Automotive Innovation (2019) 2:178–189
https://doi.org/10.1007/s42154-019-00065-1

Numerical Implementation of High-Order Vold–Kalman Filter Using Python Arbitrary-Precision Arithmetic Library

Linhe Ge[1] · Fangwu Ma[1] · Jinzhu Shi[1] · Hongbin Yin[1] · Ying Zhao[1]

Received: 31 January 2019 / Accepted: 1 May 2019 / Published online: 22 August 2019

Abstract
The Vold–Kalman (VK) order tracking filter plays a vital role in the order analysis of noise in various fields. However, owing to the limited accuracy of double-precision floating-point data type, the order of the filter cannot be too high. This problem of accuracy makes it impossible for the filter to use a smaller bandwidth, meaning that the extracted order signal has greater noise. In this paper, the Python mpmath arbitrary-precision floating-point arithmetic library is used to implement a high-order VK filter. Based on this library, a filter with arbitrary bandwidth and arbitrary difference order can be implemented whenever necessary. Using the proposed algorithm, a narrower transition band and a flatter passband can be obtained, a good filtering effect can still be obtained when the sampling rate of the speed signal is far lower than that of the measured signal, and it is possible to extract narrowband signals from signals with large bandwidth. Test cases adopted in this paper show that the proposed algorithm has better filtering effect, better frequency selectivity, and stronger anti-interference ability compared with double-precision data type algorithm.

Keywords Noise order analysis · Vold–Kalman filter · Arbitrary-precision arithmetic library

1 Introduction

Order analysis is used in a variety of applications, from basic plant machinery testing to complex automotive engine testing. It is often combined with acoustic measurements to analyze the noise, vibration, and harshness (NVH) qualities of an engine or vehicle as a whole. Automotive engineers often use order tracking methods for product evaluation and development, design validation, production testing, quality evaluation, and trouble shooting. The paper [1] reviewed some basic ideas behind different kinds of order analysis methods and compared their main advantages and limitations.

Particularly, the VK filter is a vital technique in order analysis. The main framework of the filter have been basically presented [2–4] and then on this basis, the algorithm appears in almost all NVH-related commercial software presently on the market. Because of the importance of these researches, the algorithm is also named after the main author. Based on the conventional Kalman filter, the VK filter was proposed by Vold and Leuridan in 1993 [2]. The authors found that normal tracking filters (analog or digital implementations) have limited resolution in situations where the reference RPM is rapid. Thus, the authors proposed the application of nonstationary Kalman filters for the tracking of periodic components in such noise and vibration signals, namely, the VK filter. Vold then introduced the mathematical background of the VK filter [3]. This was the first presentation of the second-generation algorithm and its theoretical foundations. This new algorithm enables the simultaneous estimation of multiple orders, effectively decoupling close and crossing orders. In another paper published the same year [4], the authors explored the advantages of the filter in detail, including: (1) RPM estimation accuracy, even for fast-changing events such as gear shifts, (2) higher-order Kalman filters, with improved shapes for extracting modulated orders, and (3) decoupling of close and even crossing orders by use of multiple RPM references. Vold et al. [5] reported the development of a new VK filter for decoupling interacting orders in multi-axle systems. Based on the foundation of the first- and second-generation VK filters, a number of studies provided further understanding of the mathematical derivation of the filters, the physical meaning of

✉ Fangwu Ma
mikema_pro@163.com

1 State Key Laboratory of Automotive Simulation and Control, Jilin University, 5988, Renmin Ave., Changchun 130025, China

their parameters, and the relationship between these parameters [6–11]. Herlufsen et al. [6] described the filter characteristics of the VK order tracking filter, investigating both the frequency response and time response of their time–frequency relationship. Pelant et al. [7] derived the detailed formulation of the filter, while Tuma [8] reported the bandwidth calculation formula for the 1st–4th order of the filter and established the relationship between the bandwidth and the weight coefficient. As an extension, the present paper presents a calculation formula for the filter bandwidth at arbitrary orders. Blough [9] explained the formulations and behavior of the filter in very straightforward and practical terms through the use of both equations and example datasets. Čala and Beneš [10] described the implementation of both first- and second-generation VK order tracking filters, with a focus on optimizing the calculations. It is worth mentioning that Vold et al. [11] considered the bandwidth of the VK filter to be limited by the numerical conditioning of the least-squares normal equations associated with its application. This suggests that even narrower bandwidths may be achieved by a direct least-squares solution using a banded version of the QR algorithm. As a more general approach, the present study adopts another method based on an arbitrary-precision floating-point arithmetic library. Similar to the VK filter, the method of transforming the filter problem into an optimization problem appears, although this has not yet become the mainstream approach. Amadou et al. [12] proposed another method that converges quickly and provides a small estimation error compared to those used for the linear time-invariant model. An offline processing approach using the preconditioned conjugate gradient method has also been proposed [13]. Pan et al. [14] further studied theoretical basis, numerical implementation and parameter of VK filter. It should be pointed out that VK filter is very useful in many fields of sound analysis, even fault diagnosis [15, 16].

When the order of the VK filter is large, it has the advantages of a flat passband and a fast-changing transition band. At the same time, smaller filter bandwidths can better isolate the influence of noise and other vibration signals. However, both cases result in larger matrix values, even beyond the precise representation of double-precision data. None of the research mentioned above can solve this problem effectively. This is the main problem considered in this paper—how to obtain higher-order and narrower passband VK filters for an arbitrary desired order and bandwidth.

To better understand how this problem is solved, there sections are introduced as follows. Section 2 describes the relevant VK filter in detail and gives the pseudocode of the related algorithm. Using an arbitrary-precision floating-point arithmetic library, the extension of this VK filter to any higher orders is explained. Section 3 presents the results from three test cases to verify the effectiveness of the algorithm. Finally, Sect. 4 gives the conclusions to this study.

2 VK Filter Formulation

This section discusses the VK filter algorithm and its numerical implementation in detail. The numerical implementation of the algorithm is given in the form of pseudocode. Readers can use the Python programming language and its arbitrary-precision numerical operation library to realize this algorithm, or contact the author to obtain the source code. The author will accept any requests with an open mind, and later relevant source code will be released on GitHub.

2.1 Analytical Solution of VK Filter

In this section, the analytical solution of the VK filter will be derived. Firstly, two basic equations, i.e., data equation and structural equation, correspond to the measurement equation and state equation of the standard Kalman filter, respectively. Based on minimizing the weighted sum of squares of the error terms of the two equations, the analytic solution of the VK filter is obtained.

The recorded signal $y(t)$ is modeled as follows:

$$y(n) = x(n)\exp(j\theta(n)) + \eta \tag{1}$$

where $\theta(n)$ is the phase of an ideal signal, that is, the integral of the angular velocity, $\theta(n) = \sum_{i=0}^{n} \omega(i)\Delta t$, η represents the noise item. The complex envelope $x(n)$ represents the signal amplitude and phase fluctuations. This equation is named the data equation.

