Adaptive Wiener Filter Based Numerical Filter with an Application to Beam Position Monitoring

KHUANJAI NACHAIYAPHUM[1], SARAWUT SUJITJORN[2†] and SUPAKORN RUGMAI[3]
[1,2]School of Electrical Engineering, Institute of Engineering, Suranaree University of Technology, Nakhon Ratchasima, THAILAND
[3]Department of Technical and Engineering, National Synchrotron Research Center, Seconded to School of Physics, Institute of Science, Suranaree University of Technology, Nakhon Ratchasima, THAILAND
[†]Corresponding author: sarawut@sut.ac.th, http://www.sut.ac.th

Abstract: - This article proposes a numerical filter having an adaptive Wiener filter (AWF) as its main component. It presents detailed investigation of the performances of the Savitzky-Golay filter (SGF) and the AWF. As a result, the AWF is superior to the SGF in terms of less distortion of the filtered waveform. The desired signal fed to the AWF can be selectively generated by using a Butterworth filter, a Savitzky-Golay filter, and a downloaded waveform, respectively. User can easily choose filter's parameters to suit their applications via a user-interface module. The proposed filter is simple, rapidly computable, and efficient to suppress noise. An application to the Siam Photon Source (Synchrotron Radiation Unit on Thailand) is also described. The filters coded in C are listed in the appendices and downloadable from our web site.

Key-Words: noise, adaptive Wiener filter, Butterworth filter, Savitzky-Golay filter, Siam Photon Source

1 Introduction

Digital filters have been applied to various engineering and scientific researches. This is due to their accuracy, flexibility and reusability with minimum modifications. Some recent developments include noise reduction in biomedical signals via digital filters [1,2]. They compared the performances of the Butterworth, elliptic, Chebyshev type I and II filters, respectively, and found out that the digital elliptic filter outperformed the others in removing power line noises and aliasing artifacts. Another interesting application is the use of the decision-based median filter (DMF), one type of non-linear filter, for the removal of impulsive noises in an image [3]. Some researchers incorporated an empirical decomposition method into a common adaptive filter to gain a very effective filter capable of handling multi-frequency interference problem occurred in partial discharge detection [4]. Cancellation of noise in acoustic signals using recursive least square (RLS) algorithm can be found in [5]. However, in the field of random noise elimination, Wiener filter has been widely applied for optimum filtering. Conventionally, the filter needs the second-order covariance for its filtering process. To obtain the covariance from direct computing of the Wiener filter is quite a computational burden. Thus, an adaptive form based on the least mean square algorithm (LMS) has been available for many years to overcome this burden [6]. The LMS algorithm uses some gradient values to adjust the filter's coefficients while minimum error is assured [7]. The filter with the LMS algorithm has been known as the adaptive Wiener filter (AWF) whose structure is shown in Fig. 1. Due to its simple structure, the computational process of filtering is straightforward and does not require any statistical parameters of the input signal. The AWF adaptively adjusts relevant coefficients according to the input signal characteristics. This type of filter is very suitable to applications in which signal and noise come within the same frequency band. The AWF requires the knowledge of a desired signal. As such, this can become a serious drawback to some applications. The Savitzky-Golay filter, introduced in the mid 1960s, has been applied for smoothing out noises in data streams [8-12]. The filter is best known for its smoothing performance, and thus widely used for scientific instruments. Even though the Wiener filter has been widely known for its seismographic application, recent developments show that its adaptive form has become a useful tool for image processing [13-15], and speech enhancement [16]. With an application in mind, our present work investigates carefully the filtering performances of the Savitzky-Golay and the adaptive Wiener filters (SGF and AWF), respectively, such that the better one could be used for removal of noise in beam-position-monitor (BPM) signals of an accelerator. The BPM signals usually contain random noise, noise from power line, and high frequency glitches caused by electronic switching devices.

To overcome the difficulties of desired signal generation for the AWF, this work proposes 3 approaches: using i) a Butterworth filter, ii) a Savitzky-Golay filter, and iii) a downloaded waveform, respectively. Each approach can be selected by the user of our proposed numerical filter via a user-interface module. This paper describes the proposed filter in details with an application to the Siam Photon Source.

2 Adaptive Wiener filter

Fig. 1 shows the structure of an adaptive Wiener filter (AWF). The filter employs the LMS method to compute and update its parameters, and weighting vectors. The error signal, $e(n)$, can be computed using equation (1).

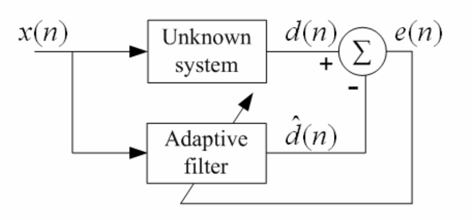


Fig.1 Structure of an adaptive Wiener filter.

$$e(n) = d(n) - \hat{d}(n) = d(n) - \mathbf{w}^T\mathbf{x}(n) \quad (1)$$

Equation (2) is used for updating the weighting vectors.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \quad (2)$$

, where

$$\mathbf{x}(n) = [x(n) \quad x(n-1) \quad \cdots \quad x(n-L+1)]^T \quad (3)$$

$$\mathbf{w}(n) = [w_1(n) \quad w_2(n) \quad \cdots \quad w_L(n)]^T \quad (4)$$

μ is an adaptive gain that can be selected according to equation (5).

$$0 < \mu < \frac{1}{\lambda_{\max}} \quad (5)$$

Fig. 2 shows the flow diagram of the LMS algorithm. It starts with initializing the variables $w(n)$ and $x(n)$. Then, it reads the current input $x(n)$ and the desired signal $d(n)$ through ADCs. The input signal is filtered through a convolution process between the input $x(n)$ and the AWF coefficients, thus resulting in the filtered signal $\hat{d}(n)$. The $\hat{d}(n)$ is compared with the signal $d(n)$, and the term $\mu e(n)$ is computed afterward. The final step is to compute the updated coefficients of the AWF. The whole process, except variable initialization, is then repeated. Our C-codes for AWF are listed in the appendix A (downloadable from http://www.sut.ac.th/engineering/electrical/carg/software/awf.cpp).

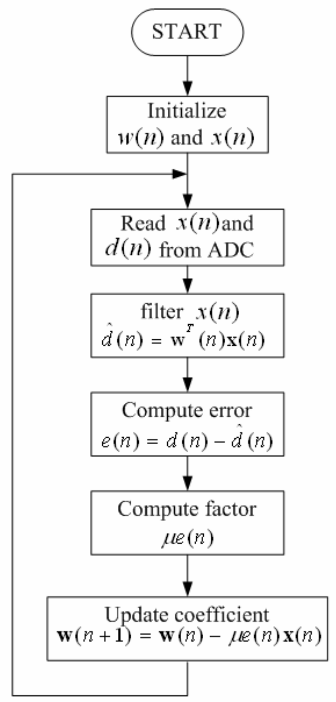


Fig. 2 Least mean square algorithm (LMS).

