Algorithmes numériques – Rapport

Interpolation et Approximation

Axel Delsol, Pierre-Loup Pissavy

Décembre 2013

Table des matières

1 Préambule

1.1 Structure du programme

Nous avons conçu un programme principal avec menus, présenté sous la forme suivante :

```
Menu principal : Interpolation et Approximation

Entrez n le nombre de points : 2
(Saisie de la série de points...)

(Affichage du tableau correspondant...)
Quelle résolution utiliser ?
1- Newton
2- Neuville
3- Régression Linéaire
4- Approximation par une fonction exponentielle
5- Approximation par une fonction "puissance"
9- Nouvelle série de points (Menu principal)
0- Quitter
Votre choix :
```

FIGURE 1.1 – Apercu : Menu Principal

2 Interpolation

L'interpolation, en analyse numérique, est un ensemble de méthode permettant d'obtenir une équation mathématique passant par tous les points d'une liste données.
Pour cette partie, les équations mathématiques recherchées sont des polynômes.

Notation pour la suite :

- La liste comporte N éléments (x_i, y_i).
- Les polynômes recherchés sont de la forme $P_{N-1}(x) = \sum_{i=0}^{N-1} a_i \cdot x^i$.

2.1 Méthode de Newton

2.1.1 Présentation

La forme du polynôme par la méthode de Newton est la suivante :

$$P_{N-1}(x) = \sum_{i=0}^{N-1} \left(a_i \cdot \prod_{j=1}^{j=i} (x - x_j) \right)$$

Pour se faire , on utilise une méthode de recherche de coefficients récursive appelée la méthode des *différences divisées*. Le calcul des valeurs des différences divisées se fait à l'aide d'une fonction :

La différence divisée de degré 0 est :

$$\forall i = 1, \ldots, N : \bigtriangledown^0 y_i = y_i$$

La différence divisée de degré k est :

$$\forall i = k+1, \ldots, N : \bigtriangledown^k y_i = \frac{\bigtriangledown^{k-1} y_i - \bigtriangledown^{k-1} y_k}{x_i - xk}$$

Ensuite, on a directement les coefficients du polynôme de Newton par la relation

$$\forall i = 1, \ldots, N-1 : a_i = \bigtriangledown^i y_{(i+1)}$$

Enfin, on peut retrouver la forme développée du polynôme à l'aide de la relation :

$$\forall i = 0, \ldots, N : P_i(x) = \begin{cases} a_{N-1} & \text{si } i{=}0 \\ a_{N-1-i} + (x - x_{N-i}) \cdot P_{i-1}(x) & \text{sinon} \end{cases}$$

2.1.2 Programme

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "polynome.h"

void newton (double ** tab, int n)
{
  int i, k;
  double** t= (double**) malloc(n*sizeof(double*));
  for (i=0; i<n; i++)
  {
    t[i]= (double*) malloc((i+1)*sizeof(double));
  }

  //initialisation des valeurs : on récupère les y.
  for (i=0; i<n; i++)
  {
```

```c
    t[i][0]=tab[1][i];
  }

  //calcul des differences divisees
  for (k=1; k<n; k++)
  {
    for (i=k; i<n; i++)
    {
      t[i][k]=(t[i][k-1]-t[k-1][k-1])/(tab[0][i]-tab[0][k-1]);
    }
  }

  //tableau de polynomes
  polynome** tabP= (polynome**) malloc(n*(sizeof(polynome*)));
  for (i=0; i<n; i++)
  {
    tabP[i]=(polynome*) malloc(sizeof(polynome));
  }
  tabP[0]->d=0;
  tabP[0]->poln=(double*) malloc(sizeof(double));
  tabP[0]->poln[0]=t[n-1][n-1];

  for (i=1; i<n; i++)
  {
    tabP[i]=addPoly(creerPoly(1,"valeur",t[n-1-i][n-1-i]),mulPoly(creerPoly(2,"valeur",-tab[0][n-1-i], 1.),
         tabP[i-1]));
  }
  redimensionnerPoly(tabP[n-1]);

  //affichage
  menuAffichage(tabP[n-1]);
  ecartPoly(tab,n,tabP[n-1]);
  printf("\n");

  //libération mémoire
  for(i=0;i<n;i++)
  {
    free(tabP[i]->poln);
    free(tabP[i]);
    free(t[i]);
  }
  free(tabP);
  free(t);
}
```

Figure 2.1 – Code : newton.c

2.2 Méthode de Neuville

2.2.1 Présentation

2.2.2 Programme

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "polynome.h"

void neuville (double ** tab, int n)
{
  int i, k;
  polynome*** t= (polynome***) malloc(n*sizeof(polynome**));
  for (i=0; i<n; i++)
  {
    t[i]= (polynome**) malloc((i+1)*sizeof(polynome*));
  }

  //initialisation des valeurs : on récupère les y.
  for (i=0; i<n; i++)
  {
    t[i][0]=creerPoly(1,"valeur", tab[1][i]);
  }

  //calcul des differences divisees
  for (k=1; k<n; k++)
  {
    for (i=k; i<n; i++)
    {
      t[i][k]=mulSPoly((1/((tab[0][i-k])-(tab[0][i]))),addPoly(mulPoly(creerPoly(2,"valeur", -(tab[0][i]), 1.), t[i-1][k-1]), mulPoly(creerPoly(2, "valeur", tab[0][i-k], -1.),t[i][k-1])));
    }
  }
  //polynome àretourner
  redimensionnerPoly(t[n-1][n-1]);

  //affichage
  menuAffichage(t[n-1][n-1]);
  ecartPoly(tab,n,t[n-1][n-1]);
  printf("\n");

  //libération mémoire
  for(i=0;i<n;i++)
  {
    for(k=0;k<i;k++)
    {
      free(t[i][k]->poln);
      free(t[i][k]);
    }
    free(t[i]);
  }
  free(t);
}
```

