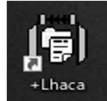


核医学検査技術学実習

実習3 フィルタ処理、畳込み
画像処理において用いられる実空間、
空間周波数フィルタ処理とその演算
方法である畳込みについて学ぶ。

Practice3.zipをダウンロードして解凍
(解凍ソフトLhacaをインストールして、
Lhacaのアイコン上へ、解凍したい
zipファイルをドロップダウンする。)



1

解凍ソフトLhacaがうまく作動しない
場合は、7zipをインストールして、
zipファイルを解凍して下さい。

2

FFT Fast Fourier Transform

FFT.exe を起動し、open fileから同フォルダ内にある
boneを選択する。

※bone: $^{99m}\text{Tc-MDP}$ のSPECT画像。骨シンチ。

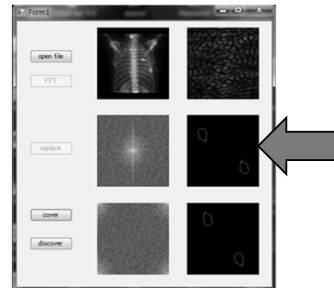
FFTをクリックして2次元フーリエ変換を行い、replace
をクリックして、パワースペクトル画像を再配置する。

discoverをクリックすると、逆フーリエ変換された画像
が表示される。

coverをクリックすると、右側の画像が消える。

3

右中の枠をクリック・ドラッグすると、
パワースペクトルが部分的に逆フーリエ変換される。



coverをクリックすると初期化される。
色々試してみてください。

4

実空間フィルタ

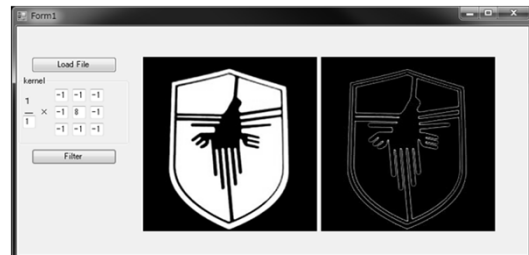
フォルダFilter内にあるFilter.exeを起動し、「Load
File」ボタンを押してBitmapまたはJPEG画像を選択す
る。(プログラムフォルダ内にあるsample.jpgを使用し
ても良い)

「Kernel」の数値(9個のフィルタ行列の要素)を
変更し、任意の3×3の実空間フィルタを作成する。

「Filter」ボタンを押し、作成したフィルタによりフィルタ
処理を行う。

5

処理を行うと以下の様な画像が得られる



6

画像に平滑化フィルタ処理(smoothing)を行う

3x3 smoothing filter

$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$
$\frac{1}{10}$	$\frac{2}{10}$	$\frac{1}{10}$
$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$

1画素の値を、その近傍にも少し影響を及ぼすようにする。合計が1になっている点に注目。(カウント合計値は変化しない)

7

平滑化フィルタ 実空間での雑音除去フィルタ

フィルタ処理前後で画素値の総和が変わらないようにフィルタ内成分の和が1になっていることに注目。

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

移動平均フィルタ (成分値が全部同じ)

最も単純な 3x3 スムージングフィルタ。中央画素の重み付けがない。輪郭の鮮明さが損なわれる。

$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$
$\frac{1}{10}$	$\frac{2}{10}$	$\frac{1}{10}$
$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{10}$

荷重平均フィルタ (成分値が中心部で大きい)

輪郭の鮮明さを維持するために中心部に重みを付けた 3x3, 5x5 スムージングフィルタ。

0	0	$\frac{1}{13}$	0	0
0	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	0
$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$
0	$\frac{1}{13}$	$\frac{1}{13}$	$\frac{1}{13}$	0
0	0	$\frac{1}{13}$	0	0

8

平成18年 国家試験

問題 97 3×3の空間フィルタを示す。画像の鮮鋭化に用いるのはどれか。ただし、数字は重み係数を示す。

1.

