Renaud Carpentier
Benoît Dépret

La physique de sup en applications avec Python

MPSI
PCSI
PTSI
MP2I

32 modélisations de sujets contemporains avec corrigés



ellipses

Références sciences

La physique de sup en applications avec Python

32 modélisations de sujets contemporains avec corrigés

MPSI
PCSI
PTSI
MP2I

Renaud Carpentier
Benoit Dépret

ellipses

Collection Références sciences

dirigée par Paul de Laboulaye
paul.delaboulaye@editions-ellipses.fr

Retrouvez tous les livres de la collection et des extraits sur www.editions-ellipses.fr

ISBN 9782340-063259
©Ellipses Édition Marketing S.A., 2022
8/10 rue la Quintinie 75015 Paris

DANGER LE PHOTOCOPILLAGE TUE LE LIVRE

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5.2° et 3°a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit constituerait une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

www.editions-ellipses.fr

Avant-propos

Cet ouvrage rassemble plus d'une trentaine de problèmes de physique contextualisés qui contiennent l'essentiel des notions présentes dans le nouveau programme de CPGE scientifiques (Classe Préparatoire aux Grandes Écoles) des filières MPSI, PCSI, PTSI et MP2I. La particularité de ces nouveaux programmes est sans nul doute l'introduction de « capacités numériques » que doivent avoir acquis les étudiantes et les étudiants au terme de leurs deux années de formation en classe préparatoire. Les problèmes que vous trouverez dans ce livre, accessibles à toutes les filières sauf précision dans la table des matières, ont été pensés pour aborder l'ensemble des domaines de la physique au programme (optique, électronique, ondes, mécanique, thermodynamique, électromagnétisme et physique quantique) mais aussi l'intégralité des capacités exigibles liées à l'utilisation du langage informatique « Python ». Suivant les situations, l'introduction de l'outil numérique permet de compléter, voire de résoudre, le problème de physique de différentes manières : création d'une illustration (tracé de courbes, d'histogrammes, de nuages de points, d'une animation…), résolution d'une équation n'ayant pas de solutions analytiques ou d'une intégrale sans primitive connue, traitement de données (filtrage, recherche d'extremum, analyse d'image…). Enfin, une attention particulière a été apportée dans le domaine des incertitudes, puisque, là encore, de nouvelles compétences sont attendues sur ce point dans les nouveaux programmes (écart normalisé, simulation de Monte-Carlo…). Afin d'avoir une vue synthétique du contenu de ce livre, vous trouverez un tableau indiquant les domaines intervenant dans chacun des problèmes, faisant office de table des matières. Au début de chaque problème est indiquée l'une des capacités numériques impliquée, telle qu'elle apparaît dans le programme officiel. Les codes Python ont été rédigés selon la convention PEP8 sauf pour les noms de variables afin de correspondre au mieux aux notations utilisées en physique.

Nous espérons que ce livre vous fera partager la perception que nous avons du monde, un émerveillement toujours renouvelé de la nature qui nous entoure d'une part et de la formidable intelligence de l'être humain d'autre part. Les sujets abordés dans ce livre sont souvent tirés de situations croisées dans le quotidien, que cela soit lors d'une visite d'un parc, d'une pratique sportive ou simplement à la maison. Parfois, certains phénomènes sont très communs, comme le « clic » d'une bouilloire électrique lorsque l'eau commence à bouillir, ou bien très singuliers, comme le surprenant « pendule à vagues ». Mystérieux auparavant (et encore actuellement pour certains !), ils sont devenus compréhensibles grâce aux connaissances accumulées après de nombreux siècles de recherche en physique. Nous disposons de nos jours de nombreux outils pour appréhender le fonctionnement du monde environnant. En plus des concepts théoriques de physique et de mathématiques, des appareils de mesure perfectionnés, les simulations numériques se révèlent souvent être une aide très précieuse. Nous espérons que ce livre sera une illustration convaincante que la physique et l'outil informatique forment une alliance efficace non seulement pour comprendre de nombreux phénomènes, mais aussi pour les prédire. Cette alliance est également un magnifique outil de conception pour inventer le monde de demain, puisqu'un grand nombre de défis sont à relever dès à présent, avec au premier plan le réchauffement climatique.

Les auteurs

Remerciements

Je tiens à remercier toutes les personnes qui, de près ou de loin, ont participé à ce projet. En tout premier lieu, j'exprime ma profonde gratitude à Benoît Dépret pour avoir accepté cette collaboration très fructueuse. Le livre que vous avez entre les mains est autant le fruit de son travail que du mien. Malgré mon expérience, je reste toujours aussi épaté par ses contributions d'une grande pertinence.

Merci à Jean-Marie Jacquet du cirque Plume, les perchistes de l'ASPTT de Besançon, Christophe Fraichot de la société Vibra-Tech, Angéline Petraud, Edith Bacher, Stéphanie Duval, Nicolas Pécheux et Jérémy Larochette pour l'aide qu'ils ont pu m'apporter à un moment ou à un autre sans oublier mes trois enfants (ce livre est en quelque sorte leur petit frère compte tenu de la place qu'il a pris pendant plusieurs mois à la maison…).

Je tiens également à remercier Paul de Laboulaye des éditions Ellipses pour la confiance qu'il continue de m'accorder ainsi que les personnes sollicitées pour la relecture : ma mère, ma belle-mère, Pierre Magnien et ma femme, qui a également participé à la mise en page finale.

Le travail colossal engagé pour la création de ce livre a été nourri par deux passions : celle pour la physique et celle pour la programmation. Je dois cette dernière principalement à mon frère, qui, en délaissant son petit ordinateur portable Canon X-07 (révolutionnaire à l'époque !) parce qu'il fallait écrire soi-même ses jeux en langage Basic, m'a fait plonger dans un monde de création infinie dès mon plus jeune âge.

Enfin, une pensée vient pour mon grand-père Maurice, à qui je dédie ce livre. Même si je n'ai malheureusement jamais eu la chance de partager avec toi l'amour des sciences, je te suis éternellement reconnaissant du précieux héritage que tu m'as légué.

Puisse ce livre participer, par sa modeste contribution, au rayonnement de l'enseignement scientifique en France.

Renaud Carpentier

Remerciements

Écrire un livre original qui mélangerait physique et programmation informatique avec uniquement des exercices que l'on aurait inventés ? J'en ai toujours secrètement rêvé ! Je suis donc tout particulièrement reconnaissant envers Renaud Carpentier de m'avoir entraîné (encore une fois !) dans cette formidable aventure, dont il est l'instigateur. Sa liste d'idées d'exercices potentiels est un puits infini non quantifiable. Un grand merci pour la confiance qu'il a bien voulu m'accorder et pour les nombreux échanges scientifiques, toujours riches d'enseignement, que nous avons pu avoir.

Le temps et l'investissement que requiert un tel ouvrage ne sauraient être envisageables sans le soutien inconditionnel de mes enfants et surtout de ma femme, Pascaline, ma collègue de Physique-Chimie en PSI, qui toujours m'inspire, me conseille et me guide dans mes choix.

Merci à mes parents pour m'avoir laissé jouer avec un `ZX81` quand j'étais petit (avec quand même une extension mémoire de 16 ko !) et ainsi démarrer mon initiation à la programmation.

Je voudrais aussi remercier toutes celles et tous ceux qui ont participé à l'organisation des différents webinaires sur les incertitudes pour la préparation des nouveaux programmes et aux échanges qui ont suivi sur les listes de diffusion. Les précieuses informations recueillies auront permis, je l'espère, de concevoir des exercices qui respectent au mieux cette vision plus moderne des incertitudes dans nos programmes d'enseignement.

La recherche de documents sur un sujet original n'est pas toujours une tâche facile. Si on rajoute à cela des illustrations libres de droit en vue d'une publication, la tâche devient même ardue. Merci donc à toutes ces personnes, connues ou inconnues, de Wikipédia à la NASA, qui font d'internet une source inépuisable d'information et de savoir. Chaque cliché ou document de cet ouvrage leur rend hommage en respectant à juste titre leur travail.

Et enfin dans l'ombre il ne faut pas oublier les « cobayeurs », ceux qui ont eu la lourde tâche de tester des exercices complètement nouveaux. Merci à Arthur Dépret et à Hugo Girin pour le temps consacré à la résolution et à la programmation des situations proposées pendant leur préparation aux concours.

Benoît Dépret

Table des matières

1. Incertitudes par temps de pluie 9
2. Accéléromètre embarqué 19
3. Exoplanète en transit 25
4. Dans l'oeil de Kepler 35
5. Voir les Alpes depuis la Franche-Comté 47
6. Étude d'un miroir courbé 57
7. Fabriquer sa lentille gravitationnelle 65
8. Impédance d'entrée d'un oscilloscope 83
9. Électronique relaxante 91
10. Mode AC d'un oscilloscope 99
11. Photodiode pour le haut débit 107
12. Câbles sous haute tension 127
13. Mirage numérique (PCSI) 139
14. Histoire de jet d'eau 157
15. Sur la nécessité d'un garde-boue… 167
16. Défi au billard 173
17. Stabilité du trafic routier 183
18. Bilame à retournement brusque 197
19. Principe d'un bol vibrant 209
20. Des cyclotrons pour la médecine 233
21. Physique du saut à la perche 255
22. Influence du loft d'un club de golf 279
23. Wave pendulum 297
24. Éphémérides 311
25. Panique à la NASA 327
26. En passant près du Soleil 349
27. À l'heure du thé 365
28. Chauffage sous contrôle 377
29. Efficacité maximale théorique des panneaux solaires photovoltaïques 393
30. Sondage atmosphérique 417
31. L'agitateur agité 431
32. Détection magnétique 453

AIDE-MÉMOIRE PYTHON 467

Table des matières

			incertitudes	optique	électronique	ondes	mécanique	thermodynamique	statique des fluides (PCSI)	électromagnétisme	quantique
1	Incertitudes par temps de pluie	p. 9	**X**								
2	Accéléromètre embarqué	p. 19	**X**		x		x				
3	Exoplanète en transit	p. 25	**X**				x				
4	Dans l'œil de Kepler	p. 35	**X**								
5	Voir les Alpes depuis la Franche-Comté	p. 47		**X**					x		
6	Étude d'un miroir courbé	p. 57		**X**							
7	Fabriquer sa lentille gravitationnelle	p. 65	x	**X**							
8	Impédance d'entrée d'un oscilloscope	p. 83	x		**X**						
9	Electronique relaxante	p. 91			**X**						
10	Mode AC d'un oscilloscope	p. 99			**X**						
11	Photodiode pour le haut débit	p. 107	x		**X**						x
12	Câbles sous haute tension	p.127				**X**	**X**				
13	Mirage numérique (PCSI)	p. 139				**X**					
14	Histoire de jet d'eau	p. 157					**X**				
15	Sur la nécessité d'un garde-boue…	p. 167					**X**				
16	Défi au billard	p. 173					**X**				
17	Stabilité du trafic routier	p. 183			x	x	**X**				
18	Bilame à retournement brusque	p. 197					**X**				
19	Principe d'un bol vibrant	p. 209					**X**				
20	Des cyclotrons pour la médecine	p. 233					**X**			x	
21	Physique du saut à la perche	p. 255	x				**X**			x	
22	Influence du loft d'un club de golf	p. 279					**X**				
23	Wave pendulum	p. 297				x	**X**				
24	Ephémérides	p. 311	**X**				**X**				
25	Panique à la NASA	p. 327					**X**				
26	En passant près du Soleil	p. 349	x				**X**				
27	A l'heure du thé	p. 365	**X**					**X**			
28	Chauffage sous contrôle	p. 377			x			**X**			
29	Efficacité maximale théorique des panneaux solaires photovoltaïques	p. 393		x	x			x	x		x
30	Sondage atmosphérique	p. 417					x	x	**X**		
31	L'agitateur agité	p. 431					**X**			**X**	
32	Détection magnétique	p. 453	x							**X**	
	AIDE-MÉMOIRE PYTHON	p. 467									

Le sigle **X** signifie que le chapitre concerné est majoritaire dans l'exercice, le sigle x veut dire quant à lui que le chapitre n'apparaît que ponctuellement.

1. Incertitudes par temps de pluie

MOTS-CLÉS : Incertitudes, incertitudes composées, méthode de Monte-Carlo

Capacités numériques (filières PCSI, MPSI, PTSI, MP2I) :
Simuler, à l'aide d'un langage de programmation ou d'un tableur, un processus aléatoire permettant de caractériser la variabilité de la valeur d'une grandeur composée.
Utiliser la fonction `hist` de la bibliothèque `matplotlib.pyplot` (sa spécification étant fournie) pour représenter les résultats d'un ensemble de tirages d'une variable aléatoire.

Il pleut. Quelle excellente occasion de revenir sur la notion d'incertitudes !

On réalise l'expérience simple consistant à mettre deux bassines dehors sous la pluie et à mesurer la quantité d'eau de pluie récupérée après quelques heures de mauvais temps. La bassine 1 est de forme carrée de côté 14,5 cm, tandis que la bassine 2 est circulaire de diamètre 29,0 cm. Ces longueurs sont mesurées à 0,5 cm près.

On note les masses d'eau recueillie par chaque bassine :

Bassine 1	Bassine 2
72 g	222 g

Les masses sont mesurées à 5 g près avec la balance de la cuisine.

1) Montrer que le rapport des masses d'eau dans chaque bassine permet de déterminer une valeur expérimentale du nombre π, notée dans la suite π_{exp}.

On souhaite évaluer l'incertitude-type sur cette mesure. On va pour cela simuler des valeurs expérimentales à l'aide de tirages aléatoires par la méthode de Monte-Carlo.

DOCUMENT : La méthode de Monte-Carlo adapté de « www.wikipedia.fr »

La méthode de Monte-Carlo désigne une famille de méthodes algorithmiques visant à calculer une valeur numérique approchée en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués au casino de Monte-Carlo, a été inventé en 1947 par Nicholas Metropolis et Stanislaw Ulam. Les méthodes de Monte-Carlo sont particulièrement utilisées pour calculer des intégrales en dimensions plus grandes que 1 (en particulier, pour calculer des surfaces et des volumes). Elles sont également couramment utilisées en physique des particules, où des simulations probabilistes permettent d'estimer la forme d'un signal ou la sensibilité d'un détecteur. La comparaison des données mesurées à ces simulations peut permettre de mettre en évidence des caractéristiques inattendues, par exemple de nouvelles particules.

On pourra utiliser l'instruction `np.random.uniform(-1,1,N)` qui renvoie un tableau de `N` nombres aléatoires compris entre -1 et 1 avec une distribution uniforme. On rappelle que `np.mean(x)` et `np.std(x,ddof=1)` renvoient respectivement la moyenne et l'écart-type des valeurs contenues dans un tableau `x`. L'instruction `plt.hist(x,bins=100)` permet de tracer l'histogramme des valeurs contenues dans `x` réparties en 100 intervalles.

On note m_1 la masse d'eau mesurée dans la bassine 1 et Δm_1 son incertitude. Celle-ci est en fait une estimation raisonnable de la demi-largeur de l'intervalle de valeurs possibles de la masse d'eau recueillie, due à la variabilité de la grandeur étudiée suite au protocole de mesure ou à la précision de l'instrument utilisé pour effectuer la mesure.

