AP3

Algorithmique: figures imposées

Code *AP3*.1: Algorithme pour calcul de la moyenne et de la variance d'une liste de nombres

```
# Itérer sur tous les éléments
# Stocker la somme et la somme des carrés
# Calcul de la moyenne: somme/N
# Calcul de la variance: On utilise le fait que
#    <(u-<u>)^2> = <(u^2 - 2*u*<u> + <u>^2)>
#                = <u^2> - 2*<u>*<u> + <u>^2
#                = <u^2> - <u>^2
```

Code *AP3*.2: Implémentation de l'algorithme de calcul de moyenne et variance

```
def moyenne_et_variance(liste):
    """(list(float)) -> float,float
    Calcul de la moyenne <u> = Sum u / N et de la variance <(u-<u>)^2>
    d'une liste de nombres en une seule passe. Renvoie un doublet
    (moyenne,variance).

    >>> moyenne_et_variance([10,10,10,10,10])
    (10.0, 0.0)
    >>> moyenne_et_variance([10,14,10,14,10,14])
    (12.0, 4.0)
    """
    somme,somme_carres = 0.0,0.0    # Initialisation des mémoires
    for u in liste:                 # Itérer sur tous les éléments
        somme += u                  # Stocker la somme
        somme_carres += u**2        # et la somme des carrés
    N = len(liste)                  # Nombre d'éléments de la liste
    moyenne = somme / N             # Calcul de la moyenne: somme/N
    # Pour le calcul de la variance: On utilise le fait que
    #    <(u-<u>)^2> = <(u^2 - 2*u*<u> + <u>^2)>
    #                = <u^2> - 2*<u>*<u> + <u>^2
    #                = <u^2> - <u>^2
    variance = somme_carres / N - moyenne**2
    return moyenne,variance
```

Code *AP3*.3: Quelques tests de l'algorithme de calcul de moyenne et variance

```
# Quelques tests
import random
import math
triee         = [(v+1)/5 for v in range(100)]
gaussienne    = [random.gauss(10,2) for k in range(100)]
exponentielle = [random.expovariate(1/10) for k in range(100)]
lognormale    = [math.log(random.lognormvariate(10,2)) for k in range(100)]
uniforme      = [random.randint(0,20) for k in range(100)]

s = ''
for distrib in [triee,gaussienne,exponentielle,lognormale,uniforme]:
    m,v = moyenne_et_variance(distrib)
    s += "La liste est de moyenne {:5.2f} et de variance {:5.2f} \n".format(m,v)

print(s)
```

Code *AP3*.4: Résultat des tests précédents

```
La liste est de moyenne 10.10 et de variance 33.33
La liste est de moyenne 10.28 et de variance  4.07
La liste est de moyenne  8.69 et de variance 100.97
La liste est de moyenne 10.32 et de variance  4.29
La liste est de moyenne  9.56 et de variance 37.79
```

Code *AP3*.5: Encore du code non commenté... S'esch ebes !

```
1  def est_ici(motif,texte,i):
2      """(str,str,int) -> bool
3      Précondition: i < len(texte) - len(motif)
4
5      >>> est_ici('les','Bonjour les amis, comment ça va ?',5)
6      False
7      >>> est_ici('les','Bonjour les amis, comment ça va ?',8)
8      True
9      >>> est_ici('les','Bonjour les amis, comment ça va ?',15)
10     False
11     """
12     k = 0
13     p = len(motif)
14     while k<p and motif[k] == texte[k+i]:
15         k = k+1
16     return k == p
17
18 def recherche(motif,texte):
19     """(str,str) -> bool
20     >>> recherche('les','Bonjour les amis, comment ça va ?')
21     8
22     >>> recherche('les','Pas de bol...')
23     False
24     """
25     n = len(texte)
26     p = len(motif)
27     if p>n:
28         return False
29     else:
30         i = 0
31         while i <= n-p and not (est_ici(motif,texte,i)):
32             i = i + 1
33         if i <= n-p:
34             return i
35         else:
36             return False
```