$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$
$\frac{2}{16}$	$\frac{4}{16}$	$\frac{2}{16}$
$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$

 2.

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

 3.

-1	-1	-1
0	0	0
1	1	1

4.

-1	0	1
-2	0	2
-1	0	1

 5.

0	-1	0
-1	4	-1
0	-1	0

解答 5

9

```
void filter1( int x[ ][260], int y[ ][260] )
{
    // Weighted smoothing filter
    int i, j;
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){
        y[i][j] = x[i-1][j-1] + x[i][j-1]*2 + x[i+1][j-1]
                + x[i-1][j]*2 + x[i][j]*4 + x[i+1][j]*2
                + x[i-1][j+1] + x[i][j+1]*2 + x[i+1][j+1] ;
    }}
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){ y[i][j] /= 16; }}
}
```

輪郭の鮮明さを維持するために中心部に重みを付けたスムージングフィルタ。

1.

$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$
$\frac{2}{16}$	$\frac{4}{16}$	$\frac{2}{16}$
$\frac{1}{16}$	$\frac{2}{16}$	$\frac{1}{16}$

10

```
void filter2( int x[ ][260], int y[ ][260] )
{
    // Simple smoothing filter
    int i, j;
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){
        y[i][j] = x[i-1][j-1] + x[i][j-1] + x[i+1][j-1]
                + x[i-1][j] + x[i][j] + x[i+1][j]
                + x[i-1][j+1] + x[i][j+1] + x[i+1][j+1] ;
    }}
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){ y[i][j] /= 9; }}
}
```

最も単純なスムージングフィルタ。中央画素の重み付けがない。輪郭の鮮明さが損なわれる。

2.

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

11

```
void filter3( int x[ ][260], int y[ ][260] )
{
    // Vertical Prewitt filter
    int i, j;
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){
        y[i][j] = x[i-1][j-1]*-1 + x[i][j-1]*-1 + x[i+1][j-1]*-1
                + x[i-1][j]*0 + x[i][j]*0 + x[i+1][j]*0
                + x[i-1][j+1] + x[i][j+1] + x[i+1][j+1] ;
    }}
}
```

Prewitt フィルタ (垂直方向)

1次微分(差分)エッジ検出フィルタ
上下方向に画素値が大きく変化する部位(輪郭)の抽出フィルタ

3.

-1	-1	-1
0	0	0
1	1	1

12

```
void filter4( int x[ ][260], int y[ ][260] )
{
    // Horizontal Sobel filter
    int i, j;
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){
        y[i][j] = x[i-1][j-1]*-1 + x[i][j-1]*0 + x[i+1][j-1]
            + x[i-1][j]*-2 + x[i][j]*0 + x[i+1][j]*2
            + x[i-1][j+1]*-1 + x[i][j+1]*0 + x[i+1][j+1] ;
    }
}
```

Sobel フィルタ (水平方向)

1次微分(差分)エッジ検出フィルタ
左右方向に画素値が大きく変化
する部位(輪郭)の抽出フィルタに
中央部の重み付けを加えている。

4.

-1	0	1
-2	0	2
-1	0	1

13

```
void filter5( int x[ ][260], int y[ ][260] )
{
    // Unsharp masking filter
    int i, j;
    for(j=2;j<=255;j++){ for(i=2;i<=255;i++){
        y[i][j] = x[i-1][j-1]*0 + x[i][j-1]*-1 + x[i+1][j-1]*0
            + x[i-1][j]*-1 + x[i][j]*4 + x[i+1][j]*-1
            + x[i-1][j+1]*0 + x[i][j+1]*-1 + x[i+1][j+1]*0 ;
    }
}
```

4近傍 先鋭化フィルタ

アンシャープマスキング。
ぼやけた部位をマスクする。
上下および左右方向に画素値が
大きく変化する部位(輪郭)の強調

5.