The matrix representation is

$$y - Cx = \eta \tag{2}$$

and the square of the error vector norm is

$$\eta^T\eta = (y - Cx)^T(y - Cx) \tag{3}$$

The structural equation can be described by the following higher-order difference equation:

$$\Delta^{n_\mathrm{d}} x(n) = \varepsilon(n) \tag{4}$$

where Δ represents the difference computation symbol, n_d is the order of the difference equation, and the value of $\varepsilon(n)$ should be sufficiently small so that the complex envelope $x(n)$ changes very slowly. The difference equation is

$$\Delta^r f(x) = \sum_{i=0}^{r} (-1)^i C_r^i f(x + r - i) \tag{5}$$

To deduce the formula and programming conveniently, the coefficients of the difference equation are expressed as d_v. This is a vector of elements

$$d_v(i) = (-1)^i C_{n_\mathrm{d}}^i \quad (i = 0, 1, 2 \ldots n_\mathrm{d}) \tag{6}$$

Thus, the difference equation can be described as follows:

$$d_v(0)x(n) + d_v(1)x(n-1) + \cdots + d_v(n_d)x(n-n_d) = \varepsilon(n) \tag{7}$$

where $i = 0, 1, \ldots, n_d$. The matrix representation is

$$\begin{bmatrix} d_v(0) & d_v(1) & d_v(2) & \ldots & d_v(n_d) & 0 & \ldots & 0 \\ 0 & d_v(0) & d_v(1) & \ldots & d_v(n_{d-1}) & d_v(n_d) & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\ 0 & 0 & 0 & d_v(0) & d_v(1) & d_v(2) & \ldots & d_v(n_d) \end{bmatrix} = \begin{bmatrix} \varepsilon(n_d) \\ \varepsilon(n_d+1) \\ \cdots \\ \varepsilon(N) \end{bmatrix} \tag{8}$$

which can be written as

$$\mathrm{Ax} = \varepsilon \tag{9}$$

The dimension of A is $(N - n_d + 1) \times N$.

The optimization objective is to minimize

$$J = r^2 \varepsilon^T \varepsilon + \eta^T \eta \tag{10}$$

where r is the weight factor. We compute

$$\frac{\partial J}{\partial x^H} = (r^2 A^T A + E)x - C^H y = 0 \tag{11}$$

which can be expressed as

$$(r^2 A^T A + E)x = C^H y \tag{12}$$

and so

$$x = (r^2 A^T A + E)^{-1} C^H y \tag{13}$$

The above formula gives the analytical solution of the VK filter for a single-order signal. For the purpose of convenience, define a new matrix

$$\mathrm{B} = (r^2 \mathrm{A}^T \mathrm{A} + \mathrm{E}) \tag{14}$$

When using regular data types, the limitations of the accuracy of double-precision floating data type mean that the identity matrix E will be submerged in addition operations if the weight factor r is too large. In the following sections, this issue will be discussed further and the relationship between r and the bandwidth of the filter in the steady state will be considered.

2.2 Frequency Response of VK Filter in Steady State

In this section, the basic principles of the VK filter are described from the perspective of the frequency domain, which contributes to a deeper understanding of the filter and provides a reference for setting reasonable weight coefficients in engineering practice. Before giving the exact calculation process, it should be emphasized that a larger weight coefficient always means a narrower bandwidth. Thus, larger weight coefficients are needed to achieve narrower bandwidths, even beyond the computational range of double-precision floating-point numbers. Firstly, by exploiting the structure of the analytical solution of the VK filter, the frequency response of the filter is obtained. The pseudocode for calculating the frequency response of the filter is then given.

The dimension of matrix A is $(N - n_d + 1) \times N$, and its elements can be represented as follows:

$$A(i_r, i_c) = \begin{cases} d_v(i_c - i_r) & (0 \le i_c - i_r \le n_d) \\ 0 & \text{other} \end{cases} \tag{15}$$

According to the matrix multiplication formula:

$$A^T A(i,j) = \sum_{k=1}^{N-n_d} A^T(i,k)A(k,j) \tag{16}$$

Thus, according to (15), assuming that $A^T A(i,j)$ is nonzero, the following relationship holds:

$$\begin{cases} 0 \le i - k \le n_d \\ 0 \le j - k \le n_d \end{cases} \tag{17}$$

This transforms to

$$\begin{cases} i - n_d \le k \le i \\ j - n_d \le k \le j \end{cases} \tag{18}$$

Let $S_u = \min(N - n_d, i, j)$ and $S_d = \max(i - n_d, j - n_d, 1)$. According to (15)–(18), the following relationships can be obtained:

$$A^T A(i,j) = \begin{cases} \sum_{k=S_d}^{S_u} A^T(i,k)A(k,j) & (S_u \ge S_d) \\ 0 & \text{other} \end{cases} \tag{19}$$

Further,

$$A^T A(i,j) = \begin{cases} \sum_{k=S_d}^{S_u} d_v(i-k)d_v(j-k) & (S_u \ge S_d) \\ 0 & \text{other} \end{cases} \tag{20}$$

If $A^T A(i,j)$ is nonzero, then $S_u \ge S_d$, that is, $|i - j| \le n_d$, which means each row of the matrix $A^T A$ has at most $2n_d + 1$ nonzero elements on the diagonal.

Let us exploit the structure of $A^T A$ and go a step further. As $A^T A$ is symmetric, the case $i \ge$ j is first considered. Assuming that $i \ge n_d + 1$ and $j \le N - n_d$, then

$$A^TA(i,j)=\begin{cases}\sum\limits_{k=i-n_\mathrm{d}}^{j} d_v(i-k)d_v(j-k) & \left(|i-j|\le n_\mathrm{d}\right)\\ 0 & \text{other}\end{cases} \tag{21}$$

In the same way, when $i<j$, assuming that $j\ge n_\mathrm{d}+1$ and $i\le N-n_\mathrm{d}$,

$$A^TA(i,j)=\begin{cases}\sum\limits_{k=j-n_\mathrm{d}}^{i} d_v(i-k)d_v(j-k) & \left(|i-j|\le n_\mathrm{d}\right)\\ 0 & \text{other}\end{cases} \tag{22}$$

From (21) and (22), it can be seen that the $2n_\mathrm{d}+1$ nonzero diagonal elements of each row of the matrix are the same, except for the first n_d rows and the last n_d rows of the matrix. Because A^TA is symmetric, $\mathrm{B}=(r^2\mathrm{A}^T\mathrm{A}+\mathrm{E})$ has the same structure as A^TA. Abbreviate the diagonal elements of each row and n_d elements after the diagonal element of matrix B as the vector b. The elements of b can be calculated as follows:

$$b_{i_b}=r^2A^TA\left(i,i+i_b\right)=r^2\sum_{k=0}^{n_\mathrm{d}-i_b} d_v(k)d_v\left(i_b+k\right) \qquad (i_b=1,2\ldots n_\mathrm{d}) \tag{23}$$

$$b_{i_b}=\left(r^2\sum_{k=0}^{n_\mathrm{d}} d_v(k)d_v(k)\right)+1(i_b=0) \tag{24}$$