3 Butterworth filter

The Butterworth filter (BF) has been widely used and best known for its maximally flat characteristic [17]. This work uses the low-pass type of which magnitude can be computed according to equation (6).

$$|H(j\omega)|^2 = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2N}} \quad ; \omega_c = 2\pi f_c \quad (6)$$

Fig. 3 illustrates the magnitude responses of the low-pass BF having the order N = 1, 3, 5, 10, and 15. With a higher order, the filter characteristic moves closer to an idealistic one. To obtain a good result for our proposed numerical filter, we have used N = 4 as discussions presented in section 6.

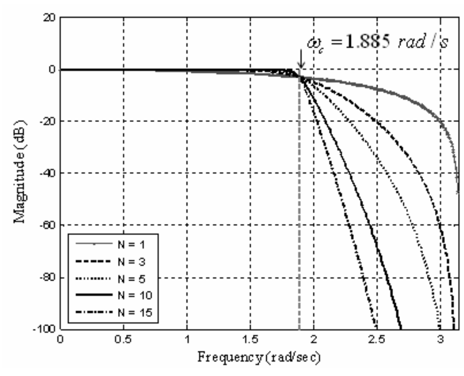


Fig. 3 Frequency responses of Butterworth filters.

4 Savitzky-Golay filter

The polynomial fit method for data smoothing and the moving window averaging method form the basis of the Savitzky-Golay filter (SGF) [8,18]. For simplification, the output of a SGF can be represented by g_i in equation (7).

$$g_i = \sum_{L=-K_L}^{K_R} c_L f_{i+L} \qquad ; i = \ldots,-2,-1,0,1,2,\ldots \quad (7)$$

The filter's coefficients, c_L, can take the form of $a_0 + a_1 i + a_2 i + \cdots a_M i^M$. The coefficient vector, $\boldsymbol{a}$ expressed as in equations (8) and (9), respectively.

$$\boldsymbol{A} \cdot \boldsymbol{a} = \boldsymbol{f} \quad ; \boldsymbol{a} = (a_0\ a_1\ a_2\ \ldots\ a_M)^T \quad (8)$$

$$(\boldsymbol{A}^T \cdot \boldsymbol{A}) \cdot \boldsymbol{a} = \boldsymbol{A}^T \cdot \boldsymbol{f} \text{ and } \boldsymbol{a} = (\boldsymbol{A}^T \cdot \boldsymbol{A})^{-1} \cdot (\boldsymbol{A}^T \cdot \boldsymbol{f}) \quad (9)$$

Since the least mean square approximation results in a linear treatment of the data, the function $\boldsymbol{f}$ in equation (9) can be replaced by a unit-vector $\mathbf{e}_L$. This yields the equation (10) for the coefficient c_L of the SGF.

$$c_L = \left\{ (\boldsymbol{A}^T \cdot \boldsymbol{A})^{-1} \cdot (\boldsymbol{A}^T \cdot \boldsymbol{e}_L) \right\}_0 = \sum_{m=0}^{M} \left\{ (\boldsymbol{A}^T \cdot \boldsymbol{A})^{-1} \right\}_{0m} L^m \quad (10)$$

, where

$$\mathbf{A} = \begin{bmatrix} (-K_L)^M & \cdots & K_L & 1 \\ \vdots & \ddots & \vdots & \vdots \\ K_R^M & \cdots & K_R & 1 \end{bmatrix} \quad (11)$$

$$L = 2K + 1 \quad ; K = K_L = K_R \quad (12)$$

There are 2 parameters, M and K, affecting the frequency response of the SGF. This is illustrated by Fig. 4 in which some magnitude ripples can be observed at high frequencies [19]. Also, a higher M results in a wider pass-band. For M = 0, the SGF acts like an averaging filter. M = K results in an all-pass characteristic. It can be noticed from Fig. 5 that the noise reduction ratios, $h_{M,K}(0)$, should be low to obtain an effective filter. Hence, $M \leq K$-2 is recommended. Our C-codes for the SGF are listed in the appendix B (downloadable from http://www.sut.ac.th/engineering/electrical/carg/software/sgf.cpp).

5 The Proposed Numerical Filter

When the desired signal, $d(n)$, is generated by a band-pass (BP) filter, our proposed filter can be represented by the block diagram in Fig. 6. The BP filter in the diagram is

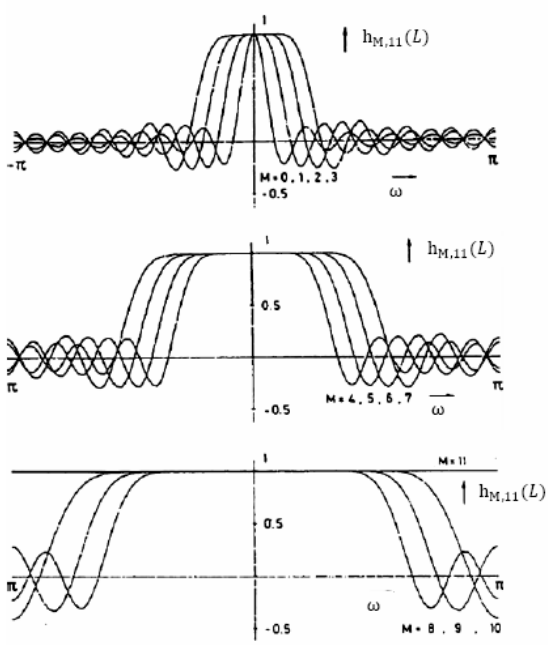


Fig. 4 Frequency responses of Savitzky-Golay filters for K = 11 and M = 0(1)11 [19].

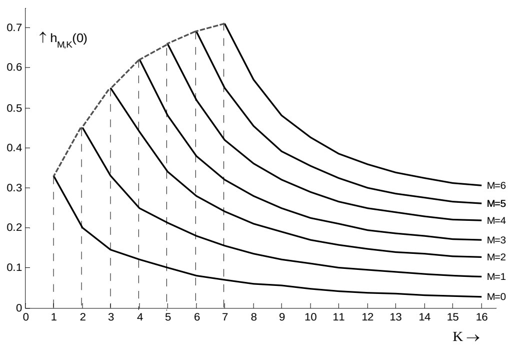


Fig. 5 Ratios of noise reduction dependence of K with M as a parameter [20].

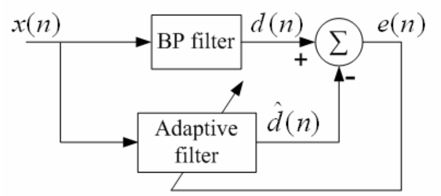


Fig. 6 Structure of an AWF using a band-pass filter to generate the desired signal, $d(n)$.