2) Créer un tableau de `N` valeurs aléatoires de la masse d'eau contenue dans la bassine 1 choisies uniformément dans l'intervalle $[m_1 - \Delta m_1 ; m_1 + \Delta m_1]$ et représenter l'histogramme correspondant. Evaluer l'incertitude-type $u(m_1)$ sur la masse m_1, définie comme l'écart-type de la distribution. Quelle relation simple lie $u(m_1)$ et Δm_1 ? Quel nom courant donne-t-on à ce type dévaluation. On pourra prendre `N = 1 000 000`.

On choisit ainsi aléatoirement tous les paramètres du problème, c'est-à-dire à la fois les masses d'eau recueillie et la taille des bassines, dans chaque intervalle de variabilité.

3) Créer un tableau contenant `N` valeurs expérimentales de π_{exp} obtenues avec `N` valeurs aléatoires des masses d'eau et de la taille des bassines et représenter l'histogramme correspondant. En déduire l'incertitude-type sur la mesure expérimentale de π_{exp}. Comparer la valeur de π_{exp} obtenues à partir des mesures et la moyenne des valeurs aléatoires calculées. Donner la valeur numérique de π_{exp} avec le bon nombre de chiffres significatifs.

Pour valider une mesure expérimentale, les laboratoires de recherche scientifique utilisent la notion d'écart normalisé (ou *Z-score*) par rapport à une valeur de référence défini par :

$$Z = \frac{x - x_{ref}}{u(x)}$$

où x est la valeur mesurée, x_{ref} est la valeur de référence (le plus souvent la moyenne des valeurs obtenues) et $u(x)$ l'écart-type de la distribution des valeurs de x.

Lorsque $|Z| \leq 2$, le résultat de la mesure est acceptable.

Lorsque $2 < |Z| \leq 3$, la mesure nécessite une attention particulière, notamment si cette situation se reproduit souvent.

Lorsque $|Z| > 3$, la mesure doit être rejetée et on recherche alors les causes possibles de cette anomalie.

4) Calculer l'écart normalisé de la mesure expérimentale en prenant comme référence la valeur théorique de π. La mesure obtenue est-elle acceptable ?

Il existe des relations donnant l'incertitude-type sur une grandeur composée connaissant l'incertitude-type sur les grandeurs intermédiaires. On a ainsi par exemple :

Grandeur	Incertitude-type
x, y	$u(x)$, $u(y)$
$z = ax + by$	$u(z) = \sqrt{\left(au(x)\right)^2 + \left(bu(y)\right)^2}$
$z = ax^{\alpha} y^{\beta}$	$\frac{u(z)}{z} = \sqrt{\left(\alpha \frac{u(x)}{x}\right)^2 + \left(\beta \frac{u(y)}{y}\right)^2}$

5) Exprimer l'incertitude-type sur π_{exp} en fonction des incertitudes-types sur les différents paramètres de l'expérience. Comparer le résultat numérique à la valeur obtenue par la méthode de Monte-Carlo. Commenter.

La méthode de Monte-Carlo peut aussi permettre de calculer des surfaces. On se propose de simuler la pluie par des tirages aléatoires bidimensionnels pour déterminer la valeur de π.

On considère le disque de rayon R, contenu dans un carré de côté $2R$. Comme les gouttes de pluie récupérées par chaque bassine, on choisit aléatoirement et uniformément des points dans le carré. La probabilité qu'un point soit contenu dans le disque est alors égale au rapport de la surface du disque sur la surface du carré.

6) Exprimer la probabilité théorique qu'un point choisi aléatoirement dans le carré soit contenu dans le disque.

7) Ecrire la fonction Python d'entête :

```
def experience(Ntirages):
```

qui renvoie une valeur expérimentale de π par la méthode de Monte-Carlo après un nombre `Ntirages` de tirages dans le carré.

8) Comparer les mesures obtenues avec `100, 1 000 et 10 000` tirages en traçant les histogrammes représentant la répartition de 1 000 valeurs. Quel est le rôle du nombre de tirages dans l'expérience numérique ainsi réalisée ? Comparer les incertitudes-types relatives à chaque expérience.

On pourra rajouter l'option `range = (2.8, 3.5)` dans `plt.hist()` pour tracer l'histogramme dans l'intervalle fixe `[2.8, 3.5]` afin de mieux comparer les résultats.

Il est d'usage de tracer l'écart normalisé pour les différentes mesures expérimentales par rapport à leur valeur moyenne pour détecter les éventuelles anomalies. On considère une série de 10 000 points de mesures obtenus avec des expériences à 1 000 tirages aléatoires.

9) Tracer l'écart normalisé relatif à ces 10 000 mesures et l'histogramme de la distribution des valeurs entre -5 et 5. Commenter.

10) Déterminer la probabilité d'avoir un *Z-score* $|Z| \leq 2$ et commenter le critère d'acceptabilité d'une mesure expérimentale.

CORRIGÉ

1) Pour déterminer la valeur théorique du rapport $\frac{m_2}{m_1}$ des masses d'eau recueillie par les bassines, on suppose que les gouttes de pluie tombant au sol sont réparties aléatoirement et uniformément en surface. Cela signifie que la quantité d'eau récupérée est en moyenne proportionnelle à la surface de la bassine et à la durée de l'expérience. On pourrait définir un courant surfacique de masse d'eau apportée par la pluie, en moyenne vertical et

uniforme, de norme j_0 (en kg.m^{-2}.s^{-1}), de sorte qu'après la durée Δt la masse d'eau récupérée dans chaque bassine vaut respectivement :

$$m_1 = j_0 S_1 \Delta t \text{ et } m_2 = j_0 S_2 \Delta t$$

où $S_1 = a_1^2$ (surface du carré de côté a_1) et $S_2 = \frac{\pi a_2^2}{4}$ (surface du disque de diamètre a_2).

On obtient ainsi :

$$\frac{m_2}{m_1} = \frac{S_2}{S_1} = \frac{\pi a_2^2}{4a_1^2}, \text{ c'est-à-dire : } \pi = \frac{4m_2 a_1^2}{m_1 a_2^2}$$

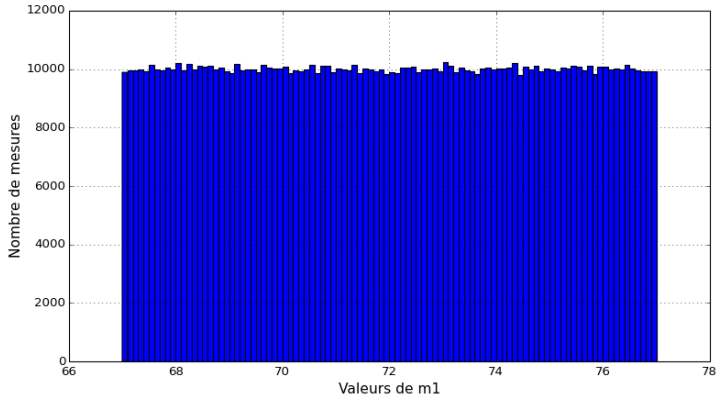
En remarquant que $a_2 = 2a_1$, on obtient simplement :

$$\pi_{exp} = \frac{m_2}{m_1} = 3,08$$

On retrouve un résultat proche de la valeur théorique de π (à 1,3 % près). Il faut cependant évaluer l'incertitude-type associée à cette mesure pour valider le protocole utilisé.

2) L'instruction `np.random.uniform()` permet de créer directement un tableau et on trace l'histogramme montrant la répartition des valeurs dans l'intervalle de variabilité :

```
m1 = 72 # g
delta_m1 = 5 # g
N = 1000000
m1_MC = m1 + delta_m1*np.random.uniform(-1, 1, N) # Valeurs aléatoires
plt.hist(m1_MC, bins=100)
```



S'agissant d'une distribution aléatoire uniforme, les valeurs tirées au hasard sont équiprobables dans l'intervalle choisi.

On obtient l'incertitude-type en calculant l'écart-type de la distribution de valeurs :

```
u_m1 = np.std(m1_MC, ddof=1)
```

Numériquement :

```
u_m1 → 2.8871250630507777 # g
```

On en déduit le rapport :

```
delta_m1/u_m1 → 1.7318266063322461
```

que l'on peut comparer à :

```
np.sqrt(3) → 1.7320508075688772
```

Statistiquement, on peut donc exprimer l'incertitude-type $u(m_1)$ en fonction de l'incertitude expérimentale Δm_1 sous la forme :

$$u(m_1) = \frac{\Delta m_1}{\sqrt{3}}$$

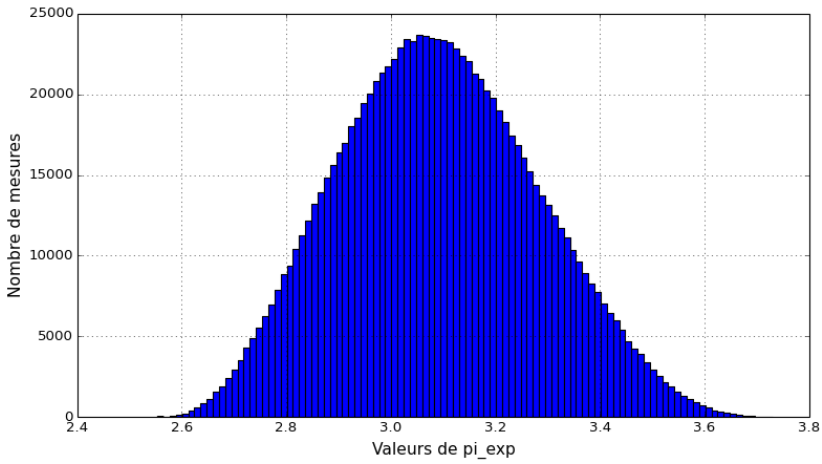
Il s'agit d'une évaluation de type B de l'incertitude-type sur une mesure unique.

3) On calcule `N` valeurs de π_{exp}, sans oublier de tenir compte de la variabilité de toutes les grandeurs en choisissant aléatoirement des masses m_1 et m_2 et des longueurs a_1 et a_2 :

```
m2 = 222 # g
delta_m2 = 5 # g
a1 = 14.5 # cm
delta_a1 = 0.5 # cm
a2 = 29.0 # cm
delta_a2 = 0.5 # cm
m2_MC = m2 + delta_m2*np.random.uniform(-1, 1, N) # Valeurs aléatoires
a1_MC = a1 + delta_a1*np.random.uniform(-1, 1, N) # Valeurs aléatoires
a2_MC = a2 + delta_a2*np.random.uniform(-1, 1, N) # Valeurs aléatoires
pi_MC = 4 * m2_MC * a1_MC**2 / (m1_MC * a2_MC**2) # Calcul de pi_exp
```

Le formalisme des tableaux `np.array` permet de calculer directement le tableau `pi_MC`. On représente l'histogramme des valeurs obtenues :

```
plt.hist(pi_MC, bins=100)
```



La répartition des valeurs obtenues suit apparemment une loi gaussienne caractéristique d'une distribution statistique normale.

On calcule maintenant l'écart-type de cette distribution, correspondant à l'incertitude-type sur chaque valeur expérimentale :

```
u_pi = np.std(pi_MC, ddof=1) → 0.18965148801364812
```

On compare la valeur expérimentale de π_{exp} et la moyenne de la distribution aléatoire :

```
pi_exp = 4 * m2 * a1**2 / (m1 * a2**2) → 3.0833333333333335
pi_moy = np.mean(pi_MC) → 3.0905710203951293
```

Ces valeurs sont proches étant donnée l'incertitude-type obtenue. Dans une approche classique on prendra le plus souvent la valeur issue des mesures expérimentales.

Finalement, un résultat numérique avec deux chiffres significatifs est suffisant :

$$\pi_{exp} = 3{,}1 \text{ et } u(\pi_{exp}) = 0{,}2$$

4) On calcule l'écart normalisé avec la définition fournie :

```
(pi_exp - np.pi)/u_pi → -0.3071914745655307
```

c'est-à-dire Z = -0,31. Avec $|Z| \leq 2$, la mesure de π obtenue est acceptable.

5) On effectue ici le calcul classique d'incertitudes composées. On a la relation :

$$\pi_{exp} = \frac{4m_2a_1^2}{m_1a_2^2}$$

qui impose l'incertitude-type relative sur π_{exp} :

$$\frac{u(\pi_{exp})}{\pi_{exp}} = \sqrt{\left(\frac{u(m_1)}{m_1}\right)^2 + \left(\frac{u(m_2)}{m_2}\right)^2 + \left(2\frac{u(a_1)}{a_1}\right)^2 + \left(2\frac{u(a_2)}{a_2}\right)^2}$$

en prenant :

$$u(m_1) = \frac{\Delta m_1}{\sqrt{3}},\ u(m_2) = \frac{\Delta m_2}{\sqrt{3}},\ u(a_1) = \frac{\Delta a_1}{\sqrt{3}} \text{ et } u(a_2) = \frac{\Delta a_2}{\sqrt{3}}$$

Numériquement :

```
pi_exp*np.sqrt((u_m1/m1)**2 + (u_m2/m2)**2
               + (2*u_a1/a1)**2 + (2*u_a2/a2)**2)
→ 0.18948793593740834
```

On retrouve la même valeur que celle obtenue avec la méthode de Monte-Carlo, au prix d'un calcul conséquent qu'on ne pourra pas forcément mener jusqu'au bout si la relation est trop compliquée ! La méthode de Monte-Carlo est un outil puissant pour déterminer efficacement l'incertitude-type sur des grandeurs composées.

6) On calcule le rapport des surfaces pour obtenir la probabilité p qu'un point tiré au hasard dans le carré se trouve aussi dans le disque :

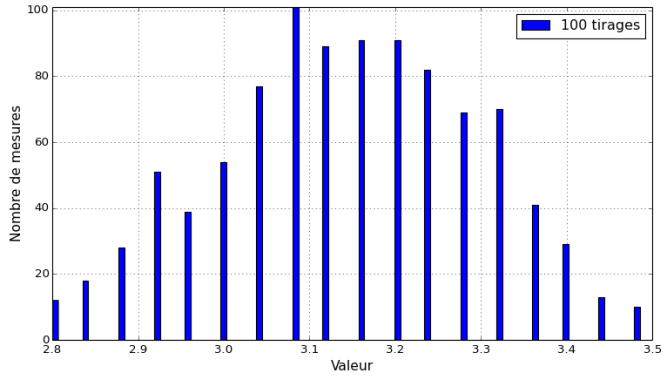
$$p = \frac{\pi R^2}{(2R)^2} = \frac{\pi}{4} = 78{,}5\ \%$$

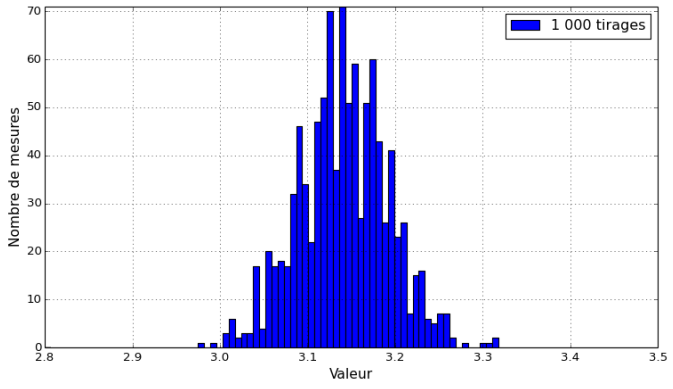
7) On utilise à nouveau la fonction `np.random.uniform(-1, 1, N)` pour obtenir aléatoirement les coordonnées (`x,y`) d'un point dans le carré de côté $R = 1$ (les résultats ne dépendent pas de la valeur de R). On teste si les point choisis sont situés dans le disque et on compte le nombre de fois où cela arrive. La probabilité de tirer un point dans le disque est le rapport du nombre de points dans le disque sur le nombre total de tirages. En identifiant à la probabilité théorique, on évalue π.

