Prép'ISIMA

Programmation avancée en C

Approximation

Avec les méthodes des droites de regression ainsi que les ajustements

Clément Payard
Mathieu Laurençot

Encadrant : M. Suzanne ÉLODIE

2021

Table des matières

1 Rappel rapide des méthodes

1.1 Cas général des méthodes

Pour les deux méthodes, le but est le même, trouver l'équation d'une droite avec un nombre de points n de la forme :

$$y = ax + b$$

1.2 Méthode des droites de régression

Pour cette méthode, nous devons utiliser la formule du cours suivante (Où $\bar{x}$ et $\bar{y}$ sont les moyenne respective de x et de y) :

$$a = \frac{\overline{xy} - \bar{x}\bar{y}}{(\bar{x^2}) - (\bar{x}^2)}$$

et

$$b = \bar{y} - a\bar{x}$$

1.2.1 Question 1

Nous appliquons la formule du cours donnée précédemment.

1.2.2 Question 2

Nous appliquerons la même méthode que la méthode à la Question 1, sauf que nous le ferons pour les deux parties du tableau. Puis nous effectuons la moyenne des deux droites pour en obtenir une unique.

1.3 Méthode des ajustements

Pour nous permettre de trouver une droite, nous devons modifier les deux équations pour arriver à une autre équation de la forme.

$$Y = AX + B$$

Nous allons donc modifier les équations initiales.

Suite à cela, nous appliquerons la méthode des droites de régression avec les variables correspondantes.

1.3.1 Question 3

$$y = ax^b$$

$$ln(y) = ln(a) + ln(a^b)$$

$$ln(y) = ln(a) + b * ln(a)$$

On pose donc $Y = ln(y)$, $X = ln(x)$, $A = b$ et $B = ln(a)$.

1.3.2 Question 4

$$y = ce^{dx}$$

$$ln(y) = ln(c) + ln(e^{dx})$$

$$ln(y) = ln(c) + d * x * ln(e)$$

On pose donc $Y = ln(y)$, $X = x$, $A = d * ln(e)$ et $B = ln(c)$.

2 Présentation des programmes commentés

Nous ne mettrons pas l'intégralité du code des différentes résolutions, mais ils sont consultables ici pour la méthode des ajustements et ici pour la méthode des régréssions.

2.1 Nos structures

Nous avons choisis d'utiliser plusieurs structures pour plusieurs cas :

1. La première, la structure *point*, qui est utilisé partout. En effet, cette structure nous permet de stocker les différents points qui nous sont nécessaire pour trouver les droites respectant la diagonalisation.
2. Puis, la structure *Maillon*. Cette structure est essentielle pour une chose : former des listes, ce qui nous permet d'ajouter un nombre de points indéfinis au départ, puis d'en rajouter, supprimer, etc, sans perdre de la mémoire.
3. La structure *Liste*, qui est la suite logique de structure *Maillon*.
4. La structure polynôme, qui contrairement au précédent tp, sera uniquement de degrès 0 ou 1.
5. Enfin, même si ce n'est pas vraiment une structure, une fonction s'appelant *transformefloatenpoly* transforme une liste de points en un tableau, permettant alors d'y accéder plus rapidement et facilement.

2.1.1 Listes/Points

1. Présentation du code général Le code pour les listes ainsi que pour les points est trouvable ici.
   Pour former un point, nous avons tout simplement besoin d'un x et d'un y associé à celui-ci. La structure des points est donc la suivante :

```
typedef struct point
{
  float x;
  float y;
} point;
```

   Puis, nous voulons rassembler ses points en des listes, nous avons donc besoin de deux structures supplémentaires : la structure *Maillon* et la struture *Liste*. Ces deux structures permettent de structurer nos données, les utiliser, et les afficher dans des graphiques.

```
typedef struct Maillon{
  struct Maillon *suiv;
  point val;
}Maillon;

typedef struct Liste{
  struct Maillon *first;
}Liste;
```

De plus, les fonctions usuelles des listes sont également créé :

(a) *creerListe*
(b) *detruireListe*
(c) ajouteDebut
(d) ajouteFin
(e) *afficheListePoints*
(f) ListLenght
(g) supprDebut
(h) supprFin
(i) supprValeur
(j) supprMaillon
(k) ListeToTabsPoints, qui transforme une liste en un tableau.
(l) ViderListe

2. Présentation des fonctions utilisées pour résoudre le problème de départ
   Les différentes fonctions utilisées sont disponible dans la fin de ici.

