La transformée de Radon bidimensionnelle

Guillaume CHARPIAT Vincent FEUVRIER
Sujet proposé par Jacques FROMENT

Le 28 avril 2002

1 Principe de la transformée de Radon

Étant donné une fonction f à 2 variables réelles, et à valeur dans $\mathbb{R}$, on définit sa transformée de Radon comme la collection indexée par $(\theta, t) \in [0, 2\pi[\times \mathbb{R}$ et donnée par

$$\mathcal{R}_f(\theta, t) = \int f(x_1, x_2)\delta(x_1 \cos\theta + x_2 \sin\theta - t)dx_1 dx_2$$

Ainsi, $\mathcal{R}_f(\theta, t)$ peut être vue comme l'intégrale de f le long de la droite de *coupe* d'angle θ avec l'axe des ordonnées et passant par le point $(t\cos\theta, t\sin\theta)$. La propriété qui va nous servir pour le calcul numérique de chacune de ces moyennes le long des lignes de coupe est la propriété de projection suivante : si on note $\widehat{f}$ la transformée de Fourier bidimensionnelle de f, alors

$$\widehat{f}(\lambda\cos\theta, \lambda\sin\theta) = \int \mathcal{R}_f(\theta, t)e^{-i\lambda t}dt$$

Ainsi, la coupe selon une droite d'angle θ avec l'axe des abcisses de la transformée de Fourier bidimensionnelle de f peut être vue comme la transformée de Fourier monodimensionnelle de $\mathcal{R}_f(t, \theta)$ par rapport à la variable t et selon la fréquence λ. On a donc que

$$\mathcal{R}_f(\theta, t) = \widehat{f}(\lambda \cos\widetilde{\theta + \lambda} \sin\theta)$$

Ici, $\tilde{.}$ désigne la transformée de Fourier monodimensionnelle inverse par rapport à λ.

On appelle domaine fréquentiel de Radon, noté $\mathcal{E}_R$, l'espace de fonctions à 2 variables engendré par les éléments de la forme

$$\left(u_{\theta,\lambda} : (x_1, x_2) \mapsto e^{i\lambda(x_1 \cos\theta + x_2 \sin\theta)}\right)_{\lambda\in\mathbb{R},\theta\in[0,2\pi[}$$

qui sont chacune les images par rotation d'angle θ de la fonction $f_\lambda(x_1, x_2) = e^{i\lambda x_1}$ (voir figure 1). Cette fonction est constante sur chacune des droites $\{x\} \times \mathbb{R}$, et oscille dans $\mathbb{C}$ à la fréquence λ sur les droites $\mathbb{R} \times \{y\}$. En fait, calculer les coefficients $\widehat{f}(\lambda\cos\theta, \lambda\sin\theta)$ correspond à donner une analyse fréquentielle de $\mathcal{R}_f$ par rapport aux fonctions $u_{\theta,\lambda}$. En effet, on a la formule de reconstruction

$$\mathcal{R}_f(\theta, t) = \int e^{i\lambda t}\widehat{f}(\lambda x_1, \lambda x_2)d\lambda$$

qui va nous permettre de calculer $\mathcal{R}_f(\theta, t)$ avec un algorithme stable numériquement.

2 Implémentation numérique

On suppose que l'image f est donnée par un tableau de taille $n \times n$ contenant des réels, avec n pair : $(f(x,y))_{0 \leq x,y < n}$. À partir des résultats théoriques précédents, on peut donner cet algorithme de calcul de $\mathcal{R}_f(\theta, t)$:

Étape 1 : FFT-2D On calcule la transformée de Fourier bidimensionnelle discrète de f, qui nous donne le tableau $(\widehat{f}(k_1, k_2))_{-\frac{n}{2} \leq k_1, k_2 < \frac{n}{2}}$

Étape 2 : conversion de cartésien à polaire On réalise la coupe du tableau $\widehat{f}$ selon les droites d'angle θ avec l'axe des abcisses ; c'est à dire qu'on obtient un paramétrage des données précédentes par rapport à θ et t.

