

Programmation et algorithmique TPL N°5

Cycle de transition **esiea** 2007-2008

L. Beaudoin

R. Erra

D. Julien

V. Guyot

La chasse au dahu !

Dans ce TPL, on se propose de travailler sur les fonctions tout en commençant à faire des simulations numériques. Et pour que tout cela soit un peu visuel, on fera une initiation à un logiciel (libre bien sûr) de dessin scientifique appelé `gnuplot` pour visualiser les simulations produites. Bonne chasse au dahu !

1 Le dahu

Comme vous le savez sûrement, le dahu est un animal mythique perpétuellement chassé. Mais pour qu'il complique la vie des chasseurs, nous allons le doter de la faculté de se déplacer. Par contre, comme le dahu n'est pas intelligent, ses déplacements se font de manière totalement aléatoires.

Écrire un programme qui :

1. fixe une position de départ au dahu D de coordonnées réelles (XD, YD) ,
2. calcule la fonction `Bouge` qui :
 - ne retourne rien,
 - prend comme paramètres d'entrée les réels XD et YD position du dahu, et le réel `PasMax` qui est le déplacement maximal possible du dahu en une fois en x ou en y ,
 - calcule le déplacement aléatoire DX compris entre $-PasMax$ et $PasMax$,
 - idem pour DY ,
 - modifie la position (XD, YD) en ajoutant (DX, DY) et on veut que ces résultats soient utilisables pour les autres fonctions du programme. À vous donc trouver le bon prototype de `Bouge` qui permette cela,
3. fasse 10 fois l'opération 2 et affiche chaque position avec une précision de 2 chiffres après la virgule.

2 Le chasseur

Maintenant que nous avons un dahu mobile, créons un piège (immobile) à dahu. Pour que le piège soit efficace, il faut que lorsque le dahu passe dans son rayon d'action, il l'attrape !

En partant de la base du programme précédent, écrire un programme qui :

1. fixe comme position de départ au dahu $(0,0)$ et comme positions pour les pièges 1, 2, 3 et 4 les positions respectives $(-5,5)$, $(5,5)$, $(5,-5)$ et $(-5,-5)$, et fixe comme rayon d'action aux pièges 5 et 1 pour le `PasMax`. Pour que l'on puisse facilement utiliser les coordonnées précédentes, créez les tableaux de flottants XP et YP qui contiennent les abscisses et les ordonnées respectivement des pièges,
2. calcule la fonction `Attrape` qui :
 - retourne 1 si le dahu est attrapé et 0 sinon,
 - prend comme paramètres d'entrée les tableaux XP et YP , `RayonAction`, XD , YD et l'entier `NumPiege` qui sera le numéro du piège qui a attrapé le dahu,
 - calcule pour chaque piège la distance du dahu au piège. Si cette distance est inférieure à `RayonAction`, le dahu est attrapé et on actualise la valeur de `NumPiege` de telle sorte que ce résultat soit visible par le reste du programme.
3. le dahu se déplace en utilisant la fonction `Bouge` si aucun piège n'a attrapé le dahu,
4. fasse les opérations 2 et 3 tant que le dahu n'est pas attrapé. Dès que le dahu est attrapé, alors s'affiche à l'écran :
`Le piège X a attrapé le dahu!!!`
où X est le numéro du piège.

3 Visualiser les résultats avec gnuplot

Nous avons une série de coordonnées (x,y) qui apparaissent à l'écran. C'est frustrant de ne pas voir se dessiner la trajectoire. Corrigions cela en utilisant un logiciel libre pour visualiser nos données. Plus tard dans l'année, on verra comment on créera nous même notre propre visualiseur intégré au programme.

3.1 Premiers pas en gnuplot

Le logiciel que nous allons utiliser s'appelle **gnuplot**. Écrivez le programme `trace.gp` (`.gp` pour dire que c'est un programme écrit en **gnuplot**) suivant :

```
# intervalle en x,y et grille de représentation
set xrange [-5:5]
set yrange [-5:5]
set grid
set size square
set title "Mon premier graphe"
set xlabel "axe des x"
set ylabel "axe des y"
# constante
R=5;
# fonction mathématique d'un cercle
CercleSup(x,xc,yc)=sqrt(R**2-(x-xc)**2)+yc
CercleInf(x,xc,yc)=-sqrt(R**2-(x-xc)**2)+yc
# tracé d'un cercle en (1,-1) de rayon R
plot CercleSup(x,1,-1) title "sup"
replot CercleInf(x,1,-1) title "inf"
pause -1
```

Quelques commentaires :