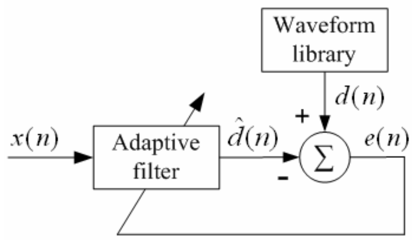


Fig. 7 Structure of an AWF using a downloaded waveform for $d(n)$.

user selective, and can be either the BF or the SGF. With a pre-defined, $d(n)$, the signal waveform of $d(n)$ can be downloaded from a library, and the diagram is reduced to that in Fig. 7. The computing process of the proposed filter is represented by the flow diagram in Fig. 8. After choosing the BP filter and its associate parameters, the user has to specify the parameter μ for the AWF to work on. Then the signal-to-noise ratio (*SNR*) is calculated according to equation (13).

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) = 20\log_{10}\left(\frac{A_{signal}}{A_{noise}}\right) \qquad (13)$$

6 Experimental Results

At first, the SGF and the AWF were tested against the Gaussian, chirp and pulse train signals to compare their effectiveness. The clean waveforms of such signals are shown in Fig. 9 a). To be used as filter inputs, these signals are mixed with white noise having *SNR* = 10 dB, and sampling frequency of 1 kHz. Fig. 9 b) shows the waveforms of the signals mixed with white noise. Fig. 10 illustrates the outputs of the SGF when M = 1 and 2, respectively, the filtered waveforms confirm the smoothing effectiveness of the SGF. However, a considerable amount of magnitude reduction in the outputs can be observed, particularly in high frequencies. This is due to the polynomial-like nature of the SGF. Fig. 11 illustrates the filtered waveforms obtained from the AWF with different values of μ. Smoothing effect is not so good as that obtained from the SGF. But, the AWF provides better results in noise reduction without degradation in signal magnitudes and waveforms. In other words, the AWF well preserves the frequency components of the original signals. It requires only a short initialization time at the beginning. This first step experiments serve to confirm that the main structure of our proposed filter should be based on the AWF. Its initialization time is not at all a drawback. Since the SGF provides very good smoothing results and behaves like a BP filter, we also adopt it for generating the desired signal $d(n)$ for the AWF.

Next, the effects of the parameters of our proposed filter are investigated. Tables 1-3 summarize the results. Firstly, the order *N* of the BF is set to 10, and the cutoff frequency f_c = 20, 30, 40, and 50 Hz, respectively. The high values of *SNR* could be expected with f_c between 30-40 Hz. Afterward, f_c = 35 Hz is chosen, and *N* = 4, 6, 8, and 10, respectively. As a result, *N* = 4 and f_c = 35 Hz can be chosen and expected to provide highly satisfactory filtering. Secondly, the parameters *M* and *K* of the SGF are studied. *M* is set to 2, and *K* = 10, 20, 30, and 40, respectively. It is found that *K* between 20-30 results in high values of *SNR*. Then, *K* is fixed to 25, and *M* = 2, 4, 6, and 8, respectively. As a results, *M* = 4 and *K* = 25 can be chosen, and expected to provide very good filtering. In terms of use of the AWF, the adaptation gain (μ) has to be chosen. For the Gaussian signal, μ = 0.0005 is used, and μ = 0.01 is used for the chirp and the pulse-train signals. Figs. 12 a) and b) illustrate the filtered signals when the BF and the SGF are used, respectively.

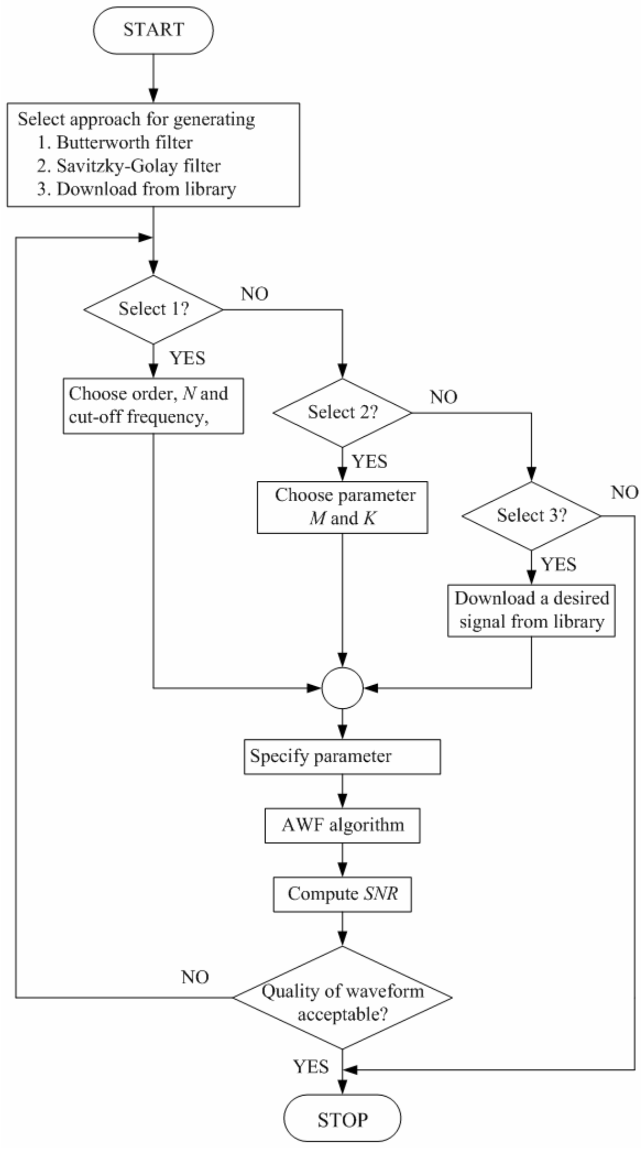
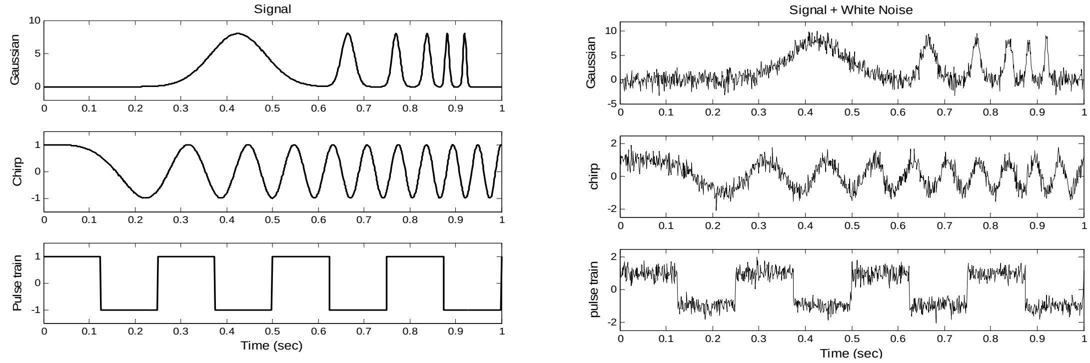
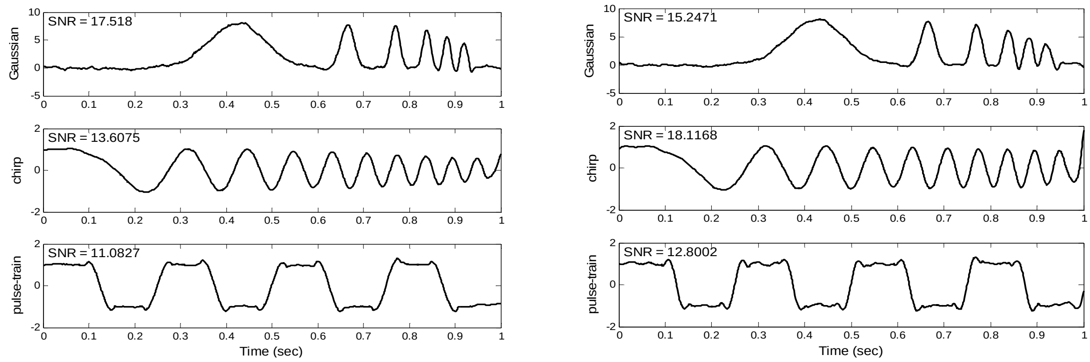