Étape 3 : FFT-1D On calcule finallement la transformée de Fourier inverse de chacune des lignes du tableau prédédent (à θ constant, on fait la transformée par rapport à la variable t), et on obtient la transformée de Radon de f.

Les étapes 1 et 3 ne présentent pas de difficultés particulières, nous allons nous intéresser au calcul numérique de la conversion de cartésien à polaire.

On choisit de faire une approximation par *plus proche voisin* pour calculer les coefficients en polaire à partir de ceux en cartésien. On procède de la façon suivante (voir gigure 2) :

- Pour k variant de $-\frac{n}{2}$ à $\frac{n}{2} - 1$, on considère la droite H_k passant, dans le plan de Fourier, par les points de coordonnées $(-\frac{n}{2}, -k)$ et $(\frac{n}{2}, k)$. On regarde les n points d'intersection de la droite H_k avec les droites d'équations $x = k, -\frac{n}{2} \leq k < \frac{n}{2}$. On les appelle les $(P_{i,k})_{-\frac{n}{2} \leq i < \frac{n}{2}}$, pour chacun des $P_{i,k}$, on prend son plus proche voisin dans $\mathbb{Z}^2$, qu'on note $Q_{i,k}$ (son abcisse est i). On prendra alors comme approximation de $\mathcal{R}_f(\theta, t)$ la suite des valeurs de $\widehat{f}$ en chacun des points $Q_{i,k}$ pour θ qui est l'angle de la droite H_k avec l'axe des abcisses.
- Pour k variant de $-\frac{n}{2}$ à $\frac{n}{2} - 1$, on considère la droite V_k passant, dans le plan de Fourier, par les points de coordonnées $(-k, -\frac{n}{2})$ et $(k, \frac{n}{2})$. On refait le même calcul d'intersection avec les droites d'équation $y = k, -\frac{n}{2} \leq k < \frac{n}{2}$, et on obtient n nouveaux coefficients à chaque fois.

Ainsi, à la fin on a en tout $2n \times n$ coefficients complexes (voir figure 4), notés $RFreq_f(k,i)$, et on est passé par chaque pixel du plan de Fourier au moins une fois. On peut aussi remarquer que l'angle θ ne varie pas uniformément en fonction de l'indice k de la ligne (voir figure 3).

L'algorithme de la transformée de Radon inverse, consiste à

- Faire une transformée de Fourier monodimensionnelle sur chacune des lignes.
- Recombiner les données selon le principe précédent de plus proche pixel, et pour les cases où il y a redondance (en effet, certains pixels sont recouverts plusieurs fois par cette méthode), on prendra la moyenne arithmétique de toutes les données recouvrant le pixel considéré.
- Faire la transformée de Fourier bidimensionnelle inverse du tableau obtenu.

Les étapes de la transformée de Radon inverse sont exactement les inverses de celles de la transformée de Radon numérotées dans le sens inverse. La transformée de Fourier bidimensionnelle a une complexité algorithmique (quand on

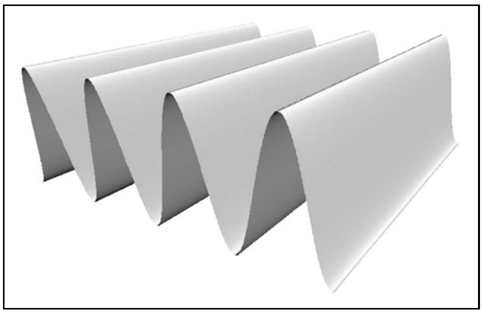

FIG. 1 – Aspect de la partie réelle d'une des fonctions de la base fréquentielle de l'espace de Radon.

