Fast and accurate recursive Gaussian filter

Martin Vicanek

1. July 2025

Abstract

Gaussian filters have many applications in DSP, smoothing, image processing, noise reduction, edge detection, etc. Inspired by Young and van Vliet[1], I propose a fast and accurate method and implementation of a recursive Gaussian filter.

1 Introduction

Thirty years ago, Young and van Vliet[1] published their paper on recursive implementation of the Gaussian Filter. I have been using it quite happily until recently, when I was challenged to assess its accuracy in a project. It turned out to be less accurate at small blur radii, so I revisited their work and, surprisingly, found out that accuracy could be improved at little cost by two modifications of their original method. The first modification is that rather than approximating the Gaussian kernel in Fourier space, we use an approximation in direct space. The second, and likely more important modification concerns discretization: While Young and van Vliet[1] use a backward difference technique (they also discuss the bilinear transform[2] as an alternative), we propose the impulse invariant method[3].

2 Continuous Gauss filter

Linear time-invariant filters may be represented as a convolution of a signal $x(t)$ with the filter's impulse response $h(t)$. In the continuous-time domain, the resulting signal $y(t)$ after applying the filter is given by

$$y(t) = \int x(\tau)h(t-\tau)d\tau. \tag{1}$$

A Gaussian filter has an impulse response of the form

$$h(t) = e^{-t^2/(2\sigma^2)} \tag{2}$$

where σ is the standard deviation. For now, $h(t)$ is normalized to unity at $t = 0$.

Since convolution is a linear process, we may split up $h(t)$ in two parts for negative and positive arguments, respectively, carry out convolutions with the one and the other, and add the results. The resulting filter action will be unchanged.

Consider a Gaussian approximation for $t \geq 0$ with three exponentials,

$$h(t) = A_0 e^{s_0 t} + A_1 e^{s_1 t} + \bar{A}_1 e^{\bar{s_1} t} \tag{3}$$

where A_0 and s_0 are real, A_1 and s_1 are complex fit parameters, and $\bar{A}_1$ and $\bar{s_1}$ their respective complex conjugates. With the values given in table 1, the function in eq.(2) is approximated with an absolute error less than $2.5 \cdot 10^{-3}$.

Index k	s_k	A_k
1	$-1.3803/\sigma$	1.4486
2	$(-1.3287 + 1.4576i)/\sigma$	$-0.2243 - 0.4814i$

Table 1: Fit parameters for Gaussian function, equations (2) and (3).

In Laplace space, convolution becomes a simple multiplication. The Laplace transform of the expression in eq.(3) reads

$$H(s) = \frac{A_0}{s - s_0} + \frac{A_1}{s - s_1} + \frac{\bar{A}_1}{s - \bar{s_1}}. \tag{4}$$

Obviously, s_0, s_1, and $\bar{s_1}$ are poles in the Laplace plane, whereas A_0, A_1, and $\bar{A}_1$ are the corresponding residues. Young and van Vliet[1] use a similar expression with one real and two complex conjugate poles, but the numerical values are different because they approximate the Gaussian in Fourier space.

3 Discrete Gauss filter

A filter may be applied to a signal of sampled values x_n at times $t = n$.[1] In that case, the equivalent of the convolution in eq.(1) becomes

$$y_n = \sum_m x_m h_{n-m} \tag{5}$$

with the filter's impulse response h_n.

There are several possibilities to obtain a discrete filter response from a continuous one. A popular choice is the bilinear transform[2], although Young and van Vliet[1] prefer backward differences to avoid ringing. In this context, however, it appears natural to go with impulse invariance[3], because it essentially samples the continuous impulse response,

$$h_n = h(t) \quad \text{at} \quad t = n. \tag{6}$$

[1] Without loss of generality, we may choose the sampling interval, usually denoted by T, as unity.