- toutes les lignes qui commencent par un `#` sont considérées comme des commentaires et sont donc ignorées par **gnuplot**,
- `set` permet d'initialiser des paramètres de **gnuplot**. Ainsi, `set xrange[-5:5]` signifie que l'intervalle de dessin est compris entre les abscisses -5 et 5. Pour les ordonnées, c'est `yrange` qu'il faut utiliser,
- `set grid` permet d'avoir la grille des abscisses/ordonnées ce qui est plus pratique pour se repérer,
- `set size square` permet d'avoir la même échelle en x et y,
- `set title` qui permet de donner un titre à la figure,
- `set xlabel`, pour mettre du texte sous l'axe des abscisses (`set ylabel` pour les ordonnées),
- `sqrt` est la fonction racine carrée et `**` est l'opérateur de puissance,
- `plot` permet de dessiner une fonction et `replot` permet de dessiner une deuxième (ou plus) fonction sur le même graphique que la première (i.e. sans effacer la première) et `title` sur la même ligne permet de fixer la légende,
- `pause -1` permet d'attendre que vous appuyer sur une touche avant de fermer la fenêtre où est affichée la courbe.

Maintenant que vous savez lire un programme **gnuplot**, il ne reste plus qu'à l'exécuter! Pour cela, allez en mode graphique (`ctr-alt-f7`), ouvrez un terminal (qui est une mini-console), descendez sur le répertoire où vous avez écrit le programme `trace.gp` et exécutez la commande

```
gnuplot trace.gp
```

Vous devriez obtenir la figure 1.

Exercice : modifier `trace.gp` pour visualiser les rayons d'action des 4 pièges de l'exercice 2.

3.2 Récupérer et afficher des données d'un programme

Maintenant que nous savons afficher des courbes en **gnuplot**, voyons comment récupérer et afficher les données de notre programme C. Lorsque l'on fait un `printf` en C, l'affichage se fait par défaut dans la console. Il est possible de dire à l'ordinateur de diriger le flux de données non pas vers la console mais vers un fichier. C'est la commande `>` en linux. En supposant que votre programme de l'exercice 2 s'appelle `chasseur.exe` et que vous souhaitez que les données qui étaient affichées à l'écran soient stockées dans le fichier `move.dat`, alors la commande est tout simplement :

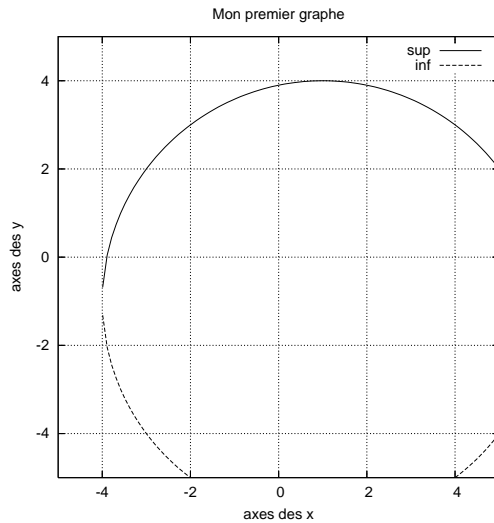


FIG. 1 – Premiers pas en gnuplot.

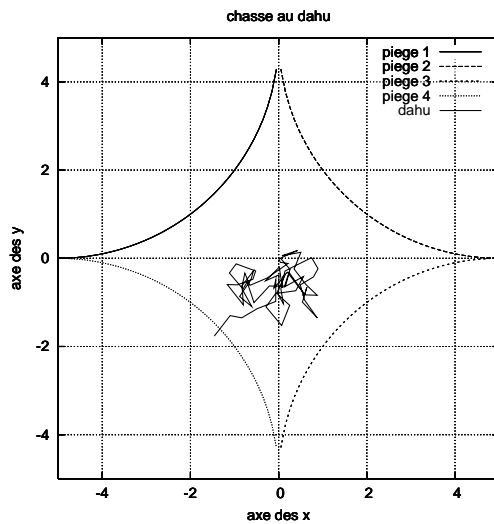


FIG. 2 – La chasse au dahu.

```
./chasseur.exe>move.dat
```

Ouvrez le fichier `move.dat` pour vérifier. Modifiez le programme de l'exercice 2 pour afficher dans la première colonne les abscisses du dahu et dans la deuxième colonne les ordonnées.