0	-1	0
-1	4	-1
0	-1	0

14

空間周波数フィルタ

フォルダ Butterworth 内にある Butterworth.exe
を起動し、「Load File」ボタンを押して、
プログラムフォルダ内にある bone ファイルを
読み込む。

「display sin cos component」ボタンを押し、
原画像のフーリエ変換を行い、の sin 成分
と cos 成分の画像を表示させる。

「FFT spectrum」ボタンを押し、パワースペ
クトルを表示する。

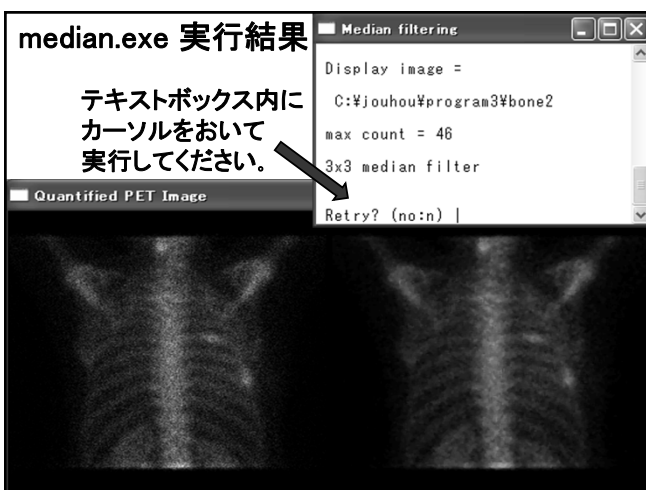
15

テキストボックス内に 0.5~0.01 の任意の遮断周
波数を設定する。

「Butterworth filter」ボタンを押し、設定した遮断周
波数からバターワースフィルタを作成し、空間周
波数フィルタ処理を行う。

「IFFT」ボタンを押し、逆フーリエ変換を行い、フィル
タ処理後の画像を表示する。

16



17

```
void median( int x[ ][260], int y[ ][260] )
{
    int i, j, k, mi, mj, ni, nj, yn, min, med[10];
    for( j=2; j<=255; j++){ for( i=2; i<=255; i++){
        k = 1;
        for( mi=-1; mi<=1; mi++){ for( mj=-1; mj<=1; mj++){
            med[k] = x[i+mi][j+mj]; k++;
        }
        for( ni=1; ni<=8; ni++){ for( nj=8; nj >= ni; nj--){
            if( med[nj+1] < med[nj] ){
                min=med[nj+1]; med[nj+1]=med[nj]; med[nj]=min;
            }
        }
        y[ i ][ j ] = med[ 5 ];
    }
}
```

median.c
メディアン
フィルタ

18

```

k = 1;
for( mi=-1; mi<=1; mi++){ for( mj=-1; mj<=1; mj++){
    med[ k ] = x[ i+mi ][ j+mj ]; k++;
}
}

```

座標 [i][j] と、その近傍の画素カウント値を配列 med[1] ~ med[9] に代入している。

配列 med は、int med[10]; と宣言している。
int med[9]; でも良さそうに思えるが、

C言語では、配列数は 0 番目から数えているので med[9]; と宣言すると、med[0] から med[8] の 9 個の配列要素だけが宣言され、med[9] は宣言されない。

19

```

for ( ni=1; ni<=8; ni++){ for ( nj=8; nj >= ni; nj--){
    if ( med[nj+1] < med[nj] ){
        min=med[nj+1]; med[nj+1]=med[nj]; med[nj]=min;
    }
}
}

```

med[1] から med[9] の値を並べ替えている (ソート sort)。この for ループが終了すると、med[1] から med[9] の中の最小値が med[1] に、最大値が med[9] に代入されている。バブルソートのアルゴリズム Bubble sort algorithm コップ底の泡が、軽いものから先に順番に昇っていくような様子をプログラムで表現している。ソート法のアルゴリズムとしては最も非効率な方法だが単純で理解しやすい。