As mentioned above, B has the same structure as A^TA, which means B is a sparse symmetric matrix with $2n_\mathrm{d}+1$ nonzero diagonal elements. According to the following relations:

$$\left(r^2A^TA+E\right)x=C^Hy \tag{25}$$

$$Bx=y' \tag{26}$$

we have that

$$y'(k)=\sum_{m=k-n_\mathrm{d}}^{k+n_\mathrm{d}} B(k,m)x(m) \tag{27}$$

After performing a Z transformation, the following formulas are obtained:

$$\begin{aligned}y'(z)=&\Big(b_{n_\mathrm{d}}Z^{-n_\mathrm{d}}+b_{n_\mathrm{d}-1}Z^{-(n_\mathrm{d}-1)}+\cdots+b_0\\ &+b_1Z^1+\ldots+b_{n_\mathrm{d}}Z^{n_\mathrm{d}}\Big)x(z)\end{aligned} \tag{28}$$

Substituting $z=\mathrm{e}^{j\omega}$ into the above, the frequency response of the filter is obtained as

$$\begin{aligned}y'\left(\mathrm{e}^{j\omega}\right)=&\Big(b_{n_\mathrm{d}}\mathrm{e}^{-n_\mathrm{d}j\omega}+b_{n_\mathrm{d}-1}\mathrm{e}^{-(n_\mathrm{d}-1)j\omega}+\cdots\\ &+b_0+b_1\mathrm{e}^{j\omega}+\cdots+b_{n_\mathrm{d}}\mathrm{e}^{n_\mathrm{d}j\omega}\Big)x\left(\mathrm{e}^{j\omega}\right)\end{aligned} \tag{29}$$

Through Euler's formula, the following mathematical relations are obtained:

$$\begin{aligned}\mathrm{e}^{-kj\omega}+\mathrm{e}^{kj\omega}=&\cos(k\omega)-j\sin(k\omega)\\ &+\cos(k\omega)+j\sin(k\omega)=2\cos(k\omega)\end{aligned} \tag{30}$$

Substituting (30) into (29), the final frequency response function of the filter is:

$$H\left(\mathrm{e}^{j\omega}\right)=\frac{x\left(\mathrm{e}^{j\omega}\right)}{y'\left(\mathrm{e}^{j\omega}\right)}=\frac{1}{b_0+2\sum\limits_{k=1}^{n_d} b_k\cos(k\omega)} \tag{31}$$

The pseudocode for calculating the frequency response of the VK filter in the steady state is given in Algorithm 1.

Algorithm 1 Frequency Response Of VK-Filter

Input: n_d diffrence Order
Output: d_v diffrence coefficient Vector

```
function CALCDIFFERENCEVECTOR(n_d)
    for i_v = 0 → n_d do
        dem ← 1
        num ← 1
        for index = 1 → i_v do
            dem ← dem × index
            num ← num × (d_v − index + 1)
        end for
        d_v[n_d − i_v] ← (−1)^{i_v} num/dem
    end for
    return d_v
end function
function CALCBDIGNVECTOR(n_d, d_v, r)
    for i_b = 0 → n_d do
        temp ← 0
        for k = 0 → n_d − i_b do
            temp ← temp + r^2 × d_v(k) × d_v(i_b + k)
        end for
        b[i_b] ← temp + 1
    end for
    return b
end function
```

In the following deduction process, both A^TA and B play an important role, but because both matrices are $N\times N$ dimensional, the number of elements in these matrices increases sharply with the signal dimension N, resulting in a dramatic increase in memory requirements. Thus, the elements of these two matrices are not all stored but are instead calculated when they are needed. The method of calculating A^TA and B is given in Algorithm 2.

Algorithm 2 A^TA & B Matrix Element Calculation

```
Input: n_d, d_v, N, i, j
Output: A^T A(i, j)
function CALCATAELEMENT(d_v, n_d, N, i, j)
    sumUpLimit ← min(N − n_d, i, j)
    sumDownLimit ← max(i − n_d, j − n_d, 1)
    if sumUpLimit < sumDownLimit then
        return 0
    else
        temp ← 0
        for k = s_d → s_u do
            temp ← temp + d_v(i − k)d_v(j − k)
        end for
        return temp
    end if
end function
function CALCBELEMENT(d_v, n_d, N, i_B, j_B)
    if i_B = j_B then
        return CALCATAELEMENT(d_v, n_d, i_B, j_B) + 1
    else
        return CALCATAELEMENT(d_v, n_d, i_B, j_B)
    end if
end function
```

2.3 Relationship Between the Filter Bandwidth and the Weighting Coefficient

On the basis of the frequency response function derived in the previous section, the relationship between the filter bandwidth and the weighting coefficient r is now discussed. Finally, an analytical solution for the weight coefficients is obtained for a certain bandwidth. For convenience, a new vector **ata** is introduced, which satisfies the following relations:

$$ata\left[i_b\right] = \sum_{k=0}^{n_d-i_b} d_v(k)d_v\left(i_b + k\right) \quad \left(i_b = 0, 1, 2 \ldots n_\mathrm{d}\right) \tag{32}$$

Substituting this into (23) and (24) yields

$$b_{i_b} = r^2 ata\left[i_b\right] \quad \left(i_b = 1, 2 \ldots n_\mathrm{d}\right) \tag{33}$$

$$b_0 = r^2 ata[0] + 1 \tag{34}$$

Substituting this into (31) gives

$$H\left(\mathrm{e}^{j\omega}\right) = \frac{1}{r^2 ata[0] + 1 + 2r^2 \sum_{k=1}^{n_\mathrm{d}} ata[k]\cos(k\omega)} \tag{35}$$

The cutoff frequency satisfies the following relationship:

$$r^2\left(ata[0] + 2\sum_{k=1}^{n_\mathrm{d}} ata[k]\cos\left(k\omega_\mathrm{c}\right)\right) + 1 = \sqrt{2} \tag{36}$$

By specifying the bandwidth of the filter, the weight coefficient can be calculated as

$$r = \sqrt{\frac{\sqrt{2} - 1}{ata[0] + 2\sum_{k=1}^{n_\mathrm{d}} ata[k]\cos\left(k\omega_\mathrm{c}\right)}} \tag{37}$$

The above formula describes the relationship between the weight coefficient and the bandwidth, and gives the physical meaning of the weight coefficient. That is, the bandwidth of the filter depends directly on the value of the weight coefficient. The larger the weight coefficient, the smaller the bandwidth of the filter. This relationship is established when the signal enters the steady state, but this does not mean that the VK filter is only suitable for steady-state systems; in fact, it is highly suitable for the unsteady state. When the bandwidth is known, the weight coefficients are computed as described in Algorithm 3.