FIGURE 2.2 – Code : neuville.c

2.3 Résultats de tests

2.3.1 Exemple tiré d'un TD

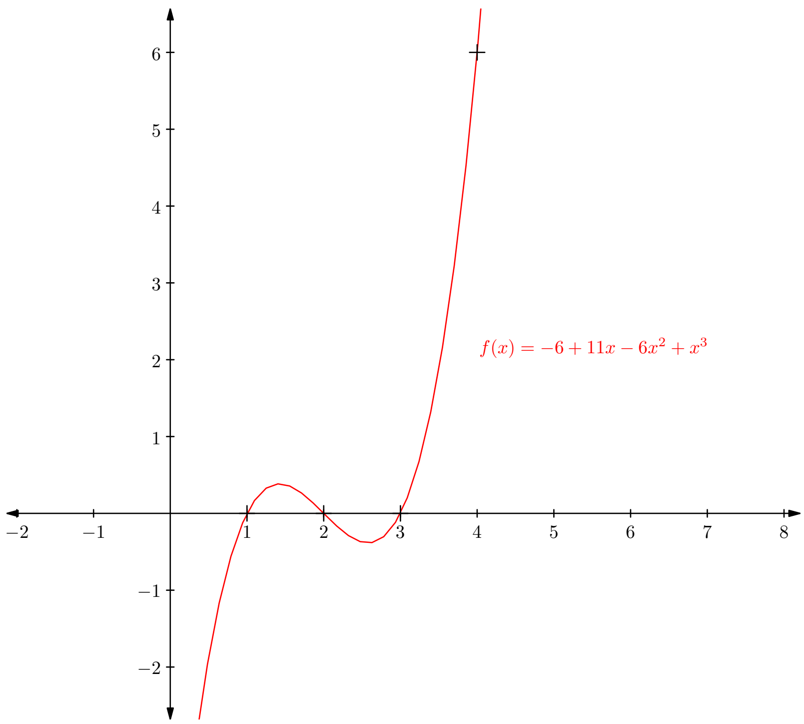
x_i	1	2	3	4
y_i	0	0	0	6

Tableau 2.3.1 – Série 1

On obtient alors :

Méthode de Newton :
$P(x) = -6.00 + 11.00 \cdot x - 6.00 \cdot x^2 + 1.00 \cdot x^3$
Erreur : 0

Méthode de Neuville :
$P(x) = -6.00 + 11.00 \cdot x - 6.00 \cdot x^2 + 1.00 \cdot x^3$
Erreur : 0



Graphique 2.3.1 – Interpolations de Newton et Neuville – (Tableau 2.3.1)

2.3.2 Densité de l'eau en fonction de la température

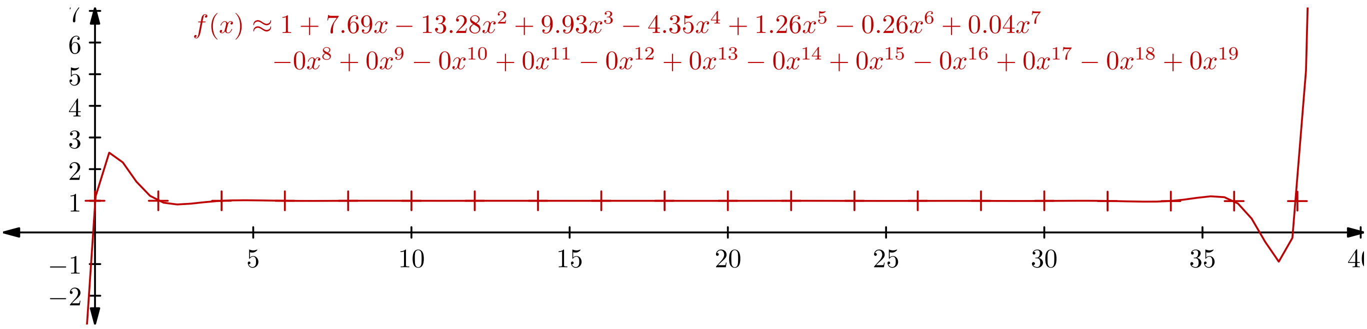
x_i	0	2	4	6	8	10	12	14	16	18
y_i	0.999870	0.999970	1.000000	0.999970	0.999880	0.999730	0.999530	0.999530	0.998970	0.998460
x_i	20	22	24	26	28	30	32	34	36	38
y_i	0.998050	0.999751	0.997050	0.996500	0.996640	0.995330	0.994720	0.994720	0.993330	0.993260

TABLEAU 2.3.2 – Mesures

On obtient alors :

Méthode de Newton :
$P(x) \approx 0.999870 + 7.693711 \cdot x - 13.276666 \cdot x^2 + 9.932303 \cdot x^3 - 4.345460 \cdot x^4 + 1.259124 \cdot x^5 - 0.258585 \cdot x^6 + 0.039240 \cdot x^7 - 0.004520 \cdot x^8 + 0.000402 \cdot x^9 - 0.000028 \cdot x^{10} + 0.000002 \cdot x^{11} - 0.000000 \cdot x^{12} + 0.000000 \cdot x^{13} - 0.000000 \cdot x^{14} + 0.000000 \cdot x^{15} - 0.000000 \cdot x^{16} + 0.000000 \cdot x^{17} - 0.000000 \cdot x^{18} + 0.000000 \cdot x^{19}$
Erreur : 0.000002166566117323

Méthode de Neuville :
$P(x) \approx 0.999870 + 7.693711 \cdot x - 13.276666 \cdot x^2 + 9.932303 \cdot x^3 - 4.345460 \cdot x^4 + 1.259124 \cdot x^5 - 0.258585 \cdot x^6 + 0.039240 \cdot x^7 - 0.004520 \cdot x^8 + 0.000402 \cdot x^9 - 0.000028 \cdot x^{10} + 0.000002 \cdot x^{11} - 0.000000 \cdot x^{12} + 0.000000 \cdot x^{13} - 0.000000 \cdot x^{14} + 0.000000 \cdot x^{15} - 0.000000 \cdot x^{16} + 0.000000 \cdot x^{17} - 0.000000 \cdot x^{18} + 0.000000 \cdot x^{19}$
Erreur : 0.000028505775100296



GRAPHIQUE 2.3.2 – Interpolation de Newton et Neuville – (Tableau 2.3.2)

2.3.3 3 séries

x_i	10	8	13	9	11	14	6	4	12	7	5
$y_i^{(1)}$	9.14	8.14	8.74	8.77	9.26	8.10	6.13	3.10	9.13	7.26	4.74
$y_i^{(2)}$	7.46	6.77	12.74	7.11	7.81	8.84	6.08	5.39	8.15	6.42	5.73
$y_i^{(3)}$	6.58	5.76	7.71	8.84	8.47	7.04	5.25	12.50	5.56	7.91	6.89

Tableau 2.3.3 – Trois séries S1, S2, S3

On obtient alors :

<u>**Série 1 :**</u>

Méthode de Newton :
$P(x) \approx -229.550000 + 299.165750 \cdot x - 173.107636 \cdot x^2 + 58.546955 \cdot x^3 - 12.731862 \cdot x^4 + 1.859906 \cdot x^5 - 0.184968 \cdot x^6 + 0.012375 \cdot x^7 - 0.000533 \cdot x^8 + 0.000013 \cdot x^9 - 0.000000 \cdot x^{10}$
Erreur : 0.000000000016217532

