GNU Octave — Fonctions et scripts

Gazette Linux n°112 — Mars 2005

Barry O'Donovan

Article paru dans le n°112 de la Gazette Linux de mars 2005.
Traduction française par Joëlle Cornavin `<jcornavi CHEZ club TIRET internet POINT fr>`**.**
Relecture de la traduction française par François Poulain `<fpoulain CHEZ enib POINT fr>`**.**

Table des matières

1. Introduction

Dans ce second article sur GNU Octave (http://www.octave.org/), je me fonderai sur les bases que j'ai abordées dans le GNU Octave — Une introduction (http://ftp.traduc.org/doc-vf/gazette-linux/html/2004/109/lg109-F.html) en présentant les fonctions et les scripts à l'aide d'un certain nombre d'exemples. L'obtention et l'installation de GNU Octave, ainsi que les sources de documentation officielle ont été étudiées dans le premier article. Référez-vous à celui-ci pour plus d'informations.

2. Les fonctions dans Octave

Comme n'importe quel autre langage de programmation, Octave offre une prise en charge complète pour créer des fonctions. Les fonctions sont un outil essentiel qui permet de fractionner de gros problèmes en un certain nombre de petites tâches. Une fonction doit effectuer une tâche spécifique et doit l'exécuter correctement. Ces critères sont très importants. Plus la tâche qu'une fonction exécute est spécifique, plus elle sera réutilisable ; bien que vous puissiez l'écrire pour vous aider à résoudre votre problème actuel, si elle est bien définie, vous pourrez vous en servir dans de nombreux problèmes futurs. Par « exécuter correctement », j'entends que la fonction devra donner la réponse correcte à une entrée valide tout en signalant une erreur en cas d'entrée invalide ; ce devrait être un véritable « boîte noire ». Une fois écrite et testée, elle devra être de qualité suffisante pour que vous puissiez lui faire confiance sans avoir à la revérifier dans tous vos problèmes futurs.

Une fonction mathématique extrêmement simple qui fait défaut à Octave et dont j'ai eu besoin récemment est la fonction factorielle. Elle est définie ainsi :

Donc, par exemple, 5! = 5·4·3·2·1 = 120. Il existe un certain nombre d'algorithmes pour implémenter cette fonction dont je décrirai certains, pour donner une bonne introduction aux fonctions dans Octave. Commençons par la solution d'une itération :

Listing 1 : lg_factorial1.m (outils/lg112-E/lg_factorial1.m)

Par conséquent, une définition de fonction dans Octave est :

Les fonctions devront être enregistrées à part dans un fichier texte ayant un nom identique à celui de la fonction elle-même, avec l'extension `.m`. Le fichier ci-dessus serait donc enregistré sous `lg_factorial1.m`. Si vous essayez maintenant d'appeler la fonction `lg_factorial1()`, Octave cherche dans la liste des répertoires spécifiés par la variable `LOADPATH` les fichiers se terminant par `.m` qui ont le même nom de base que le nom de la fonction. Si vous voulez créer un référentiel de fonctions sur votre ordinateur et faire inclure à `LOADPATH` ce répertoire automatiquement, vous pouvez ajouter la ligne suivante :

au fichier de configuration d'Octave, `~/.octaverc`. Octave vérifie également le chemin d'accès spécifié dans la variable intégrée `DEFAULT_LOADPATH` qui inclut le répertoire de travail actuel par défaut.

Une fonction peut admettre un certain nombre d'arguments sous la forme d'une liste entre parenthèses séparée par une virgule après le nom de la fonction. Des valeurs de renvoi multiples peuvent également être définies :

Il y a deux règles supplémentaires dans la définition mathématique de la fonction factorielle :

Incorporons ces règles dans notre définition de fonction :

Listing 2 : lg_factorial2.m (outils/lg112-E/lg_factorial2.m)

La fonction teste d'abord pour s'assurer que l'entrée est valide (non négative). Si elle ne l'est pas, elle renvoie une erreur à l'aide de la fonction intégrée `error()`. Elle ne se contente pas d'afficher le message d'erreur, elle affiche également une trace (*traceback*) de toutes les fonctions conduisant à l'erreur. C'est très utile pour les programmeurs qui résolvent des problèmes complexes avec beaucoup de fonctions car ils peuvent résoudre le problème incriminé très rapidement.

Si l'entrée est valide, nous testons le cas zéro et nous employons la commande de retour pour terminer la fonction si elle est vraie. Contrairement à de nombreux autres langages de proogrammation, l'instruction de retour d'Octave n'admet aucun argument, donc il est essentiel que la(les) valeur(s) de retour soi(en)t définie(s) avant de rencontrer un appel de retour comme je l'ai fait avec l'instruction **answer = 1** ci-dessus.