Algorithm 3 Weight Coefficients Calculation

```
Input: n_d, d_v, ω_c
Output: r weighting coefficient
function CALCATAVECTOR(d_v, n_d)
    for i_b = 0 → n_d do
        temp ← 0
        for k = 0 → n_d − i_b do
            temp ← d_v(k)d_v(i_b + k)
        end for
        ata[i_b] ← temp
    end for
    return ata
end function
function CALCWEIGHTCOFFICIENTS(ata, n_d, ω_c)
    sumTemp ← 0
    for k = 1 → n_d do
        sumTemp ← ata[k]cos(kω_c)
    end for
    r = sqrt( (√2−1) / (ata[0]+2×sumTemp) )
end function
```

Once the order and bandwidth of the filter have been determined, the weight coefficient of the filter can be

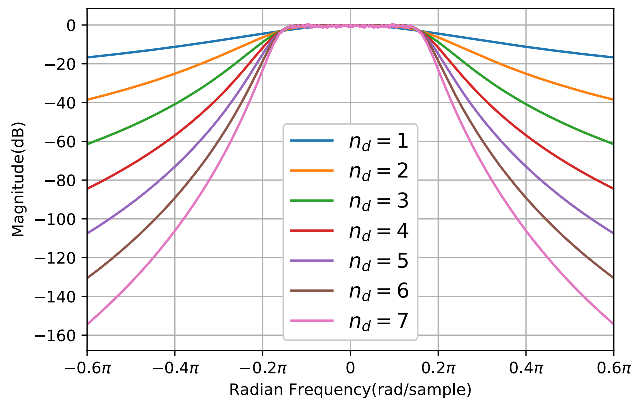

Fig. 1 Frequency response of VK filter under double-precision floating data type

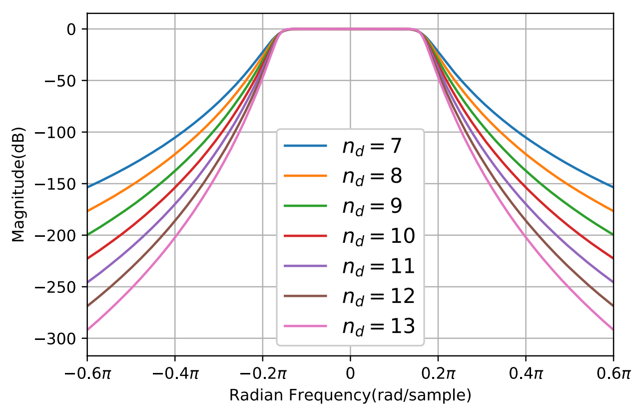

Fig. 2 Frequency response of VK filter under 50-decimal-place floating data type

obtained. The frequency response function of the filter is then given by (35). As shown in Fig. 1, the frequency response of the filter varies with the difference order. The higher the order, the better the flatness of the passband and the narrower the transition band. In other words, filters with high differential orders offer better frequency selection. Note that, when the difference order is 7, the response curve fluctuates at the passband. For difference orders of 8 or more, the filter cannot be designed at this bandwidth. Of course, this phenomenon is the result using double-precision floating-point numbers.

As shown Fig. 2, there is no such problem for an algorithm using arbitrary-precision floating-point numbers. After using a high-precision floating-point number, the filter passband response fluctuation at difference order 7 disappears, and filters with a differential order of 8 or more can be designed without fluctuation.

2.4 Maximum of Weight Coefficient

The results in the previous section indicate that higher-order filters produce flatter passband bandwidths and faster transition band changes. The higher order also means that the diagonal elements of A^TA are larger. A smaller bandwidth ensures better frequency selectivity and a greater weight factor r. Both these factors increase the value of the diagonal elements of r^2A^TA. If the diagonal elements of r^2A^TA are too large, the limitations of double-precision floating-point accuracy imply that, when the diagonal elements of r^2A^TA are added to 1, the value 1 is ignored, and the solution will fail. In this section, the minimum bandwidth, i.e., the maximum weight coefficient, is derived under different filter orders. Firstly, the double-precision data type is examined, and then arbitrary-precision numbers are explored.

Firstly, the method of measuring the precision of arbitrary-precision floating-point numbers is introduced. There are two terms involved, referred to as **prec** and **dps**. The term **prec** denotes the binary precision (measured in bits), whereas **dps** denotes the decimal precision. Binary and decimal precisions are approximately related through the formula **prec** = 3.33 × **dps**. For example, a precision of roughly 333 bits is required to hold an approximation of **dps**, that is, accurate to 100 decimal places (actually, slightly more than 333 bits are used). Double-precision floating-point numbers, on most systems, correspond to 53 bits of precision. For double-precision floating-point numbers, the maximum number that can be accurately represented is $2^{53} = 9{,}007{,}199{,}254{,}740{,}992$.

However, approaching this number should be avoided, because the missing decimal part will affect the accuracy of the result.

Assuming that a 10-bit binary number is reserved to ensure the accuracy of the calculation, the maximum value of the diagonal element of a matrix should be less than $2^{43} = 8{,}796{,}093{,}022{,}208$.

The maximum element value of A^TA is found on the diagonal of the matrix. More accurately, it will be the first element of the ata vector, that is, ata[0]. To ensure the accuracy of the calculation results, the following equation should be satisfied:

$$r^2 \mathrm{ata}[0] \le 2^{43} \tag{38}$$

The weight coefficients corresponding to different orders of difference are shown in Fig. 3.

The relation between the cutoff frequency and the weight coefficient of the filter is given by (35). Using this formula, the minimum bandwidth, corresponding to the maximum weight coefficient, can be calculated. However, from the point of view of the equation, solving the