Méthode de Neuville :
$P(x) \approx -229.550000 + 299.165750 \cdot x - 173.107636 \cdot x^2 + 58.546955 \cdot x^3 - 12.731862 \cdot x^4 + 1.859906 \cdot x^5 - 0.184968 \cdot x^6 + 0.012375 \cdot x^7 - 0.000533 \cdot x^8 + 0.000013 \cdot x^9 - 0.000000 \cdot x^{10}$
Erreur : 0.000000000012119518

<u>**Série 2 :**</u>

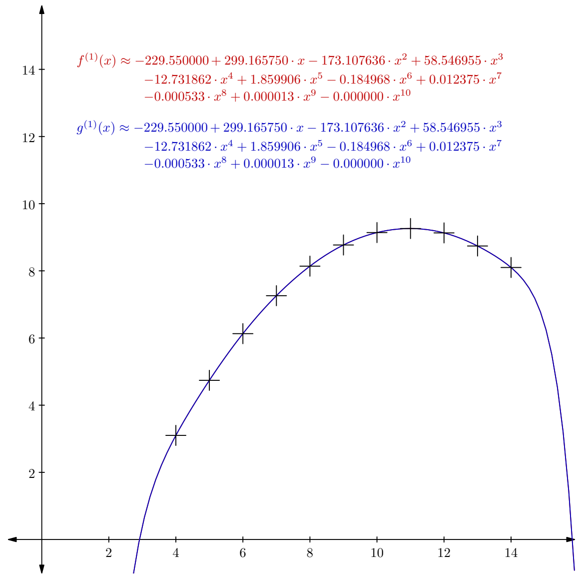
Méthode de Newton :
$P(x) \approx -12345.190000 + 16608.066492 \cdot x - 9870.941498 \cdot x^2 + 3416.593892 \cdot x^3 - 763.094009 \cdot x^4 + 114.979985 \cdot x^5 - 11.842442 \cdot x^6 + 0.823658 \cdot x^7 - 0.037039 \cdot x^8 + 0.000973 \cdot x^9 - 0.000011 \cdot x^{10}$
Erreur : 0.000000000774325735

Méthode de Neuville :
$P(x) \approx -12345.190000 + 16608.066492 \cdot x - 9870.941498 \cdot x^2 + 3416.593892 \cdot x^3 - 763.094009 \cdot x^4 + 114.979985 \cdot x^5 - 11.842442 \cdot x^6 + 0.823658 \cdot x^7 - 0.037039 \cdot x^8 + 0.000973 \cdot x^9 - 0.000011 \cdot x^{10}$
Erreur : 0.000000001033081661

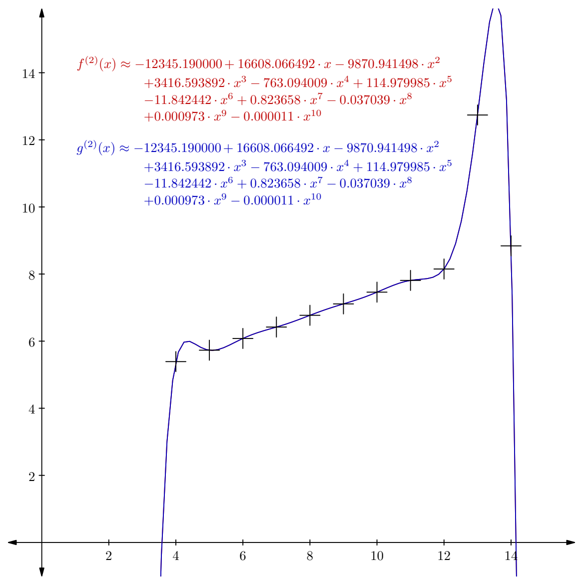
<u>**Série 3 :**</u>

Méthode de Newton :
$P(x) \approx -568559.640000 + 739678.381270 \cdot x - 424130.450858 \cdot x^2 + 141275.523224 \cdot x^3 - 30298.693006 \cdot x^4 + 4375.222059 \cdot x^5 - 431.155992 \cdot x^6 + 28.652640 \cdot x^7 - 1.229803 \cdot x^8 + 0.030806 \cdot x^9 - 0.000342 \cdot x^{10}$
Erreur : 0.000000081067879843

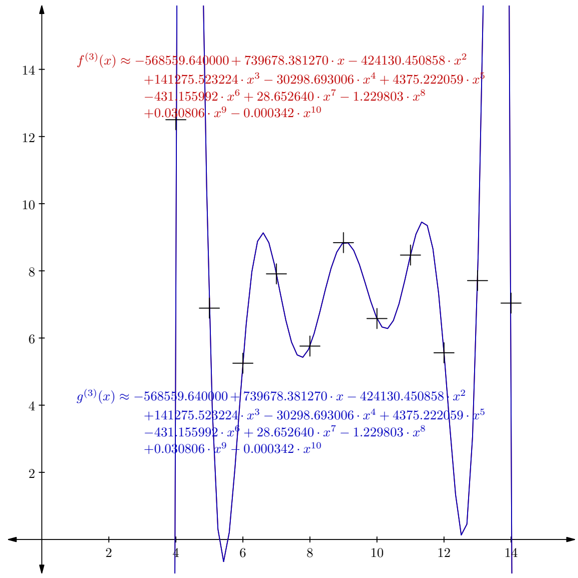
Méthode de Neuville :
$P(x) \approx -568559.640000 + 739678.381270 \cdot x - 424130.450858 \cdot x^2 + 141275.523224 \cdot x^3 - 30298.693006 \cdot x^4 + 4375.222059 \cdot x^5 - 431.155992 \cdot x^6 + 28.652640 \cdot x^7 - 1.229803 \cdot x^8 + 0.030806 \cdot x^9 - 0.000342 \cdot x^{10}$
Erreur : 0.000000035876804073



Graphique 2.3.3 – Interpolations de Newton et Neuville – Série 1 (Tableau 2.3.3)



Graphique 2.3.4 – Interpolations de Newton et Neuville – Série 2 (Tableau 2.3.3)



Graphique 2.3.5 – Interpolations de Newton et Neuville – Série 3 (Tableau 2.3.3)

2.3.4 Dépenses et Revenus

x_i (R)	752	855	871	734	610	582	921	492	569	462	907
y_i (D)	85	83	162	79	81	83	281	81	81	80	243
x_i (R)	643	862	524	679	902	918	828	875	809	894	
y_i (D)	84	84	82	80	226	260	82	186	77	223	

