TP n°5
Histogramme et densité de probabilité d'une d'image, mélange de gaussiennes, algorithme EM

Partie 1

Image de base :

Image seuillée à 150 :

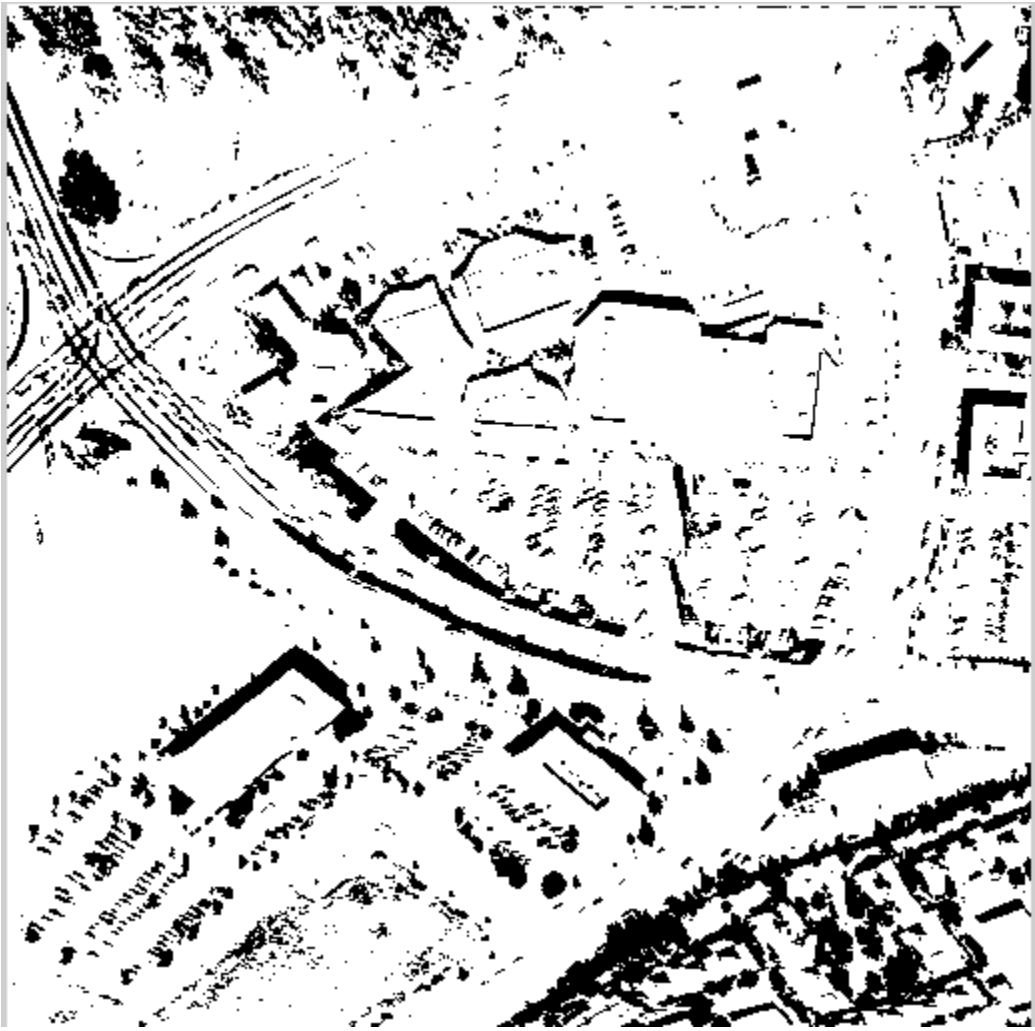
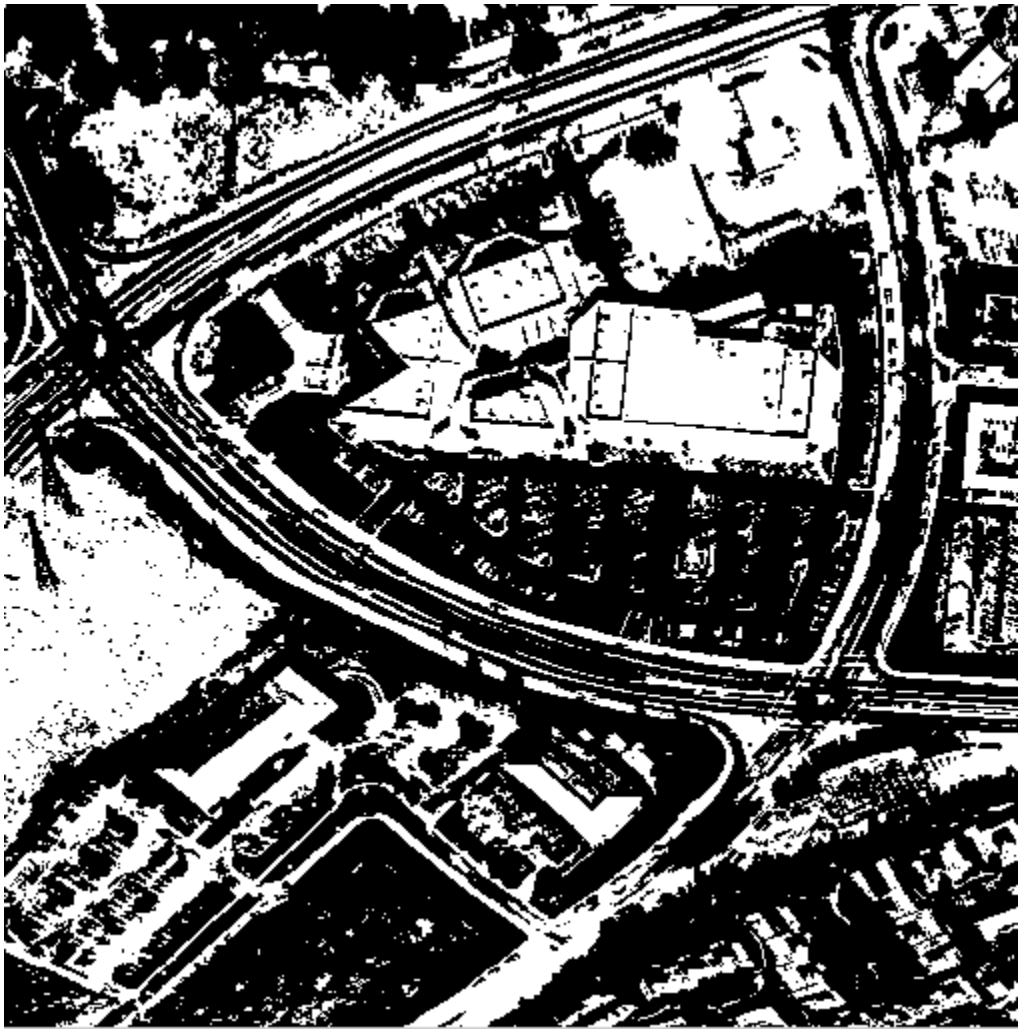


Image seuillée à 200 :



On remarque que plus on seuille une image avec des valeurs hautes, et plus celle-ci devient sombre (on perd en précision de couleurs car on en conserve que 2).