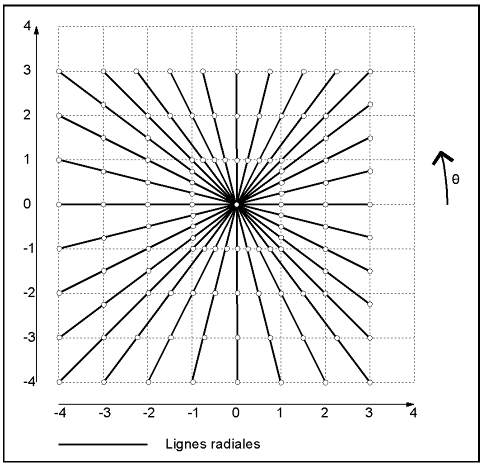

FIG. 2 – Le passage de rectangulaire à polaire : les points $P_{i,k}$ sont représentés par des petits cercles.

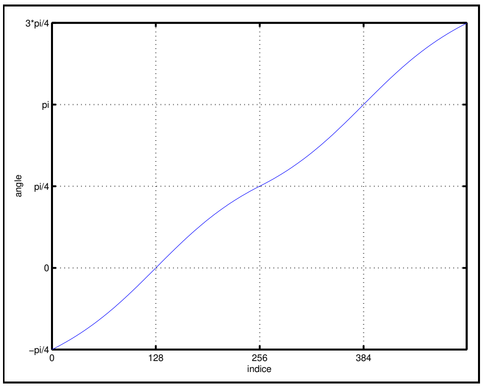

FIG. 3 – Le passage de rectangulaire à polaire : représentation des angles θ en fonction de l'indice k considéré. Comme on peut le voir, il ne s'agit pas d'une droite ; on peut considérer que l'image a subi une petite déformation dans l'espace de Fourier avant traitement par rapport à la transformée de Radon rigoureuse.

Fig. 4 – Un exemple de transformée de Radon. *En haut à gauche* : l'image originale. *En haut à droite* : sa transformée en polaire. *En bas à gauche* : la norme des coefficients complexes de la transformée en polaire de sa transformée de Fourier bidimensionnelle (2ème étape). *En bas à droite* : norme des coefficients complexes de la transformée de Radon complète (3ème étape).

utilise l'algorithme de transformée rapide) de l'ordre de $O(n^2 \log n)$; celle de la conversion en polaire $O(n^2)$; celle de la transformée de Fourier monodimensionnelle $O(n \log n)$, mais il faut la faire $2n$ fois, donc $O(n^2 \log n)$ en tout. La complexité algorithmique totale de notre méthode est donc $O(n^2 \log n)$, la même que la transformée de Radon inverse, car chacune des inverses des étapes a la même complexité que l'étape correspondante.

3 Considérations algorithmiques - amélioration possible

L'intérêt de la méthode proposée, est d'abord qu'elle permet d'avoir en sortie des données de taille rectangulaire : un tableau de dimensions $2n \times n$. En effet, on aurait pu imaginer une solution où pour le passage en polaire pour des angles θ proche des diagonales (droites d'angle $\frac{\pi}{2}$ et $-\frac{\pi}{2}$) on aurait gardé plus de coefficients (en effet, il y a plus de pixels traversés) que pour les droites d'angle proche de 0 ou $\frac{\pi}{2}$. De plus, un autre avantage, est qu'elle est très stable par rapport à un calcul direct de la transformée de Radon par une seule sommation, sans passer par le plan de Fourier.

En effet, considérons que l'image après la transformée de Radon ait subi une compression. Pour rester dans le cas général, on va modéliser l'effet de celle-ci par un bruit aléatoire multiplicatif après la transformée monodimensionnelle inverse sur les lignes : $\widetilde{RFreq_f}(k,i) = (1+\epsilon(k,i))RFreq_f(k,i)$ où les $\epsilon(k,i)$ sont des bruits gaussiens centrés en 0 de variance σ^2 sur $\mathbb{Z}^2$ et supposés indépendants (hypothèse assez discutable, car ne fonctionnera pas avec toutes les compressions possibles). Alors, on peut déjà montrer que chaque pixel (k_1, k_2) du plan de Fourier est recouvert au moins $\frac{n}{\max(|k_1|,|k_2|)} - 1$ fois lors de la conversion de polaire en rectangulaire. Fixons k_1 et k_2, et notons $(k_l, i_l)_{1 \leq l \leq m}$ les m points (k,i) avant conversion en polaire qui vont recouvrir (k_1, k_2) lors de la conversion en rectangulaire. Donc, puisqu'on a choisi de faire la moyenne, on aura après compression :