Tableau 2.3.4 – Série non triée

x_i (R)	752	855	828	734	809	610	582	492	569	462	643	862	524	679
y_i (D)	85	83	82	79	77	81	83	81	81	80	84	84	82	80

Tableau 2.3.5 – Série triée : Partie Basse

x_i (R)	902	918	871	875	921	907	894
y_i (D)	226	260	162	186	281	243	223

Tableau 2.3.6 – Série triée : Partie Haute

On obtient alors :

Partie Basse :

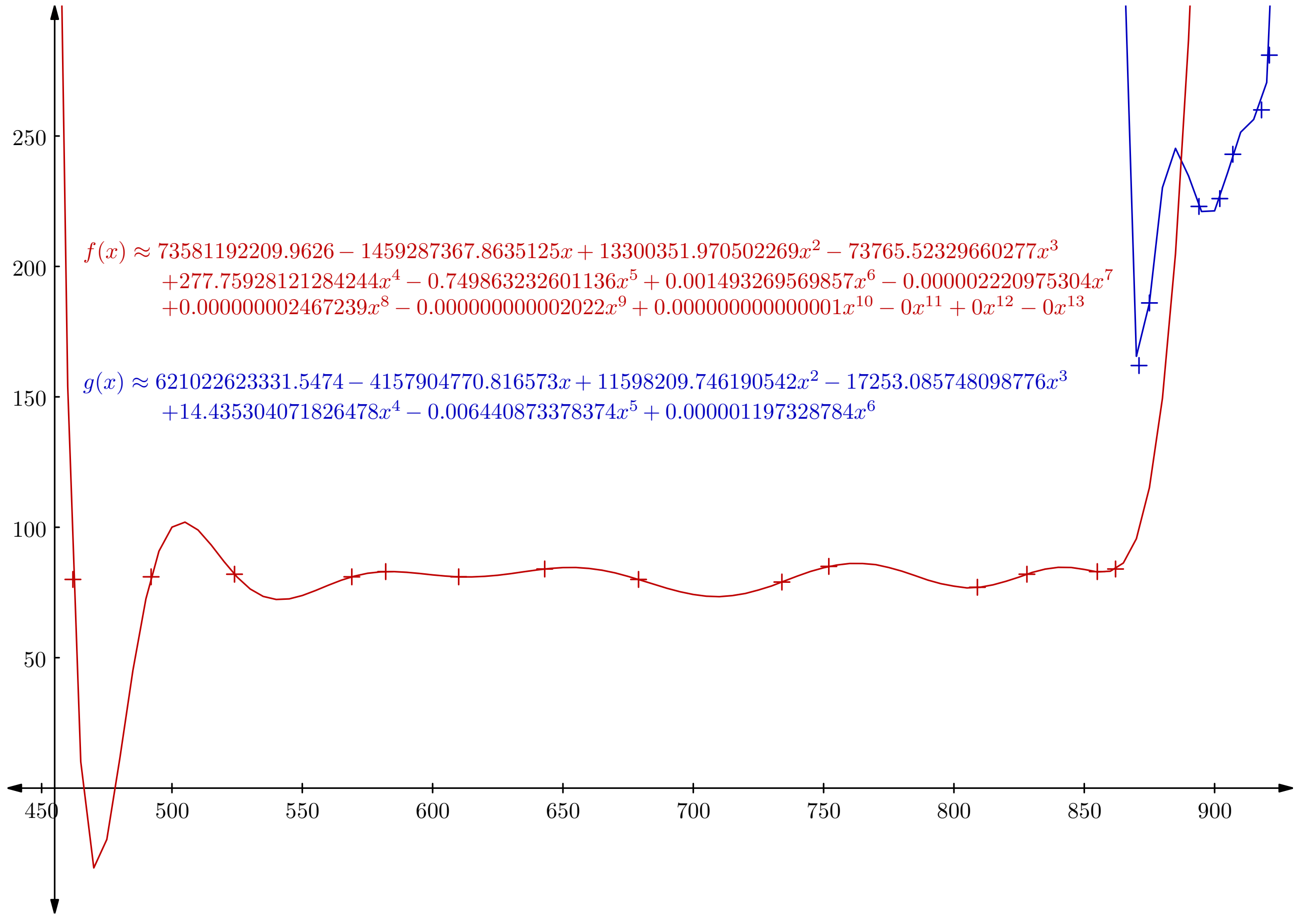
Méthode de Newton :
$P(x) \approx 73581192209.962601 - 1459287367.863513 \cdot x + 13300351.970502 \cdot x^2 - 73765.523297 \cdot x^3 + 277.759281 \cdot x^4 - 0.749863 \cdot x^5 + 0.001493 \cdot x^6 - 0.000002 \cdot x^7 + 0.000000 \cdot x^8 - 0.000000 \cdot x^9 + 0.000000 \cdot x^{10} - 0.000000 \cdot x^{11} + 0.000000 \cdot x^{12} - 0.000000 \cdot x^{13}$
Erreur : 0.018460432587224723

Méthode de Neuville :
$P(x) \approx 73581192209.952133 - 1459287367.863373 \cdot x + 13300351.970501 \cdot x^2 - 73765.523297 \cdot x^3 + 277.759281 \cdot x^4 - 0.749863 \cdot x^5 + 0.001493 \cdot x^6 - 0.000002 \cdot x^7 + 0.000000 \cdot x^8 - 0.000000 \cdot x^9 + 0.000000 \cdot x^{10} - 0.000000 \cdot x^{11} + 0.000000 \cdot x^{12} - 0.000000 \cdot x^{13}$
Erreur : 0.192499821369502971

Partie Haute :

Méthode de Newton :
$P(x) \approx 621022623331.547363 - 4157904770.816573 \cdot x + 11598209.746191 \cdot x^2 - 17253.085748 \cdot x^3 + 14.435304 \cdot x^4 - 0.006441 \cdot x^5 + 0.000001 \cdot x^6$
Erreur : 0.000429349286215646

Méthode de Neuville :
$P(x) \approx 621022623331.548340 - 4157904770.816572 \cdot x + 11598209.746191 \cdot x^2 - 17253.085748 \cdot x^3 + 14.435304 \cdot x^4 - 0.006441 \cdot x^5 + 0.000001 \cdot x^6$
Erreur : 0.002443850040435791



Graphique 2.3.6 – Interpolation de Newton et Neuville – (Tableaux 2.3.5 & 2.3.6)