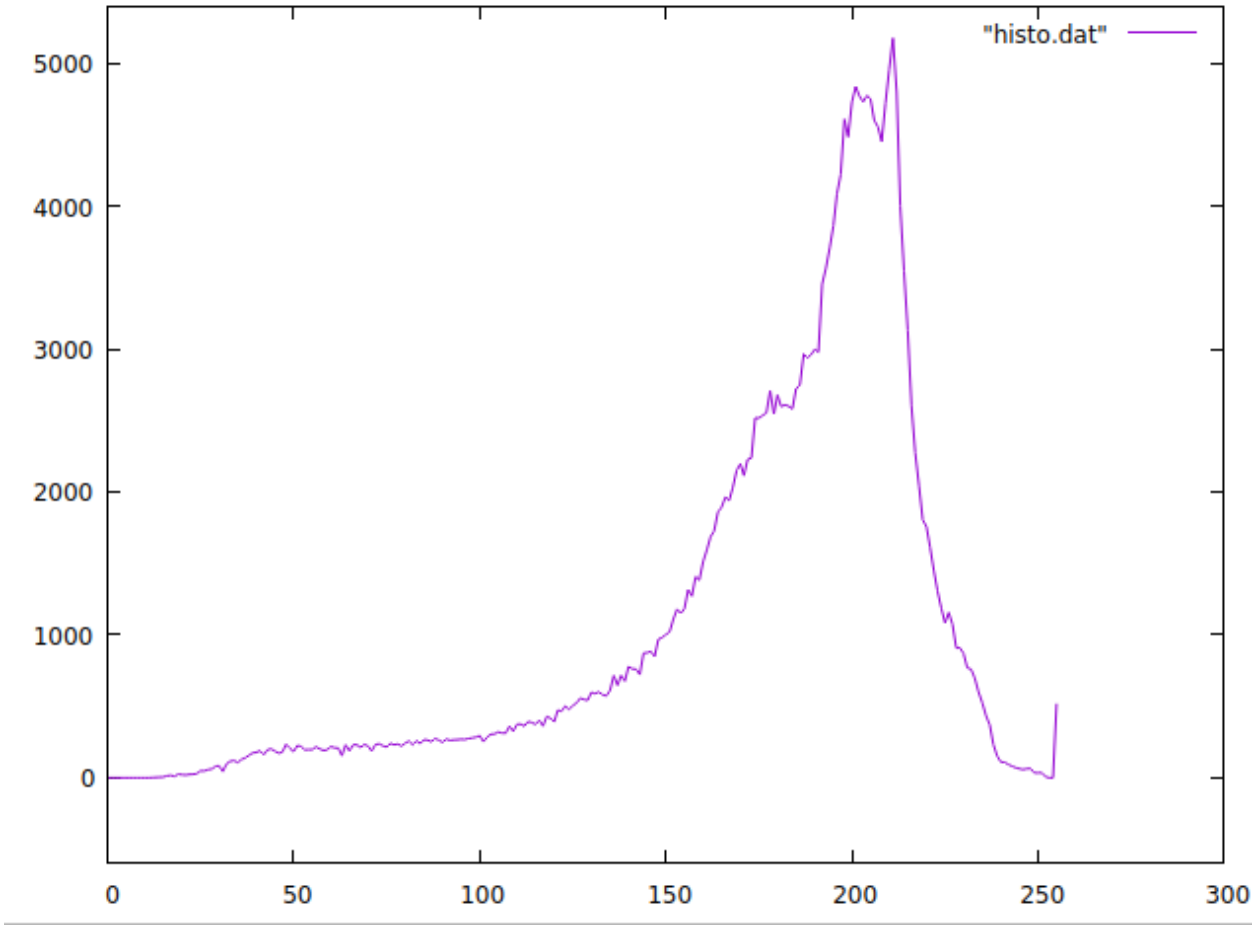
Partie 2

Histogramme de l’image de base

```
//Fichier permettant de sauvegarder l'histogramme d'une image données dans un fichier correspondant
#include <stdio.h>
#include "image_ppm.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
  char cNomImgLue[250], cNomHisto[250];
  int nH, nW, nTaille;


  if (argc != 3){
       printf("Usage: ImageIn.pgm HISTO\n");
       exit (1) ;
  }

   sscanf (argv[1],"%s",cNomImgLue);
   sscanf (argv[2],"%s",cNomHisto);


   OCTET *ImgIn;

   lire_nb_lignes_colonnes_image_pgm(cNomImgLue, &nH, &nW);
   nTaille = nH * nW;

   allocation_tableau(ImgIn, OCTET, nTaille);
   lire_image_pgm(cNomImgLue, ImgIn, nH * nW);


    int* nbElementsLus = new int[256]{};

    for (int i=0; i < nH; i++){
        for (int j=0; j < nW; j++){
            nbElementsLus[ImgIn[i*nW+j]]++;
        }
    }


    //Ecriture du fichier .bat
    for(int i = 0; i<256; i++){
        cout << nbElementsLus[i] << endl;
    }

    ofstream monFlux(cNomHisto);

    if(monFlux){
        for(int i = 0; i<256; i++){
```

```
            monFlux << i << " " << nbElementsLus[i] << endl;
        }
    }
    else
    {
        cout << "ERREUR: Impossible d'ouvrir le fichier." << endl;
    }
   free(ImgIn);
   return 1;
}
```

