Online ISSN 1848-3380, Print ISSN 0005-1144
ATKAFF 53(4), 382–397(2012)

Pieter V. Reyneke, Norman Morrison, Derrick G. Kourie, Corné de Ridder

Polynomial Filtering: To any degree on irregularly sampled data

DOI 10.7305/automatika.53-4.248
UDK 621.372.852.1:519.2/.6;621.396.969.3
IFAC 5.8.3; 1.1

Original scientific paper

Conventionally, polynomial filters are derived for evenly spaced points. Here, a derivation of polynomial filters for *irregularly* spaced points is provided and illustrated by example. The filter weights and variance reduction factors (VRFs) for both expanding memory polynomial (EMP) and fading-memory polynomial (FMP) filters are programmatically derived so that the expansion up to *any* degree can be generated. (*Matlab* was used for doing the symbolic weight derivations utilizing *Symbolic Toolbox* functions.) Order-switching and length-adaption are briefly considered. Outlier rejection and Cramer-Rao Lower Bound consistency are touched upon. In terms of performance, the VRF and its decay for the EMP filter is derived as a function of length (n) and the switch-over point is calculated where the VRFs of the EMP and FMP filters are equal. Empirical results verifying the derivation and implementation are reported.

Key words: Radar tracking filters, Polynomial approximation, Smoothing, State estimation, Satellite tracking, Discrete time filters, Laguerre processes, Legendre processes, Polynomial approximation, Smoothing methods, Interpolation, Extrapolation

Polinomno filtriranje: postizanje bilo kojeg stupnja na nepravilno uzorkovanim podacima. Polinomni filtri uobičajeno se rade za ravnomjerno raspoređene točke u prostoru. U ovom radu dana je derivacija polinomnih filtara za neravnomjerno raspoređene točke. Težinske vrijednosti filtra i faktori smanjenja varijance (VRF-ovi) za polinom proširene memorije (EMP) i polinom oslabljenje memorije (FMP) su programski podržani tako da se može napraviti ekspanzija do bilo kojeg stupnja. Kratko su razmotreni i promjena poretka i adaptacija dužine filtra. Dotaknute su i metode odbijanja jako raspršenih rezultata i Cramer-Raove konzistencije donje granice. VRF i njegovo opadanje za EMP filtar izvedeno je kao funkcija duljine (n) i izračunata je točka prijelaza gdje su VRF-ovi od EMP i FMP filtara jednaki. Predočeni su empirijski rezultati koji verificiraju izvod i implementaciju.

Ključne riječi: filtri za praćenje radara, polinomna aproksimacija, izglađivanje, procjena stanja, praćenje satelita, diskretni vremenski filtri, Laguerreovi procesi, Legendreovi procesi, aproksimacija polinomom, metode izglađivanja, interpolacija, ekstrapolacija

1 INTRODUCTION

The 1-step predictor and current-estimate filters introduced by Morrison in [1] and [2] both *recursively* calculate a least squares approximation of a process model, in effect determining the state of a polynomial curve fitted over noisy input observations. The origin of the fitted curve is on the last added observation's timestamp.

The expanding memory polynomial filter (EMP) adheres to the Cramér-Rao lower bound (CRLB[1]) on variance, thus it is CRLB consistent [2]. However, the fading memory polynomial filter (FMP) is not CRLB consistent.

In the present context, *recursive* refers to adding one observation at a time. To date these filters have been subject to the constraint that observations have to be evenly spaced. However, in natural environments, measurements cannot always be assigned to an integer type time batch without losing accuracy. Reasons therefore are threefold:

- Floating point time values more accurately position updates in time, leading to less ambiguity caused by natural variations (related to, for example, temperature, target clutter, occlusions, etc.).
- Detector or algorithm design may result in non-deterministic jumps in the update intervals, or in uneven even/odd time-interval symmetry. Examples where variable update intervals can be expected include: nodding algorithms, zig-zag sweep detectors, linear sensor integration and asynchronous mode

[1]The Cramér-Rao lower bound is a limit to the variance that can be attained by a linear unbiased estimator of a parameter of a distribution. See [3], [4] for fundamentals, [5] for a well-explained application and [1] for the proof of CRLB consistency, specifically for polynomial filters.

changing.

For the above mentioned reasons this research proposes a variable-step extension to the polynomial filters derived in [1] and [2], refining the proposed filters in [6].