Applying impulse invariance to eq.(4) yields the filter response in z-space,

$$H(z) = \frac{A_0}{1 - p_0 z^{-1}} + \frac{A_1}{1 - p_1 z^{-1}} + \frac{\bar{A_1}}{1 - \bar{p_1} z^{-1}}, \tag{7}$$

where the poles p_0, p_1, and $\bar{p_1}$ in z-space are obtained from the poles s_0, s_1, and $\bar{s_1}$ in Laplace space,

$$p_k = \exp(s_k), \quad k = 1, 2. \tag{8}$$

The second and third term on the right side of eq.(7) may be lumped together to form a biquad. We may denote its contribution by $H_2(z)$. In the usual notation,

$$H_2(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{9}$$

with

$$b_0 = 2\mathrm{Re}(A_1), \qquad b_1 = -2\mathrm{Re}(A_1 \bar{p_1}), \qquad a_1 = -2\mathrm{Re}(p_1), \qquad a_2 = |p_1|^2. \tag{10}$$

Likewise, the first term on the right side of eq.(7) represents a 1-pole filter $H_1(z)$,

$$H_1(z) = \frac{b}{1 + a z^{-1}} \quad \text{with} \quad b = A_0 \quad \text{and} \quad a = -p_0. \tag{11}$$

We have omitted indices for a and b in eq.(11) since there is only one of each, and also to distinguish them from the biquad coefficients in eq.(9).

4 Recursive implementation

In the preceding section we have derived expressions for the Gauss filter response in z space. Since these are rational functions of z^{-1}, the filter may be implemented recursively,

$$\begin{aligned} u_n &= b x_n - a u_{n-1} \\ v_n &= b_0 x_n + b_1 x_{n-1} - a_1 v_{n-1} - a_2 v_{n-2}, \end{aligned} \tag{12}$$

where u_n is the result of the 1-pole and v_n the result of the biquad filter.

Eqs.(12) account for the positive part (including zero) of the Gauss filter h_n with $n = 0, 1, 2, \ldots$. The negative part is obtained by a similar recursion in the opposite direction (backward pass),

$$\begin{aligned} u_n^{\mathrm{back}} &= b x_n - a u_{n+1}^{\mathrm{back}} \\ v_n^{\mathrm{back}} &= b_0 x_n + b_1 x_{n+1} - a_1 v_{n+1}^{\mathrm{back}} - a_2 v_{n+2}^{\mathrm{back}}, \end{aligned} \tag{13}$$

The final result y_n of the Gauss filter is given by the sum of all contributions,

$$y_n = u_n + v_n + u_n^{\mathrm{back}} + v_n^{\mathrm{back}} - x_n. \tag{14}$$

We have to subtract x_n, refer to last term in eq.(14), because h_0 appears in both the forward and the backward pass, so x_n is counted twice.

It is common to normalize the Gaussian filter such that a constant input signal is unchanged, in other words, the filter is transparent for DC. This is achieved by dividing the result by the following constant,

$$\text{Norm} = 2\left(\frac{b}{1+a} + \frac{b_0+b_1}{1+a_1+a_2}\right) - 1 \tag{15}$$

The expression on the right hand side of eq.(15) follows from the z-transform of eq.(14) at $z = 0$, noting that u_n and v_n are the results of filters $H_0(z)$ and $H_1(z)$, respectively, acting on x_n.

Below is a code snippet implementing the proposed algorithm. It is not particularly optimized for the sake of clarity.

```
// given an array x[n] of size N and sigma
// return Gaussian blur in array y[n] of size N
// need a temporary array temp[n] of size N

// s-poles and residues
A0  =  1.4486;
A1r =  0.2243;
A1i =  0.4814;
s0  = -1.3803/sigma;
s1r = -1.3287/sigma;
s1i =  1.4576/sigma;

// z-poles
p0  = fexp(s0);
p1r = fexp(s1r)*fcos(s1i);
p1i = fexp(s1r)*fsin(s1i);

// filter constants
b  = A0;
a  = -p0;
b0 = 2*A1r;
b1 = 2*(A1R*p1R + A1i*p1i);
a1 = -2*p1r;
a2 = p1r*p1r + p1i*p1i;
invNorm = 0.5/(b/(1.0 + a) + (b0 + b1)/(1.0 + a1 + a2) - 0.5);

// forward pass ////////////////////////////////////////

// initialize recursion (depends on boundary conditions)
x1 = u = v1 = v2 = 0.0;

// perform recursion
for (n = 0; n < N; n++) {
    u = b*x[n] - a*u;
    v = b0*x[n] + b1*x1 - a1*v1 - a2*v2;
    temp[n] = u + v;
    v2 = v1;
    v1 = v;
    x1 = x[n];
```

```
}

// backward pass /////////////////////////////////////

// initialize recursion (depends on boundary conditions)
x1 = u = v1 = v2 = 0.0;

// perform recursion
for (n = N-1; n >= 0; n--) {
    u = b*x[n] - a*u;
    v = b0*x[n] + b1*x1 - a1*v1 - a2*v2;
    y[n] = ( temp[n] + u + v - x[n] )*invNorm;
    v2 = v1;
    v1 = v;
    x1 = x[n];
}
```