(a) moyXY, qui nous permet de faire la moyenne des X et des Y
(b) moyX et moyY, qui nous permet de faire la moyenne
(c) moyXmoyY
(d) moyXcar
(e) carmoyX
(f) lnListe
(g) moyXY2tab

2.1.2 Polynômes

Le code pour les polynôme est trouvable ici.

Comme vous le savez, le C n'inclue pas de type "polynôme". Nous avons donc dû creer la structure suivante :

```
typedef struct Polynome{
  float *p;
  int maxDeg;
}polynome;
```

Cette structure à pour but de prendre un tableau, où l'indice du tableau nous permet de trouver le degrès de x, et les valeurs stockés dans le tableau sont les différents coefficients pour chaque x du polynôme.

Nous avons également fait les diverses fonctions de bases :

1. *polynome *creePolynome(int maxDeg)*, qui nous permet de créer un pointeur sur un polynôme.
2. void destroyPolynome(polynome *p), qui supprime un polynôme.
3. void affichepolynome(polynome *p), comme son nom l'indique, affiche le polynôme passé en paramètre.
4. polynome *transformefloatenpoly(float unfloat), qui convertie un flottant en un polynôme.
5. polynome *addPolynome(polynome *p1, polynome *p2), qui nous permet d'additionner (et dans le même temps de soustraire) deux polynôme entre eux

```
int i;
polynome *poly =
  creePolynome((p1->maxDeg > p2->maxDeg) ? p1->maxDeg : p2->maxDeg);

for (i = 0; i < p1->maxDeg + 1; ++i)
  {
    /* printf("Degrèse de p1 %d \n", p1->maxDeg); */
```

```
      poly->p[i] = p1->p[i];
    }

  for (i = 0; i < p2->maxDeg + 1; ++i)
    {
      poly->p[i] += p2->p[i];
    }
  destroyPolynome(p1);
  destroyPolynome(p2);
  return poly;
```

6. polynome *multPolynome(polynome *p1, polynome *p2). Cette fonction multiplie deux polynômes entre eux.

```
  int i, j;
  polynome *poly = creePolynome(p1->maxDeg + p2->maxDeg);

  for (i = 0; i < p1->maxDeg + 1; ++i)
    {
      for (j = 0; j < p2->maxDeg + 1; ++j)
        {
  poly->p[i + j] += p1->p[i] * p2->p[j];
        }
    }
  destroyPolynome(p1);
  destroyPolynome(p2);
  return poly;
```

2.2 Présentation de la méthode droites de régression

2.2.1 Simple

Pour résoudre avec cette méthode, nous avons tout simplement effectué, grâce à nos fonctions usuelles, les calculs "normaux" pour faire une droite de régression.

Nous avons également pensé à libérer la mémoire de nos différents tableaux.

```
polynome *UneDroiteReg(Liste l){
  polynome *P = creePolynome(1);
  int n = ListLenght(l);
  float **Res = ListeToTabsPoints(l);
  P->p[1] = (moyXY(Res, n) - moyX(Res, n) * moyY(Res, n))/(moyXcar(Res, n) - carmoyX(
  P->p[0] = moyY(Res, n) - moyX(Res, n)*((moyXY(Res, n) - moyX(Res, n) * moyY(Res, n)
  free(Res[1]);free(Res[0]);free(Res);
  return P;
}
```

2.2.2 Double

Pour résoudre avec deux droites de régression, nous avons d'abord séparé en deux notre tableau de points. Puis, nous avons appliqué la première méthode distinctement sur les deux tableaux. Enfin, nous avons fait la moyenne de ces deux droites pour calculer la dernière droite (qui est le polynôme renvoyé).

```
polynome *DeuxDroiteReg(Liste l){
float **Res = ListeToTabsPoints(l);
```

```
  int n = ListLenght(l);
  float moy = moyY(Res, n);
  Maillon *m = l.first;
  Liste l1 = creerListe();
  Liste l2 = creerListe();
  for(int i = 0; i < n; i++){
    if(m->val.y < moy){
      ajouteFin(&l1, m->val);
    }else {
      ajouteFin(&l2, m->val);
    }
    m = m->suiv;
  }
  polynome *p1 = UneDroiteReg(l1);
  polynome *p2 = UneDroiteReg(l2);
  polynome *Ret = creePolynome(1);
  Ret->p[0] = (p1->p[0] + p2->p[0])/2;
  Ret->p[1] = (p1->p[1] + p2->p[1])/2;

  detruireListe(l1);
  detruireListe(l2);
  free(Res[1]);free(Res[0]);free(Res);
  return Ret;
}
```

2.3 Présentation de la méthode des ajustements

Le code pour la méthode des ajustements est disponible ici.