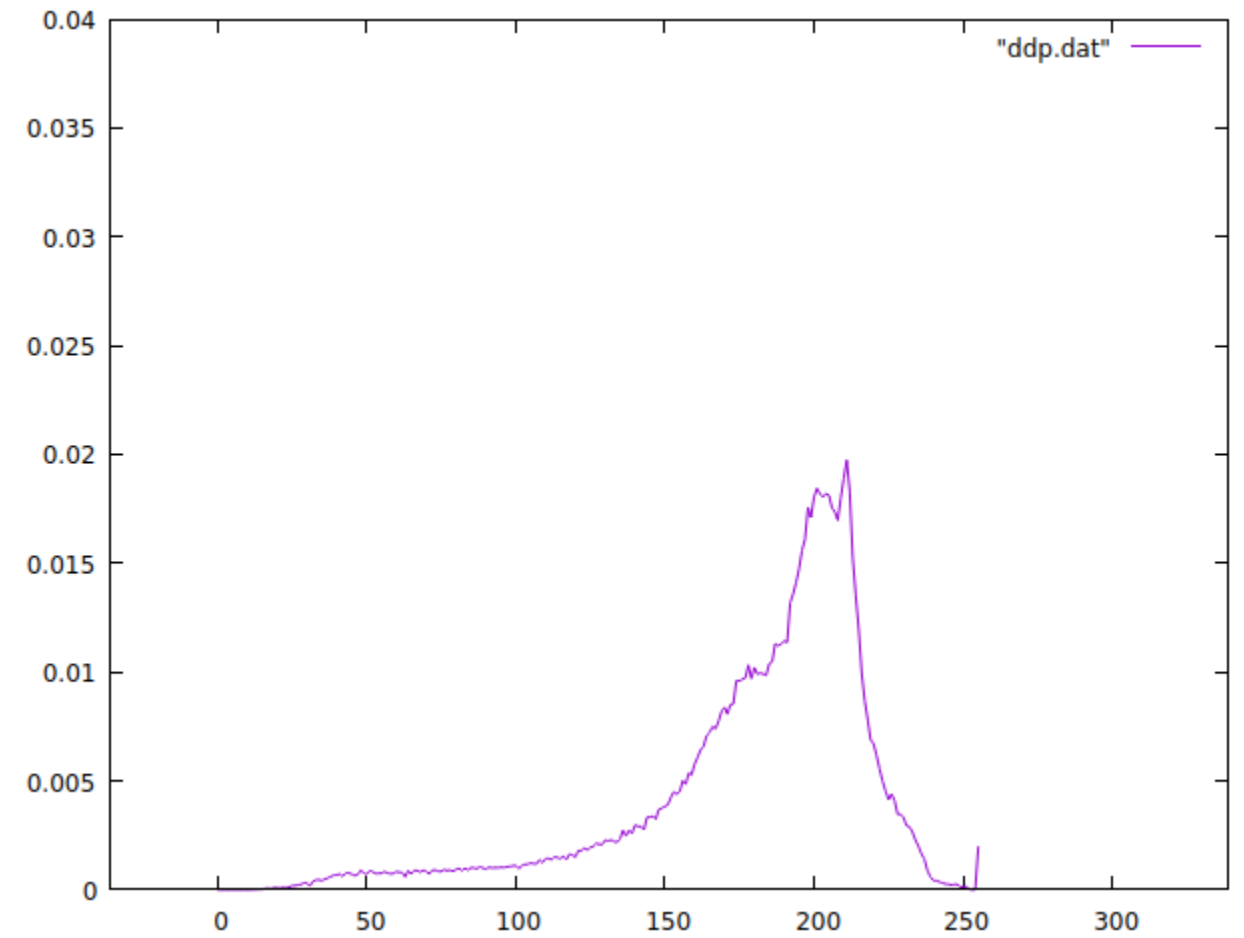
>plot “histo.dat” with lines



DDP de l’image de base

```
//Fichier permettant de sauvegarder l'histogramme d'une image données dans un fichier correspondant
#include <stdio.h>
#include "image_ppm.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
  char cNomImgLue[250], cNomDDP[250];
  int nH, nW, nTaille;


  if (argc != 3){
       printf("Usage: ImageIn.pgm HISTO\n");
       exit (1) ;
  }

   sscanf (argv[1],"%s",cNomImgLue);
   sscanf (argv[2],"%s",cNomDDP);


   OCTET *ImgIn;

   lire_nb_lignes_colonnes_image_pgm(cNomImgLue, &nH, &nW);
   nTaille = nH * nW;

   allocation_tableau(ImgIn, OCTET, nTaille);
   lire_image_pgm(cNomImgLue, ImgIn, nH * nW);


    double* nbElementsLus = new double[256]{};

    for (int i=0; i < nH; i++){
        for (int j=0; j < nW; j++){
            nbElementsLus[ImgIn[i*nW+j]]++;
        }
    }

    for (int i=0; i < 256; i++){
        nbElementsLus[i] /= (double) nTaille;
    }
    //Ecriture du fichier .bat
    for(int i = 0; i<256; i++){
        cout << nbElementsLus[i] << endl;
    }

    ofstream monFlux(cNomDDP);

    if(monFlux){
        for(int i = 0; i<256; i++){
           monFlux << i << " " << nbElementsLus[i] << endl;
        }
```

```
    }
    else
    {
        cout << "ERREUR: Impossible d'ouvrir le fichier." << endl;
    }

   free(ImgIn);
   return 1;
}
```



On se rend compte qu'il s'agit de la même courbe mais avec des valeurs ne pouvant aller qu'à 1 maximum (plage de valeurs en "y" plus faible).

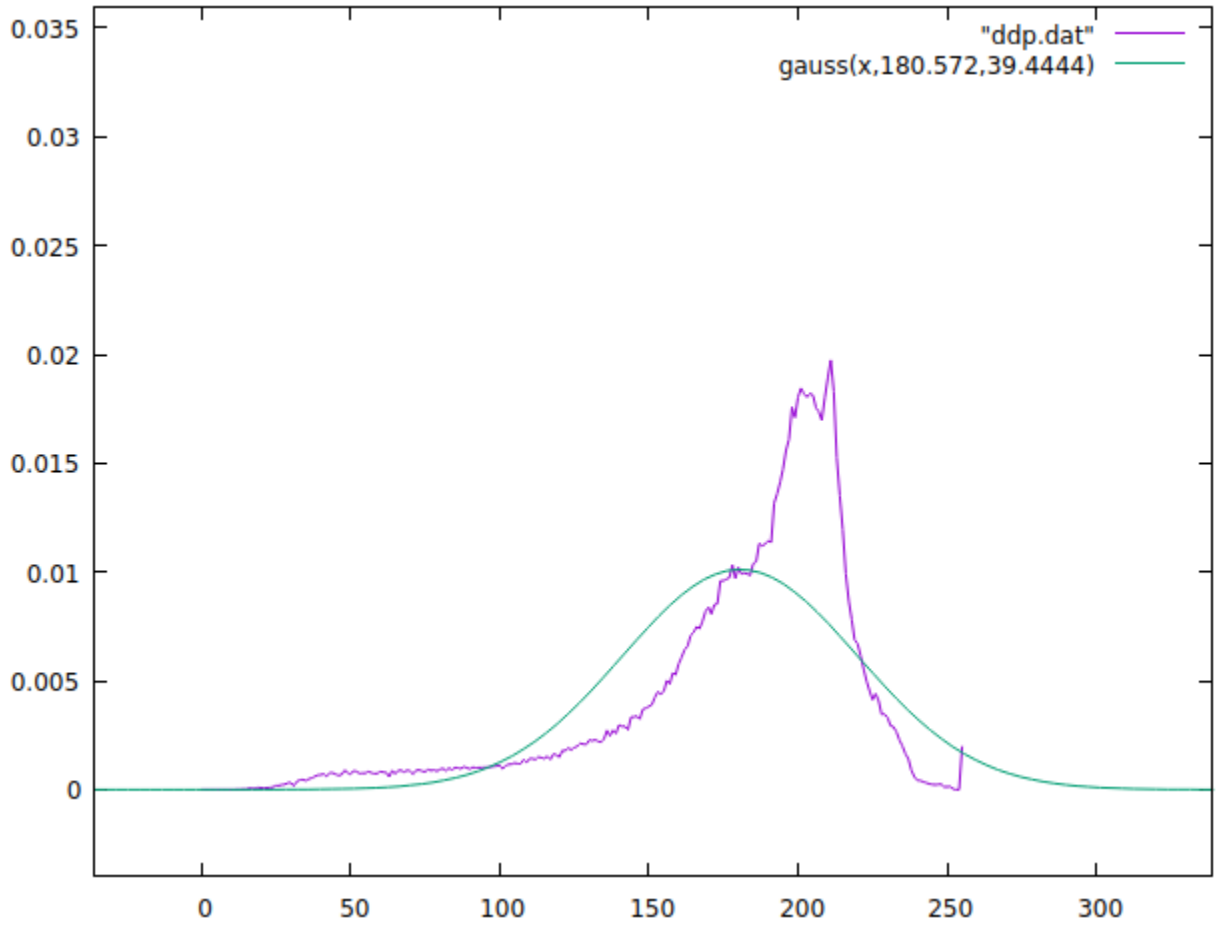
En effet, la somme de la totalité des probabilités de répartition vaut 1. La courbe de la DDP à l'air de ne former qu'un seul pic important, entre 150 et 230, qui correspond aux niveaux de gris les plus présents dans l'image.