2.4 Comparaison

3 Approximation

3.1 Régression linéaire

3.1.1 Présentation

3.1.2 Programme

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "polynome.h"

void mapping(double** from, double** to, int n, char* fn)
{
  int i, j;
  if (strcmp(fn,"exponentielle")==0)
  {
    for (j=0; j<n; j++)
    {
      to[0][j]=from[0][j];
    }
    for (j=0; j<n; j++)
    {
      to[1][j]=log(from[1][j]);
    }
  }
  else if (strcmp(fn,"puissance")==0)
  {
    for (i=0; i<2; i++)
    {
      for (j=0; j<n; j++)
      {
        to[i][j]=log(from[i][j]);
      }
    }
  }
}

double moyenneElements(double** tab,int l, int n)
{
  double resultat = 0.;
  double cpt = 0.;
  int i;
  for(i=0;i<n;i++)
  {
    resultat = resultat + tab[l][i];
    cpt = cpt + 1.;
  }
  resultat = resultat/cpt;
  return resultat;
}

double moyenneElementsCarres(double** tab,int l, int n)
{
  double resultat = 0;
```

```
  double cpt = 0;
  int i;
  for(i=0;i<n;i++)
  {
    resultat = resultat + pow(tab[1][i],2);
    cpt = cpt + 1;
  }
  resultat = resultat/cpt;
  return resultat;
}

double moyenneProduitElements(double** tab, int n)
{
  double resultat = 0;
  double cpt = 0;
  int i;
  for(i=0;i<n;i++)
  {
    resultat = resultat + tab[0][i]*tab[1][i];
    cpt = cpt + 1;
  }
  resultat = resultat/cpt;
  return resultat;
}

reglinD(double** tab, int n)
{
  double a0 = 0;
  double a1 = 0;
  double xb, yb, xcb, xyb; // b pour barre et c pour carre
  printf("Nous cherchons le polynome de degré 1 sous la forme a0 + a1*x.\n");

  //calculs
  xb = moyenneElements(tab,0,n);
  yb = moyenneElements(tab,1,n);
  xcb = moyenneElementsCarres(tab,0,n);
  xyb = moyenneProduitElements(tab,n);

  a1 = (xyb-xb*yb)/(xcb-pow(xb,2));
  a0 = yb-xb*a1;

  // creation et affichage du polynome
  polynome *P = creerPoly(2,"valeur",a0,a1);
  menuAffichage(P);

  //statistiques
  ecartPoly(tab,n,P);
  printf("\n");

  //libération mémoire
  free(P->poln);
  free(P);
}

reglinE(double** tab, int n) //y=c(e^(dx)) <=> ln(y)=ln(c)+xd => c=e^(a0) & d=a1
{
  int i;
  double c = 0;
  double d = 0;
  double a0 = 0;
  double a1 = 0;
  double xb, yb, xcb, xyb; // b pour barre et c pour carre
  double** t = (double**) malloc(2*sizeof(double*)); // contiendra le mapping de tab
  for(i=0;i<2;i++)
  {
    t[i] = (double*) malloc (n*sizeof(double));
  }
  printf("Nous cherchons une approximation sous la forme c*(e^(d*x)).\n");

  //calculs
  mapping(tab, t, n, "exponentielle");
```

```c
  xb = moyenneElements(t,0,n);
  yb = moyenneElements(t,1,n);
  xcb = moyenneElementsCarres(t,0,n);
  xyb = moyenneProduitElements(t,n);

  a1 = (xyb-xb*yb)/(xcb-pow(xb,2.));
  a0 = yb-xb*a1;
  d = a1;
  c = exp(a0);

  //affichage
  printf("P(x) = %20.18f*exp(%20.18f*x)\n",c,d);

  //statistiques
  ecartExpo(tab,n,c,d);
  printf("\n");

  //libération mémoire
  for (i=0; i<2; i++)
  {
    free(t[i]);
  }
  free(t);
}

reglinP(double ** tab, int n) //y=a(x^b) <=> ln(y)=ln(a)+b*ln(x) => a=e^(a0) & b=a1
{
  int i;
  double a = 0.;
  double b = 0.;
  double a0 = 0.;
  double a1 = 0.;
  double xb, yb, xcb, xyb; // b pour barre et c pour carre
  double** t = (double**) malloc(2*sizeof(double*)); // contiendra le mapping de tab
  for(i=0;i<2;i++)
  {
    t[i] = (double*) malloc (n*sizeof(double));
  }
  printf("Nous cherchons une approximation sous la forme a*(x^(b)).\n");

  //calculs
  mapping(tab, t, n, "puissance");
  xb = moyenneElements(t,0,n);
  yb = moyenneElements(t,1,n);
  xcb = moyenneElementsCarres(t,0,n);
  xyb = moyenneProduitElements(t,n);

  a1 = (xyb-xb*yb)/(xcb-pow(xb,2));
  a0 = yb-xb*a1;
  b = a1;
  a = exp(a0);

  //affichage
  printf("P(x) = %20.18f*x^(%20.18f)\n",a,b);

  //statistiques
  ecartPui(tab,n,a,b);
  printf("\n");

