érie 12.3.510 érie 22.3.611 érie 1 et 22.3.712 érie 13.2.519 érie 23.2.619 érie 1 et 23.2.719

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 par une droite
4- Régression Linéaire par une fonction exponentielle
5- Régression Linéaire par une fonction puissance
9- Nouvelle série de points (Menu principal)
0- Quitter
Votre choix :
```

Figure 1.1 – Apercu : Menu Principal

2 Interpolation

2.1 Méthode de Newton

2.1.1 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++)
  {
    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 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);
}
```

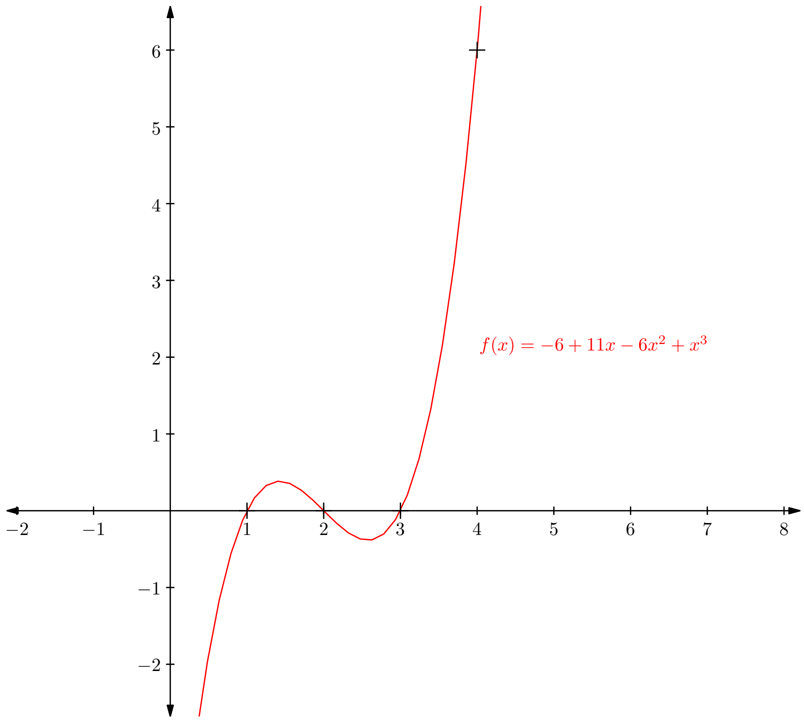
FIGURE 2.2 – Code : neuville.c

2.3 Résultats de tests

x_i	1	2	3	4
y_i	0	0	0	6

TABLEAU 2.3.1 – Série 1

Méthode de Newton : $P(x) = -6.000000 + 11.000000 \cdot x - 6.000000 \cdot x^2 + 1.000000 \cdot x^3$
Méthode de Neuville : $P(x) = -6.000000 + 11.000000 \cdot x - 6.000000 \cdot x^2 + 1.000000 \cdot x^3$



GRAPHIQUE 2.3.1 – Interpolation de Newton et Neuville – (Tableau 2.3.1)

x_i	0.000000	2.000000	4.000000	6.000000	8.000000	10.000000	12.000000	14.000000	16.000000	18.00000
y_i	0.999870	0.999970	1.000000	0.999970	0.999880	0.999730	0.999530	0.999530	0.998970	0.99846

Tableau 2.3.2 – 3.1 Densité de l'eau en fonction de la température

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

Erreur moyenne : 0.000005

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

Erreur moyenne : 0.000033

x_i	10.000000	8.000000	13.000000	9.000000	11.000000	14.000000	6.000000	4.000000	12.000000	7.00000
y_i	9.140000	8.140000	8.740000	8.770000	9.260000	8.100000	6.130000	3.100000	9.130000	7.26000

Tableau 2.3.3 – 3.2.1 Trois séries

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

Erreur moyenne : 0.000000

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

Erreur moyenne : 0.000000

x_i	10.000000	8.000000	13.000000	9.000000	11.000000	14.000000	6.000000	4.000000	12.000000	7.00000
y_i	7.460000	6.770000	12.740000	7.110000	7.810000	8.840000	6.080000	5.390000	8.150000	6.42000