Pour la résolution par la méthode des ajustement, et après avoir résolu l'équation du type $Y = AX + B$, nous commençons tout d'abord, pour les deux méthodes, de transformer notre liste en tableau, ainsi que de créer le polynôme de degrés 1. Puis, grâce à la fonction *lnListe*, nous créons un second tableau qui contiendra les points avec application de la fonction logarithme (car on a posé $Y = ln(y)$, $X = ln(x)$, $A = b$ et $B = ln(a)$ pour la question 3 et $Y = ln(y)$, $X = x$, $A = d * ln(e)$ et $B = ln(c)$ pour la question 4).

```
polynome *P = creePolynome(1);
int n = ListLenght(listedepoint);
float **Tnormal = ListeToTabsPoints(listedepoint);
float **Tln = lnListe(Tnormal, n);
```

2.3.1 Question 3 (Exponentiel)

Nous initialisons les variables *a* et *b*, qui seront des variables temporaires.

Puis, nous calculons les moyennes de X et de Yln, pour faciliter la lecture du code et rendre le code plus facilement compréhensible.

```
float moyenneX = moyX(Tnormal, n);
float moyenneYln = moyY(Tln, n);
```

Subséquemment, nous pouvons effectuer le calcul des ajustement de la puissance appliquant la formule du cours, ainsi qu'en utilisant la formule posé précédemment.

Le premier calcul sera pour calculer l'exposant présent devant x, donc d.

```
a = (moyXY2tab(Tnormal, Tln, n) - moyX(Tnormal, n) * moyY(Tln, n)) /
      (moyXcar(Tnormal, n) - carmoyX(Tnormal, n));
```

Et le second sera pour la coefficient présent devant x.

```
float temp = moyenneYln - (a * moyenneX);
  b = exp(1) * exp(temp);
```

Enfin, nous pouvons retourner le polynôme P.

2.3.2 Question 4 (Puissance)

Nous initialisons les variables *a* et *b*, qui seront des variables temporaires.

Subséquemment, nous pouvons effectuer le calcul des ajustement de la puissance appliquant la formule du cours, ainsi qu'en utilisant la formule posé précédemment.

Le premier calcul sera pour calculer l'exposant, donc b.

```
b = (moyXY(Tln, n) - moyX(Tln, n) * moyY(Tln, n)) /
    (moyXcar(Tln, n) - carmoyX(Tln, n));
```

Et le second sera pour la coefficient présent devant x.

```
float temp = moyenneYnl - (b * moyenneXnl);
  a = exp(temp);
```

Enfin, nous pouvons retourner le polynôme P.

3 Présentation des Jeux d'essais avec commentaires

Les différents résultats sont disponibles en faisant les test grâces aux fonctionnalités implémentées dans le programme.

3.1 Série S

Liste de points sans difficulté particulière, mais pas dans le bon ordre.

Résultats : On peut voir que les résultats sont les bons : l'approximation à l'air (graphiquement) efficace.

3.2 Les trois séries :

Suites de points normaux. Difficultés potentielles : aucune, sauf pour le dernier qui ne pourra sûrement pas être calculé (car ce n'est pas une suite de points avec différents x)

Résultats :

1. Les approximations sont les bonnes, aucune difficulté.
2. Les approximations sont bonnes, aucune difficulté également.
3. Ne peut pas se calculer : en effet, les points sont sur le même X. Les approximations sont donc incalculables (et infaisable).

3.3 Dépenses mensuelles et revenus :

Pour ces données, il y a un grand nombre de données, ainsi qu'un axe des x qui commence avec de "grandes valeurs".

Difficultés potentielles : précisions dû aux résultats importants obtenus.

