Corrigé du TD du 26 janvier 2006

On commence par tracer les courbes: `plot([tan(x),1/x],x=0..7,y=-5..5);`
On prend les domaines suivants pour les trois premières racines positives: $[0.5, 1]$, $[3, 4]$ et $[6, 7]$.

1 Dichotomie

1)

```
dichotomie:=proc(a0,b0,e,f)
   local a,b,c;
      a:=a0; b:=b0; c:=(a+b)/2;
      while evalf((b-a)/2)>evalf(e) do
         if evalf(f(a)*f(c))<0 then b:=c;
            else a:=c;
         end if;
         c:=(a+b)/2;
      end do;
      evalf(c);
end proc;
```

Selon qu'on regarde `e` comme une précision ou une tolérance sur l'erreur, on va prendre des critères d'arrêt différents. Ici, on prend `while evalf((b-a)/2)>evalf(e)` parcequ'on veut sortir de la boucle lorsqu'on a une précision suffisante sur la racine. On prendrait `while evalf(abs(f(c)))>evalf(e)` si on voulait juste que l'approximation respecte une certaine tolérance sur l'erreur commise.

2)

`dichotomie(0.5,1,10^(-5),x->tan(x)-1/x);` $\rightarrow 0.86033$
`dichotomie(3,4,10^(-5),x->tan(x)-1/x);` $\rightarrow 3.42562$
`dichotomie(6,7,10^(-5),x->tan(x)-1/x);` $\rightarrow 6.43729$

3) On veut tracer des graphes qui montrent si les approximations de la racine x_0 de l'équation $f(x) = 0$ sont bonnes. Ici, notre algorithme `dichotomie` construit une suite de valeurs $(c_n)_n$ (les centres des segments $[a, b]$) qui va converger vers x_0. Les programmes `grapherreur` et `grapherreurlog` vont simplement stocker dans une liste les couples $(n, f(c_n))$ ou $(n, \log_{10}(|f(c_n)|))$, puis dessiner la ligne brisée correspondante.

```
grapherreur:=proc(a0,b0,e,f)
   local a,b,c,L,i;
      a:=a0; b:=b0; c:=(a+b)/2; L:=[[1,evalf(f(c))]]; i:=1;
```

```
      while evalf((b-a)/2)>evalf(e) do
         if evalf(f(a)*f(c))<0 then b:=c;
            else a:=c;
         end if;
         c:=(a+b)/2;
         i:=i+1;
         L:=[op(L),[i,evalf(f(c))]];
      end do;
      plot(L);
end proc:

grapherreurlog:=proc(a0,b0,e,f)
   local a,b,c,L,i;
      a:=a0; b:=b0; c:=(a+b)/2; L:=[[1,evalf(log10(abs(f(c))))]]; i:=1;
      while evalf((b-a)/2)>evalf(e) do
         if evalf(f(a)*f(c))<0 then b:=c;
            else a:=c;
         end if;
         c:=(a+b)/2;
         i:=i+1;
         L:=[op(L),[i,evalf(log10(abs(f(c))))]];
      end do;
      plot(L);
end proc;
```

On obtient les résultats suivants (pour les commandes `grapherreur(0.5,1,10^(-5),x->tan(x)-1/x);` et `grapherreurlog(0.5,1,10^(-5),x->tan(x)-1/x);`):

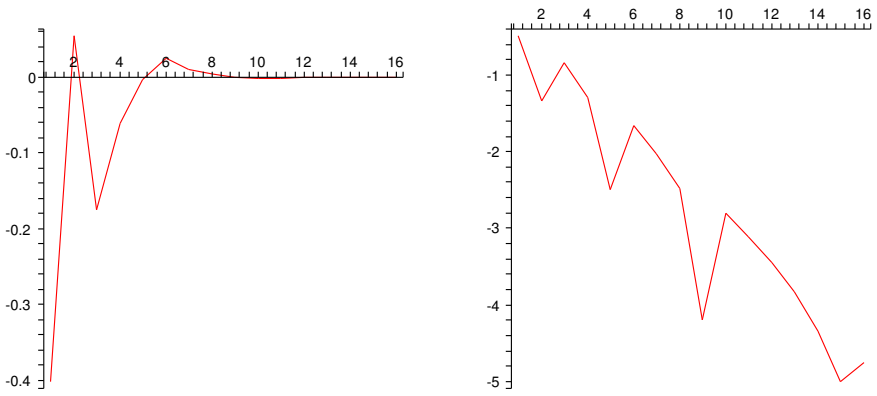

Figure 1: On dessine la progression de l'erreur, itération après itération et son logarithme.