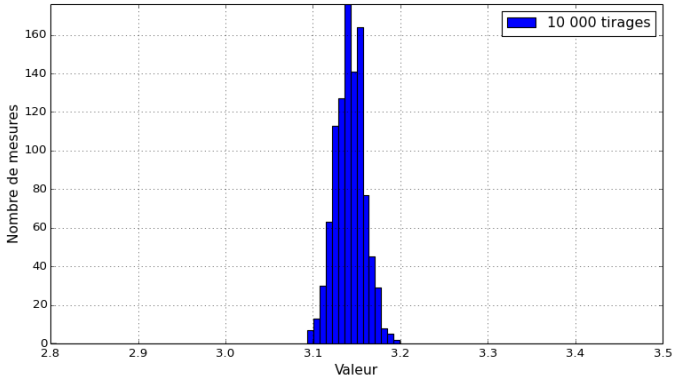
```
def experience(Ntirages):
    x = np.random.uniform(-1, 1, Ntirages) # Tirages aléatoires
    y = np.random.uniform(-1, 1, Ntirages) # Tirages aléatoires
    r = np.sqrt(x**2 + y**2) # Calcul de la distance au centre
    compteur = 0 # Initialisation du compteur
    for k in range(Ntirages):
        if r[k] <= 1: # Le point est situé dans le disque
            compteur = compteur + 1
    return 4 * compteur / Ntirages # Valeur expérimentale de pi
```

8) On crée un tableau contenant les 1 000 mesures de π avec l'expérience numérique, on calcule l'incertitude-type de la distribution de valeurs et on trace l'histogramme correspondant :

```
Ntirages = 100 # Nombre de tirages
N = 1000 # Nombre de mesures
mesures = np.array([experience(Ntirages) for i in range(N)])
u_pi = np.std(mesures, ddof=1) # Incertitude-type
plt.hist(mesures, range=(2.8, 3.5), bins=100)
```







Les valeurs obtenues à chaque expérience sont d'autant plus resserrées autour de la valeur moyenne que le nombre de tirages est important, comme le confirment les valeurs de l'incertitude-type :

Nombre de tirages	Incertitude-type
100	0.168537350277
1 000	0.0512813104044
10 000	0.0163772400774

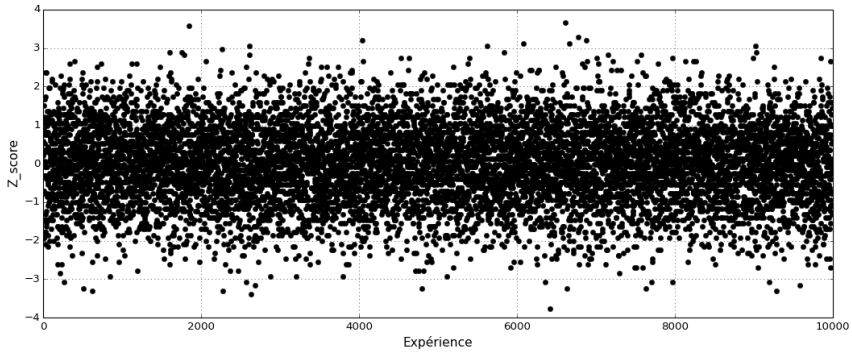
Le nombre de tirages aléatoires dans la simulation numérique impose en fait la variabilité de la grandeur mesurée, que l'on ne peut généralement pas contrôler expérimentalement.

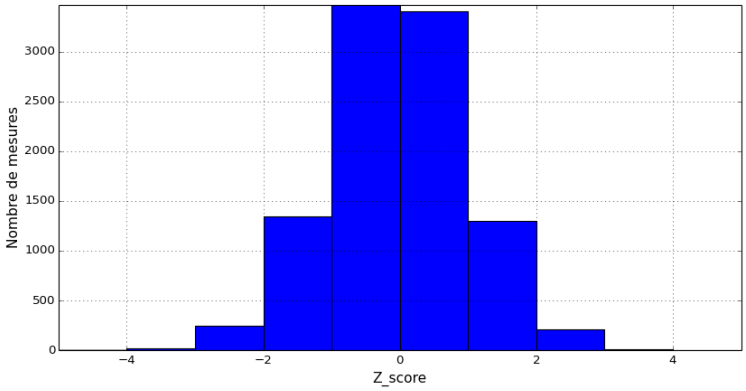
9) On crée le tableau des écarts normalisés pour chaque point de mesure :

```
mesures = np.array([experience(Ntirages) for i in range(N)])
pi_moy = np.mean(mesures) # Moyenne
u_pi = np.std(mesures, ddof=1) # Incertitude-type
z_score = (mesures - pi_moy)/u_pi # Ecart normalisé
```

et on trace à la fois l'écart normalisé et l'histogramme correspondant :

```
plt.plot(z_score, 'ko')
plt.hist(z_score, range=(-5,5), bins=10)
```





Les valeurs de l'écart normalisé sont essentiellement situées entre -2 et 2, ce qui valide globalement les mesures expérimentales.

10) On estime sur l'histogramme qu'il y a environ 500 mesures telles que $|Z| > 2$ (en sommant les valeurs entre -3 et -2 d'une part et entre 2 et 3 d'autre part). Sur 10 000 mesures, cela représente environ 5 % des mesures. On peut confirmer ce résultat en comptant le nombre de mesures telles que $|Z| \leq 2$ avec la fonction suivante :

```
def confiance(z_score, seuil):
    N = len(z_score)
    compteur = 0
    for k in range(N):
        if abs(z_score[k]) <= seuil:
            compteur = compteur + 1
    return compteur / N
```

On en déduit :

```
confiance(z_score, 2) → 0.956
```

Statistiquement, une mesure expérimentale prise au hasard a donc bien environ 95 % de chance d'avoir un écart normalisé tel que $|Z| \leq 2$.

Cette probabilité est encore plus grande si on considère $|Z| \leq 3$:

```
confiance(z_score, 3) → 0.9965
```

La valeur de l'écart normalisé permet donc bien de détecter les anomalies de mesure : si une mesure (ou pire s'il y en a plusieurs) a un écart normalisé supérieur à 3 (en valeur absolue), il y a de fortes chances qu'il s'agisse d'un problème de manipulation ou une anomalie du protocole de mesures plutôt qu'un écart statistique.

2. Accéléromètre embarqué

MOTS-CLÉS : Mécanique, accéléromètre, filtrage numérique

Capacité numérique (filières PCSI, MPSI, PTSI, MP2I) : mettre en œuvre la méthode d'Euler explicite afin de résoudre une équation différentielle d'ordre 1.

DOCUMENT : The Shard adapté de « www.wikipedia.fr »

The Shard est un gratte-ciel de bureaux et de logements de luxe situé dans l'arrondissement de Southwark, sur la rive sud de la Tamise et face à la Cité de Londres (Royaume-Uni). Inaugurée le 5 juillet 2012, la tour fut baptisée The Shard à la suite des critiques qu'elle avait essuyées de la part d'English Heritage, qui affirmaient que le bâtiment serait comme « un éclat de verre transperçant le cœur du vieux Londres » (a shard of glass through the heart of historic London). Cette tour, qui au moment de son inauguration est l'immeuble le plus haut du Royaume-Uni et de l'Union européenne, a ouvert ses portes au public le 1er février 2013, deux étages étant réservés aux visites payantes, permettant ainsi d'offrir une des plus belles vues sur la capitale britannique.

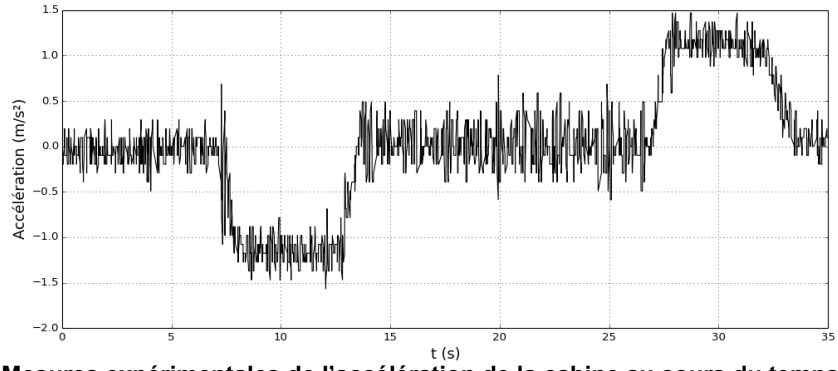
Style	Néo-futurisme et architecture high-tech
Matériau	Structure en matériau composite, façade en verre
Hauteur de l'antenne	309,6 mètres
Hauteur du toit	304,1 mètres
Hauteur du dernier étage	245 mètres
Étages	72 (habitables) 95 (avec étages techniques)
Superficie totale	127 489 m^2
Nombre d'ascenseurs	44



(Photo : Colin, licence CC)

The Shard à Londres

En visite à Londres, un physicien monte dans un ascenseur du *Shard* et déclenche l'accéléromètre de son smartphone. Il enregistre la courbe ci-après représentant l'accélération verticale de la cabine (orientée positivement vers le haut) au cours du temps pendant le trajet.



Mesures expérimentales de l'accélération de la cabine au cours du temps

On propose d'utiliser la fonction suivante `acceleration(t, u_a)` qui simule en Python l'accélération expérimentale mesurée à l'instant `t` avec l'incertitude-type `u_a` :

```
def acceleration(t, u_a): # Accélération avec l'incertitude u_a
    t0 = [0, 7.1, 8.4, 12.4, 13.9, 26.9, 27.9, 32.2, 33.2, 35] # s
    a0 = [0, 0, -1.1, -1.1, 0, 0, 1.1, 1.1, 0, 0] # Accélération (m/s²)
    if t <= 0 or t >= 35 :
        acc = 0.
    else:
        i = 0
        while t > t0[i + 1]:
            i += 1
        acc = a0[i] + (a0[i + 1] - a0[i])/(t0[i + 1] - t0[i])*(t -
t0[i])
    return np.random.normal(acc, u_a)
```

1) Estimer l'incertitude expérimentale sur la mesure de l'accélération fournie par l'accéléromètre et reproduire le tracé de l'accélération mesurée au cours du temps. On prendra `N` = 3 500 points.

On cherche à réduire le bruit aléatoire introduit par le dispositif en lissant la courbe par un procédé de moyenne glissante : on calcule une moyenne des valeurs sur un intervalle mobile pour obtenir une nouvelle liste d'échantillons moyens centrés sur chaque intervalle.

2) Ecrire la fonction d'entête :

```
def moyenne_glissante(x, i, n):
```

renvoyant la moyenne des `n` valeurs d'une liste `x` centrées sur l'élément d'indice `i`.

3) En déduire la fonction d'entête :

```
def lissage(t, a, n):
```

permettant le calcul de la moyenne glissante sur `n` échantillons de la liste `a` et la moyenne glissante sur `n` échantillons de la liste de temps correspondante `t`. La fonction devra renvoyer une nouvelle liste des instants et la liste des échantillons lissés correspondants.

4) Tracer le graphe de la courbe lissée. Commenter l'influence de la valeur de `n`. Préciser le type de filtrage numérique ainsi réalisé.

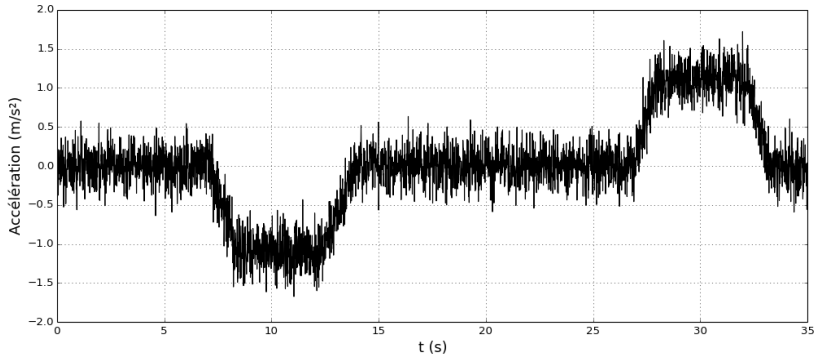
5) De combien d'étages le physicien-voyageur s'est-il déplacé et dans quel sens ?

CORRIGÉ

1) Les valeurs expérimentales se répartissent sur un intervalle de largeur totale $2\Delta a = 0{,}8$ m.s^{-2}, d'où l'incertitude-type sur l'accélération : $u(a) = \dfrac{\Delta a}{\sqrt{3}} = 0{,}2$ m.s^{-2} avec une évaluation de type B. Pour reproduire les données, on crée une liste des temps entre $t = 0$ et $t = 35$ s avec N points et la liste des accélérations correspondantes avec le code fourni :

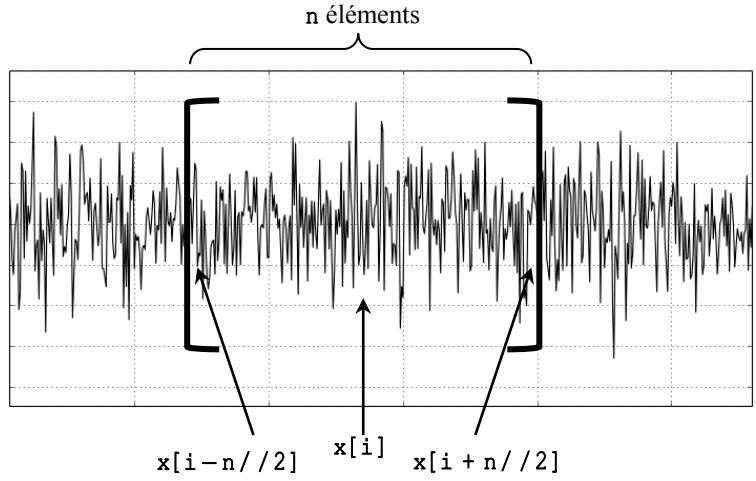
```
N = 3500 # Nombre de points
tmax = 35 # s Durée de l'expérience
dt = tmax/(N - 1) # Pas de temps
t = [k * dt for k in range(N)] # Liste des temps
u_a = 0.2 # m/s² Incertitude-type sur l'accélération
a = [acceleration(t[k], u_a) for k in range(N)] # Liste des accélérations
```

On trace l'accélération simulée obtenue au cours du temps :



Ce graphe modélise correctement le signal mesuré.

