Расширенный биномиальный фильтр для быстрого размытия по Гауссу

Абстрактный

Алгоритм расширенного биномиального фильтра — это очень простой и эффективный алгоритм размытия по Гауссу, в котором время обработки каждого пикселя не зависит от радиуса размытия.

Размытие по Гауссу — широко используемый фильтр для многих эффектов, особенно для обработки изображений. Важно иметь быстрый и простой алгоритм вычислений. Время выполнения большинства алгоритмов расчета размытия по Гауссу, таких как биномиальная последовательность, пропорционально радиусу размытия r. Этот недостаток сложности O(r) на пиксель делает такой алгоритм очень медленным для больших радиусов размытия.

Расширенный биномиальный фильтр является аппроксимацией обычного биномиального фильтра с постоянной сложностью O(1), что делает время выполнения на пиксель независимым от радиуса размытия.

Точность аппроксимации выбирается степенью аппроксимации.

Ключевые особенности

• Постоянная сложность O(1) на пиксель, независимая от радиуса размытия. •

Минимальные показания пикселей, также не зависящие от радиуса размытия. •

Вычисления так же просты (и быстры), как и размытие

прямоугольника. • Выбор адекватной аппроксимации с желаемой точностью

Саммери Алгоритм

использует то преимущество, что интеграл от постоянной функции очень легко вычислить: это всего лишь одно умножение. Двойной интеграл от постоянной функции вычисляется легко: нужно лишь сложить разности в начале и конце. Этот метод можно распространить на любое количество последовательных интегралов постоянной функции.

Функция сначала выводится n раз, а затем упрощается до последовательности ступенчатых функций. Теперь алгоритм берет эти простые пошаговые функции и снова интегрирует их n раз, чтобы получить аппроксимацию исходной функции.

Алгоритм также использует то преимущество, что вычисляются последовательные пикели. Таким образом, вместо повторного суммирования значений для каждого пикселя добавляются только коэффициенты. Эти дополнения выполняются для всех дискретных производных. Таким образом, нужно будет сделать всего несколько дополнений на пиксель. В своей простейшей форме первой степени алгоритм выполняет только размытие прямоугольника. Но приближение третьей степени достаточно точно для большинства 8-битных изображений.

Этот принцип аппроксимации не ограничивается размытием по Гауссу, но может использоваться и для других функций фильтра.

Этот алгоритм особенно полезен при больших радиусах размытия.

Ключевые слова: размытие по Гауссу, эффективный алгоритм, биномиальный фильтр, аппроксимация, свертка, градиент.

Содержание

Сокращения

КАС	Система компьютерной алгебры
ДПФ	Дискретное преобразование Фурье
БПФ	Быстрое преобразование Фурье
ДЛЯ	Конечная импульсная характеристика
БИХ	Бесконечный импульсный отклик
Пиксель	Элемент изображения
[illegible]	Среднеквадратичный
СК ИПСМ	Обработка изображений с разделенным ядром с использованием конечных автоматов
УСМ	Нерезкая маскировка

1. Математическая основа. В этой главе рассматриваются

математические основы алгоритма. Если вас интересует только алгоритм, вы можете пропустить эту главу, но она поможет лучше понять теорию работы

Для математического описания изображение мыслится как непрерывная аналоговая функция. Изображение состоит из пространственной области S, которая представляет собой диапазон возможных положений x и y изображения, и области диапазона P, которая представляет собой диапазон возможных значений пикселей: p = f (x,y).

1.1. Необычные функции

Для аппроксимации к усочной функции необходима ступенчатая функция. Ступенчатая функция Хевисайда определена как:

$$\Theta(x) = \begin{cases} 0, & x < 0 \\ \frac{1}{2}, & x = 0 \\ 1, & x > 0 \end{cases} \qquad (1)$$

Ступенчатая функция Хевисайда представляет собой интеграл дельта-функции Дирака: $d\Theta = \delta$.

Прямоугольная функция определяется как $rect(x) = \Theta\left(x + \frac{1}{2}\right) - \Theta\left(x - \frac{1}{2}\right) = \sum_{i=0}^{1} \frac{(-1)^i}{2} \frac{2x + 1 - 2i}{|2x + 1 - 2i|}$.

Еще одна не столь распространенная необходимая функция — это абсолютное значение с его производной и интегралом:

$$\frac{d}{dx}|x| = sgn(x) = 2\Theta(x) - 1$$

$$\int |x|\,dx = \frac{x|x|}{2} + C$$

Значение ступенчатой функции Хевисайда в точке x = 0 не имеет значения для этого приложения.

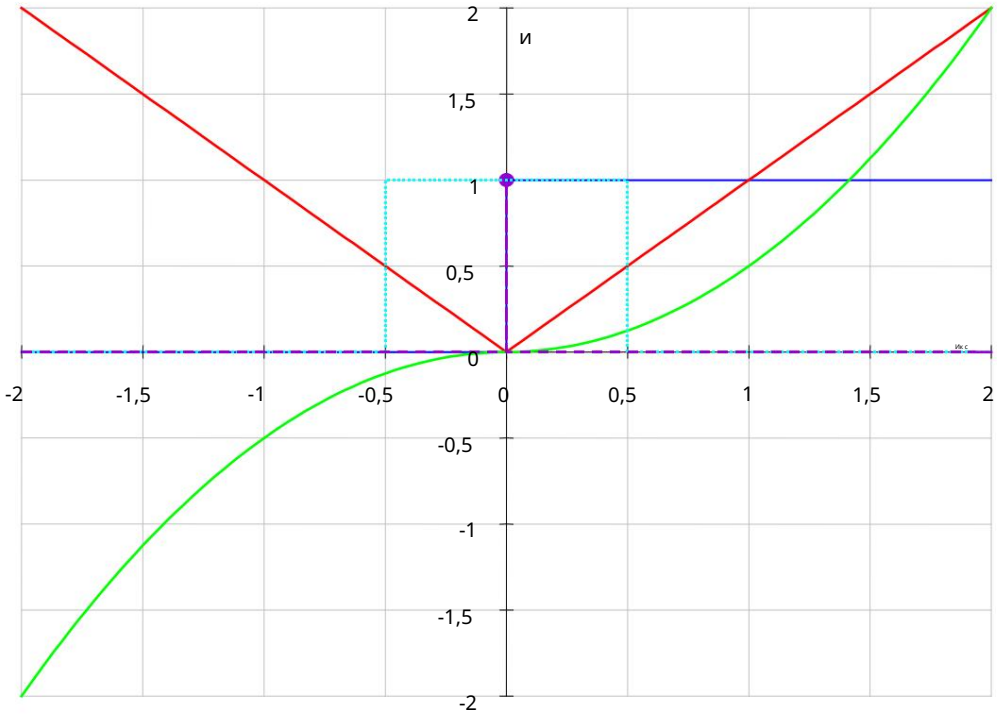


Рисунок 1 Необычные функции

а. Дельта-функция Дирака $\delta(x)$

б. Прямоугольная функция

в. Ступенчатая функция Хевисайда: $\Theta(x)$

d. Абсолютное значение: $|x|$

е. Интегрирование абсолютного значения: $\frac{x|x|}{2}$

Функция пола x также необходима для дискретных последовательностей.

1.2. Размытие по Гауссу

Размытие по Гауссу использует функцию Гаусса для расчета преобразования. Это приводит к очень гармоничным и плавным результатам. Для каждого нового пикселя устанавливает средневзвешенное значение окрестности этого пикселя.

В математике это можно выразить как свертку (нем. Faltung) функции изображения f(x) с помощью функции размытия g(x): (

ж з)(х) (f ⊗ г)(х) (s) f (s) г рамм x - s d s .

Коэффициенты ядра матричной свертки выбираются так, чтобы аппроксимировать распределение Гаусса. Радиус размытия определяет размер ядра.

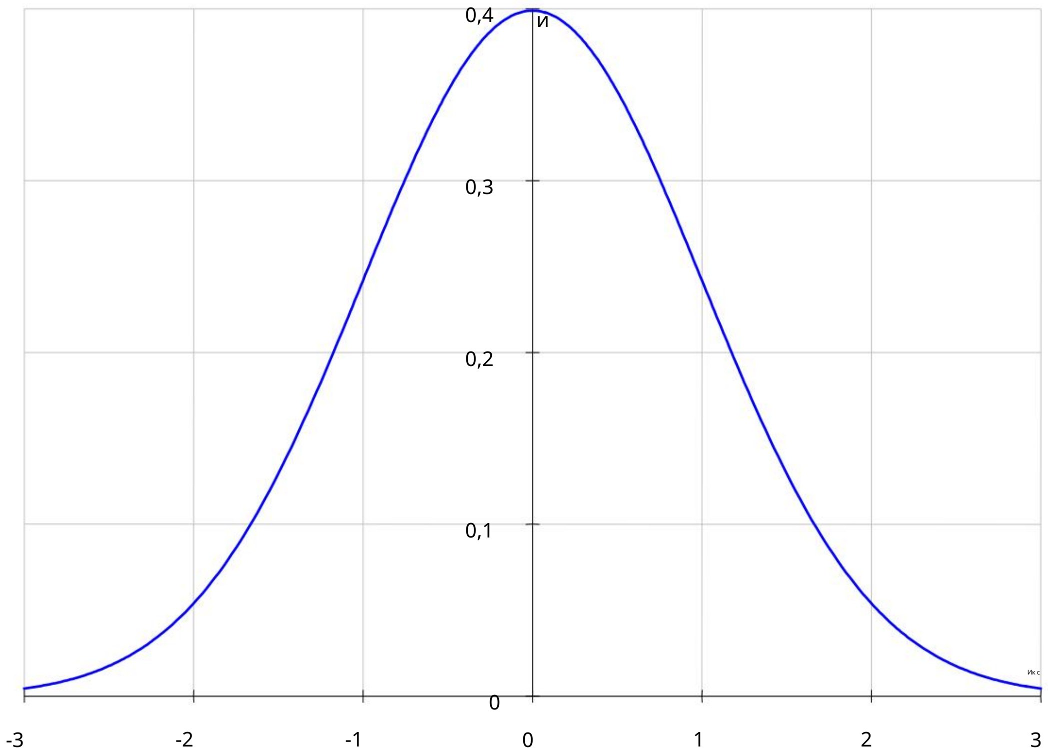


Рисунок 2 Функция Гаусса для σ = 1

$$и "=" \frac{1}{\sqrt{2Пи}} Это^{-\frac{икс^2}{2}}$$

Уравнение функции размытия по Гауссу: g

$$g_{PC,}(Икс) "=" \frac{1}{п \sqrt{2Пи}} Это^{-\frac{(Икс - м)^2}{2п^2}} .$$

Параметр μ — это ожидаемое значение, а σ — стандартное отклонение, которое представляет собой радиус размытия в случае размытия по Гауссу.

Для облегчения следующих расчетов используется стандартное нормальное отклонение, где σ установлено равным единице, а μ установлено равным нулю. Эта простая функция показана на рисунке 2:

$$г(Икс) "=" \frac{1}{\sqrt{2Пи}} Это^{-\frac{Икс^2}{2}} \qquad (2)$$

Стандартное отклонение можно рассчитать на основе стандартного нормального отклонения: $g_{PC,}(Икс) "=" \frac{1}{п} г\left(\frac{Икс - м}{п}\right)$.

Упрощение учитывается, если оно влияет на расчет.

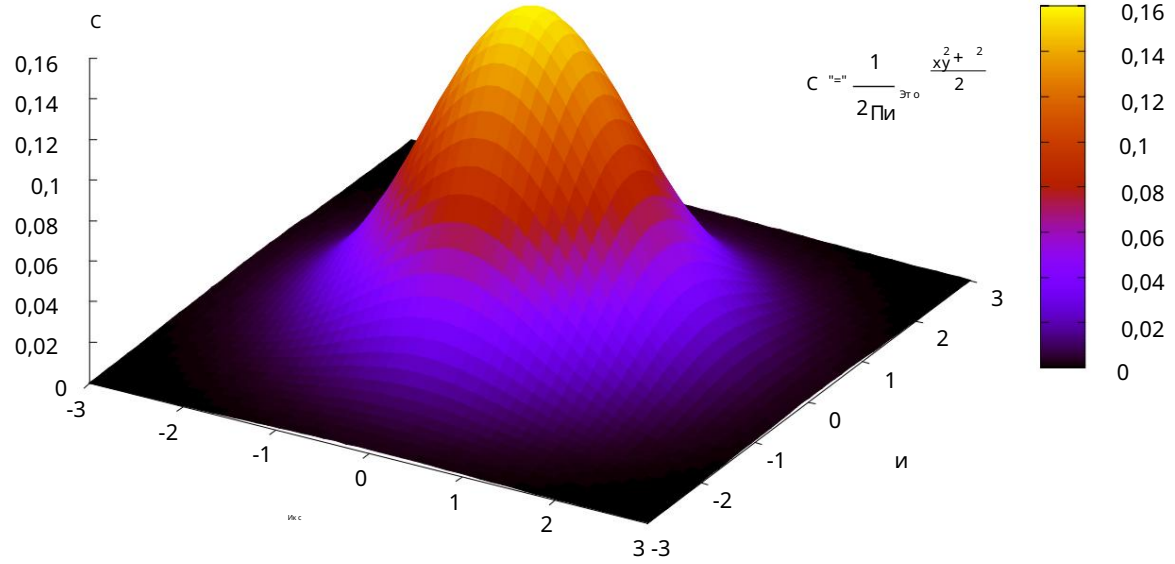
Функции фильтра не ограничиваются одним измерением. Для изображений необходима двумерная функция. Если функция фильтра является линейно разделимой, n- мерная функция фильтра может быть вычислена с помощью n одномерных

функции: $()г(н Икс "=" г(х_1, х_2, К, х_н) "=" \prod_{(я) 1 =}^{н} г(Икс_я)$.

Функция фильтра Гаусса двух измерений на рисунке 3 равна $g(,) ху "=" \frac{1}{2пс^2} Это^{-\frac{x^2 + y^2}{2п^2}} "=" ()() х у$.

Можно выбрать разное размытие для направлений x и y : $g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)}$

или даже выполнить размытие для любого направления θ: $g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{(x^2\cos^2\theta+y^2\sin^2\theta)\sigma_x^2+(x^2\sin^2\theta+y^2\cos^2\theta)\sigma_y^2+xy\sin 2\theta(\sigma_x^2-\sigma_y^2)}{2\sigma_x^2\sigma_y^2}}$.



Изображение 3 Колоколообразная форма Гаусса в 3D

Функциональная область $G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} dx = 1$.

Для двух измерений объем под функцией равен $g(x,y)dxdy = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} dxdy = 1$.

Ошибка аппроксимации может быть измерена с помощью среднеквадратического значения (RMS) между точными значениями g(x) и аппроксимация функции b(x): $\sigma = \sqrt{(g(x)-b(x))^2}$. Ошибка максимальна = 1 для одного приближение b(x) = 0. Поскольку изображения двумерный функция Гаусса линейно разделима, двумерная ошибка может быть измерена с помощью $\sigma = \sqrt{dxdy(g(x)g(y)-b(x)b(y))^2}$.

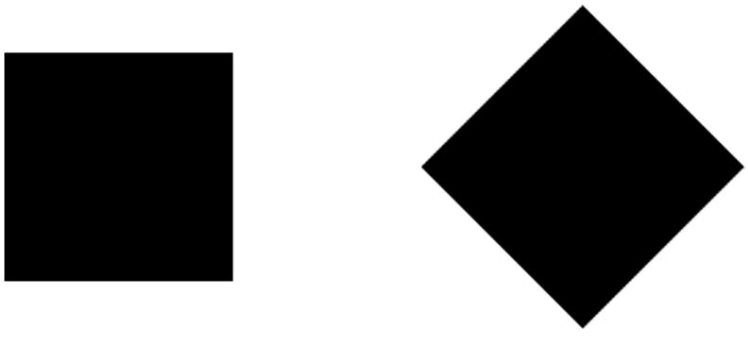


Рисунок 4. Пример изображения без размытия.

На рисунке 5 показано размытие по Гауссу изображения 4 с двумя квадратами, один из которых повернут на 45 градусов. Оба выглядят одинаково и очень плавно, как и ожидалось от размытия.

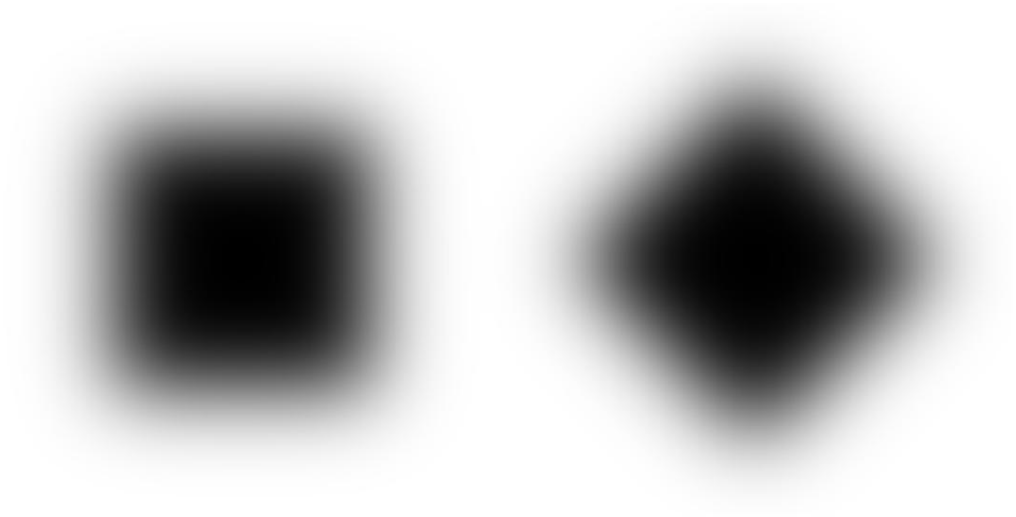


Рисунок 5. Размытое изображение по Гауссу.

Стандартное отклонение определяется как

$$\sigma = \sqrt{\int (x-\mu)^2 b(x)\,dx}, \quad (3)$$

где b(x) обозначает функцию плотности.

Ожидаемая стоимость определяется как

$$\mu = \int x\, b(x)\,dx \quad . \quad (4)$$

Стандартное отклонение и ожидаемое значение определяются уравнением (3) и (4) для функции плотности функция Гаусса, поскольку

$$\int (x-\mu)^2 g_{\sigma,\mu}(x)\,dx = \int \frac{(x-\mu)^2}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx = \sigma^2 \text{ и } \int x\, g_{\sigma,\mu}(x)\,dx = \int \frac{x}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx = \mu.$$

1.3. Размытие коробки

Вычисление функции Гаусса является дорогостоящим. Полиномиальные аппроксимации обещают лучшую производительность. Вопрос в том, какой полином является лучшим компромиссом между точностью и производительностью?

Размытие рамки — первое упрощение. Для 1-й степени приближения функция Гаусса заменяется константой.

Размытие поля просто суммирует значения вокруг квадрата. Это легко сделать, но результат не такой гладкий, как размытие по Гауссу.

Функция фильтра размытия поля (прямоугольная функция) — $b1. \quad (x) = \Theta\left(x + \frac{c}{2}\right) - \Theta\left(x - \frac{c}{2}\right)$,

где Θ обозначает ступенчатую функцию Хевисайда, c — ширина, а a — высота ящика.

Одним из условий определения константа a является то, что площадь под функцией должна быть одинаковой для аппроксимации и функции Гаусса:

$$\int_0^{c/2} b(x)\,dx = \int_0^{c/2} a\,dx = 1, \text{ поэтому } a = \frac{1}{c/2} = \frac{2}{c}.$$

Второе условие состоит в том, что стандартное нормальное отклонение для аппроксимации и функции Гаусса согласно уравнению(2) одинаково:

$$\sigma^2 = \int_{-c/2}^{c/2} x^2 b(x)\,dx = \frac{2}{c}\int_{-c/2}^{c/2} x^2\,dx = \frac{c^2}{12}, \text{ поэтому } \sigma = \frac{c}{2\sqrt{3}} \text{ и } a = \frac{1}{2\sqrt{3}\sigma}.$$

Функция размытия поля с этими определениями:

$$b1(\text{и}_{\text{х}}) \text{ "=" } \frac{\Theta + \left(\Theta\sqrt{3}\right)\text{х} \quad \left(\text{и}_{\text{х}\text{с}} \quad \sqrt{3}\right)}{2\,3\sqrt{}} \tag{5}$$

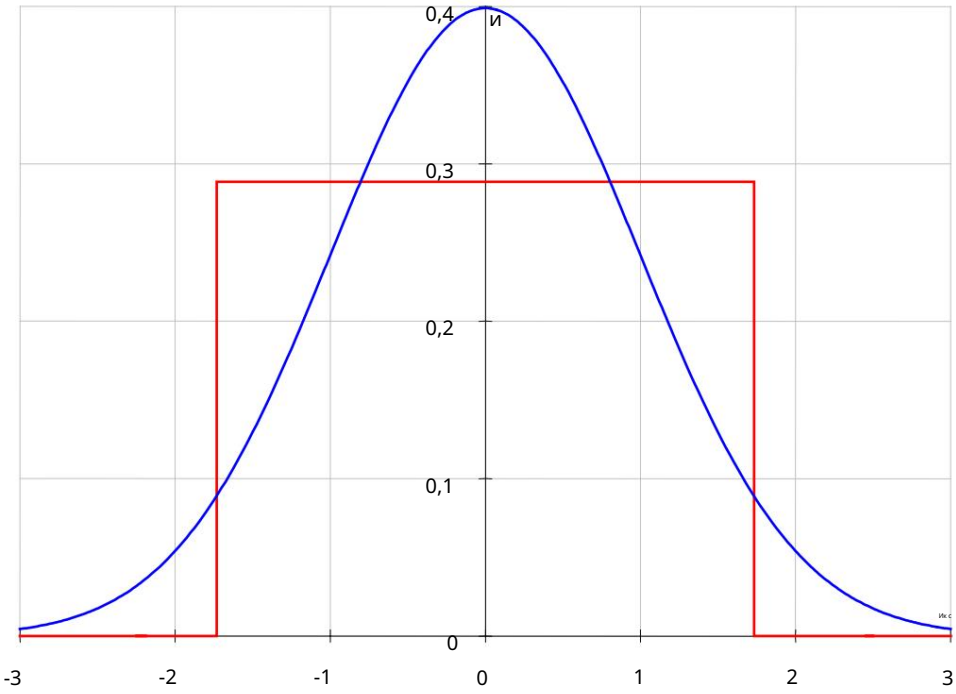


Рисунок 6: Функция размытия рамки

а. Размытие коробки

б. Размытие по Гауссу

Когда площадь функции Гаусса определяется как максимальная ошибка, ошибка 1-й степени приближение $\text{и}_1 \text{ "=" } \sqrt{\text{ }(\text{ })\text{г}(\text{ })\text{б}(\text{ })\text{х}\,\text{д}\text{х}^2}$ "=" 0,2036969520 . Двумерная ошибка составляет 0,1511405983.