A note on boundary conditions: As a result of the blur effect, each point spreads out to some extent into its neighborhood. By the same token, each blurred point receives contributions from its neighbors. At the boundaries, this has two consequences: (i) Boundary points receive (unknown) contributions from outside the data set. Assumptions have to be made, depending on context. The simplest possibility, which is adopted in the code above, is to set outside data equal to zero. (ii) Boundary points spread out beyond the data set. Depending on context, one may want to take this enlargement into account (not included in the above code).

5 Results and Discussion

Figure 1 shows the impulse response (IR) for a moderate Gaussian blur with $\sigma = 1$. The present scheme is more accurate than [1], although both IRs are not too far away from the Gaussian.

Figure 2 shows the error of the two methods. At $n = 1$ and $n = -1$, the error of [1] peaks at about 10% of the IR maximum in Figure 1.

Figure 3 shows various IRs for $\sigma = 5$. Besides the present scheme and [1], the result of three consecutive box blurs with box sizes 9, 9, and 11, respectively, is included. The agreement with the Gaussian is fair for all three methods.

Figure 4 depicts the error of each method for $\sigma = 5$. The method of [1] has about the same maximum error as the three consecutive box blurs (about 4% of the IR maximum in Figure 3), while the present scheme is clearly better.

In conclusion, we presented an approximation to the discrete Gaussian filter which is more accurate than other methods with comparable complexity.

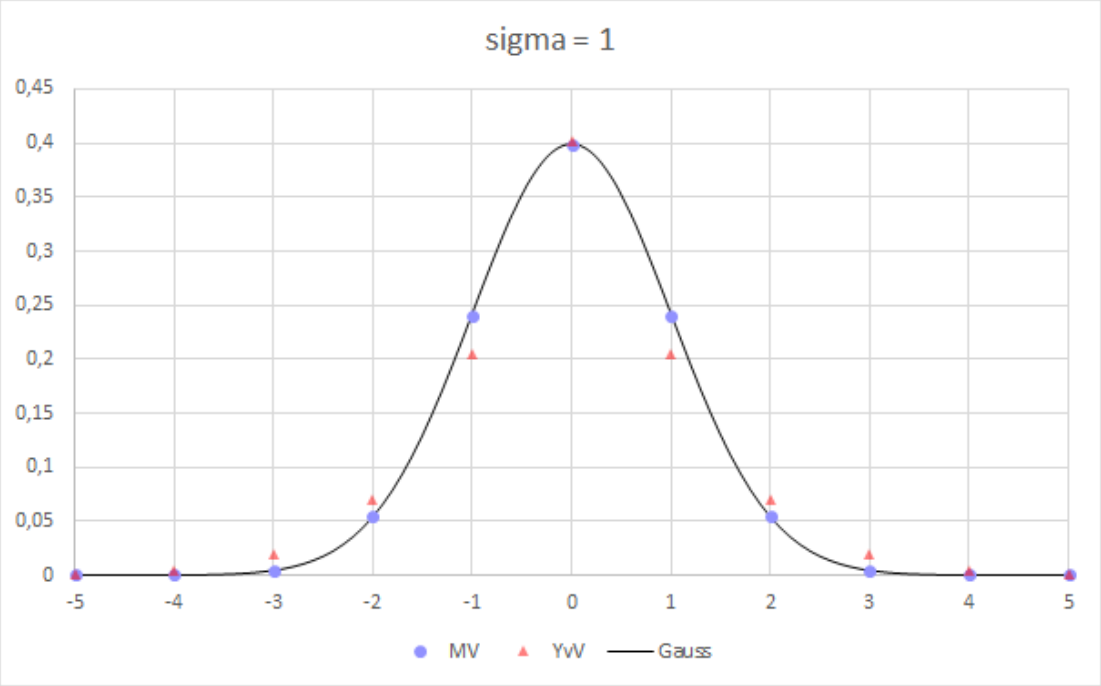

Figure 1: Gaussian blur with $\sigma = 1$.

