Analyse Numérique

Corrigés des Travaux Pratiques 2013 – 2014
Séance 1

2. On peut procéder comme suit :

```
A = rand(5,5);     % création d'une matrice aléatoire 5x5
L = eye(5);         % création d'une matrice identité 5x5
L                                % est-elle  bien identité?
L(:,1) = A(:,1)/A(1,1);      % copier & diviser colonne 1
L(2:end,1) = -L(2:end,1);           % inverser le signe
L*A                               % éléments hors diagonale
               % de la première colonne de L*A sont nuls
L
inv(L)                     % constatez que la seule différence
                         % entre inv(L) et L est le changement
          % de signe des éléments de la première colonne
```

3. Une manière d'écrire la fonction (enregistrée obligatoirement dans le fichier `prdiag.m` du répertoire de travail) est :

```
function p = prdiag(A)
% P = PRDIAG(A) retourne le produit des éléments
% se trouvant sur la diagonale principale de A
p = 1;
[m, n] = size(A);
for i = 1:min(m,n)
    p=p*A(i,i);
end
```

La vérification peut se faire comme suit :

```
A=tril(rand(5,5)); % partie triangulaire inferieure
                   % d'une matrice aléatoire
A                  % est-elle bien triangulaire?
det(A)             % déterminant de A
prdiag(A)          % produit des éléments diagonaux
% alors?
```

Notez que le commentaire au début de la fonction `prdiag` a pour but d'expliquer ce que cette fonction fait ; il est aussi affiché lorsqu'on entre `help prdiag`.

4. La première fonction peut s'introduire en ligne de commande :

```
fun1 = @(x) exp(x./pi)./log10(x.+pi);
```

Notez l'utilisation des opérations élément par élément ; bien que ces opérations ne soient pas indispensables pour évaluer la fonction `fun1` en un seul point donné, elles permettent aussi d'évaluer `fun1` en un certain nombre de points simultanément ; en particulier, pour un vecteur `x`, `fun1(x)` retournera alors le vecteur dont la $i$eme composante sera `fun1(x(i))`. Cela devient utile, par exemple, pour afficher un graphe à l'aide de la commande `plot`, pour laquelle on doit fournir les vecteurs `x` des abscisses et `y` des ordonnées. Dans ce cas on peut procéder comme suit :

```
fun1 = @(x) exp(x./pi)./log10(x.+pi);
x = -1:0.01:1;
y = fun1(x);
plot(x,y)
```

Remarque : comme `pi` est une contante, la première opération élément par élément `x./pi` est identique à l'opération vectorielle de multiplication par l'inverse d'un scalaire, dans notre cas il s'agit de `x/pi` . De plus, Octave accepte l'addition d'un scalaire avec un vecteur : le scalaire est alors rajouté à tous les éléments de ce vecteur. Ainsi, l'opération `x.+pi` est équivalente à `x+pi` . En résumé, la fonction `fun1` peut aussi s'introduire en ligne de commande comme

```
fun1 = @(x) exp(x/pi)./log10(x+pi);
```

Pour ce qui est de la deuxième fonction, elle ne peut pas s'écrire en une seule instruction ; on peut par contre créer un fichier `fun2.m` comme suit

```
function y = fun2(x)
if(x==0)
    y = 1;
else
    y = sin(x)./x;
end
```

Dans ce cas on ne peut pas passer un vecteur comme argument (à cause de la structure `if-else-end`). La construction du vecteur `y` des ordonnées peut se faire en utilisant une boucle :

```
x = -1:0.01:1;
for i=1:length(x)
    y(i) = fun2(x(i));
end
plot(x,y)
```