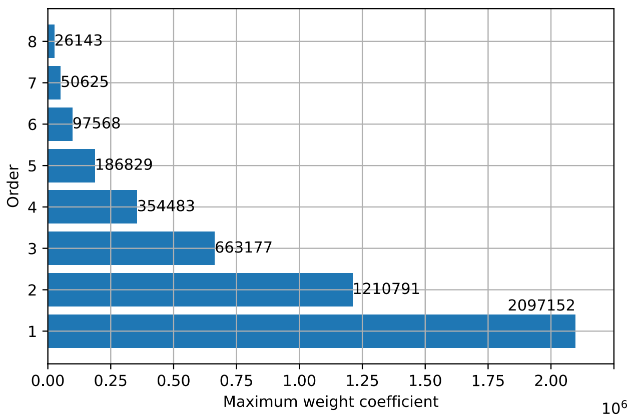

Fig. 3 Maximum weight coefficients under double-precision floating data type

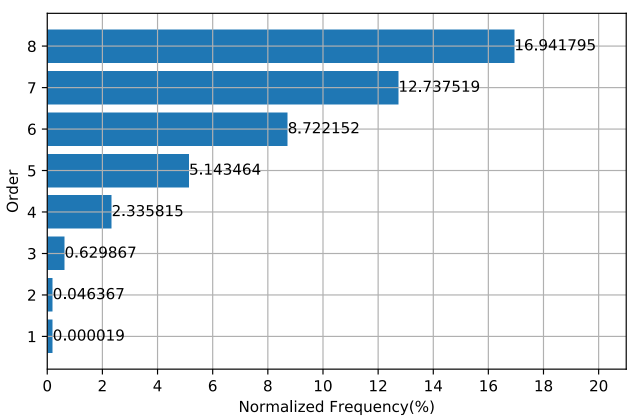

Fig. 4 Minimum bandwidth under **dps** = 53 floating data type

weight coefficient is fairly straightforward, assuming that the bandwidth, namely, the cutoff frequency, is known. However, the reverse process is quite complicated. Obviously, it is necessary to solve nonlinear equations. The analytical solution cannot be obtained but can only be realized through some numerical algorithm. At the same time, to realize an algorithm for an arbitrary difference order and avoid the inaccuracy of double-precision floating-point numbers, an algorithm for solving nonlinear equations based on arbitrary-precision numbers will be needed. For this purpose, intersection-based solvers such as 'anderson' or 'ridder' are recommended. Usually, they converge quickly and are very reliable. These solvers are especially suitable for cases where only one solution is available and the interval of the solution is known, which is the case for determining the cutoff frequency, assuming that the bandwidth is known. The minimum bandwidth under the **dps** = 53 floating data type is shown in Fig. 4.

The following describes an extension to arbitrary-precision floating-point numbers, based on which the maximum allowable weight coefficients under the corresponding accuracy can be calculated (or parameters with more practical physical significance, i.e., the minimum bandwidth). Similar to double-precision data, two parameters are required, dps, which again denotes the number of decimal places, and dps_{re}, which denotes the number of reserved decimal places needed to ensure the accuracy of the calculation. The maximum weight coefficient can be calculated by the following formula:

$$r^2 ata[0] \le 2^{dps-dps_{re}} \tag{39}$$

2.5 Numerical Implementation of VK Filter

This section describes how a numerical method can be used to solve the analytic solution of the filter. Although various numerical methods have been developed to solve the above equation, the relevant numerical algorithms should be discussed for two main reasons. The first, and most important, reason is that an arbitrary-precision floating-point arithmetic library is used to implement the filter. The second reason is that full use should be made of the structural characteristics of sparse matrices to accelerate the calculation. Thus, the problem is not whether the problem can be solved, but how to solve it efficiently and how to embed it into the arbitrary-precision floating-point arithmetic library.

The filter solution can be obtained by solving the linear equation $\mathrm{B}x = y'$ using Cholesky factorization. In this method, the matrix is decomposed into the product of a lower-triangular and an upper-triangular matrix, which can be expressed as $\mathrm{B} = LL^T$. The Cholesky–Banachiewicz and Cholesky–Crout algorithms can be expressed as follows:

$$L_{1,1} = \sqrt{B_{1,1}} \tag{40}$$

$$L_{i,1} = B_{i,1} / L_{1,1} \tag{41}$$

$$L_{j,j} = \frac{1}{L_{j,j}} \sqrt{B_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2} \tag{42}$$

$$L_{i,j} = \frac{1}{L_{j,j}} \sqrt{B_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k}} \qquad (i > j) \tag{43}$$

The lower-triangular matrix L has only $n_{\mathrm{d}} + 1$ nonzero diagonal elements (this is proved below).

$$B_{i,j} = 0 \quad \text{if} \quad |i-j| > n_\text{d} \tag{44}$$

For the first column of L,

$$L_{i,1} = \frac{B_{i,1}}{L_{1,1}} = 0 \quad \text{if} \quad i-1 > n_\text{d} \tag{45}$$

For the second column of L

$$L_{i,2} = \frac{1}{L_{1,1}}\left(B_{i,2} - L_{i,1}L_{2,k}\right) = 0 \quad \text{if} \quad i-2 > n_\text{d} \tag{46}$$

and so on. It can be inferred that

$$L_{i,m} = \frac{1}{L_{1,1}}\left(B_{i,2} - \sum_{k=1}^{m-1} L_{i,k}L_{m,k}\right) = 0 \quad i-m > n_\text{d} \tag{47}$$

From a rigorous point of view, mathematical induction can be used to prove that the above equation holds. Through the above method, it can be proved that L is a lower-triangular matrix with $n_\text{d}+1$ nonzero diagonal elements. This structural feature greatly reduces the computational complexity of Cholesky factorization. Equations (40)–(43) can be rewritten as follows:

$$L_{1,1} = \sqrt{B_{1,1}} \tag{48}$$

$$L_{i,1} = \begin{cases} \frac{B_{i,1}}{L_{1,1}} & (i \le n_\text{d}+1) \\ 0 & \text{else} \end{cases} \tag{49}$$

$$L_{j,j} = \frac{1}{L_{j,j}}\sqrt{B_{j,j} - \sum_{k=\max(1,j-n_\text{d})}^{j-1} L_{j,k}^2} \tag{50}$$

$$L_{i,j} = \begin{cases} \frac{1}{L_{j,j}}\sqrt{B_{i,j} - \sum_{k=\max(1,\max(1,j-n_\text{d}))}^{j-1} L_{i,k}L_{j,k}} & (0 < i-j \le n_\text{d}) \\ 0 & \text{other} \end{cases} \tag{51}$$

As L is diagonally sparse, the memory requirements can be reduced by storing only the nonzero elements of L. This matrix is called Ls. L is a $N \times N$ dimensional matrix, but Ls is $N \times n_\text{d}$ dimensional.

Algorithm 4 L_s Matrix Element Calculation

Input: n_d, d_v, N

Output: L_s (Ls is $N \times n_d + 1$ matrix)

```
function putLsparseMatix(L_m, L_n, eleValue)
/*Lm is row index of L,Ln is coloum index of L*/
/*eleValue is Value of L(Lm,Ln)*/
    L_s[L_m − L_n, L_n] ← eleValue
end function
function getLComp(L_m, Ln, eleValue)
    if L_m < L_n||L_n < L_m − n_d then
        return 0
    else
        return L_s[L_m − L_n, L_n − 1]
    end if
end function
function setLComp(L_m, Ln, eleValue)
    temp ← getBComp(1,1)
    L_s[0, 0] ← √temp
    for index = 1 → n_d + 1 do
        temp ← getBComp(index,1)/L_s[0, 0]
        putLMatrix(index,1,temp)
    end for
    for index = 2 → N do
        SU ← index − 1
        SD ← max(1, index − n_d)
        sum ← 0
        for sumIndex = SD → SU do
            temp ← getLComp(index,sumIndex)
            sum ← sum + temp^2
            temp ← getBComp(index,index) − sum
            putLMatrix(index, index, temp^0.5)
        end for
        tempIndex ← index + 1
        if 1 + index > min(n_d + index, N + 1) then
            pass
        else
            for tempIndex = 1 + index → min(n_d + index, N + 1) do
                SU_1 ← index − 1
                SD_1 ← max(tempIndex − n_d, 1)
                sum_1 ← 0
                if SU_1 < SD_1 then
                    sum_1 ← 0
                else
                    for sumInex_1 = SD_1 → SU_1 do
                        temp ← getLComp(tempIndex, sumIndex_1)
                        sum_1 ← sum_1 + temp^2
                        temp ← (−sum_1 + getBComp(tempIndex,index))/getLComp(index,index)
                        putLMatrix(tempIndex,index,temp)
                    end for
                end if
            end for
        end if
    end for
end function
```

After LU factorization, the solution of the VK filter can be obtained by forward reduction and backward substitution:

$$Bx = LUx = LL^T x = y' \tag{52}$$

Let

$$L^T x = z \tag{53}$$

Then,

$$Lz = y' \tag{54}$$

Equations (53) and (54) can be solved using a row-by-row method. Firstly, y' is obtained; this is equal to $C^H y$. The pseudocode for this process is given in Algorithm 5.