  //libération mémoire
  for (i=0; i<2; i++)
  {
    free(t[i]);
  }
  free(t);
}
```

FIGURE 3.1 – Code : reglin.c

3.2 Ajustement exponentiel

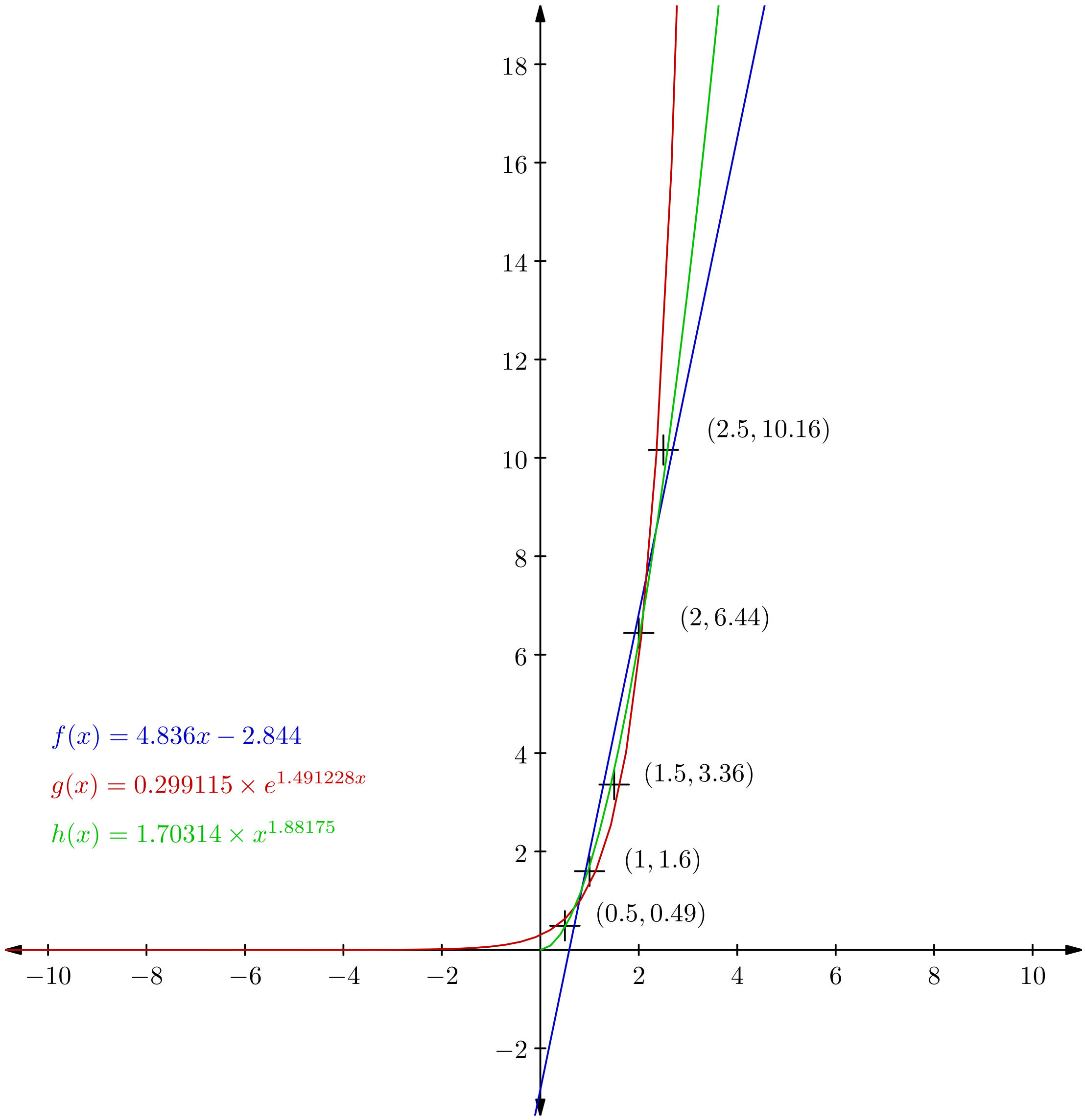
3.3 Ajustement de type “puissance”

3.4 Résultats de tests

3.4.1 Exemple tiré d'un TD

x_i	0.5	1	1.5	2	2.5
y_i	0.49	1.6	3.36	6.44	10.16

TABLEAU 3.4.1 – Série 1

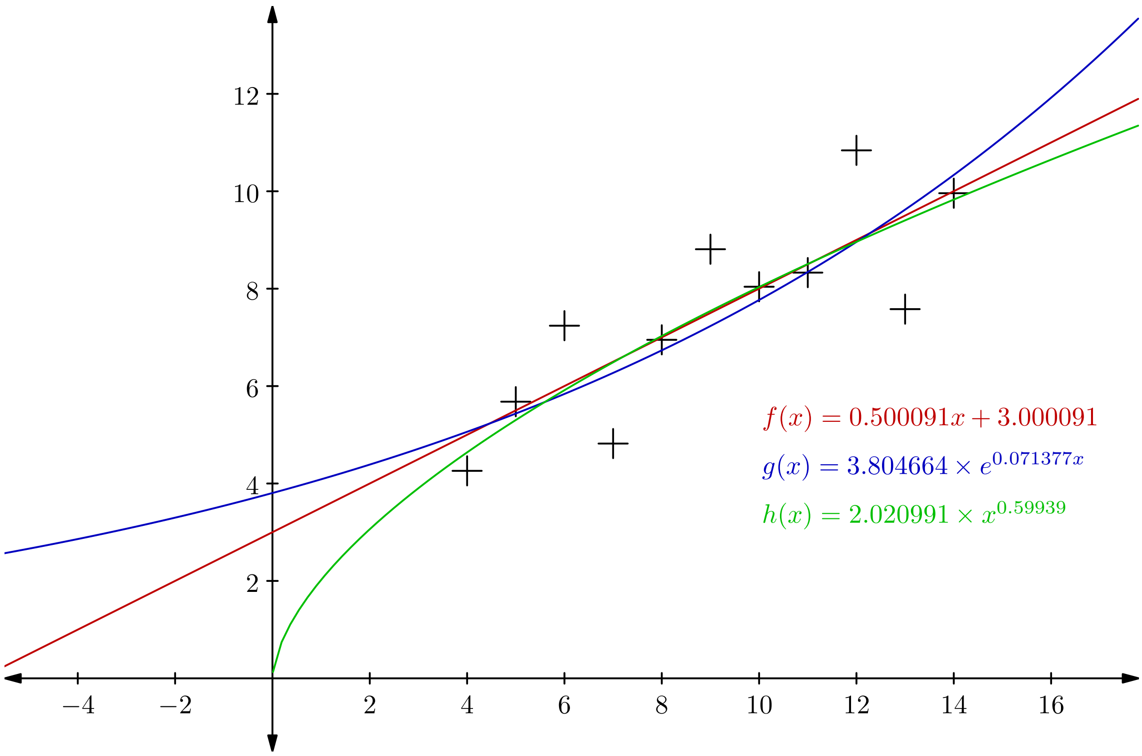


GRAPHIQUE 3.4.1 – Régression linéaire – (Tableau 3.4.1)

3.4.2 Série d'Anscombe

x_i	10	8	13	9	11	14	6	4	12	7	5
$y_i^{(A)}$	8.04	6.95	7.58	8.81	8.33	9.96	7.24	4.26	10.84	4.82	5.68

TABLEAU 3.4.2 – Série dûe à Anscombe



GRAPHIQUE 3.4.2 – Régression linéaire – (Tableau 3.4.2)

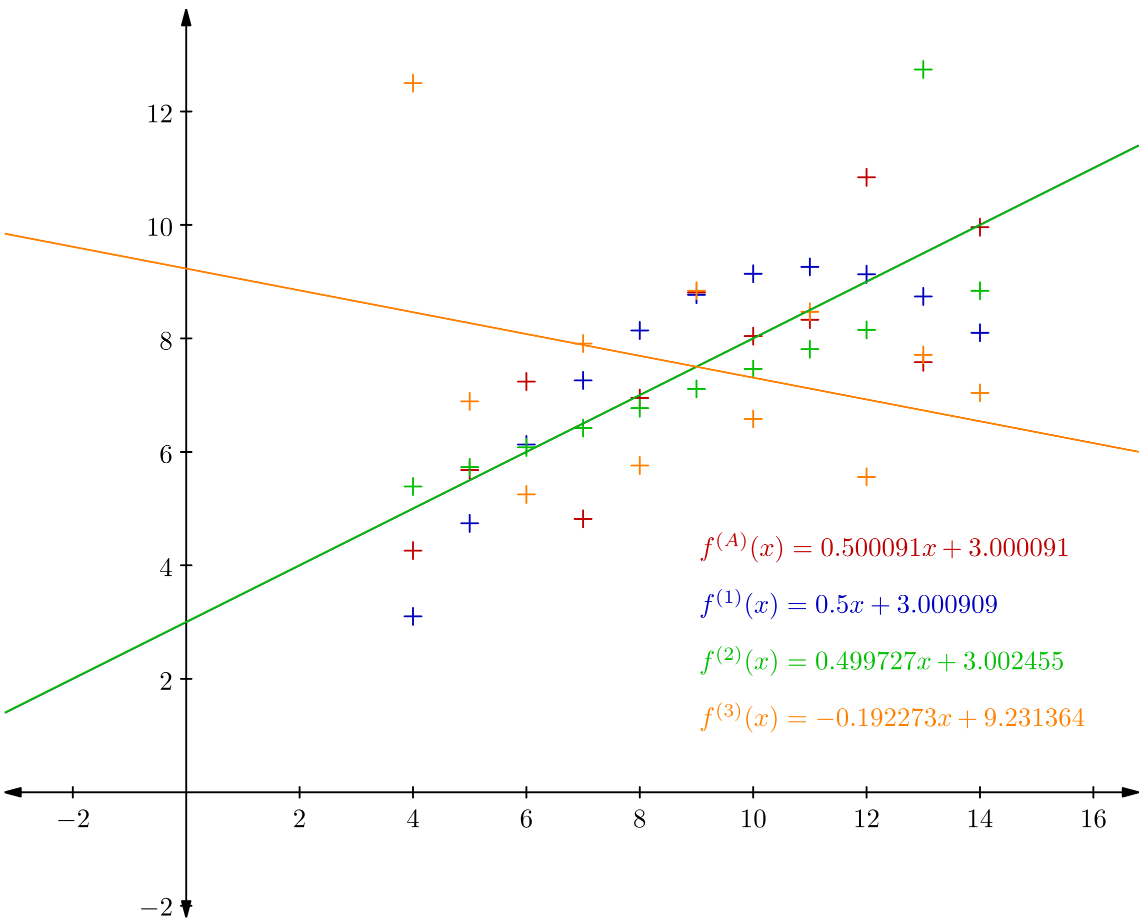
3.4.3 3 séries

x_i	10	8	13	9	11	14	6	4	12	7	5
$y_i^{(1)}$	9.14	8.14	8.74	8.77	9.26	8.10	6.13	3.10	9.13	7.26	4.74
$y_i^{(2)}$	7.46	6.77	12.74	7.11	7.81	8.84	6.08	5.39	8.15	6.42	5.73
$y_i^{(3)}$	6.58	5.76	7.71	8.84	8.47	7.04	5.25	12.50	5.56	7.91	6.89
$y_i^{(A)}$	8.04	6.95	7.58	8.81	8.33	9.96	7.24	4.26	10.84	4.82	5.68

TABLEAU 3.4.3 – 3 séries $S^{(1)}$, $S^{(2)}$ et $S^{(3)}$ comparées à Anscombe

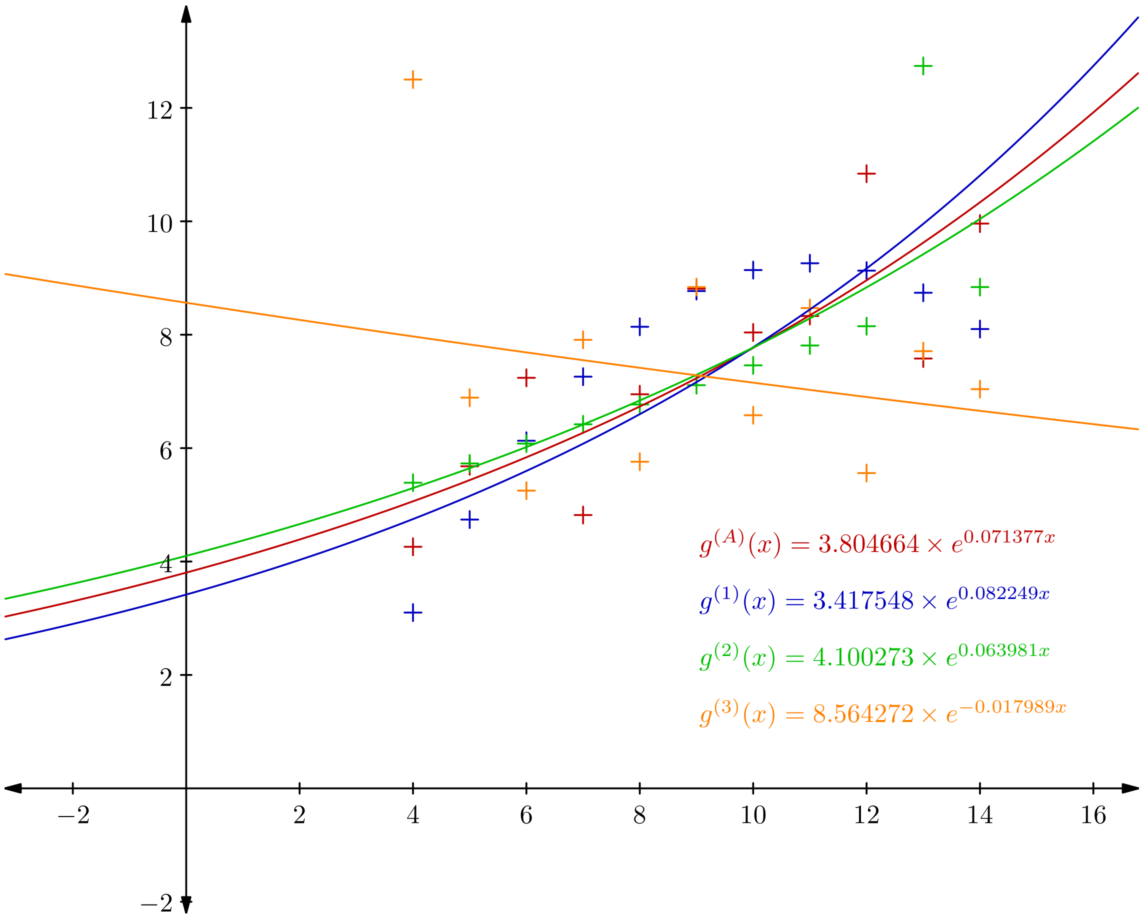