Partie 3

```
//Fichier permettant de connaître la moyenne et la variance (et écart type) de la couleur d'une image en
//NDG
#include <stdio.h>
#include "image_ppm.h"
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>

using namespace std;

int main(int argc, char* argv[])
{
  char cNomImgLue[250];
  int nH, nW, nTaille;


  if (argc != 2){
       printf("Usage: ImageIn.pgm\n");
       exit (1) ;
  }

   sscanf (argv[1],"%s",cNomImgLue);

   OCTET *ImgIn;

   lire_nb_lignes_colonnes_image_pgm(cNomImgLue, &nH, &nW);
   nTaille = nH * nW;

   allocation_tableau(ImgIn, OCTET, nTaille);
   lire_image_pgm(cNomImgLue, ImgIn, nH * nW);


    double* nbElementsLus = new double[256]{};


    for (int i=0; i < nH; i++){
        for (int j=0; j < nW; j++){
            nbElementsLus[ImgIn[i*nW+j]]++;
        }
    }

    double moyenne = 0.;
    for(int i = 0; i<256;i++){
        moyenne += (i*nbElementsLus[i]);
    }
    moyenne/=(double)nTaille;

    cout << "MOYENNE : " << moyenne << endl;

    double variance = 0.;
    for(int i = 0; i<256;i++){
        variance += (i*i*nbElementsLus[i]);
    }
```

```
    variance = (variance/(double)nTaille) - (moyenne*moyenne);
    cout << "VARIANCE : " << variance << endl;
    cout << "ECART TYPE : " << sqrt(variance) << endl;

   free(ImgIn);
   return 1;
}
```

En traçant la DPP et la gaussienne, on obtient le graphique suivant :
> gauss(x,mu,sigma)=1/(sigma*sqrt(2.*pi))*exp(-(x-mu)**2./(2.*sigma**2))

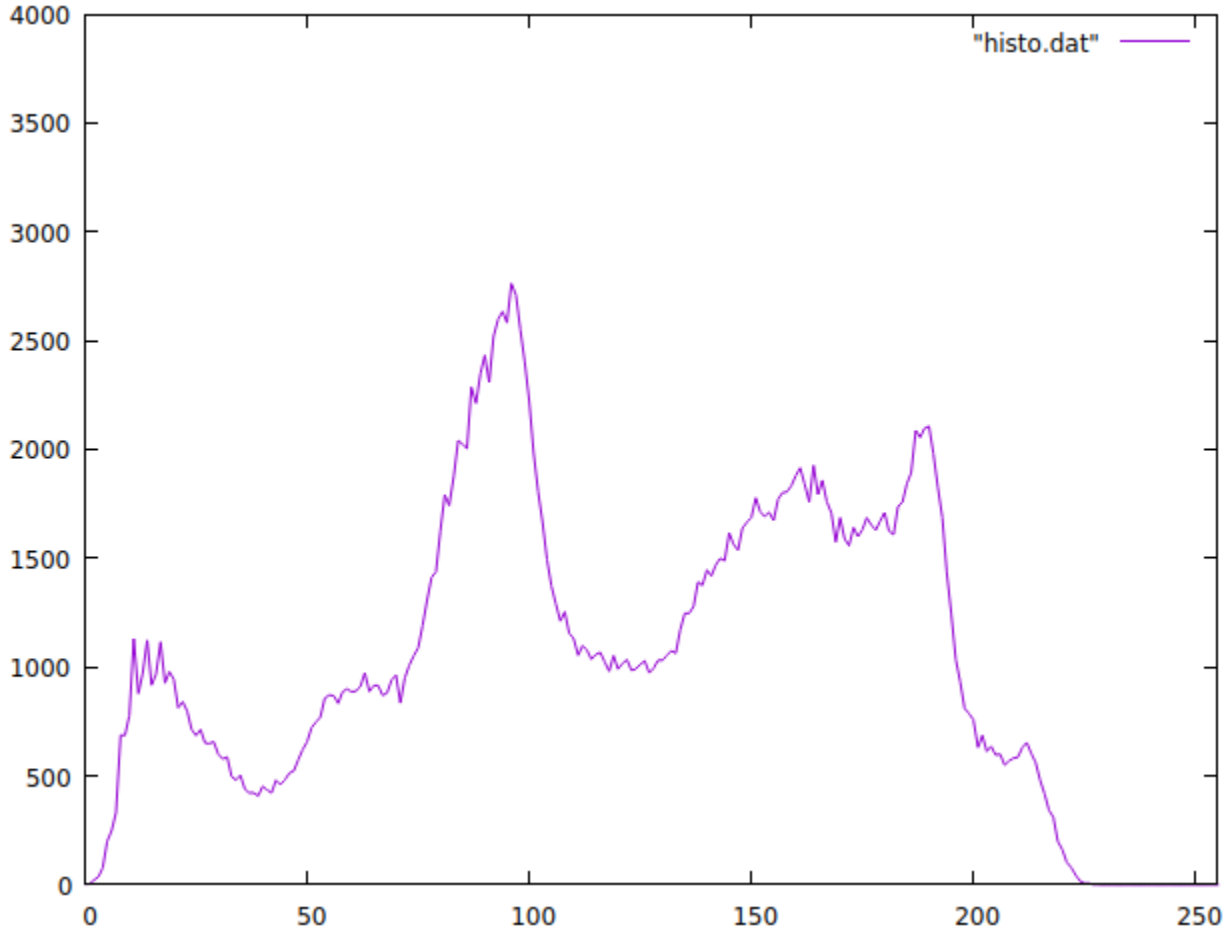


On en déduit que notre fonction de densité de répartition pour cette image est approximable par une loie gaussienne de paramètres donnés par variance.cpp (un seul pic marqué).