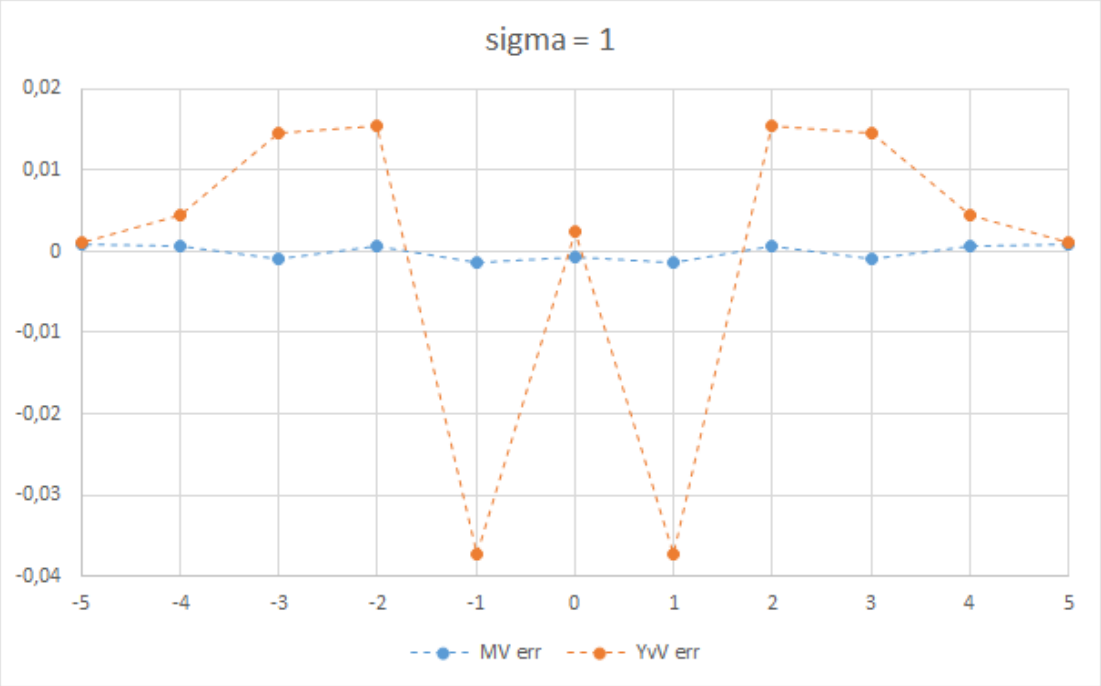

Figure 2: Gaussian blur error with $\sigma = 1$.