TABLEAU 2.3.4 – 3.2.2 Trois séries

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

Erreur moyenne : 0.000000

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

Erreur moyenne : 0.000000

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

Tableau 2.3.5 – Série1 dépense

Méthode de Newton : $P(x) = 524166663.148073 - 8065079.743259 \cdot x + 55479.774423 \cdot x^2 - 224.689243 \cdot x^3 + 0.593281 \cdot x^4 - 0.001067 \cdot x^5 + 0.000001 \cdot x^6 - 0.000000 \cdot x^7 + 0.000000 \cdot x^8 - 0.000000 \cdot x^9 + 0.000000 \cdot x^{10}$

Erreur moyenne : 0.000046

Méthode de Neuville : $P(x) = 524166663.148078 - 8065079.743259 \cdot x + 55479.774423 \cdot x^2 - 224.689243 \cdot x^3 + 0.593281 \cdot x^4 - 0.001067 \cdot x^5 + 0.000001 \cdot x^6 - 0.000000 \cdot x^7 + 0.000000 \cdot x^8 - 0.000000 \cdot x^9 + 0.000000 \cdot x^{10}$

Erreur moyenne : 0.000043

x_i	643.000000	862.000000	524.000000	679.000000	902.000000	918.000000	828.000000	875.000000	809.00
y_i	84.000000	84.000000	82.000000	80.000000	226.000000	260.000000	82.000000	186.000000	77.000

Tableau 2.3.6 – Série2 dépense

Méthode de Newton : $P(x) = 264070485740.100494 - 3124699358.171593 \cdot x + 16351348.965791 \cdot x^2 - 49678.667726 \cdot x^3 + 96.596340 \cdot x^4 - 0.124686 \cdot x^5 + 0.000107 \cdot x^6 - 0.000000 \cdot x^7 + 0.000000 \cdot x^8 - 0.000000 \cdot x^9$

Erreur moyenne : 0.005505

Méthode de Neuville : $P(x) = 264070485740.100800 - 3124699358.171597 \cdot x + 16351348.965791 \cdot x^2 - 49678.667726 \cdot x^3 + 96.596340 \cdot x^4 - 0.124686 \cdot x^5 + 0.000107 \cdot x^6 - 0.000000 \cdot x^7 + 0.000000 \cdot x^8 - 0.000000 \cdot x^9$

Erreur moyenne : 0.005585

x_i	752.000000	855.000000	871.000000	734.000000	610.000000	582.000000	921.000000	492.000000	569.00
y_i	85.000000	83.000000	162.000000	79.000000	81.000000	83.000000	281.000000	81.000000	81.000

TABLEAU 2.3.7 – Série1-2 dépense

Méthode de Newton : $P(x) = 245839213382200524800.000000 - 6969795463389496320.000000 \cdot x + 93583577551725008.0000$ $x^2 - 791298544244389.125000 \cdot x^3 + 4725727913097.273438 \cdot x^4 - 21189691764.383732 \cdot x^5 + 74020805.461603 \cdot x^6 - 206287.504897 \cdot x^7 + 465.835063 \cdot x^8 - 0.860816 \cdot x^9 + 0.001309 \cdot x^{10} - 0.000002 \cdot x^{11} + 0.000000 \cdot x^{12} - 0.000000 \cdot x^{13} + 0.000000 \cdot x^{14} - 0.000000 \cdot x^{15} + 0.000000 \cdot x^{16} - 0.000000 \cdot x^{17} + 0.000000 \cdot x^{18} - 0.000000 \cdot x^{19} + 0.000000 \cdot x^{20}$

Erreur moyenne : 21268371516.333332

Méthode de Neuville : $P(x) = 245839213382189940736.000000 - 6969795463389208576.000000 \cdot x + 93583577551721264.000$ $x^2 - 791298544244361.000000 \cdot x^3 + 4725727913097.125000 \cdot x^4 - 21189691764.383118 \cdot x^5 + 74020805.461601 \cdot x^6 - 206287.504897 \cdot x^7 + 465.835063 \cdot x^8 - 0.860816 \cdot x^9 + 0.001309 \cdot x^{10} - 0.000002 \cdot x^{11} + 0.000000 \cdot x^{12} - 0.000000 \cdot x^{13} + 0.000000 \cdot x^{14} - 0.000000 \cdot x^{15} + 0.000000 \cdot x^{16} - 0.000000 \cdot x^{17} + 0.000000 \cdot x^{18} - 0.000000 \cdot x^{19} + 0.000000 \cdot x^{20}$

