



Utilisation de NEDUM dans Octave

Philippe Mallet et Anthony Thevenin

CERFACS / Global Change and Climate Modelling Team

Mai 2010

WN-CMGC-10-42

SOMMAIRE

Introduction :.....	3
Installation d'Octave :	3
Fonction Importdata :	4
Configuration :.....	11
Remarque sur l'utilisation des fonction d'intégration :	12
Changements effectués dans le code de NEDUM :.....	12
ANNEXES :.....	16

Introduction :

NEDUM (*Gusdorf et Hallegatte 2007a,b, Gusdorf et al. 2008*) est un modèle économique urbain permettant de simuler l'évolution continue de villes (prenant en compte les aspects structurels, et les conséquences socio-économiques), développé par l'ENM-CIRED et utilisé dans le cadre du projet STAE-RTRA ACCLIMAT. Alors que ce modèle est développé à l'aide du logiciel Matlab, les contraintes liées au projet imposent de réaliser un portage sous Octave qui est un logiciel similaire à Matlab mais contrairement à lui, il est open source. Ce document traite de l'utilisation de NEDUM sous Octave. On y aborde l'installation et la configuration d'Octave, les problèmes rencontrés et leurs solutions.

Installation d'Octave :

Pour installer Octave, on pourra aller consulter ces deux sites qui sont assez utiles :

<http://wiki.octave.org/wiki.pl?CategoryInstall>

et (très chaudement recommandé) :

http://iste.epfl.ch/cours_matlab/

Bien évidemment, le site officiel :

<http://www.gnu.org/software/octave/>

Octave est un logiciel qui s'utilise en ligne de commande. Mais il y a possibilité d'utiliser une interface graphique comme « Qt octave » (disponible sous Linux comme sous Windows).

Sous Windows, pour écrire les scripts, il est possible de continuer à utiliser l'éditeur de Matlab ou un éditeur comme « notepad++ ».

Pour pouvoir utiliser certaines fonctions, il faudra installer les paquets suivants :

- miscellaneous
- io
- odepkg
- optim
- statistics

Qui sont disponibles ici :

<http://octave.sourceforge.net/packages.php>

Note : Ces paquets ainsi que « notepad++ » sont déjà fournis avec l'installateur Windows.

Fonction Importdata :

Pour importer des données provenant de fichiers CSV (Comma Separated Value), NEDUM utilise la fonction `importdata`. Cette fonction permet aussi d'importer des images ou de l'audio, mais ces fonctionnalités ne sont pas utilisées dans le cadre de NEDUM.

Dans Octave, « `importdata` » n'existant pas, une fonction portant le même nom a été écrite pour pouvoir importer le format de données utilisées dans NEDUM. Cette fonction n'a pas vocation à reproduire le comportement exact de son modèle, mais d'être cohérent avec l'utilisation qui en sera faite dans NEDUM. Ci-dessous est décrit le comportement cette fonction « `importdata` » sous Octave et sous Matlab.

Sous Octave, l'appel à la fonction se fait comme ceci :

```
« importdata('nomDuFichier','délimiteur'); »
```

Voici ce qu'elle fait :

- Ouvre le fichier.
- Lit la première ligne. Chaque champ séparé par le délimiteur spécifié est rangé dans un tableau cellulaire nommé « `colheaders` ». Si un des champ est vide (attention à ne pas mettre un délimiteur en fin de ligne !), la fonction affiche un message d'erreur et s'arrête en retournant un objet vide.
- Lit le fichier ligne par ligne. Chaque ligne vide est ignorée.
- Si une ligne n'est pas vide, elle est ajoutée à une matrice nommée « `data` » qui sera retournée à la fin de la fonction.
- Cette matrice contient autant de colonnes qu'il y a de nom de colonne. Soit « `_n` » le nombre d'éléments sur la ligne « `i` ». Soit « `m` » le nombre de nom de colonne. Si $n < m$, les colonnes « `n+1` » à « `m` » de la ligne « `i` » seront rempli par des « `NA` » (« `Not Available` »). Seront également remplacés par `NA` tous les blanc rencontrés. Tous les éléments qui ne sont pas de type numérique seront remplacés par `NaN` (« `Not a Number` »). Si $n > m$, les éléments situés après la $n^{\text{ième}}$ colonne seront ignorés et ne figureront pas dans la matrice. À la fin de la fonction, un message informe l'utilisateur si la matrice contient des `NaNs` et/ou des `NAs` et/ou si des valeurs ont été ignorées.
- Les chaînes « `inf` », « `Inf` », « `NaN` » et « `nan` » sont considérées comme des nombres.
- La fonction retourne une structure contenant le tableau cellulaire « `colheaders` » et la matrice « `data` ».