2) On représente les `n` valeurs aléatoires dans un intervalle centré sur l'indice `i` :



On calcule la somme, puis la moyenne, des éléments d'indices `i-n//2` à `i+n//2` (en supposant que cet intervalle est bien défini) :

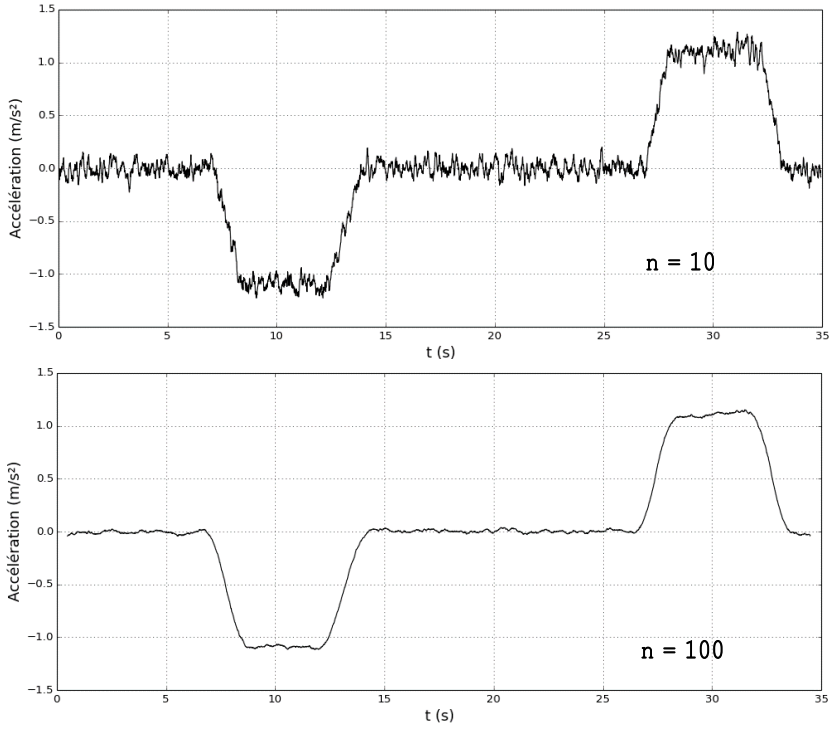
```
def moyenne_glissante(x, i, n):
    somme = 0 # Initialisation de la somme
    for k in range(n):
        somme = somme + x[i - n//2 + k]
    return somme/n
```

3) On calcule la moyenne glissante de la liste des temps et de la liste des accélérations pour toutes les valeurs de `i` possibles, c'est-à-dire entre `n//2` et `N-n//2` :

```
def lissage(t, a, n):
    t_liss, a_liss = [], [] # Initialisation des listes de sortie
    for i in range(n//2, len(t) - n//2):
        t_liss.append(moyenne_glissante(t, i, n))
        a_liss.append(moyenne_glissante(a, i, n))
    return t_liss, a_liss
```

4) On appelle la fonction et on trace la courbe lissée pour `n = 10` et `n = 100` :

```
t_liss, a_liss = lissage(t, a, n)
plt.plot(t_liss, a_liss)
```



La courbe est d'autant plus lissée que la valeur de `n` augmente. L'intervalle de temps sur lequel on calcule la moyenne doit rester petit devant la durée caractéristique d'évolution du signal si on ne veut pas déformer les mesures expérimentales. L'algorithme de moyenne glissante permet de réduire l'amplitude d'un bruit aléatoire par un filtrage numérique passe-

bas puisque la moyenne permet de faire disparaître les oscillations rapides du signal (de hautes fréquences). La fréquence de coupure est de l'ordre de l'inverse de la durée sur laquelle on effectue la moyenne.

```
print('Fréquence de coupure pour n = ', n, ':', 1/t[n], 'Hz')
Fréquence de coupure pour n = 10 : 9.997142857142856 Hz
Fréquence de coupure pour n = 100 : 0.9997142857142856 Hz
```

La fréquence de coupure est d'autant plus basse que le nombre `n` de points est grand.

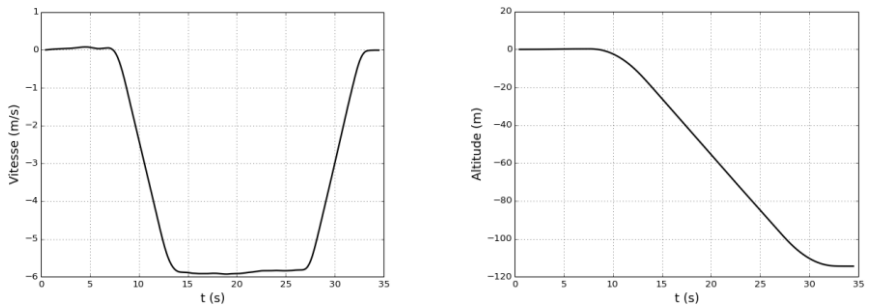
5) On relie l'accélération mesurée à la vitesse verticale de déplacement v_z et à l'altitude z du physicien :

$$a_z = \frac{\mathrm{d}v_z}{\mathrm{d}t} \text{ et } v_z = \frac{\mathrm{d}z}{\mathrm{d}t}$$

Il s'agit donc d'intégrer l'accélération et la vitesse en considérant une vitesse initiale nulle (car l'ascenseur est à l'arrêt quand on monte dedans !) et une origine arbitraire $z = 0$. On procède à une intégration numérique en utilisant la méthode d'Euler. On utilise les données lissées avec `n = 100` en prenant garde à la taille des listes, raccourcies à cause de la moyenne glissante.

```
# Calcul de la position et de la vitesse :
z, vz = [0], [0]
Npoints = N - n # Taille des listes lissées
for k in range(Npoints - 1):
    vz.append(vz[k] + a_liss[k]*dt) # Calcul de la vitesse
    z.append(z[k] + vz[k]*dt) # Calcul de l'altitude
```

On trace enfin l'évolution de la vitesse et de l'altitude au cours du temps :



Après un temps d'attente d'environ 7 s (le temps que les personnes montent), l'ascenseur voit sa vitesse diminuer (d'où l'accélération négative pendant cette première phase) pour atteindre une vitesse de -5,9 m.s^{-1} (soit 21 km/h). L'ascenseur descend alors à vitesse constante pendant environ 14 s (on retrouve une accélération nulle), avant de ralentir (la vitesse augmente et l'accélération devient positive) et de s'arrêter (vitesse nulle et accélération nulle). L'altitude ne fait que diminuer pendant tout ce temps et passe de $z = 0$ à $z = -114$ m.

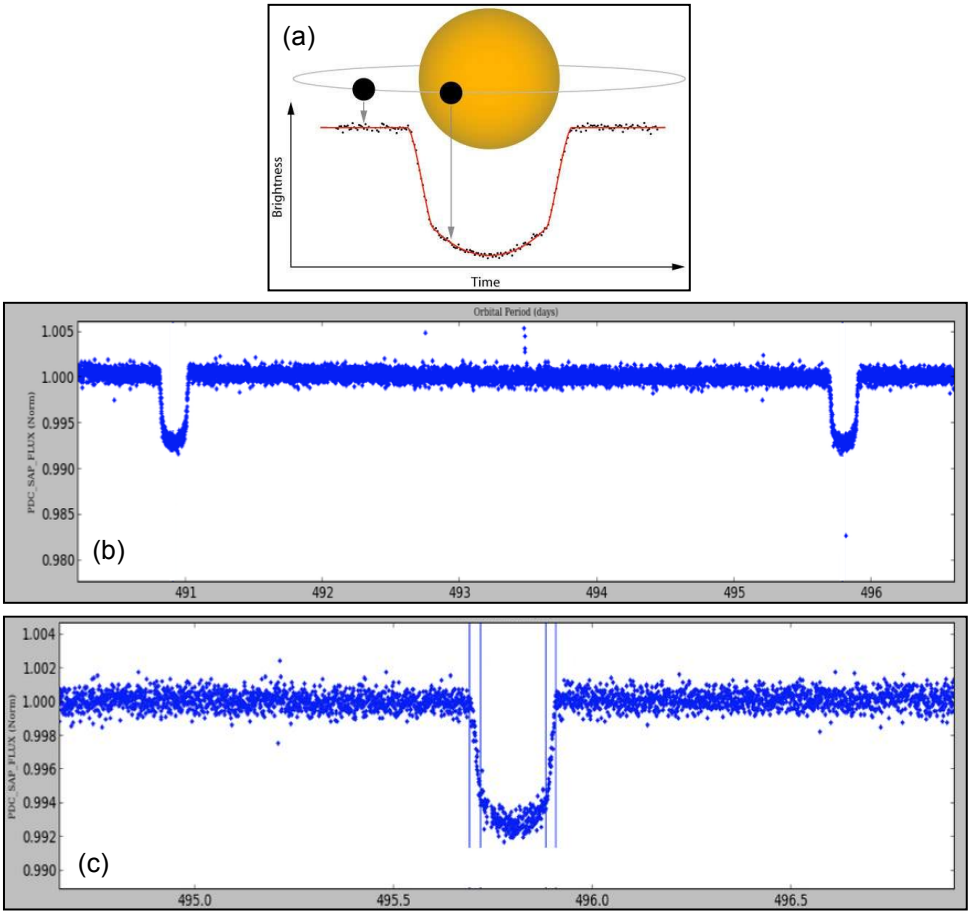
Le physicien a donc parcouru 114 m vers le bas. La tour du *Shard* comporte 95 étages avec un dernier étage à 245 m d'altitude, ce qui représente 2,6 m entre deux étages consécutifs en moyenne. Le trajet en ascenseur a finalement transporté le physicien 44 étages plus bas, sans que l'on puisse connaître l'étage de départ !

3. Exoplanète en transit

MOTS-CLÉS : Exoplanète, méthode de Monte-Carlo, incertitudes-types

Capacité numérique (filières PCSI, MPSI, PTSI, MP2I) : simuler, à l'aide d'un langage de programmation ou d'un tableur, un processus aléatoire de variation des valeurs expérimentales de l'une des grandeurs – simulation Monte-Carlo – pour évaluer l'incertitude sur les paramètres du modèle.

Parmi les techniques de détection des exoplanètes, la méthode du transit est la plus utilisée. Le passage d'une planète devant son étoile résulte en une (faible) diminution de la luminosité apparente de l'étoile vue depuis la Terre. Le télescope spatial Kepler a ainsi permis à lui seul de détecter plus de 2 600 exoplanètes confirmées, soit plus de la moitié des exoplanètes connues à ce jour. La méthode du transit nécessite de scruter en permanence les étoiles du ciel nocturne à la recherche du signal caractéristique du passage d'une planète.



(a) Principe d'un transit planétaire. (b) Luminosité normalisée de l'étoile Kepler-7 en fonction du temps (en jours). (c) Détail de la luminosité au moment du transit.

Kepler-7 est une étoile de la constellation de la Lyre. Des observations photométriques ont permis d'obtenir sa masse (égale à 1,36 masse solaire) et son rayon (2,02 fois plus grand que celui du Soleil). L'observation par le télescope Kepler a mis en évidence une baisse périodique de sa luminosité due à la planète Kepler-7b en orbite autour de l'étoile. La forme du signal dépend des caractéristiques de la planète et de son orbite.

La luminosité apparente de l'étoile est proportionnelle à sa surface visible depuis la Terre. La luminosité normalisée est le rapport de la luminosité apparente de l'étoile pendant le transit sur la luminosité apparente de l'étoile en dehors du transit : la luminosité normalisée vaut ainsi 1 lorsque l'étoile n'est pas masquée par la planète et diminue lorsque la planète passe devant l'étoile lors du transit. On note α le rapport des diamètres apparents de la planète et de l'étoile. α est a priori inconnu et pourra être déterminé par comparaison entre les données simulées et les données expérimentales, sachant que le plus souvent $\alpha << 1$.

Afin d'avoir une simulation indépendante des caractéristiques du système stellaire étudié, on travaille avec des grandeurs adimensionnées. On considère ainsi que, vue de la Terre, l'étoile est un disque de rayon 1 de centre O, origine d'un repère cartésien dont l'axe Oz est dans la direction étoile-Terre. Le centre de la planète en transit a pour coordonnées adimensionnées (x,y). Son rayon adimensionné vaut α. Au cours de son mouvement orbital autour de l'étoile, l'exoplanète peut être située devant ou derrière l'étoile.

Il n'existe pas de formule simple donnant la surface apparente de l'étoile lors du transit, c'est pourquoi on se propose de calculer numériquement cette surface par la méthode de Monte-Carlo qui utilise des tirages aléatoires.

On rappelle que `np.random.uniform(-1, 1)` renvoie un nombre aléatoire compris entre -1 et 1 avec une distribution uniforme et que `np.mean(u)` et `np.std(u, ddof=1)` renvoient respectivement la moyenne et l'écart-type des valeurs contenues dans une liste `u`. On donne quelques caractéristiques physiques en fin de problème.

1) Ecrire la fonction d'entête :

```
def mesure_luminosite(x, y, alpha, n):
```

qui renvoie une valeur simulée de la luminosité apparente normalisée de l'étoile lorsque la planète de coordonnées adimensionnées `(x, y)` et de rayon adimensionné `alpha` passe devant l'étoile. On utilisera la méthode de Monte-Carlo avec un nombre `n` de tirages aléatoires.

Le nombre de tirages caractérise en fait la précision de la mesure simulée et pourra être déterminé par comparaison entre les résultats de la simulation et les données expérimentales.

Le système photométrique à bord du télescope spatial Kepler fait plusieurs mesures de la luminosité apparente afin d'effectuer un traitement statistique des mesures expérimentales.

2) Ecrire la fonction d'entête :

```
def mesure_Kepler(x, y, alpha, n):
```

qui renvoie la moyenne et l'incertitude-type de la luminosité apparente normalisée de l'étoile au cours du transit en considérant une série de 10 mesures successives.

Combien de tirages `n` faudrait-il considérer pour obtenir la précision des mesures expérimentales ?

On suppose que l'orbite de Kepler-7b est circulaire de rayon a, de période orbitale T, et que le plan de l'orbite est confondu avec le plan xOz. Le système stellaire est donc vu « par la tranche » depuis la Terre. Les coordonnées adimensionnées de la planète dans l'espace s'écrivent alors :

$$\begin{cases} x(t) = \dfrac{a}{R}\sin\left(\dfrac{2\pi}{T}(t-t_0)\right) \\ y(t) = 0 \\ z(t) = \dfrac{a}{R}\cos\left(\dfrac{2\pi}{T}(t-t_0)\right) \end{cases}$$

où R est le rayon de l'étoile Kepler-7.

3) Déterminer graphiquement les valeurs de T et t_0 dans le cas de Kepler-7b à partir des résultats expérimentaux fournis.

4) Montrer que les lois de la mécanique céleste permettent de déterminer a.

5) Quelle serait l'expression de $y(t)$ *pendant le transit* si l'orbite de Kepler-7b était inclinée d'un angle β par rapport à l'axe Oz ? Déterminer l'angle maximal d'inclinaison de l'orbite permettant l'observation du transit de l'exoplanète depuis la Terre. En déduire la probabilité de découvrir une exoplanète comme Kepler-7b par la méthode du transit.

On cherche à présent à reproduire numériquement les résultats expérimentaux avec la simulation afin de vérifier les caractéristiques de l'exoplanète.