20

はじめに ni が 1 の値で nj のループに入る。
nj の値は 8 から 1 までの値に変化しながらループを回る。
最初は nj が 8 なので、if 文の中身を具体的に書くと

```

if ( med[ 9 ] < med[ 8 ] ){
    min=med[ 9 ]; med[ 9 ]=med[ 8 ]; med[ 8 ]=min;
}

```

もし med[9] が med[8] より小さければ med[9] と med[8] の値を入れ替える。変数の入れ替え作業を swap という。swap 作業には 第三の変数 (ここでは min) が必要になる。

ni が 1 の状態で、nj が 8 から 1 まで変化し終わると、med[1] の値は、med[1] から med[9] の最小値になっている。小さい値が泡のように上に昇っていくので バブルソート法。

21

次に ni が 2 の値で nj のループに入る。
nj の値は 8 から 2 までの値に変化しながらループを回る。
(med[1] は最小値が確定なのでループに入る必要がない。)

ni が 2 の状態で、nj が 8 から 2 まで変化し終わると、med[2] の値は、med[2] から med[9] の中の最小値になっている。

次に ni が 3 の値で nj のループに入る。
nj の値は 8 から 3 までの値に変化しながらループを回る。
(med[1]、med[2] は 1 番目、2 番目に小さい値が確定なのでループに入る必要がない。)

同様の操作を ni が 8 になるまで繰り返すと、med[1] に最小値、med[2] に 2 番目に小さい値、med[3] に 3 番目に小さい値、・・・、med[9] に 9 番目に小さい値が入る。中央値は med[5] なので、y[i][j] = med[5]; とする。

22

smoothing filter は、輪郭が不明瞭化し画像がぼやける。
median filter は smoothing filter より輪郭が保たれる。
median filter は、微細なノイズを除去する効果がある。
bone6noise は、10ピクセルおきに画素値 100 を入れた像。
median では、このノイズを完全に消すことができる。
smoothing ではノイズを完全には消せない。



23

高速フーリエ変換 (FFT : Fast Fourier Transform)

フーリエ変換を高速に計算するアルゴリズム。

1942年に Danielson と Lanczos が発明。

プログラムによるフーリエ変換は、波形データが離散的なので、基本式に現れる $e^{(2\pi f t) i}$ の項を、 W^{nk} と変形して巧みに解いている。

高速逆フーリエ変換 (IFFT : Inverse FFT) もほとんど同じアルゴリズムなので、両方可可能な関数が作れる。

FFT、IFFT におけるデータの制限としては、データ数が 2 の階乗でなければならない。

24

FFT2 Fast Fourier Transform

FFT2.exeを起動し、open fileから同フォルダ内にあるboneを選択する。

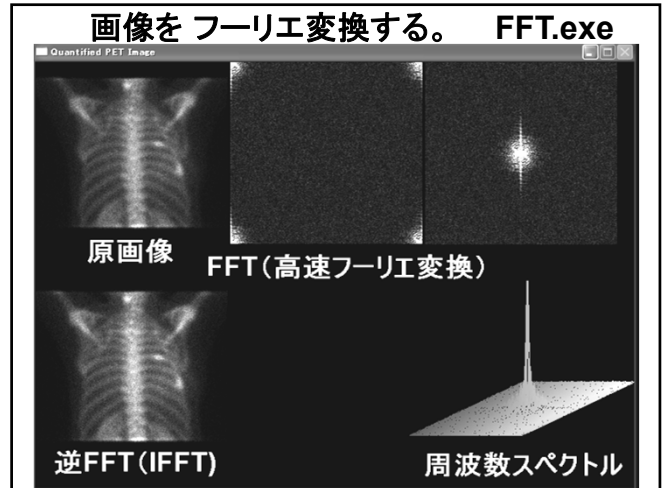
※bone: $^{99m}\text{Tc-MDP}$ のSPECT画像。骨シンチ。