Fig. 8 Computing process of the proposed numerical filter.



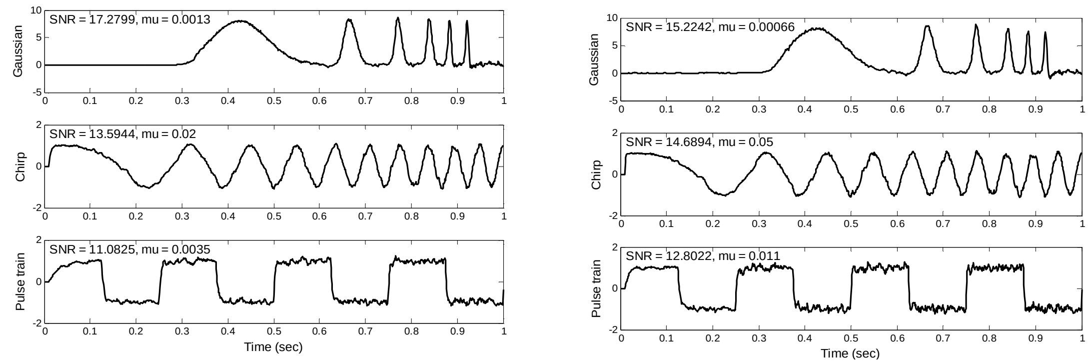
a) Clean original signals. b) Mixed with white noise.

Fig. 9 Test signals.



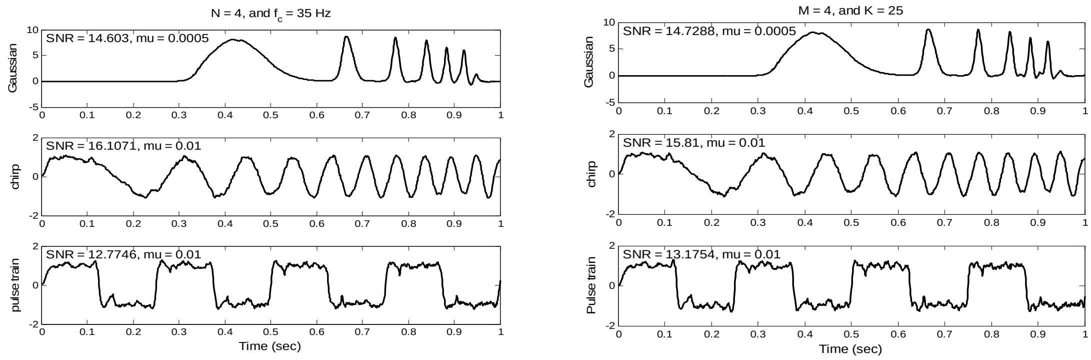
a) $M = 1$ b) $M = 2$

Fig. 10 Experimental results (SGF).



a) b)

Fig. 11 Experimental results (AWF).



a) BF ($N = 4$ and f_c =35 Hz) b) SGF ($M = 4$ and $K = 25$)

Fig. 12 Experimental results (proposed filter).

Table 1 SNRs between using BF and SGF: desired signal is Gaussian.

Butterworth filter			Savitzky-Golay filter		
N	f_c (Hz)	SNR	M	K	SNR
10	20	11.9260	2	10	14.7381
10	30	14.4249	2	20	14.7031
10	40	14.5629	2	30	14.2349
10	50	14.4230	2	40	13.1132
4	35	14.6030	2	25	14.4440
6	35	14.5342	4	25	14.7288
8	35	14.4893	6	25	14.7062
10	35	14.4691	8	25	14.7132

Table 2 SNRs between using BF and SGF: desired signal is chirp.

Butterworth filter			Savitzky-Golay filter		
N	f_c (Hz)	SNR	M	K	SNR
10	20	12.0742	2	10	15.6686
10	30	16.0311	2	20	16.0302
10	40	16.0533	2	30	16.2126
10	50	15.9007	2	40	14.9993
4	35	16.1071	2	25	16.1756
6	35	16.1287	4	25	15.8100
8	35	16.1321	6	25	15.6522
10	35	16.1353	8	25	15.5630

Table 3 SNRs between using BF and SGF: desired signal is pulse-train.

Butterworth filter			Savitzky-Golay filter		
N	f_c (Hz)	SNR	M	K	SNR
10	20	11.7292	2	10	13.1172
10	30	12.3873	2	20	13.1200
10	40	12.6736	2	30	12.5385
10	50	13.1935	2	40	11.9890
4	35	12.7746	2	25	12.8503
6	35	12.6160	4	25	13.1754
8	35	12.5040	6	25	13.0511
10	35	12.4058	8	25	12.9269

Another approach of our tests is to use some pre-defined waveforms downloaded from the library. Figs. 13-16 illustrate the test results corresponding to the downloaded Gaussian, chirp, pulse-train and sine signals, respectively. The best filtered outputs can be obtained for the cases of the Gaussian, and the pulse-train signals as shown in the Figs. 13 and 15, respectively. Fig. 14 shows that the chirp signal is the best output corresponding to the same desired signal downloaded. However, care must be taken to interpret the results as the chirp and the pulse-train signals contain some frequency components in the same bands as those of the desired signal $d(n)$. Fig. 16 illustrates a useful case of detecting a signal of certain frequency that might be a component as shown by the insets.

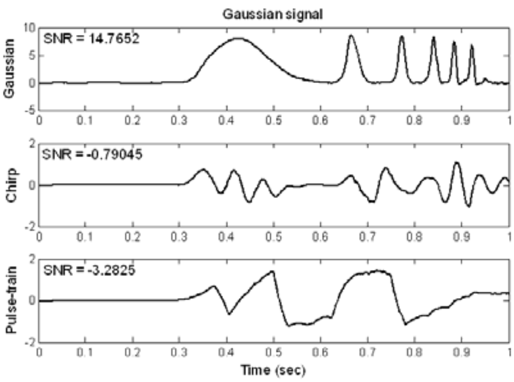


Fig. 13 Detection of Gaussian signal via downloaded waveform.

