TP 14- Proposition de solutions

Solution 1 Approximation de tangente

tangente.py

```python
def tan(x):
    x=x%pi
    if x>pi/2:
        x=x-pi
    assert pi/2-abs(x)>10**(-10),'erreur Df'
    if abs(x)<10**(-6):
        return(x)
    else:
        t=tan(x/2)
        return 2*t/(1-t*t)
```

Afin d'éviter un double appel récursif, on pré-calcule la tangente de l'arc moitié avant d'appliquer la formule.

Solution 2 Puissance de 2 d'un entier

1. Un invariant est $a = 2^i b$.
2. Version récursive de puiss2 :

puiss.py

```python
def puiss2_R(a):
    if a%2==1:
        return 0,a
    else:
        i,b=puiss2_R(a//2)
        return i+1,b
```

3. Tableau des appels et des résultats :

Appels	Résultats
puiss2_R(224)	(5,7)
puiss2_R(112)	(4,7)
puiss2_R(56)	(3,7)
puiss2_R(28)	(2,7)
puiss2_R(14)	(1,7)
puiss2_R(7)	(0,7)

On vérifie par :

```
>>> puiss2_R(112)
(5, 7)
```

Solution 3 La fonction ens génère la liste des éléments de L sans répétition.

Solution 4 ⇛ $u_0 = 0$, $u_1 = -1$, $u_2 = 0$ et $\forall n \geq 0$

$$u_{n+3} = \sqrt{|u_{n+1}u_n|} - n + u_{n+2}$$

On peut réécrire pour $i \geq 3$:

$$u_i = \sqrt{|u_{i-2}u_{i-3}|} - i + 3 + u_{i-1}$$

Itératif

```python
def u1(n):
    u,v,w=0,-1,0
    if n==0:
        return u
    elif n==1:
        return v
    elif n==2:
        return w
    else:
        for i in range(3,n+1):
            u,v,w=v,w,np.sqrt(abs(u*v))-i+3+w
        return w
```

Récursif

```python
def r1(n):
 if n==0:
  return 0
 elif n==1:
  return -1
 elif n==2:
  return 0
 else:
  return np.sqrt(abs(r1(n-2)*r1(n-3)))-n+3+r1(n-1)
```

⇛ $u_0 = 2$, $v_0 = 10$ et $\forall n \geq 1$

$$u_n = \frac{u_{n-1} + v_{n-1}}{2} \qquad v_n = \sqrt{u_{n-1}v_{n-1}}$$

Itératif

```python
def u2(n):
    u,v=2,10
    if n==0:
        return u,v
    else:
        for i in range(1,n+1):
            u,v=(u+v)/2,np.sqrt(u*v)
        return u,v
```

Récursif

```python
def ru2(n):
    if n==0:
        return 2
    else:
        return (ru2(n-1)+rv2(n-1))/2

def rv2(n):
    if n==0:
        return 10
    else:
        return np.sqrt(ru2(n-1)*rv2(n-1))
```

Solution 5 Etude de la récursivité dans le cas d'une suite récurrente d'ordre 2 :

Version itérative

```python
def ite_u(n):
    assert type(n)==int and n>-1,'rand invalide'
    u,v=1,-6
    if n==0:
        return u
    elif n==1:
        return v
    else:
        for i in range(2,n+1):
            u,v=v,v-u
        return v
```

Version récursive

```python
def rec1_u(n):
    assert type(n)==int and n>-1,'rand invalide'
    if n==0:
        return 1
    elif n==1:
        return -6
    else:
        return rec1_u(n-1)-rec1_u(n-2)
```

Version hybride

```python
def rec2_u(n):
    assert type(n)==int and n>-1,'rand invalide'
    if n in T[0]:
        i=T[0].index(n)
        return T[1][i]
    else:
        u=rec2_u(n-1)-rec2_u(n-2)
        T[0].append(n)
        T[1].append(u)
        return u

# initialisation de T :
T=[[0,1],[1,-6]]
```

Voici quelques comparaisons d'efficacité temporelle de ces fonctions :

n	ite_u	rec1_u	rec2_2
15	1.5 e-05	2.3 e-03	1.6 e-04
20	1.9 e-05	3.2 e-02	2.3 e-04
25	2.5 e-05	2.9 e-01	5.6 e-04
30	2.7 e-05	3.5	3.3 e-04
35	2.5 e-05	40.4	3.8 e-04
80	2.4 e-05		1.2 e-03
900	2.0 e-04		9.9 e-02

Solution 6 Fractale de carrés
La solution suivante est utilise une fonction récursive.