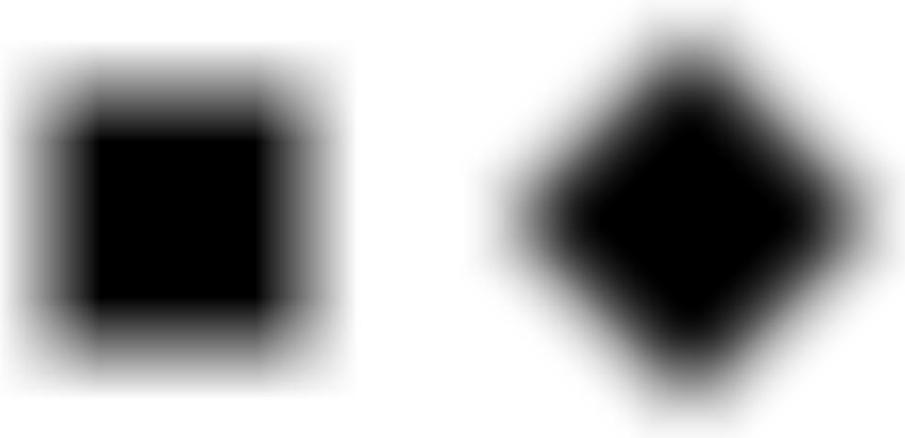


Рисунок 7: Изображение с размытием рамки

Квадрат размытия прямоугольника все еще можно увидеть на рисунке 7. Кроме того, два размытых квадрата не выглядят такой же.

Преимущество размытия по рамке — это его скорость . Поскольку у размытие применяется к каждому значению, две суммы соседних значений практически идентичны Вместо расчета отдельных сумм добавляются только разницы предыдущей суммой. Таким образом, время выполнения каждого значения является постоянным и не зависит от радиуса размытия.

Можно ли каким-то образом объединить размытие по квадрату и размытие по Гауссу, чтобы получить выгоду от обоих алгоритмов?

1.4. Производные Гаусса

Размытие прямоугольника — это своего рода упрощение формы колокола Гаусса. Вместо функции добавляются постоянные значения. Производная коробчатой функции представляет собой две дельта-функции Дирака с пиками в начале и в конце. Интегрирование просто выполняется во время итерации по всем значениям. Это можно использовать для лучшей адаптации аппроксимации к форме колокола. Ступенчатая функция снова интегрируется для построения функции наклона. Две интеграции почти так же просты, как одна, поскольку к у значения необходимо добавлять только при изменениях.

Высшая степень производной состоит только из констант. Он рассчитывается путем сложения соответствующих значений пикселей.

Фильтрующая функция степени n аппроксимируется n+1 соседними полиномиальными функциями степени n. Все производные двух полиномов в соседних точках равны(аналогично сплайнам), за исключением последней производной, которая состоит из ступенчатых функций. Высота и ширина ступенчатых функций определяют функцию фильтра.

Преимущество этого приближения состоит в том, что n-й вывод состоит только из постоянных значений, констант фильтра. Интегрирование просто выполняется путем добавления новых значений в соседних точках двух полиномов для каждой производной. Это делает время расчета независимым от радиуса фильтра.

1,5. Линейная аппроксимация.

Колоколообразная форма Гаусса адаптируется с помощью треугольной функции.

Вторая степень приближения функции Гаусса равна $b(x) = a(|x + c| - 2|x| + |x - c|)$.

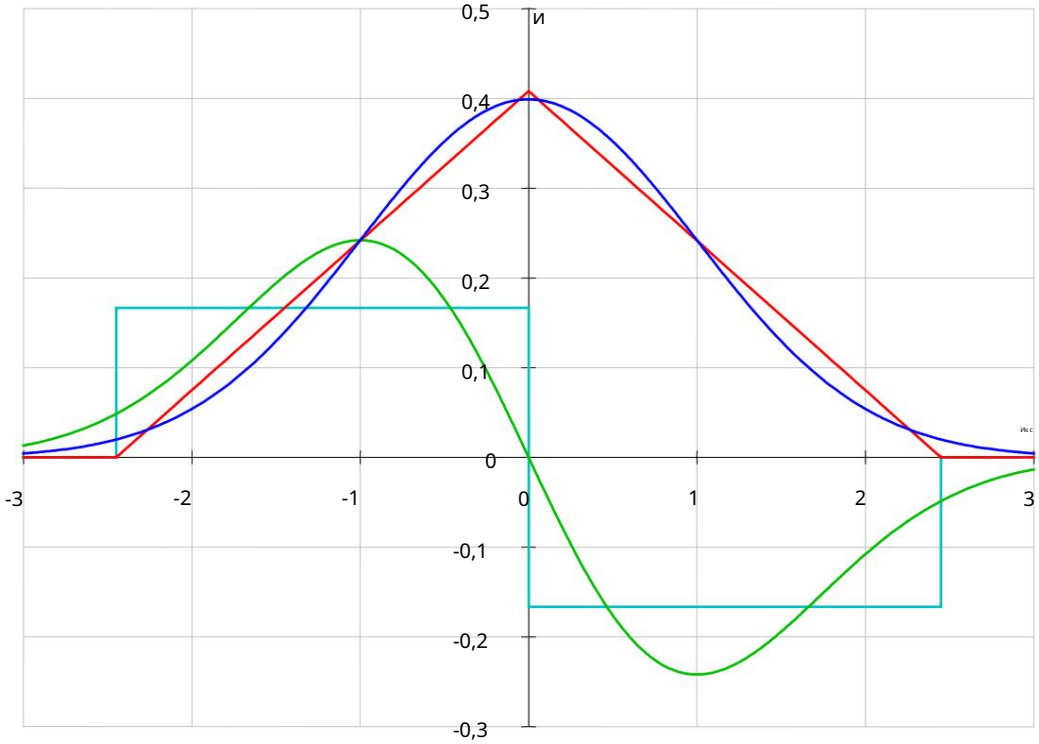


Рисунок 8: Линейная аппроксимация

а. $y = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

б. y = b2(x)

в. $y = \frac{x}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

д. y = s2(x)

Первые условия для констант a и c такие же, как и раньше:

$$b(x) dx = a(x + c - 2x + x - c) dx = 1 = 1.$$

Второе условие – снова то же значение отклонения:

$$\sigma^2 = x^2 b(x) dx = \frac{x^2}{2c^2}(x + c - 2x + x - c) dx = \frac{c^2}{6} = 1. \text{ Итак, } a = 1/12 \text{ и } c = 6.$$

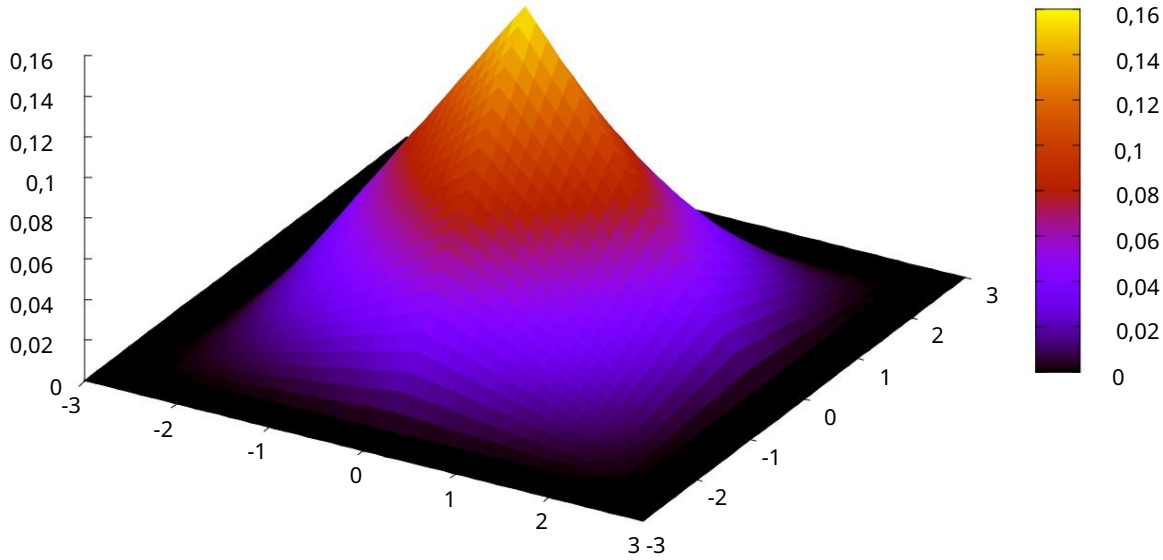
Таким образом, вторая степень приближения функции Гаусса равна:

$$б(2)_{Икс} = \frac{\left| \sqrt{6}\,2 + x \right| \left| \sqrt{6} \right|}{12} \quad (6)$$

Ступенчатая функция второго приближения представляет собой вывод $c_{2\,Икс} = \frac{БД_{2}(Икс)}{Д_{Икс}}$:

$$c2_{(Икс)} = \frac{\Theta + (Икс\ \Theta + \sqrt{\Theta})\,6\,2\ (Икс)\ (Икс - \sqrt{6})}{6} \quad (7)$$

Погрешность 2-й степени приближения составляет $и_{2} = \sqrt{\int ()\,г(Икс)\,б()_{2}\,x\,д\,x^{2}}$ = 0,04652434331 . Двумерная ошибка равна 0,03530803161.



Изображение 9: Размытие треугольника

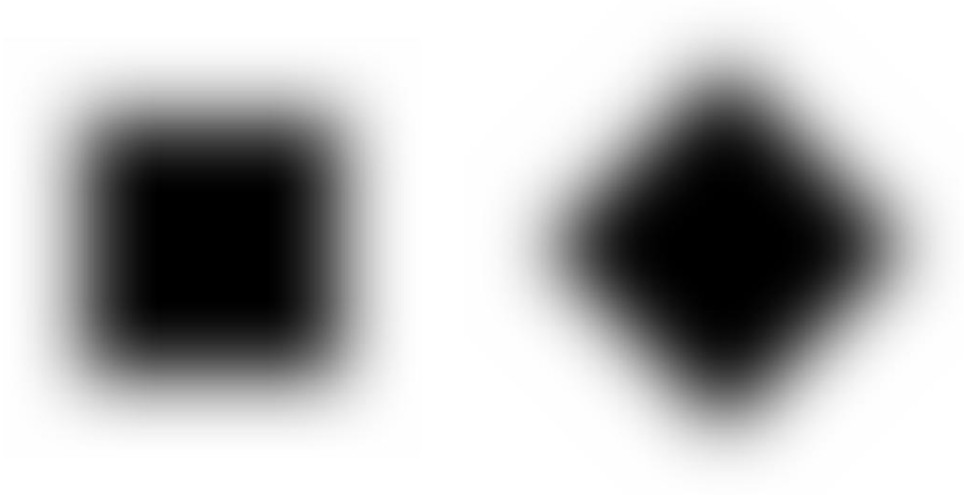


Рисунок 10: Размытое изображение треугольника.

Изображение на рисунке 10 предполагает хорошее приближение к размытию по Гауссу, подтвержденное расчетами.
Применение аппроксимации к тестовому изображению показывает заметной разницы с размытием по Гауссу.
Максимальная погрешность в несколько процентов видна в графическом редакторе с высоким усилением пикселей. Это соответствует стандартному 8-битному редактированию изображений.

1.6. Квадратное приближение

Процесс аппроксимации можно продолжить, если вторая степень недостаточно точна. Колоколообразная функция Гаусса дифференцируется и аппроксимируется треугольниками. Затем треугольники интегрируются, как это делалось раньше.

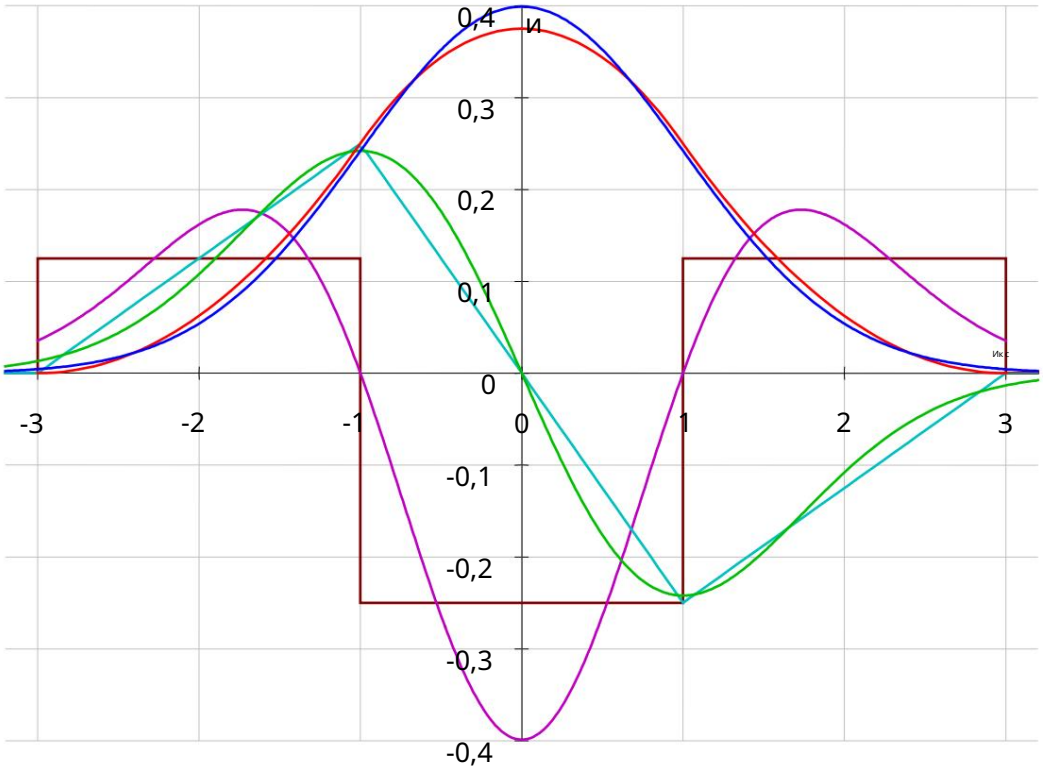


Рисунок 11:

Приближение третьей степени уже. Это $\varphi = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

б. y = b3(x)

против. 1-я производная d. y = b3'(x)

Это. 2-я производная

ф. y = s3(x)

Для приближения третьей степени вторая производная функции Гаусса на рисунке 11e (пурпурный график) упрощается до ступенчатых функций. Для простого приближения ширина трех ступенек (коричневый график) одинакова. Максимум отрицательного шага в два раза превышает максимум положительного шага, поскольку у первая производная аппроксимации в точке x = 0 должна быть равна нулю(площадь коричневого графика в целом должна быть равна нулю). Это приводит к следующей

ступенчатой функции $s_3(x) = a(\Theta(x+2c) - \Theta(x+c) - \Theta(x-c) + \Theta(x-2c))$

c — ширина ступеней, а — амплитуда. Двукратная первообразная s3(x) составляет

$$b_3(x) = \frac{a}{16}\left((x+2c)^2 - 3(x+c)^2\left|x+c\right| + 3(x-c)^2\left|x-c\right| - (x-2c)^2\left|x-2c\right|\right)$$

Условия для констант такие же, как и для предыдущих приближений. Площадь под функцией должна быть одинаковой для аппроксимации и функции Гаусса:

$$b_3(\infty) = \int_{-3c/2}^{3c/2} b_3(x)\,dx = \int_{-3c/2}^{3c/2} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,dx = 1.$$

Итак, первое условие приводит к переменному току $y^3 = 1$. Для второго условия стандартное отклонение должно быть равно единице:

$$\int_{-3c/2}^{3c/2} x^2 b_3(x)\,dx = \frac{c^4}{16} = 1, \text{ поэтому } a = 1/8 \text{ и } c = 2.$$

Таким образом, ступенчатая функция третьего приближения равна

$$\frac{1}{8}\left(\Theta(x+4) - 3\Theta(x+2) + 3\Theta(x) - \Theta(x-2)\right) \quad (8)$$

и $b_3(x) = \frac{1}{32}\left((x+4)^3\left|x+4\right| - 4(x+2)^3\left|x+2\right| + 6x^3|x| - 4(x-2)^3\left|x-2\right| + (x-4)^3\left|x-4\right|\right)$.

Погрешность 3-й степени приближения составляет и3 "=" √(( ) г( ) б( ) 3 х д х 2) "=" 0,02531428515 . Два

размерная ошибка равна 0,01954370265.

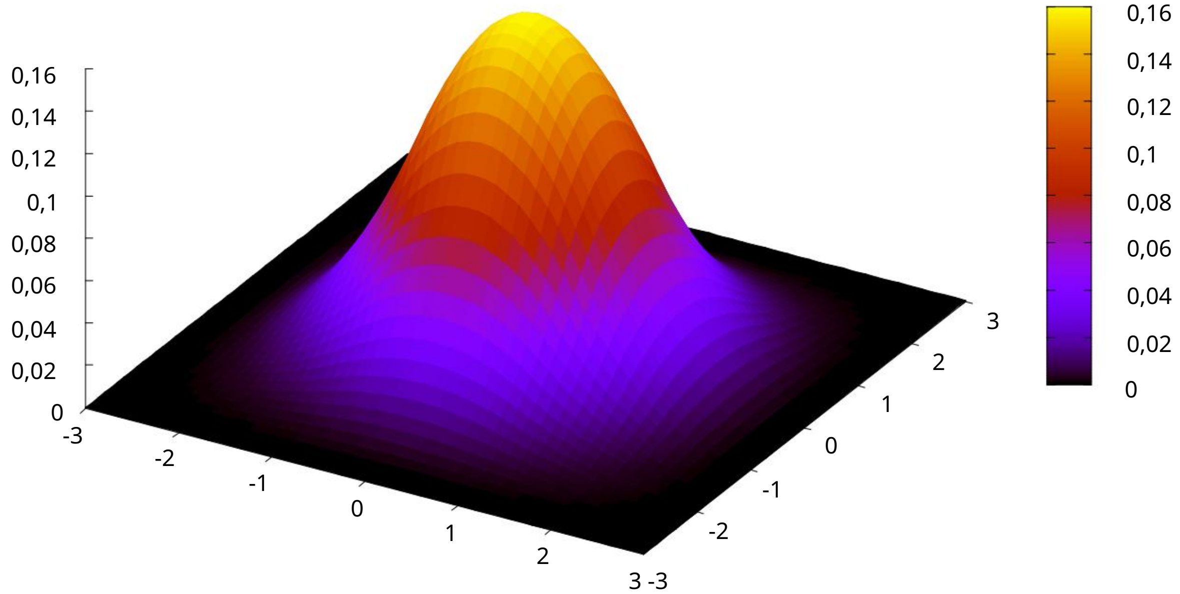


Рисунок 12: Аппроксимация квадрата

1.7. Высшая степень приближения

Тот же подход, что и раньше, можно использовать для расчета четвертой степени приближения. Формула для ступенчатой функции 4-й степени выводится при том же условии, что и раньше: равная ширина шага. (

s4()x = a(Θ x + 2c)+ Θ x - 2c -)4Θ x (+ c - 4)Θ x - (c + 6Θ) x ( ))

6 = (1)2 4 а ((Икс + 2в)2 2в 2в + ( ) Икс 2 +2c4 ( ) Икс с 2 + c4 ( ) Икс с 2 Икс c6 x x 2)

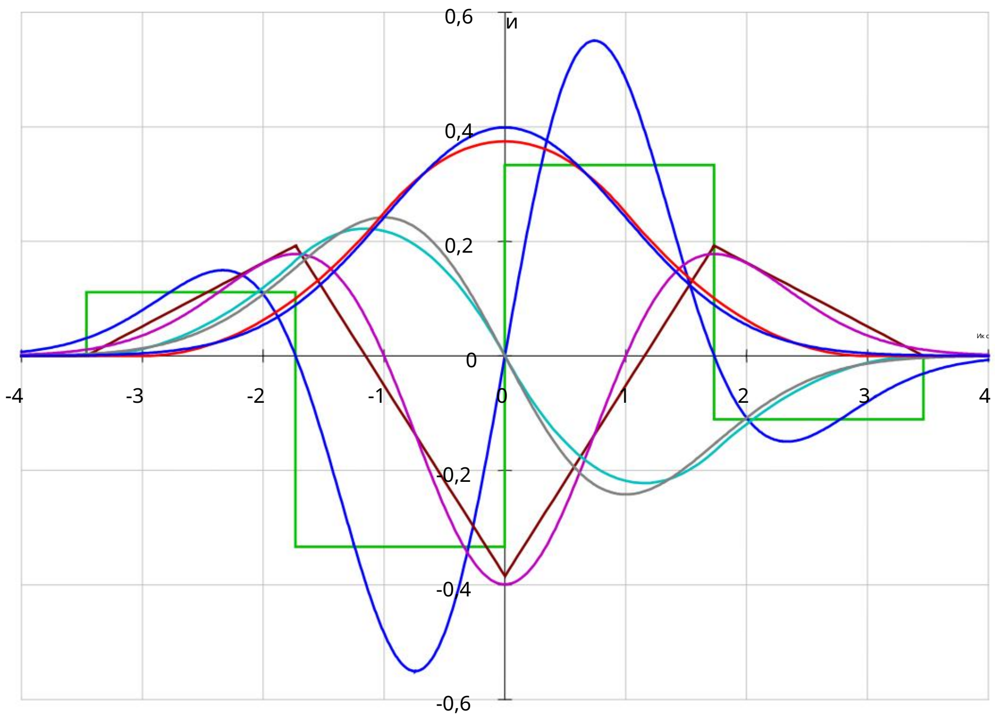


Рисунок 13: Функции аппроксимации четвертой степени

а. и "=" 1 / √2Пи Это − x²/2

б. y = b4(x)

в. Производная 1-й степени d. 1-я степень ок .

Это. Производная 2-й степени

ф. 2-я степень ок .

г. Производная 3-й степени h. ступенчатая функция s4(x)

Константы получены из тех же условий, что и ранее:

$$б_4(x) = \int_{-2c}^{2c} б_4(x)\,dx = \frac{1}{\sqrt{2\pi}} \int e^{-\frac{x^2}{2}} \quad и \quad \pi^2 = \int x^2 g_4(x)\,dx = \frac{c^4}{9} = 1,$$

что составляет a = 1/9 и c = √3.

Ступенчатая функция 4-й степени приближения

$$s_4(x) = \frac{1}{9}\left[2\Theta(x+2\sqrt{3}) - 4\Theta(x+\sqrt{3}) + 6\Theta(x) - 4\Theta(x-\sqrt{3}) + \Theta(x-2\sqrt{3})\right] \quad и$$

$$б(x) = \frac{1}{108}\left[\left|x+2\sqrt{3}\right|^3 - 4\left|x+\sqrt{3}\right|^3 + 6|x|^3 - 4\left|x-\sqrt{3}\right|^3 + \left|x-2\sqrt{3}\right|^3\right].$$

Ошибка $e_4 = \sqrt{\int (s_4(x) - б(x))^2\,dx} = 0{,}01812064909$, двумерная ошибка равна 0,01405961129.

Возможно дальнейшее увеличение степени аппроксимации. Процедура всегда одинакова, и должна быть возможность определить подход для любой степени.