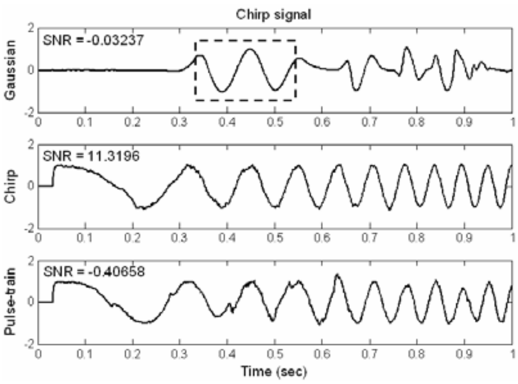


Fig. 14 Detection of chirp signal via downloaded waveform.

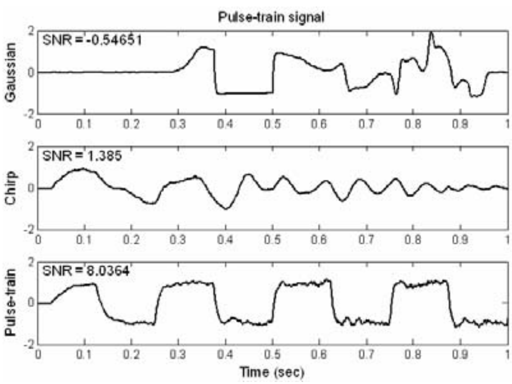


Fig. 15 Detection of pulse-train signal via downloaded waveform.

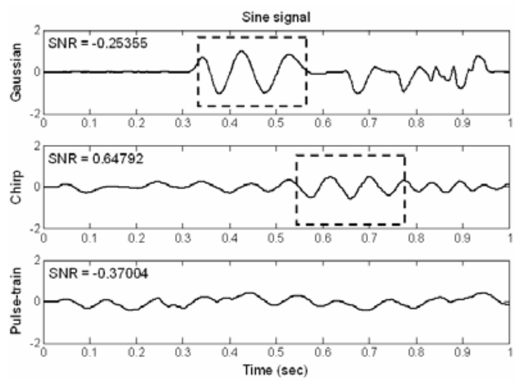


Fig. 16 Detection of sine signal via downloaded waveform.

7 Application

A useful application of our proposed numerical filter is to smooth the display of the noisy orbital signals obtained from the beam position monitor (BPM) of the Siam Photon Source (Synchrotron Radiation Unit on Thailand). Now, it operates at 1.2 GeV and is capable of generating the x-ray radiation. Fig. 17 depicts the Siam Photon Source model consisting of an injection system and a storage ring as the main components. The dark arrows in the figure indicate the 20 BPMs each of which operates on a sampling rate of 2.5 kHz. In practice, white noise is found to be the main contamination to the BPM's signals [21-24]. Some noises of deterministic nature, such as 50 Hz-noise from main supply, 100 Hz-noise or higher from some switching circuits, etc., may be found at very low amplitudes. Such deterministic noises can be sifted easily by some notch or band-stop filters. Our proposed filter is applied to the original signal obtained from one BPM as shown in Fig. 18 a). As a result, the filtered signals shown in Fig. 18 b) and c) are obtained from using the BF and the SGF to generate the signal $d(n)$, respectively. The BF parameters are N = 10 and f_c = 2.5 kHz. Those of the SGF are M = 2 and K = 10. Both types of filters provide highly satisfactory results. Selection of the BP filters is the operator's choice.

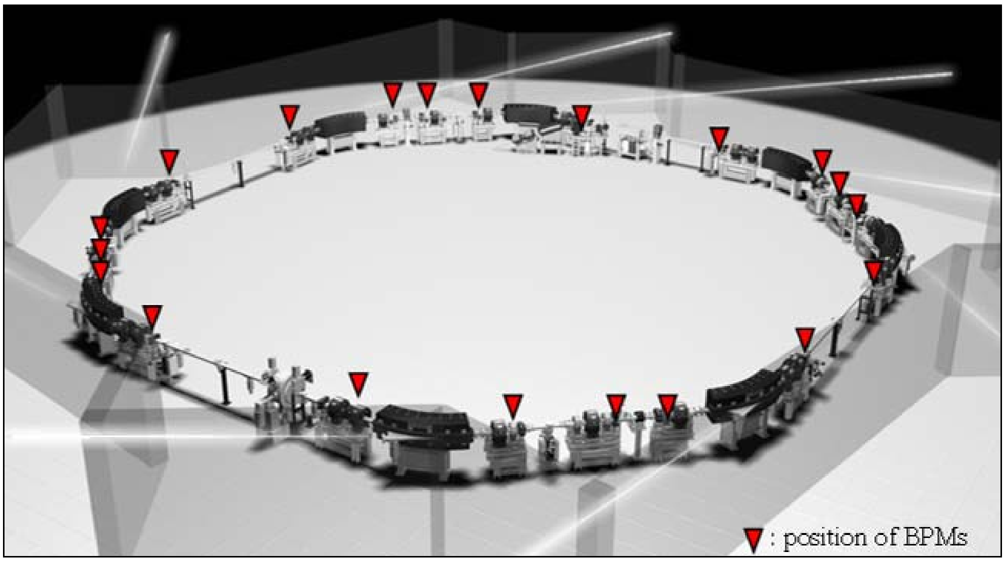
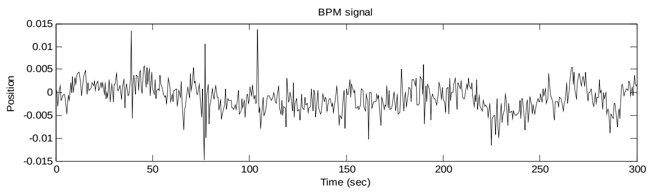
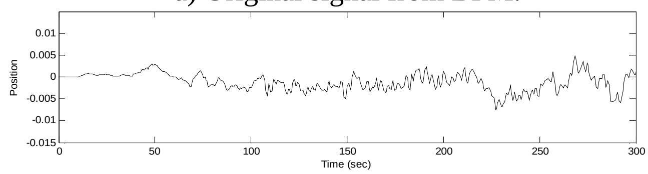


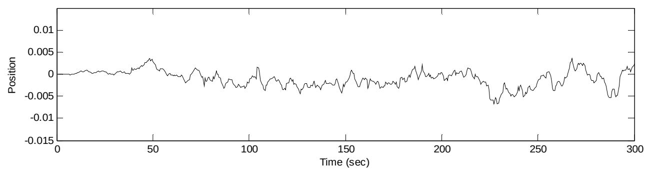
Fig. 17 Positions of 20 BPMs around the storage ring.



a) Original signal from BPM.



b) Result using BF to generate the signal $d(n)$.



c) Result using SGF to generate the signal $d(n)$.