Que voit-on sur ces graphiques? Sur le premier, on voit clairement que $f(c_n) \to 0$ donc notre algorithme fonctionne. Cependant, il n'est pas très lisible au voisinage de 0. Plutôt que de faire plusieurs zooms pour voir comment la courbe se rapproche de 0, on trace le log (de la valeur absolue) de l'erreur et on lit en

ordonnée la précision atteinte (en abscisse, on a le nombre d'itérations). Ici, la courbe est à peu près linéaire, on gagne une décimale dans l'erreur toutes les trois itérations environ (au bout de 16 itérations, on a une précision de 10^{-5}).

2 Point fixe

Ici, on va résoudre numériquement une équation du type $g(x) = x$ et essayer d'appliquer ce savoir-faire à la résolution d'une équation $f(x) = 0$.

1) Supposons qu'on a une fonction g qui vérifie les hypothèses du théorème du point fixe. Alors quel que soit le nombre x_0 dans I, la suite $(x_n)_{n\geq 0}$ définie par $x_{n+1} = g(x_n)$ converge vers la solution de l'équation $g(x) = x$. Ainsi, pour trouver numériquement une approximation de la solution, il faut prendre une condition initiale (quasi-) quelconque et itérer g un grand nombre de fois. C'est ce que fait le programme suivant:

```
pointfixe:=proc(x0,e,g)
   local x;
      x:=x0;
      while evalf(abs(g(x)-x))>e do
         x:=g(x);
      end do;
      evalf(x);
end proc;
```

Ici, on arrête d'itérer g lorsque $|g(x_n) - x_n|$ est suffisamment petit (c'est notre critère d'arrêt).

2) On résoud le problème $\cos(x) = x$ avec cette méthode (c'est-à-dire $g = \cos$). On prend $I = [0, 1]$. On a bien $\cos(I) \subset I$ et $|\cos'(x)| < 1$ sur I donc la fonction

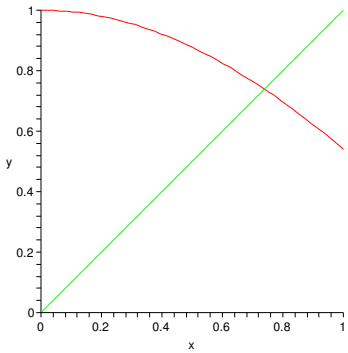

Figure 2: Graphes des fonctions cos et identité.

cos vérifie les hypothèses du théorème du point fixe sur $[0, 1]$.
On applique notre algorithme sur n'importe quel point de I, 0.5 par exemple, et on trouve (pour $\epsilon = 10^{-8}$), la solution approchée suivante: 0.73908514.

Remarque: Ici, ϵ est seulement un critère d'arrêt, pas une précision sur la racine: a priori, on ne peut pas prétendre que $|g(x_n) - x_n| \leq \epsilon \Rightarrow |l - x_n| \leq \epsilon$ (l est le point fixe de g). Pourtant ça fonctionne. $|g(x_n) - x_n| \simeq |x_n - l||g'(l) - 1|$ [1], autrement dit, l'erreur commise sur la racine est proportionnelle au critère d'arrêt.

3) Le premier réflexe, quand on veut transformer un problème $f(x) = 0$ en un problème $g(x) = x$, est de poser $g(x) := x - f(x)$. Mais attention! La fonction g risque de ne pas être contractante! C'est le cas pour $f(x) = \tan(x) - \frac{1}{x}$. Si on pose $g(x) = x - \tan(x) + \frac{1}{x}$, on a $g'(x) = -\tan^2(x) - \frac{1}{x^2}$, et $|g'(0.86)| \simeq 2.7$ donc g n'est pas contractante au voisinage du point fixe.

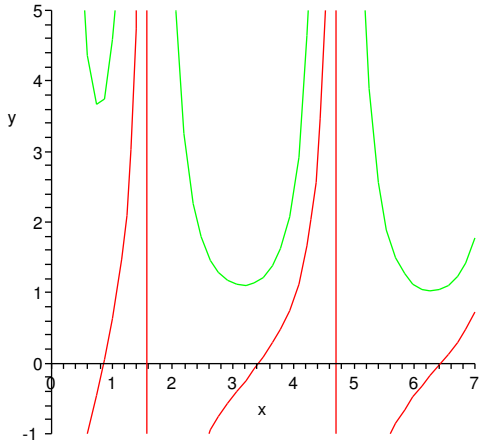

Figure 3: Graphes de $\tan(x) - \frac{1}{x}$ et de sa dérivée. On voit qu' au voisinage de la première racine, la dérivée vaut environ 4, donc on va prendre $\lambda = 4$.