Résultats :Les polynômes ont soit des coefficients très importants, soit des coefficients presque négligeable. En revanche, on peut constater qu'il y a des approximations de calcul dans les deux méthodes. En effet, les deux polynômes finaux ne sont pas exactement les mêmes, même si ils sont tous les deux du même ordre de grandeur.

3.4 Série chronologique avec accroissement exponentiel

L'exponentiel de la forme $y = ce^{dx}$ est parfaitement sur les données, ce qui prouve que cette approximation est adaptée.

3.5 Vérification de la loi d Pareto

L'exponentiel de la forme $y = ax^b$ est parfaitement sur les données, ce qui prouve la vérification de cette loi.

3.6 Commentaire global

Pour ses différents jeux d'essais, on peut constater plusieurs choses :

1. La méthode des droites de régression et deux droites de régression ne donnent pas la même chose
   Et c'est normal. En effet, avec l'un, nous appliquons la formule sur l'ensemble des points. Tandis que sur la deuxième, nous séparons le tableau en deux, puis effectuons une moyenne des deux droites obtenues. Leur efficacité dépend donc des points donnés, et est donc "aléatoire".
2. Les méthodes d'ajustements sont plus ou moins précises en fonction des cas (comme on peut le constater dans le programme)
3. Évaluation des coûts : Pour les droites d'approximation, la complexité est de l'ordre (o)n^2, elle est également de (o)n^2 pour les méthodes des questions 3 et 4.

4 SDL

Le code qui nous permet de gérer la fenêtre SDL est disponible ici.

4.1 Début

Pour démarrer SDL, nous devons initialiser de nombreuses variables, comme par exemple :

— La variable *Stape*, qui nous permet de fermer SDL si elle est égale à 0,
— *size*, qui va nous permettre de gérer la taille de l'écran,
— des variables permettant de garder un nombre d'image par seconde (fps) constant et agréable
— des variables permettant de détecter où le curseur de la souris se trouve sur l'écran
— les varibles permettant de dessiner le graphique
— etc.

4.2 Affichage et fonctionnalité

4.2.1 Affichage

Pour effectuer l'affichage d'une fenêtre SDL, nous devons passer par une boucle *while*.

Puis, nous distinguerons trois cas grâce à un *if* (et *else if*).

1. Dans le premier cas, SDL dessinera l'écran, s'il n'a pas été dessiné depuis un certain temps
2. Sinon, nous vérifierons également si les courbes sont en adéquation avec les polynômes. Si ce n'est pas le cas, nous entrons alors dans le *else if* qui va nous permettre d'écraser l'image précédente. Enfin,

3. si nous passons les deux conditions précédentes, nous devons **absolument** endormir le Central Processing Unit (CPU). Cela nous permet de ne pas utiliser tout le processeur de l'ordinateur.

Puis, nous avons aussi un cas de débogage. En effet, si l'on n'est pas entré dans le while depuis une seconde ou plus, il peut y avoir un problème. On recommence alors une seconde "propre", en mettant certaines variables à 0.

4.2.2 Fonctionnalités :

Divers fonctionnalités sont présentes : Vous pouvez afficher la fenêtre grâce à la touche "g". Vous pouvez désormais voir la liste des points, les courbes représentant les différentes méthodes d'approximation, ainsi que la liste de points à droite.

De plus, si jamais vous voulez rajouter des points à la liste, cette fonctionnalité est disponible grâce au bouton gauche de la souris. Le bouton droit aura pour effet de supprimer le point sélectionné.

Le curseur aura alors une position (x et y) qui sera automatiquement ajouté dans la liste des points. Les courbes ainsi que les polynômes vont s'adapter automatiquement !

D'autre part, vous pouvez zoomer et dézoomer grâce à la molette de la souris.

Enfin, vous pouvez activer ou désactiver les différentes courbes des fonctions en appuyant sur leur nom.

4.3 Fin de SDL

La fonction *end-sdl* nous permet de fermet la fenêtre SDL proprement, ainsi que faire les opérations nécessaires pour vider la mémoire qui a besoin d'être libéré.

5 Conclusion sur les méthodes

Différentes méthodes peuvent être utilisées pour une approximation. En effet, dans certains cas, les droites d'approximations seront plus efficaces, tandis que, par exemple, pour la loi de Pareto, nous préférerons une des méthodes d'ajustement.