrement l'intervalle d'intégration en n sous-intervalles $[x_j, x_{j+1}]$ où

$$x_0 = a, \quad x_n = b, \quad \text{et} \quad \forall j \in [\![0, n]\!], \quad x_j = a + jh, \quad \text{avec } h = \frac{b-a}{n}$$

Ensuite, on remplace f par un polynôme P_j qui interpole f sur chacun des « petits » segments $[x_j, x_{j+1}]$.

– Si P_j interpole f au point x_j, alors le graphe de P_j est une droite horizontale : c'est la méthode des rectangles.

– Si P_j interpole f aux points x_j et x_{j+1}, alors le graphe de P_j est une droite affine : c'est la méthode des trapèzes.

– Si P_j interpole f aux points x_j, $\frac{x_j+x_{j+1}}{2}$ et x_{j+1}, alors le graphe de P_j est une parabole : c'est la méthode de SIMPSON 1/3.

Enfin, on somme les intégrales de chaque polynôme P_j sur $[x_j, x_{j+1}]$, et on obtient une valeur approchée de I :

$$I = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x)\mathrm{d}x = \sum_{j=0}^{n-1} \left(\int_{x_j}^{x_{j+1}} P_j(x)\mathrm{d}x + E_j \right) \approx \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} P_j(x)\mathrm{d}x$$

Comme ces méthodes ont déjà été introduites à la section **??**, on ne s'attarde que sur la méthode de SIMPSON 1/3 composée.

On se place sur l'intervalle $[x_j, x_{j+1}]$ et on note $\xi_j = \frac{x_j+x_{j+1}}{2}$; alors d'après les formules des différences divisées, le polynôme d'interpolation de f aux points (x_j, ξ_j, x_{j+1}) est donné par

$$P_j(x) = f[x_j] + f[x_j,\xi_j](x-x_j) + f[x_j,\xi_j,x_{j+1}](x-x_j)(x-\xi_j)$$

On en déduit, après calculs,

$$\int_{x_j}^{x_{j+1}} P_j(x)\mathrm{d}x = \frac{h}{6}\left(f(x_j) + 4f(\xi_j) + f(x_{j+1})\right)$$

Par sommation on obtient

$$\begin{aligned} I &\approx \frac{b-a}{6n}\left(f(x_0) + 4f\left(\tfrac{x_0+x_1}{2}\right) + 2f(x_1) + 4f\left(\tfrac{x_1+x_2}{2}\right) + \cdots + 2f(x_{n-1}) + 4f\left(\tfrac{x_{n-1}+x_n}{2}\right) + f(x_n)\right) \\ &\approx \frac{b-a}{6n}\left(f(x_0) + 2\sum_{j=1}^{n-1} f(x_j) + 4\sum_{j=0}^{n-1} f\left(\tfrac{x_j+x_{j+1}}{2}\right) + f(x_n)\right) \\ &\approx \frac{b-a}{6n}\sum_{k=0}^{2n} w_k f(a_k) \quad \text{avec} \quad a_k = a + k\cdot\frac{b-a}{2n} \text{ et } w_k = \begin{cases} 1 \text{ si } k = 0 \text{ ou } 2n \\ 4 \text{ si } 0 < k < 2n \text{ et } k \text{ impair} \\ 2 \text{ si } 0 < k < 2n \text{ et } k \text{ pair} \end{cases} \end{aligned}$$

On peut montrer que l'erreur commise est majorée par

$$|E| \leqslant \frac{1}{2880}\cdot h^4 \cdot \left\|f^{(4)}\right\| \cdot (b-a) \qquad (**)$$

On dit alors que la méthode est d'ordre 4 ; noter que la méthode est exacte pour tout polynôme de degré au plus 3.
Pour conclure cette section, revenons un instant sur l'inégalité $(**)$: cette majoration de l'erreur possède un intérêt théorique indéniable ; néanmoins, elle ne permet pas de déterminer directement l'ordre du pas h à choisir pour obtenir une approximation de I avec une précision donnée. En effet, comment déterminer une borne de la dérivée quatrième de f ? Numériquement, ce problème est trop complexe. En pratique, on applique la méthode avec deux pas différents (h et $2h$ pour minimiser les évaluations de f) et on utilise la différence des deux approximations numériques comme estimation de l'erreur du moins bon résultat.

3 Le nombre π et les arctangentes

3 1 Le nombre π et Machin

Python possède un module de calcul en multiprécision avec les quatre opérations arithmétiques de base et la racine carrée : xcom]getcontext().prec

Python

```
In [72]: from decimal import*
# on regle la precision a 30 chiffres
In [73]: getcontext().prec = 30
# on rentre les nombres entre apostrophes, comme des chaines
In [74]: deux=Decimal('2')
In [75]: deux.sqrt()
Out[75]: Decimal('1.41421356237309504880168872421')
In [76]: getcontext().prec = 50
In [77]: deux.sqrt()
Out[77]: Decimal('1.4142135623730950488016887242096980785696718753769')
```

Mais ça ne va pas nous avancer à grand chose puisque pour avoir 15 bonnes décimales avec la méthode de SIMPSON, il nous a fallu dix millions d'itérations.
Faisons un petit détour par l'histoire...
Tout commence à peu près en 1671, quand l'écossais James GREGORY découvre la formule suivante :

$$\arctan(x) = \sum_{k=0}^{+\infty} \frac{(-1)^k x^{2k+1}}{2k+1}$$

L'inévitable LEIBNIZ en publie une démonstration en 1682 dans son *Acta Eruditorum* où il est beaucoup question de ces notions désuètes que sont la géométrie et la trigonométrie.
Dans la même œuvre, il démontre ce que nous appelons aujourd'hui le critère spécial des séries alternées :

Soit (u_n) une suite réelle alternée. Si $(|u_n|)$ est décroissante et converge vers 0 alors :

– $\sum u_n$ *converge ;*

– $|R_n| \leqslant |u_{n+1}|$ *où* (R_n) *est la suite des restes associés à* $\sum u_n$.

On peut même envisager d'évoquer une démonstration de ce théorème en terminale puisqu'on y parle essentiellement de suites adjacentes...
On en déduit donc que :

$$\left|\arctan(x) - \sum_{k=0}^{n} \frac{(-1)^k x^{2k+1}}{2k+1}\right| < \frac{x^{2n+3}}{2n+3}$$

En 1706, l'anglais John MACHIN en démontra la fameuse formule :

$$4\arctan\frac{1}{5} - \arctan\frac{1}{239} = \frac{\pi}{4}$$

ce qui lui permit d'obtenir 100 bonnes décimales de π sans l'aide d'aucune machine mais avec la formule proposée par son voisin écossais.

On appelle polynômes de GREGORY les polynômes $G_n(X) = \sum_{k=0}^{n} \frac{(-1)^k X^{2k+1}}{2k+1}$.

Écrivez une fonction **`greg(a,N)`** qui donne une valeur de $G_N(a)$.
Pour cela, on aura besoin de travailler en multiprécision.
On crée une fonction **`pi_machin`** : que fait-elle ?

Python

```
def pi_machin(d):
    getcontext().prec = d + 2
    rap = 2*log(5)/log(10)
    N = int(d/rap) + 1
    return 4*(4*greg(5,N) - greg(239,N))
```

On récupère sur Sage une approximation de π.

Sage

```
sage: pi.n(digits=1000)
```

On la copie dans Python et on compare :

Python

```
PI =
     Decimal('3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798
```

En une seconde et demie, on a mille bonnes décimales de π...

Python

```
In [91]: PI-pi_machin(1000)
Out[91]: Decimal('-1.5E-1000')
```