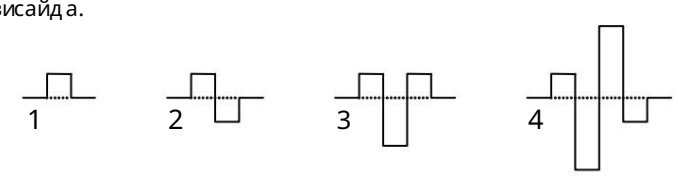
1.8. Произвольная степень приближения

Предыдущий расчет помогает определить произвольную градусную функцию. Чтобы аппроксимировать форму колокола Гаусса в n-й степени функция имеет n шагов одинаковой ширины. Каждая ступень высокая, но с чередующимся знаком $n-1$ а ширина равна 1/c. Таким образом, шаги для n = 1 — это {+1}, для n = 2 — это {+1, –1}, для n = 3 — это {+1, —2, +1}, для n = 4 — это {+1 , –3, +3, –1} и так далее. Формула такой биномиальной ступенчатой функции:

$$s_n(x) = \sum_{k=0}^{n-1} (-1)^k \binom{n-1}{k} \Theta\left(x + c\left(\frac{n}{2} - k\right)\right) = a \sum_{k=0}^{n-1} (-1)^k \binom{n-1}{k} \Theta\left(x + c\left(\frac{n}{2} - k\right)\right) \qquad (9)$$

где Θ обозначает ступенчатую функцию Хевисайда.

Функции sn(x) выглядят так:



Когда sn(x) интегрируется n–1 раз, это дает n-ю степень функции аппроксимации Гаусса bn(x).

Константы a и c должны быть определены так, чтобы площади аппроксимируемой функции колокола и функции Гаусса были одинаковыми.

Это условия: $б_n(x) = 2\int_0^{n/2} s_n(x)\,dx = \int \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,dx = 1$.

m-й первообразный sn(x) равен (m 0): $b_{m,k}(x) = \frac{a}{2^m c^m} \sum_{k=0}^{n} (-1)^k \binom{n}{k} \frac{(cx + n/2 - k)^{m+1}}{|cx + n/2 - k|}$.

Константа интегрирования всегда равна нулю, чтобы получить аппроксимацию функции Гаусса для $|x| > \frac{n}{2c}$ $б_{approx}(x) = 0$.

Для первого условия значений функций a и c $|x| > \frac{n}{2c}$ равны нулю, поэтому интеграл равен

$$2\int_0^{n/2} б_{approx,1}(x)\,dx = 2б_n \frac{n}{2c} = \frac{a}{c^n} = 1, \quad \text{потому что} \quad \frac{1}{a} = c^n = \left|\sum_{k=0}^{n} (-1)^k \binom{n}{k} \left(\frac{n}{2} - k\right)^n\right| = n!$$

Последнее крайнее правое равенство описано в [2], глава 1.2.6, стр. 64 Уравнение.

$$н! \sum_{к=0}^{н} к! \binom{0}{знать} = \sum_{к=0}^{н} к! \sum_{я=0}^{н} \binom{я}{к} \binom{н}{я} (-1)^{в} = \sum_{я=0}^{н} (-1)^{в} \binom{н}{я} \sum_{к=0}^{я} к! \binom{я}{к} = \sum_{я=0}^{н} (-1)^{в} \binom{н}{я} я^{н} = \sum_{я=0}^{н} (-1)^{в} \binom{н}{я} я^{н} \quad (34)$$ (10)

с помощью $П_с(\text{так как},) = \sum_{я} \binom{И+н}{К} \binom{}{я} (-1)^{в} = \binom{с}{знать}$. В этом случае s = 0 P0(n,k) равен единице, если n = k

и ноль в противном случае. Два крайних правых равенства уравнения (10) верны только потому, что при этом исключаются все члены, кроме члена n = k.

Ps(n,k) доказывается индукцией[1] : при n = 0 член Ps(0,k) равен $\sum_{я} \binom{И+0}{К} (-1)^{я} = \binom{с}{К}$ и если Ps(n,k) истинно,

$П_с(\ ,\ ) + 1$, поскольку $\sum_{я} \binom{И+}{К} \binom{н+1}{я} (-1)^{н+1+я} = \sum_{я} \binom{И+}{К} \binom{н}{я} (-1)^{н+1+я}$

$$= \sum_{я} \binom{И+}{К} \binom{н}{я-1} (-1)^{в(1)} + \sum_{я} \binom{И+}{К} \binom{н}{я} (-1)^{в} = \binom{с+1}{знать} - \binom{с}{знать} = \binom{с}{знать-1}.$$

Аппроксимация теперь приняла вид: b $$б_{м,н}(Икс) = \frac{с^{нм}}{2м!} \sum_{я=0}^{н} (-1)^{я} \binom{н}{я} \frac{(сх + н + / 2я)^{м+1}}{|сх + н + / 2я|}.$$

bm,n(x) необходимо проинтегрировать m = n – 1 раз, чтобы получить: b $$б_н(Икс) = \frac{с}{2(н-1)!} \sum_{я=0}^{н} (-1)^{я} \binom{н}{я} \frac{(это + 2 сх)^{н}}{|сх это + 2|}.$$

Для второго условия снова стандартное отклонение аппроксимации должно быть равно единице. Это можно решить с помощью интеграции по частям:

$$П^2 = 2 \int_0^{Nс/2} х^2 б(х) г = 2 \frac{н}{2с} б_{пп,} \frac{н}{2с} - 4 \frac{н}{2с} б_{нн+1,} \frac{н}{2с} + 4б_{нн+2,} \frac{н}{2с} = 2 \frac{н}{2с}^2 - \frac{1}{2} \frac{2н}{с} \frac{н}{4с} + 4 \frac{пп(3\ 1)}{48с^2} = 1$$

с $\sum_{я=0}^{н} (-1)^{я} \binom{Н}{я} ()$ является $^{1н+} = \sum_{я=0}^{н} (-1)^{в} \binom{Н}{я} я^{н+1} = \frac{нн(+1)!}{2}$ и $\sum_{я=0}^{н} (-1)^{я} \binom{Н}{я} (В)^{н+2} = \frac{(3\ 1)(+2)!\ ннн}{24}$.

Следовательно, константа с равна $$с = \sqrt{12}.$$ (11)

Аппроксимация n-й степени функции Гаусса на рисунке 14 для n > 0 составляет:

$$б_н(Икс) = \frac{\sqrt{3н}}{12(н-1)!} \sum_{я=0}^{н} (-1)^{я} \binom{Н}{я} \frac{(хн\sqrt{12/2н}\ я)^{н}}{|хн\sqrt{12/2н}\ я|}$$ (12)

В главе 1.10 к тому же уравнению применяется другой подход.

Ступенчатая функция для аппроксимации n-й степени составляет:

$$с_н(Икс) = \sqrt{\frac{н}{12}} \sum_{я=0}^{н} ()^{я} \binom{Н}{я} \text{ч е}\left(Икс\sqrt{\frac{пп}{12}} я + \frac{}{2}\right)$$ (13)

[1] Это упражнение 19 главы 1.2.6 в [2].

Уравнения (12) и (13) для n 4 подтверждаются расчетами предыдущих глав. Функции равнынулюдля значений вне |x| > 3в = 3п. На рисунке 14 показаныграфики уравнения (12) для разных степеней аппроксимации в сравнении с функцией Гаусса.

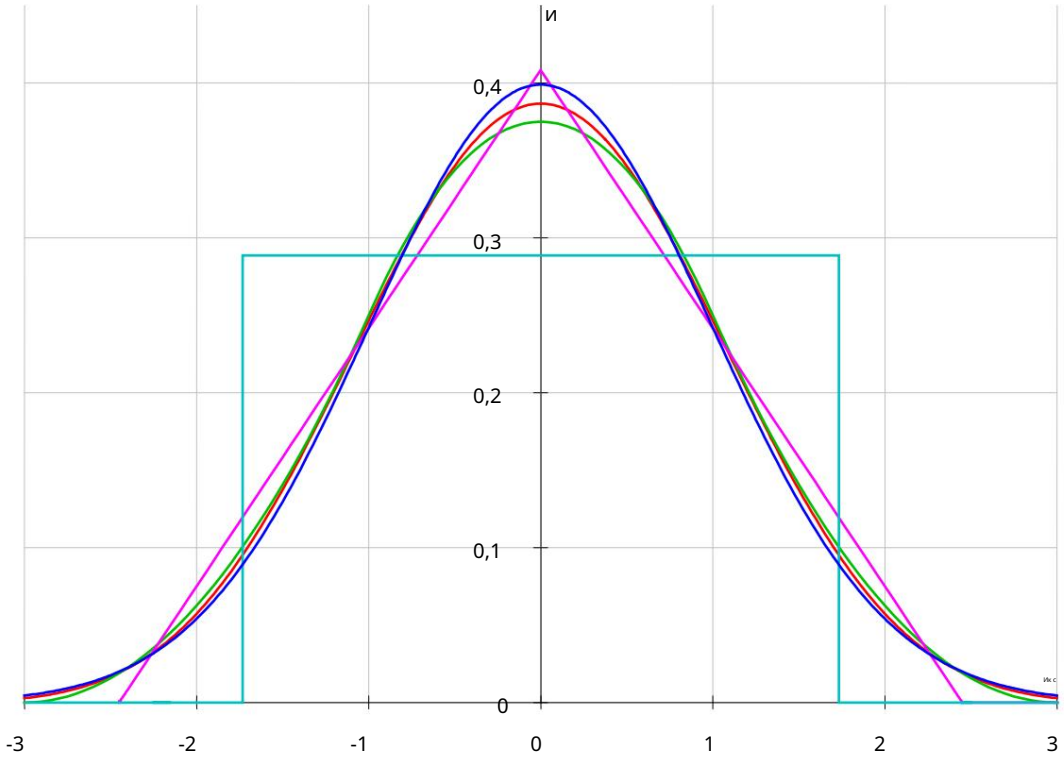


Рисунок 14:
аппроксимация
n-й степени
а. Размытие по
Гауссу б. 1-я степень ,
размытие коробки
в. 2-я степень ,
линейное размытие
д. 3-я степень ,
квадратное
размытие е. 5-я
степень , четверное размытие.

1.9. Точность

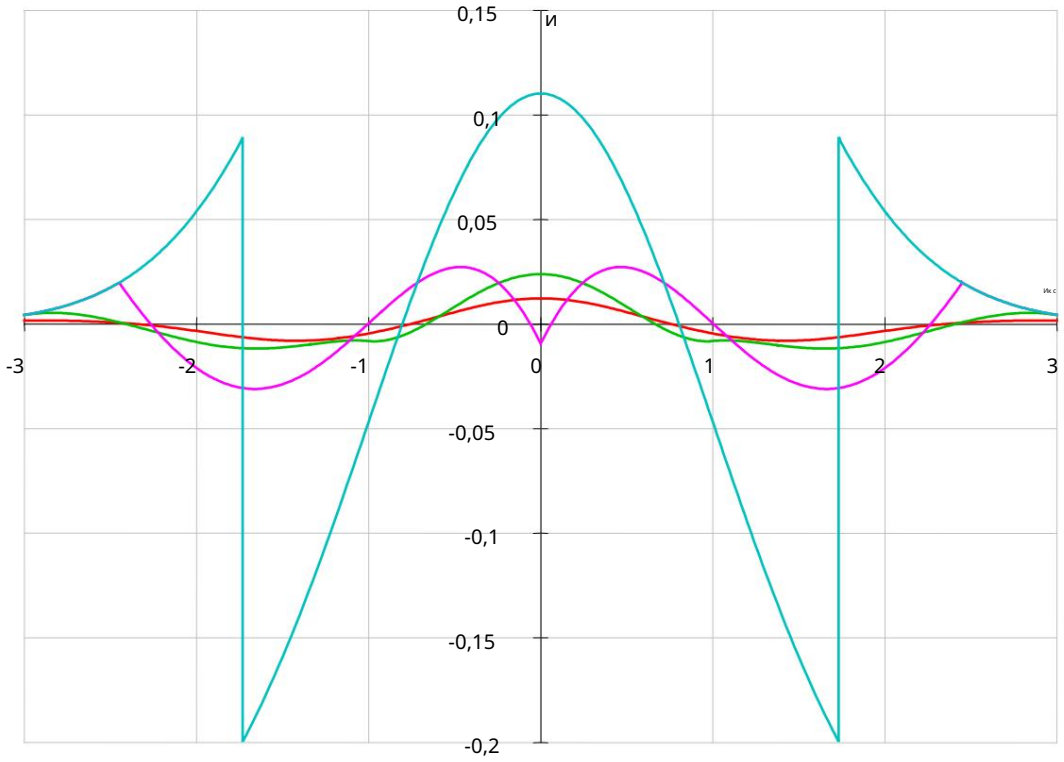


Рисунок 15:
аппроксимация
ошибка
а. 1-я степень ,
размытие коробки
б. 2-я степень ,
треугольник с. 3-я
степень ,
квадратное
размытие d. 5-я степень , четверо

Точность одномерной аппроксимации степени n составляет

Machine Translated by Google

Это "=" $\sqrt{\int (г( ) - б( ) x)^2 дx}$ "=" $\sqrt{2\int_0^{\sqrt{3н}} \left(\frac{1}{\sqrt{2Пи}} Это^{-\frac{икс^2}{2}} - б( )x\right)^2 x + \frac{1 \text{ ярд } 3н}{2\sqrt{Пи}}}$ что касается: ru $\frac{1}{15н8}$.

Ошибка аппроксимации всей функции не имеет такого большого значения. Ошибка в частотном спектре следующей главы более важна и имеет более низкую экспоненциальную зависимость от n.

Двумерная ошибка

Это "=" $\sqrt{\iint (g( )g( ) - b( )b( ))^2 xyxy}$ "=" $\sqrt{4\int_0^{\sqrt{3н}}\int_0^{\sqrt{3}} \left(\frac{1}{2Пи} Это^{-\frac{xy^{2}+{2}}{2}} - ( )н д x сик си\right)^2 + \frac{1 \text{ эрф } 3 б( )б н^2}{4Пи}}$,

где erf(x) обозначает функцию ошибок: erf( ) икс "=" $\frac{2}{\sqrt{Пи}}\int_0^{икс} е^{-т^2} д т$.

Выбранное приближение не является идеальным, поскольку оно медленно сходится к функции Гаусса, существуют лучшие решения. Но у этого есть преимущество простоты вычислений для дискретных данных.

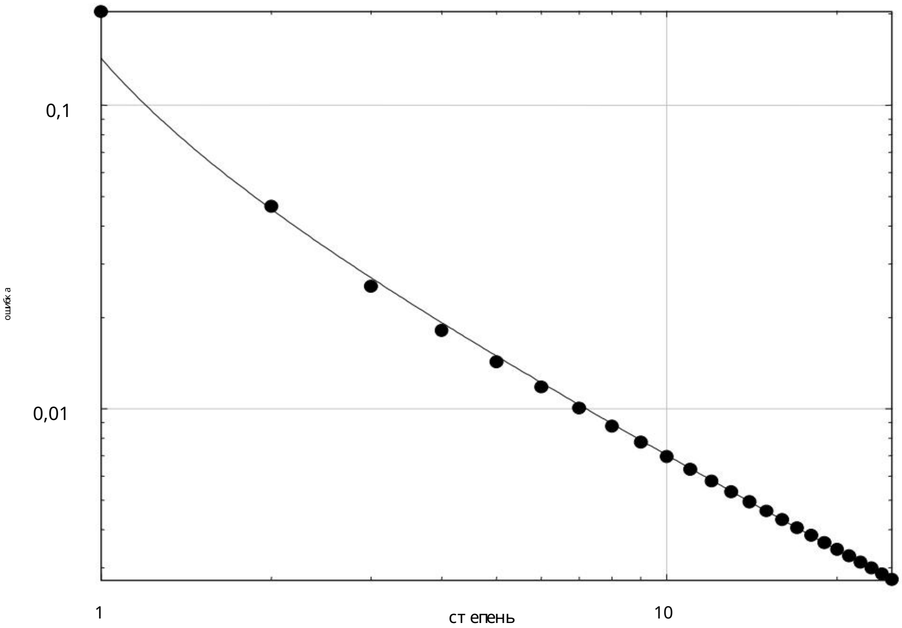


Рисунок 16:
Точность одномерной аппроксимации

1.10. Частотный спектр

Пространственная область функции размытия может быть преобразована в частотную область с помощью преобразования Фурье. Это преобразование позволяет сравнить размытие по Гауссу с приближениями в отношении затронутого частотного спектра.

Интегральная теорема Фурье для функции f(x) определяется как:

$$е( ) "=" \frac{1}{Пи}\int_0^{\infty} df( ) \text{ потому что } ой( ) x a( ) \text{ потому что } ой "=" \int_0^{\infty} ой c + c( ) \sin d ой c) ой \quad (14)$$

со спектральными функциями: $a( ) ой "=" \frac{1}{Пи}\int f(x) \cos x ой d x$ и $c( ) ой "=" \frac{1}{Пи}\int f(x) \sin x ой d x$. (15)

Комплексная часть c(ω) спектра равна нулю для четных функций от f(x), которые здесь используются, а действительная часть a(ω) равна нулю для нечетных.

Частотный спектр размытия по Гауссу уравнения (2):

$$Пи a_г( ) ой "=" \frac{1}{\sqrt{2Пи}}\int Это^{-\frac{x^2}{2}} \text{ тело } ой д x x е "=" ^{-\frac{ой^2}{2}} . \quad (16)$$

Преобразование Фурье bn(x), кажется, вычислить немного сложнее. Но согласно теореме о свертке преобразование Фурье в пространственной области является точечным произведением в частотной области.

Преобразование Фурье прямоугольной (коробчатой) функции представляет собой кардинальный синус:

rect( )Икс cos d ойxx "=" $\frac{(1)}{2}$ 2 1 2 Икс + я / 2 1 2 Икс + я тело d ойxx "=" грехой 2 / ой 2 .

Свертка двух одинаковых коробчатых функций представляет собой треугольную (линейную)

три( )Икс "=" прямоугольник ( )Икс прямоугольник( )Икс "=" прямой( ) прямой(sxs) д "=" $\frac{(функция 2)}{2}$ (+ 1) х я / Икс + 1 я .

Таким образом, преобразование Фурье согласно теореме представляет собой квадрат кардинального синуса:

tri( )Икс тело d ойxx "=" $\frac{((1)}{2}$ 2 х + 1) я / Икс + 1 я телойдxx "=" $\left(\frac{грехой 2}{ой 2}\right)^2$.

И еще одна свертка с помощью прямоугольной функции дает:

(tri( )Икс rect( )) cos Икс d ойxx "=" $\frac{(1)}{16}$ 3 (2 3 2) + х я / 2 3 2 Икс я телойдxx "=" $\left(\frac{грехой 2}{ой 2}\right)^3$.

При дальнейшем умножении обеих областей уравнение пространственной области становится хорошо известной закономерностью

Когда операция x*n определяет n-кратную итерацию свертки с самой собой (степень свертки), тогда n-я степень кардинального синуса является преобразованием Фурье n-й степени свертки)

прямоугольная функция: прямоугольник * ( )Икс cos d ойxx "=" $\frac{(1)}{(н 1)!}$ н (+ х / 2 н я / хн + / 2 я телойдxx "=" $\left(\frac{грехой 2}{ой 2}\right)^н$. (17)

Уравнение (17) справедливо даже для n = 0, поскольку прямоугольник * ( )Икс cos d ойxx "=" δ( )Икс cos d ойxx "=" 1 .

Обратным преобразованием Фурье получаем: rect*n (x) "=" $\frac{1}{Пи}$ $\int_0$ $\left(\frac{грехой 2}{ой 2}\right)^н$ телойд Икс ой (18)

а для n = 0 другое определение дельта-функции Дирака как: δ(x) , "=" $\frac{1}{Пи}$ $\int_0$ телойд Икс ой "=" $\int_0$ телоПдх с .

Аналогичное соотношение существует для ступенчатой функции из-за сложной части преобразования Фурье:

Θ ( )Икс sin d ойxx "=" $\int_0$ грехойдxx "=" $\frac{1}{ой}$, и обратного преобразования: ( ) Θ = + Икс $\frac{1}{2}$ $\frac{1}{Пи}$ $\int_0$ $\frac{грех ойИкс}{ой}$ дой. (19)

Каким-то образом на предыдущих страницах была получена приведенная в действие свертка прямоугольной функции: bn(x) = √n/12 rect*n (x √n/12).

Таким образом, частотный спектр π an(ω) аппроксимации bn(x) уравнения (12) представляет собой просто степень кардинального синуса:

Пи а н ( )ой "=" $\int_{-\sqrt{3н}}^{\sqrt{3н}}$ б( )н потому что d ойxx "=" $\left(\frac{грехой\sqrt{3/н}}{ой\sqrt{3/н}}\right)^н$ (20)

Уравнения (16) и (20) также предполагают замечательный предел: прик лечь его п $\left(\frac{н}{Икс}\right)^{н^2}$ Икс н = Это $\frac{2х}{6}$, (21)

с тех пор, как я решил п $\frac{Икс}{н}$ Икс н = Лим м 0 $\frac{1}{м^2}$ $\frac{грехмх}{мх}$ "=" Лим м 0 $\frac{потому что МХ МХ мгрех}{2 мгрех мх^2 а}$ "=" Лим м 0 $\frac{х 2мх соз}{2 мгрех а МХ 6cos мх}$ = $\frac{Икс^2}{6}$.