The biggest advantage of recursive EMP and FMP filters, other than being extremely fast, is that their use of discrete orthogonal basis functions eliminates the need for matrix inversions in the auto-regression (AR) process.

We prefer to distinguish between *orthogonal* and *orthonormal* function sets, although *orthogonal* is the term generally attributed to a function set being both orthogonal and normalised. This opinion is becoming generally acceptable and is reflected by Chihara in [7].

NOMENCLATURE

In what follows, the following notation is used:

- t represents time in seconds;
- τ represents the constant expected update period (or system batch time) in seconds;
- δ represents delta-time in seconds;
- η (a real value) represents time in τ's;
- ζ (a real value) represents delta-time in τ's;
- Z the state on the last timestamp, the coefficients of a polynomial function; and
- Z^* the estimated state at a time other than the last timestamp.

Additionally, the term **order** is used to refer to a non-negative integer value attributed to a fit, a differential equation, a process model or a filter model; and the term **degree** is used to refer to a value attributed to a polynomial.

The remainder of this article is laid out as follows. Section 2 points to application areas for the proposed filtering technique. As background Section 3 provides a solution to the classical linear tracking differential equation and in Section 4 a motivation for a change in the state transition matrix is presented. An overview of the solutions for two discrete orthonormal (DON) polynomial function sets is given in Sections 5.1 and 5.2 respectively. Furthermore it is shown in Section 5.3 that matrix-inversion can be avoided. Section 5 provides the underpinnings of Section 6. An explanation of the extension of the current-estimate filter to a variable-step polynomial filter is given in Section 6.1. Thereafter recursive weight updates for the EMP and FMP filters are symbolically derived from the DON polynomials. Section 7 provides methods of auto-initialisation, combining, length-adaption and order switching for polynomial filters. Section 8 reports on results obtained from trial runs on simulated polynomial data thereby verifying the prediction capability of a variable-step implementation. The smoothing results on noisy, irregular, real data can be found in 8.2. Results obtained during missile testing are provided in 8.3. The article is concluded in Section 9.

To enhance the flow of the article, all relevant Matlab code has been included in APPENDIX A. The reader is referred to the code where applicable.

2 APPLICATION AREAS

Apart from the application areas dealing with irregularly updating sensors mentioned in Section 1, two other types of application areas can be distinguished: firstly, areas where polynomial filtering to a higher degree may be required; and secondly, areas where extremely fast execution speed are required.

Polynomial fit application areas where PFs to a higher degree were successfully demonstrated include:

(a.) *Plant Control:* when measurement noise in plant monitoring systems is low enough, higher degree fitting and predicting becomes feasible. New dynamic behavior can be observed in this manner.

(b.) In *inertial kinematics*, position derivatives are already defined up to the sixth order, as follows: velocity (1^{st}), acceleration (2^{nd}), jerk (3^{rd}), snap (4^{th}), crackle (5^{th}) and pop (6^{th}), terminology proposed by Gibbs and Gragert in 1996 [8]. These may require estimation and plotting in the near future.

(c.) In *structural dynamics*, e.g. earth quake modeling, higher degree fits may increase measurement and modeling accuracy.

(d.) Finally, in *tracking*, if higher degrees are observable in the underlying truth of a tracking scenario then, depending on system noise levels, one can consider increasing the memory timespan and doing a higher degree fit than four. This is especially true in 2D imaging systems where very high dynamic behavior is observable in the *end game* of such target intercept systems. Furthermore in 3D systems higher degree estimation enables the tracking of sinusoidal functions and hence circular manoeuvres more accurately.

Note that one should be careful not to use a higher degree filter than the natural degree present in the underlying truth of the controlled system, or points to be approximated.

The proposed variable-step filters have been applied in two areas where extremely fast execution speeds are required:

(a.) *Image integration:* An array of more than 10 000 polynomial filters was recently used to integrate and average an imaging sensor with a longish history length of 200 frames. Subtracting the averaged image from the current led to improved Signal-to-Noise ratio and suppressed unwanted artifacts. The successfully demonstrated filter array may be implemented in VHDL to lessen processing load maintaining the required real-time processing requirement.

(b.) *Constant false-alarm rate (CFAR) mean estimator* filters for LADAR ranging and range-gate applications require fast robust pulse detection for time-of-flight measurement and *pulse repetition interval* (PRI) tracking. Signals are commonly sampled at above 2 Gsamples/s. The CFAR commonly thresholds any deviation of more than 3σ from a mean. PFs can be used as a good mean-estimator for the segmenting, detection and tracking of wanted pulses.