6) Ecrire la fonction d'entête :

```
def xyz(t):
```

renvoyant les coordonnées adimensionnées `x`, `y`, `z` de Kepler-7b dans l'espace à l'instant `t`.

7) Estimer la durée `dt` entre deux points expérimentaux sur le graphe fourni. Déterminer les instants `tmin` et `tmax` de début et de fin d'expérience.

8) Ecrire la boucle principale permettant de remplir les tableaux `t` des instants d'acquisition, `lum_norm` de la luminosité apparente normalisée de l'étoile au cours du temps et `u_lum_norm` de l'incertitude-type associée à chaque mesure de luminosité.

9) En déduire le tracé de la luminosité apparente normalisée au cours du temps en faisant apparaître les barres d'incertitude.

Tester le programme avec `alpha = 0.4` et `n = 1 000`. L'exécution de la simulation numérique peut prendre beaucoup de temps. On pourra utiliser un indicateur de progression du programme (en pourcentage).

10) Quelles sont les valeurs de `alpha` et `n` qui reproduisent le plus fidèlement possible les données expérimentales ? Ajuster la valeur de `y` et en déduire l'angle d'inclinaison β. Commenter.

11) En déduire le rayon de Kepler-7b et comparer les caractéristiques obtenues avec celles de Jupiter.

12) Quelle serait la taille de la plus petite exoplanète détectable par le télescope Kepler ? Commenter.

Données physiques	
Masse du Soleil : $M_\odot = 1{,}99.10^{30}$ kg	Rayon du Soleil : $R_\odot = 6{,}96.10^{5}$ km
Masse de Jupiter : $M_J = 1{,}90.10^{27}$ kg	Rayon de Jupiter : $R_J = 69{,}9.10^{3}$ km
Distance Terre-Soleil : 1 au (unité astronomique) = $1{,}50.10^{8}$ km	
Distance Jupiter-Soleil : 5,2 au	
Constante de gravitation universelle : $\mathcal{G} = 6{,}67.10^{-11}$ m^3.kg^{-1}.s^{-2}	

L'exercice « Dans l'œil de Kepler » propose d'analyser une image numérique à la recherche d'un signal de transit et de préciser les critères de détection d'une exoplanète.

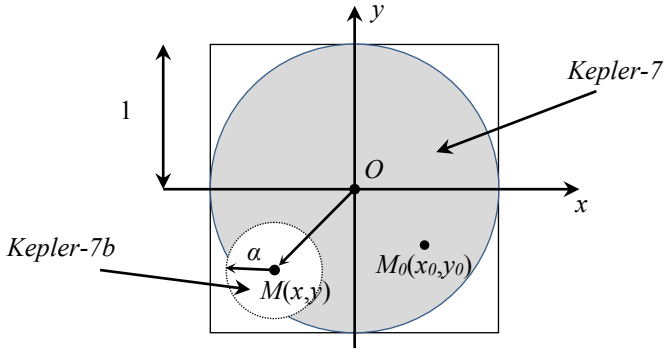
CORRIGÉ

1) La luminosité normalisée est le rapport de la surface apparente de l'étoile, en partie masquée par la planète, sur sa surface apparente totale, sans planète. Pour déterminer cette surface, on utilise la méthode de Monte-Carlo : un point est choisi aléatoirement dans un carré de côté 2 ; on évalue alors la probabilité que ce point appartienne à la surface apparente visible de l'étoile, disque de rayon 1 inscrit dans le carré, pendant la phase de transit, puis on normalise le résultat pour obtenir la luminosité normalisée cherchée.

On appelle $M_0(x_0,y_0)$ un point choisi aléatoirement dans le carré.

Ce point appartient au disque de rayon 1 (l'étoile) dès lors que : $x_0^2 + y_0^2 \leq 1$, et se situe en dehors du disque de centre M et de rayon α (la planète) lorsque : $(x - x_0)^2 + (y - y_0)^2 > \alpha^2$.

Au cours des `n` tirages, on compte le nombre de fois que chaque point se trouve dans la surface apparente de l'étoile à l'aide de la variable `compteur`. En l'absence de planète devant l'étoile, le rapport `compteur/n` serait égal au rapport de la surface du disque sur la surface du carré, soit simplement π/4. On en déduit la luminosité normalisée.



La méthode de Monte-Carlo permet de déterminer la surface grisée

On écrit ainsi la fonction suivante :

```
def mesure_luminosite(x, y, alpha, n):
    compteur = 0 # Initialisation du compteur
    for k in range(n): # On effectue n tirages dans le carré de côté 2
        x0 = np.random.uniform(-1, 1)
        y0 = np.random.uniform(-1, 1)
        if (x0**2 + y0**2) <= 1 and ((x0 - x)**2+(y0 - y)**2) >
alpha**2:
            compteur = compteur + 1 # Point dans la surface apparente
    return 4 * compteur / n / np.pi # Luminosité normalisée
```

2) On crée la liste de 10 mesures de la luminosité pour calculer la moyenne et l'incertitude-type sur cet échantillon de valeurs :

```
def mesure_Kepler(x, y, alpha, n):
    mesures = [] # Liste de mesures
    for i in range(10): # On effectue les 10 mesures de luminosité
        mesures.append(mesure_luminosite(x, y, alpha, n))
    return np.mean(mesures), np.std(mesures, ddof=1)
```

On teste la fonction en calculant la luminosité normalisée en dehors du transit : on choisit par exemple `x = 2`, `y = 2` et `alpha = 0.1`, pour que la planète se trouve « en dehors » de l'étoile. Théoriquement, cette luminosité normalisée vaut exactement 1, correspondant à un nombre infini de tirages aléatoires. En fait, le calcul de la luminosité normalisée par la méthode de Monte-Carlo permet de simuler la variabilité de la mesure expérimentale.

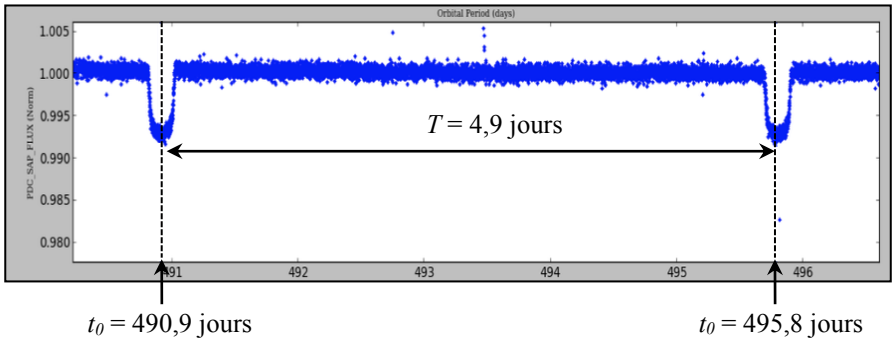
```
#Pour n = 100 tirages
mesure_Kepler(2, 2, 0.1, 100) → (0.9816676889908106, 0.037506025604356376)
#Pour n = 1 000 tirages
mesure_Kepler(2, 2, 0.1, 1000) → (1.001020930070784, 0.01359728229504637)
```

La luminosité normalisée est bien voisine de 1 et l'incertitude-type sur la mesure diminue lorsque le nombre de tirages augmente. D'après les graphes expérimentaux, la dispersion statistique des mesures correspond à un intervalle de l'ordre de 0,001. Pour atteindre une telle précision dans la simulation numérique, il faut effectuer au moins 100 000 tirages :

```
mesure_Kepler(2, 2, 0.1, 100000) → (1.0000965581613, 0.00119202728941061)
```

Le calcul devient malheureusement très long lorsque le nombre de tirages est grand.

3) On mesure la période T et la date t_0 du transit sur le graphe expérimental fourni.



On a ainsi : $T = 4{,}9$ jours et on choisit de prendre $t_0 = 495{,}8$ jours (490,9 jours convient aussi puisque le signal de transit est périodique).

Notons qu'une planète qui tourne autour de son étoile en moins de 5 jours, cela a de quoi surprendre ! La découverte des exoplanètes montre une richesse et une variété de configurations planétaires que les scientifiques n'imaginaient pas jusqu'alors !

Dans le programme de simulation numérique, on définit les grandeurs en unités SI :

```
T = 4.9 * 24 * 3600 # Période orbitale (s)
t0 = 495.8 * 24 * 3600 # Décalage temporel (s)
```

4) Kepler-7b, de masse m, est en orbite circulaire de rayon a autour de Kepler-7 de masse $M = 1{,}36 M_\odot$. La loi de la quantité de mouvement appliquée à la planète dans le référentiel Kepler-7-centrique s'écrit, en coordonnées sphériques :

$$m\vec{a} = -m\frac{v^2}{a}\vec{e_r} = -\mathcal{G}\frac{Mm}{a^2}\vec{e_r} \quad \text{où} \quad v = \frac{2\pi a}{T} \quad \text{est la vitesse orbitale de Kepler-7b.}$$

On en déduit simplement la troisième loi de Kepler, sous la forme :

$$\frac{a^3}{T^2} = \frac{\mathcal{G}M}{4\pi^2}$$

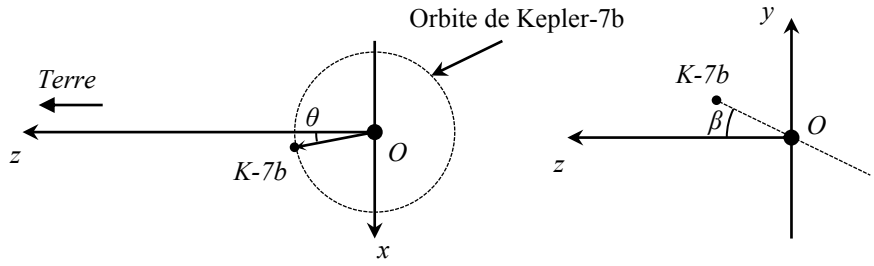
ce qui permet d'obtenir :

$$\boxed{a = \left(\frac{\mathcal{G}MT^2}{4\pi^2}\right)^{1/3}}$$

Numériquement : $a = 9{,}4.10^9$ m $= 6{,}2.10^{-2}$ au

A titre de comparaison, la planète Mercure se trouve à 0,39 au du Soleil, avec une période orbitale de 88 jours. Kepler-7b est environ 6 fois plus proche de son étoile, ce qui explique sa vitesse de révolution élevée.

5) Lors du transit vu depuis la Terre, la planète Kepler-7b se trouve pratiquement sur l'axe Oz, entre la Terre et Kepler-7.



Configuration étudiée pendant le transit

On a alors $\theta \ll 1$, de sorte que $\boxed{y = \frac{a}{R}\sin\beta}$. Pour que la planète passe effectivement devant son étoile et que le transit soit visible depuis la Terre, il faut que $|y| < 1 + \alpha \approx 1$, c'est-à-dire : $-\frac{R}{a} < \sin\beta < \frac{R}{a}$. Numériquement, $\sin\beta \approx \beta$ avec : –0,15 rad < β < 0,15 rad pour que le transit puisse être détecté. Les valeurs possibles de l'inclinaison de l'orbite vont de $-\frac{\pi}{2}$ à $\frac{\pi}{2}$, donc la probabilité de détecter Kepler-7b par la méthode du transit vaut $\frac{\Delta\beta}{\pi}$, soit voisine de 0,1 ! Seuls les systèmes exoplanétaires vus par la tranche pourront être détectés par cette méthode.

6) On définit les différents paramètres physiques de la simulation :

```
MSoleil = 1.99e30 # Masse du Soleil (kg)
RSoleil = 6.96e8 # Rayon du Soleil (m)
M = 1.36 * MSoleil # Masse de Kepler-7
R = 2.02 * RSoleil # Rayon de Kepler-7
G = 6.67e-11 # Constante de gravitation (m**3/kg/s²)
au = 1.50e11 # Unité astronomique (m)
a = 0.063 * au # Rayon orbital
```

La fonction `xyz` renvoie les coordonnées adimensionnées de la planète à l'instant `t` :

```
def xyz(t):
    xP = a / R * np.sin(2*np.pi/T*(t - t0))
    yP = 0
    zP = a / R * np.cos(2*np.pi/T*(t - t0))
    return xP, yP, zP
```

7) D'après les graphes expérimentaux, on mesure environ 10 points par heure, soit un point toutes les 6 minutes. On choisit les instants `tmin` et `tmax` de début et de fin de l'expérience pour reproduire les données fournies.

```
tmin = 490 * 24 * 3600 # Instant de début (s)
tmax = 497 * 24 * 3600 # Instant de fin (s)
dt = 3600 / 10 # Intervalle de temps entre deux points (s)
```

8) La simulation comporte une boucle `while` qui remplit les listes demandées à chaque instant de l'expérience. La condition d'entrée de boucle porte sur le temps, qui ne doit pas dépasser `tmax` à la fin de l'expérience. Avec les valeurs proposées, la boucle comportera environ $1{,}7.10^3$ itérations.

```
# Initialisation des listes
t = []
lum_norm = []
u_lum_norm = []
# Début de la boucle
temps = tmin # Initialisation du temps
while temps < tmax: # Condition d'entrée de boucle
    x, y, z = xyz(temps)
    if z > 0: # La planète est devant l'étoile : c'est le transit !
        lum, u_lum = mesure_Kepler(x, y, alpha, n)
    else: # La planète est derrière l'étoile
```

```
        lum, u_lum = mesure_Kepler(2, 2, alpha, n) # La planète est
loin !
    # Remplissage des listes
    t.append(temps)
    lum_norm.append(lum)
    u_lum_norm.append(u_lum)
    temps = temps + dt
# Conversion en tableaux (pour une manipulation pratique des données)
t = np.array(t)
lum_norm = np.array(lum_norm)
u_lum_norm = np.array(u_lum_norm)
```

9) On teste le programme avec les valeurs proposées :

```
alpha = 0.4
n = 1000
```

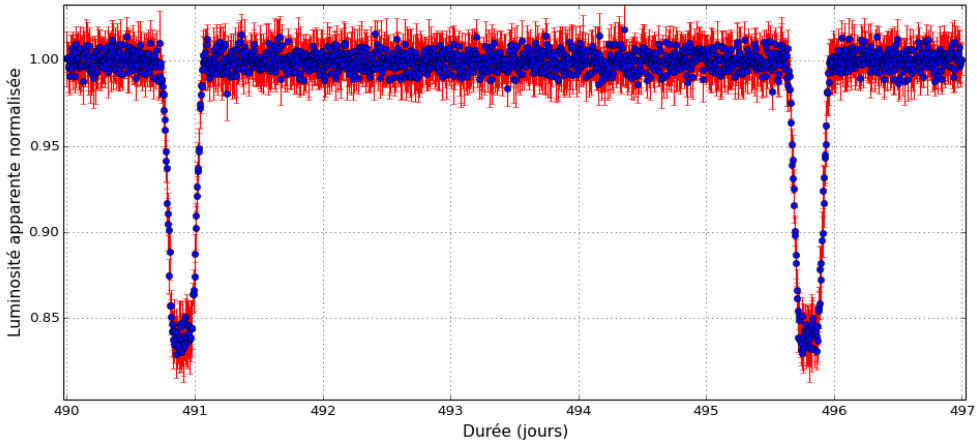
On rajoute dans la boucle précédente l'affichage d'un indicateur de progression :

```
while temps < tmax :
    ...
    print(round((temps - tmin)/(tmax - tmin)*100, 1), '%')
```