Exemple d'utilisation de la fonction sur un fichier texte :

- contenu de « fichiertest.txt » :

```
a;b;c
chaîne;12;13;888
21;inf;Inf
nan;NaN;33
41;
;
```

- Résultats dans Octave :

```
>> importdata('fichiertest.txt',';')
Attention : le fichier « fichiertest.txt » contient des valeurs non numériques.
Attention : il manque des valeurs dans le fichier « fichiertest.txt ».
Attention : le fichier « fichiertest.txt » contient des colonnes superflues.
        Elles seront ignorées.

ans =
{
  colheaders =
  {
    [1,1] = a
    [1,2] = b
    [1,3] = c
  }
  data =
    NaN    12    13
    21    Inf    Inf
    NaN    NaN    33
    41    NA    NA
    NA    NA    NA
}
```

Sous Matlab, l'appel se fait comme ceci (entre crochets, arguments optionnels) :

```
« importdata('nomDuFichier',['délimiteur',[headerlines]]); »
```

Son comportement :

- Ouvre le fichier, si le délimiteur et headerlines ne sont pas précisés, la fonction essaiera de les deviner. Headerlines correspond au nombre de lignes qu'il faut sauter avant de lire les données numériques.
- En plus du tableau cellulaire « colheaders » et de la matrice « data », la fonction retourne un tableau cellulaire « textdata ». Ce tableau contient toutes les lignes de texte avant les lignes de données. Elle retourne également « headerlines » sous forme d'un scalaire.
- Les lignes vides sont ignorées.

- C'est la première ligne de données qui détermine le nombre de colonnes de la matrice. Si ce nombre est égal au nombre d'éléments séparés par le délimiteur dans la ligne de texte au dessus, alors cette ligne de texte deviens le tableau cellulaire « colheaders ». Sinon, il n'y a pas de tableau « colheaders » mais juste un tableau « textdata » et la matrice « data ».

Exemples d'utilisation de la fonction sur un fichier texte :

- contenu de « fichiertest.txt » :

```
a;b;c
11;12;13
21;22;23
```

- il y a trois nom de colonnes et trois colonnes sur la première ligne de données.
- Résultats dans Matlab :

```
>> m=importdata('fichiertest.txt')
m =
    data: [2x3 double]
    textdata: {'a' 'b' 'c'}
    colheaders: {'a' 'b' 'c'}
```

```
>> m.data
ans =
    11    12    13
    21    22    23
```

- contenu de « fichiertest.txt » :

```
a;b;c;d
11;12;13
21;22;23
```

- Il y a un nom de colonne de plus que de colonnes sur la première ligne de données.
- Résultats dans Matlab :

```
>> m=importdata('fichiertest.txt')
m =
    data: [2x3 double]
    textdata: {'a;b;c;d'}
```

```
>> m.data
ans =
    11    12    13
    21    22    23
```

- Soit « n » le nombre d'éléments de la première ligne de données. Si une ligne contient « m » éléments tel que $m < n$, alors la ligne correspondante dans la matrice « data » contiendra les « m » éléments et les colonnes « m » à « n » seront rempli par « NaN ».

- contenu de « fichiertest.txt » :

```
a;b;c
11;12;13
21;
```

- Résultats dans Matlab :

```
>> m=importdata('fichiertest.txt')
```

```
m =
    data: [2x3 double]
    textdata: {'a' 'b' 'c'}
    colheaders: {'a' 'b' 'c'}
```

```
>> m.data
```

```
ans =
    11    12    13
    21   NaN   NaN
```

- Si $m > n$, les « n » premiers éléments seront rangés dans une ligne de la matrice « data », puis les « n » éléments suivants sur la ligne suivante de la matrice et ainsi de suite. Si « m » n'est pas un multiple de « n », le reste est mis dans un ligne qui sera complétée par des « Nan ».