Fig. 18 Experimental results of the proposed filter applied to a BPM.

8 Conclusion

This paper has presented a numerical filter based on the AWF's structure. Extensive studies of the SGF and the AWF performances are detailed. The AWF is superior to the SGF in terms of noise cancellation without signal magnitude degradation. Generation of the desired signal, $d(n)$, required by the AWF is user selective. Three modes of $d(n)$ generation are available: using the BF, SGF and library of waveforms, respectively. The users can also select filter's parameters to suit their applications. An application to the display smoothing of a BPM of the Siam Photon Source is described.

9 Acknowledgement

This work has been supported by the National Synchrotron Research Center (NSRC), and Suranaree University of Technology, Thailand. The first author greatly acknowledges the financial support of the NSRC for her conference participation. S. Sujitjorn greatly acknowledges the support of Suranaree University of Technology for his expenses.

References:

[1] M. S. Chawan, R. A. Agarwala and M. D. Uplane, Digital elliptic filter application for noise reduction in ECG signal, *WSEAS Trans. on Electronics*, Vol. 3, No.1, 2006, pp. 65-70.

[2] M. S. Chawan, R. A. Agarwala and M. D. Uplane, Application of Chebyshev type II digital filter for noise reduction in ECG signal, *WSEAS Trans. on Circuits and Systems*, Vol. 4, No.10, 2005, pp. 1260-1266.

[3] K. S. Srinivasan and D. Ebenezer, A new class of cascaded non-linear filter for removal of high-density impulse noise in an image, *WSEAS Trans. on Systems*, Vol. 4, No.5, 2005, pp. 682-690.

[4] C-J. Huang, Y. Qian, B-Y. Xu and X-C. Jiang, Application of EMD based adaptive filtering algorithm to suppress DSI in partial discharge detection, *WSEAS Trans. on Circuits and Systems*, Vol. 5, No.1, 2006, pp. 117-122.

[5] K. J. Kim, J. K. Kim, I. S. Kim and S. W. Nam, Nonlinear active noise control using a filtered-x RLS algorithm, *WSEAS Trans. on Systems*, Vol. 5, No.8, 2006, pp. 1802-1807.

[6] S. Haykin, *Adaptive Filter Theory*, 4th Edition, Prentice-Hall, 2002

[7] V. Saeed, *Advanced Digital Signal Processing and Noise Reduction*, John Wiley & Sons, 2006

[8] A. Savitzky and M. J. E. Golay, Smoothing differentiation of data by simplified least squares procedures. *Anal. Chem.*, Vol.36, No.8, 1964, pp. 1627-1639.

[9] A. Ergin, M. J. Vilabo, A. Tchouassi, R. Greenel and G. A. Thomas, Detection and analysis of glucose at metabolic concentration using Raman spectroscopy, *Proc. IEEE Conf. Bioengineering.*, 2003, pp. 337-338.

[10] I. Nakajima, H. Juzoji, Y. Zhao and N. Hamamoto, DSP technology in wearable satellite terminals for ETS-VIII < Savitzky - Golay smoothing filter >, *7th Int. Workshop on Digital Signal Processing Techniques for Space Communications*, 2001, pp. 131-136.

[11] S. Bakkali, Using Savitzky-Golay filtering method to optimize surface phosphate deposit “disturbances”. *CIINDET Conf. Engineering*, 2007, Vol.10, No.35, pp. 62-67.

[12] J. L. Guiñón, E. Ortega, J. García-Antón and V. Pérez-Herranz, Moving average and Savitzki-Golay smoothing filters using Mathcad. *ICEE Int. Conf. Engineering Education*, 2007, 39.

[13] I-H. Jang and N-C. Kim, Denoising of images using locally adaptive Wiener filter in wavelet domain, *IEICE Trans. on Information and Systems*, Vol. E84-D, No.4, 2001, pp. 495-501.

[14] E. Ercelebi and S. Koc, Lifting-based wavelet domain adaptive Wiener filter for image enhancement, *IEE Proc. Vision, Image and Signal Processing*, Vol. 153, No.1, 2006, pp. 31-36.

[15] R. Hardie, A fast image super-resolution algorithm using an adaptive Wiener filter, *IEEE Trans. on Image Processing*, Vol. 16, No.12, 2007, pp. 2953-2964.

[16] M.A. Akhaee, A. Ameri, F.A. Marvasti, Speech enhancement by adaptive noise cancellation in the wavelet domain. *IEEE Int. Conf. Information, Communications and Signal Processing*, 2005, pp. 719-723.

[17] I. Anatol, *Handbook of Filter Synthesis*, John Wiley & Sons, 2005

[18] H. William, P. Brian, A. Saul and T. William, *Numerical Recipes in C.* Cambridge University Press, 1992

[19] P. Steffen, On digital smoothing filters: A brief review of closed form solutions and two new filter approaches, *Circuits, Syst. Signal Processing*, Vol.5, No.2, 1986, pp.187-210.

[20] S. Jae and V. Alan, *Advanced Topics in Signal Processing*, Prentice-Hall, 1988

[21] D. S. Jonathan and E. Richard, Low noise position sensitive detector for optical probe beam deflection measurements. *Rev. Sci. Instrum*, Vol.67, No.7, 1996, pp. 2481-2484.

[22] D. S. Jonathan, Shot noise in x-ray measurements with p-i-n diodes, *Rev. Sci. Instrum*, Vol.76, No.7, 2005, pp. 076101(1) – (3).

[23] H. Schopper, *Advances of Accelerator Physics and Technologies*, World Scientific Publishing, 1993

[24] L. Johnson, J. Faust, W. Pierce, and M. Stangenes, A low frequency beam position monitor, *IEEE Trans. on Nuclear Science*, 1967, pp. 1106-1110.

Notation lists:

c_L	coefficients of SGF
$d(n)$	desired signal
$\hat{d}(n)$	output signal of AWF
$e(n)$	error signal
f_c	cutoff frequency (Hz)
f_i	original data
g_i	output signal of SGF
K_L	left datum point of the L^{th} datum
K_R	right datum point of the L^{th} datum
L	window width (odd)

M polynomial order
N filter order
$\mathbf{A}$ design matrix of the fitting problem
$\boldsymbol{a}$ coefficient vector of polynomial
$\boldsymbol{f}$ data vector
$h_{M,K}(L)$ frequency response of SGF
$h_{M,K}(0)$ ratio of the noise reduction
$\mathbf{w}(n)$ weighting vector of AWF
$\mathbf{x}(n)$ input vector of AWF
ω_c cutoff frequency
$\lambda_{\max}$ maximum eigenvalue of correlation matrix of input signal
μ adaptation gain