Série (1) :
Regression linéaire par une droite : $P(x) = 3.000909 + 0.500000 \cdot x$
Erreur moyenne : 0.967934

Série (2) :
Regression linéaire par une droite : $P(x) = 3.002455 + 0.499727 \cdot x$
Erreur moyenne : 0.715967



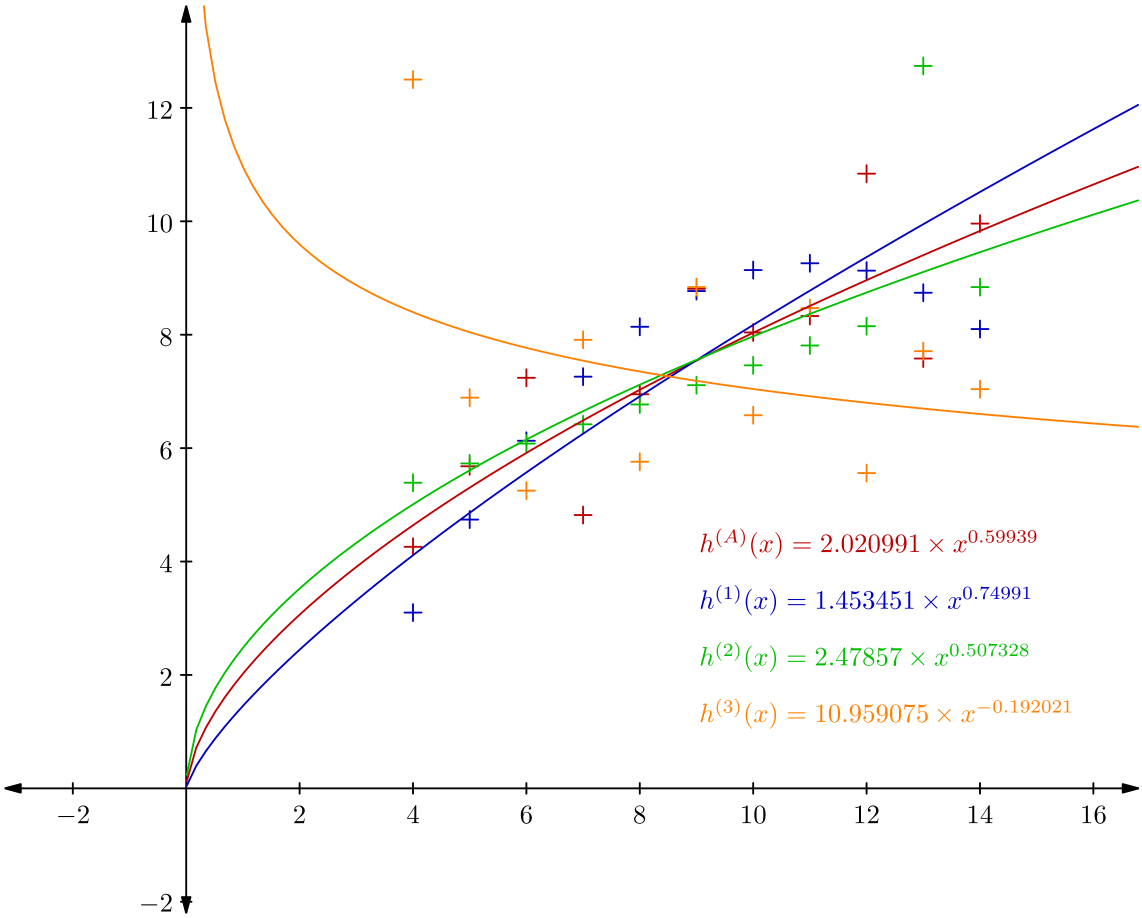
GRAPHIQUE 3.4.3 – Régression linéaire – (Tableau 3.4.3)

Regression linéaire par une exponentielle :$P(x) = 3.417548 \cdot \exp(0.082249 \cdot x)$
Erreur moyenne : 1.187786
Regression linéaire par une exponentielle :$P(x) = 4.100273 \cdot \exp(0.063981 \cdot x)$
Erreur moyenne : 0.590601



GRAPHIQUE 3.4.4 – Approximation par ajustement exponentiel – (Tableau 3.4.3)

Regression linéaire par une puissance : $P(x) = 1.453451 \cdot x^{0.749910}$
Erreur moyenne : 0.950634
Regression linéaire par une puissance :$P(x) = 2.478570 \cdot x^{0.507328}$
Erreur moyenne : 0.682932



GRAPHIQUE 3.4.5 – Approximation par ajustement "puissance" – (Tableau 3.4.3)

3.4.4 Dépenses mensuelles et revenus

x_i (R)	752	855	871	734	610	582	921	492	569	462	907
y_i (D)	85	83	162	79	81	83	281	81	81	80	243

TABLEAU 3.4.4 – Série 1

x_i (R)	643	862	524	679	902	918	828	875	809	894