Code *AP3*.6: Tests sur l'algorithme de recherche dans une chaîne de caractères

```
george = "George Perec"
la_disparition = """Anton Voyl n'arrivait pas à dormir. Il alluma. Son Jaz
marquait minuit vingt. Il poussa un profond soupir, s'assit dans son lit,
s'appuyant sur son polochon. Il prit un roman, il l'ouvrit, il lut; mais il
n'y saisissait qu'un imbroglio confus, il butait à tout instant sur un mot
dont il ignorait la signification. Il abandonna son roman sur son lit. Il alla
à son lavabo; il mouilla un gant qu'il passa sur son front, sur son cou. Son
pouls battait trop fort. Il avait chaud. Il ouvrit son vasistas, scruta la
nuit. Il faisait doux. Un bruit indistinct montait du faubourg. Un carillon,
plus lourd qu'un glas, plus sourd qu'un tocsin, plus profond qu'un bourdon,
non loin, sonna trois coups. Du canal Saint-Martin, un clapotis plaintif
signalait un chaland qui passait. Sur l'abattant du vasistas, un animal au
thorax indigo, à l'aiguillon safran, ni un cafard, ni un charançon, mais
plutôt un artison, s'avançait, traînant un brin d'alfa. Il s'approcha, voulant
l'aplatir d'un coup vif, mais l'animal prit son vol, disparaissant dans la
nuit avant qu'il ait pu l'assaillir. """
sacre_coco = "Ça, c'est fort mon coco !"

motifs = ['e','e','e','Georg','Friedrich','Coco']
textes = [george,la_disparition,sacre_coco]*2

s = ''
for i in range(len(motifs)):
    resultat = recherche(motifs[i],textes[i])
    if resultat or resultat is 0:
        s += "Le motif apparaît à l'indice {}\n".format(resultat)
    else:
        s += "Le motif n'apparaît pas !\n"

print(s)
```

Code *AP3*.7: Résultat des tests précédents

```
Le motif apparaît à l'indice 1
Le motif n'apparaît pas !
Le motif apparaît à l'indice 6
Le motif apparaît à l'indice 0
Le motif n'apparaît pas !
Le motif n'apparaît pas !
```

Code *AP3*.8: Recherche dans une liste triée par dichotomie

```
# Regarder au milieu de la liste
# Si l'élément milieu est plus grand que l'élément cherché
#    itérer sur la première partie de la liste (avec mémoire)
# Si l'élément milieu est plus petit que l'élément cherché
#    itérer sur la seconde partie de la liste (avec mémoire)
# Sinon, c'est qu'on a trouvé !
```

Code *AP3*.9: Recherche par dichotomie, version récursive

```
def dichotomie_rec(element,liste,memoire=0):
    """(float,list(float),int) -> int ou None
    Précondition: la liste doit être triée.

    Algorithme de recherche dichotomique d'un élément dans une liste *triée*.

    >>> dichotomie_rec(5,[1,3,4,5,17])
    3
    >>> dichotomie_rec(1,[1,3,4,5,17])
    0
    >>> dichotomie_rec(16,[1,3,4,5,17])
    """
    if len(liste) == 0: return None # On ne peut pas chercher dans une liste vide...
    # Regarder au milieu de la liste
    milieu = len(liste)//2  # Division "entière": 3//2 = 1, 9//2 = 4, etc.
    # Si l'élément milieu est plus grand que l'élément cherché
    if liste[milieu] > element:
        if milieu == 0:      # Si le "milieu" est déjà à gauche
            return None      # c'est que l'élément n'est pas dans la liste...
        # Sinon itérer sur la première partie de la liste
        return dichotomie_rec(element,liste[:milieu-1],memoire)
    # Si l'élément milieu est plus petit que l'élément cherché
    elif liste[milieu] < element:
        if milieu+1 == len(liste): # Si le "milieu" est déjà à droite
            return None            # c'est que l'élément n'est pas dans la liste...
        # Sinon itérer sur la seconde partie de la liste
        return dichotomie_rec(element,liste[milieu+1:],memoire + milieu + 1)
    # Sinon, c'est qu'on a trouvé !
    else: return milieu + memoire
```

Code *AP3*.10: Dichotomie directe sans copie de liste

```
# Initialisation des bornes
# Tant que les bornes sont dans le bon sens
#     On regarde au milieu
#     Si l'élement cherché est plus grand que l'élément milieu
#         On déplace la borne gauche (légèrement à droite du milieu)
#     Sinon, si l'élement cherché est plus petit que l'élément milieu
#         On déplace la borne droite (légèrement à gauche du milieu)
#     Sinon (cas d'égalité)
#         On renvoie l'indice du milieu
# Si on est sorti de la boucle, c'est que l'élément n'existe pas dans la liste
```