- contenu de « fichiertest.txt » :

```
a;b;c
11;12;13
21;22;23;31;32;33;41
```

- Résultats dans Matlab :

```
>> m=importdata('fichiertest.txt')
```

```
m =
    data: [4x3 double]
    textdata: {'a' 'b' 'c'}
    colheaders: {'a' 'b' 'c'}
```

```
>> m.data
```

```
ans =
    11    12    13
    21    22    23
    31    32    33
    41   NaN   NaN
```

- Si le premier élément d'une ligne n'est pas un nombre ou n'est pas vide, cette ligne et celles qui suivent seront ignorées pour le remplissage de la matrice. Par contre, une ligne débutant par un nombre et contenant au moins une chaîne de caractères dans le reste de la ligne sera prise en compte et tous les éléments qui suivent la chaîne, elle comprise, seront remplacés par des « NaN ». Dans ce cas, les éléments après le n^{ième} seront ignorés ainsi que les lignes suivantes et ce même si elle contiennent des données.

- contenu de « *fichier1.txt* » :

```
a;b;c
11;12;13
21;22;23
chaîne;32;33
41
```

- Résultats dans Matlab :

```
>> m=importdata('fichier1.txt')
m =
    data: [2x3 double]
    textdata: {'a' 'b' 'c'}
    colheaders: {'a' 'b' 'c'}
```

```
>> m.data
ans =
    11    12    13
    21    22    23
```

- contenu de « *fichier2.txt* » :

```
a;b;c
11;12;13
21;chaîne;23;31
31;32;33
```

- Résultats dans Matlab :

```
>> m=importdata('fichier2.txt')
m =
    data: [2x3 double]
    textdata: {'a' 'b' 'c'}
    colheaders: {'a' 'b' 'c'}
```

```
>> m.data
ans =
    11    12    13
    21   NaN   NaN
```

- Les chaînes « inf », « Inf », « NaN » et « nan » sont considérées comme des nombres sauf si « NaN » et « nan » débutent une ligne, ils sont alors considérés comme une chaîne de caractères. On se retrouve alors dans la situation décrite par le point précédent.
- Si toutes les lignes commencent par une chaîne de caractères, puis un séparateur et des données, le tableau « textdata » est rempli de lignes contenant cette chaîne. La matrice « data » est elle rempli selon la première ligne contenant des données.

- contenu de « fichiertest.txt » :

```
a;b;c
```

```
chaîne;11;12;13
```

```
chaîne;21;22;23
```

```
chaîne;31;32;33
```

- Résultats dans Matlab :

```
>> m=importdata('fichiertest.txt')
```

```
m =
```

```
data: [3x3 double]
```

```
textdata: {4x1 cell}
```

```
>> m.data
```

```
ans =
```

```
11 12 13
```

```
21 22 23
```

```
31 32 33
```

```
>> m.textdata
```

```
ans =
```

```
'a;b;c'
```

```
'chaîne'
```

```
'chaîne'
```

```
'chaîne'
```

- contenu de « *fichierstest.txt* » :

```
chaîne;a;b;c
```

```
chaîne;11;12;13
```

```
chaîne;21;22;23
```

```
chaîne;31;32;33
```

- Résultats dans Matlab :

```
>> m=importdata('fichierstest.txt')
```

```
m =
```

```
data: [3x3 double]
```

```
textdata: {4x4 cell}
```

```
>> m.data
```

```
ans =
```

```
11 12 13
```

```
21 22 23
```

```
31 32 33
```

```
>> m.textdata
```

```
ans =
```

```
'chaîne' 'a' 'b' 'c'
```

```
'chaîne' [] [] []
```

```
'chaîne' [] [] []
```

```
'chaîne' [] [] []
```

REMARQUES SUR L'UTILISATION D'« importdata » DANS OCTAVE : il faut préciser le délimiteur du fichier à importer lors de son appel (ce qui n'est pas obligatoire sous Matlab) et il n'y a pas d'argument « headerlines » comme sous Matlab. Il est obligatoire de respecter le format : une ligne dans le fichier = une ligne dans la matrice « data ».

L'appel à « importdata » dans Octave (par exemple en tapant « importdata('lgare.txt','\t') », va afficher le contenu de ce qui a été chargé. Dans Matlab, seule la taille des éléments est affichée. Pour obtenir un comportement similaire sous Octave, il faut utiliser l'option « struct_levels_to_print(0) » en ligne de commande ou dans le fichier « .octaverc » (cf. « configuration »).