$$\widetilde{\widehat{f}}(k_1,k_2) = \frac{1}{m}\sum_{l=1}^{m} \widetilde{RFreq_f}(k_l,i_l) = \widehat{f}(k_1,k_2)\left(1+\frac{1}{m}\sum_{l=1}^{m}\epsilon(k_l,i_l)\right)$$

Or $\frac{1}{m}\sum_{l=1}^{m}\epsilon(k_l,i_l)$ est un bruit gaussien d'après nos hypothèses, et de variance $\frac{\sigma^2}{m}$, donc en moyenne, les coefficients dans le plan de Fourier dorrespondant aux basses fréquences seront donnés précisément après compression, car avec une variance beaucoup plus faible (de l'ordre de $\frac{\sigma^2 \max(|k_1|,|k_2|)}{n}$ pour n grand). On pourrait imaginer utiliser une autre méthode de moyenne pour la redondance, qui fasse baisser la variance encore plus vite, mais dans ce cas on aurait intérêt à mieux formaliser la compression après la transformée de Radon, ce qui n'est pas notre propos.

A Annexes

A.1 Code des procédures Megawave

A.1.1 Code source de la fonction rectpolar

Ceci est le fichier `rectpolar.c` ; cette fonction calcule la transformée en coordonnées polaire d'une image à coefficients complexes.

```
/*-------------------------- MegaWave2 Command ----------------------------*/
/* mwcommand
   name = {rectpolar};
   version = {"1.0"};
   author = {"Guillaume Charpiat, Vincent Feuvrier, Claire Jonchery, Amandine Robin"};
   function = {"Rectangular to Polar Coordinates Transformation"};
   usage = {
       'I':input_im->in_im          "imaginary input (Fimage)",
       'R':output_re<-out_re      "real output (Fimage)",
       'C':output_im<-out_im  "imaginary output (Fimage)",
       input_re->in_re              "real input (Fimage)"
   };
*/

#include <stdio.h>
#include <math.h>
#include "mw.h"

#define near(x) ((((x)-floor(x))>.5)?(ceil(x)):(floor(x)))


/* NB :
 * The input Fimages must be square-shaped.
 * If the size of the image (its width or length) is n, the size of the output
   (Fimage) will be n * 2n;
 * calling this module with in_im=NULL is possible and means
   in_im(x,y) = 0 everywhere;
 * calling this module with out_re=NULL or out_im=NULL
   is possible (if you are not interested on one of these outputs).
*/


void rectpolar(in_re, in_im, out_re, out_im)
     Fimage in_re, in_im;
     Fimage out_re, out_im;
{
  int     indic1, indic2, j;
  long n,i,j1;
  float x, y;
  int x1, y1;

  if (out_re == NULL) indic1 = 0; else indic1 = 1;
  if (out_im == NULL) indic2 = 0; else indic2 = 1;

  if ((indic1 == 0) && (indic2 == 0)) mwerror(USAGE,1, "At least one output needed\n");
  if (in_re == NULL) mwerror(USAGE,1, "Input is empty...\n");

  n = in_re->nrow;

  if (!(n == in_re->ncol)) mwerror(USAGE,1, "The image is not square-shaped");

  /***** Allocate input imaginery image (black) if necessary *****/
```

```
  if (in_im == NULL) {
    in_im = mw_change_fimage(in_im, n, n);
    if (in_im == NULL) mwerror(FATAL,1, "Not enough memory");
    mw_clear_fimage(in_im, 0);
  }

  /******  Allocate output ******/

  if (indic1 == 1) {
    out_re = mw_change_fimage(out_re, 2*n, n);
    if (out_re == NULL)  mwerror(FATAL,1, "Not enough memory");
  }

  if (indic2 == 1) {
    out_im = mw_change_fimage(out_im, 2*n, n);
    if (out_im == NULL)  mwerror(FATAL,1, "Not enough memory");
  }


  /**** Allocate Fsignal results ****/
  /*  if ((aux = mw_change_fsignal(aux, size)) == NULL) mwerror(FATAL,1, "Not enough memory");*/

  /*** Convert rectangular to polar coordinates ***/
  /** i : ième ligne de la transformée de Radon **/
  /** j : jième colonne de la ième ligne **/

  for (i=0; i<(n+1); i++) {
    for (j=0; j<(n+1); j++) {

      x = j-1./2;
      y = i + j*(1-2.*i/n)-1./2;

<       x1 = near(x);
      y1 = near(y);

      if (x1 < 0) x1 = 0;
      if (x1 > (n-1)) x1 = n-1;
      if (y1 < 0) y1 = 0;
      if (y1 > (n-1)) y1 = n-1;

      j1 = j;
      if (j1 > (n/2)) j1--;

      if (indic1 == 1) out_re->gray[j1+n*i] = in_re->gray[x1+n*y1];
      if (indic2 == 1) out_im->gray[j1+n*i] = in_im->gray[x1+n*y1];

    }
  }

  for (i=1; i<n; i++) {
    for (j=0; j<(n+1); j++) {


      x = i+j*(1-2.*i/n)-1./2;
      y = n - j - 1./2;

      x1 = near(x);
      y1 = near(y);

      if (x1 < 0) x1 = 0;
      if (x1 > (n-1)) x1 = n-1;
```

```
      if (y1 < 0) y1 = 0;
      if (y1 > (n-1)) y1 = n-1;

      j1 = j;
      if (j1 > (n/2)) j1--;

      if (indic1 == 1) out_re->gray[j1+n*i+n*n] = in_re->gray[x1+n*y1];
      if (indic2 == 1) out_im->gray[j1+n*i+n*n] = in_im->gray[x1+n*y1];

    }
  }



  /***** free output images if necessary ******/
  /*mw_delete_fsignal(aux);*/

}
```