Il faut faire plus attention dans la définition de g. On appelle α la racine de $f(x) = 0$ et on pose $g(x) = x - \frac{f(x)}{\lambda}$ avec $\lambda \simeq f'(\alpha)$. Ainsi, résoudre $f(x) = 0$ est toujours équivalent à résoudre $g(x) = x$ et en plus, g est contractante au voisinage de α [2], ce qui garantit que le principe du point fixe va fonctionner. Pour $f(x) = \tan(x) - \frac{1}{x}$, on a $f(\alpha) \simeq 4$ donc on pose $\lambda = 4$ et

$$g(x) = x - \frac{\tan(x)}{4} + \frac{1}{4x}$$

On tape la commande `pointfixe(0.75,10^(-5),x->x-(tan(x)-1/x)/4);` et on obtient 0.86033 (si on avait tapé `pointfixe(0.75,10^(-5),x->x-(tan(x)-1/x));`, on aurait obtenu -6.437294610...).

[1] $|g(x_n) - x_n| = |g(l + (x_n - l)) - (l + (x_n - l))| \simeq |g(l) + (x_n - l)g'(l) - l - (x_n - l)| = |x_n - l||g'(l) - 1|$.

[2] En effet, $g'(\alpha) = 1 - \frac{f'(\alpha)}{\lambda} \simeq 0$.

3 Newton

```
newton:=proc(x0,e,f)
   local x,X,fprime;
      x:=x0;
      fprime:=D(f);
      X:=x-f(x)/fprime(x);
      while abs(X-x)>e do
         x:=X;
         X:=X-f(X)/fprime(X);
      end do;
end proc;
```

Le critère d'arrêt est $|x_{n+1} - x_n| \leq \epsilon$, soit $\left|\frac{f(x_n)}{f'(x_n)}\right| \leq \epsilon$, ce qui revient à peu près à $|f(x_n)| \leq \epsilon |f'(\alpha)|$. L'algorithme s'arrête donc lorsque l'erreur par rapport à 0 est suffisamment petite.

```
erreurnewton:=proc(x0,e,f)
   local x,X,fprime,i,L;
      x:=x0;
      fprime:=D(f);
      X:=x-f(x)/fprime(x);
      i:=2;
      L:=[[1,x],[2,X]];
      while abs(X-x)>e do
         i:=i+1;
         x:=X;
         X:=X-f(X)/fprime(X);
         L:=[op(L),[i,log10(abs(f(X)))]];
      end do;
      plot(L);
end proc;
```

On trace le graphe de l'erreur en prenant soin d'augmenter au pralable le nombre de chiffres significatifs:
`Digits:=100; erreurnewton(1.5,10^(-13),x->tan(x)-1/x);`
Au lieu d'une sorte de droite, on a une sorte d'arc de parabole qui témoigne de la convergence quadratique de la méthode. A chaque itération, le nombre de chiffres significatifs double. La méthode est très efficace.

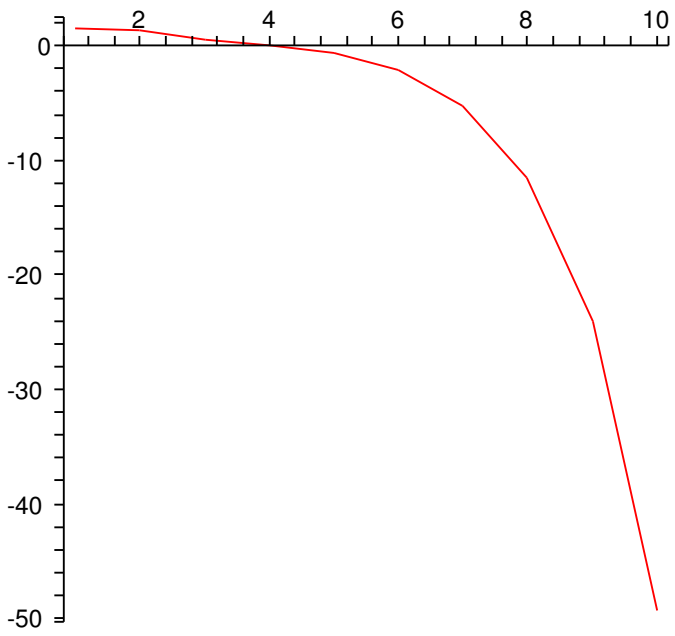

Figure 4: Tracé du log de l'erreur avec la méthode de Newton. Ici, la convergence est quadratique: la précision double à chaque itération.