Maintenant, qu'arrive-t-il si `lg_factorial2()` est appelée sans aucun argument ?

Nous pouvons également vérifier pour nous assurer qu'un nombre valide d'arguments a été passé à la fonction à l'aide de la variable intégrée `nargin`. Cette variable est automatiquement initialisée au nombre d'arguments passés. Pendant que nous y sommes, nous devrions également essayer de nous assurer qu'un type de données valide est passé. Malheureusement, Octave n'a pas de fonction `isinteger()` mais nous pouvons vérifier que c'est un nombre réel et non un vecteur. Si un non entier réel est passé, il est arrondi automatiquement à la valeur inférieure par l'opérateur `interval` (2:n).

Listing 3 : lg_factorial3.m (outils/lg112-E/lg_factorial3.m)

Cet exemple présente la fonction intégrée `usage()`. Elle est très similaire à `error()` en ce qu'elle affiche une trace des fonctions conduisant à l'appel pour aider au débogage mais, au lieu d'afficher « error: ... », elle affiche « usage: ... ». Les fonctions `isscalar()` et `isreal()` vérifient que l'argument fourni est une valeur scalaire (par opposition à un vecteur, une chaîne, etc.) et un nombre réel (par opposition à un nombre complexe), respectivement. Le point d'exclamation `!` placé avant inverse le test, de sorte qu'il lit comme si `n` n'était pas un scalaire ou `n` un nombre réel, puis renvoie une erreur.

Vous avez également remarqué que j'ai changé le code pour calculer la factorielle. J'utilise à présent la fonction intégrée `prod()` qui calcule le produit des éléments dans un vecteur donné. Dans ce cas, le vecteur donné est le `interval` 1:n.

Dans l'article précédent, j'ai mentionné qu'Octave comporte une documentation exhaustive qui est accessible via la commande **info** de GNU/Linux. Il y a aussi une fonction d'aide intégrée pour chaque commande. Par exemple :

Une fonction « bien écrite » devrait permettre le recours à la fonction d'aide d'une manière similaire. Le texte d'aide est considéré comme le premier bloc de non *copyright* (voir les commentaires ci-dessous) provenant d'un fichier de fonction :

Listing 4 : lg_factorial4.m (outils/lg112-E/lg_factorial4.m)

L'appel de la fonction d'aide donne à présent :

Un autre algorithme courant pour calculer la factorielle d'un nombre consiste à employer une fonction récursive (une fonction récursive est une fonction qui s'appelle elle-même). Elle peut être implémentée dans Octave (en ignorant le contrôle d'erreur pour l'instant) ainsi :

Listing 5 : lg_factorial5.m (outils/lg112-E/lg_factorial5.m)

3. Test de l'efficacité de la fonction

L'utilisation dans les universités d'Octave a tendance à prendre la forme de simulations à grande échelle et chronophages. À tel point qu'il est important de savoir comment écrire du code à la fois rapide et efficace aussi bien que tester l'efficacité du code que vous écrivez.

Commençons par comparer les trois algorithmes ci-dessus pour calculer la factorielle d'un nombre. Octave possède deux fonctions pour démarrer et arrêter un « minuteur d'horloge » (*wall-clock timer*) :

Pour justifier la comparaison, le seul contrôle que nous effectuons est si n == 0, sinon nous supposerons que c'est un entier positif. Nous utiliserons le programme du listing 5 ci-dessus, la version itérative présentée dans lg_factorial6.m (outils/lg112-E/lg_factorial6.m) et la version de `prod()` présentée dans lg_factorial7.m (outils/lg112-E/lg_factorial7.m). Les commandes exécutées pour la comparaison et les temps se trouvent ici :

Premièrement, la fonction récursive a pris beaucoup plus de temps que prévu. Tout comme avec la plupart des autres langages de programmation, appeler des fonctions (même récursives) exige beaucoup de temps système (*overhead*). Le résultat qui risque de vous surprendre est que l'emploi de la fonction `prod()` est environ six fois plus rapide qu'une itération. C'est parce que Octave, comme Matlab, est assez lent pour itérer et il faudrait donc l'éviter autant que possible.

Comme je l'ai déjà établi, il est important que les fonctions soient bien écrites et bien documentées, en particulier si vous souhaitez partager votre code. D'aucuns peuvent penser que l'effet du contrôle d'une entrée valide peut considérablement ralentir l'exécution d'une fonction. Comparons l'algorithme d'itération dans lg_factorial6.m (outils/lg112-E/lg_factorial6.m) et le même algorithme mais un contrôle d'erreur complet dans lg_factorial8.m (outils/lg112-E/lg_factorial8.m) :

Il y a clairement une différence, mais quand vous considérez que chaque fonction est appelée 100 000 fois, le temps ajouté pour le contrôle d'erreur n'est que de 0,000085 seconde par appel de fonction.