Code *AP3*.11: Implémentation directe de l'algorithme de recherche par dichotomie

```
def dichotomie_dir(element,liste):
    """(float,list(float),int) -> int ou None
    Précondition: la liste doit être triée.
    Algorithme direct de recherche dichotomique dans une liste triée sans
    souci potentiel de copie de gros tableaux.
    >>> dichotomie_dir(5,[1,3,4,5,17])
    3
    >>> dichotomie_dir(1,[1,3,4,5,17])
    0
    >>> dichotomie_dir(16,[1,3,4,5,17]) """
    g,d = 0,len(liste)-1 # Initialisation des bornes, g pour gauche, d pour droite
    while g <= d:       # Tant que les bornes sont dans le bon sens
        m = (g+d)//2    # On regarde au milieu (m)
        if element > liste[m]:  # Si l'élement cherché est plus grand
            g = m+1     # On déplace la borne gauche (légèrement à droite du milieu)
        elif element < liste[m]:# Sinon, si l'élement cherché est plus petit
            d = m-1     # On déplace la borne droite (légèrement à gauche du milieu)
        else: return m # Sinon (cas d'égalité), on renvoie l'indice du milieu
    return None # Si on est sorti de la boucle, l'élément n'existe pas dans la liste
```

Code *AP3*.12: Implémentation directe de l'algorithme de recherche par dichotomie, version non commentée

```
def dichotomie_dir_no_comment(liste):
    g,d = 0,len(liste-1)
    while g <= d:
        m = (g+d)//2
        if   element > liste[m]: g = m+1
        elif element < liste[m]: d = m-1
        else: return m
    return None
```

Code *AP3*.13: Quelques tests des deux algorithmes précédents

```
def teste_dichotomie(elements,listes,algo):
    """Test d'un algorithme de recherche en comparant à list.index().
    * elements est une liste des éléments à chercher;
    * listes est la liste des listes dans lesquelles chercher
      (de même taille que elements, puisqu'à chaque élément correspond
      une liste dans laquelle il faut le rechercher);
    * algo est l'algorithme à tester."""
    s = ''
    for i in range(len(elements)):  # Boucle sur les éléments à tester
        reponse = algo(elements[i],listes[i])
        # list.index() raises ValueError if the value is not present
        try:                            # On essaie donc
            reponse_attendue = listes[i].index(elements[i])
        except ValueError:              # Et si ValueError est soulevé
            reponse_attendue = None  # On renvoie None comme nos fonctions
        if reponse is reponse_attendue:
            s+= "On trouve bien {} à l'indice {}\n".format(elements[i],reponse)
        else:
            s+= "ATTENTION, " + str(elements[i]) + " est trouvé en " + str(reponse)
            s+= " au lieu de " + str(reponse_attendue) + "\n"
    return s

deux = [1,5]
trois= [1,5,10]
long = list(range(100))
repetitions = [1,5,5,5,5,5,5,10]
elements = [1,3,5,10,1,3,5,10,0,23,72.5,99,110]
listes   = [deux,deux,deux,deux,trois,trois,trois,trois,long,long,long,long,long]
elements+= [0,5,10]
listes  += [repetitions,repetitions,repetitions]

s = "Algorithme récursif:\n"
s+= teste_dichotomie(elements,listes,dichotomie_rec) + "\n\n"
s+= "Algorithme direct:\n"
s+= teste_dichotomie(elements,listes,dichotomie_dir)
print(s)
```

Code *AP3*.14: Résultats des tests précédents

```
Algorithme récursif:
ATTENTION, 1 est trouvé en None au lieu de 0
On trouve bien 3 à l'indice None
On trouve bien 5 à l'indice 1
On trouve bien 10 à l'indice None
ATTENTION, 1 est trouvé en None au lieu de 0
On trouve bien 3 à l'indice None
On trouve bien 5 à l'indice 1
On trouve bien 10 à l'indice 2
On trouve bien 0 à l'indice 0
ATTENTION, 23 est trouvé en None au lieu de 23
On trouve bien 72.5 à l'indice None
On trouve bien 99 à l'indice 99
On trouve bien 110 à l'indice None
On trouve bien 0 à l'indice None
ATTENTION, 5 est trouvé en 4 au lieu de 1
On trouve bien 10 à l'indice 7


Algorithme direct:
On trouve bien 1 à l'indice 0
On trouve bien 3 à l'indice None
On trouve bien 5 à l'indice 1
On trouve bien 10 à l'indice None
On trouve bien 1 à l'indice 0
On trouve bien 3 à l'indice None
On trouve bien 5 à l'indice 1
On trouve bien 10 à l'indice 2
On trouve bien 0 à l'indice 0
On trouve bien 23 à l'indice 23
On trouve bien 72.5 à l'indice None
On trouve bien 99 à l'indice 99
On trouve bien 110 à l'indice None
On trouve bien 0 à l'indice None
ATTENTION, 5 est trouvé en 3 au lieu de 1
On trouve bien 10 à l'indice 7
```