FFTをクリックして2次元フーリエ変換を行い、replaceをクリックして、パワースペクトル画像を再配置する。

discoverをクリックすると、逆フーリエ変換された画像が表示される。

coverをクリックすると、右側の画像が消える。

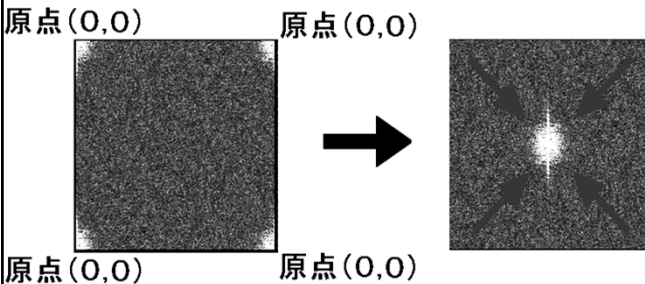
25



26

2次元FFTの出力である2次元周波数分布データは、4隅が原点(0,0) (= 直流成分) になっているのが面倒な点である。

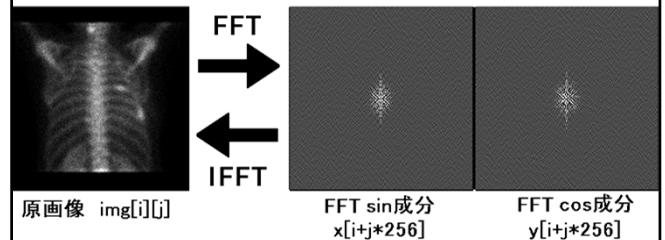
中心が原点になるように各々1/4領域を反転する



27

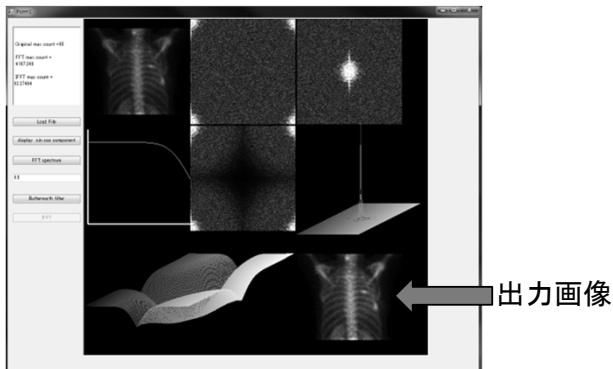
原画像をx[]に入れて2次元FFTを行うと、2次元周波数分布のsin成分がx[]に、cos成分がy[]に出力される。