On trace enfin la luminosité en fonction du temps (exprimé en jours pour plus de lisibilité) avec la syntaxe proposée pour l'affichage des barres d'erreur :

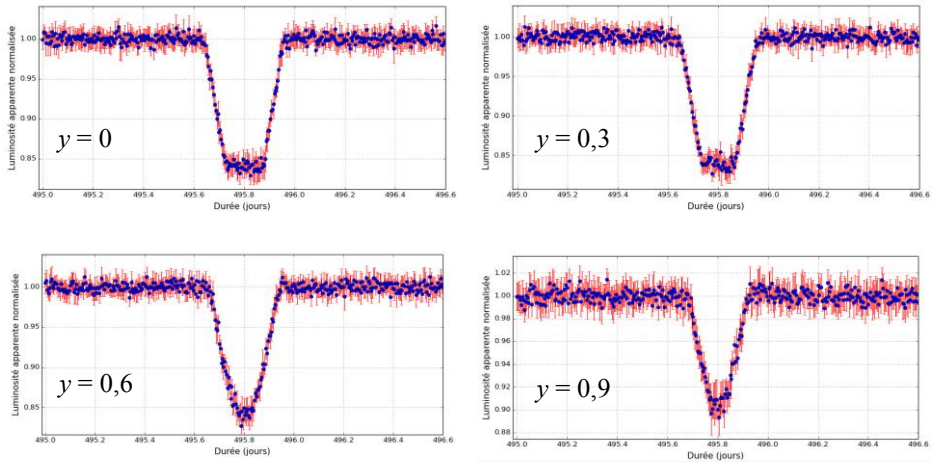
```
plt.xlabel('Durée (jours)')
plt.ylabel('Luminosité apparente normalisée')
plt.errorbar(t/(24*3600), lum_norm, yerr=u_lum_norm, fmt='bo')
```

On obtient le graphe suivant :



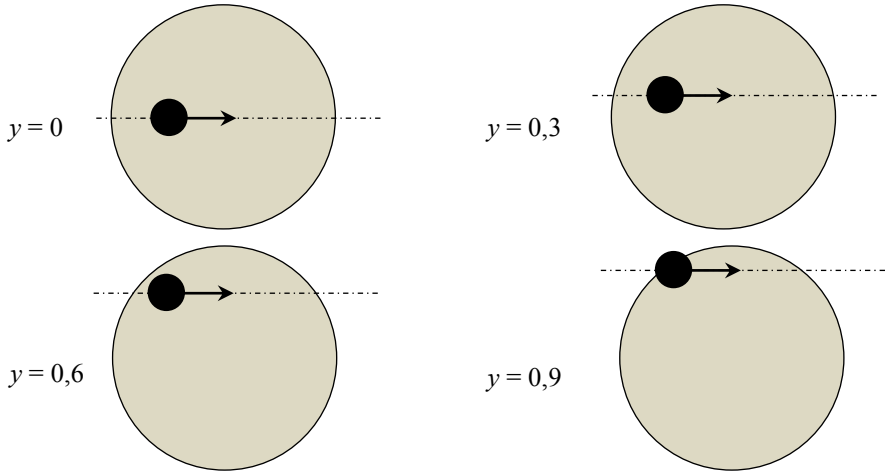
Le résultat de cette simulation montre bien l'existence du transit de la planète avec une diminution périodique de la luminosité de l'étoile. Cependant la dispersion des mesures obtenue est bien plus grande que celle observée expérimentalement et la baisse de luminosité lors du transit est trop importante. Il faut maintenant ajuster les paramètres de la simulation.

10) La forme du profil de luminosité dépend de l'angle β, par l'intermédiaire de y :



Profils de luminosité pendant le transit pour différentes valeurs de y

Ces situations correspondent en fait à différentes positions de la planète dans le disque apparent de l'étoile et influent notamment sur la durée du transit, la forme du profil de luminosité et la valeur de la luminosité minimale :



D'autre part, on peut prévoir la luminosité minimale de l'étoile lors du transit, c'est-à-dire lorsque la planète est entièrement devant l'étoile (en supposant que la valeur de l'inclinaison β permette cette situation). La surface apparente de l'étoile vaut en effet dans ce cas $\pi(1-\alpha^2)R^2$. En dehors du transit, la surface apparente vaut πR^2 si bien que la luminosité apparente normalisée minimale au moment du transit s'écrit : $L_{\min} = \dfrac{\pi(1-\alpha^2)R^2}{\pi R^2} = 1-\alpha^2$. On en déduit la valeur du rayon adimensionné α de la planète

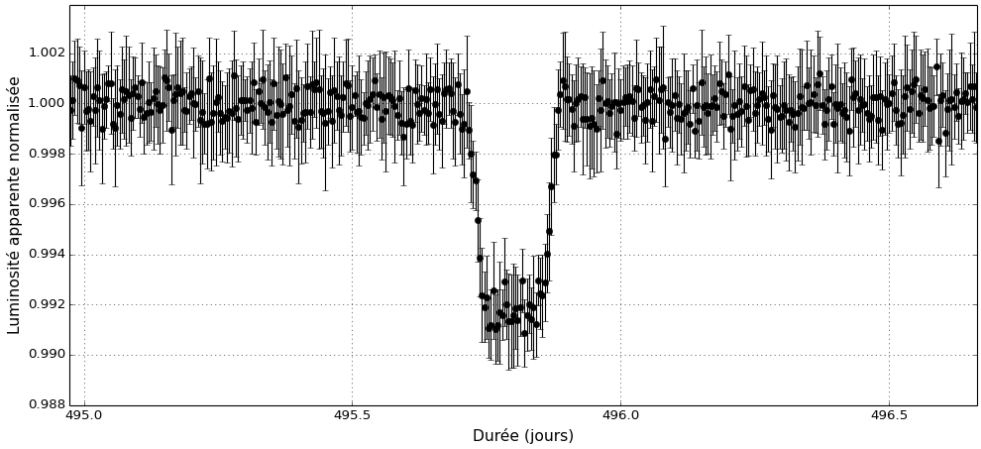
sous la forme : $\alpha = \sqrt{1 - L_{\min}}$. On lit sur le graphe expérimental $L_{min} = 0{,}992$ ce qui correspond à $\alpha = 0{,}09$.

Enfin, on impose `n = 100 000` pour que la mesure de la luminosité par la méthode de Monte-Carlo donne une incertitude-type comparable aux données expérimentales.

Après quelques essais, on peut ajuster la forme du profil de luminosité au moment du transit en changeant la valeur de `y`. On choisit alors `y = 0.8`, ce qui correspond à une inclinaison de l'orbite de Kepler-7b d'un angle $\beta = 0{,}12$ rad = 6,9°.

Pour limiter le temps de calcul, on étudie le transit entre les instants 494,5 jours et 497 jours, comme sur le cliché expérimental fourni.

Après un temps de calcul conséquent (qui fera sourire le lecteur de 2030 !), on obtient le graphe suivant :



Simulation numérique du transit de Kepler-7b

On retrouve par la simulation numérique le profil de luminosité mesuré expérimentalement par le télescope Kepler et ses instruments.

11) Le rayon de Kepler-7b vaut $R_{K7b} = \alpha R = 1{,}3.10^5$ km , c'est-à-dire 1,8 fois le rayon de Jupiter. Avec de telles caractéristiques, Kepler-7b fait partie d'une catégorie d'exoplanètes appelée *Jupiter chaud* (*Hot Jupiter* en anglais) : il s'agit d'une planète géante (probablement gazeuse), qui tourne très vite autour de son étoile et portée à très haute température à cause de la proximité avec celle-ci. La température à sa surface est estimée à plus de 1 500 K !

12) La plus petite baisse de luminosité apparente normalisée détectable est de l'ordre de l'incertitude-type sur la mesure, c'est-à-dire environ 0,001. La luminosité minimale pendant le transit vaudrait alors : $L_{min} = 0{,}999$ correspondant à la plus valeur de α détectable : $\alpha = \sqrt{1 - L_{\min}} = 0{,}03$. Une telle planète autour de Kepler-7 aurait un rayon de l'ordre de 4.10^4 km, soit plus petite que Saturne (et légèrement plus grosse que Uranus ou Neptune).

4. Dans l'œil de Kepler

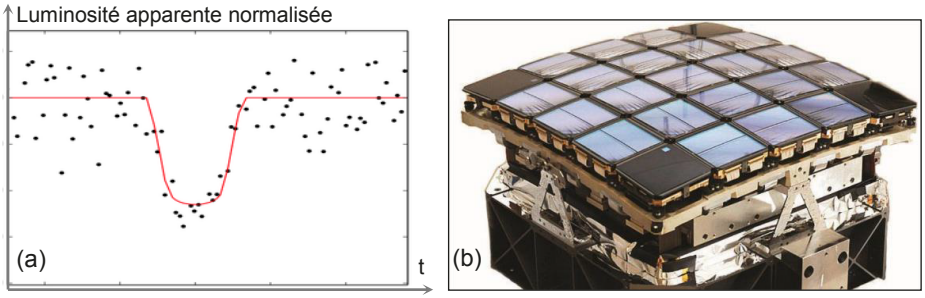
MOTS-CLES : Astronomie, traitement d'images, incertitudes

Capacité numérique (filières PCSI, MPSI, PTSI, MP2I) : Utiliser la fonction `random.normal` de la bibliothèque `numpy` (sa spécification étant fournie) pour simuler un processus aléatoire.

Le télescope spatial Kepler, développé par la NASA, a permis la détection de 2 662 exoplanètes par la méthode des transits : le passage d'une planète devant son étoile se traduit, depuis la Terre, par une légère baisse de la luminosité apparente de celle-ci, ce qui permet de déterminer la période de révolution et la taille de l'exoplanète. La mission Kepler, initialement prévue de 2009 à 2013, a pu être prolongée jusqu'en 2018 (mission K2), date à laquelle le télescope s'est mis en sécurité par manque de carburant. Depuis 2021, le télescope spatial James Webb (*JWST – James Webb Space Telescope*) poursuit l'observation spatiale avec des instruments encore plus performants. On se propose de comprendre le traitement des données effectué pour détecter le transit d'une exoplanète à partir des images numériques reçues par le télescope Kepler.

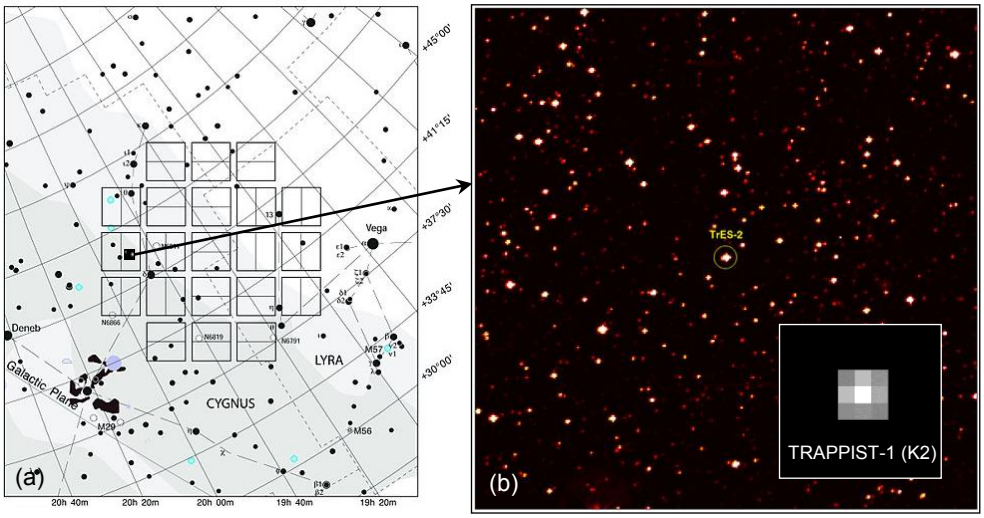
DOCUMENT : Données sur la mission Kepler d'après « www.wikipedia.fr »

Le champ d'observation de 115 deg^2 de Kepler lui donne une probabilité bien plus élevée de détecter des planètes de type terrestre que le télescope spatial Hubble, qui a un champ d'observation de seulement 10 minutes d'arc^2. De plus, Kepler est destiné à la détection de transits planétaires, tandis que le télescope spatial Hubble est utilisé pour aborder une large variété de questions scientifiques, et n'observe que rarement un seul champ stellaire en continu. Parmi environ un demi-million d'étoiles situées dans le champ d'observation de Kepler, à peu près 150 000 étoiles ont été sélectionnées pour observation, et elles sont observées simultanément, le vaisseau mesurant les variations de leur éclat toutes les 30 minutes. Cela offre une meilleure chance de voir un transit. De plus, la probabilité de 1 sur 215 signifie que si 100 % des étoiles observées avaient le même diamètre que le Soleil, et que chacune avait une planète tellurique de type terrestre sur une orbite identique à celle de la Terre, Kepler en trouverait environ 465 ; mais si seulement 10 % des étoiles observées sont de la sorte, alors il en découvrirait 46. La mission est bien adaptée pour déterminer la fréquence de planètes similaires à la Terre orbitant autour d'autres étoiles.



(a) Signal de transit caractéristique du passage d'une exoplanète devant son étoile
(b) Le détecteur comporte 42 capteurs CCD et un total de 95 millions de pixels
(Crédit : Dr. David Koch, NASA, domaine public)

Le télescope Kepler observe en permanence la même zone du ciel, au voisinage des constellations du Cygne et de la Lyre, dans la direction d'un bras spiral de notre galaxie. Chaque capteur CCD reçoit la même image du ciel profond au cours du temps, permettant l'analyse en continu des variations de la luminosité apparente de quelques étoiles choisies.



(a) Schéma de la région de l'espace observée par Kepler
(chaque rectangle correspond à la portion vue par un des détecteurs CCD)
(b) Détail du champ d'observation. TrEs-2 (ou Kepler-1), première étoile observée par le télescope spatial Kepler, possède une planète de type Jupiter chaud
(en médaillon) Trappist-1 (observée par la mission K2) comporte 7 planètes en orbite
(Photo : NASA - Domaine public – https://fr.wikipedia.org/wiki/Kepler_(télescope_spatial))

La détection d'exoplanètes par le télescope Kepler est essentiellement basée sur le traitement d'images numériques. On utilisera à titre d'exemple l'image (b) représentée ci-dessus :

> Télécharger l'image `Kepler_First_Light_Detail_TrES-2.jpg` à partir du site Wikipédia (cliquer sur l'image pour l'agrandir, puis clic droit et *Enregistrer l'image sous…*)

> Convertir en ligne l'image au format `.png` (Mots-clés : *convertir jpg en png*) et l'enregistrer dans un dossier de travail (par exemple : `c:\La physique en applications\` à adapter si besoin) sous le nom : `Kepler_First_Light_Detail_TrES-2.png`

Les instructions Python suivantes permettent de charger l'image en mémoire et de l'afficher avec la librairie `matplotlib.pyplot` :

```
dossier= 'c:/La physique en applications/' # Nom du dossier (à adapter)
image = plt.imread(dossier + 'Kepler_First_Light_Detail_TrES-2.png')
CCD = np.uint8(255*image) # Conversion des données en entiers 8 bits
plt.imshow(CCD, cmap='gray', interpolation='None')
plt.show() # Affichage de l'image
```

L'objet `CCD` obtenu est un tableau `np.array` que l'on interprète comme une image en couleurs. Chaque pixel est un triplet `[R,V,B]` d'entiers des composantes de rouge, vert et

bleu. On désactive la fonction d'interpolation lors de l'affichage pour pouvoir visualiser les pixels de l'image. L'option `cmap='gray'` est utilisée pour des images en niveaux de gris.