В следующей таблице приведен частотный спектр степеней первого приближения:

Спектр степени n 0-й	Пространственная область bn(x)	Частотная область πan(ω)
степени (тождественность):	д(х)	1
1-я степень (размытие рамки):	$\frac{\Theta x\left(+3-\sqrt{\theta x}\right)-3\quad\left(\quad\sqrt{}\right)}{23\sqrt{}}$	$\frac{\text{грех ой}\sqrt{3}}{\text{ой}\sqrt{3}}$
2-я степень (линейная):	$\frac{\left\lvert x+6\sqrt{}-2x+x-6\quad\sqrt{}\right\rvert}{12}$	$\frac{1\text{ потому что ой}\sqrt{}\text{ро }6}{3\text{ой}^2}$
3-я степень (квадрат):	$\frac{\left(x+3x\right)\lvert+3-x\rvert-3x\left(-3-3x\right)\rvert+1x+\lvert1+3\left(x-1x-\right)\rvert\quad\left(\quad\right)\quad\rvert}{32}$	$\frac{3\text{ греха ой}\quad\text{грех 3ой}}{4\text{ой}^3}$
4-я степень (кубическая):	$\frac{\left\lvert+23\sqrt{x}\right\rvert^3\quad\left\lvert32\sqrt{}34\rvert^3343\,6+x--x\sqrt{}+-x-+x\quad\sqrt{}\right\rvert^3\quad\rvert\rvert}{108}$	$\frac{6\,2\text{ потому что ой}\sqrt{}2\,3\,8\text{ потому что ой}\sqrt{}3}{9\text{ой}^4}$

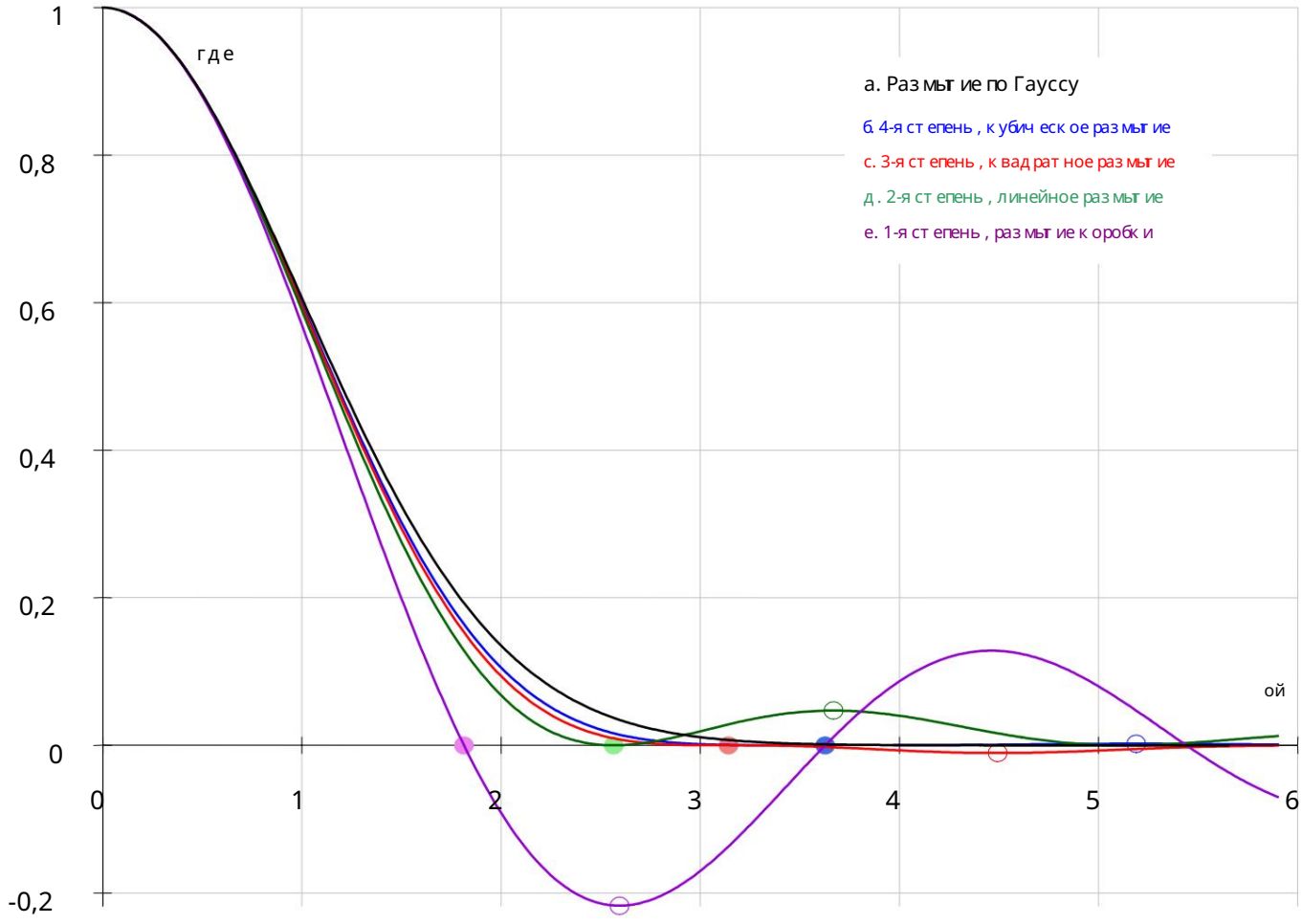


Рисунок 17: Частотный спектр аналогового приближения

Более высокие частоты в спектре аппроксимаций более низкой степени подавляются не так сильно, как гауссова свертка.

Для любой степени аппроксимации n существует минимальная частота ω0, которая полностью подавляется: an(ω0) = 0.
Эта частота рассчитывается по формуле $\omega_0 = \pi\sqrt{n/3}$ и обозначается точкой на рисунке 17.

Первые максимумы частотных спектрах аппроксимаций представляют собой наибольшую ошибку этой степени при ω > ω0 и обозначены кружком. Частоту ωp этого пика можно было найти, положив дифференцирование функции равным нулю: a'n(ωp) = 0.

Уравнение для решения этого вывода будет иметь вид $\sqrt{\tan}\ \omega p\ 3/n\sqrt{=}\ \omega p\ 3/n$ для диапазона $\pi < \omega\sqrt{p\ 3/n} < 2\pi$ и не имеет символического решения. Численный расчет степени n составляет: ωp = 2,594271160 n. $\sqrt{}$

Величина пиковой ошибки составляет: π an(ωp) = (–4,603338849)–n . (22)

2. Алгоритм.

Дискретное изображение исходит из аналогового мира. Но для обработки изображения компьютерами необходимо его оцифровать. Это получается из аналогового изображения путем выборки и квантования. На аналоговое изображение накладывается регулярная сетка, состоящая из элементов изображения (пикселей), и каждому пикселю присваивается целое число, значение которого представляет яркость или яркость.

Такое изображение имеет две основные характеристики:

- Пространственная область S, представляющая обычно прямоугольную сетку пикселей, обычно в двух измерениях строки и столбцов. Возможные значения пространственной области определяют разрешение изображения. Пространство размером 2048x1536 пикселей представляет собой, например, изображение размером 3 мегапикселя.
- Область диапазона R, представляющая значение пикселя p. Область диапазона может быть выражена с помощью одного значения для серых изображений или тройка чисел для цветных изображений. Обычные диапазоны—3x8 или 3x16 бит.

Биномы

Центральная предельная теорема утверждает, что биномиальное распределение может быть аппроксимировано распределением Гаусса, если n достаточно велико (теорема Муавра – Лапласа). Обратив этот принцип, можно получить разумную аппроксимацию гауссовой функции с помощью биномиальной функции распределения.

Аппроксимация биномиального распределения центральной предельной теоремой для больших n :

$$\text{н}_{\text{я}}\ \text{пп}^{\text{п}}(1\quad)^{\text{в}}\quad\frac{1}{\sqrt{2\text{Пинапример п}}})\ \text{и п}^{\frac{(\text{я нп}^{2}}{2\,1\,(}}\ .\qquad(23)$$

Гауссово приближение вероятности p = 1/2 для больших n : $\text{н}_{\text{я}}\ \frac{1}{2^{\text{н}}}\ \sqrt{\frac{2}{\text{Пинет}}}^{\text{и}}\ ^{\frac{2}{\text{н}}}{}_{\text{я}}\ \frac{\text{н}}{2}^{2}$.

Таким образом, простые биномиальные целые числа можно использовать для быстрого вычисления функции Гаусса, которое при прямом вычислении будет намного медленнее.

2.1. Степень приближения

Полиномиальные функции выводятся n раз, пока не останутся только константы Эти константы вместе с координатами x частей полинома определяют функцию фильтра. Производные необходимо снова интегрировать, чтобы получить функцию фильтра. Численное интегрирование константы очень просто. Здесь пригодится то, что вычисляются последовательные пикели. Таким образом, интеграция предыдущего пикселя уже рассчитана. Остаётся только добавить разницу. В этом случае алгоритм очень прост.

Размытие по Гауссу двумерного изображения применяется как два независимых одномерных расчета.

Степень приближения к форме гауссова колокола определяет точность. Когда степень равна единице, аппроксимация представляет собой размытие прямоугольника. Степень n = 2 представляет собой линейное (треугольное) приближение. Степень n = 3 —это квадратичная аппроксимация и так далее.

В то же время степень определяет количество пикселей, суммируемых за итерацию.

Хотя алгоритм является лишь приближением, результат достаточно точен для большинства приложений. По максимально возможной степени аппроксимации расширенный биномиальный фильтр равен обычному биномиальному фильтру (конечно, имеющему то же время работы).

2.2. Расчет

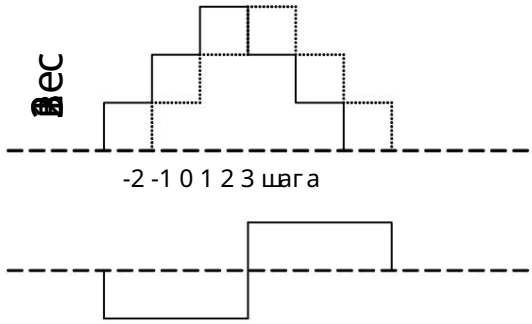


Рисунок 18: раздвижное окно

Размытие упрощено до формы треугольника. Чем выше значения, тем больше вес пикселя.

Размытие теперь выполняется с помощью алгоритма скользящего окна. На рисунке 18 приблизительное треугольное размытие фактического пикселя показано сплошной линией, а желаемое размытие следующего пикселя — пунктирной линией. Разница между этими двумя функциями показана ниже. Оно меняется только перед размытием (шаг 3) путем добавления значения пикселя, на пике (шаг 1) путем вычитания двойного значения и в конце (шаг -2) путем повторного добавления значения пикселя.

В большинстве случаев это приближение будет достаточно точным, но используемый алгоритм можно доработать для большей точности. Вначале мы всегда начинаем с шага функции. Уровни представляют собой числа треугольника Паскаля с чередующимся знаком. Разница двух функций, один сдвиг на шаг, составляет предыдущую функцию

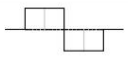
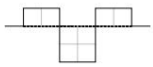
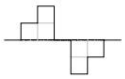
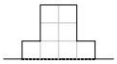
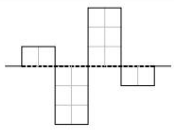
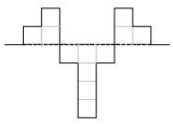
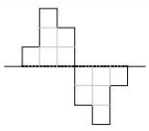
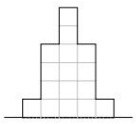
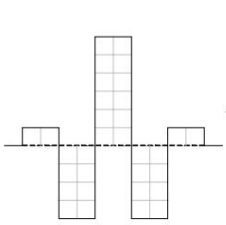
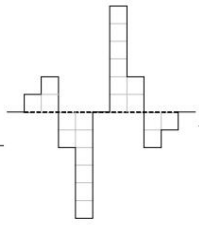
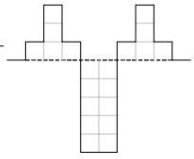
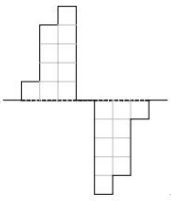
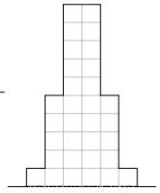
n Шаг и аппроксимации для радиуса размытия 2	Веса wn,2(k)
1 а: 	а: 1 1
2 а:  б: 	а: 1 1 -1 -1 б: 1 2 1
3 а:  б:  с: 	а: 1 1 -2 -2 1 1 б: 1 2 0 -2 -1 в: 1 3 3 1
4 а:  б:  с:  д: 	а: 1 1 -3 -3 3 3 -1 -1 б: 1 2 -1 -4 -1 2 1 в: 1 3 2 -2 -3 -1 д: 1 4 6 4 1
5 а:  б:  с:  д:  Это: 	а: 1 1 -4 -4 6 6 -4 -4 1 1 б: 1 2 -2 -6 0 6 2 -2 -1 в: 1 3 1 -5 -5 1 3 1 д: 1 4 5 0 -5 -4 -1 е: 1 5 10 10 5 1

Рисунок 19: расчетные веса

На рисунке 19 показаны расчетные веса разных степеней n. Весовые числа правого столбца представляют собой простое сложение чисел сверху и слева, как треугольник Паскаля.

Если степень аппроксимации является радиус размытия, алгоритм вычисляет точно целочисленное представление размытия по Гауссу, числа треугольника Паскаля. Конечно, усилия по его приобретению аналогичны прямому расчету. Преимущество состоит в том, что степень не обязательно должна быть такой же большой, как радиус размытия, что особенно полезно для больших радиусов размытия.

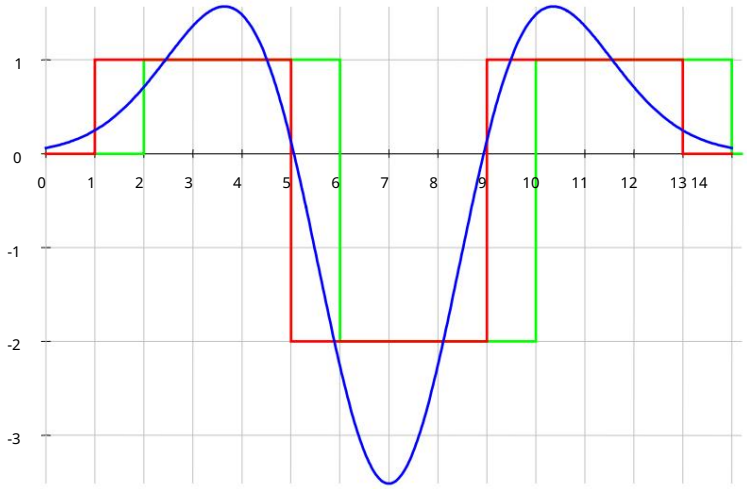


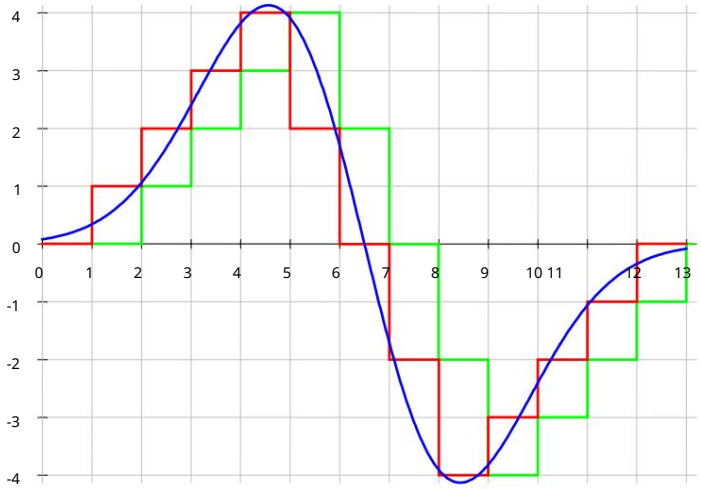
Рисунок 20: Расчетная сетка, степень 3, радиус 4

а) 2-я производная

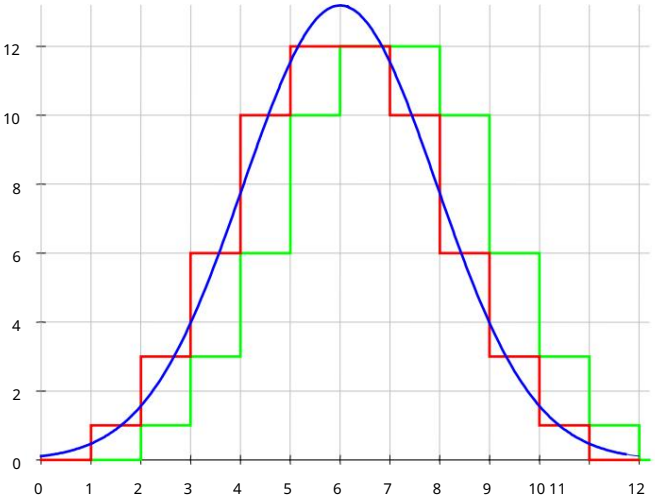
красный: текущий

пиксель зеленый:

следующий пиксель синий: производная Гаусса



б) 1-я производная



в) дискретное приближение

Предположим, что красная функция на рисунке 20c, которая очень похожа на форму колокола Гаусса, уже рассчитана. Теперь следует вычислить зеленую функцию следующего пикселя справа. Разница между красной и зеленой функцией на рисунке 20c изображена как красная функция на рисунке 20b. Далее предположим, что эта красная функция уже рассчитана, и необходимо вычислить следующую разность, которая изображена как зеленая функция на рисунке 20b. Теперь разница между красной и зеленой функцией отображается красным цветом.

функция на рисунке 20a. Опять же, эта функция уже рассчитана, и для разницы с зеленой функцией следующего пикселя нужно добавить только четыре значения.

Как работает алгоритм? Предположим, что три красные функции на рисунках 20a, b и c уже настроены при инициализации. Необходимо рассчитать размытие следующего пикселя на рисунке 20c (зеленом). Красная функция изображения 20b добавляется к красной функции изображения 20c, чтобы получить зеленую функцию изображения 20c. Теперь красная функция изображения 20a добавляется к красной функции изображения 20b, чтобы получить зеленую функцию изображения 20b для следующего пикселя. Чтобы получить зеленую функцию изображения 20a, значение пикселя 1 добавляется, значение пикселя 5 вычитается два раза, значение пикселя 9 добавляется два раза и значение пикселя 13 вычитается. Вот и все! При 6 сложениях (плюс одно умножение) и доступе к 4 пикселям размытие следующего пикселя рассчитывается для общей ширины размытия, равной 10.

Если 3-я степень приближения все еще недостаточно точна, можно применить более высокие степени. Они работают так же, как объяснялось ранее. Таким образом можно получить любую необходимую точность. Конечно, более высокая точность более сложна, но время выполнения все равно не зависит от радиуса размытия.

2.3. Пример расширенной биномиальной последовательности:

Определения:

p дискретный радиус размытия (ширина шага)

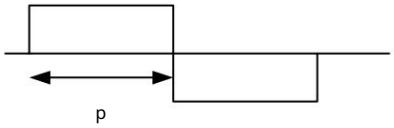
n степень аппроксимации

m m-е накопление ступенчатой функции

k шагов по горизонтали

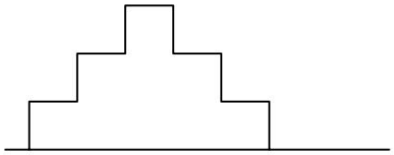
n = 2, r = 3

m = 0



k: 0 1 2 3 4 5

m = 1



Для m > 0 вес представляет собой сумму чисел непосредственно сверху и слева, как показано в правом столбце на рисунке 20. Эта таблица весовых функций параметров m и k:

	0	1	2	3	k
0	a0	a1	a2	a3	и
1	a0	a0+a1	a0+a1+a2	a0+a1+a2+a3	$\sum_{i=0}^{k} a_i$ есть
2	a0	2a0+a1	3a0+2a1+a2	4a0+3a1+2a2+a3	$\sum_{i=0}^{k} (k+1-i)\, a_i$ есть
3	a0	3a0+a1	6a0+3a1+a2	10a0+6a1+3a2+a2	$\sum_{i=0}^{k} \frac{(k+2)(k+1)}{2} a_i$
м	a0	ma0+a1	$\frac{m+1}{2}$ a0+a1+a2 и в	$\frac{m+2}{3}$ a0+$\frac{m+1}{2}$ a1+a2+a3 и в	$\sum_{i=0}^{k} \frac{(k+m-i)(k+m-1-i)}{2}$ a_i

В этой таблице представлен расчет размытия пикселя k в горизонтальном направлении и шаг накопления m. вертикально. Каждая запись представляет собой сумму элемента выше плюс слева.

Дискретный вес шага k для первой строки m = 0 составляет $a_k = (-1)^{k} \binom{n}{k} \; \lfloor k/d \rfloor$ потому что $\lfloor k/d \rfloor = \lfloor k/d \rfloor$, где

обозначает функцию пола. Таким образом, веса wn,r(k) расширенной биномиальной последовательности шага k (для m = n − 1 накоплений) равны

$$w_{n,r}(k) = \sum_{i=0}^{k} (-1)^{i} \binom{n}{i} \binom{n-1+k-ir}{k} \;=\; \binom{n}{k}_{r-1} \qquad [p > 0] \qquad (24)$$

Коэффициенты последовательности wn,r(k) являются полиномиальными коэффициентами степенного одномерного многочлена $(1 + x + x^2 + \dots + x^{p})^{n} = \sum_{k=0} \binom{n}{k}_{p} x^{k}$, что также можно выразить как рекурсию $\binom{n}{k}_{p} = \sum_{i=0}^{p} \binom{n-1}{k-i}_{p}$, или

сумма полиномиальных коэффициентов:

$$\binom{n}{k}_{p} = \sum_{\substack{0 \le k_1, k_2, \dots, k_p \le n \\ k_1 + 2k_2 + \dots + p k_p = k}} \binom{n}{k_1, k_2, \dots, k_p}, \quad \text{например} \quad \binom{n}{k_1, k_2, \dots, k_p} = \frac{n!}{(n - k_1 - \dots - k_p)!\, k_1! \cdots k_p!}.$$

Параметр p n и r выбирают степень аппроксимации и дискретный радиус.

Для r = 1 или n = 0 вычисление размытия расширенного бинома сводится к тождественной функции $w_{n,1}(k) = w_{0,r}(k) = \binom{n,0}{k} = \binom{0,r}{k} = \binom{0}{k}$ вообще без размытия.

Для n = 1 веса составляют $w_{1,r}(k) = \lfloor k/r \rfloor - \binom{k}{0}$. При r = 2 функция представляет собой биномиальный коэффициент $w_{n,2}(k) = \binom{n}{k}$

а при r = 3 это своего рода трёхчленный коэффициент $w_{n,3}(k) = \binom{n}{k}_{2} = \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{n}{k-i} \binom{k-i}{i}$.

Размер последовательности (k-диапазона) равен s = n(r − 1). Вне этого диапазона 0 ≤ k ≤ s функция равна нулю.

Сумма по всем коэффициентам (общий вес):

$$W_{nr} = \sum_{k=0}^{s} w_{n,r}(k) = r^{n} \qquad (25)$$

Это значение также определяет необходимый диапазон типов данных, которые алгоритм должен обрабатывать в переменных для вычислений. В противном случае при расчете произойдет переполнение. Это значение быстро растет с увеличением степени аппроксимации.

Последовательность коэффициентов симметрична: wn,r(k) = wn,r(s – k).

Дисперсия для дискретных функций составляет:

$$\sigma^2 = \frac{1}{W_{n,r}} \sum_{k=0}^{s} \left(k - \bar{k}\right)^2 w_{n,r}(k) = \frac{1}{r^{n}} \sum_{k=0}^{s} k^2 \binom{n}{k}_{r-1} - \bar{k}^2 = \frac{n\left(r^2 - 1\right)}{12},$$

где $\bar{k}$ обозначает среднее значение, половину размера: $\bar{k} = \frac{1}{W_{n,r}} \sum_{k=0}^{s} k\, w_{n,r}(k) = \frac{s}{2} = \frac{n(r-1)}{2}$.

Таким образом, стандартное отклонение равно:

$$\sigma = \sqrt{\frac{n\left(r^2 - 1\right)}{12}} \qquad (26)$$

Это поправочный коэффициент между дискретным радиусом r и радиусом размытия. σ для аппроксимации n–1 степень н. Фактор r $\sqrt{2}$ аналогично значению квантования, поскольку при r = 1 размытие не применяется (тождество функция).

При r = 2 радиус σ равно стандартному отклонению биномов: $\sigma = \frac{\sqrt{n}}{2}$.