Appendices

A. C-codes for AWF (http://www.sut.ac.th/engineering/electrical/carg/software/awf.cpp)

```
// C-Codes for Adaptive Wiener filter (AWF)
// Created by Khuanjai Nachaiyaphum, CARG-SUT, Jan-2008
// School of Electrical engineering, Suranaree University of
Technology, Nakhon Ratchasima, THAILAND

#include <iostream>
#include <iterator>
#include <vector>
#include <algorithm>
#include <numeric>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cmath>

typedef std::vector<double> VectorT;

VectorT adaptive_wiener_filter(const VectorT& t,   // Time (sec)
      const VectorT& x,   // Input signal
      const VectorT& d,   // Desired signal
      const double mu,    // Adaptation gain or step-size parameter
      const int N)        // Window size
{
   int signal_size = x.size();
   VectorT y(signal_size, 0);
   VectorT h(N, 0);
   VectorT x1(N, 0);
   double error;

   for (int n = N; n <= signal_size; n++)
   {
      std::reverse_copy(x.begin()+n-N, x.begin()+n, x1.begin());

      y[n-1] = std::inner_product(h.begin(), h.end(), x1.begin(),  0.0);
      // errors between desired and filtered signals
      error = d[n-1] - y[n-1];
for (int j = 0; j < h.size(); j++)
      // updating filter's coefficients
      h[j] = h[j] + (mu * error * x1[j]);
   }

   return y;
}
//function for downloading waveforms
void load_data(const std::string& filename,
VectorT& t, VectorT& x, VectorT& d)
{
   std::ifstream fin(filename.c_str());
   if (!fin.is_open())
   {
      std::cerr << "File not found!\n";
      exit(1);
   }

   std::string record_line;
   double c1, c2, c3;
   // read every line from the stream
   while (std::getline(fin, record_line))
   {
      std::istringstream ss(record_line);
      if (ss >> c1 >> c2 >> c3)
      {
         t.push_back(c1);    //storing time data in #1 column
         d.push_back(c2);    //storing desired signal data in #2
                             column
         x.push_back(c3);    //storing input signal data in #3
                             column
      }
   }

   fin.close();
}
// function for recording the signals  t, d, x, y
void save_data(const std::string& filename,
            const VectorT& t, const VectorT& x,
            const VectorT& d, const VectorT& y)
{
   std::ostringstream output;
   for (int n = 0; n < t.size(); n++)
   {
      output << std::setw(25) << std::setprecision(16) <<
std::scientific << t[n] << "\t"
            << std::setw(25) << std::setprecision(16) <<
std::scientific << d[n] << "\t"
            << std::setw(25) << std::setprecision(16) <<
std::scientific << x[n] << "\t"
            << std::setw(25) << std::setprecision(16) <<
std::scientific << y[n] << std::endl;
   }
   // display the data of  t, d, x, y
   std::cout << output.str() << std::endl;

   std::ofstream fout(filename.c_str());

   fout << output.str();
   fout.close();
```

```
}
//function to calculate SNR
double calc_snr(const VectorT& d, const VectorT& y)
{
   double asignal = 0.0;
   double anoise = 0.0;
   int k;

   for (k = 0; k < d.size(); k++)
   {
      // calculate power of signal without noise
      asignal += pow(fabs(d[k]), 2.0);

      // calculate power of signal with noise
            - noise is obtained from the difference between the
            reference input and the filtered output
     anoise += pow(fabs(d[k] - y[k]), 2.0);

   }

   // calculate SNR
  double snr = 10.0 * log10(asignal / anoise);
   return snr;
}

int main(void)
{
   VectorT t, x, d, y;

   //load time-data (t), input signal data (x), and desired
  signal (d) from files: ‘gaussian_signal.txt’,‘chirp_signal.txt’,
  ‘pulse_signal.txt’

   load_data("pulse_signal.txt", t, x, d);
   // invoke AWF
   y = adaptive_wiener_filter(t, x, d, 0.005, 32);

   // record output into file ‘output.txt’
   save_data("output.txt", t, x, d, y);
   double SNR = calc_snr(d, y);

   // display SNR
   std::cout << "SNR = " << SNR << std::endl;

   return 0;
}
```