Maintenant, pour importer et dessiner ces données dans `gnuplot`, il suffit de modifier la commande `plot` comme suit :

```
plot "move.dat" with lines lt 1 title "dahu"
```

Par défaut, la commande `plot` va considérer que la première colonne du fichier `move.dat` correspond aux abscisses et la deuxième aux ordonnées des points à tracer. L'extension `.dat` du fichier de données est purement conventionnelle. Le paramètre `with lines` signifie qu'il faut relier les points l'un après l'autre. Si vous ne précisez pas `with lines`, alors les points seront tracés sans être relié (essayez!). Le paramètre `lt 1` permet de choisir la couleur du tracé (abréviation de line type). Dans cette exemple on a pris le choix 1. Vous pouvez bien entendu essayer d'autres possibilités (comme -1, 2 ou 3 par exemple).

Exercice : modifier `trace.gp` pour visualiser les rayons d'action des 4 pièges de l'exercice 2 et la trajectoire du dahu (données du fichier `move.dat`). Vous devriez obtenir quelque chose d'approchant la figure 2.

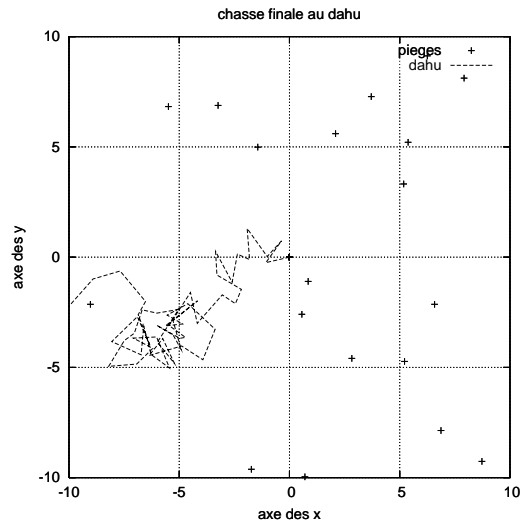


FIG. 3 – La chasse finale au dahu.

4 Le dahu a t'il de la chance ?

Dans la section 2, le dahu n'avait aucune chance de s'en sortir ce qui n'est pas très moral. Corrigeons cela en limitant le périmètre de chasse au dahu, en choisissant un nombre aléatoire de pièges et en répartissant aléatoirement ces pièges à dahu. La chasse s'arrêtera soit si le dahu est attrapé par l'un des pièges, soit si le dahu sort indemne de la zone de chasse.

Écrire un programme qui :

- fixe comme dimension du terrain de chasse en abscisses (constante `DimTerrainChasseX`) la valeur 20, en ordonnées (constante `DimTerrainChasseY`) la valeur 20, fixe comme nombre maximal de pièges (constante `NBPIEGESMAX`) la valeur 20, prenne pour `PasMax` la valeur 3. et pour `RayonAction` la valeur 1.,
- choisisse aléatoirement un nombre de pièges `NbPieges` entre 0 et `NBPIEGESMAX` qui seront posés sur le terrain de chasse,
- pour chaque piège, choisisse aléatoirement une position en `x` entre $-\text{DimTerrainChasseX}/2.$ et $\text{DimTerrainChasseX}/2.$ (idem en `y`).
- écrive dans le fichier `move_final.dat` dans la première colonne l'abscisse de chaque piège, l'ordonnée dans la seconde et 0.00 dans la troisième et quatrième colonne. Chaque coordonnée aura une précision de 2 chiffres après la virgule. Chaque colonne est séparée de la suivante par une tabulation.
- modifie la fonction `Attrape` précédente en ajoutant au prototype les variables entières `NumPiege` qui est le numéro du piège qui attrape le dahu (-1 si aucun piège n'attrape le dahu) et `NbPieges`. On souhaite bien entendu que le résultat de `NumPiege` soit visible pour le reste du programme,
- boucle tant que le dahu n'est pas sorti du terrain de chasse ou ne soit pas attrapé.

Maintenant, pour le bouquet final, vous souhaitez bien évidemment visualiser le résultat de votre programme sous `gnuplot`. La seule chose qui vous en empêche pour l'instant est de savoir comment dessiner les pièges en n'utilisant pour le tracé que les 2 premières colonnes, et la trajectoire du dahu en n'utilisant que les 2 dernières. Pour cela, il suffit d'ajouter sur la ligne du `plot` le paramètre `using x:y` qui utilisera pour abscisse la colonne numéro `x` et pour ordonnées la colonne `y` (qui n'est pas forcément la voisine de `x` d'ailleurs).

Exercice : modifier le programme `trace_gp` pour visualiser vos résultats. Une indication : que font les lignes suivantes ?

```
plot "move_final.dat" using 1:2 title "pieges"
replot "move_final.dat" using 3:4 with lines title "dahu"
```

Vous devriez obtenir un résultat approchant la figure 3, avec bien entendu une trajectoire et une répartition de pièges différentes.