Si les données à importer ont été produites sous Windows et que Octave est utilisé sous Linux, il faut convertir les caractères de fin de ligne dans les fichiers. Pour cela, on peut utiliser les commandes « dos2unix » ou « flip » (se référer au manuel pour leur utilisation). Si les fichiers ont été compressés dans une archive zip, la commande « unzip -a nomDeLArchive.zip » convertira les caractères de fin de ligne lors de la décompression.

Enfin, pour que les lettres accentuées s'affichent correctement, il peut être intéressant de convertir l'encodage des fichiers.

Pour connaître l'encodage d'un fichier, utiliser la commande :

```
« file -i nomDuFichier ».
```

Ce qui donne (par exemple) :

```
« nomDuFichier: text/plain; charset=iso-8859-1 ».
```

Et pour le convertir en UTF-8 :

```
iconv -f iso-9959-1 -t UTF-8 nomDuFichier --output nouveauNomDuFichier
```

Configuration :

Pour utiliser de façon permanente des options, il faut les enregistrer dans un fichier nommé « .octaverc » qui doit se trouver dans le répertoire utilisateur.

Afin d'avoir un comportement proche de celui de Matlab, il faut lancer Octave avec le mode « traditionnel » : « octave --traditional » en ligne de commande.

(Voir : <http://www.gnu.org/software/octave/doc/interpreter/Command-Line-Options.html#Command-Line-Options>).

REMARQUE : avec le mode traditionnel, lorsqu'on veut sauver les variables de la session à l'aide de la fonction « save », celle-ci les enregistre par défaut dans un format binaire propre à Matlab (« .mat »). Cependant, il est possible que cet enregistrement ne se fasse pas correctement et qu'il ne soit plus possible d'ouvrir ces fichiers avec la fonction « load » d'Octave. Si c'est le cas, il faut essayer l'option « default_save_options -binary » en ligne de commande ou dans le fichier « .octaverc ». Ainsi les données sont enregistrées dans le format natif d'Octave ce qui ne doit donc plus poser de problèmes (fichier non compatible avec Matlab).

(plus d'infos sur les formats utilisables par Octave :

http://www.gnu.org/software/octave/doc/interpreter/Simple-File-I_002fo.html#Simple-File-I_002fo)

Pour utiliser la fonction « importdata » avec Octave, il est conseillé de créer un répertoire spécifique où l'enregistrer et de charger ce répertoire dans l'environnement d'Octave au démarrage (à faire une fois pour toute dans le fichier de configuration d'Octave). Ainsi, son utilisation sera complètement transparente. Quel que soit le répertoire dans lequel on se trouvera, on ne sera pas obligé d'y copier cette fonction si on souhaite l'utiliser. De plus, si elle se trouve dans le répertoire de travail et qu'on souhaite utiliser Matlab dans ce même répertoire, elle entrerait en conflit avec la fonction « importdata » native de Matlab.

Pour ajouter un répertoire, il faut ajouter la ligne suivante dans le fichier « octaverc » :

```
« addpath('/chemin/du/repertoire/MesFonctionsOctave/') ».
```

(Voir : http://iste.epfl.ch/cours_matlab/base.html#demarrer_quitter ; rubrique « Prologue et épilogue »)

Remarque sur l'utilisation des fonction d'intégration :

Dans les versions de NEDUM antérieures au 19 mai 2010, les résultats obtenus après l'utilisation de la fonction « ode45 » d'Octave étaient différents de ceux obtenus sous Matlab. Cependant une simple intégration par la méthode d'Euler donnait des résultats semblables sous les deux logiciels (mais différents de ceux donnés par « ode45 »). Nous avons donc diminué le pas de temps pour affiner le résultat de cette méthode qui est peu précise.

Suite à ce changement de pas de temps, il s'avère que la fonction « ode45 » d'Octave donne des résultats corrects (par rapport à ceux de Matlab).

D'autres fonctions d'intégration sont disponibles dans le paquet « ode » d'Octave, avec un document qui résume toutes les méthodes utilisées par les différentes fonctions ainsi que leur utilisation.

Il est recommandé au lecteur d'essayer différentes combinaisons de fonctions et de pas de temps afin de trouver celle qui lui permettra d'obtenir des résultats correspondant à ses attentes.

