Initiation à Matlab

Disponible en ligne depuis

https://niess.github.io/matlab-instru/

Réponses aux exercices de la 2ème session

V. Niess

lundi 21/11/2016

Exercice 1 : et pourtant il est rond! ... #1

> *script d'exemple : cercle.m*

- On paramètrise les coordonnées `(x, y)` d'un point du cercle en fonction de l'angle trigonométrique phi, selon:

```
phi = [-pi : pi / 6 : pi];
x = cos(phi);
y = sin(phi);
```

- Pour tracer `y` en fonction de `x` on utilise la commande `plot`:

```
plot(x, y, 'ko-');
```

  Le 3 eme argument, `'ko-'`, code le format d'affichage. Soit dans ce cas des marqueurs circulaires et un trait continu entre les points. On peut observer *deux aberrations* sur le graphe: Le cercle apparait comme un polygone et elliptique de surcroît.

Exercice 1 : et pourtant il est rond! ... #2

- Le cercle apparait elliptique du fait que les unités des axes x et y ne sont pas identiques.

  > Il est possible d'égaliser manuellement les axes en redimensionnant la fenêtre de la figure avec la souris. Cependant, pour résoudre automatiquement ce problème il est plus commode utiliser la commande :
  >
  > ```
  > axis equal
  > ```

- Les points échantillonnés sur le cercle sont connectés par des lignes droites. Aussi, même après égalisation des axes le cercle n'apparait toujours pas circulaire mais comme un polygone.

  > Pour résoudre ce second problème il faut diminuer le pas d'échantillonnage en phi, e.g. :
  >
  > ```
  > phi = [-pi : pi / 100 : pi];
  > ```

Exercice 2 : les émoticônes anonymes

> *script d'exemple : emoticones.m*

- Pour réaliser les émoticônes on peut se servir d'un cercle pour la tête et les yeux et d'un demie cercle pour la bouche. Par exemple, les commandes ci-dessous réalisent le tracé de `:)`.

```
% construction du cercle unité
phi = -pi:pi/100:pi;
x = cos(phi);
y = sin(phi);

% Emoticône pour :)
figure(1);
plot(x, y, 'k');                          % La tête
axis equal;
hold on;
plot(0.2 * x + 0.4, 0.2 * y + 0.4, 'b');   % L'œil droit
plot(0.2 * x - 0.4, 0.2 * y + 0.4, 'b');   % L'œil gauche
plot(0.5 * x(y<0), 0.5 * y(y<0), 'r');     % La bouche
```

Exercice 2b : nommer les émoticônes

> *script d'exemple : emoticones_b.m*

- Par exemple, pour `:(`, la séquence de code suivante embellit puis sauve votre ouvrage, selon:

```
title('Monsieur Grognon');
xlabel('Largeur');
ylabel('Hauteur');
saveas(gcf, 'grognon.fig');
```

> Notez l'utilisation de la commande `gcf` qui renvoie l'identifiant de la figure courante.

Exercice 3 : fréquence des lettres ... #1

> *script d'exemple : denombrement.m*

- La forme fonctionnelle de `help` renvoie une chaîne de caractères contenant le texte de l'aide. Combinée avec la fonction `lower` on obtient le texte convertit en minuscules, selon :

```
texte = lower(help('plot'));
```

- Pour convertir le texte en un tableau de codes `ACSII` on peut utiliser la fonction `double` ou simplement soustraire le 1er caractère de l'alphabet, soit `'a'`. En effet les codes `ASCII` pour `a`, `b`, `c`, `d` sont consécutifs. Ainsi `'a' - 'a'` renvoie `0`, `'b' - 'a'` renvoie `1`, ..., et `'z' - 'a'` renvoie `25`. Soit :

```
codes = texte - 'a';
```

  réalise la conversion en codes ASCII translatés de `'a'`.

Exercice 3 : fréquence des lettres … #2

- Pour dénombrer les lettres on peut utiliser la fonction `histc` avec pour classes (*bins en anglais*) `[0;1[`, `[1;2[`, …, `[24;25[` et `[25]`. Soit :

```
bins = [0 : 25];
n = histc(codes, bins);
```

- Pour trier les résultats, on ordonne les occurrences `n` et les indices correspondant selon :

```
[n, K] = sort(n);
bins = bins(K);
```

> Notez que par défaut `sort` trie du plus petit au plus grand.