On donne quelques instructions utiles pour le traitement d'une image `a` :

```
np.zeros((n, m, c), dtype=np.uint8) # Tableau n lignes et m colonnes dont
chaque élément est un tableau de c entiers 8 bits rempli avec des zéros
a.shape # tuple donnant la taille du tableau pour chacune de ses
dimensions
a.dtype # donne le type des éléments du tableau a
a[i, j] # Pixel [R,V,B] situé en ligne i et colonne j
np.sum(a) # somme des éléments de a (conversion en uint32 ou float32)
np.mean(a) # moyenne des éléments de a (résultat en float32)
np.max(a) # renvoie la valeur maximale contenue dans le tableau a
a2 = np.copy(a) # Fournit une copie indépendante du tableau a
```

Chaque composante de couleur d'un pixel est de type `np.uint8`, c'est-à-dire entier 8 bits non signé (*unsigned integer*). On rappelle que les entiers 8 bits sont codés en binaire de 00000000 à 11111111 pour des valeurs décimales allant de 0 à 255. En cas de dépassement de capacité, le codage en `np.uint8` se fait modulo 256.

1) Charger et afficher l'image `CCD`.

2) Calculer en Mo la taille de l'image dans la mémoire de l'ordinateur. Déterminer le nombre maximal de couleurs que l'on peut obtenir avec ce format, ainsi que le nombre maximal de couleurs possibles dans cette image.

3) Afficher la valeur `[R, V, B]` d'un pixel `(i0, j0)` de l'étoile TrEs-2 repérée sur l'image. Calculer la somme `CCD[i0, j0, 0] + CCD[i0, j0, 1] + CCD[i0, j0, 2]` des composantes de couleur et comparer avec `np.sum(CCD[i0, j0])`. Commenter.

Pour simplifier la détection automatique des étoiles les plus brillantes de l'image, il faut tout d'abord la convertir en niveaux de gris, c'est-à-dire en une image de même taille mais dont l'intensité de chaque pixel est la moyenne arrondie des composantes `[R, V, B]` du pixel en couleur d'origine.

4) Ecrire la fonction d'entête :

```
def niveau2gris(a_couleur):
```

qui renvoie une nouvelle image correspondant à la conversion en niveaux de gris de l'image `a_couleur`.

On appelle barycentre d'une image en niveaux de gris le pixel dont la position est la moyenne des positions des pixels de l'image, pondérée par l'intensité de chaque pixel, et dont l'intensité est la somme des intensités des pixels de l'image.

5) Ecrire la fonction d'entête :

```
def barycentre_image(a, i0, j0, L):
```

qui renvoie la position et l'intensité du barycentre de la zone de taille `L x L` de l'image `a` centrée sur `(i0, j0)`.

Pour déterminer la position d'une étoile, on propose de partir d'un pixel quelconque de l'étoile, dont l'intensité est supérieure à un seuil fixé, donc probablement proche du centre de l'étoile. On cherche alors le barycentre de la zone de taille `L x L` centrée sur ce pixel. Ce

barycentre est le centre d'une nouvelle zone dont on cherche à nouveau le barycentre, et ainsi de suite jusqu'à ce que le barycentre de la zone soit situé au centre de celle-ci. On assimile enfin ce barycentre au centre de l'étoile cherchée et son intensité à la luminosité de l'étoile.

6) Ecrire la fonction d'entête :

```
def barycentre_etoile(a, i, j, L):
```

qui renvoie la position et l'intensité du barycentre d'une étoile dont on connaît la position `(i, j)` d'un pixel proche du centre.

7) On recherche les pixels de l'image dont l'intensité est supérieure à une valeur seuil, définie par le réel p (entre 0 et 1) multiplié par l'intensité maximale présente dans l'image (entre 0 et 255). Ecrire la fonction d'entête :

```
def trouve_etoiles(a, p, L, netoiles):
```

qui cherche la position et la luminosité de toutes les étoiles de l'image `a` pour un seuil `p` donné. La fonction devra renvoyer la liste des `netoiles` les plus brillantes. On expliquera pourquoi il faut travailler avec une copie de l'image de laquelle on supprimera les étoiles au fur et à mesure de la recherche.

On pourra utiliser l'instruction `u.sort()` qui modifie et ordonne la liste `u` dans l'ordre croissant de ses éléments (si les éléments sont des listes ou des tuples, l'ordre utilisé est l'ordre lexicographique).

Pour vérifier que la détection des étoiles s'est bien passée, on souhaite afficher l'image originale en couleur sur laquelle on vient placer une croix à l'endroit de chaque étoile détectée.

8) Ecrire la fonction d'entête :

```
def croix(a_couleur, etoiles, couleur):
```

qui renvoie une copie de l'image `a_couleur` sur laquelle on a repéré les étoiles de la liste `etoiles` avec une croix. `couleur` est un triplet `[R, V, B]` définissant la couleur utilisée pour le tracé.

9) Donner la liste d'instructions permettant de visualiser les dix étoiles les plus brillantes de l'image, avec un seuil défini à $p = 0{,}98$ et une zone de recherche de largeur $L = 15$ pixels. Tester le programme en changeant le nombre d'étoiles les plus brillantes. Commenter le résultat obtenu.

Une fois les étoiles sélectionnées, le logiciel de traitement de la mission Kepler va mesurer et enregistrer la luminosité apparente de chaque étoile toutes les 30 minutes à la recherche d'un signal de transit. Du fait de la variabilité des différents paramètres de l'étoile et du bruit de détection, la luminosité apparente normalisée d'une étoile varie aléatoirement autour d'une valeur moyenne, même en l'absence de transit.

L'exercice *Exoplanète en transit* étudie en détails un signal de transit. On se propose ici de simuler simplement le signal reçu pour mettre en évidence les critères de détection.

On rappelle quelques instructions de traitement statistique de données :

```
np.random.normal(x_moy, u_x, n) # renvoie un tableau np.array contenant n
valeurs de moyenne x_moy avec un écart-type u_x
```

```
np.mean(x) # renvoie la moyenne d'une liste x de valeurs
np.std(x, ddof=1) # renvoie l'écart-type expérimental (ou incertitude-type) associé à la liste x de valeurs
plt.errorbar(x, y, yerr=u_y, fmt='ko') # trace le graphe de la liste y avec l'incertitude-type u_y en fonction de la liste x avec des points expérimentaux et les barres d'incertitude associées
```

10) La luminosité apparente normalisée est le rapport de la luminosité apparente de l'étoile sur la luminosité apparente de l'étoile en l'absence de transit. Déterminer la valeur moyenne attendue de la luminosité apparente normalisée d'une étoile en dehors d'une phase de transit. Donner les instructions permettant de créer les tableaux `t` et `lum` contenant respectivement les instants d'acquisition du signal pendant 10 jours et les valeurs simulées de la luminosité apparente relative pour une étoile. On prendra l'heure comme unité de temps et un écart-type relatif de 2 % pour la valeur de la luminosité apparente normalisée.

Les valeurs contenues dans `t` et `lum` sont maintenant supposées avoir été mesurées par le système de détection d'une étoile en l'absence de transit.

11) Calculer la luminosité apparente normalisée moyenne sur 10 jours et l'incertitude-type sur chaque mesure. Quelle est l'incertitude-type associée à la moyenne obtenue ? Commenter. Tracer l'évolution de la luminosité apparente relative de l'étoile en fonction du temps avec les barres d'incertitude associées.

12) Définir et tracer l'écart normalisé (encore appelé `z-score`) associé à chaque point de mesure en fonction du temps. Quelle est la probabilité que cet écart normalisé soit supérieur à 2 ? supérieur à 3 ? Combien de mesures cela représente-t-il sur 10 jours d'acquisition ?

Une étoile comme TrEs-2 possède une exoplanète (appelée TrEs-2b) en orbite à la période de 2,47 jours. La durée du transit est de 5,9 h. Une telle planète est appelée Jupiter chaud (ou *Hot Jupiter*) du fait de sa taille (semblable à celle de Jupiter) et de sa température élevée due à la proximité avec son étoile (qui explique sa faible période orbitale). Pendant le transit, la luminosité apparente normalisée de l'étoile vaut 90 % de la luminosité apparente normalisée moyenne en absence de transit, avec les mêmes valeurs d'incertitudes-types.

13) Créer un nouveau tableau des luminosité apparentes normalisées permettant de simuler le signal de transit observé pendant 10 jours.

14) Tracer la luminosité apparente normalisée de l'étoile au cours du temps. A quelle condition le transit de l'exoplanète pourra-t-il être détecté ? Comment peut-on être sûr qu'il s'agit bien d'un signal de transit et non d'un écart statistique ?

CORRIGÉ

1) On suit les étapes proposées pour télécharger, renommer et afficher l'image. Attention à ne pas confondre `'\'` et `'/'` dans le nom du dossier.

2) On affiche dans un premier temps la taille de l'image :

```
CCD.shape → (493, 492, 3)
```

L'image comporte 493 lignes, 492 colonnes et 3 couches correspondant aux composantes de couleur `[R,V,B]`. Chaque composante est un entier 8 bits, de taille 1 octet par définition. Chaque pixel nécessite donc 3 octets de mémoire pour coder les 3 composantes de couleur. Avec ses 242 556 pixels, l'image a une taille totale de 0,73 Mo dans la mémoire vive de l'ordinateur. Chaque composante peut valoir 0 à 255, soit 256 valeurs différentes. Il existe donc 256^3 = 16,8 millions de couleurs `[R,V,B]` possibles avec ce format. En revanche, il ne pourra exister dans l'image plus de couleurs que de pixels, c'est-à-dire ici environ $2{,}4.10^5$ couleurs différentes au maximum.

3) On zoome sur l'étoile TrEs-2 de l'image et on affiche le pixel situé en ligne 244 et colonne 246 :

```
CCD[244, 246] → array([241, 255, 254], dtype=uint8)
```

Les valeurs `[R,V,B]` sont proches de 255 car le pixel est très lumineux.

On évalue la somme des composantes de couleur :

```
CCD[244, 246, 0] + CCD[244, 246, 1] + CCD[244, 246, 2] → 238
```

Ce résultat est surprenant, dans la mesure où 241+255+254 = 750 ! D'ailleurs ce calcul est parfois accompagné du message d'avertissement :

```
RuntimeWarning: overflow encountered in ubyte_scalars
```

En fait, chaque composante est codée avec des entiers 8 bits, de 0 à 255. Le résultat attendu pour cette somme (c'est-à-dire 750) dépasse la capacité du codage 8 bits. On pourra vérifier que 750 = 238 + 2 x 256 = 238 (modulo 256), qui est bien le résultat renvoyé. Les calculs avec des entiers 8 bits ne devront pas dépasser la capacité sous peine de fausser les résultats.

Lorsque l'on utilise l'instruction `np.sum()`, on obtient :

```
np.sum(CCD[244, 246]) → 750
```

Le résultat est maintenant correct car le calcul se fait en augmentant la capacité de codage (ici en entier 32 bits). Il faudra y penser lors de la manipulation des entiers 8 bits.

4) On va d'abord créer une image de la même taille que l'image d'origine, mais ne possédant qu'une seule couche d'intensité. On calcule la moyenne des composantes de couleur à l'aide de l'instruction `np.mean()` pour éviter les dépassements de capacité. La conversion de la moyenne en entier est automatique puisque la nouvelle image est codée en entiers 8 bits.

```
def niveau2gris(a_couleur):
    n, m, c = a_couleur.shape # Taille de l'image d'origine
    NB = np.zeros((n, m), dtype=np.uint8) # Nouvelle image
    for i in range(n):
        for j in range(m):
            NB[i, j] = np.mean(a_couleur[i, j]) # Moyenne
    return NB
```

5) On utilise la définition du barycentre pour calculer la moyenne pondérée des positions des pixels en balayant la zone de largeur `L x L`. On prendra garde aux bornes utilisées puisque `(i0, j0)` est au centre de la zone.

```
def barycentre_image(a, i0, j0, L):
    i2, j2, intensite = 0, 0, 0 # Initialisation
    for i in range(i0 - L//2, i0 + L//2): # On balaye la zone
        for j in range(j0 - L//2, j0 + L//2):
            intensite = intensite + a[i, j]
            i2 = i2 + a[i, j]*i
            j2 = j2 + a[i, j]*j
    return i2//intensite, j2//intensite, intensite
```

6) Partant du pixel de départ, on calcule la position du barycentre de la zone centrée sur la position précédente. La boucle `while` poursuit les itérations jusqu'à ce que le barycentre ne bouge plus.

```
def barycentre_etoile(a, i, j, L):
    i1, j1 = i, j
    i2, j2, intensite = barycentre_image(a, i1, j1, L)
    while (i2, j2) != (i1, j1):
        i1, j1 = i2, j2
        i2, j2, intensite = barycentre_image(a, i1, j1, L)
    return i2, j2, intensite
```

7) On calcule tout d'abord le seuil de détection en intensité, par le produit `p*np.max(a)`, qui permet de sélectionner les pixels les plus lumineux. On balaye ainsi l'image en évitant la zone de largeur `L` tout autour pour permettre le calcul de la position du barycentre. Dès que l'on a détecté une étoile, on met les pixels de la zone de détection correspondante à 0 sur la copie de l'image pour éviter de compter plusieurs fois les pixels lumineux de cette étoile.