A.1.2 Code source de la fonction rectpolar_inv

Ceci est le fichier `rectpolar_inv.c`; cette fonction calcule la transformée de polaire en rectangulaire d'une image à coefficients complexes de taille $n \times 2n$ où n est pair.

```
/*--------------------------- MegaWave2 Command -----------------------------*/
/* mwcommand
   name = {rectpolar_inv};
   version = {"1.0"};
   author = {"Guillaume Charpiat, Vincent Feuvrier"};
   function = {"Polar to Rectangular Coordinates Transformation"};
   usage = {
       'R':output_re<-out_re              "real output (Fimage)",
       'I':output_im<-out_im          "imaginary output (Fimage)",
       'C':input_im->in_im  "imaginary input (Fimage)",
       input_re->in_re      "real input (Fimage)"
   };
*/

#include <stdio.h>
#include <math.h>
#include "mw.h"

#define near(x) ((int)((((x)-floor(x))>.5)?(ceil(x)):(floor(x))))

/* NB :
 * The input Fimages must be 2n * n;
 * The output Fimages will be square-shaped.
 * calling this module with in_im=NULL is possible and means
   in_im = 0 everywhere;
 * calling this module with out_re=NULL is not possible;
   but you can call this module with out_im=NULL
   (if you are not interested on this output).
*/


void rectpolar_inv(out_re, out_im, in_re, in_im)
     Fimage out_re, out_im;
     Fimage in_re, in_im;
```

```
{
  Fimage A;
  int indic,indic3;
  long   i, n, j;
  int j1;
  float x, y;
  long x1, y1;


  /*  Fsignal aux;*/

  if (out_re == NULL) indic3 = 0; else indic3 = 1;
  if (out_im == NULL) indic = 0; else indic = 1;
  if ((indic == 0) && (indic3 == 0)) mwerror(USAGE,1, "No output specified");

  if (in_re == NULL) mwerror(USAGE,1, "Input is empty...\n");

  n = in_re->ncol;

  /**** Allocate A ****/

  A = mw_new_fimage();
  A = mw_change_fimage(A, n, n);

  if (A == NULL) mwerror(FATAL,1, "Not enough memory");

  mw_clear_fimage(A,0);

  /**** Allocate output ****/

  if (indic3 == 1) {
    out_re = mw_change_fimage(out_re, n, n);
    if (out_re == NULL)  mwerror(FATAL,1, "Not enough memory");
    mw_clear_fimage(out_re, 0);
  }

  if (indic == 1) {
    out_im = mw_change_fimage(out_im, n, n);
    if (out_im == NULL)  mwerror(FATAL,1, "Not enough memory");
    mw_clear_fimage(out_im, 0);
 }