Exercice 3 : fréquence des lettres … #3

- Pour obtenir les 3 caractères les plus utilisés il faut convertir les 3 derniers codes en chaîne de caractères, selon:

  ```
  lettres = char('a'+bins(end:-1:1))
  ```

  Pour `plot` on obtient `'eto'` et pour `syntax` on a `'eat'`. En combinant les deux textes d'aide on trouve `'eta'` qui sont bien les 3 lettres les plus fréquentes en Anglais.

- La fonction `bar` vous permet également de réaliser un tracé de la fréquence des lettres sous forme d'histogramme. Consultez l'aide pour plus d'informations.

Exercice 4 : erreur numérique ... #1

> *script d'exemple : exponentielle.m*

- Pour calculer la dérivée numérique de la fonction exponentielle on peut utiliser les fonctions Matlab `diff` ou `gradient`. Pour un vecteur `y` de taille `n`, ces fonctions sont analogues aux opérations suivantes :

```
% diff
d = y(2:end) - y(1:end-1)
%gradient
g(1) = y(2) - y(1);
g(2:end-1) = (y(3:end) - y(1:end-2)) / 2;
g(end) = y(end) - y(end-1)
```

> Notez que `diff` renvoie un tableau de taille `n-1` approchant la dérivée de y par la droite alors que `gradient` renvoie un tableau de taille `n` approchant la dérivée par la droite pour le 1^{er} indice, par la gauche pour le dernier et des 2 cotés pour les points intermédiaires.

Exercice 4 : erreur numérique ... #2

- Du fait que `gradient` est centré, sauf aux bords, il est plus pratique à utiliser que `diff`, dans ce cas de figure. Ainsi on aura par exemple :

```
dx = 0.01;
x = 0 : dx : 1;
yd = gradient(y) / dx;
```

- Pour calculer la primitive numérique on dispose de la fonction `cumtrapz` qui effectue une intégrale cumulée de `y` par les trapèzes. Soit :

```
yp = cumtrapz(y) * dx;
```

> On pourra remarquer que `cumtrapz` renvoie la primitive de `y` qui vaut 0 en `x(1) = 0`. Soit `exp(x)-1`. Pour utiliser la primitive conventionnelle, i.e. `exp(x)`, il faut donc ajouter `1` au résultat de `cumtrapz`, soit :
>
> ```
> yp = yp + 1
> ```

Exercice 4 : erreur numérique ... #3

- On peut tracer l'erreur relative, `e`, en échelle logarithmique avec la fonction `semilogy` selon :

```
semilogy(x, abs(yd - y) ./ y, 'r-');
```

- On constate que l'erreur numérique relative sur la dérivée est constante sauf aux bords de l'intervalle ou elle est 2 ordres de grandeur plus grande. Un calcul analytique donne :

$$e = \frac{\sinh(dx)}{dx} - 1 \simeq \frac{dx^2}{6}$$

  soit une erreur qui augmente quadratiquement avec le pas d'échantillonnage.

- Pour la primitive numérique on pourra constater que l'erreur relative n'est pas constante sur l'intervalle `[0;1]` mais continue de croître avec `dx`.

Exercice 5 : tchou-tchou! ... #1

> *script d'exemple : analyse.m*

- Pour charger l'enregistrement sonore on utilise la fonction `load`, selon :

```
load('train');
```

- L'amplitude acoustique est codée dans le vecteur `y` et la fréquence d'échantillonnage dans la variable `Fs`. Vous pouvez tracer l'évolution de l'amplitude acoustique au cours du temps avec la commande plot, selon :

```
dt = 1 / Fs;
t = [0 : length(y) - 1] * dt;
plot(t, y, 'k');
```

- On peut constater que le signal est oscillant. Son enveloppe se décompose en deux blocs de 0.4s et 0.9s de durée, à 0.1s d'intervalle. Ces blocs correspondent à 2 coups de sifflet du train.