Erreur moyenne : 27563644709.000000

2.4 Comparaison

3 Approximation

3.1 Régression linéaire

3.1.1 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[l][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");
  xb = moyenneElements(t,0,n);
  yb = moyenneElements(t,1,n);
```

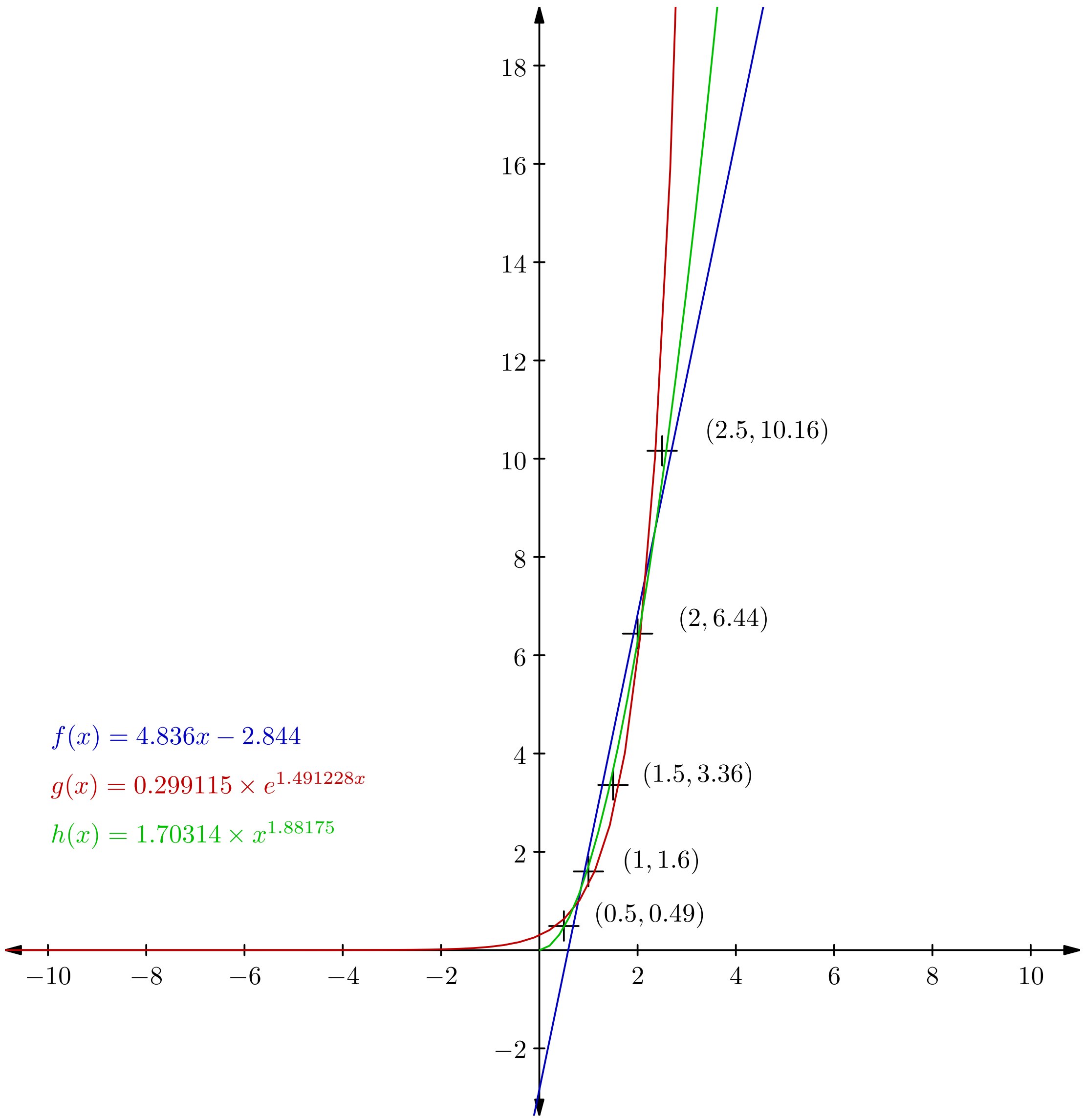
```c
  xcb = moyenneElementsCarres(t,0,n);
  xyb = moyenneProduitElements(t,n);