On stocke la position et la luminosité de chaque étoile dans une liste, en mettant la valeur de la luminosité en premier, sous la forme `[intensite, iE, jE]`, pour pouvoir mettre ensuite les termes dans l'ordre lexicographique (donc de luminosité croissante) avec l'instruction `liste.sort()`. Enfin on récupère la position des étoiles dans la liste `etoiles` en sélectionnant `netoiles` termes de `liste` à partir de la fin.

```
def trouve_etoiles(a, p, L, netoiles):
    seuil = p * np.max(a) # Calcul du seuil d'intensité
    n, m = a.shape
    a2 = np.copy(a)
    liste = []
    for i in range(L, n - L): # On balaye l'image sauf les bords
        for j in range(L, m - L):
            if a2[i, j] >= seuil: # Pixel brillant
                iE, jE, intensite = barycentre_etoile(a, i, j, L)
```

```
                liste.append([intensite, iE, jE]) # Etoile détectée
                a2[iE - L//2 : iE + L//2, jE - L//2 : jE + L//2] = 0
                # Mise à zéro des pixels de la zone de recherche
    liste.sort() # On classe les étoiles par luminosité croissante
    etoiles = []
    for k in range(netoiles):
        intensite, i, j = liste[-1 - k] # On commence par la fin
        etoiles.append([i, j])
    return etoiles # Liste des étoiles sélectionnées
```

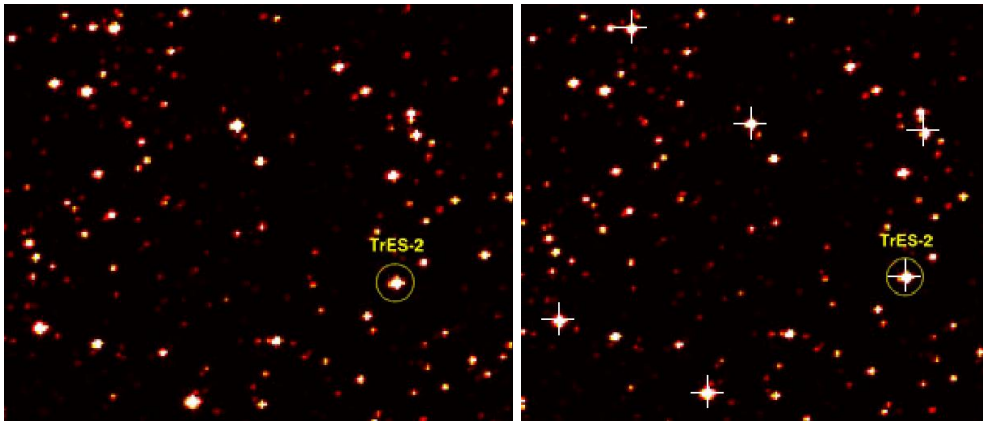
8) On va tracer une croix à l'endroit des étoiles sélectionnées sur une copie de l'image de départ. Il suffit de balayer les pixels horizontalement et verticalement de part et d'autre de chaque étoile et de remplacer la valeur des pixels par `couleur`. On choisit ici des croix de 20 pixels de large.

```
def croix(a_couleur, etoiles, couleur):
    taille = 10 # Demi-largeur de la croix
    a2 = np.copy(a_couleur) # Copie de l'image
    for k in range(len(etoiles)):
        i0, j0 = etoiles[k] # Position d'une étoile
        for i in range(i0 - taille, i0 + taille):
            a2[i, j0] = couleur # Trait vertical
        for j in range(j0 - taille, j0 + taille):
            a2[i0, j]=couleur # Trait horizontal
    return a2
```

9) On reprend les différentes fonctions définies jusqu'à présent :

```
image = plt.imread(dossier
                   + 'Kepler_First_Light_Detail_TrES-2.png')
CCD = np.uint8(255*image) # Image originale en couleur
CCD_NB = niveau2gris(CCD) # Image en niveaux de gris
etoiles = trouve_etoiles(CCD_NB, 0.98, 15, 10) # Liste des 10 étoiles
CCD_croix = croix(CCD, etoiles, [255, 255, 255])
plt.imshow(CCD_croix, cmap='gray', interpolation='None')
plt.show()
```

On a choisi de tracer les croix en blanc (`[255,255,255]`). On a représenté ci-après un détail de l'image originale et avec la détection automatique des étoiles les plus brillantes.



L'algorithme proposé fonctionne correctement. Dans certains cas (étoiles proches les unes des autres ou au bord de l'image), il faudra procéder à un réajustement manuel de la position.

10) Par définition, la luminosité apparente normalisée de l'étoile en l'absence de transit vaut exactement 1 (sans dimension).

On enregistre le signal toutes les 30 minutes pendant 10 jours :

```
tmax = 10 * 24 # h Durée d'acquisition
dt = 0.5 # h Pas de temps
N = int(tmax/dt) + 1 # Nombre de points
t = np.array([k * dt for k in range(N)]) # Tableau des instants
```

Un écart-type relatif de 2 % sur la luminosité apparente normalisée se traduit ici par un écart-type absolu de 0,02. On crée directement un tableau de `N` valeurs de luminosité apparente normalisée avec l'instruction `np.random.normal()` :

```
lum=np.random.normal(1., 0.02, N)
```

11) On évalue simplement la moyenne et l'écart-type sur chaque terme du tableau `lum` comme si on venait de récupérer les mesures expérimentales :

```
np.mean(lum) → 0.99956958621307557 # Moyenne
np.std(lum, ddof=1) → 0.021040803575076282 # Incertitude-type
```

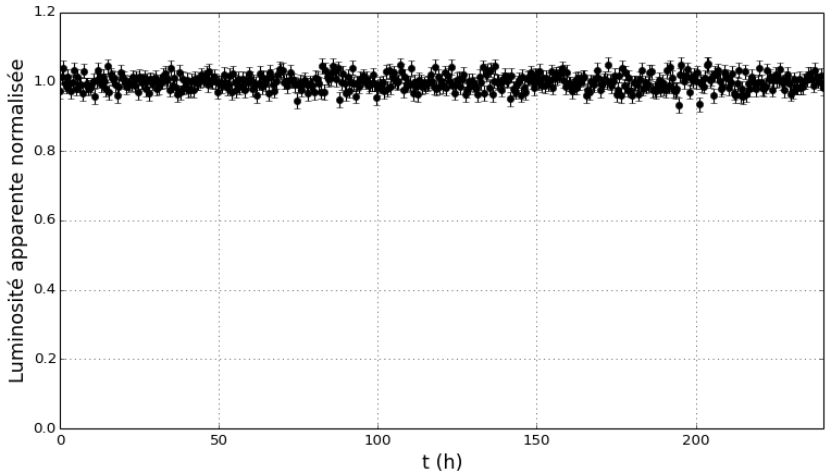
L'incertitude-type sur la moyenne dépend du nombre de points de mesure et sera d'autant plus faible que l'on calcule la moyenne sur un grand nombre de valeurs :

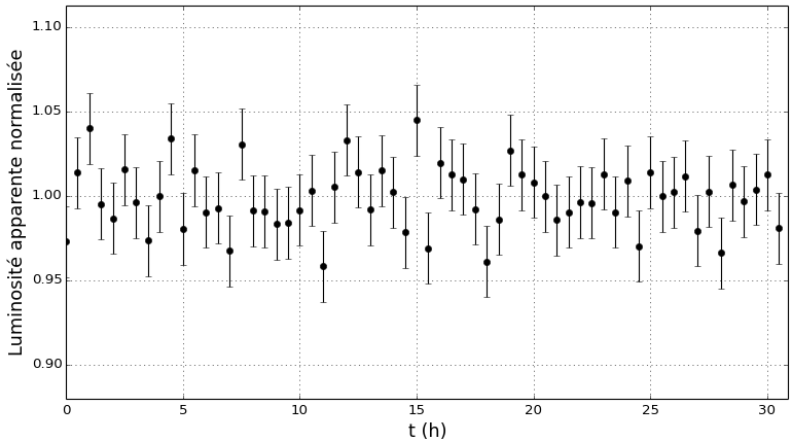
```
np.std(lum, ddof=1)/np.sqrt(N) → 0.00095937806331386618
```

On visualise le graphe des mesures expérimentales simulées avec les barres d'incertitudes identiques pour toutes les valeurs :

```
u_lum = np.std(lum, ddof=1)
plt.errorbar(t, lum, yerr=u_lum, fmt='ko')
```

Les graphes ci-après montrent bien la dispersion des mesures autour de la moyenne :





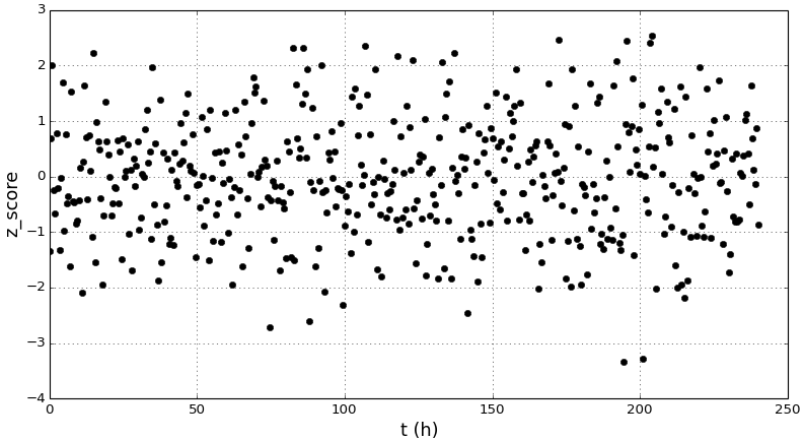
12) L'écart normalisé (ou `z-score`) d'une valeur x est défini par la relation :

$$\frac{x - x_{moy}}{u(x)}$$

où x_{moy} est la valeur moyenne et $u(x)$ l'incertitude-type.

L'écart normalisé permet de visualiser facilement les écarts à la moyenne.

```
zscore = (lum - np.mean(lum))/u_lum
plt.plot(t, zscore, 'ko')
```



Le nuage de points se répartit de part et d'autre de la valeur nulle. Statistiquement, la probabilité que l'écart normalisé soit inférieur à 2 (en valeur absolue) est de 95 %. Cette probabilité passe à 99,7 % pour un écart-normalisé inférieur à 3.

La probabilité d'avoir un `z-score` supérieur à 2 vaut donc 5 %, ce qui représente environ 24 points sur les 481 valeurs obtenues en 10 jours. Pour un `z-score` supérieur à 3, la probabilité passe à 2 %, ce qui représente environ 10 valeurs.

13) On utilise une fonction qui modélise simplement le signal de transit à partir de la période orbitale T et de la durée du transit Δt. Le signal étant périodique, il faut d'abord

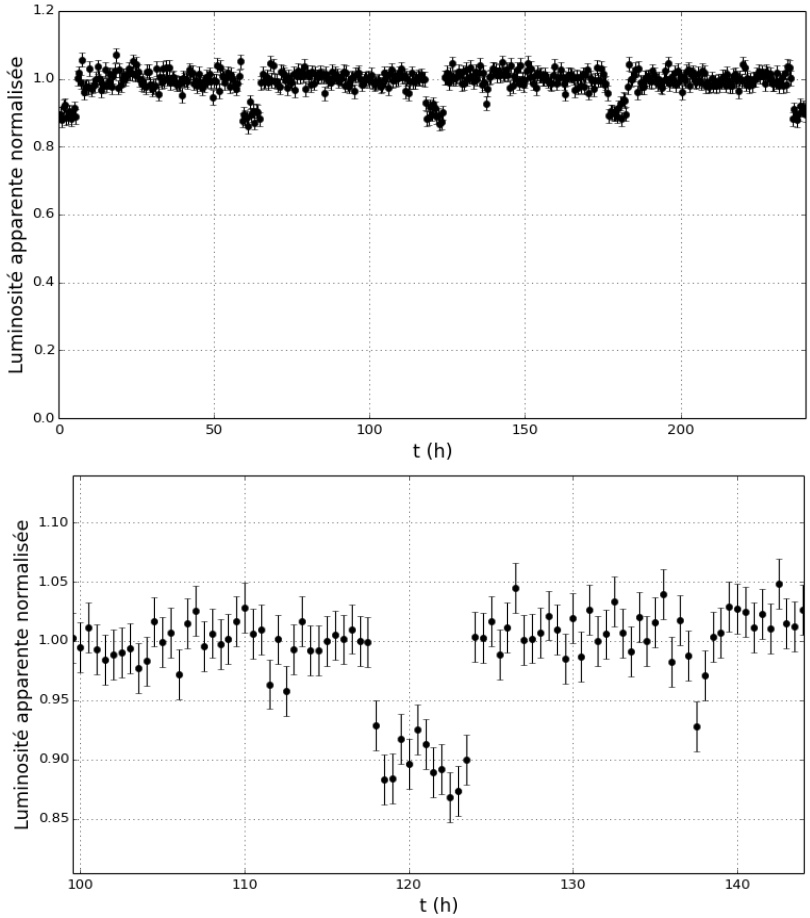
ramener l'instant t considéré entre 0 et T en calculant par exemple le reste de la division entière de t par Δt.

```
def transit(t, T, delta_t):
    t0 = int(t) % int(T) # t0 est compris entre 0 et T
    if t0 <= delta_t:
        return np.random.normal(0.9, 0.02) # Phase de transit
    else:
        return np.random.normal(1., 0.02) # Hors du transit
```

On définit la période et la durée du transit et on crée le tableau des valeurs de luminosité apparente normalisée au cours du temps :

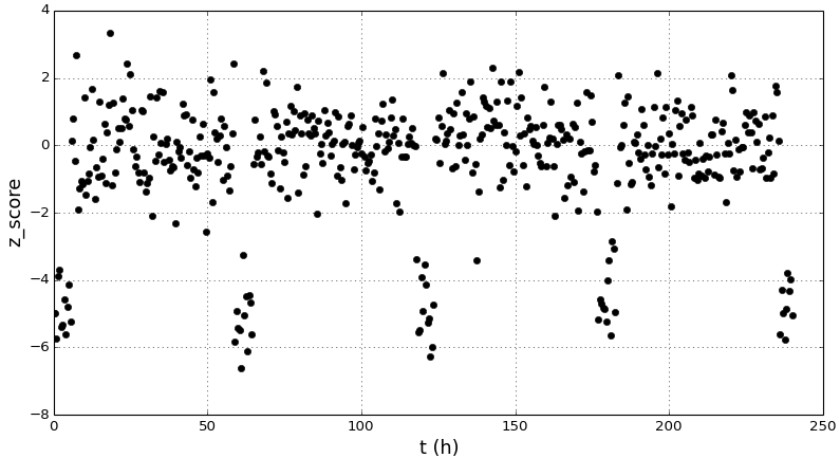
```
T = 2.47 * 24 # h Période orbitale
delta_t = 5.9 # h Durée du transit
lum_transit = np.array([transit(t[k], T, delta_t) for k in range(N)])
```

14) On trace la luminosité apparente normalisée de l'étoile sur une durée de 10 jours :



On retrouve avec cette simulation simple la forme typique d'un signal de transit.

Pour pouvoir détecter le transit, la variation de luminosité apparente normalisée doit être supérieure à l'incertitude-type sur chaque mesure. Afin de mettre en place un critère de détection, on trace l'écart normalisé par rapport à la situation sans transit. Pour que le transit sorte du bruit statistique, on peut imposer un `z-score` minimal (en valeur absolue). On a vu qu'il y a 2 % de chance qu'une mesure ait un `z-score` supérieur à 3 même en l'absence de transit. Cet écart statistique pourrait alors être confondu avec un signal de transit. En prenant un `z-score` supérieur à 4, on limite ce genre de situation, tout en imposant un signal de transit plus grand.



Dans tous les cas, une mesure de transit ne pourra être confirmée que si le phénomène se répète périodiquement dans le temps. Un transit isolé peut représenter le passage de n'importe quel objet entre le télescope et l'étoile, tandis qu'un transit périodique est le signe d'un mouvement orbital. En pratique, après deux transits, une première mesure de période orbitale est possible. On a confirmation de l'existence de l'exoplanète lorsqu'un nouveau signal de transit est effectivement observé une période plus tard. D'une durée de 9 ans, la mission Kepler-K2 a pu détecter des exoplanètes avec une période orbitale maximale d'environ 3 ans. La détection d'exoplanètes très éloignées de leur étoile, avec des grandes périodes orbitales, nécessite beaucoup de temps avec la méthode des transits.