Code *AP3*.15: Correction de la dichotomie récursive

```
1  def dichotomie_rec(element,liste,memoire=0):
2      milieu = len(liste)//2
3      if liste[milieu] > element:
4          if milieu == 0:
5              return None
6          # C'est ici que l'on doit corriger: le dernier élément est exclu
7          # donc on ne doit pas appeler
8          # dichotomie_rec(element,liste[:milieu-1],memoire)
9          # mais
10         return dichotomie_rec(element,liste[:milieu],memoire)
11     elif liste[milieu] < element:
12         if milieu+1 == len(liste):
13             return None
14         return dichotomie_rec(element,liste[milieu+1:],memoire + milieu + 1)
15     else: return milieu + memoire
```

Code *AP3*.16: Résultat des tests de la dichotomie récursive corrigée

```
Dichotomie récursive corrigée:
On trouve bien 1 à l'indice 0
On trouve bien 3 à l'indice None
On trouve bien 5 à l'indice 1
On trouve bien 10 à l'indice None
On trouve bien 1 à l'indice 0
On trouve bien 3 à l'indice None
On trouve bien 5 à l'indice 1
On trouve bien 10 à l'indice 2
On trouve bien 0 à l'indice 0
On trouve bien 23 à l'indice 23
On trouve bien 72.5 à l'indice None
On trouve bien 99 à l'indice 99
On trouve bien 110 à l'indice None
On trouve bien 0 à l'indice None
ATTENTION, 5 est trouvé en 4 au lieu de 1
On trouve bien 10 à l'indice 7
```

Code *AP3*.17: Recherche du zéro d'une fonction par dichotomie

```
    # On regarde f(a), f(b) et f(milieu)
    # Tant que |f(milieu)| > epsilon choisi
    #     Si f(a) et f(milieu) sont de signes différents
    #         On bouge b vers le milieu
    #     Si f(b) et f(milieu) sont de signes différents
    #         On bouge a vers le milieu
    #     On regarde les nouveaux f(a), f(b) et f(milieu)
    # Renvoie de la valeur de milieu en fin de boucle
    # (c'est qu'il est assez proche de 0)
```

Code *AP3*.18: Implémentation de la recherche du zéro par dichotomie

```
1  def zero_par_dichotomie(f,a,b,epsilon=1e-6):
2      """Recherche du zéro d'une fonction par dichotomie. Pour être sûr de
3      marcher correctement, il ne doit y avoir qu'un seul zéro dans l'intervalle
4      considéré. Néanmoins, si on a un peu de chance, il est possible que
5      l'algorithme trouve au moins un zéro."""
6      m  = (a+b)/2.0           # On regarde f(a), f(b) et f(milieu)
7      fa,fb,fm = f(a),f(b),f(m)
8      while abs(fm) > epsilon: # Tant que |f(milieu)| > epsilon choisi
9          if fa*fm < 0:        # Si f(a) et f(milieu) sont de signes différents
10             b = m            # On bouge b vers le milieu
11         elif fb*fm < 0:      # Si f(b) et f(milieu) sont de signes différents
12             a = m            # On bouge a vers le milieu
13         else:                # Sinon, c'est qu'il y a un problème...
14             raise ValueError("""f(a) et f(b) sont de même signe,
15             nombre pair ou nul de zéro sur l'intervalle, je préfère m'arrêter""")
16         m  = (a+b)/2.0       # On regarde les nouveaux f(a), f(b) et f(milieu)
17         fa,fb,fm = f(a),f(b),f(m)
18     # Renvoie de la valeur de milieu en fin de boucle
19     # (c'est qu'il est assez proche de 0)
20     return m
```