Dans l'installation de base d'Octave, la fonction « lode » permet elle aussi de résoudre des équations différentielles, mais il faut faire attention car ses arguments d'entrée sont différents de ceux utilisés pour les fonctions « odeXX ». Dans Octave, la commande « help lode » permettra d'avoir des informations sur cette fonction et la commande « help lode_options » des informations sur ses options.

Changements effectués dans le code de NEDUM :

Ce sont les changements effectués sur la version du 20 mai 2010. Ces changements sont répertoriés par fichiers et par ligne.

NEDUM.m :

ligne 20 : ajout de « close all; »

Cela ferme toutes les fenêtres contenant des graphiques qui seraient encore ouvertes après une simulation précédente.

Ainsi on est sûr que les graphiques à l'écran correspondent bien à la simulation en cours car sinon Octave n'ouvre pas une nouvelle fenêtre, mais trace dans celles qui sont déjà ouvertes. Si quelque part dans le programme on demande à ouvrir une nouvelle fenêtre (avec l'instruction « figure »), on se retrouvera avec plus de fenêtres que la simulation ne le demande.

ligne 82 : mise en commentaire de « la pause »

On évite ainsi l'intervention de l'utilisateur au cours d'une simulation (nécessaire dans le cadre de la plateforme).

ligne 129 : création de « fctAintegrer » alias de la fonction « evolution »

Permet à la ligne suivante de donner en argument « fctAintegrer » à la fonction « ode45 » au lieu de « evolution » avec tous ses paramètres.

Cela permet également de bien voir que la fonction « ode45 » reçoit quatre arguments : la fonction à intégrer, les temps aux quels faire l'intégration, les conditions initiales, les options.

lignes 135 à 168 : mise en œuvre de la méthode de Heun.

C' est une méthode d'intégration encore peu précise, mais plus que la méthode d'Euler.

La variable « h » a été introduite pour plus de clareté.

Le code a été modifié pour travailler avec des vecteurs colonne afin ne pas avoir à faire de transposées dans le calcul, ce qui implique qu'à la ligne 166, le résultat est transposé pour rester compatible avec le reste du programme.

lignes 147 à 150 et 160 à 163 : mise en commentaires pour éviter l'affichage de warning.

options.m :

ligne 29 : remplacement de l'option « Euler » par « Heun » et mise à zéro.

parameters.m :

ligne 34 : remplacement du vecteur « t » pour n'aller que jusqu'à dix.

compute_equilibrium.m :

lignes 54 et 62 : l'option « Display » a été laissée, mais elle n'est pas (encore) supportée par Octave (d'où le warning lors de l'exécution).

evolution.m :

ligne 36 : changement de « disp(sprintf(» par « fprintf » et ajout de « \n » dans la chaîne de caractères.

Au contraire de « disp », « sprintf » et « fprintf » permettent une écriture formatée des variables.

« `sprintf` » permet d'écrire dans un objet de type « `string` » (d'où le « `s` »), il faut donc utiliser « `disp` » pour afficher ce qu'on écrit dedans.

« `fprintf` » permet d'écrire dans un fichier si on lui précise ou d'afficher directement à l'écran si non.

Il faut ajouter « `\n` » dans la chaîne de caractère de « `fprintf` » car contrairement à « `disp` », il ne fait pas de retour à la ligne.

importfile.m :

ligne 1 : ajout de l'argument « `delimiteur` ».

ligne 9 : ajout de l'argument « `delimiteur` ».

charge_temps.m :

lignes 24 et 94 : ajout de l'argument « `'\t'` » à l'appel de la fonction « `importfile` ».

ligne 158 : remplacement du « `if... disp...` » qui affichait la progression tous les 20 % par « `waitbar(index/length(cent))` » qui permet d'afficher simplement la progression en pourcentage.

Bien que pratique et ludique, il est possible que l'utilisation de cette fonction ralentisse légèrement l'exécution de la boucle.

import_data.m :

lignes 24 à 27 : ajout de l'argument « `'\t'` » à l'appel de la fonction « `importfile` ».

courbe_new.m :

ligne 1 : ajout de « `figure` » afin de ne pas écraser le premier jeu de courbe lié au fait que « la pause » ait été supprimée dans NEDUM, ainsi le premier jeu est sûr d'être analysé.

lignes 9,15 et 20 : ajout d'une légende.