Algorithm 5 $y'\&\theta$ Calculation

```
function GETTHETAVECTOR(ω, fs)
/*ω[N] is angular velocity vector*/
    temp ← 0
    lastTemp ← 0
    for index = 0 → N − 1 do
        θ[index] ← temp
        temp ← lastTemp + ω[index]/fs
        lastTemp ← temp
    end for
    return θ
end function
function GETCOMPLEXY(θ, y)
    for index = 0 → N − 1 do
        y'[index] ← y[index]
        (cos(θ[index]) − jsin(θ[index]))
    end for
    return y'
end function
```

The process of solving (52) is forward reduction using the following equations:

$$z_1 = y'_1 / L_{1,1} \tag{55}$$

$$z_k = \frac{\left(y'_k - \sum_{i=\max(1,k-n_d)}^{k-1} L_{ki} z_i \right)}{L_{k,k}} \quad k = 2, 3, \ldots, N \tag{56}$$

The process of solving (53) is backward substitution using the following equations:

$$x_N = z_N / L_{NN} \tag{57}$$

$$x_k = \frac{z_k - \sum_{i=k}^{\min(N,n_d+k)} L_{i,k} x_i}{L_{k,k}} k = N-1, \ldots, 2, 1 \tag{58}$$

Ultimately, the filter solution is obtained. As mentioned above, the complex envelope x_k represents the signal amplitude and phase fluctuations. It does not represent the time-domain solution of the filter. In fact, the time-domain solution of the filter can be calculated as follows:

$$x_k = 2\mathrm{real}\left(\left(x_k \right) \exp\left(j\theta(k) \right) \right) \tag{59}$$

where real represents the real part of the complex number. Equations (55)–(59) can be represented by the pseudocode in Algorithm 6.

Algorithm 6 Cholesky Factorization

```
function CHOLESKYFACTORIZATION(y', N, n_d)
    z[0] ← y'[0]/GETLCOMP(1,1)
    for index = 2 → N do
        upIndex ← index − 1
        downIndex ← max(1, index − n_d)
        sum ← 0
        for sumIndex = downIndex → upIndex do
            sum ← sum + z[sumIndex − 1]×
            GETLCOMP(index,sumIndex)
        end for
    end for
    z[index − 1] = (y'[index − 1] − sum)/
    GETLCOMP(index,index)
    x[N − 1] ← Z[N − 1]/GETLCOMP(N,N)
    for index = N − 1 → 1 do
        upIndex ← min(N, index + n_d)
        downIndex ← index + 1
        sum ← 0
        for sumIndex = downIndex → upIndex do
            sum ← sumx[sumIndex − 1]×
            GETLCOMP(sumIndex,index)
        end for
        x[index − 1] = (z[index − 1] − sum)/
        GETLCOMP(index,index)
    end for
    return x
end function
function TIMEDOMAINSOLUTION(x, θ)
    for index = 1 → N do
        x[index − 1] ← 2 × x[index − 1]×
        (cos(θ[index − 1]) + jsin(θ[index − 1])
    end for
    return x
end function
```

3 Validation of VK Filter Algorithm

This section presents three test cases that verify the effectiveness of the proposed algorithm. In the first test case, the filter effectiveness is tested under different intensities of background white noise by adding white noise to a sine wave signal. The second test case is similar to the first, but another sine wave signal with a different frequency is added. In the

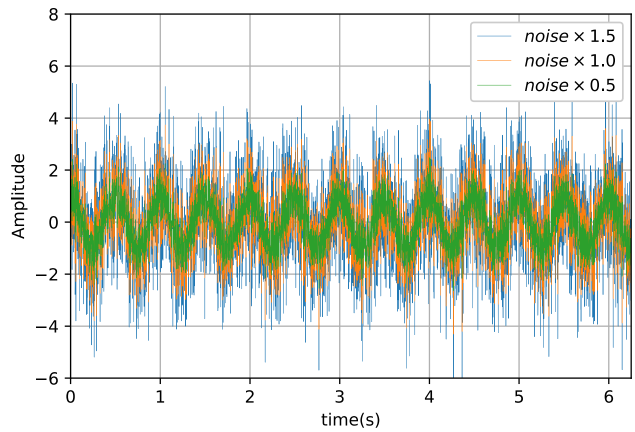

Fig. 5 Single sine waveform signal with white noise

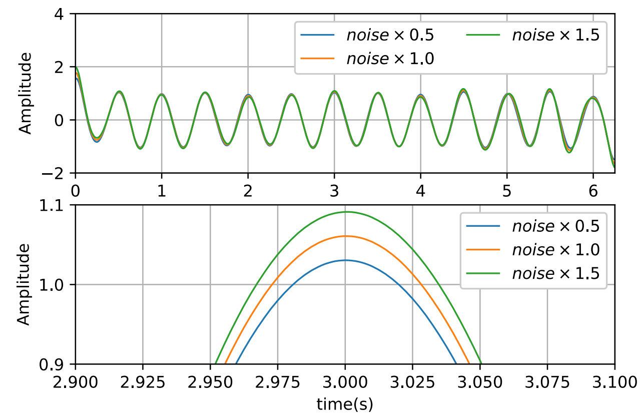

Fig. 6 Extracted single sine waveform with bandwidth of 2 Hz

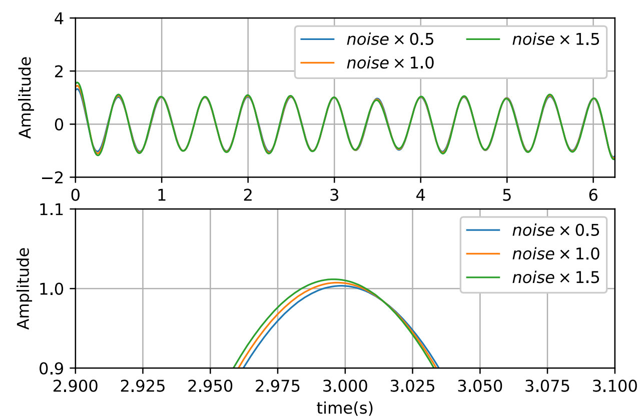

Fig. 7 Extracted single sine waveform with bandwidth of 0.8 Hz

second test case, the two sine wave signals with different frequencies are accurately extracted and the noise is isolated. In the third case, actual measurement data are used. This is a MATLAB example, and so the MATLAB algorithm is compared with the algorithm presented in this paper.