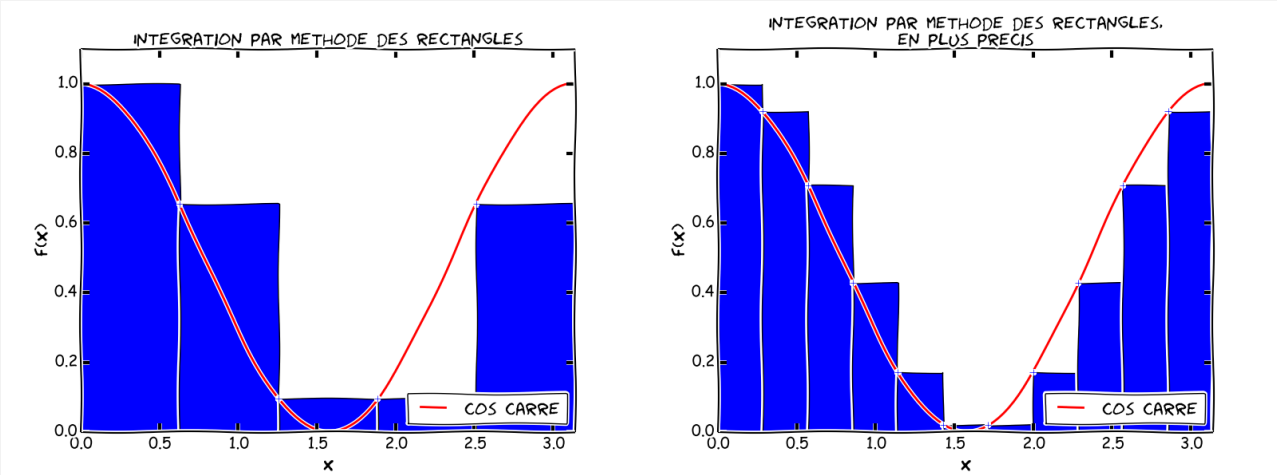
Code *AP3*.19: Des tests, des tests, oui mais des Panzani !

```
def teste_fonctions(fonctions,intervalles,epsilons):
    s= ''
    for i in range(len(fonctions)):
        f = fonctions[i]
        a,b = intervalles[i]
        epsilon = epsilons[i]
        try:
            zero = zero_par_dichotomie(f,a,b,epsilon)
            valeurs = (f.__name__,zero,a,b,epsilon)
            s+= "{} s'annule en {:.6f} sur [{},{}] à {} près.\n".format(*valeurs)
        except ValueError:
            s+= "Pas de zéro trouvé pour {} sur [{},{}].\n".format(f.__name__,a,b)
    return s

from math import *  # sin, cos et pi

def f(x): return x**2-2
def g(x): return 2-x**2
def h(x): return pi-x**2

fs = [f,f,f,g,h,sin,sin,cos,cos,cos,cos]
ints=[(0,2),(0,2),(0,2),(0,2),(0,2),(-1,1),(3,4),(-1,1),(1,2),(0,6),(0,10)]
eps= [1e-1,1e-3,1e-5,1e-6,1e-6] + [1e-6]*6
s = teste_fonctions(fs, ints, eps)
print(s)
```

Code *AP3*.20: Résultat des tests précédents

```
f s'annule en 1.437500 sur [0,2] à 0.1 près.
f s'annule en 1.414062 sur [0,2] à 0.001 près.
f s'annule en 1.414215 sur [0,2] à 1e-05 près.
g s'annule en 1.414214 sur [0,2] à 1e-06 près.
h s'annule en 1.772454 sur [0,2] à 1e-06 près.
sin s'annule en 0.000000 sur [-1,1] à 1e-06 près.
sin s'annule en 3.141592 sur [3,4] à 1e-06 près.
Pas de zéro trouvé pour cos sur [-1,1].
cos s'annule en 1.570797 sur [1,2] à 1e-06 près.
cos s'annule en 1.570796 sur [0,6] à 1e-06 près.
cos s'annule en 7.853982 sur [0,10] à 1e-06 près.
```

Code *AP3*.21: Illustration de la méthode d'intégration par rectangles



Code *AP3*.22: Intégration par méthode des rectangles

```python
def integration_rectangle_n(f,a,b,n=100):
    """Fonction permettant d'intégrer la fonction f sur l'intervalle [a,b]
    par méthode des rectangles en découpant en n sous-intervalles."""
    estimation = 0.0    # L'approximation de l'intégrale
    h = (b-a)/n         # La longueur des clôtures
    x = a               # On se place à gauche de la clôture
    for i in range(n): # On passe par toutes les clôtures
        estimation += h*f(x)  # On rajoute la contribution
        x += h                # Et on met x à jour
    return estimation

def integration_rectangle_precision(f,a,b,epsilon=1e-6):
    """Intégration de f sur [a,b] par méthode des rectangles en supposant que
    lorsque l'intégration avec deux fois plus de points donne la même chose à
    epsilon près, c'est qu'on a atteint la précision voulue (il existe bien
    entendu des contre-exemples, mais la plupart du temps, ça marche...)"""
    # On commence par estimer avec n = 1
    estimation1 = integration_rectangle_n(f,a,b,1)
    # On va comparer à n = 2
    n = 2
    estimation2 = integration_rectangle_n(f,a,b,n)
    # Tant que les deux estimations diffèrent de plus de epsilon
    while abs(estimation1-estimation2) > epsilon:
        estimation1 = estimation2 # La "nouvelle" estimation devient l'ancienne
        n *= 2                    # On multiplie le nombre de clôture par 2
                                  # Et on fait une "vraie" nouvelle estimation
        estimation2 = integration_rectangle_n(f,a,b,n)
    # Tant qu'à faire, si on a fait le calcul, autant renvoyer l'intégrale de
    # plus grand nombre de points intermédiaires...
    return estimation2
```