2.3.1. Расширенные биномиальные графики

На рисунке 21 показана дискретная расширенная биномиальная последовательность $w_{n,r}(k)$ (красный) для разных радиусов r и степеней аппроксимации n в сравнении с функцией Гаусса $g(x)$ (синий).

Обе оси функции Гаусса масштабируются до расширенного бинома, чтобы сделать графики сопоставимыми.

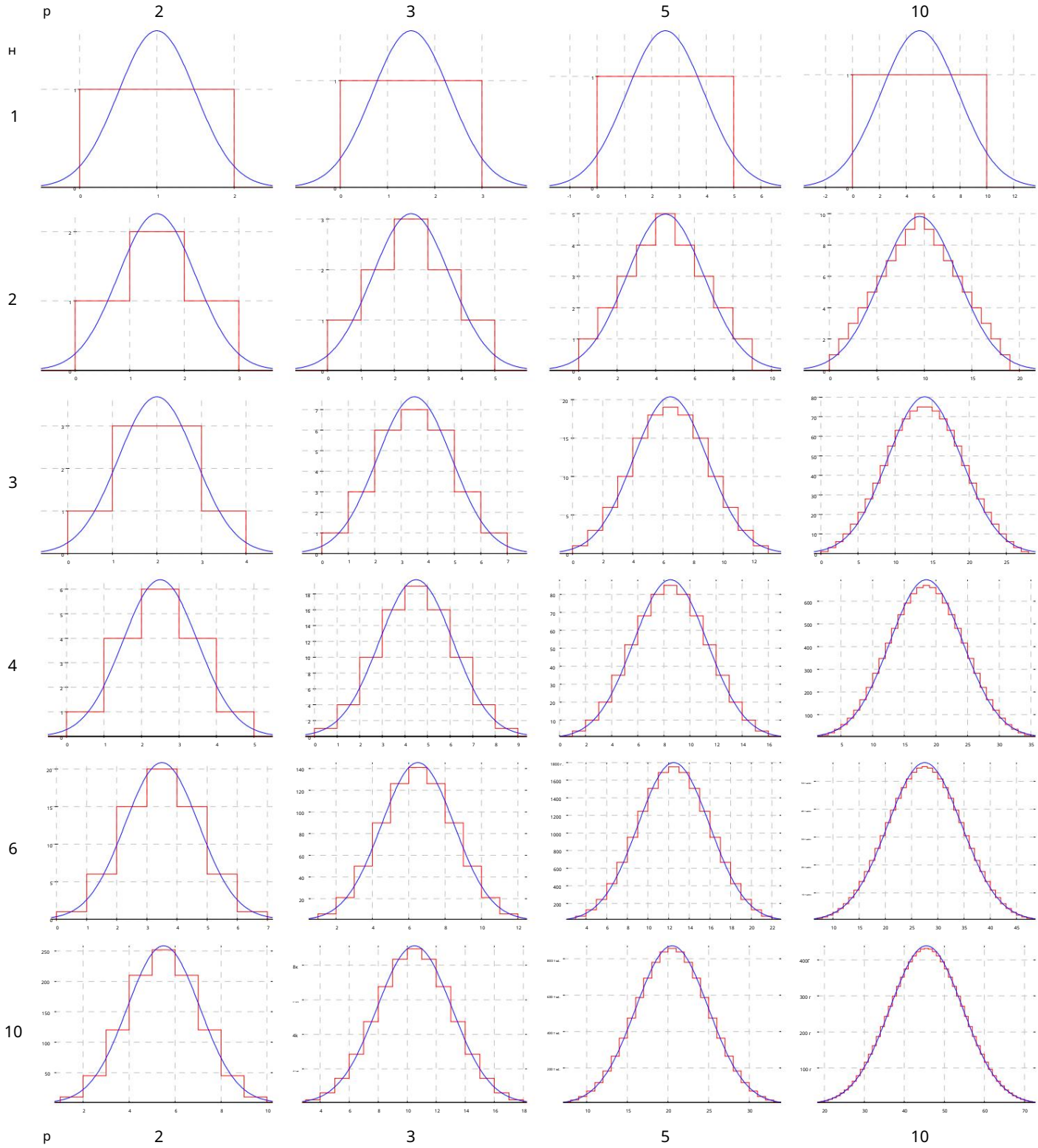


Рисунок 21: Графики расширенной биномиальной функции

Обратите внимание, что расширенные биномиальные коэффициенты представляют собой дискретную упорядоченную последовательность целых чисел, а не непрерывную функцию.

Строго говоря, сравнение дискретной и непрерывной функций не совсем корректно, поскольку аналоговая функция размытия радиуса r = 1 по-прежнему выполняет размытие аналогового изображения, тогда как дискретная функция размытия биномов дискретного радиуса r = 1 этого не делает.

2.4. Дискретный частотный спектр

Дискретные значения, такие как расширенная биномиальная функция, также могут быть преобразованы из пространственной области в частотную область с помощью дискретного преобразования Фурье (ДПФ).

Последовательность n комплексных чисел ak,... , an-1 преобразуется в последовательность n комплексных чисел âk,... , ân-1 методом ДПФ по формуле:

$$\hat{a}_k = \sum_{j=0}^{n-1} a_j \, e^{-2\pi i \frac{jk}{n}} \quad \text{для } [k = 0, \dots, n-1]. \tag{27}$$

Если последовательность является четной функцией действительных чисел, то от мнимой части преобразования можно избавиться. Эта симметрия может быть достигнута условием aj = an-j.

ДПФ для последовательности из n/2 действительных чисел aj = an-j дает

$$\hat{a}_k = \sum_{j=0}^{n-1} a_j \, e^{-2\pi i \frac{jk}{n}} = \sum_{j=0}^{n-1} a_j \cos\left(2\pi \frac{jk}{n}\right), \tag{28}$$

потому что

Преобразованная последовательность âk также становится четным рядом âk = ân-k из-за симметрии тригонометрической функции. То же самое относится и к обратному ДПФ.

Это уравнение ДПФ немного раздражает, поскольку оно не имеет никакого отношения к частоте спектра ω. Для лучшего понимания преобразование Фурье расширенной биномиальной последовательности разработано на основе уравнения (15). Чтобы сделать спектры сопоставимыми с предыдущими графиками, обе оси расширенной биномиальной функции размытия масштабируются до распределения Гаусса. Такая «нормализованная» дискретная функция размытия расширенного бинома последовательность рассчитывается по формуле:

$$e_{n,\sigma}(k) = \frac{c}{\sqrt{2\pi}\,\sigma} \sum_{x=0}^{n} w_{n,r}(x)\, s(x-k) \cdot \Delta x \tag{29}$$

Функция s(x) определяет функцию выборки пикселя из изображения (преобразование). На рисунке 22 показаны различные возможности построения аналогового изображения из трех дискретных значений пикселей 2, 4, 3 и наоборот.

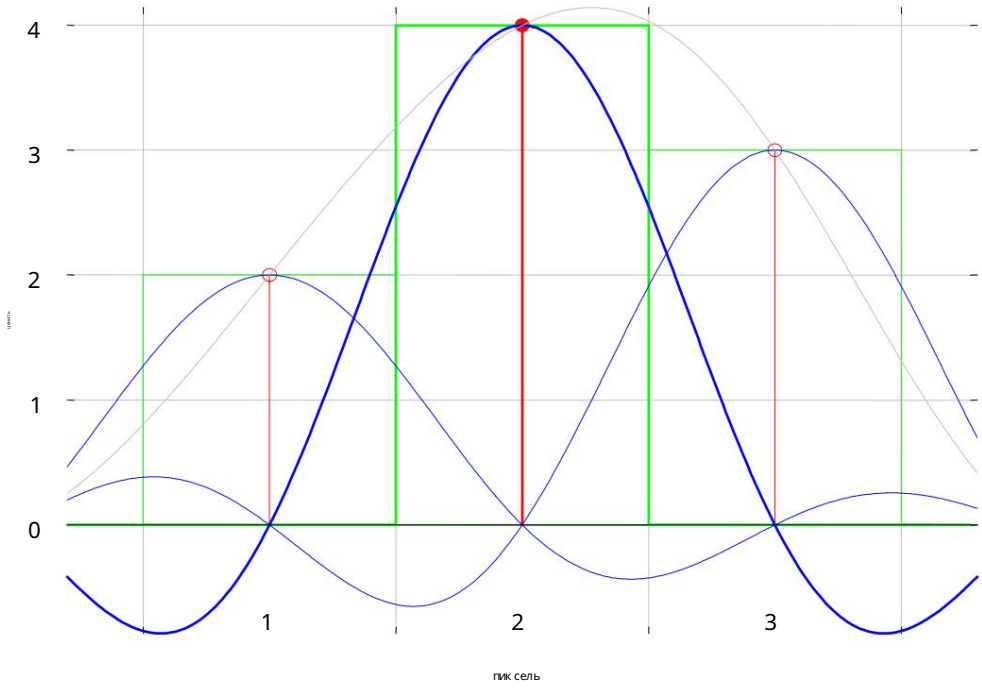


Рисунок 22:
Функции выборки пикселей s(x)

а) красный: дискретное

значение б) зеленый: среднее значение

в) синий: значение,
ограниченное

частотой (серый: сумма синего цвета)

Тогда функция частотного спектра составит:

$$\text{Спектр}(\omega) = \int f(x)\cos(\omega x)\,dx = \frac{c}{\sqrt{2\pi}\,\sigma} \sum_{k=0}^{n} w_{n,r}(k)\cos(\omega k - \pi) = \frac{1}{\sqrt{2\pi}\,\sigma} \cdot \frac{c}{\sqrt{2\pi}\,\sigma} \sum_{k=0}^{n} w_{n,r}(k)\cos(\omega k) \quad \text{потому что} \quad \omega = \frac{2\pi}{n}. \tag{30}$$

Поскольку функция fn,r(x) четная, комплексная часть равна нулю.

В следующей таблице представлены различные возможности функции выборки s(x) и ее преобразования Фурье ŝ(ω).

Пиксель выбран как	Функция s(x)	Преобразование Фурье ŝ(ω)	Частотный спектр Пиан, г(ω)
а) дискретное значение в центре пикселя	д(х)	1	1/р ∑ w (·) к=0 н с, потому что ок/так
среднее значение (целое) б) площади пикселя	прямоугольник (х)	грех ω/2 / ω/2	2П/р н грех ой/2П ой к=0 с w (·), потому что ок/так
в) значение, ограниченное частотой	грех Пикс / Пикс	ох / 2 Пи прямоугольный	1/р н прямой ой/2ПС к=0 с w (·), потому что ок/так

Возможны другие функции выборки.

Теперь можно было вычислить дискретный спектр последовательности пикселей.

Параметр круговой частоты ω станет расстоянием между дискретными пикселями p и его также необходимо масштабировать:

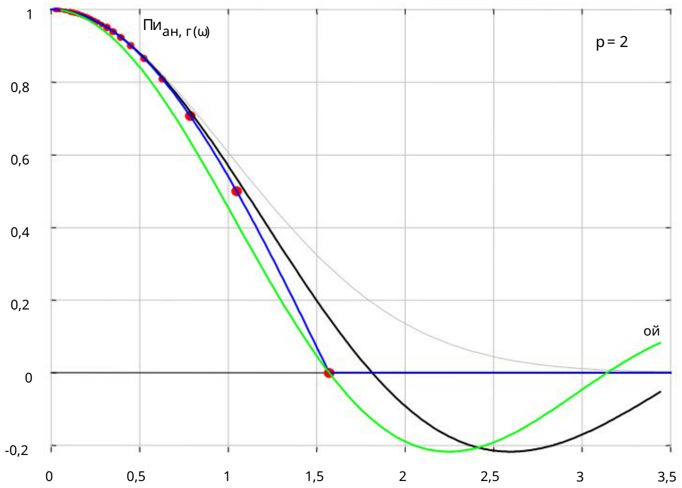
$$П "=" \frac{2ПС}{ой} "=" \frac{Пи}{ой}\sqrt{\frac{2 нет(1)}{3}} . \qquad (31)$$

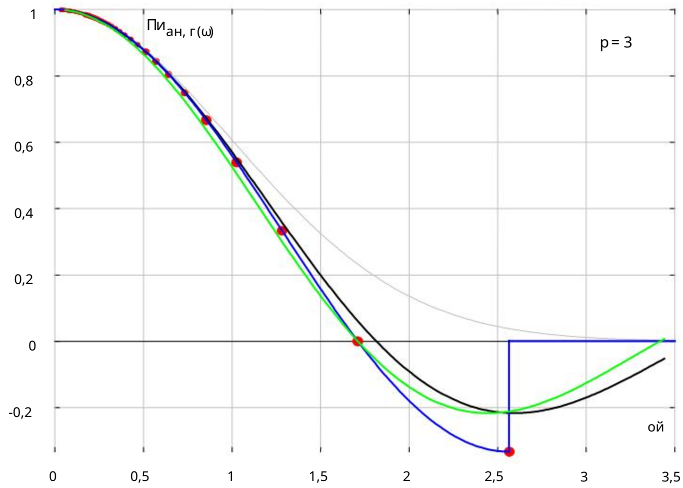
Поскольку изображение состоит из дискретных пикселей, спектр также содержит только дискретные значения расстояния пикселя (периода пространства) p. В случае выборочной функции а) частотный спектр станет ДПФ и теперь выглядит гораздо лучше.

больше похоже на уравнение (28):

$$Пиа_{нет} \frac{2ПС}{пиар} = \frac{1}{н} \sum_{к=0}^{с} w(\cdot) \cos 2 \frac{Пи ок}{П} . \qquad [целое число p, p > 0] (32)$$

Оставшееся отклонение связано с необходимостью разного масштабирования оси.





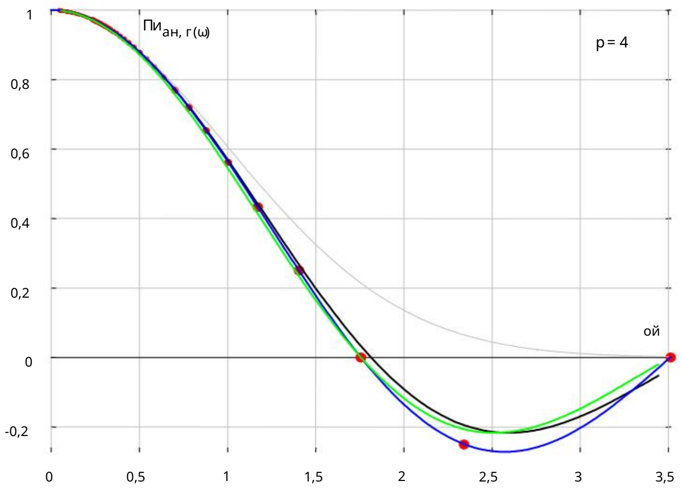


Рисунок 23: дискретные спектры 1-й степени (размытие прямоугольника)

для функций выборки а) красный:

дискретное значение пикселя b)

зеленый: среднее значение пикселя

с) синий: значение, ограниченное

частотой черный: спектр аналоговой аппроксимации

a1(ω) серый: гауссов спектр ag(ω)

На графиках рисунка 23 показан частотный спектр первой степени приближения для разных радиусов r.
Точки на графиках находятся на дискретных значениях возможных дискретных периодов времени. Дискретные точки начинаются со значения p = 2 из-за эффекта псевдонимов.

Максимальная видимая частота изображения составляет только половину максимальной частоты пикселей (периода пространства p = 1) из-за эффекта псевдонимов. В соответствии с теоремой выборки Найквиста-Шеннона затронутая частота изображения (скорость Найквиста) составляет половину частоты пикселя.

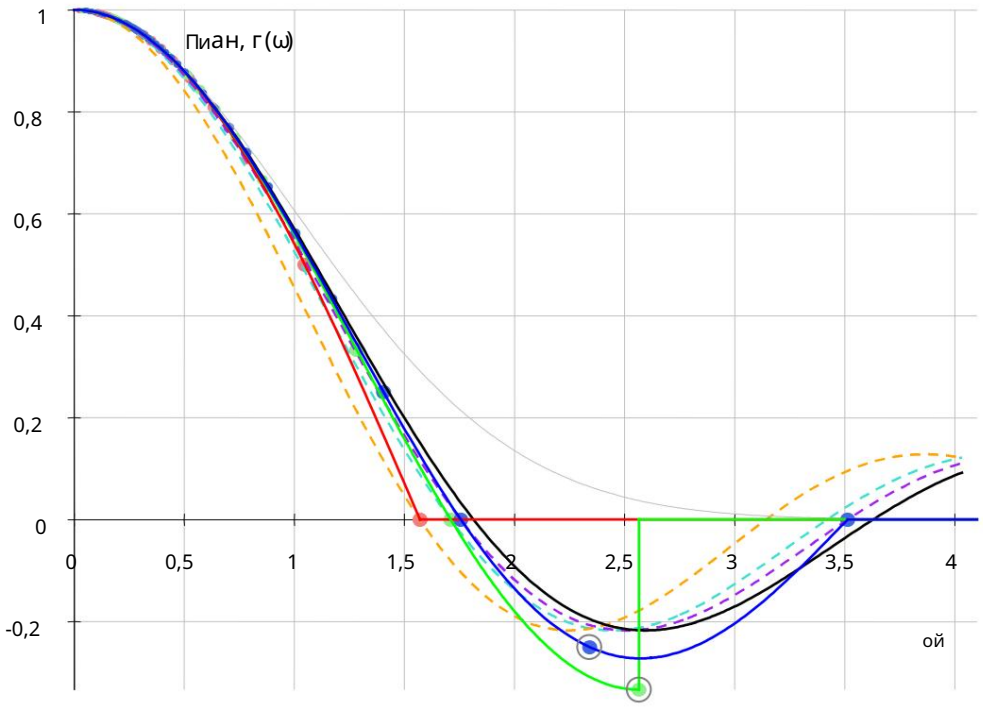


Рисунок 24: дискретный спектр 1-й степени (блочное размытие)

черный: аналоговый, a1(ω)

красный: r = 2, a1,2(ω)

зеленый: r = 3, a1,3(ω) синий:

r = 4, a1,4(ω)

Спектр выборки и пикселей: а)

дискретный: точки б)

средний: пунктирная линия в)

ограниченный частотой: линия

График и функций на рисунке 24 выглядят немного перегруженными, но изображение просто содержит все три предыдущих графика в одном для сравнения.

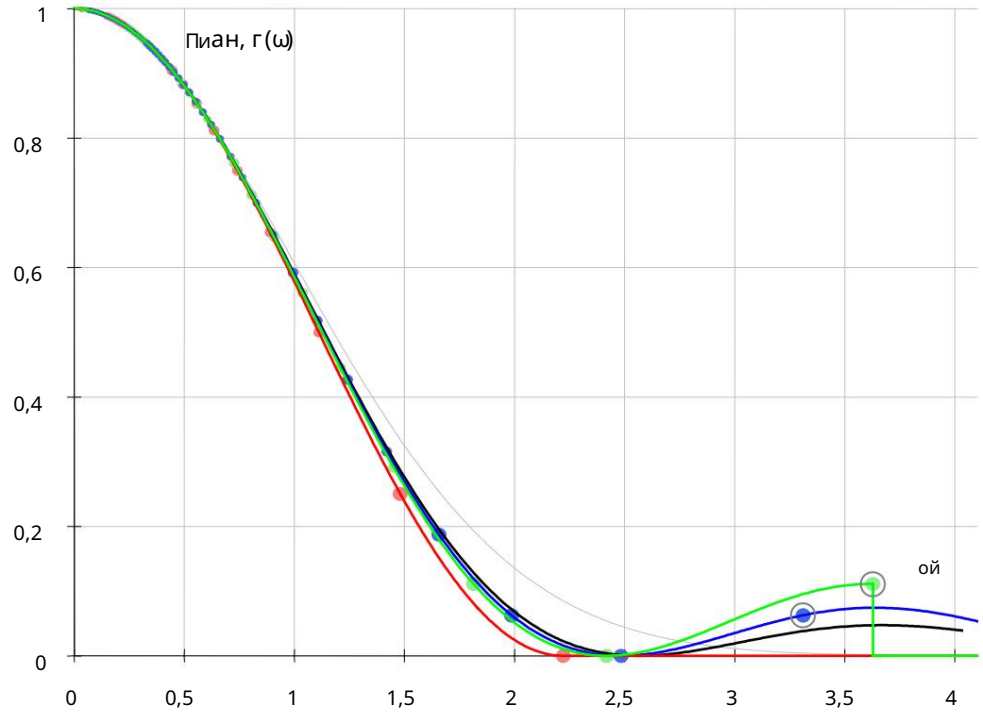


Рисунок 25: дискретный спектр 2-й степени (линейное размытие)

черный: аналог, a2(ω)

красный: r = 2, a2,2(ω)

зеленый: r = 3, a2,3(ω)

синий: r = 4, a2,4(ω)

На графиках рисунков 25–27 показан частотный спектр an,r(ω) различных степеней аппроксимации и радиусов. Нарисованы только дискретный спектр (точки) и частотно-ограниченный спектр (линия).

Черной функцией для сравнения показана аналоговая спектральная функция an(ω) уравнения (20) соответствующей степени аппроксимации. Функция Грея представляет собой спектр гауссова размытия ag(ω).