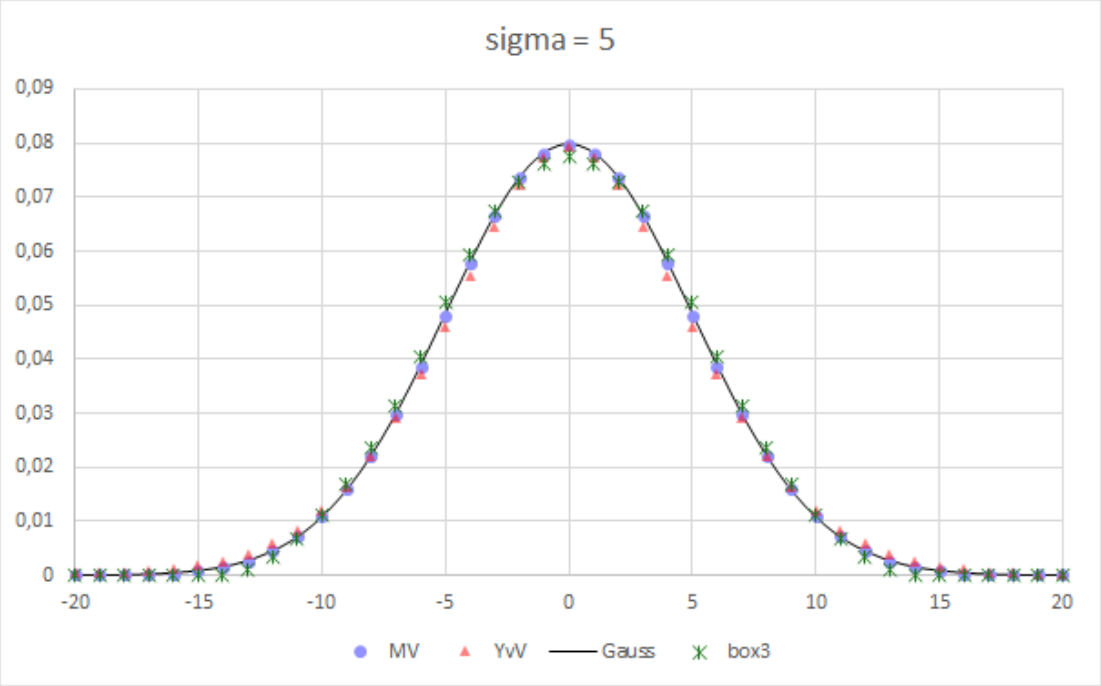

Figure 3: Gaussian blur with $\sigma = 5$.

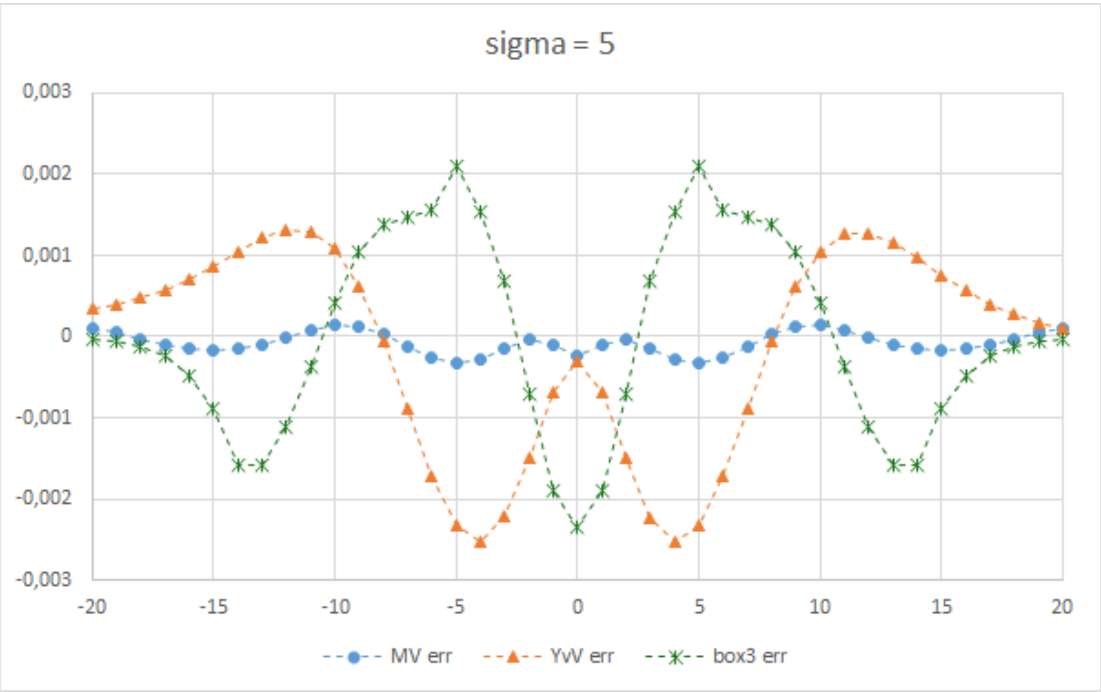

Figure 4: Gaussian blur error with $\sigma = 5$.

References

[1] Ian T. Young, Lucas J. van Vliet, Recursive implementation of the Gaussian filter, Signal Processing 44 (1995) pp. 139–151. https://repository.tudelft.nl/file/File_f6003e2f-33a1-42f2-bfa7-2593a44b711a?preview=1

[2] J. O. Smith III, Introduction to Digital Filters, 2007. https://www.dsprelated.com/freebooks/filters/Digitizing_Analog_Filters_Bilinear.html

[3] J. O. Smith III, Physical Audio Signal Processing, 2010. https://www.dsprelated.com/freebooks/pasp/Impulse_Invariant_Method.html