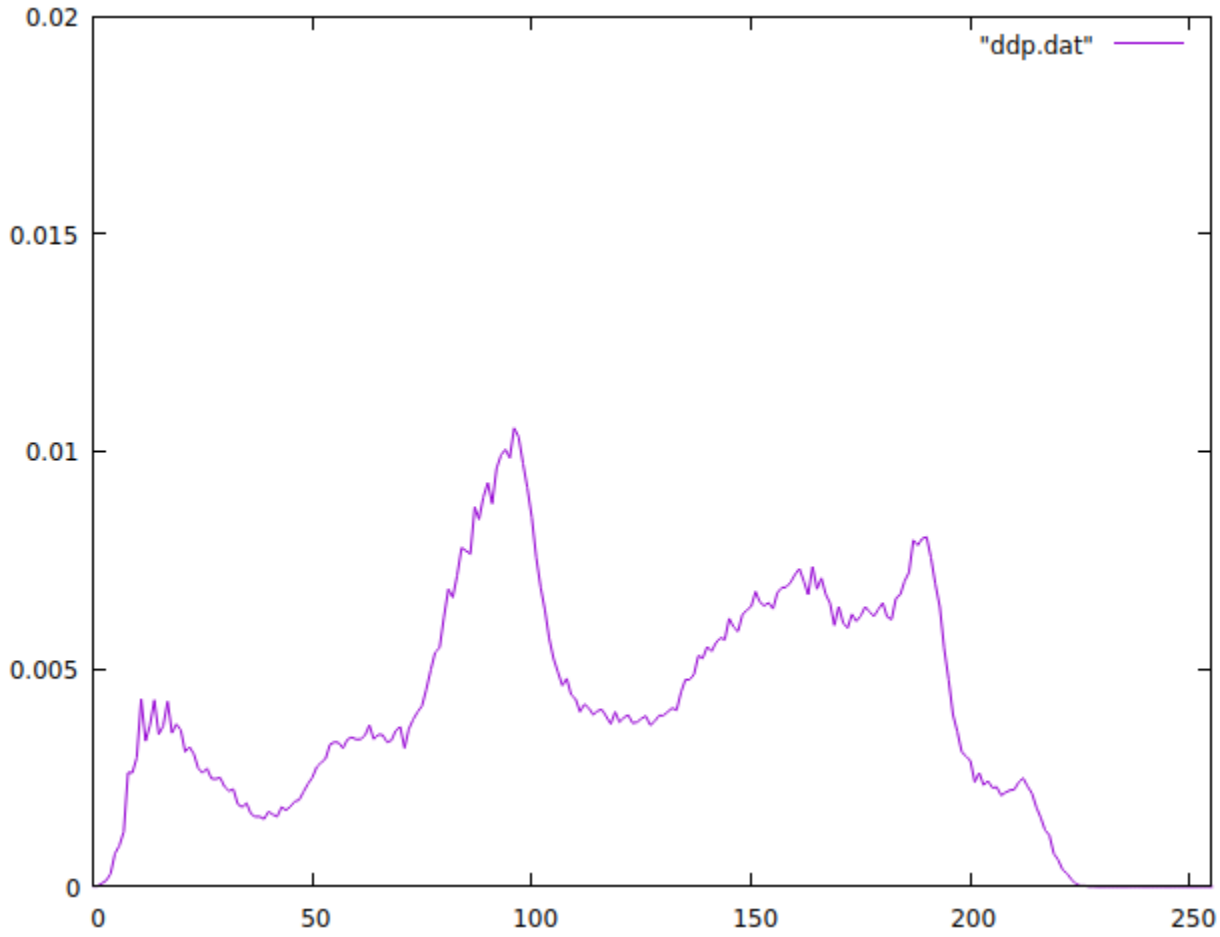
Partie 4

Autre image à tester :

Histogramme

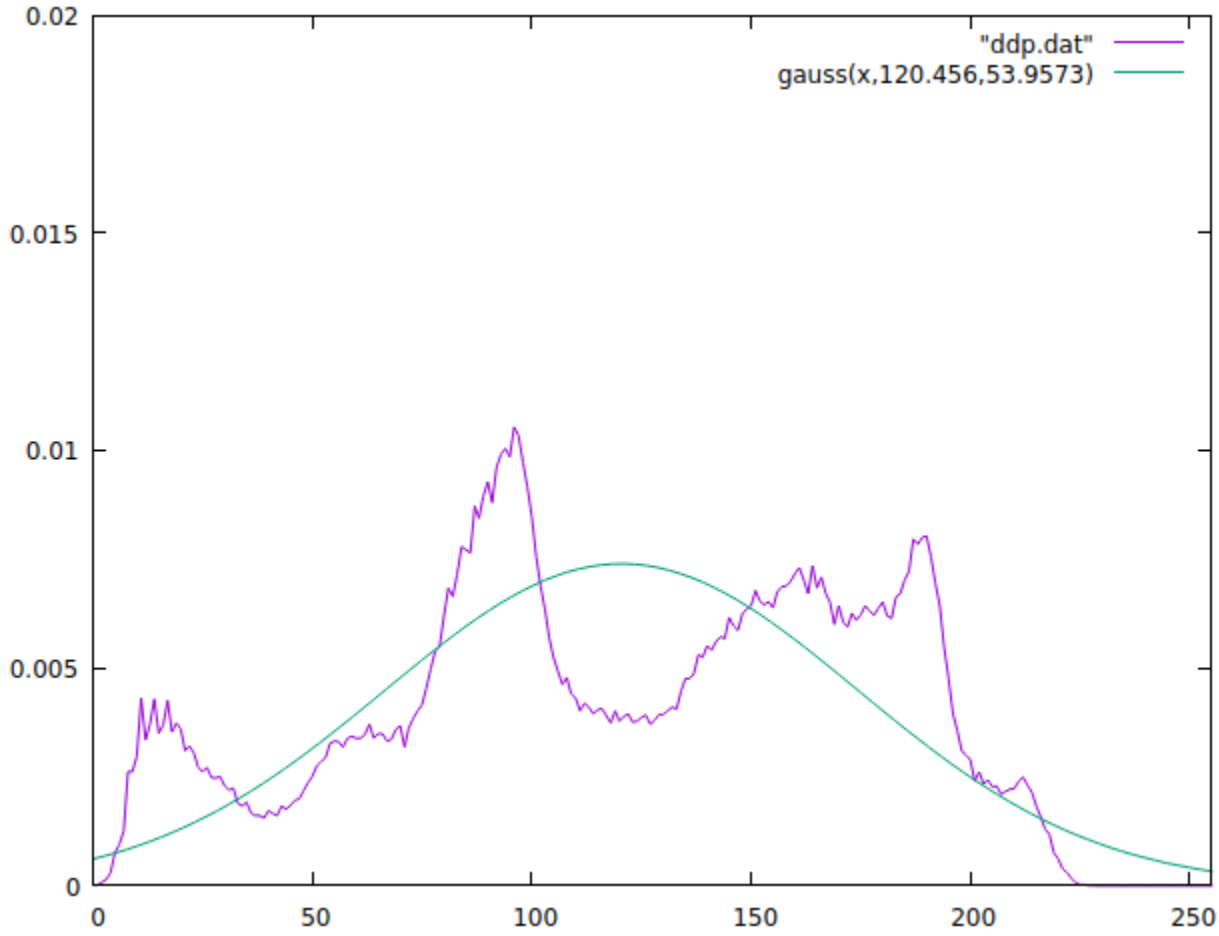


DDP associée



On se rend compte que la courbe de la DDP de cette image est plus complexe que la précédente, car il y a plusieurs pics marqués différents.

En approximant avec une Gaussienne telle que donnée par variance.cpp, on obtient le résultat suivant :



On se rend compte que l'approximation de la DPP d'une telle image n'est pas faisable par une seule gaussienne, car on se retrouve à perdre trop d'informations essentielles. Ici, on remarque qu'il y a au moins trois pics marqués (0-50, 75-110, 140-200), or notre gaussienne ne peut en représenter qu'un.

On doit donc trouver une autre façon d'approcher la DDP de telles images.