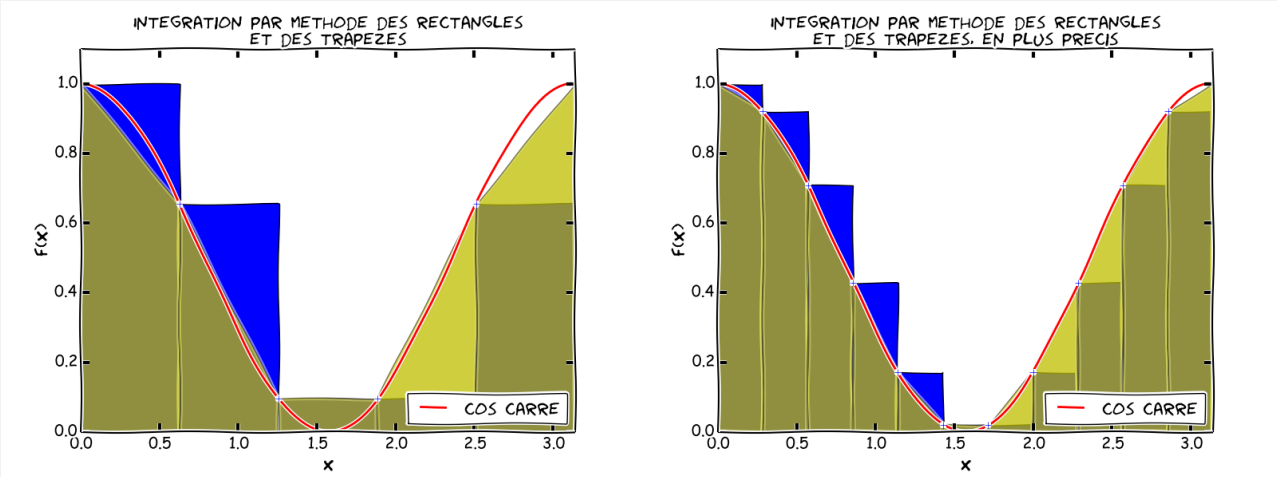
Code *AP3*.23: Procédure de tests pour la méthode des rectangles

```
from math import * # La base

# Les fonctions courantes
def lineaire(x): return x
def parabole(x): return x**2
def cubique(x):  return x**3

# deux fonction "faite pour" faire dérailler la méthode en epsilon
def bizarre(x): # Version linéaire
    y = x%4
    if   y < 1: return  4*y
    elif y < 2: return -4*(y-2)
    elif y < 3: return -1*(y-2)
    elif y < 4: return  1*(y-4)

def bizarre2(x): # Version quadratique
    y = x%4
    if   y < 2: return  4 - 4*(y-1)**2
    elif y < 4: return -1 + 1*(y-3)**2

# Deux alias (dans l'idée de réutiliser les tests pour l'intégration trapèze):
def int_n(f,a,b,n):       return integration_rectangle_n(f,a,b,n)
def int_p(f,a,b,epsilon): return integration_rectangle_precision(f,a,b,epsilon)

def teste_integration(fonctions,intervalles):
    """ Tests génériques d'intégration par nombre d'intervalle ou par
    précision. Requière les fonctions int_n(f,a,b,n) et int_p(f,a,b,epsilon)
    d'être définies pour marcher."""
    s  = ''
    N  = [10,100,1000]
    eps= [1e-2,1e-4,1e-6]
    for i in range(len(fonctions)):
        f = fonctions[i]
        a,b = intervalles[i]
        s+= 'Pour {} sur [{},{}]:\n'.format(f.__name__,round(a,4),round(b,4))
        for n in N:
            I = int_n(f,a,b,n)
            s += "n={:5d}:\t I = {:+f} | ".format(n,I)
        s += "\n"
        for e in eps:
            I = int_p(f,a,b,e)
            s += "e={:1.0e}:\t I = {:+f} | ".format(e,I)
        s += "\n"
    return s

fonctions  = [lineaire,parabole,cubique,cos,sin,cos,sin,bizarre,bizarre,bizarre2,bizarre
intervalles= [(0,1)]*3 + [(0,2*pi)]*2 + [(0,pi)]*2 + [(0,8),(0,8.0002)]*2
print(teste_integration(fonctions,intervalles))
```