3.1 Extraction of a Signal from Background White Noise

In this section, the VK filter is used to extract a sine waveform signal from background white noise. The data for testing the algorithm are shown in Fig. 5. The added noise obeys a Gaussian probability distribution. Three sets of noise with different expectations are added to the sine wave signal with an amplitude of 1 and frequency of 2 Hz. The expectations of the noise signals are 0.5, 1, and 1.5, respectively. As can be seen from Fig. 5, the greater the expected white noise, the more violent the fluctuation is. Note that the sampling frequency of the signal is 800 Hz.

A 4th order VK filter with bandwidth of 2Hz is designed. Note that the arbitrary-precision arithmetic capability allows an arbitrary bandwidth and order to be allocated. The filtering effect is shown in Fig. 6. More noise is introduced when the bandwidth is 2 Hz, and the greater the noise, the greater the distortion of the result.

As a contrast, a 4th order VK filter with bandwidth of 0.8 Hz is designed. As shown in Fig. 7, although the greater noise results in greater distortion of the waveform from a microscopic point of view, from a macroscale point of view, the filtering results almost coincide with the sine wave signal. This shows that a narrower bandwidth can better isolate the influence of noise. Note that the design parameters of the filter are beyond the range of double-precision data. With the help of an arbitrary-precision arithmetic library, a narrowband signal can be extracted from the full signal with high sampling frequency. This is the advantage of arbitrary-precision floating-point arithmetic algorithms.

3.2 Extraction of a Multi-component Signal from Background White Noise

In this section, two sine waveform signals of different frequencies are extracted from background white noise. The frequencies of these two signals are 2 Hz and 4 Hz, respectively, and both have an amplitude of 1. Similarly, the added noise obeys a Gaussian probability distribution and has an expectation of 1.5. The signals with and without noise are shown in Fig. 8. The signal without noise is obtained by adding two sine wave signals. This signal, with added white noise, yields the signal with noise.

A 4th order VK filter with bandwidth of 1 Hz is designed. As shown in Fig. 9, the amplitude and frequency of the two extracted signals are basically 2 Hz and 4 Hz, respectively, the same as the original signal. The two sine wave signals can be extracted from the noise, and there is no interference between them. There is a slight fluctuation in the amplitude, and a smaller bandwidth could be set to suppress this fluctuation. However, without using the algorithm based on arbitrary-precision floating-point arithmetic, this cannot be achieved, which

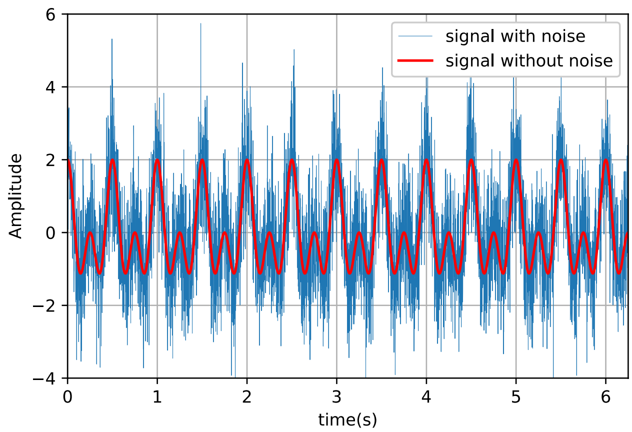

Fig. 8 Two added sine waveform signals with white noise

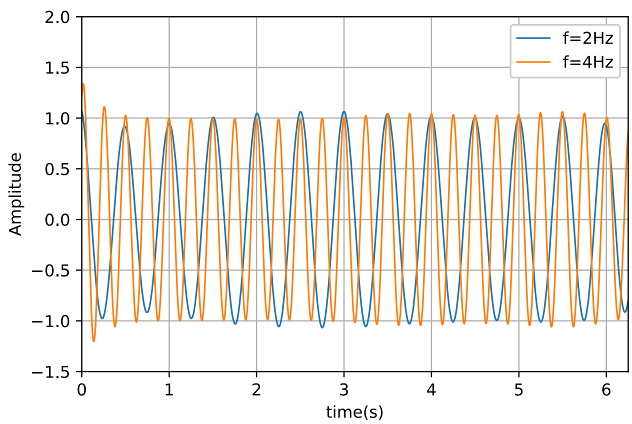

Fig. 9 Two sine waveform signals extracted from white noise

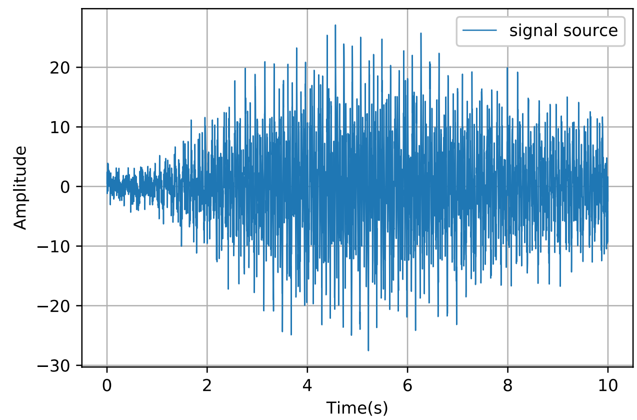

Fig. 10 Real helicopter vibration signal

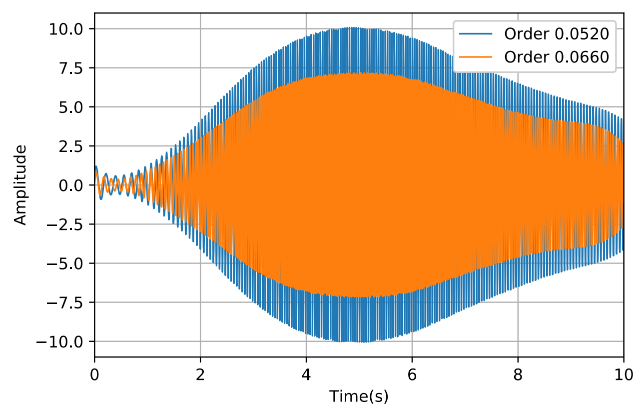

Fig. 11 Extracted orders of 0.0520 and 0.0660 from the helicopter vibration signal

demonstrates the advantages of the arbitrary-precision algorithm. Obviously, test cases with more than two signals could be considered, but this would make the figure appear very cluttered. Two signals are sufficient to verify the feasibility of the algorithm and are easier to understand and demonstrate.

3.3 Extraction from Real Measurement Signal

The data processed in this section are derived from actual measurement signals, namely, vibration data from an accelerometer in the cabin of a helicopter during a run-up and coast-down of the main motor. The data are taken from the MATLAB Signal Processing Toolbox.