Partie 5

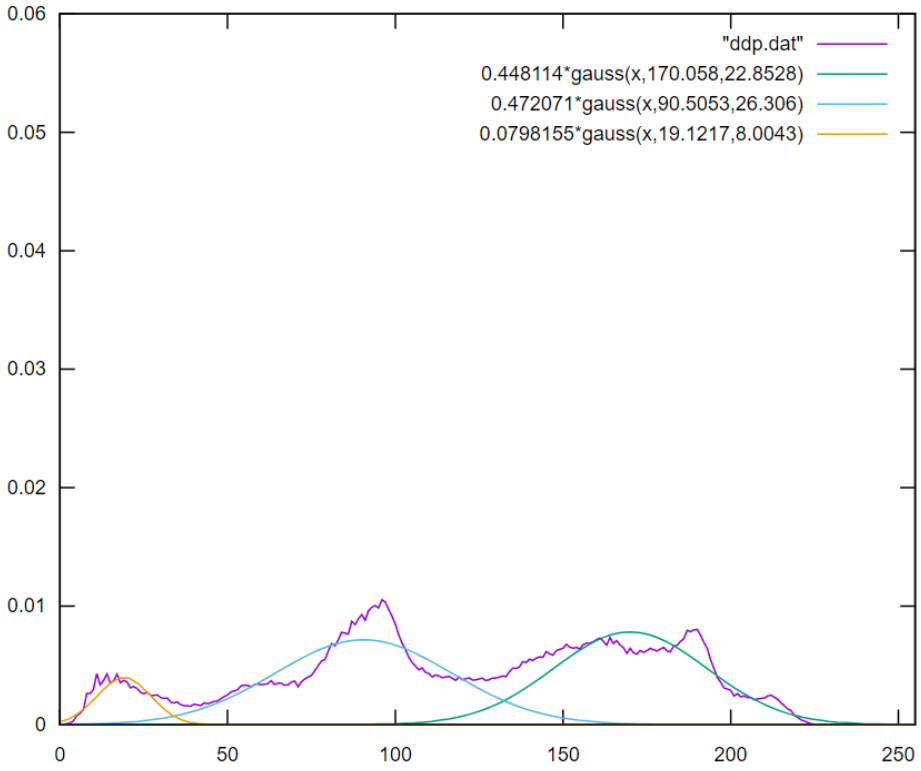
On peut approximer l'image de la partie précédente par un mélange de gaussiennes, au lieu d'une seule.

On se rend d'abord bien compte qu'en ne générant qu'une seule gaussienne par le fichier EM.cpp, on obtient les mêmes valeurs que pour le fichier variance.cpp pour les mêmes images que précédemment :

- Pour la première image : Model 1 : Gaussian (180.572, 39.4444) with proportion of 1
- Pour la seconde image : Model 1 : Gaussian (120.456, 53.9573) with proportion of 1

En utilisant 3 gaussiennes pour la seconde image, on se retrouve avec les courbes suivantes (+ DDP) :

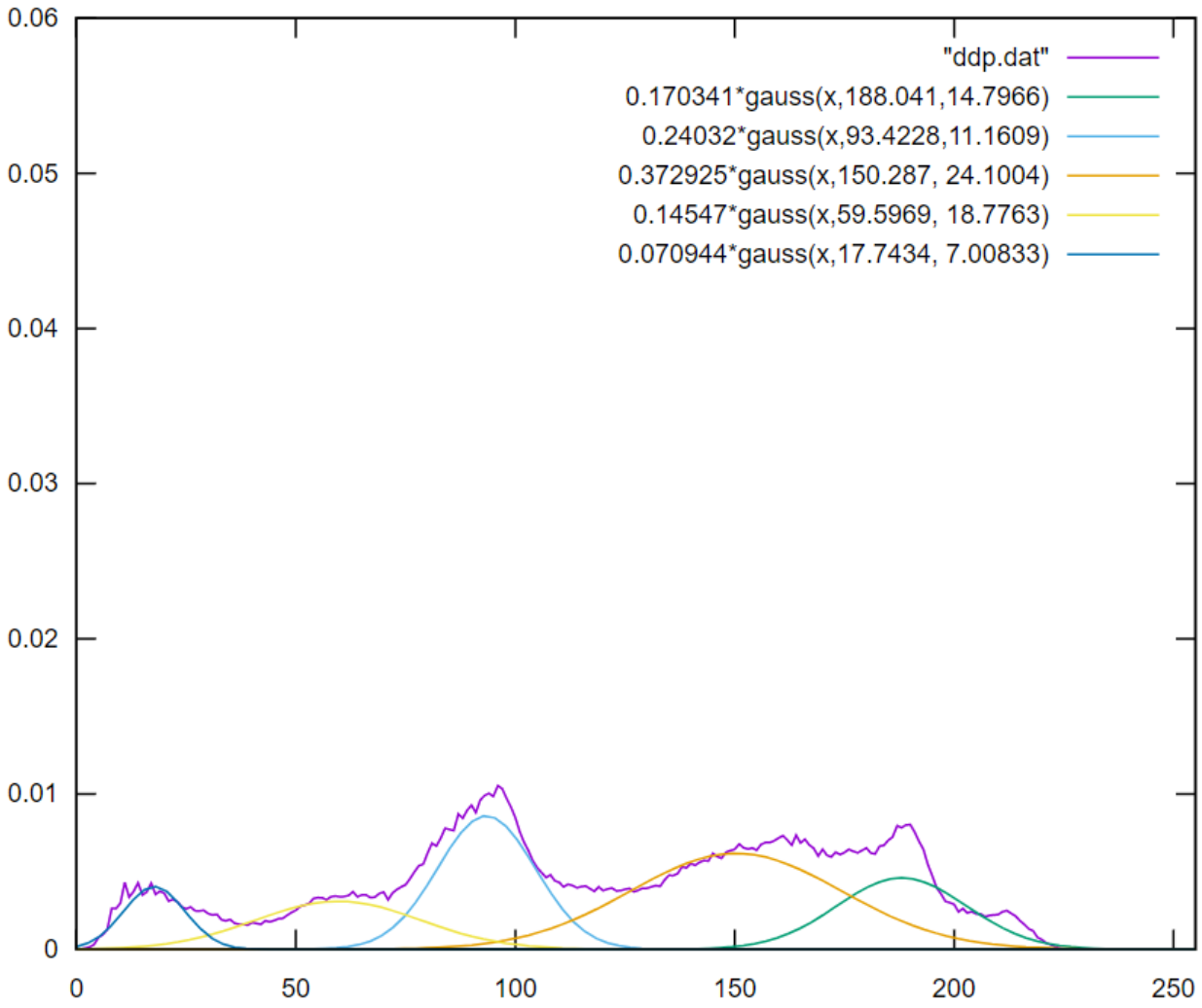
- Model 1 : Gaussian (170.058, 22.8528) with proportion of 0.448114
- Model 2 : Gaussian (90.5053, 26.306) with proportion of 0.472071
- Model 3 : Gaussian (19.1217, 8.0043) with proportion of 0.0798155



On remarque que ce mélange de gaussienne est déjà une bonne approche pour la DDP.