Code *AP3*.24: Résultats des tests pour la méthode des rectangles

```
Pour lineaire sur [0.0,1.0]:
n=   10:  I = +0.450000 | n=  100:  I = +0.495000 | n= 1000:  I = +0.499500 |
e=1e-02:  I = +0.492188 | e=1e-04:  I = +0.499939 | e=1e-06:  I = +0.499999 |
Pour parabole sur [0.0,1.0]:
n=   10:  I = +0.285000 | n=  100:  I = +0.328350 | n= 1000:  I = +0.332834 |
e=1e-02:  I = +0.325562 | e=1e-04:  I = +0.333272 | e=1e-06:  I = +0.333332 |
Pour cubique sur [0.0,1.0]:
n=   10:  I = +0.202500 | n=  100:  I = +0.245025 | n= 1000:  I = +0.249500 |
e=1e-02:  I = +0.242249 | e=1e-04:  I = +0.249939 | e=1e-06:  I = +0.249999 |
Pour cos sur [0.0,6.2832]:
n=   10:  I = +0.000000 | n=  100:  I = -0.000000 | n= 1000:  I = -0.000000 |
e=1e-02:  I = -0.000000 | e=1e-04:  I = -0.000000 | e=1e-06:  I = -0.000000 |
Pour sin sur [0.0,6.2832]:
n=   10:  I = +0.000000 | n=  100:  I = -0.000000 | n= 1000:  I = -0.000000 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour cos sur [0.0,3.1416]:
n=   10:  I = +0.314159 | n=  100:  I = +0.031416 | n= 1000:  I = +0.003142 |
e=1e-02:  I = +0.006136 | e=1e-04:  I = +0.000096 | e=1e-06:  I = +0.000001 |
Pour sin sur [0.0,3.1416]:
n=   10:  I = +1.983524 | n=  100:  I = +1.999836 | n= 1000:  I = +1.999998 |
e=1e-02:  I = +1.998393 | e=1e-04:  I = +1.999975 | e=1e-06:  I = +2.000000 |
Pour bizarre sur [0.0,8.0]:
n=   10:  I = +5.760000 | n=  100:  I = +5.990400 | n= 1000:  I = +6.000000 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour bizarre sur [0.0,8.0002]:
n=   10:  I = +5.760368 | n=  100:  I = +5.990428 | n= 1000:  I = +6.000000 |
e=1e-02:  I = +0.001600 | e=1e-04:  I = +6.000000 | e=1e-06:  I = +6.000000 |
Pour bizarre2 sur [0.0,8.0]:
n=   10:  I = +7.680000 | n=  100:  I = +7.987200 | n= 1000:  I = +7.999872 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour bizarre2 sur [0.0,8.0002]:
n=   10:  I = +7.680192 | n=  100:  I = +7.987255 | n= 1000:  I = +7.999878 |
e=1e-02:  I = +0.003200 | e=1e-04:  I = +7.999972 | e=1e-06:  I = +8.000000 |
```

Code *AP3*.25: Illustration de la méthode d'intégration par trapèzes



Code *AP3*.26: Intégration par méthode des trapèzes

```python
def integration_trapeze_n(f,a,b,n=100):
    """Fonction permettant d'intégrer la fonction f sur l'intervalle [a,b]
    par méthode des trapèzes en découpant en n sous-intervalles."""
    estimation = 0.0    # L'approximation de l'intégrale
    h = (b-a)/n         # La longueur des clôtures
    x = a               # On se place à gauche de la clôture
    for i in range(n): # On passe par toutes les clôtures
        # On rajoute l'aire du trapèze correspondant
        estimation += h*(f(x)+f(x+h))/2
        x += h                  # Et on met x à jour
    return estimation

def integration_trapeze_precision(f,a,b,epsilon=1e-6):
    """Intégration de f sur [a,b] par méthode des trapèzes en supposant que
    lorsque l'intégration avec deux fois plus de points donne la même chose à
    epsilon près, c'est qu'on a atteint la précision voulue (il existe bien
    entendu des contre-exemples, mais la plupart du temps, ça marche...)"""
    # On commence par estimer avec n = 1
    estimation1 = integration_trapeze_n(f,a,b,1)
    # On va comparer à n = 2
    n = 2
    estimation2 = integration_trapeze_n(f,a,b,n)
    # Tant que les deux estimations diffèrent de plus de epsilon
    while abs(estimation1-estimation2) > epsilon:
        estimation1 = estimation2 # La "nouvelle" estimation devient l'ancienne
        n *= 2                    # On multiplie le nombre de clôture par 2
                                  # Et on fait une "vraie" nouvelle estimation
        estimation2 = integration_trapeze_n(f,a,b,n)
    # Tant qu'à faire, autant renvoyer l'intégrale la plus précise
    return estimation2
```