Une autre astuce importante quand on écrit des fonctions Octave est d'éviter de redimensionner les matrices inutilement. Le manuel lui-même stipule que si vous construisez une seule matrice résultante à partir d'une série de calculs, il faut définir la taille de la matrice résultante d'abord, puis y insérer des valeurs. Voici une illustration graphique de la lenteur que cela peut représenter :

4. Scripts Octave

Les fichiers de scripts Octave ne sont qu'une séquence de commandes Octave qui sont évalués comme si vous les avez saisis vous-même à l'invite d'Octave. Ils sont utiles pour configurer des simulations là où vous souhaitez varier certains paramètres pour chaque exécution sans avoir à resaisir les commandes à chaque fois, pour des séquences de commandes qui n'appartiennent logiquement pas à une fonction et pour automatiser certaines tâches.

Les scripts Octave se trouvent dans un fichier portant l'extension `.m`, comme les fonctions, si ce n'est qu'un fichier de script ne doit pas commencer par le mot-clé `function`. Notez que les variables définies dans un script partagent le même espace de noms (ou portée) que les variables définies à l'invite d'Octave.

Voici un exemple simple de script Octave qui calcule le temps nécessaire à l'utilisateur pour créer un tableau d'entiers d'une taille donnée :

Listing 6 : lg_calc_time.m (outils/lg112-E/lg_calc_time.m)

Une fois enregistré dans un fichier nommé de façon apppropriée, vous pouvez l'exécutez comme suit :

5. Scripts Octave exécutables

Il est également possible d'avoir des fichiers de script Octave exécutables comme nous avons des scripts Bash exécutables. C'est une fonctionnalité très intéressante d'Octave pour les gros problèmes où un mélange de programmes, d'outils ou d'applications peut s'avérer nécessaire pour les résoudre.

L'exemple du listing 6 ci-dessus peut facilement être converti en un programme Octave exécutable en ajoutant une seule ligne au début :

Listing 7 : lg_calc_time.sh (outils/lg112-E/lg_calc_time.sh)

Assurez-vous simplement que le chemin d'accès au programme Octave est correct pour votre système (`/usr/bin/octave`), rendez le script exécutable et lancez-le comme suit :

Si vous appelez le script avec des arguments en ligne de commande, ils seront disponibles au moyen de la variable intégrée `argv` et le nombre d'arguments sera contenu dans la variable `nargin` que nous avons déjà évoquée. Un exemple simple en est présenté dans lg_calc_time2.sh (outis/lg112-E/lg_calc_time2.sh), qui est identique au dernier exemple, sauf qu'il lit la taille du tableau à partir de la ligne de commande.

6. En-têtes conventionnels et notes de copyright

Comme je l'ai indiqué dans l'article précédent, Octave n'est pas aussi complet que Matlab pour les fonctions spécialisées. Les développeurs d'Octave sont à l'écoute de nouveaux ajouts bien écrits et « robustes ». Il y a des conventions pour écrire des fichiers de fonction qui comportent des informations sur l'auteur, des restrictions de *copyright*, une date de création, un numéro de version, etc. Les fonctions d'inclusion avec Octave devront être distribuées avec une licence *open source* appropriée (vous trouverez d'autres informations dans l'annexe A de la documentation officielle (http://www.octave.org/doc/octave.html) — en anglais). Avec des en-têtes et des notes de *copyright* correctes, voici à quoi devrait ressembler ma fonction factorielle :

Listing 8 : lg_factorial.m (outils/lg112-E/lg_factorial.m)

7. Le mot de la fin

Il y a de nombreuses astuces et techniques pour écrire du code Octave efficace. Par exemple, on note l'utilisation non sans imagination de la fonction `reshape()` pour les opérations entre les éléments de la même matrice, de façon à éviter une itération. Je suis à l'affût de ce genre d'astuces et si j'en obtiens assez, je les regrouperai dans un article.

Barry O'Donovan est diplômé de la *National University of Ireland* (Galway), avec un *B.Sc. (Hons)* en informatique et en mathématiques. Il prépare actuellement une thèse d'informatique avec le Information Hiding Laboratory (http://www.ihl.ucd.ie/), au *University College Dublin* (Irlande) dans le secteur du marquage numérique audio.

Barry utilise Linux depuis 1997 et son choix actuel va à Fedora Core. Il est membre du Irish Linux Users Group (http://www.linux.ie/). Quand il ne travaille pas à sa thèse, on peut le voir faire un peu de Open Hosting (http://www.openhosting.ie/), au *pub* avec ses amis ou faire de la course dans le parc de sa ville.