Среднее значение как сглаживание.

Для того чтобы получить сглаженное значение складываем ближайшие в радиусе r , и усредняем:

$$g_{avg}(x) = \frac{1}{2\mathrm{r}} \int_{t-r}^{t+r} g(t)\,dt. \qquad (1)$$

Для дискретной функции $b[i]$:

$$b_{avg}[i] = \frac{1}{2\mathrm{r}+1} \sum_{j=i-r}^{i+r} b[j]. \qquad (2)$$

Дискретная конволюция или свёртка.

Здесь $b[i-j]$ задаёт вес в сумме для семпла, отстоящего на j от значения семпла i :

$$(a*b)[i]=\sum_{j=-r}^{r} a[j]b[i-j]. \tag{3}$$

Для непрерывной функции это:

$$(f*g)(x)=\int_{-\infty}^{\infty} f(t)\,g(x-t)\,dt. \tag{4}$$

Если радиус функции $g(x)$ ограничен r , тогда:

$$(f*g)(x)=\int_{-r}^{r} f(t)\,g(x-t)\,dt. \tag{5}$$

Псевдокод дискретной конволюции.

```
int
convolve(int a[], int b[], int r, int i)
{
   int j, s;

   s = 0;

   for (j = -r; j <= r; j++) {
      s += a[j]*b[i-j];
   }

   return s;
}
```

Связь между дискретным и непрерывным сигналом:

$$a[i] = f(i). \tag{6}$$

Мы просто взяли значения непрерывной функции в дискретные моменты времени. Иногда в литературе встречается такая запись с периодом дискретизации T . Тогда (6) выглядит так:

$$a[i] = f(Ti). \tag{7}$$

Восстановление или фильтрация сигнала $a[i]$ фильтром $f(x)$:

$$(a * f)(x) = \sum_i a[i] f(x - i). \tag{8}$$

На практике фильтр всегда конечный.

Если фильтр ограничен радиусом r тогда фильтрация, или восстановление, сигнала $a[i]$ фильтром $f(x)$:

$$(a*f)(x)=\sum_{i=x-r}^{x+r} a[i]f(x-i). \qquad (9)$$

Сумма исключает точки где $x-i$ больше или равно радиусу r .

Псевдокод фильтрации (восстановления).

```
/* Здесь float (*filter)(float t) — функция фильтра. */

float
reconstr(int a[], float (*filter)(float t), int r, float x)
{
   int i;
   float s, t;

   s = 0.0;

   for (i = -r; i <= r; i++) {
      t = x — i;
      if (fabs(t) < r)
         s += a[i] * (*filter)(t);
   }

   return s;
}
```

2-х мерная конволюция для сигналов.

$$(a*b)[i,j]=\sum_{i'}\sum_{j'} a[i',j']\,b[i-i',j-j']. \qquad (10)$$

Если же b конечный фильтр радиуса r :

$$(a*b)[i,j]=\sum_{i'=i-r}^{i+r}\sum_{j'=j-r}^{j+r} a[i',j']\,b[i-i',j-j']. \qquad (11)$$

Псевдокод дискретной конволюции для двухмерных сигналов.

```
int
convolve2d(int a[][], int b[][], int r, int i, int j)
{
   int k, l, s;

   s = 0;

   for (k = -r; k <= r; k++) {
      for (l = -r; l <= r; l++) {
         s += a[k][l] * b[i-k][j-l];
      }
   }

   return s;
}
```

Гауссов фильтр — самый гладкий.

$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{12}$$

Двухмерный Гауссов фильтр это $f_g(x) f_g(y)$:

$$f_{2,g}(x, y) = \frac{1}{2\pi} (e^{-x^2/2} e^{-y^2/2}) = \frac{1}{2\pi} e^{-r^2/2}. \tag{13}$$

Полученный фильтр является разделяемым фильтром. Это очень полезное свойство, которое позволит нам быстрее вычислять конволюцию.

Разделяемые фильтры.

Вспомним определение дискретной конволюции.

$$(a*b)[i,j]=\sum_{i'=i-r}^{i+r}\sum_{j'=j-r}^{j+r} a[i',j']b[i-i',j-j'].$$

Так как наш фильтр это $f_g(x)f_g(y)$, то в дискретной форме он эквивалентен $a[x]a[y]$.

Подставив его в определение дискретной конволюции получим:

$$(a*b)[i,j]=\sum_{i'=i-r}^{i+r}\sum_{j'=j-r}^{j+r} a[i']a[j']b[i-i',j-j']. \qquad (14)$$

Из второй суммы можно смело «вытащить» $a[x]$ из цикла, т.к. она зависит только от i' .

В итоге, получим такую формулу:

$$(a*b)[i,j]=\sum_{i'=i-r}^{i+r} a[i']\sum_{j'=j-r}^{j+r} a[j']b[i-i',j-j']. \quad (15)$$

Сумма справа может быть представлена как:

$$S[k]=\sum_{j}{}' a[j']b[k,j-j']. \quad (16)$$

Тогда:

$$(a*b)[i,j]=\sum_{i'=i-r}^{i+r} a[i']S[i-'i] \quad (17)$$

Сейчас мы попробуем уменьшить вычислительную сложность приблизительно от $O(r^2)$ до $O(r)$ последовательной оптимизацией кода.

Псевдокод конволюции без оптимизации.

```
void
filter_image(float img[][], float out[][], int w, int h,
             float (*flt)(int, int), int r)
{
   int x, y;
   int i, j;

   for (y = 0; y < h; y++) {
      for (x = 0; x < w; x++) {
         for (i = -r; i <= r; i++) {
            out[y][x] = 0;
            for (j = -r; j <= r; j++) {
               out[y][x] += flt(i,j)*img[y-i][x-j];
            }
         }
      }
   }
}
```

Так как наш фильтр разделяемый, вынесем $flt(i)$ из цикла.
Функция фильтра симметрична, аргумент — радиус.

```
void
filter_image2(float img[][], float out[][], int w, int h,
              float (*flt)(int), int r)
{
   int x, y;
   int i, j;

   for (y = 0; y < h; y++) {
      for (x = 0; x < w; x++) {
         for (i = -r; i <= r; i++) {
            out[y][x] = 0;
            for (j = -r; j <= r; j++) {
               out[y][x] += flt(j)*img[y-i][x-j];
            }
            out[y][x] *= flt(i);
         }
      }
   }
}
```

Проведя оптимизацию внутри циклов а также уменьшив размер изображения на радиус фильтра (дабы избежать краевых случаев для простоты) получим:

```
void
filter_image2d(float img[][], float out[][], int w, int h,
               float (*flt)(int), int r)
{
    int x, y;
    int i;
    float s[w];

    for (y = r; y < h-r; y++) {
        for (x = 0; x < w; x++) {
            s[x] = 0;
            for (i = -r; i <= r; i++)
                s[x] += flt(i)*img[y-i][x];
        }
        for (x = r; x < w-r; x++) {
            out[y][x] = 0;
            for (i = -r; i <= r; i++)
                out[y][x] += flt(i)*s[x-i];
        }
    }
}
```

Обратный фильтр — резкость.

Если I изображение, α параметр, d единичный импульс, то:

$$I_{sharp} = (1+\alpha) I - \alpha (f_{g,\sigma} * I) = ((1+\alpha) d - \alpha f_{g,\sigma}) * I, \quad (18)$$

$$I_{sharp} = f_{sharp}(\alpha, \sigma) * I. \quad (19)$$