A helicopter has several rotating components, including the engine, gearbox, and the main and tail rotors. Each component rotates at a known, fixed rate with respect to the main motor, and each may contribute some unwanted vibrations. The frequency of the dominant vibration components can be related to the rotational speed of the motor to investigate the source of high-amplitude vibrations. The helicopter in this example has four blades in both the main and the tail rotors. Important components of vibration from a helicopter rotor may be found at integer multiples of the rotational frequency of the rotor when the vibration is generated by the rotor blades. The signal in this test case is a time-dependent voltage, vib, sampled at a rate of 500 Hz. The data include the angular speed of the turbine engine, and a vector **t** of time instants. The ratios of rotor speed to engine speed for each rotor are stored in the variables main Rotor Engine Ratio and tail Rotor Engine Ratio and have values of 0.0520 and 0.0660, respectively. The signal is shown in Fig. 10.

A filter with a difference order of 4 and bandwidth of 1 Hz was designed. The orders to be extracted are 0.0520 and 0.0660, which have the two largest amplitudes of all the orders. The filtering result is shown in Fig. 11. In contrast, Fig. 12 shows the filtering result of the 3rd order filter using the MATLAB algorithm. From these figures, it can be seen that the envelope fluctuation is obviously more violent than that given by the algorithm proposed in this paper. Because the graphs are also drawn in MATLAB, they are slightly different from those given by the Python code.

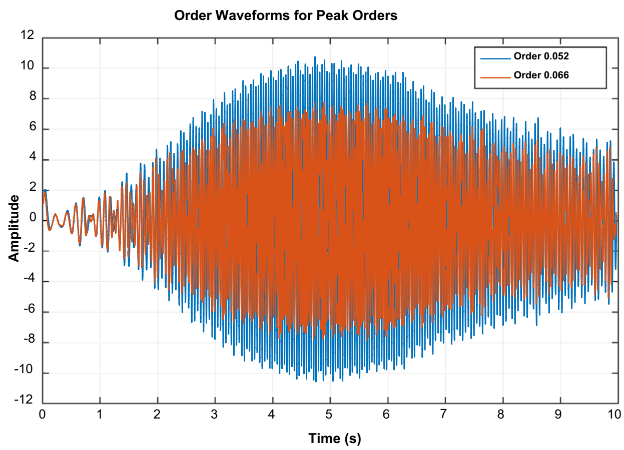

Fig. 12 Order signals extracted in MATLAB

4 Conclusions

This paper has presented the relevant theoretical and numerical implementations of a VK filter in detail. Using the pseudocode given in this paper, the VK filter algorithm based on arbitrary-precision floating-point numbers can be easily realized. The main body of this paper is Sect. 2, where the analytical solution of an arbitrary-order VK filter was given and the relationship between the filter bandwidth and the weighting coefficient r was obtained. The frequency response of various difference orders was also derived. In this process, the use of arbitrary-precision floating-point numbers successfully avoids the problem of high-order filter passband fluctuations. Finally, the proposed numerical method was used to determine the VK filter with reduced computational complexity by facilitating the use of arbitrary-precision algorithms.

Three test cases show that the proposed algorithm has better filtering effect, better frequency selectivity, and stronger anti-interference ability compared with double-precision data type algorithm. The main contribution of this paper is to overcome the problem whereby the bandwidth of the VK filter cannot be too narrow by using an arbitrary-precision floating-point arithmetic library. Based on this library, a filter with arbitrary bandwidth and arbitrary difference order can be implemented whenever necessary. From the practical application point of view, the numerical implementation of the algorithm is also given in detail, so that according to the ideas and methods of this paper, using Python to implement related algorithms is a brisk job.

Acknowledgements The paper is supported by the National Science Foundation for Young Scientists of China, Intelligent collaboration control of all-terrain vehicle via active attitude, and four-wheel steering control systems (Grant No. 51705185).

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Brandt, A., Lagö, T.L., Ahlin, K., et al.: Main principles and limitations of current order tracking methods. Sound Vib. **39**, 19–22 (2005)
2. Vold, H., Leuridan, J.: High resolution order tracking at extreme slew rates, using Kalman tracking filters. SAE Paper Number 931288 (1993)
3. Vold, H., Mains, M., Blough, J.: Theoretical foundations for high performance order tracking with the Vold–Kalman tracking filter. SAE Paper Number 972007 (1997)
4. Vold, H., Deel, J.: Vold–Kalman order tracking: new methods for vehicle sound quality and drive train NVH applications. SAE Paper Number 972033 (1997)
5. Vold, H., Mains, M., Corwin-Renner, D.: Multiple axle order tracking with the Vold–Kalman tracking filter. Sound Vib. Mag. **31**, 30–34 (1997)
6. Herlufsen, H., Gade, S., Konstantin-Hansen, H., et al.: Characteristics of the Vold–Kalman order tracking filter. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Process **6**, 3895–3898 (2000)
7. Pelant, P., Tuma, J., Benes, T.: Vold–Kalman order tracking filtration in car noise and vibration measurements. In: Proceedings of Internoise, Prague (2004)
8. Tuma, J.: Setting the passband width in the Vold–Kalman order tracking filter. In: 12th International Congress on Sound and Vibration, (ICSV12), Paper 719, Lisabon (2005)
9. Blough, J.R.: Understanding the Kalman/Vold–Kalman order tracking filters formulation and behavior. In: Proceedings of the SAE Noise and Vibration Conference, SAE paper No. 2007-01-2221 (2007)
10. Čala, M., Beneš, P.: Implementation of the Vold–Kalman order tracking filters for online analysis. In: 23rd International Congress on Sound and Vibration 2016 (ICSV 23) **1**, 367–374 (2016)
11. Vold, H., Miller, B., Reinbrecht, C., et al.: The Vold–Kalman order tracking filter implementation and application. In: 2017 International Operational Modal Analysis Conference
12. Amadou, A., Julien, R., Edgard, S., et al.: A new approach to tune the Vold–Kalman estimator for order tracking. Ciba-Geigy A-G, Switz (2016)
13. Feldbauer, C., Holdrich, R.: Realisation of a Vold–Kalman tracking filter—a least square problem. In: Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-000), Verona, Italy (2000)
14. Pan, M.C., Chu, W.C., Le, D.D.: Adaptive angular-velocity Vold–Kalman filter order tracking–theoretical basis, numerical implementation and parameter investigation. Mech. Syst. Signal. Process. **81**, 148–161 (2016)
15. Zhao, D., Li, J.Y., Cheng, W.D., et al.: Vold-Kalman generalized demodulation for multi-faults detection of gear and bearing under variable speeds. Procedia Manuf. **26**, 1213–1220 (2018)
16. Feng, Z.P., Zhu, W.Y., Zhang, D.: Time-frequency demodulation analysis via Vold-Kalman filter for wind turbine planetary gearbox fault diagnosis under nonstationary speeds. Mech. Syst. Signal. Process. **128**, 93–109 (2019)