y_i (D)	84	84	82	80	226	260	82	186	77	223

TABLEAU 3.4.5 – Série 2

Série 1 :
Regression linéaire par une droite :$P(x) = -98.368005 + 0.312192 \cdot x$
Erreur moyenne : 38.488186

Regression linéaire par une exponentielle :$P(x) = 24.011644 \cdot \exp(0.002124 \cdot x)$
Erreur moyenne : 33.486916

Regression linéaire par une puissance :$P(x) = 0.015258 \cdot x^{1.356482}$
Erreur moyenne : 36.660388

Série 2 :
Regression linéaire par une droite :$P(x) = -164.266162 + 0.381480 \cdot x$
Erreur moyenne : 46.455702

Regression linéaire par une exponentielle :$P(x) = 14.780629 \cdot \exp(0.002657 \cdot x)$
Erreur moyenne : 45.099159

Regression linéaire par une puissance :$P(x) = 0.000785 \cdot x^{1.793989}$
Erreur moyenne : 47.682118

3.4.5 Série chronologique avec accroissement exponentiel

x_i	88	89	90	91	92	93	94	95	96	97
y_i	5.89	6.77	7.97	9.11	10.56	12.27	13.92	15.72	17.91	22.13

Tableau 3.4.6 – Série

3.4.6 Vérification de la loi de Pareto

x_i	20	30	40	50	100	300	500
y_i	352	128	62.3	35.7	6.3	0.4	0.1

TABLEAU 3.4.7 – Relation entre revenu et nombre de personnes ayant un revenu supérieur

4 Conclusion