B. C-codes for SGF (http://www.sut.ac.th/engineering/electrical/carg/software/sgf.cpp)

```
// C-Codes for Savitzky-Golay filter (SGF)
// Created by Khuanjai Nachaiyaphum, CARG-SUT, Jan-2008
// School of Electrical engineering, Suranaree University of
Technology, Nakhon Ratchasima, THAILAND

#include <iostream>
#include <iterator>
#include <vector>
#include <algorithm>
#include <numeric>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cassert>
#include <cmath>
typedef std::vector<double> VectorT;
typedef std::vector<VectorT> MatrixT;


/**
 * Display matrix
 */
std::ostream& operator <<(std::ostream& xout, const
MatrixT& mat)
{
   const int row = mat.size();
   const int col = mat[0].size();

   if (row == 0)
   {
      xout << std::endl << std::setw(10) << "[ ]" << std::endl;
      return xout;
   }

   xout << std::endl;
   for(int i = 0; i < row; i++)
   {
      if (i == 0)
         xout << "  [";
      else
         xout << "   ";

      for(int j = 0; j < col; j++)
      {
         xout << std::setprecision(6) << std::scientific << mat[i][j];

         if (j != col-1)
            xout << ", ";
      }

      if (i != row-1)
         xout << ";" << std::endl;
   }
   xout << "]" << std::endl;

   return xout;
}

/**
 * QR - Orthogonal-triangular decomposition.
 */
void mat_qr(const MatrixT& a, MatrixT& q, MatrixT& r)
         // function to calculate the QR matrix
{
   const int m = a.size();
   const int n = a[0].size();
   assert(m >= n);

   int i, j, k;
```

```
    VectorT rdiag(n);
    MatrixT mat = a;

    for (k = 0; k < n; k++)
    {
       // compute 2-norm of k-th column without
under/overflow
       double norm = 0.0;
       for (i = k; i < m; i++)
          norm = hypot(norm, mat[i][k]);

       if (norm != 0.0)
       {
          // Form k-th Householder vector
          if (mat[k][k] < 0.0)
             norm = -norm;

          for (i = k; i < m; i++)
             mat[i][k] /= norm;

          mat[k][k] += 1.0;

          // apply transformation to remaining columns
          for (j = k+1; j < n; j++)
          {
             double s = 0.0;
             for (i = k; i < m; i++)
                s += mat[i][k] * mat[i][j];

             s = -s / mat[k][k];

             for (i = k; i < m; i++)
                mat[i][j] += s * mat[i][k];
          }
       }

       rdiag[k] = -norm;
    }

    // calculate the Q matrix
    q = MatrixT(m, VectorT(n));

    for (k = n-1; k >= 0; k--)
    {
       for (i = 0; i < m; i++)
          q[i][k] = 0.0;

       q[k][k] = 1.0;

       for (j = k; j < n; j++)
       {
          if (mat[k][k] != 0.0)
          {
             double s = 0.0;

             for (i = k; i < m; i++)
                s += mat[i][k] * q[i][j];
             s = -s / mat[k][k];

             for (i = k; i < m; i++)
                q[i][j] += s * mat[i][k];
          }
       }
    }

    // calculate the R matrix
    r = MatrixT(n, VectorT(n));
    for (int t = 0; t < n; t++)
    {
       for (int j = 0; j < n; j++)
       {
          if (t < j)
             r[t][j] = mat[t][j];
          else if (t == j)
             r[t][j] = rdiag[t];
          else
             r[t][j] = 0.0;
       }
    }
}

/**
 * Matrix multiplication
 */
MatrixT mat_multiply(const MatrixT& a, const MatrixT& b)
{
    const int l_row = a.size();
    const int l_col = a[0].size();
    const int r_row = b.size();
    const int r_col = b[0].size();

    // check matrix multiply rule.
    assert(l_col == r_row);

    MatrixT ret_mat(l_row, VectorT(r_col));

    for(int i = 0; i < l_row; i++)
    {
       for(int j = 0; j < r_col; j++)
       {
          double sum = 0.0;

          for (int k = 0; k < l_col; k++)
             sum += a[i][k] * b[k][j];

          ret_mat[i][j] = sum;
       }
    }

    return ret_mat;
}
VectorT filter(const VectorT& _b, const VectorT& _a, const
VectorT& x)                // calculate signal convolution
       {
          assert(_b.size() != 0 &&
                 _a.size() != 0 &&
                 x.size() != 0);
    assert(_a[0] != 0.0);

    int n, nb, na;
```

```
   VectorT b = _b;
   VectorT a = _a;
   VectorT y(x.size());

   // If a[0] is not equal to 1, the filter coeffcients are
   normalized by a[0]
   if (a[0] != 1.0)
   {
      for (nb = 0; nb < b.size(); nb++)
         b[nb] /= a[0];

      for (na = 0; na < a.size(); na++)
         a[na] /= a[0];
   }
   VectorT outputs(a.size());
   VectorT inputs(b.size());
   for (n = 0; n < x.size(); n++)
   {
      outputs[0] = 0.0;
      inputs[0] = x[n];

      for (nb = b.size()-1; nb > 0; nb--)
      {
         outputs[0] += b[nb] * inputs[nb];
         inputs[nb] = inputs[nb-1];
      }
      outputs[0] += b[0] * inputs[0];

      for (na = a.size()-1; na > 0; na--)
      {
         outputs[0] += -a[na] * outputs[na];
         outputs[nb] = outputs[nb-1];
      }

      y[n] = outputs[0];
   }

   return y;
}

/**
 * Savitzky Golay filter
 *
 *
 * PARAMETERS:
 *  M : Cut-off frequency (between 0 to 6)
 *
 *  K : Noise reduction factor
 *        M = 0;  2 <= K <= 15
 *        M = 1;  3 <= K <= 25
 *        M = 2;  4 <= K <= 35
 *        M = 3;  5 <= K <= 55
 *        M = 4;  6 <= K <= 65
 *        M = 5;  7 <= K <= 38
 *        M = 6;  8 <= K <= 20
 */
VectorT savitzky_golay_filter(const VectorT& t, // Time (sec)
              const VectorT& x,   // Input signal
              const int M,          // Cut-off frequency
              const int K)          // Noise reduction
factor
{
   int i, j;
   int npoints = x.size();
   VectorT y;
   MatrixT q, r;
   VectorT c(2*K+1);

   MatrixT a(2*K+1, VectorT(M+1, 1.0));
           // function to calculation the A matrix

   VectorT kk;
   for (i = -K; i <= K; i++)
      kk.push_back(i);

   for (j = M-1; j >= 0; j--)
   {
      for (i = 0; i < 2*K+1; i++)
         a[i][j] = kk[i] * a[i][j+1];
   }

   mat_qr(a, q, r);                        // function to
calculate the QR matrix

   for (i = 0; i < 2*K+1; i++)
      c[i] = q[i][M] / r[M][M];         // calculate the
filter's coefficients from the QR matrix

   std::reverse(c.begin(), c.end());   // c(2*K+1:-1:1)
   y = filter(c, VectorT(1, 1.0), x);   // calculate
signal convolution
   for (i = 0; i < npoints; i++)
   {
      if (i < K)
         y[i] = x[i];
      else if (i < npoints-K)
         y[i] = y[i+K];
      else
         y[i] = x[i];
   }

   return y;
}

void load_data(const std::string& filename,
           VectorT& t, VectorT& d, VectorT& x)
           // function to download signal data of t, x, d
{
   std::ifstream fin(filename.c_str());
   if (!fin.is_open())
   {
      std::cerr << "File not found!\n";
      exit(1);
   }

   std::string record_line;
   double c1, c2, c3;
   // read every line from the stream
   while (std::getline(fin, record_line))
```

```
    {
        std::istringstream ss(record_line);
        if (ss >> c1 >> c2 >> c3)
        {
            t.push_back(c1);    // store time data in #1 column
            d.push_back(c2);    // store desired signal data in #2
                                column
            x.push_back(c3);    // store input signal data in #3
                                column
        }
    }

    fin.close();
}

void save_data(const std::string& filename,
            const VectorT& t, const VectorT& d,
            const VectorT& x, const VectorT& y) // function to
record the data: t, d, x, y
{
    std::ostringstream output;
    for (int n = 0; n < t.size(); n++)
    {
        output << std::setw(25) << std::setprecision(16) <<
std::scientific << t[n] << "\t"
            << std::setw(25) << std::setprecision(16) <<
std::scientific << d[n] << "\t"
            << std::setw(25) << std::setprecision(16) <<
std::scientific << x[n] << "\t"
            << std::setw(25) << std::setprecision(16) <<
std::scientific << y[n] << std::endl;
    }
// std::cout << output.str() << std::endl; // display the data t, x, y, d

    std::ofstream fout(filename.c_str());

    fout << output.str();
    fout.close();

}
// calculate SNR
double calc_snr(const VectorT& d, const VectorT& y)
{
    double asignal = 0.0;
            double anoise = 0.0;
            int k;

            for (k = 0; k < d.size(); k++)
            {
                // calculate power of signal without noise
                asignal += pow(fabs(d[k]), 2.0);


        // calculate power of signal with noise – noise is obtained
            from the difference between the reference and the
            filtered signals
        anoise += pow(fabs(d[k] - y[k]), 2.0);
    }

    double snr = 10.0 * log10(asignal / anoise); // calculate SNR
    return snr;
}

int main(void)
{
    VectorT t, d, x, y;

    load_data("pulse_signal.txt", t, d, x);       // load signal data
from file: 'pulse_signal.txt'

    y = savitzky_golay_filter(t, x, 2, 30);       // invoke SGF
    save_data("output.txt", t, d, x, y);          // record outputs
into file

    double SNR = calc_snr(d, y);                  // calculate SNR
    std::cout << "SNR = " << SNR << std::endl; // display SNR

    return 0;

}
```