Code *AP3*.27: Résultats des tests pour la méthode des trapèzes

```
Pour lineaire sur [0.0,1.0]:
n=   10:  I = +0.500000 | n=  100:  I = +0.500000 | n= 1000:  I = +0.500000 |
e=1e-02:  I = +0.500000 | e=1e-04:  I = +0.500000 | e=1e-06:  I = +0.500000 |
Pour parabole sur [0.0,1.0]:
n=   10:  I = +0.335000 | n=  100:  I = +0.333350 | n= 1000:  I = +0.333334 |
e=1e-02:  I = +0.335938 | e=1e-04:  I = +0.333344 | e=1e-06:  I = +0.333333 |
Pour cubique sur [0.0,1.0]:
n=   10:  I = +0.252500 | n=  100:  I = +0.250025 | n= 1000:  I = +0.250000 |
e=1e-02:  I = +0.250977 | e=1e-04:  I = +0.250015 | e=1e-06:  I = +0.250000 |
Pour cos sur [0.0,6.2832]:
n=   10:  I = -0.000000 | n=  100:  I = -0.000000 | n= 1000:  I = -0.000000 |
e=1e-02:  I = -0.000000 | e=1e-04:  I = -0.000000 | e=1e-06:  I = -0.000000 |
Pour sin sur [0.0,6.2832]:
n=   10:  I = -0.000000 | n=  100:  I = -0.000000 | n= 1000:  I = -0.000000 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour cos sur [0.0,3.1416]:
n=   10:  I = +0.000000 | n=  100:  I = -0.000000 | n= 1000:  I = -0.000000 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour sin sur [0.0,3.1416]:
n=   10:  I = +1.983524 | n=  100:  I = +1.999836 | n= 1000:  I = +1.999998 |
e=1e-02:  I = +1.998393 | e=1e-04:  I = +1.999975 | e=1e-06:  I = +2.000000 |
Pour bizarre sur [0.0,8.0]:
n=   10:  I = +5.760000 | n=  100:  I = +5.990400 | n= 1000:  I = +6.000000 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour bizarre sur [0.0,8.002]:
n=   10:  I = +5.766881 | n=  100:  I = +5.990995 | n= 1000:  I = +6.000032 |
e=1e-02:  I = +0.032008 | e=1e-04:  I = +0.032008 | e=1e-06:  I = +0.032008 |
Pour bizarre sur [0.0,8.2]:
n=   10:  I = +6.461600 | n=  100:  I = +6.086368 | n= 1000:  I = +6.080031 |
e=1e-02:  I = +3.280000 | e=1e-04:  I = +3.280000 | e=1e-06:  I = +3.280000 |
Pour bizarre sur [0.0,7.0]:
n=   10:  I = +6.300000 | n=  100:  I = +6.496000 | n= 1000:  I = +6.499990 |
e=1e-02:  I = -3.500000 | e=1e-04:  I = -3.500000 | e=1e-06:  I = -3.500000 |
Pour bizarre2 sur [0.0,8.0]:
n=   10:  I = +7.680000 | n=  100:  I = +7.987200 | n= 1000:  I = +7.999872 |
e=1e-02:  I = +0.000000 | e=1e-04:  I = +0.000000 | e=1e-06:  I = +0.000000 |
Pour bizarre2 sur [0.0,8.002]:
n=   10:  I = +7.688306 | n=  100:  I = +7.988384 | n= 1000:  I = +7.999985 |
e=1e-02:  I = +0.063968 | e=1e-04:  I = +0.063968 | e=1e-06:  I = +8.000016 |
Pour bizarre2 sur [0.0,8.2]:
n=   10:  I = +8.367280 | n=  100:  I = +8.151194 | n= 1000:  I = +8.149329 |
e=1e-02:  I = +8.149823 | e=1e-04:  I = +8.149381 | e=1e-06:  I = +8.149333 |
Pour bizarre2 sur [0.0,7.0]:
n=   10:  I = +8.435000 | n=  100:  I = +8.664390 | n= 1000:  I = +8.666644 |
e=1e-02:  I = +8.665285 | e=1e-04:  I = +8.666645 | e=1e-06:  I = +8.666666 |
```