Exercice 5 : tchou-tchou! ... #2

- Pour analyser le spectre en fréquence du signal on utilise la fonction `fft`. Comme le signal est réel le module de sa transformée de Fourier est paire, i.e. $Y(-f) = Y(f)^*$. Aussi on peut se contenter d'analyser les fréquences positives. Soit :

```
Y = fft(y);
nf = floor(length(y) / 2);
Y = Y(1:nf) * dt;
f = [0:nf-1] / length(y) * Fs;
```

> La résolution en fréquence est $\Delta f = \frac{F_s}{\text{length}(y)} = 0.636 \text{ Hz}$.

- Pour tracer le spectre on utilise une échelle logarithmique en `dB` mieux adaptée à la grande dynamique d'amplitudes perceptibles à l'oreille. Soit par exemple :

```
plot(f, 20 * log10(abs(Y)), 'k');
```

Exercice 5 : tchou-tchou! ... #3

- Le spectre de fréquence du signal présente 3 pics clairement visibles aux fréquences de 700 Hz, 890 Hz et 1170 Hz. Aux fréquences triples on observes des harmoniques de rang 3 de ces mêmes pics. On peut également observer quelques autres pics de plus faible amplitude.
- On peut avoir rapidement un aperçu temps-fréquence du signal avec la fonction `specgram`, selon :

  ```
  [Y, f, t] = specgram(y, [], Fs);
  ```

  Cette fonction calcul le spectrogramme du signal, c'est a dire sa transformée de Fourier sur une fenêtre glissante. Le second argument, laissé vide ici, détermine le nombre de points utilises pour la transformée de Fourier. Pour tracer le résultat on peut utiliser les fonctions `contour` ou `imagesc`.

Exercice 6 : filtrage d'un bruit blanc ... #1

> *script d'exemple : filtre.m*

- La fonction `psd` accepte en argument la fréquence d'échantillonnage en plus du vecteur de données, `y`. Elle renvoie alors la densité spectrale de bruit et les fréquences correspondantes, selon:

  ```
  [Px, f] = psd(s, Nf, Fs);
  ```

  Le second argument, `Nf`, est le nombre de points utilisés pour calculer la transformée de Fourier. La densité spectrale de bruit est la moyenne quadratique de plusieurs transformées de Fourier d'échantillons de `y` de taille `Nf`.

  - Il est judicieux d'utiliser une puissance de 2, l'algorithme de FFT étant alors plus rapide.
  - Il faut faire un compromis dans le choix de `Nf`. Plus `Nf` est grand, plus la résolution en fréquence sera grande mais moins les moyennes obtenues seront précises. Pour le signal utilisé, `Nf = 2^11 = 2048` donne un bon rendu.

Exercice 6 : filtrage d'un bruit blanc ... #2

- En traçant la densité spectrale du bruit on constatera que son contenu fréquentiel est sensiblement plat, avec une equi-répartition dans toutes les gammes de fréquence. On dit que le *bruit est blanc*, quand il contient toutes les fréquences en parts égales, par analogie avec les spectres de couleur. Le signal, par contre est piqué à basse fréquence, en dessous de 10 Hz. Aussi, un filtrage passe-bas, ne sélectionnant que les faibles fréquences, permettra d'améliorer nettement la qualité de la mesure.
- On peut réaliser un filtre passe bas d'ordre 6 avec la fonction`butter` de la `toolbox signal`. Par exemple, pour une fréquence de coupure `fc = 4` Hz :

```
fc = 4; % Hz
[B, A] = butter(6, fc / (Fs / 2))
```

Exercice 6 : filtrage d'un bruit blanc ... #3

- La mesure filtrée est très propre comparée aux données brutes. Vous pouvez comparer les résultats à l'oreille avec la fonction `sound`. Cependant, il subsiste *plusieurs défauts* :
  - Le résultat n'est pas parfaitement Gaussien car des ondulations basse fréquence du bruit subsistent.
  - L'amplitude du signal filtré est légèrement plus faible. Pour rejeter efficacement le bruit on a également été obligé de couper une partie du signal.
  - Le signal filtré est en retard par rapport au signal vrai. Le filtrage a introduit un déphasage.
- Ces défauts sont *irréductibles*. En augmentant l'ordre du filtre on fait un filtrage plus `serré` mais on introduit également des instabilités numériques. Pour corriger ces défauts il faudrait utiliser une autre stratégie: l'ajustement des données brutes par la forme du signal lorsque celle-ci est connue, soit une Gaussienne ici. Ceci fait l'objet de l'exercice 6b.