En augmentant le nombre de Gaussienne pour la seconde image de deux, on se retrouve avec les gaussiennes suivantes :

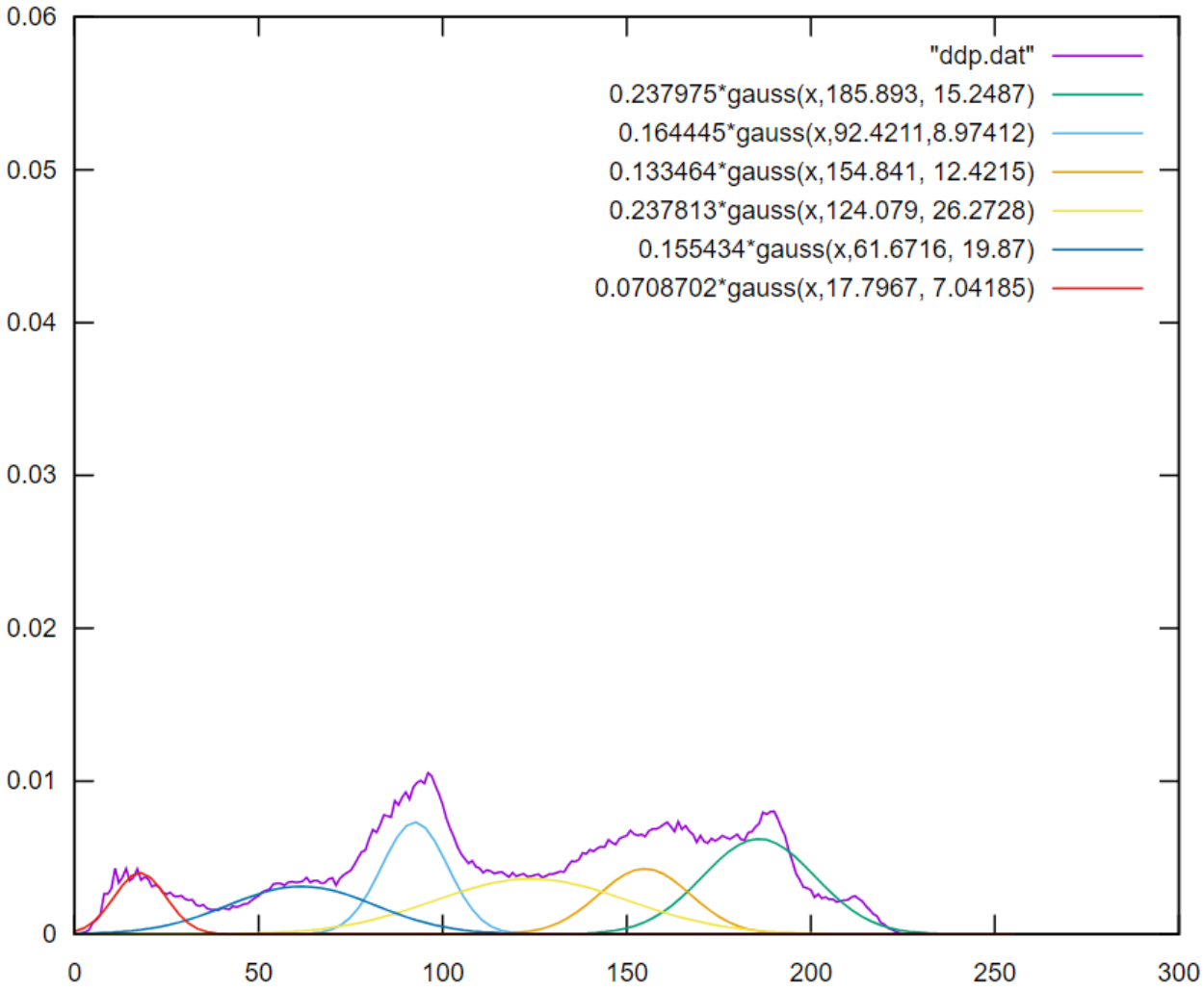
- Model 1 : Gaussian (188.041, 14.7966) with proportion of 0.170341
- Model 2 : Gaussian (93.4228, 11.1609) with proportion of 0.24032
- Model 3 : Gaussian (150.287, 24.1004) with proportion of 0.372925
- Model 4 : Gaussian (59.5969, 18.7763) with proportion of 0.14547
- Model 5 : Gaussian (17.7434, 7.00833) with proportion of 0.070944



Ici, on obtient une bien meilleure approximation sans trop avoir à augmenter le nombre de gaussiennes.

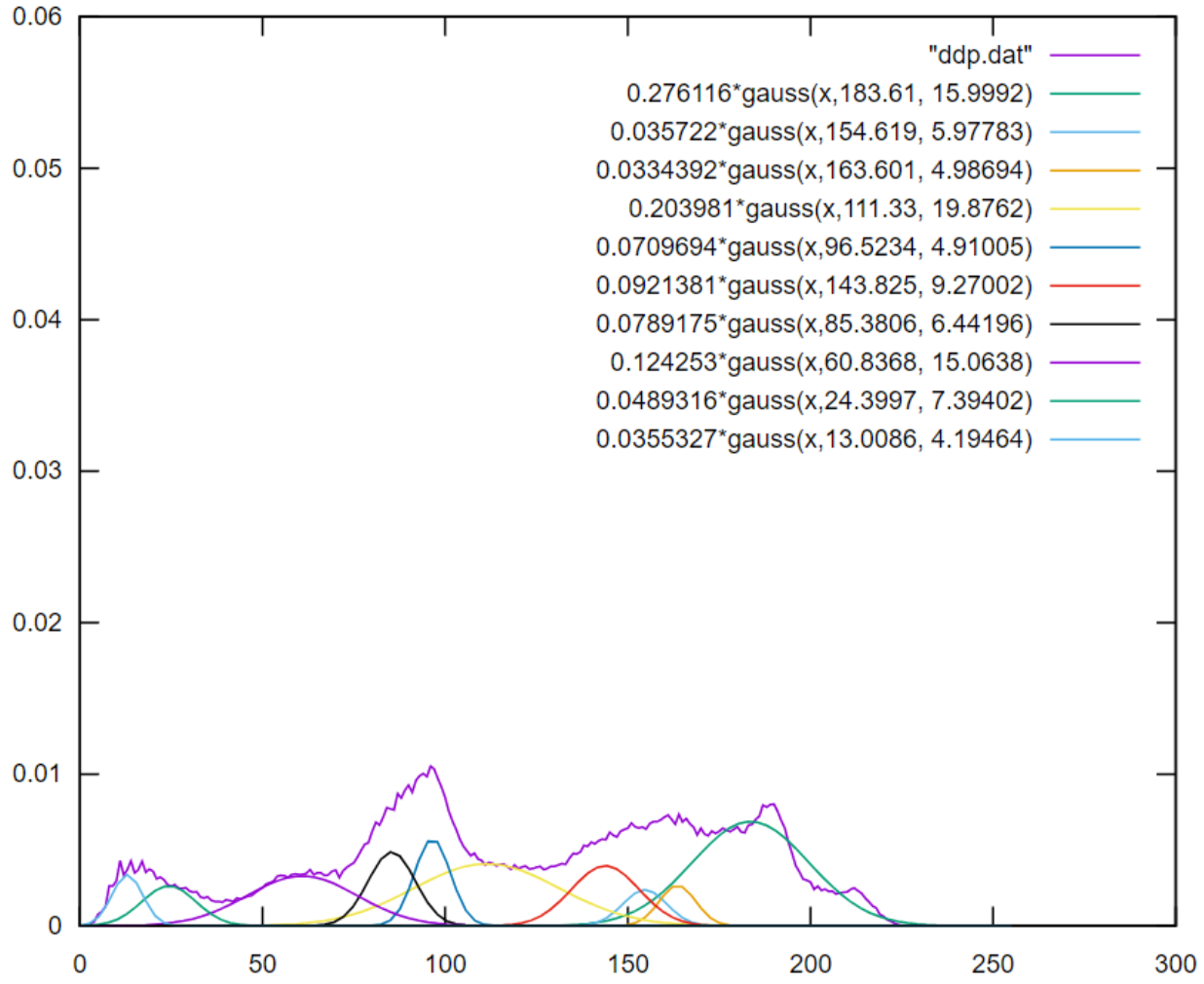
On peut encore en augmenter le nombre :