  /***** Allocate input imaginery image (black) if necessary *****/

  if ((in_im == NULL) && (indic == 1)) {
    indic = 0;
    mw_clear_fimage(out_im, 0);
  }

  /**** Allocate Fsignal results ****/
  /*  if ((aux = mw_change_fsignal(aux, size)) == NULL)
        mwerror(FATAL,1, "Not enough memory");*/

  /*** Convert rectangular to polar coordinates ***/
  /** i : ième ligne de la transformée de Radon **/
  /** j : jième colonne de la ième ligne **/

  for (i=0; i<(n+1); i++) {
    for (j=0; j<(n+1); j++) {

      x = j-1./2;
      y = i + j*(1-2.*i/n)-1./2;
```

```
      x1 = near(x);
      y1 = near(y);

      if (x1 < 0) x1 = 0;
      if (x1 > (n-1)) x1 = n-1;
      if (y1 < 0) y1 = 0;
      if (y1 > (n-1)) y1 = n-1;

      j1 = j;
      if (!(j1 == (n/2))) {
if (j1 > (n/2)) j1--;

(A->gray[x1+n*y1])++;

if (indic3 == 1)
  out_re->gray[x1+n*y1] = (((A->gray[x1+n*y1])-1)*(out_re->gray[x1+n*y1]) + in_re->
          gray[j1+n*i])/(A->gray[x1+n*y1]);


if (indic == 1)
        out_im->gray[x1+n*y1] = ((A->gray[x1+n*y1]-1)*(out_im->gray[x1+n*y1]) + in_im->
          gray[j1+n*i])/(A->gray[x1+n*y1]);
      }
    }
  }

  for (i=1; i<n; i++) {
    for (j=0; j<(n+1); j++) {

      x = i+j*(1-2.*i/n)-1./2;
      y = n - j - 1./2;

      x1 = near(x);
      y1 = near(y);

      if (x1 < 0) x1 = 0;
      if (x1 > (n-1)) x1 = n-1;
      if (y1 < 0) y1 = 0;
      if (y1 > (n-1)) y1 = n-1;

      j1 = j;
      if (!(j1 == (n/2))) {
if (j1 > (n/2)) j1--;
(A->gray[x1+n*y1])++;
if (indic3 == 1)
  out_re->gray[x1+n*y1] = ((A->gray[x1+n*y1]-1)*(out_re->gray[x1+n*y1]) + in_re->
          gray[j1 + n*i + n*n])/(A->gray[x1+n*y1]);
if (indic == 1)
        out_im->gray[x1+n*y1] = ((A->gray[x1+n*y1]-1)*(out_im->gray[x1+n*y1]) + in_im->
          gray[j1 + n*i + n*n])/(A->gray[x1+n*y1]);
      }
    }
  }

  /***** free output images if necessary ******/
  /*  mw_delete_fsignal(aux);*/

  mw_delete_fimage(A);
  A = NULL;
}
```

A.2 Remerciements

Merci à Jacques Froment pour son aide dans ce travail sur la transformée de Radon, ainsi qu'à Claire Jonchery et Amandine Robin qui avaient à réaliser la transformée en *ridgelets* utilisant la transformée de Radon, et qui nous ont aidé, en particulier pour l'utilisation de Megawave.