Par rapport à l'ancienne version de NEDUM:

Dans la version antérieure au 20 mai 2010 était utilisée la fonction « `fsolve` ». Parmi les données envoyées à cette fonction, se trouvait une valeur « `inf` » qui était le résultat d'une division par zéro effectuée précédemment. Alors que Matlab ne tient pas compte d'une telle valeur, Octave le fait ; ce

qui change le résultat. Un rapport de bug a été soumis afin que dans les prochaines versions d'Octave le comportement soit le même que sous Matlab.

En attendant, si le lecteur doit utiliser cette fonction, il est recommandé de tester les données avant de les envoyer à « fsolve » et de les remplacer par une valeur par défaut. Par exemple en faisant :
« `var(isinf(var)) = 0` »

ANNEXES :

```
1 function [out, delimiter] = importdata(varargin)
2     switch nargin % test du nombre d'arguments.
3         case {1},
4             disp('vous devez spécifier un délimiteur');
5             return;
6         case {2},
7             FileName = varargin{1}; % ler argument.
8             delimiter = sprintf(varargin{2}); % pr que "\t" soit
correctement pris en compte.
9             otherwise
10                disp('Trop d'arguments');
11                return;
12            end
13
14        file_id = fopen(FileName, 'rt');
15
16        string = fgetl(file_id); % lere ligne récupérée.
17        out.colheaders = strsplit(string,delimiter); % coupe string en morceaux
à chaque délimiteur
18        % range les morceaux dans un tableau cellulaire.
19        [lines,cols] = size(out.colheaders);
20
21        if sum(cellfun("isempty",out.colheaders))~=0 % cellfun retourne un
tableau de 0 ou 1.
22            disp('Attention :un nom de colonne est vide');
23            return;
24        end
25
26        no = 0;
27        NaNs = false;
28        NAs = false;
29        colSuperflues = false;
30
31        while ~ feof(file_id) % on lit le fichier ligne par ligne jusqu'a la
fin.
32
33            string = fgetl(file_id);
34            a = strsplit(string,delimiter);
35
36            if ~(isempty(a)) % on ne traite que les lignes non vides.
37
38                [linesa,colsa] = size(a);
39
40                if colsa>cols % on a plus de colonnes dans les données
que de noms.
41                    colSuperflues = true;
42                end
43
44                for i = 1:min(cols,colsa)
45                    c = a(1,i); % ième élément de la ligne.
46                    if cellfun("isempty",c) % «c» est vide.
47                        NAs = true;
48                        c = NA;
49                    else
50                        c = str2num(char(c));
51                        % char converti un tableau cellulaire
```

```

52         % de chaines de caractères en matrice
53         % de chaines de caractères.
54         % Si effectué sur un tableau de
55         % plusieurs éléments, la plus grande
56         % chaine déterminera la taille de
57         % toutes les autres qui seront
58         % complétées par des espaces.
59         % On ne veut pas de ces espaces donc
60         % on convertit élément par élément.
61         %
62         % str2num converti une chaine de
63         % caractères en valeur numérique.
64
65         if isempty(c)
66             % si on a une chaine de
67             % caractères non convertible
68             % en valeur numérique,
69             % str2num met un blanc.
70             % on a donc une valeur qui
71             % n'est pas un nombre.
72             NaNs = true;
73             c     = nan;
74         end
75     end
76     d(i) = c; % on range chaque élément dans un
77     % vecteur.
78 end
79
80 if colsa < cols % il manque des données.
81     NaNs = true;
82     d(colsa+1:cols) = NA;
83 end
84
85 no = no + 1;
86 out.data(no,:) = d;
87 % on remplit la matrice data ligne par ligne.
88 end
89 end
90
91 if NaNs
92     fprintf('Attention : le fichier « %s » contient des valeurs non
numériques.\n',FileName);
93 end
94
95 if NaNs
96     fprintf('Attention : il manque des valeurs dans le fichier « %s
».\n',FileName);
97 end
98
99 if colSuperflues
100     fprintf('Attention : le fichier « %s » contient des colonnes
superflues.\n
Elles seront ignorées.\n',FileName);
101 end
102
103 status = fclose(file_id) ;
104
105 % Fin importdata

```