Exercice 7: on a perdu les coefficients! ... #1

> *script d'exemple : algebre.m*

- Le calcul de la moyenne, μ_i, du i$^{\text{ème}}$ étudiant peut s'écrire selon :

$$\mu_i = \sum_{j=1}^{5} c_j n_{ij} \; / \; \sum_{j=1}^{5} c_j$$

où les c_j sont les coefficients recherchés et les n_{ij} les notes du i$^{\text{ème}}$ étudiant dans la j$^{\text{ème}}$ matière.

- On peut poser :

$$x_i = c_i \; / \; \sum_{j=1}^{5} c_j$$

et se ramener à la résolution d'un système d'équations linéaires de la forme $AX = B$, avec `A` la matrice des notes n_{ij}, et `B` le vecteur des moyennes μ_i.

Exercice 7: on a perdu les coefficients! … #2

- Dans un monde `idéal` on pourrait se contenter des résultats des cinq 1^{er} étudiants et inverser la sous matrice carrée de `A` correspondante pour trouver les x_i. Cependant, *du fait des arrondis* lors de la retranscription des moyennes cette solution ne donne *pas le bon résultat*. On pourrait alors décider de recommencer l'exercice pour tous les choix de cinq étudiants parmi l'effectif total de 12, puis moyenner les x_i obtenus. Une solution plus élégante et mieux définie formellement est l'utilisation du *pseudo-inverse*, selon :

  ```
  X = pinv(A) * B;
  ```

  Le vecteur `X` ainsi obtenu représente le meilleur compromis pour les valeurs de coefficients normalisées, *au sens des moindres carrés*. C'est a dire les coefficients normalisés qui reproduisent au mieux l'ensemble des moyennes, minimisant $|AX - B|^2$.

Exercice 7: on a perdu les coefficients! … #3

- Pour calculer la normalisation `sum(c)` on utilise la valeur connue de `c(2) = 3`, pour l'Anglais. Soit :

```
c = round(3 * X / X(2))
```

ou les coefficients ont été arrondis à l'entier le plus proche avec la fonction `round`. La solution est :

```
c = [3; 3; 5; 6; 6];
```

Exercice 1b : interpolation du cercle … #1

> *script d'exemple : interpolation.m*

- L'interpolation linéaire est le mode par défaut de `interp1`. Soit :

```
phip = 0 : pi/100 : 2 * pi;
xl = interp1(phi, x, phip);
```

  réalise l'interpolation linéaire de `x` en fonction de `phi` aux points `phip`. Pour l'interpolation par les splines la syntaxe est la même mais en utilisant la fonction `spline`.

- Le tracé de `(x, y)` en trait plein et `(xl, yl)` sur un même graphe montre que *par défaut Matlab réalise une interpolation linéaire pour connecter les points sur le graphe*. L'interpolation par les splines, par contre, restitue quasi-parfaitement la forme du cercle. Cependant, il n'est pas rigoureusement possible à partir des seuls valeurs échantillonnées `(x, y)` de déterminer si la courbe d'origine était un cercle ou un polygone. Les deux solutions sont donc tout aussi valables.

Exercice 1b : interpolation du cercle … #2

- De façon générale:
  - L’interpolation linéaire est plus robuste car elle garantie que la valeur interpolée est comprise entre les valeurs échantillonnées. Les interpolations d’ordre supérieurs, comme les splines, peuvent par contre produire des valeurs très différentes.
  - L’interpolation par les splines assure la continuité des dérivées au points de mesure. Elle restitue donc mieux la forme du signal d’origine au voisinage des mesure si ses dérivées sont bien continues.