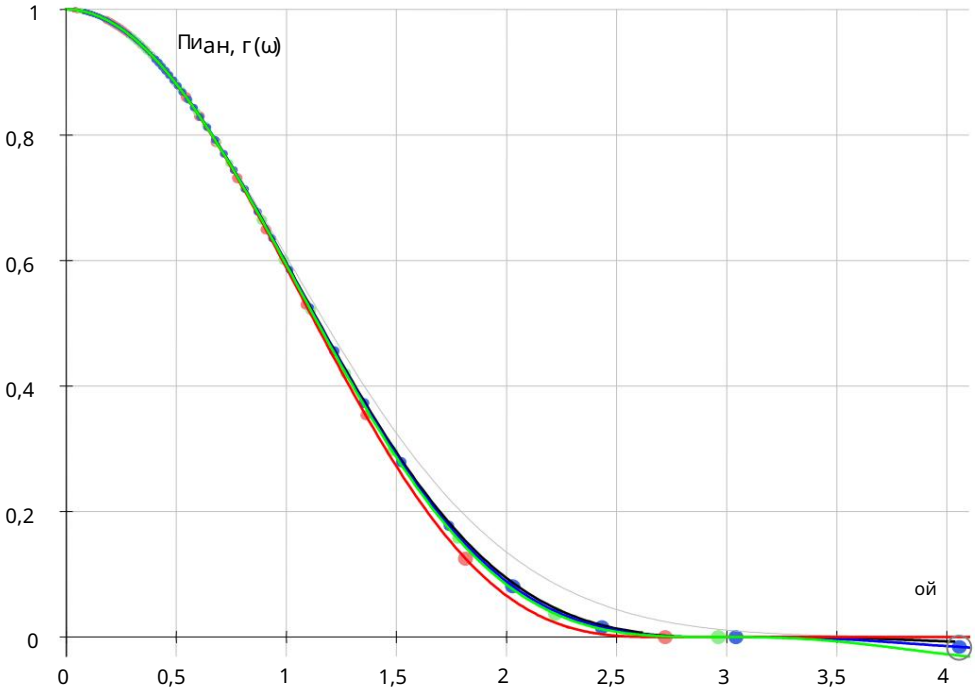


Рисунок 26: дискретный спектр 3-й степени (квадратное размытие)

черный: аналог, a3(ω)

красный: r = 2, a3,2(ω)

зеленый: r = 3, a3,3(ω)

синий: r = 4, a3,4(ω)

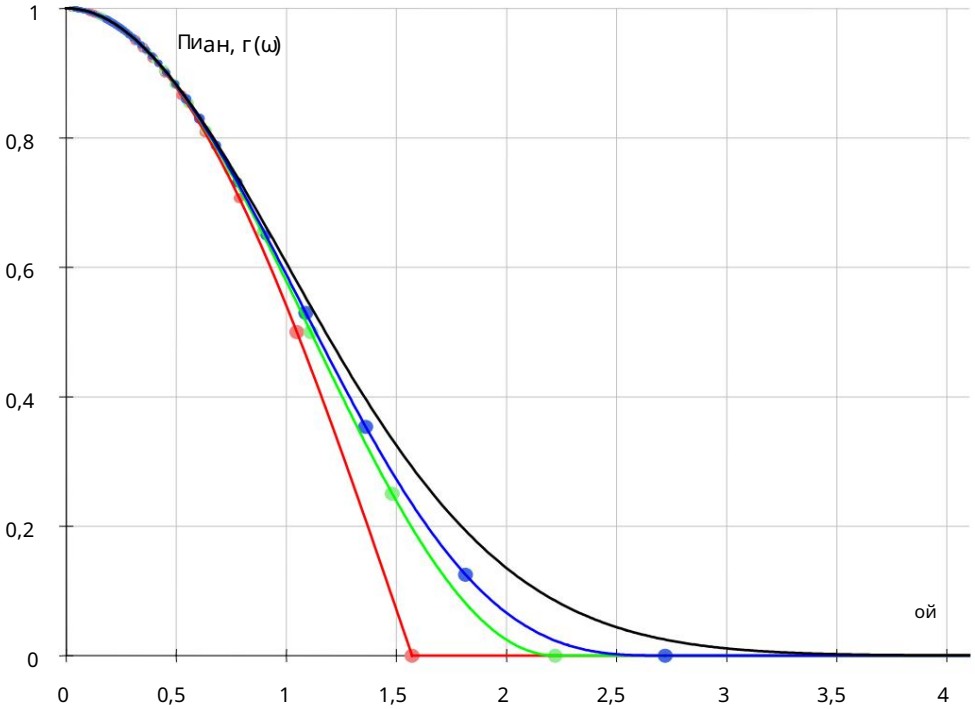


Рисунок 27: дискретный спектр радиуса r = 2 (биномиальное размытие)

черный: гауссовский, ag(ω)

красный: 1-я степень

зеленый: (прямолинейная) 2-я степень (линейная

синий: 3-я степень (квадрат)

График на рисунке 27 также поясняет, почему биномиальная последовательность подходит для расчета размытия. Дискретный спектр изображения не содержит выброса частоты

Дискретная ошибка аппроксимации:

Для любой степени аппроксимации n и радиуса r существует минимальная частота ω0 = 2, которая полностью подавляется: an,r(ω0) = 0 для дискретного периода p = r.

$$пс/p = Пи\sqrt{n(1-1/r^2)}/3$$

Что наиболее важно для аппроксимации, так это то, что для определенной частоты ω0, которая уже полностью подавлена в размытом изображении, существует максимальная ошибка более высоких частот ωp > ω0, которая не полностью подавлена (как уже рассчитано для аналогового приближения в главе 1.10).). Эта пиковая ошибка при первом выбросе частоты меч ена на графиках кружком. Этого не происходит при биномиальном размытии r = 2.

Для больших r >> 2 эта пиковая ошибка такая же, как ошибка, уже рассчитанная по уравнению(22), но, к сожалению, худший случай дискретного приближения имеет место для радиуса r = 3 и составляет : π an,3(ωp) = (–3)–n . Вторая по величине ошибка возникает при r = 4 и составляет : π an,4(ωp) = (–4)–n , что ненамного превышает ошибку аналогового приближения. Возможно, можно избежать худшего случая и всегда работать с радиусами r > 3.

2.5. Искусственные образцы изображений

Могут ли эти результаты анализа быть подтверждены образцами изображений? Не все более высокие частоты изображения подавляются согласно графикам спектра. Что произойдет , если применить алгоритм аппроксимации к специально подготовленным искусственным изображениям? Такие результаты видны на рисунке 28. Цвета соответствуют цветам графиков для определенных значений n и r.

Как показывают графики спектра, более высокие частоты подавляются. Но для более низких степеней аппроксимации, особенно для размытия прямоугольника, не все частоты подавляются достаточно хорошо. Этот недостаток виден и в образцах. Хотя частоты малых радиусов размытия на изображениях подавляются, они снова появляются при больших радиусах размытия. Этот эффект исчезает при более высокой степени приближения.

Обратите внимание, что спектр на графиках иногда бывает отрицательным. Это фазовый сдвиг частоты изображения на 180 градусов, который также виден на образцах изображений на рисунке 28.

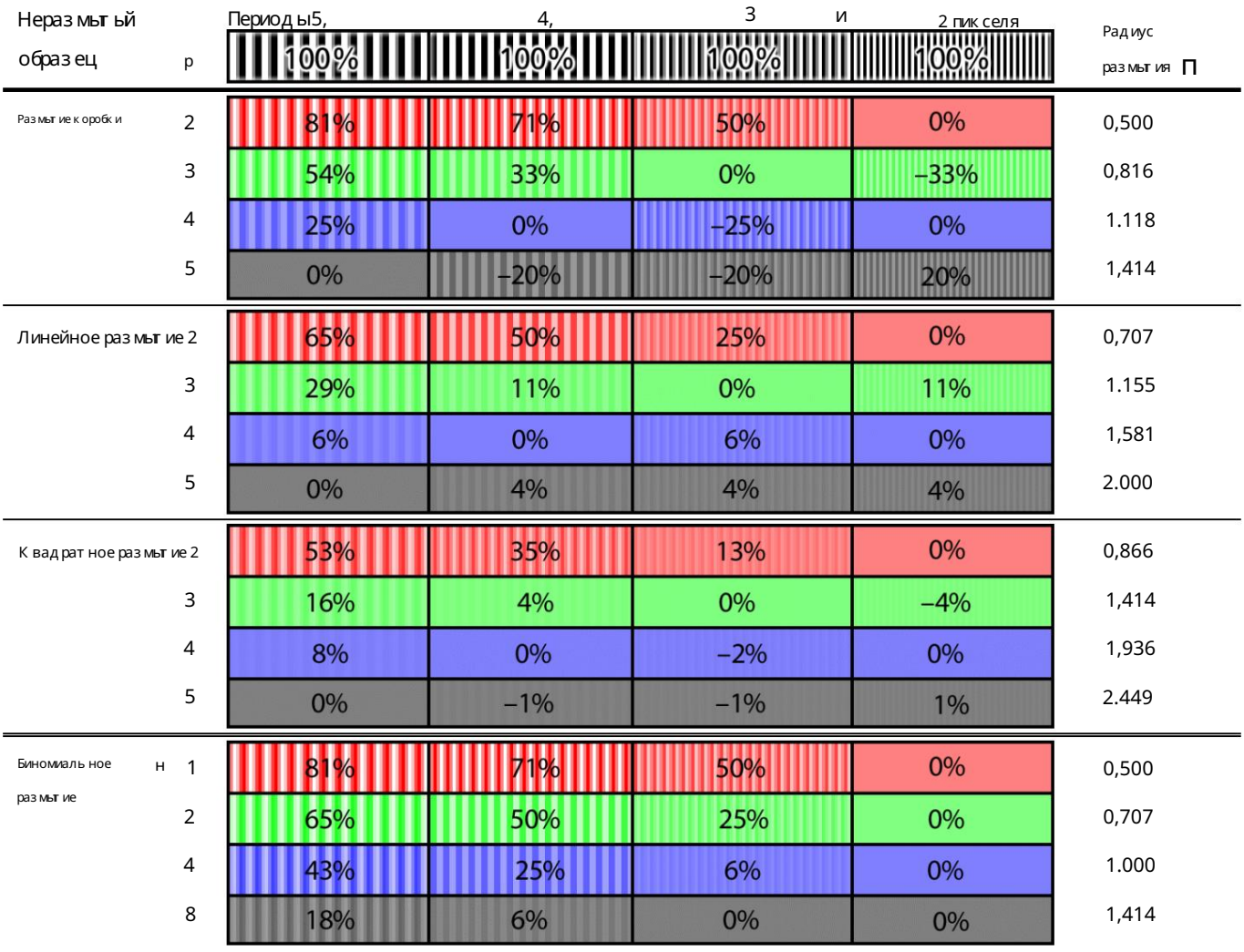


Рисунок 28: Оставшийся контраст размытых образцов изображений, содержащих определенные частоты изображения.

Числовые значения пикселей размытых изображений точно подтверждают расчет дискретного спектрального анализа. Проценты на рисунке представляют оставшийся контраст после применения расширенного биномиального фильтра размытия (без размытия = контраст 100%).

Эти примеры демонстрируют специально разработанные тестовые сценарии с одинаковой частотой и максимальным контрастом, которые редко встречаются в обычных изображениях. Но особенно низкие степени, поскольку у размытие поля для быстрых приближений, следует использовать в приложениях только с осторожностью.

Заключение:

Математический вывод спектрального анализа: если r стремится к бесконечности, дискретный спектр an,r(ω) биномиальной последовательности wn,r(k) приближается к спектру an(ω) аналоговой функции bn(x). Если степень n стремится к бесконечности, спектр an(ω) аналоговой функции bn(x) приближается к спектру ag(ω) функции Гаусса g(x).

Практический вывод спектрального анализа: максимальная погрешность частоты рассчитывается по уравнению(22), которое можно использовать как эмпирическое правило для необходимой степени аппроксимации. Для 8-битных изображений необходимая степень равна, например, n = ln 28 / ln 4,6 = 3,63, поэтому третья степень достаточно точна, чтобы избежать «повторного появления эффекта более высоких частот». (Если необходима абсолютная точность , следует принять худший случай для дискретной аппроксимации радиуса r = 3: n = ln 28 / ln 3 = 5.) Эта ошибка полностью юисчезает для биномиального размытия (r = 2).

Преимущества алгоритма: Расширенный биномиальный расчет выполняется быстро, легко вьн исляется и значительно снижает сложность программ.

Недостатки алгоритма: Более низкие степени аппроксимации все равно могут содержать высокочастотные части изображения. Целочисленные значения, обрабатываемые для вьн ислений, становятся очень большими для больших радиусов размытия (может потребоваться 64-битная арифметика или плавающая запятая). Минимальный радиус ограничен (что более актуально для более высоких степеней приближения).

2.6. Реализация. Реализация в

коде довольно проста, несмотря на сложные уравнения, используемые для объяснения теории алгоритма. Это простое расширение размытия поля.

Расширенное биномиальное размытие: для

каждого столбца выполнить
свертку строк . Для каждой

строки выполнить
свертку столбцов.

Свертка:

Инициализация различий пикселей
Для каждого пикселя
Добавьте разности для всех степеней i: Dif[i+1] += Dif[i]
Запишите следующий пиксель : Dif[градус]/вес.
Добавьте следующие биномиальные разности (для всех комбинаций i):
Dif[0] += (-1)^i * комбинация(степень ,i) * пиксель (радиус*i)

Инициализация выполняется так же, как и сама свертка, с диапазоном первого пикселя, но пиксели за пределами свертки игнорируются.

Сложность этой реализации, как и количество считываний пикселей, составляет O(n) и зависит только от степени аппроксимации n.

Примеры различных степеней аппроксимации приведены в главе исходного кода в C-коде.
Использование аппроксимации вместо точного размытия по Гауссу на самом деле не является недостатком. Прежде всего, данные изображения квантуются, поэтому даже точный расчет может быть лишь приближением к аналоговой функции Гаусса.
Во-вторых, функция Гаусса имеет бесконечный диапазон, тогда как расширенный биномиальный алгоритм имеет ограниченный диапазон, что упрощает расчет .
Алгоритм, представленный в этой статье, не ограничивается размытием по Гауссу. Его универсальная концепция может быть применена и к другим функциям.

2.6.1. пансионеры

Многие функции фильтра, такие как размытие по Гауссу, требуют для расчета окружающих пикселей. Но за границей пикселей не существует.

Самый простой способ обработки границы—просто повторить пиксель на границе за границей. Именно таким методом в этом случае обрабатывают большинство программ обработки изображений (например, Photoshop). Таким образом, доступ к пикселям за пределами границы изображения насыщен до пикселя границы

Большинство примеров программ в приложении делают то же самое, чтобы сделать алгоритм не слишком сложным. Они получают доступ к пикселям вне изображения и предполагают, что хост-программа может справиться с этой задачей.

Но, строго говоря, такая обработка не подходит для размытия. Если предположить, что изображение полностью белое с черной рамкой в один пиксель, размытие не только видит черную границу в один пиксель, потому что каждый пиксель за пределами изображения черный. Это приводит к размытой черной границе изображения.

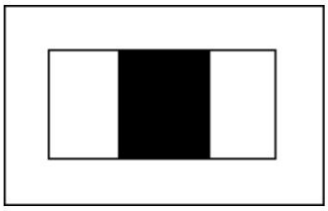
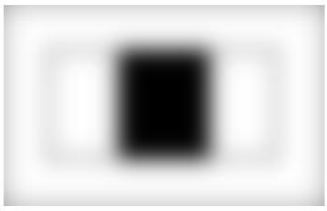


Рисунок 29а: образец изображения



б: правильное размытие границ

с: неправильное размытие границ

Правильная обработка заключалась бы отдельном взвешивании каждого пикселя вблизи границы, следовательно, игнорировании пикселей за пределами изображения в расчетном размытии. Но этот подход имеет недостаток: он более сложен и замедляет скорость выполнения.

К счастью, правильную обработку границ легко реализовать в расширенном биномиальном алгоритме первой степени. Накопленное размытие пикселей первой степени представляет собой треугольник. Этот треугольник нужно просто обрезать, чтобы избежать влияния гипотетического пикселя за границей. В главе 3.2.5 показан пример программы для правильной обработки границы

2.6.2. Радиус размытия с плавающей запятой. Недостаток

использования (расширенных) биномов для расчета размытия заключается в том, что возможны только целые числа радиусов размытия. Это не так уж и плохо, поскольку эффективный радиус бора представляет собой часть целочисленного радиуса, рассчитанного по уравнению(26). Но было бы удобно не ограничиваться целыми числами.

Такое вычисление радиуса размытия с плавающей запятой возможно путем вычисления двух последовательных радиусов r и r+1 и выполнения правильной интерполяции между результатами.

В главе 3.2.5 показан пример программы расчета радиуса размытия с плавающей запятой путем интерполяции второй степени аппроксимации.

2.6.3. Пиксельный буфер

Каждому алгоритму размытия необходим доступ к окружающим пикселям. Поскольку двумерное размытие выполняется двумя последовательными одномерными размытиями, необходим некоторый буфер изображения для хранения временных результатов первого запуска. Поскольку доступ к пикселям может быть дорогостоящим, проблему можно решить с помощью юнебольшого буфера пикселей.

Буфер пикселей сохраняет исходные пиксели в одном измерении текущего окружающего размытия. Таким образом, уже рассчитанное размытие может быть немедленно сохранено без перезаписи исходного пикселя, а доступ к исходному пикселю для расчета ускоряется за счет буфера пикселей.

2.7. Другой эффективный алгоритм размытия. Для ускорения

вычислений для увеличения радиуса размытия используются другие методы Например, можно выполнить размытие с помощью юпреобразования БПФ. Таким образом, время выполнения каждого значения равно O(log(r)). Но такой алгоритм очень сложен и требует огромных затрат на вычисления. В большинстве случаев это не стоит затраченных усилий.
Другая возможность —использовать конечный автомат (SKIPSM), но для этого требуется большой временный буфер [7].

Янг и ван Влит [4] используют рекурсивную реализацию фильтра Гаусса, сложность которого также не зависит от радиуса размытия. Недостатком этого подхода является то, что длина временного выходного потока бесконечна. Для практической реализации поток приходится где-то обрезать, что вносит ошибку в расчет. Реализация рекурсивного фильтра Гаусса Тома Фиддамана (немного улучшенная) представлена в последней главе.

Понижение дискретизации — это еще один алгоритм, который использует эффект, заключающийся в том, что фильтр размытия по Гауссу уменьшает высокочастотные компоненты изображения. Из-за этого эффекта нижних частот изображение может быть уменьшено перед применением фильтра Гаусса. Алгоритм такого фильтра размытия по Гауссу с понижающей дискретизацией прост:

понижающая дискретизация изображения с помощью блочного размытия.
Выполните обычное размытие по Гауссу с уменьшенным радиусом размытия.

Снова повысьте дискретизацию изображения с помощью билинейной интерполяции.

Пример реализации размытия по Гауссу с пониженной дискретизацией находится в приложении к исходному коду.

Трудно сравнивать время работы разных алгоритмов. Скорость выполнения очень сильно зависит от реализации. Чтобы дать приблизительную оценку, в следующей таблице показано время выполнения для разных радиусов размытия:

Время выполнения	Радиус размытия 1	10	100	O() p
Алгоритм Обычный	195%	389%	2280%	O(r)
Понижающая выборка	592%	196%	100%	$O(1/r^2)$
Расширенный бином	168%	170%	190%	O(1)
Рекурсивный	203%	205%	217%	O(1)

Таблица подтверждает мнение о том, что традиционный алгоритм имеет линейную зависимость радиуса размытия, а расширенный биномиальный и рекурсивный алгоритм зависимости не имеют. Сама понижающая выборка также не имеет зависимости, но радиус размытия делится на коэффициент понижающей дискретизации, и поэтому алгоритм работает быстрее с увеличением радиуса.

Конечно, все алгоритмы могут быть дополнительно оптимизированы

2.8. Обнаружение края

Типичным применением размытия по Гауссу является обнаружение краев. Края изображения можно обнаружить по разнице последовательных пикселей данных изображения. Но этот подход имеет тот недостаток, что шум ошибочно обнаруживается как края. Поэтому было бы удобно размыть изображение перед обнаружением края и потом дифференцировать размытие.

К счастью, эти два процесса, размытие и вывод, можно оптимизировать с учетом правил свертки.
Дифференцирование свертки — это то же самое, что выполнение свертки с помощью дифференцированной функции. Абсолютное значение (величина градиента) принимается в качестве информации о ребре: ( ) ( )

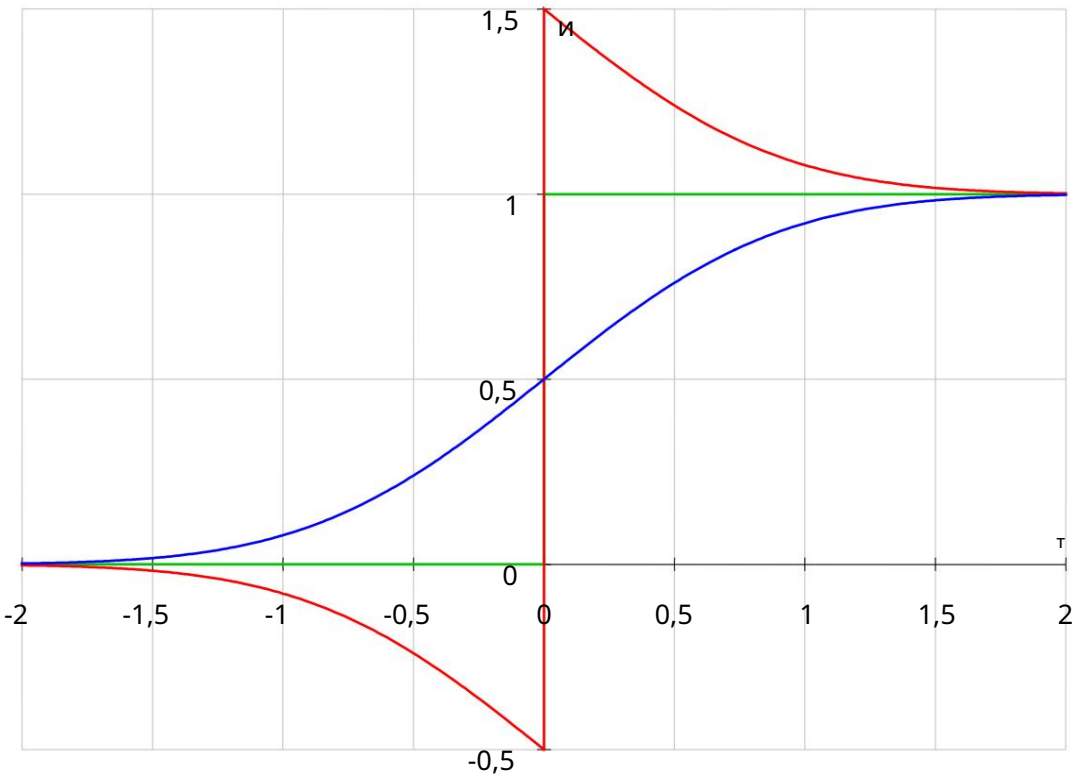
$|\Delta g| = |D(f*g)| = |f*Dg| = \sqrt{(f*g_{икс})^2 + (f*g_{и})^2}$, где D обозначает биномиальный дифференциальный оператор. Это особенно полезно, поскольку вывод уже является частью расширенного

Пример реализации обнаружения границ путем свертки с помощью расширенного биномиального алгоритма, добавленного в главе исходного кода. Программа короткая и быстрая, но обрабатывает изображение в оттенках серого. Поэтому цветные края с одинаковым значением серого не распознаются.

Этот метод обнаружения границ полезен во многих приложениях, особенно для зашумленных изображений. Его можно использовать в качестве маски для других операций, таких как размытие или повышение резкости, чтобы защитить определенные части изображения от изменений.

2.9. Пример повышения резкости

Размытие по Гауссу необходимо во многих алгоритмах фильтров. Одним из примеров применения является заточка. При резкости можно было увидеть отрицаемое размытие.



Изображение 30:

фильтр резкости и размытия

а. исходный ввод б. фильтр размытия

в. фильтр резкости

На рисунке 30 показана реакция фильтра размытия и резкости. Выходной сигнал повышения резкости: резкость = 2×вход – размытие.

Пример программы для фильтра размытия-резкости находится в приложении с исходным кодом. Размытие и резкость имеют отдельные ползунки и применяются одновременно. Если оба ползунка имеют одинаковое значение, конечно, ничего не произойдет. Разницу вносит ползунок порога. Если локальный контраст (выбранного радиуса) ниже порогового значения, применяется только размытие. Для локального контраста выше порога применяется как размытие, так и повышение резкости.
Таким образом, для низкого контраста, такого как шум, применяется фильтр размытия, тогда как более высокий контраст усиливается.