x[]とy[]を、4隅が原点(0,0)になっている状態で2次元逆フーリエ変換(IFFT)の関数に入力すると、x[]に元に戻った像が出力される。

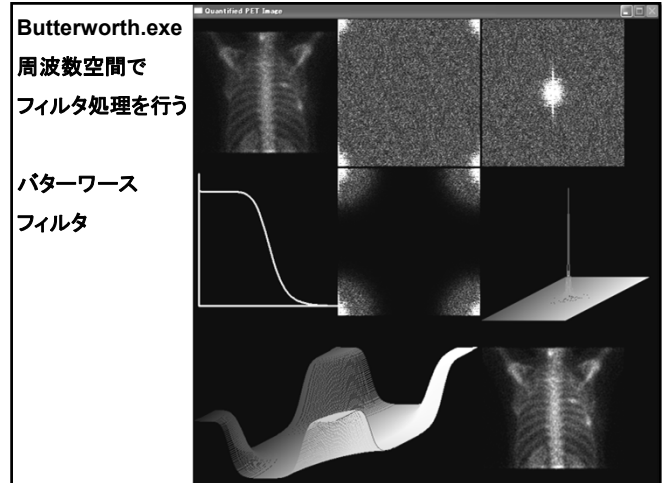


28

結果として、以下の様な画像が得られる。



29



30

テキストボックス内に0.5~0.01の任意の遮断周波数を設定する。

「Butterworth filter」ボタンを押し、設定した遮断周波数からバターワースフィルタを作成し、空間周波数フィルタ処理を行う。

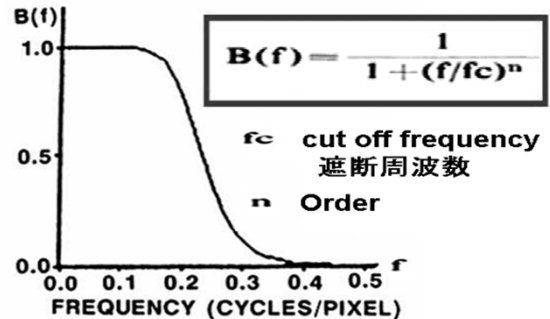
「IFFT」ボタンを押し、逆フーリエ変換を行い、フィルタ処理後の画像を表示する。

31

画像を周波数空間でフィルタ処理する。

バターワースフィルタ Butterworth filter

ノイズ成分が相対的に多い高周波成分を抑制。

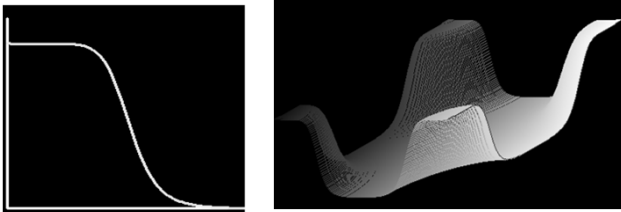


32

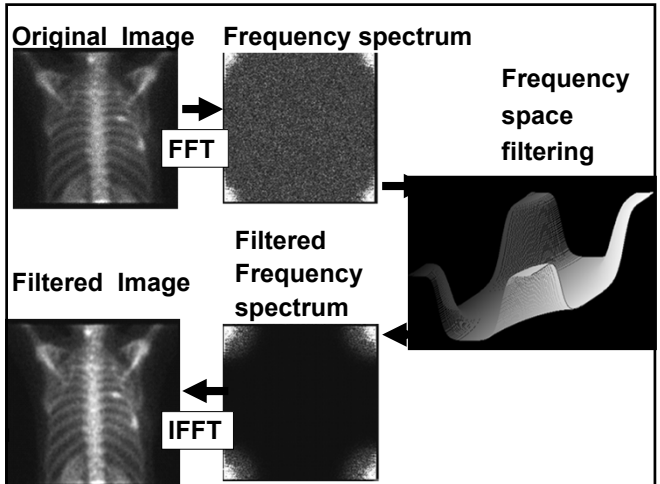
1次元の Butterworth filter を 2次元にしている。

2次元周波数分布のsin成分 $x[]$ 、cos成分 $y[]$ は4隅が原点(0,0) (= 直流成分) になっている。4隅近傍が低周波成分の領域。

したがって、2次元の Butterworth filter $B2[][]$ を、 $x[]$ 、 $y[]$ の4隅を残すように作成している。



33

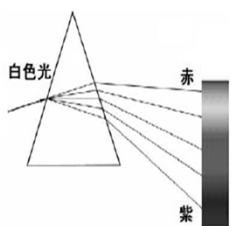


34



周波数空間を理解することは難しい印象を持つが、周波数空間や周波数分布を表現しているもの一般にも結構ある。

たとえば楽譜は、音楽の周波数分布、曲のフーリエ変換とも解釈できる。五線譜の下側の音符は低音、低周波数成分を表し、上側の音符は高音、高周波成分を示している。



虹は白色光に含まれる色の周波数分布を示している。雨粒中で屈折しやすい高周波の紫色成分から、屈折しにくい低周波の赤色成分まで、周波数の順序で色が並んでいる。

35