Nous modélisons un carré par

- le point milieu de sa base : (a,b),
- un vecteur unitaire et normal à sa base et dirigé vers le *haut* : n
- la longueur d'un côté : r

Initialisation

```python
a=0  # abscisse du milieu de la base
b=0   # son ordonnee
n=[0,1]  # vecteur normal a la base
r=1    # longueur de la base

k=0.6  # rapport homothetique
e=float(input("Longueur de finition : "))
```

➜ Recherche des coordonnées des quatre sommets du carré. Effectuons une rotation d'angle $\frac{\pi}{2}$ de $n=\begin{pmatrix}n_0\\n_1\end{pmatrix}$ dans un repère O.N.D. :

$$Rot_{\frac{\pi}{2}}(n)=\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}_{(\theta=\frac{\pi}{2})} n=\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}\begin{pmatrix}n_0\\n_1\end{pmatrix}=\begin{pmatrix}-n_1\\n_0\end{pmatrix}$$

$$A:\quad \begin{pmatrix}a\\b\end{pmatrix}-\frac{r}{2}\begin{pmatrix}-n_1\\n_0\end{pmatrix} \qquad B:\quad \begin{pmatrix}a\\b\end{pmatrix}-\frac{r}{2}\begin{pmatrix}-n_1\\n_0\end{pmatrix}+\begin{pmatrix}n_0\\n_1\end{pmatrix}$$

$$C:\quad \begin{pmatrix}a\\b\end{pmatrix}+\frac{r}{2}\begin{pmatrix}-n_1\\n_0\end{pmatrix}+\begin{pmatrix}n_0\\n_1\end{pmatrix} \qquad D:\quad \begin{pmatrix}a\\b\end{pmatrix}+\frac{r}{2}\begin{pmatrix}-n_1\\n_0\end{pmatrix}$$

➜ Caractéristiques des deux nouveaux carrés à considérer :

- $\left[B,\frac{1}{\sqrt{2}}\left(n+Rot_{\frac{\pi}{2}}(n)\right),r\times k\right]$
- $\left[C,\frac{1}{\sqrt{2}}\left(n-Rot_{\frac{\pi}{2}}(n)\right),r\times k\right]$

fractale_rec.py

```
def carre(a,b,n,r):
 if r>e:
  # sommets du carre
  xx=[a+n[1]*r/2,a+n[1]*r/2+r*n[0],
        a-n[1]*r/2+r*n[0],a-n[1]*r/2,a+n[1]*r/2]
  yy=[b-n[0]*r/2,b-n[0]*r/2+r*n[1],
        b+n[0]*r/2+r*n[1],b+n[0]*r/2,b-n[0]*r/2]
  plt.plot(xx,yy,color='k')

  # deux nouveaux carres
  aa=[a+n[1]*r/2+r*n[0],a-n[1]*r/2+r*n[0]]
  bb=[b-n[0]*r/2+r*n[1],b+n[0]*r/2+r*n[1]]
  n1=[(n[0]+n[1])/sqrt(2),(n[1]-n[0])/sqrt(2)]
  n2=[(n[0]-n[1])/sqrt(2),(n[1]+n[0])/sqrt(2)]
  carre(aa[0],bb[0],n1,r*k)
  carre(aa[1],bb[1],n2,r*k)
```

Affichage

```
import matplotlib.pyplot as plt
plt.close()
plt.figure('Fractale de carres')

carre(x,y,n,r)

plt.axis('scaled')
plt.xlim(-2.2,2.2)
plt.ylim(-0.2,2.6)
plt.axis('off')
```