Без размытия этот фильтр работает как USM-фильтр. Кроме того, можно применить размытие, которое влияет только на низкий контраст (шум).

2.10. Рекомендации

[1] Уильям Х. Пресс, Сол А. Теукольский, Уильям Т. Веттерлинг и Брайан П. Фланнери: числовые рецепты,
Издательство Кембриджского университета, 3-е издание, 2007 г., http://sdu.ictp.it/nr/index.html.

[2] Дональд Э. Кнут: Искусство компьютерного программирования, Том. 1, 3-е издание, Аддисон Уэсли, 1997 г.

[3] Ганс Дж. Диршмид: Математические основы электротехники, Vieweg, 1992.

[4] Ян Т. Янг, Лукас Дж. ван Влит: Рекурсивная реализация фильтра Гаусса, Делфт, 1995, http://www.ph.tn.tudelft.nl/~lucas/
publications/1995/SP95TYLV/SP95TYLV.pdf

[5] Мэтью Обери, Уэйн Люк: Биномиальные фильтры, Журнал обработки сигналов СБИС, 1995,
http://www.doc.ic.ac.uk/~wl/papers/bf95.pdf

[6] Ян Т. Янг, Лукас Дж. ван Влит, Питер В. Вербек: Рекурсивный фильтр по Гауссу, ICPR 1998, http://www.ph.tn.tudelft.nl/~lucas/publications/1998/
ICPR98LVTYPV/ ICPR98LVTYPV.pdf

[7] Фредерик М. Валлица, Джон В.В. Миллерб: Эффективный алгоритм размытия по Гауссу с использованием конечных автоматов, Бостон,
1998, www.engin.umd.umich.edu/~jwvm/ece581/21_GBlur.pdf .

[8] Ян Т. Янг, Ян Дж. Гербрандс, Лукас Дж. ван Влит: Основы обработки изображений, Делфтский университет.
of Technology, V2.2, http://www.diplib.org/home222 [9] Дэйв Хейл:

Рекурсивные фильтры Гаусса, Отчет CWP 546, 2006 г., http://ww.cwp.mines.edu/Meetings/
Project06/ cwp546.pdf

[10] Сильвен Пэрис, Пьер Корнпробст, Джек Тамблин и Фредо Дюран: нежное введение в двустороннюю связь.
Фильтрация и ее применение, 2008 г., http://people.csail.mit.edu/sparis/bf_course/

2.11. Инструменты

Для разработки этого алгоритма потребовалось несколько инструментов.

- Система компьютерной алгебрыMaxima использовалась для символьной математики и графиков: http://maxima.sourceforge.net/. • Алгоритмы

были протестированыс помощью юFiltermeister, создателя графического плагина: http://www.filtermeister.com/.

3. Исходный код

3.1. Максима команды

```
/* Пакет ырисования и функции необходимыдля графиков функций */ load(functs); загрузить
(рисовать );

/* расширенная биномиальная функция ия */
b(n,x):=sqrt(3*n)/(12*(n-1)!)*sum((-1)^i*combination(n,i)*( sqrt(n/12)*x+n/2-
i)^n/abs(sqrt(n/12)*x+n/2-i),i,0,n);

/* аналоговый спектр */
a(n,x):=(sin(x*sqrt(3/n))/(x*sqrt(3/n)))^n;

/* расширенная биномиальная последовательность
*/ w(n,r,k):=sum((-1)^floor(i/r)*combination(n-1,floor(i/r))*combination(n+ki -2,ki),i,0,k);


/* дискретный спектр */
u(n,r,x):=sum(w(n,r,k)*cos(x*(kn*(r-1)/2)/sqrt(n*(r^ 2-1)/12))k,0,n*(r-1)) /r^n;


/* прямоугольная свертка */ rect(n,x):=sum((-1)^i/
(2*(n-1)!)*combination(n,i)*(x+n/2-i) ^n/abs(x+n/2-
я), я, 0, п);

/* расширенный биномиальный и гауссов график */
Plot_bi(n,r):=( k:(n*rn)/2, s:sqrt(n*(r^2-1)/12), /* среднее, отклонение */ draw2d(axis_top=false, axis_right =false,
  сетка=true, xaxis=true, yaxis=false, axis_left=false, xtics_axis=true, ytics_axis=true, цвет =красный,
    явный(w(n,r,floor(x+1/2)),x ,k-3*s,k+3*s), цвет =синий, явный(r^n*exp(-((xk)/s)^2/2)/
    sqrt(2*%pi)/s, x,к -3*c,к +3*c)))$
```

3.2. ПрограммыFiltermeister

Программы для Filtermeister были написаны для тестирования различных алгоритмов. Filtermeister — это плагин для Photoshop с языком программирования, подобным C, и простым доступом к информации об изображениях.

Filtermeister доступен для тестирования на сайте http://www.filtermeister.com/.

Если вынезнакомыс Filtermeister, несколько слов о программах на языке C:

Filtermeister (в бета-версии 7) не знает C-директив (например, #define) и пользовательских функций. Поэтому иногда код может выглядеть немного странно. Нормальных C-массивов тоже не существует . Одномерный целочисленный массив размером 1024 (N_CELLS) доступен через функции put/get. Трехмерные массивыдоступныс помощью юспециальных функций.

Пиксельный буфер использует целочисленный массив put/get и рассчитывает на то, что доступ ограничен кольцевым буфером размером N_CELLS.

Все примерыпрограмм работают с 8- и 16-битными изображениями в любом цветовом режиме.

Размытие по рамке (приближение первой степени) слишком простое, а размытие неудовлетворительно для программы

- Программы2-й, 3-й и 4-й степени приближения являются примерами реализации алгоритма и могут быть легко расширеныдо любой степени. • Программа произвольной степени аппроксимации

предназначена для сравнения влияния различных степеней на

изображений.

- Улучшенный пример второй степени правильно обрабатывает большие и малые радиусыи границы • Пример обнаружения границ и

размытия-резкости демонстрирует некоторое применение алгоритма. • Примерырекурсивного и пониженного размытия по Гауссу

—это два других алгоритма размытия.

3.2.1. Аппроксимация второй степени Программа аппроксимации

второй степени представляет собойзначительное усовершенствование метода размытия прямоугольника. Это тоже простои имеет лишь небольшую

погрешность аппроксимации. %ffp

Категория: "easy.Фильтр"
Название: "Гаусс2"
Версия: "1.1"
Имя файла: «gauss2.8bf»
Автор: «Алоис Зингль»
Авторское право:"© 2010"
URL-адрес: "http://free.pages.at/easyfilter/gauss.html"
Описание: «Вторая степень расширенного биномиального алгоритма фильтра размытия по Гауссу».

ctl(0): "Радиус (пиксель)", делитель = (i0=3), диапазон=(0, (int)(N_CELLS*i0/4.9))

OnFilterStart:
{ // устанавливаемотступы к радиус размытия
int radius = DoesProxy?(ctl(0)+zoomFactor/2)/zoomFactor:ctl(0);
NeedPadding = sqrt(6.)*radius/i0+1; BandWidth = sqrt(6) = коэффициент степени //
100+4*needPadding;
isTileable = !doingProxy; 20 вернуть // разбиваем большие изображения
ложь;
}

Для каждой плитки:
{
int radius = sqrt(6.*ctl(0)*ctl(0)/(scaleFactor*scaleFactor*i0*i0)+1);

for (z = 0; z < Z; ++z) { // для всех цветовых плоскостей

for (x = x_start; x < x_end; ++x) { // вертикальное размытие...

int dif = 0, sum = (радиус*радиус)>>1;
for (y = y_start-2*radius; y < y_end; ++y)
{ // внутренний цикл вертикального размытия
если (y >= y_start)
{
pset(x, y, z, sum/(радиус*радиус)); dif += get(y- // устанавливаем различия в
radius)-2*get(y); } else if (y+radius >= y_start) dif пикселях: // вверх размытых +1, -2, +1
-= 2*get(y);
sum += dif += p = src(x, y+radius, z); put(p, y+радиус); // накапливаем буфер размытия
пиксели пикселей следующего
} // и //
} Икс
//Обновляем индикатор выполнения и отменяем если ЭКУ клавиша была нажата
if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();

for (y = y_start; y < y_end; ++y) { // горизонтальное размытие...

int dif = 0, sum = (радиус*радиус)>>1; for (x =
x_start-2*радиус; x < x_end; ++x)
{ // внутренний цикл вертикального размытия
если (x >= x_start)
{
pset(x, y, z, sum/(радиус*радиус)); dif += get(x- // устанавливаем размытый
радиус)-2*get(x); пиксель сумма различия: вверх +1, -2, +1

```
                Гауссу } else if (x+radius >= x_start) dif -= 2*get(x);
                sum += dif += p = pget(x+radius, y, z); положить (p, x+радиус);   // накапливаем размытие пикселей //
                                                                                  буферизуем следующий пиксель
            }   // //  Икс
        }   у //
60  }   цветовая плоскость

    вернуть истину;       //Сделанный!
}

OnFilterEnd:
{
    обновлениеПрогресс (0, 1);
    вернуть ложь;
}
```

3.2.2. Приближение третьей степени

Программа третьей степени приближения достаточно точна для большинства приложений. Единственным недостатком этого приближения 3-й степени является смещение изображения на полпикселя для каждой половинырадиуса размытия пикселя. %ffp

```
Категория: "easy.Фильтр"
Название: "Гаусс3"
Версия: "1.1"
Имя файла: «gauss3.8bf»
Автор: «Алоис Зингль»
Авторское право:"© 2010"
URL-адрес:          "http://free.pages.at/easyfilter/gauss.html"
10 Описание: «Третья степень расширенного биномиального алгоритма фильтра размытия по Гауссу».

ctl(0): «Радиус (пиксель)», делитель = 2, диапазон = (0,N_CELLS/3)

OnFilterStart:
{
    int radius = DoesProxy?(ctl(0)+zoomFactor/2)/zoomFactor:ctl(0); NeedPadding = 3*радиус/2+2;

    BandWidth = 100+4*needPadding;
    isTileable = !doingProxy; 20 вернуть       // разбиваем большие изображения
ложь;
}

Для каждой плитки:
{
    int radius = sqrt(1.0+ctl(0)*ctl(0)/(scaleFactor*scaleFactor));

    для (z=0; z < Z; ++z) {                                   // для всех цветовых плоскостей

        for (x = x_start; x < x_end; ++x) {                   // размытие по вертикали для каждой строки
30
            интервал диф = 0, дер = 0; сумма с плавающей запятой = 0;
            for (y = y_start-3*radius; y < y_end; ++y) {
                                                              // внутренний цикл вертикального размытия
                если (y >= y_start) {

                    dif += 3*(get(y)-get(y+radius))-get(y-radius); pset(x, y, z, (int)(sum/   // {+1,-3,+3,-1} //
                    (радиус*радиус*радиус)));                                                 установливаем размытый пиксель
                } else if (y+radius >= y_start) dif += 3*(get(y)-get(y+radius));
                    иначе if (y+2*radius >= y_start) dif -= 3*get(y+radius);
                sum += der += dif += p = src(x, y+3*radius/2, z); put(p, y+2*радиус);   // накапливаем размытие пикселей //
                                                                                        буферизуем следующий пиксель
            }   // и //
        }       Икс
        //Обновляем индикатор выполнения и отменяем, если была нажата клавиша ЭКУ
        if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();

        for (y = y_start; y < y_end; ++y) {                   // размытие по горизонтали для каждого столбца

            интервал диф = 0, дер = 0; сумма с плавающей запятой = 0;
50          for (x = x_start-3*radius; x < x_end; ++x) {      // внутренний цикл горизонтального размытия

                если (x >= x_start) {
```

```
                dif += 3*(get(x)-get(x+radius))-get(x-radius); pset(x, y, z, (int)(sum/ // {+1,-3,+3,-1} //
                (радиус*радиус*радиус))); установливаем размытый пиксель
            } else if (x+radius >= x_start) dif += 3*(get(x)-get(x+radius));
                иначе if (x+2*radius >= x_start) dif -= 3*get(x+radius);
            sum += der += dif += p = pget(x+3*radius/2, y, z); put(p, x+2*радиус); // накапливаем размытие пикселей //
                                                                буферизуем следующий пиксель
60          } // // Икс
        } у //
        цветовая плоскость //
    } Вернуть истину; Готово!
}

OnFilterEnd:
{
    обновлениеПрогресс (0, 1);
    вернуть ложь;
70 }
```

3.2.3. Четвертая степень приближения

Приближение четвертой степени почти не отличается от истинного размытия по Гауссу и может использоваться даже для 16-битных изображений. %ffp

```
Категория: "easy.Фильтр"
Название: "Гаусс4"
Версия: "1.1"
Имя файла: «gauss4.8bf»
Автор: «Алоис Зингль»
Авторское право:"© 2010"
URL-адрес: "http://free.pages.at/easyfilter/gauss.html"
10 Описание: «Четвертая степень расширенного биномиального алгоритма фильтра размытия по Гауссу».

ctl(0): "Радиус (пиксель)", range=(0,(int)(N_CELLS/4/1.73)) // ограничиваем буфер

OnFilterStart:
{
    int radius = делать Proxy?(ctl(0)+scaleFactor/2)/scaleFactor:ctl(0); NeedPadding = 2*1,73*радиус+3;
    isTileable = !doingProxy; BandWidth = // 1,73 = sqr(12/grade) = коэффициент степени // разделение
    100+4*needPadding; больших изображений

20 вернуть ложь;
}

Для каждой плитки:
{
    int radius = sqrt(3.0*ctl(0)*ctl(0)/(scaleFactor*scaleFactor)+1); интервал x, y, r, p;

    float Weight = 1,0/((double)радиус*радиус*радиус*радиус);

    для (z=0; z < Z; ++z) { // для всех цветовых плоскостей
30
        for (x = x_start; x < x_end; ++x) { // вертикальное размытие...

            float dif = 0., der1 = 0., der2 = 0., sum = 0.;
            for (y = y_start-4*radius; y < y_end; ++y) {
                                                // инициализирующие значения для первой настройки размытия
                если (y >= y_start)
                {                       //{+1,-4,+6,-4,+1}
                    dif += -4*(get(y-радиус)+get(y+радиус))+6*get(y)+get(y-2*радиус);
                    pset(x, y, z, (int)(сумма*Вес)); } еще // установливаем размытый пиксель
40
                {
                    if (y+3*radius >= y_start) dif -= 4*get(y+radius); if (y+2*radius >= y_start) dif += // -4, //
                    6*get(y); if (y+ radius >= y_start) dif -= 4*get(y-radius); +6, // -4,(+1)}

                }
                sum += der1 += der2 += dif += p = src(x, y+2*radius-1, z); put(p, y+2*радиус); // накапливаем размытие
                                                // пиксель буфера, минимальный размер буфера: 4*радиус
            } // и //
        } Икс
50      //Обновляем индикатор выполнения и отменяем если клавиша была нажата
        if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();
```

```
    for (y = y_start; y < y_end; ++y) {                    // горизонтальное размытие...

        float dif = 0., der1 = 0., der2 = 0., sum = 0.;
        for (x = x_start-4*radius; x < x_end; ++x) {
                                                           //  набор  вверхначальные значения для первого пикселя
            если (x >= x_start)
            {                          //{+1,-4,+6,-4,+1}
60              dif += -4*(get(x-радиус)+get(x+радиус))+6*get(x)+get(x-2*радиус);
                pset(x, y, z, (int)(сумма*Вес)); } еще                  // устанавливаем размытый пиксель

            {
                if (x+3*radius >= x_start) dif -= 4*get(x+radius); if (x+2*radius >= x_start) dif +=    // -4, //
                6*get(x); if (x+ radius >= x_start) dif -= 4*get(x-radius);                            +6, // -4,(+1)}

            }
            sum += der1 += der2 += dif += p = pget(x+2*radius-1, y, z); put(p, x+2*радиус);          // накапливаем размытие //
                                                буфер пикселей, минимальный размер буфера: 4*радиус
70          }  // //  икс
        }  y //
        цветовая плоскость  //
    } Вернуть  истину;    Готово!
}

OnФильтерЭнд:
{
    обновлениеПрогресс (0, 1);
    вернуть  ложь;
80 }
```

3.2.4. Произвольная степень приближения

Степень аппроксимации можно выбирать свободно. Этот пример программы только проверяет алгоритм. Его можно использовать для сравнения различных степеней приближения эффекта размытия. В Filtermeister реализация фиксированной степени проще и быстрее. Требуется как минимум Filtermeister версии 1.0 beta 9. %ffp

```
Категория: "easy.Фильтр"
Название: «Гаусс»
Версия: "1.1"
Имя файла: "gauss.8bf"
Автор: «Алоис Зингль»
Авторское право:"© 2010"
URL:              "http://free.pages.at/easyfilter/gauss.html"
10 Описание: «Произвольная степень расширенного биномиального алгоритма фильтра размытия по Гауссу».

ctl(0): "Радиус (пик сели)", делитель  = (i0=5), диапазон = (0,255*i0)
ctl(1): «Степень  аппроксимации», диапазон = (1, 8), страница = 2
ctl(2): StaticText, "n = 1, r = 1, Radius = 0,00", size = (100,*)

OnФильтерСтарт:
{                                                          // устанавливаем отступы  к  радиус размытия
    int radius = DoesProxy?(ctl(0)+zoomFactor/2)/zoomFactor:ctl(0);
    NeedPadding = ctl(1)*radius*sqrt(3.0/ctl(1))/i0+3; 20 BandWidth =          // коэффициент  степени
100+4*needPadding;
    isTileable = !doingProxy; allocArray(0,                  // разбиваем большие
    ctl(1), 0, 0, 8); вернуть  ложь;                         изображения // двойной массив для производных размытия

}

OnCtl(n):
{           // буфер пикселей получения/вставки содержит только N_CELLS записей, поэтому ограничьте значения ползунка //
    int f = N_CELLS*i0*sqrt(ctl(1)/12.0)-i0; if (e == FME_VALUECHANGED          градусный коэффициент
    && n < 2 && ctl(0)*ctl(1) > f)
30      setCtlVal(1-n, f/ctl(n)); вернуть  ложь;          // ограничиваем необходимый буфер пикселей  к  N_CELLS

}

Для каждой плитки:
{
    int радиус = (ctl(0)+scaleFactor/2)/scaleFactor; int x, y, z, p, i, степень  = ctl(1);    //  Радиус размытия

    радиус = sqrt(12,0*радиус*радиус/градус/i0/i0+1);                  коэффициентный//
```

```
    setCtlTextv(2,"n = %d, r = %d, Radius = %.2f", ctl(1), radius,                    // степень обзора, дискретная и //
40      sqrt(ctl(1)*(radius*radius-1)/12.0)*scaleFactor);                                эффективный радиус

    для (z=0; z < Z; ++z) {                                                           // для всех цветовых плоскостей

      //    Обновить индикатор выполнения и отменить, если ЭКУ клавиша была нажата
      if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();

      for (x = x_start; x < x_end; ++x) {                                             // вертикальное размытие...

        двойной диф = 0, сумма;
50      ffillArray (0, 0,0);                                                          // устанавливаем производные к 0

        for (y = y_start-grade*radius; y < y_end; ++y) {                              // внутренний цикл вертикального размытия
                                                                                      // максимальный размер буфера: радиус*градус //
          put(src(x, y+grade*(radius-1)/2, z), y);                                    пиксель буфера

          for (p = 1, i = 0; i <= степень ; ++i) {                                    // накапливаем различия

            если (y+(степень -i)*радиус < y_start) сломать ; dif += p*get(yi*radius); // все еще на стадии инициализации,
            p = p*(i-степень )/(i+1);                                                 разрываем //
60                                                                                    пиксель // следующий бином

          }

          for (sum = dif/radius, p = степень -1; p--; ) fputArray(0, p, 0, 0, sum =  // накапливаем размытие пиксель //
            fgetArray(0, p, 0, 0)+sum/radius);                                        замедляем ФМ

          if (y >= y_start) pset(x, y, z, (int)sum);                                  // устанавливаем размытый пиксель
        }  // и //
      }          Икс
      for (y = y_start; y < y_end; ++y) {                                             // горизонтальное размытие...
70
        двойной диф = 0, сумма;
        ffillArray (0, 0,0);                                                          // устанавливаем производные к 0

        for (x = x_start-grade*radius; x < x_end; ++x) {
                                                                                      // внутренний цикл горизонтального
          put(pget(x+grade*(radius-1)/2, y, z), x);                                   размытия // пиксель буфера

          for (p = 1, i = 0; i <= степень ; ++i) {                                    // накапливаем различия

80          если (x+(степень -i)*радиус < x_start) сломать ; dif += p*get(xi*radius); // все еще на стадии инициализации,
            p = p*(i-степень )/(i+1);                                                 разрываем //
                                                                                      пиксель // следующий бином

          }

          for (sum = dif/radius, p = степень -1; p--; ) fputArray(0, p, 0, 0, sum =  // накапливаем размытие пиксель //
            fgetArray(0, p, 0, 0)+sum/radius);                                        замедляем ФМ

          if (x >= x_start) pset(x, y, z, (int)sum);                                  // устанавливаем размытый пиксель
        }  // //  Икс
90    }  у //
      цветовая плоскость //
    } Вернуть истину;    Готово!
}

ОнФильтерЭнд:
{
    обновлениеПрогресс (0, 1);
    свободныйМассив (0);
    вернуть ложь ;
100 }
```