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

  //affichage
  printf("P(x) = %f*exp(%f*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) = %f*x^(%f)\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 Résultats de tests

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

TABLEAU 3.2.1 – Série 1



GRAPHIQUE 3.2.1 – Régression linéaire – (Tableau 3.2.1)

Regression linéaire par une droite : $P(x) = 1.001302 - 0.000186 \cdot x$
Erreur moyenne : 0.000665
Regression linéaire par une exponentielle : $P(x) = 1.001308 \cdot \exp(-0.000187 \cdot x)$
Erreur moyenne : 0.000667
Regression linéaire par une puissance : Pas de résultat, lors du changement de variable, on calcule $\log_2(0)$ qui n'est pas défini.
Regression linéaire par une droite : $P(x) = 3.000909 + 0.500000 \cdot x$
Erreur moyenne : 0.967934
Regression linéaire par une exponentielle :$P(x) = 3.417548 \cdot \exp(0.082249 \cdot x)$

x_i	0.000000	2.000000	4.000000	6.000000	8.000000	10.000000	12.000000	14.000000	16.000000	18.00000
y_i	0.999870	0.999970	1.000000	0.999970	0.999880	0.999730	0.999530	0.999530	0.998970	0.99846

TABLEAU 3.2.2 – 3.1 Densité de l'eau en fonction de la température

x_i	10.000000	8.000000	13.000000	9.000000	11.000000	14.000000	6.000000	4.000000	12.000000	7.00000
y_i	9.140000	8.140000	8.740000	8.770000	9.260000	8.100000	6.130000	3.100000	9.130000	7.26000

TABLEAU 3.2.3 – 3.2.1 Trois séries

Erreur moyenne : 1.187786
Regression linéaire par une puissance : $P(x) = 1.453451 \cdot x^{0.749910}$
Erreur moyenne : 0.950634

x_i	10.000000	8.000000	13.000000	9.000000	11.000000	14.000000	6.000000	4.000000	12.000000	7.00000
y_i	7.460000	6.770000	12.740000	7.110000	7.810000	8.840000	6.080000	5.390000	8.150000	6.42000

TABLEAU 3.2.4 – 3.2.2 Trois séries

Regression linéaire par une droite : $P(x) = 3.002455 + 0.499727 \cdot x$
Erreur moyenne : 0.715967
Regression linéaire par une exponentielle :$P(x) = 4.100273 \cdot \exp(0.063981 \cdot x)$
Erreur moyenne : 0.590601
Regression linéaire par une puissance :$P(x) = 2.478570 \cdot x^{0.507328}$
Erreur moyenne : 0.682932
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
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
Regression linéaire par une droite :$P(x) = -112.658491 + 0.324356 \cdot x$
Erreur moyenne : 43.378231
Reression linéaire par une exponentielle :$P(x) = 21.399929 \cdot \exp(0.002238 \cdot x)$
Erreur moyenne : 40.094599
Regression linéaire par une puissance :$P(x) = 0.007872 \cdot x^{1.453059}$
Erreur moyenne : 42.696155

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

Tableau 3.2.5 – Série1 dépense

x_i	643.000000	862.000000	524.000000	679.000000	902.000000	918.000000	828.000000	875.000000	809.000
y_i	84.000000	84.000000	82.000000	80.000000	226.000000	260.000000	82.000000	186.000000	77.000

Tableau 3.2.6 – Série2 dépense

x_i	752.000000	855.000000	871.000000	734.000000	610.000000	582.000000	921.000000	492.000000	569.000
y_i	85.000000	83.000000	162.000000	79.000000	81.000000	83.000000	281.000000	81.000000	81.000

Tableau 3.2.7 – Série1-2 dépense

4 Conclusion