Exercice 8: récursivité et Fractales ... #1

> *script d'exemple : julia.m*

- On définie une fonction `julia` qui prend en argument une matrice de valeurs Z_n et renvoie $Z_{n+1} = Z_n^2 + c$, selon :

```
function Z = julia(Z)
c = 0.285 + j * 0.01; % par exemple
Z = Z.^2 + c
```

- La fonction `julia` représente également les valeurs du module de Z_{n+1}. Pour avoir un meilleur contraste on effectue une bijection de `[0;+inf[` dans `]0;1]` en traçant `exp(-abs(Z))` au lieu de `abs(Z)`. La fonction `imagesc` permet d'afficher une matrice de valeurs en image `bitmap`. Soit :

```
imagesc(exp(-abs(Z)));
axis equal;
```

Exercice 8: récursivité et Fractales ... #2

- La palette de couleur peut être changée avec la commande `colormap`. La palette `hot` affiche les valeurs en code thermique: rouge (chaud) pour les grandes valeurs, `1` ici, ou bleu (froid) pour les petites valeurs, soit `0` ici. La syntaxe est :

  ```
  colormap hot;
  ```

  L'aide de `hsv` donne une liste des palettes disponibles.

- Pour afficher la correspondance entre les valeurs des éléments de la matrice et la couleur affichée on dispose de la commande `colorbar`. La syntaxe est simplement :

  ```
  colorbar;
  ```

Exercice 8: récursivité et Fractales ... #3

- Pour initialiser les valeurs de `Z` selon un échantillonnage régulier du plan complexe on utilise la fonction `meshgrid`. Par exemple :

```
x0 = [-1.5 : 5E-03 : 1.5];
y0 = [-1.5 : 5E-03 : 1.5];
[X0, Y0] = meshgrid(x0, y0);
Z = X0+j*Y0;
```

  pour réaliser un échantillonnage de `[-1.5;1.5] x [-1.5;1.5]` par pas de `5E-03`.

- Pour observer comment le rayon de Z_n évolue en fonction de la valeur initiale Z_0, il suffit ensuite d'appeler récursivement la fonction `julia`.

Exercice 6b: ajustement Gaussien … #1

> *scripts d'exemple : ajustement.m, modele.m et objectif.m*

- L'initialisation des valeurs de paramètres `X0` du modèle conditionne en grande partie les performances d'un ajustement. Un choix pertinent permet une convergence plus rapide de la minimisation ainsi que d'éviter une convergence vers des solutions miroirs. Pour l'amplitude `X0(1)` du signal on peut se contenter de prendre `max(y)`. Pour la valeur centrale et l'écart type, `X0(2)` et `X0(3)`, comme le signal occupe l'essentiel de notre échantillon de mesure, on peut assimiler les valeurs `y` à une densité de probabilité et estimer ses moments d'ordre 1 et 2. Soit :

```
p = y / trapz(y);
X0(2) = trapz(p .* t);
X0(3) = sqrt(trapz((t - X(2)) .^2 .* p));
```

Exercice 6b: ajustement Gaussien ... #2

- L'ajustement de `X` se fait avec la fonction `fminsearch` selon :

  ```
  X = fminsearch('objectif', X0, [], t, y);
  ```

  Notez que l'on doit passer `t` et `y` en arguments supplémentaires pour la fonction d'objectif. Par ailleurs, le 3^{eme} argument est un tableau vide, indiquant ainsi que l'on utilise les options de minimisation par défaut. On aurait également pu passer un structure d'options générée avec la fonction `optimset`.

- En traçant sur un même graphe les valeurs simulées, les vraies valeurs de signal, le modèle initial et le modèle ajusté on peut constater que:
  - Le modèle ajusté est en parfait accord avec le vrai signal.
  - Le modèle ajusté améliore significativement l'accord au signal par rapport au modèle initial.

Exercice 6b: ajustement Gaussien … #3

- La connaissance précise de la forme du signal, et du bruit permet de retrouver ses caractéristiques: amplitude, position et largeur. Le bon accord entre le modèle ajusté et le signal vrai tient cependant au fait que l'on dispose d'un large échantillon de mesure. Dans un cas pratique:
    - *On ne connait pas la forme exacte du signal et du bruit*. On ne peut que faire des hypothèses à ce sujet. C'est une source d'incertitude systématique par la suite.
    - *L'échantillon de mesure est plus réduit*, introduisant des incertitudes d'ordre statistique. Vous pouvez par exemple refaire l'ajustement avec une mesure simulée contenant 10 fois moins de points pour comparer. L'étude de la distribution de la fonction objectif permet d'estimer ces incertitudes statistiques.

    > Certains algorithmes d'ajustement, `polyfit` par exemple, s'accompagnent d'une estimation de l'erreur statistique.