3.2.5. Улучшенное приближение второй степени

В этом примере программа использует вторая степень приближения, как и раньше, но с двумя улучшениями. Во-первых, вместо ограничения радиуса размытия целым числом он может обрабатывать плавающие точки для небольших радиусов размытия. И во-вторых, размытие границ пикселя рассчитывается правильно (лучше, чем в PhotoShop). %ffp

```
/*
    Быстрое размытие по Гауссу расширенным биномиальным алгоритмом первой степени
```

```
 *
    простой: очень простой расчет размытия быстрый: нет
 *
    зависимости времени выполнения от радиуса размытия точный:
 *
    правильный расчет размытия пиксельных границ большой диапазон размытия радиуса с
 *
    плавающей запятой работает для всех цветовых режимов, включая
 *
    битовое и мозаичное изображение 16
10 */

Категория: "easy.Фильтр"
Название: «Размытие по Гауссу»
Версия: «1.0»
Имя файла: "gauss.8bf"
Автор: «Алоис Зингль»
Авторские права:"© 2010 GPL"
URL-адрес: "http://free.pages.at/easyfilter/gauss.html"
Описание: «Алгоритм быстрого фильтра размытия по Гауссу».
20

// i0 = CtlDivisor(0); необходимое количество ячеек для получения/вставки: 2,45*2*радиус+1
ctl(0): «Радиус (пиксель)», делитель = (i0=10), диапазон = (0, (int)(N_CELLS*i0/(2.45*2)-1))

OnFilterStart:
{                                                    // устанавливаем ступы к радиус размытия
    int Radius = ctl(0)/(doingProxy ? ZoomFactor*i0 : i0) + 1;
    NeedPadding = 2,45*Радиус+1; BandWidth =     // 2,45 = sqr(12/градус) = градусный коэффициент
    100+3*needPadding;
    isTileable = !doingProxy; 30 вернуть                    // разбиваем большие изображения
ложь;
}

Для каждой плитки:
{
    float Weight = 2,45*ctl(0)/(i0*scaleFactor), fRadius = sqrt(Weight*Weight+1);
    int x, y, z, p, Радиус = пол (fRadius); fРадиус -= Радиус; Вес =          // целочисленный радиус
    Радиус*Радиус +                                      размытия // доля радиуса
    fРадиус*(2*Радиус+1);

40 для (z = 0; z < Z; ++z)                                  // для всех цветовых плоскостей
    {
        for (x = x_start; x < x_end; ++x) {                 // вертикальное размытие...

            float sum = src(x, y_start-Radius-1, z), dif = -sum;

            for (y = y_start-2*Radius-1; y < y_end; ++y)
            {                                               // внутренний цикл вертикального
                p = источник (x, y+радиус, z); put(p,       размытия //
                y+Радиус); сумма += dif +                   следующий пиксель //
50              fRadius*p; диф += p;                        пиксель буфера // накопление радиуса пикселя

                если (y >= y_start)
                {
                    интервал s = 0, ш= 0; сумма             // коррекция размытия границ //
                    -= get(y-Radius-1)*fRadius; dif += get(y- дополнение для дробного размытия //
                    Radius)-2*get(y);                       разница: сумма -2верх +1, +1

                    // обрезаем накопленную область размытия пиксельза границей // предположим: добавленные
                    значения пикселей за границей = граница значения отрезана в
60                  p = Радиус-y; если (p                   // верхняя часть к
                    > 0)
```

```
                {
                    p = p*(p-1)/2 + fRadius*p;
                    s += get(0)*p;
                    w += p;
                }
                p = y+Radius-Y+1; if (p>                    // нижняя часть к отрезать
                0)
                {
70                  p = p*(p-1)/2 + fRadius*p;
                    s += get(Y-1)*p;
                    w += p;
                }
                pset(x, y, z, (int)((sum-s)/(Weight-w)));          // устанавливаем размытый пиксель

            } else if (y+Radius >= y_start) dif -= 2*get(y);
        }  // и //
    }       Икс
    //Обновляем индикатор выполнения и отменяем, если ESC клавиша была нажата
80  if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();

    for (y = y_start; y < y_end; ++y) {                    // горизонтальное размытие...

        float sum = pget(x_start-Radius-1, y, z), dif = -sum;

        for (x = x_start-2*Radius-1; x < x_end; ++x)
        {                                                  // внутренний цикл вертикального
            p = pget(x+Radius, y, z); put(p, x+Radius);    размытия //
            сумма += dif + fRadius*p;                      следующий пиксель //
90          dif += p;                                      пиксель буфера // накопление размытия пикселей

            если (x >= x_start)
            {
                int ervals = 0, w = 0; сумма               // коррекция размытия границ //
                -= get(x-Radius-1)*fRadius; dif += get(x-  дополнение для дробного размытия //
                Radius)-2*get(x);                          разлиumия: -2верх +1, +1

                // обрезаем накопленную область размытия пикселя за границей // обрезаем левую часть
                p = Radius-x; если (p                                  к
100             > 0)
                {
                    p = p*(p-1)/2 + fRadius*p;
                    s += get(0)*p;
                    w += p;
                }
                p = x+Radius-X+1; если (p>                 // правая часть к отрезать
                0)
                {
                    p = p*(p-1)/2 + fRadius*p;
110                 s += get(X-1)*p;
                    w += p;
                }
                pset(x, y, z, (int)((sum-s)/(Weight-w)));          // устанавливаем размытый пиксель

            } else if (x+Radius >= x_start) dif -= 2*get(x);
        }  // //  Икс
    }  y //
}  цветовая плоскость
```

```
120 возвращает истину; }          //Сделанный!


    OnFilterEnd:
    {
        обновлениеПрогресс (0, 1);
        вернуть ложь;
    }
```

3.2.6. Пример обнаружения краев Этот пример программы обнаруживает края изображений путем определения размытия. Он использует только значения серого пикселя и может быть улучшен, рассматривая все цветовые плоскости отдельно перед суммированием информации о краях. Он работает только для 8-битных изображений RGB и использует алгоритм аппроксимации размытия второй степени. %ffp

```
    Категория: "easy.Фильтр"
    Название: «Обнаружение края»
    Версия: «1.0»
    Имя файла: «edgedetection.8bf»
    Автор: «Алоис Зингль»
    Авторское право:"© 2008"
    URL-адрес:          "http://free.pages.at/easyfilter/gauss.html"
10  Описание: «Обнаружение монохромных краев путем производной свертки».
    // быстро и просто: работает только в          Режим RGB 8 бит, учитывает только значения серых пикселей.

    Поддерживаемые режимы: RGBMode

    ctl(0): «Размытие (пикселя)», делитель = 2, диапазон = (1,N_CELLS/3)

    OnFilterStart:
    {
        int Blur = MakingProxy?(ctl(0)+scaleFactor/2)/scaleFactor:ctl(0);                    // радиус размытия
20      NeedPadding = 3*Размытие/2+2;
        BandWidth = 100+4*needPadding;
        isTileable = !doingProxy; вернуть ложь;     // разбиваем большие изображения

    }

    Для каждой плитки:
    {
        int Blur = max(1, (ctl(0)+scaleFactor/2)/scaleFactor); int x, y, z, p, Размытие2 = (3*Размытие+1)/  // радиус размытия
        2;
30      двойной вес = 1,0/(Размытие*Размытие*Размытие);

        for (x = x_start; x<x_end; ++x) {                    // размытие по вертикали для каждой строки

            int dif = 0, der = 0, сумма = 0;
            for (y = y_start-3*Blur; y < y_end; ++y) {                    // инициализируем строку

                p = srcp(x, y+Blur2); p = 5*Rval(p)       // значение пикселя //
                +9*Gval(p)+2*Bval(p); put(p, y+3*Размытие); диф += p; if  конвертируем //          K      уровень серого
                (y+2*Blur >= y_start) dif -=                пиксель буфера, минимальный размер буфера: 3*Размытие аккумулирует
40              3*get(y+2*Blur);                        различия {+1, // // -3, // +3, -1}
                if (y+Blur >= y_start) dif += 3*get(y+Blur); если (y >= y_start)

                                                        //
```

```
        {
            диф -= получить (y);
            p = (der*Размытие*Вес/2)+2048; psetp(x, y,      // деривация и .. // .. размытие
            (int)(сумма*Вес)+(p<<12));                                  хранится в      12   точность битов

        } Сумма += ; + = различие;                  // накапливаем размытие пик селей
    }   // и //
}         Икс

//Обновляем индикатор выполнения и отменяем, если      ЭКУ   клавиша была нажата
если (updateProgress(doingProxy*y_start+y_end, doingProxy?y_end-y_start:Y))
    прерывание();

for (y = y_start; y < y_end; ++y) {                          // размытие по горизонтали для каждого столбца

    int dif = 0, der = 0, сумма = 0; int dif2 = 0, sum2     // размытие //
    = 0, p2; for (x = x_start-3*Blur; x < x_end; +          производное размытие //
    +x) {                                                       инициализация столбца

        p = pgetp(x+Blur2, y)&0xffffff; положить (p,          // значение пик селя //
        x+3*Размытие); диф +=                 пик сель буфера, минимальный размер буфера: 3*Blur //
        p>>12; диф2 += p&0xfff; if (x+2*Blur >= x_start) {накапливаем различия {+1, // -3,


        p= получить (x+2*Размытие); диф -= 3*(p>>12); диф2 -= 3*(p&0xfff);
        }
        if (x+Blur >= x_start) {                          //                                  +3,

        p= получить (x + размытие); диф += 3*(p>>12); диф2 += 3*(p&0xfff);
        }
        если (x >= x_start) {                             //                                  -1}

        p= получить (x); диф -= p>>12; диф2 -= p&0xfff;

        p= 2*сумма*Вес-4096; p2 =              // горизонтальные края //
        сумма2*Размытие*Вес; p =              вертикальные края // //
        мин(255, sqr(p*p+p2*p2)/12); psetp(x, y, p*0x10101);          сумма   вверх края
                                              установка пик селей RGB
        }
        сумма += ; + = различие; сумма2 += диф2;              // накапливаем размытие пик селей
    }   // //  Икс
}   и

вернуть  истину;     //Сделанный!
}

OnFilterEnd:
{
    обновлениеПрогресс (0, 1);
    вернуть  ложь ;
}
```

3.2.7. Пример размытия-резкости

В этом примере показано применение фильтра размытия для повышения резкости изображений. См. главу 2.8 для объяснения

слайдера. %ffp Категория: "easy.Filter"
Название: «Размытие — резкость »

```
Версия: «1.0»
Имя файла: «blursharpen.8bf»
Автор: «Алоис Зингль»
Авторское право:"© 2008"
URL-адрес: "http://free.pages.at/easyfilter/gauss.html"
Описание: «Улучшите контрастность путем размытия и повышения резкости».

ctl(0): «Радиус (пиксели)», val = 4, делитель = 2, диапазон = (0,340)
ctl(1): «Размытие (%)», диапазон = (0, 100), значение = 32
ctl(2): «Резкость (%)», val = 64
ctl(3): «Порог (0..255)», val = 16

ОнФильтерСтарт:
{
    int Blur = MakingProxy?(ctl(0)+scaleFactor/2)/scaleFactor:ctl(0); NeedPadding = 3*Размытие/2+2;          // радиус размытия

isTileable = !doingProxy;                          // разбиваем большие изображения
    BandWidth = 100+4*needPadding;
    вернуть ложь;
}

Для каждой плитки:
{
    int Blur = sqrt(1.0+ctl(0)*ctl(0)/(scaleFactor*scaleFactor)); int x, y, z, i, Размытие2 = (3*Размытие+1)/   // радиус размытия
    2;
    двойной вес = Размытие*Размытие*Размытие;
int th = (imageMode<GRAY16MODE?1:128)*ctl(3);                                  // порог

    для (z=0; z < Z; ++z) {                  // для всех цветовых плоскостей

        if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();

        for (x = x_start; x < x_end; ++x) {                              // размытие по вертикали для каждой строки

            двойной диф = 0., дер = 0., сумма = 0.;
            for (y = y_start-3*Blur; y < y_start; ++y) {                                  // инициализируем строку
40
                сумма += дер; дер += диф; p =                 // накапливаем размытие пикселя //
                источник(x, y+Blur2, z); put(p,          значение пикселя //
                y+3*Размытие); диф += p;                  пиксель в буфере, минимальный размер буфера: 3*Blur //
                if (y+2*Blur                          накапливаем различия {+1, // -3, //
                >= y_start) dif -= 3*get(y+2*Blur); if (y+Blur >= y_start) dif += 3*get(y+Blur);
                                                                                                    +3,     (-1)}
            }
            для (; y < y_end; ++y)
            {                                                  // внутренний цикл вертикального
50              сумма += дер; дер += диф; pset(x, y,          размытия // накапливаем размытие
                z, (int)(сумма/вес)); p = источник(x, y+Blur2, z);          пикселей // устанавливаем
                put(p, y+3*Размытие); dif +=              размытый пиксель //
                p+3*(get(y+Blur)-                          следующий пиксель // пиксель буфера
                get(y+2*Blur))-get(y);                                                  // {+1,-3,+3,-1}
            }  // и //
        }          Икс

        for (y = y_start; y < y_end; ++y) {                              // размытие по горизонтали для каждого столбца

60              двойной диф = 0., дер = 0., сумма = 0.;
```

```
        for (x = x_start-3*Blur; x < x_start; ++x) {                    // инициализируем столбец

            сумма += дер; дер += диф; p =                    // накапливаем размытие пикселя //
            pget(x+Blur2, y, z); положить (p,              значение пикселя //
            x+3*Размытие); диф += p; if                    пиксель в буфере, минимальный размер буфера: 3*Blur //
            (x+2*Blur >=                          накапливаем различия {+1, // -3, //
            x_start) dif -= 3*get(x+2*Blur); if (x+Blur >= x_start) dif += 3*get(x+Blur);
                                                                                          +3,      (-1)}
        }
70      для (; x < x_end; ++x)
        {                                                   // внутренний цикл горизонтального
            сумма += дер; дер += диф; p = (int)            размытия // накапливаем пиксельное
            (сумма/вес)-src(x, y, z); p = (2*ctl(1)*p - ctl(2)*(abs(p-th)-          размытие // локальный контраст
            abs(p+th)+2*p))/200; pset(x, y, z, src(x, y, z)+p);                                  // размытие - резкость

            p = pget(x+Blur2, y, z); положить (p,          // следующий
            x+3*Размытие); dif +=                          пиксель // пиксель буфера
            p+3*(get(x+Blur)-get(x+2*Blur))-get(x);                                  // {+1,-3,+3,-1}
        }  // //  Икс
80    }  y //
      цветовая плоскость //
    } Вернуть истину;    Готово!
}

ОнФильтрЭнд ОнФильтрЭнд:
{
    обновлениеПрогресс (0, 1);
    вернуть ложь;
}
```

3.2.8. Пример размытия по Гауссу с пониженной дискретизацией

```
90 Пример быстрого размытия по Гауссине путем понижения разрешения изображения.
    %ffp

    Название: "downsampleGauss"
    Автор: «Алоис Зингль».
    Версия: «Август 2008 г.»
    Авторские права :"© 2008 GPL"
    Категория :"easy.Filter"
    Имя файла: «downsampleGauss.8bf»
    URL-адрес          :"http://free.pages.at/easyfilter/gauss.html"
10  Описание: «Быстрый фильтр размытия по Гауссу путем понижения разрешения».

    ctl(0): «Размытие (пиксели)», делитель =10, диапазон=(0,500), страница=100, гамма=200

    ОнФильтрСтарт
    {
        double s = (doingProxy?(ctl(0)+scaleFactor/2)/scaleFactor:ctl(0))/7.4+0.9; isTileable = !doingProxy;                  // сигма

        NeedPadding = s*sqrt(log(256.0)/log(2.0));                          // Радиус разрешения 8 бит

20      i0 = max((int)s/4, 1); s = -i0*i0/(s*s); я1 =       // масштаб пониженной
        -32768; for (я = 0; я <=            выборки // расчет колоколообразной формы
        NeedPadding/i0; я+     Гаусса Matica
        +)
        {
            put((int)(32768.0*exp(i*i*s)), i);
```

```
        по Гауссу i1 += get(i)*2;
    }
    BandWidth = 100+4*needPadding+i0;
    // здесь нет allocArray, поскольку X,x_end,.. еще не обновляются после изменений
вернуть ложь;
}

Для каждой плитки:
{
    int x, y, z, Blur = NeedPadding/i0-1; int xe = (x_end-x_start)/      // радиус размытия
    i0;
    int ye = (y_end-y_start)/i0;

    если (getArrayDim(0,0)==0)
        allocArrayPad(0, xe, ye, 2, 4, Blur+1);                          // буфер изображения с пониженной дискретизацией

    for (z=0; z < planesWithoutAlpha; ++z) {

        if (updateProgress(y_start+(y_end-y_start)*z/Z, Y)) abort();

        // понижающая дискретизация блока
        for (y = -Blur; y < ye+Blur; ++y)
            for (x = -Blur; x < xe+Blur; ++x) {

                int xi, yi, s = (i0*i0)>>1;
                для (yi = 0; yi < i0; ++yi)
                    для (xi = 0; xi < i0; ++xi)
                        s += src(x*i0+xi+x_start, y*i0+y_start+yi, z);
                putArray(0, x, y, 0, s/(i0*i0));
            }

        // вертикальное размытие...
        for (x = -Blur; x <= xe+Blur; ++x) for (y = -1; y <= ye; ++y)
            {

                двойная сумма = 0;
                for (i = y-Blur; i <= y+Blur; ++i)
                    sum += getArray(0,x,i,0)*get(abs(iy));                // пиксель * масса
                putArray(0, x, y, 1, (int)(sum/i1));
            }

        // горизонтальное размытие...
        for (y = -1; y <= ye; ++y)
            for (x = -1; x <= xe; ++x) {

                двойная сумма = 0;
                for (i = x-Blur; i <= x+Blur; ++i)
                    sum += getArray(0,i,y,1)*get(abs(ix));                // пиксель * масса
                putArray(0, x, y, 0, (int)(sum/i1));
            }

        // линейная повышающая дискретизация
        for (y = -1; y <= ye; ++y)
            for (x = -1; x <= xe; ++x) {

                int xi, yi, я = i0>>1;
                int a = getArray(0, x, y, 0),                             б= getArray(0, x+1, y, 0)-a;
```

```
                int c = getArray(0, x, y+1, 0), d = getArray(0, x+1, y+1, 0)-c;
                for (yi = 0; yi < i0; ++yi)
                    for (xi = 0; xi < i0; ++xi)
                        pset(x*i0+xi+x_start+i, y*i0+y_start+yi+i, z,
                                ((b*xi+a*i0)*(i0-yi)+(d*xi+c*i0)*yi+2*i*i)/(i0*i0));
            }
        }
90  return true; }      //Сделанный!


    OnFilterEnd:
    {
        updateProgress(0, 1);
        freeArray(0);
        return false;
    }
```

3.2.9. Пример рекурсивного размытия по Гауссу

Реализация Тома Фиддамана (немного улучшенная) рекурсивного размытия по Гауссу согласно Янгу и др. [4]. %ffp

```
// Рекурсивный гауссов демо-код // реализован
Томом Фиддаманом, www.sd3.info/pf828 // из статьи: // // // 1995, 139-151. //
http://www.ph.tn.tudelft.nl/
            Название: Рекурсивная реализация фильтра Гаусса Авторы И.Т. Янг, Л.Дж. ван
            Влит в: Signal Processing, vol. 2,
~lucas/publications/1995/SP95TYLV/SP95TYLV.html      44,     нет.

10
// меняется на          Алоис Зингль: //
продолжает добавленное граничное условие вместо отраженных границ // (пиксель границы опирается за а
границу) // изображение обрабатывается в виде плиток


    Категория: "easy.filter"
    Название: «Рекурсивная гауссова»
    Версия: «1.0»
    Имя файла: «recursivegauss.8bf»
20  Автор: «Том Фиддаман, Алоис Зингл»
    URL:            "http://free.pages.at/easyfilter/gauss.html"
    Описание: «Рекурсивная реализация фильтра Гаусса».

    ctl(0): «Размытие (пиксель)»

    OnFilterStart:
    {
        isTileable = !doingProxy; needPadding =    // разбиваем большие изображения
        4*(doingProxy?(ctl(0)+scaleFactor/2)/scaleFactor:ctl(0));
30  bandWidth = 100+4*needPadding;
        return false;
    }

    Для каждой плитки:
    {
        int радиус = needPadding;
        double b3 = max(радиус-4, радиус*2/3)/4,0, wn, wn1, wn2, wn3;
```

```
    двойной b0 = 1,57825 + b3*(2,44413 + b3*(1,4281 + 0,422205*b3));
    двойной b1 = b3*(2,44413 + b3*(2,85619 + 1,26661*b3))/b0;
40 двойной b2 = (-1,4281 - 1,26661*b3)*b3*b3/b0;
    b3 = 0,422205*b3*b3*b3/b0;
    b0 = 1,0 - (b1+b2+b3);

    для (z = 0; z < Z; z++) {                    // циклически перебираем все каналы

      //Обновляем индикатор выполнения и отменяем, если ЭКУ клавиша была нажата
      if (updateProgress((1-doingProxy)*y_start*Z*2+(y_end-y_start)*(2*z+1), (doingProxy?y_end-y_start:Y)*Z*2))

          прерывание();
50
      //   делать КОЛОННЫ

      for (x = x_start; x < x_end; x++) {                    // проходим по всем строкам

        y = y_start-радиус;
        wn1 = wn2 = wn3 = источник(x,y,z);          // инициализируем значение

        в то время как (++y < y_start) {        // инициализируем строку
60
          wn = b0*src(x,y,z) + b1*wn1+b2*wn2+b3*wn3;
          wn3 = wn2; wn2 = wn1; wn1 = wn;
        }

        в то время как (++y < y_end)        // реальный проход вперед
        {
          wn = b0*src(x,y,z) + b1*wn1+b2*wn2+b3*wn3;
          pset(x,y,z, (int)wn);
          wn3 = wn2; wn2 = wn1; wn1 = wn;
        }
70
        while (++y < y_end+радиус) {                // отправляем процесс вперед

          wn = b0*src(x,y,z) + b1*wn1+b2*wn2+b3*wn3;
          put((int)wn, y);
          wn3 = wn2; wn2 = wn1; wn1 = wn;

        } wn1 = wn2 = wn3 = источник(x,y-1,z);          // инициализируем значения

        while (--y > y_end) {                // инициализируем в обратном направлении
80
          wn = b0*get(y) + b1*wn1+b2*wn2+b3*wn3;
          wn3 = wn2; wn2 = wn1; wn1 = wn;
        }

        while (y-- > y_start) {                // реальный проход назад

          wn = b0*pget(x,y,z) + b1*wn1+b2*wn2+b3*wn3;
          pset(x,y,z, (int)wn);
          wn3 = wn2; wn2 = wn1; wn1 = wn;
90      }
      }  // для Икс

      //Обновляем индикатор выполнения и отменяем, если ЭКУ клавиша была нажата
      if (updateProgress((1-doingProxy)*y_start*Z*2+(y_end-y_start)*(2*z+2),
```

```
                                    (doingProxy?y_end-y_start:Y)*Z*2))
            прерывание();

        //    делать Ряды    (немного проще, поскольку у строки не разделена плитки)
100     for (y = y_start; y < y_end; y++) {                    // проходим по всем столбцам

            int p = pget(x_end-1,y,z); wn1 = wn2 =          // запоминаем граничный пиксель
            wn3 = pget(x_start,y,z);

            for (x = x_start; x < x_end; x++) {                    // реальный проход вперед

                wn = b0*pget(x,y,z) + b1*wn1+b2*wn2+b3*wn3;
                pset(x,y,z, (int)wn);
                wn3 = wn2; wn2 = wn1; wn1 = wn;
110         }

            for (; x < x_end+radius; x++) {                    // отправляем процесс вперед

                wn = b0*p + b1*wn1+b2*wn2+b3*wn3;
                put((int)wn, x);
                wn3 = wn2; wn2 = wn1; wn1 = wn;

            } wn1 = wn2 = wn3 = p;              // инициализируем значения

120         в то время как (--x > x_end)        // инициализируем в обратном направлении
            {
                wn = b0*get(x) + b1*wn1+b2*wn2+b3*wn3;
                wn3 = wn2; wn2 = wn1; wn1 = wn;
            }

            while (x-- > x_start) {                    // реальный проход назад

                wn = b0*pget(x,y,z) + b1*wn1+b2*wn2+b3*wn3;
                pset(x,y,z, (int)wn);
130             wn3 = wn2; wn2 = wn1; wn1 = wn;
            }
        }  // для у // для
    }                C

    //Остановить обработку и применить эффект
    вернуть истину;
}

OnФильтерЭнд:
140 {
    обновлениеПрогресс (0, 1);
    вернуть ложь;
}
```