- Model 1 : Gaussian (185.893, 15.2487) with proportion of 0.237975
- Model 2 : Gaussian (92.4211, 8.97412) with proportion of 0.164445
- Model 3 : Gaussian (154.841, 12.4215) with proportion of 0.133464
- Model 4 : Gaussian (124.079, 26.2728) with proportion of 0.237813
- Model 5 : Gaussian (61.6716, 19.87) with proportion of 0.155434
- Model 6 : Gaussian (17.7967, 7.04185) with proportion of 0.0708702



Avec une dizaine de gaussienne :

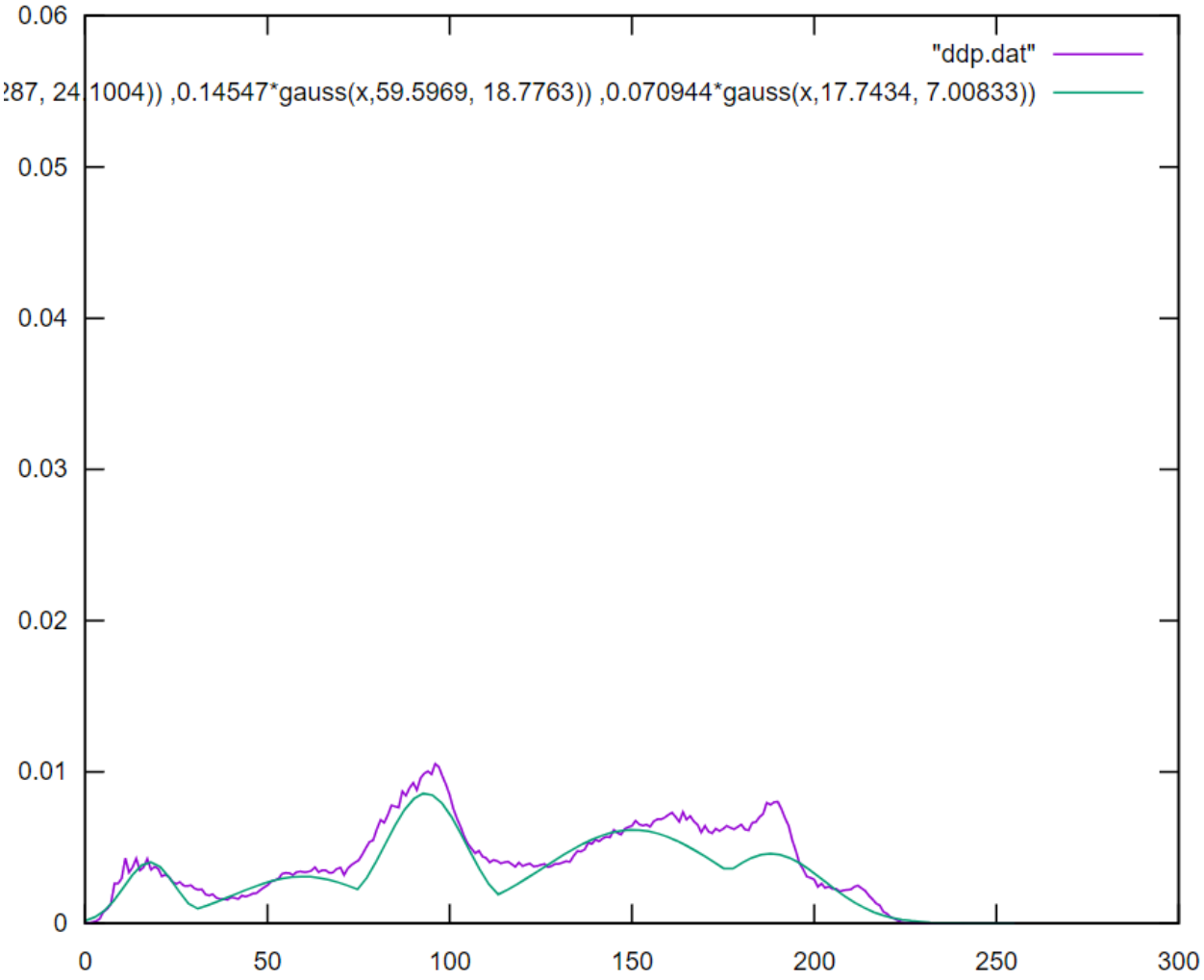
- Model 1 : Gaussian (183.61, 15.9992) with proportion of 0.276116
- Model 2 : Gaussian (154.619, 5.97783) with proportion of 0.035722
- Model 3 : Gaussian (163.601, 4.98694) with proportion of 0.0334392
- Model 4 : Gaussian (111.33, 19.8762) with proportion of 0.203981
- Model 5 : Gaussian (96.5234, 4.91005) with proportion of 0.0709694
- Model 6 : Gaussian (143.825, 9.27002) with proportion of 0.0921381
- Model 7 : Gaussian (85.3806, 6.44196) with proportion of 0.0789175
- Model 8 : Gaussian (60.8368, 15.0638) with proportion of 0.124253
- Model 9 : Gaussian (24.3997, 7.39402) with proportion of 0.0489316
- Model 10 : Gaussian (13.0086, 4.19464) with proportion of 0.0355327



On se rend compte qu'augmenter le nombre de gaussiennes n'est pas nécessaire, passez un certain montant. En effet, le rapport qualité d'approximation gagnée/informations simplifiées n'est plus si important.
De plus, on remarque que l'on va même perdre de l'information (certains pics ne sont tout simplement plus représentés).

On peut donc déduire que cinq gaussiennes semble être un bon nombre pour simplifier la DDP de la seconde image.

>max(x,y) = (x>y)? x : y
>plot "ddp.dat" with lines, max(max(max(max(0.170341*gauss(x,188.041,14.7966), 0.24032*gauss(x,93.4228,11.1609)) ,0.372925*gauss(x,150.287, 24.1004)) ,0.14547*gauss(x,59.5969, 18.7763)) ,0.070944*gauss(x,17.7434, 7.00833))


"ddp.dat"
287, 24 1004)) ,0.14547*gauss(x,59.5969, 18.7763)) ,0.070944*gauss(x,17.7434, 7.00833))
0.06
0.05
0.04
0.03
0.02
0.01
0
0
50
100
150
200
250
300