3 MODELING A CLASSICAL DIFFERENTIAL EQUATION

To explain polynomial filters, consider modeling the classical 2^{nd}-order linear differential equation (DE), where it is assumed that the 3^{rd} derivative is 0.

$$\dot{r} = \frac{dr}{dt} \qquad \ddot{r} = \frac{d\dot{r}}{dt} \qquad \dddot{r} = \frac{d\ddot{r}}{dt} = 0 \tag{1}$$

If r is defined as $r = [r_1,\ r_2,\ r_3]^T = [r,\ \dot{r},\ \ddot{r}]^T$, the DE becomes:

$$\dot{r_1} = r_2 \qquad \dot{r_2} = r_3 \qquad \dot{r_3} = 0 \tag{2}$$

This may be written in matrix form as follows:

$$\dot{r} = Ar \tag{3}$$

where:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

Note that $r = Z$, which is the n^{th} order state vector.

$\Psi(t) = e^{At}$ is a solution for the linear DE system in Equation 3. The system has infinitely many solutions depending on the initial conditions. Note that $\Psi(\delta)$ can be used as a state transition matrix (STM). An STM is used to shift the validity instant of a DE system's state vector from time t by an amount δ, a real number, to time $t + \delta$ via a simple matrix multiplication (Equation 6).

Thus prediction can be done without recalculating the state (Z). By using the STM one can shift the state along time as follows: $Z^*(t+\delta) = \Psi(\delta)Z(t)$. Note, the asterisk indicates that the newly calculated state is an estimate at $t + \delta$. The asterisk is omitted at t since t is where the last fit was calculated.

One can either choose the *Padé expanded* STM, $\Psi(\delta)$, or the *commonly used* STM for polynomial filtering $\Phi(\delta)$, [1, 6]. We prefer using the *Padé expanded* STM for the derivation. (See Section 4.)

The linear DE in Equation 3 has the following discrete solution [9] :

$$z(t+\delta) = \Psi(\delta)z(t) \tag{5}$$

The state transition matrix $\Psi(\delta)^2$, for the 2^{nd}-order DE in Equation 3, is:

$$\Psi(\delta) = \begin{bmatrix} 1 & \delta & \frac{\delta^2}{2!} \\ 0 & 1 & \delta \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

and as will be seen, Equation 37, written out for the 2^{nd} order variable-step filter update, renders:

$$\begin{aligned} z_{2t} &= z^*_{2\,t,t-\delta} + 2!\Gamma_2 e_n \\ z_{1t} &= z^*_{1\,t,t-\delta} + \Gamma_1 e_n \\ z_{0t} &= z^*_{0\,t,t-\delta} + \Gamma_0 e_n \end{aligned}$$

where the Γ's for the 2^{nd}-degree for EMP and FMP for use with the Ψ are simply (see Sections 6.4 and 6.6):

$$\begin{aligned} \Gamma_2(n) &= \tfrac{(2!)30}{(n+3)^{(3)}} & \Gamma_2(\theta) &= \tfrac{2!(1-\theta)^3}{2} \\ \Gamma_1(n) &= \tfrac{18(2n+1)}{(n+3)^{(3)}} & \Gamma_1(\theta) &= \tfrac{3}{2}(1-\theta)^2(1+\theta) \\ \Gamma_0(n) &= \tfrac{3(3n^2+3n+2)}{(n+3)^{(3)}} & \Gamma_0(\theta) &= 1-\theta^3 \end{aligned}$$

Note that the EMP filter is self-initializing and can be switched with no degrading effects to the FMP filter at the instant when their variance reduction factors (VRF) are equal. (See Section 7.2.) Because of its fading memory, the FMP filter effectively possesses a fixed memory[3], and so it has the advantage of being able to follow sinusoids. It can therefore be used to approximate circular trends as well.

4 THE TWO STATE TRANSITION MATRICES FROM WHICH TO CHOOSE

An STM is used to shift a DE system's state vector from time t by an amount δ, a real number, to time $t + \delta$ via a simple multiplication. This is done without recalculating the state (Z). Thus, using the STM one can shift the state

[2]The original expanding memory polynomial (EMP) and fading memory polynomial (FMP) filters as derived in Chapters 9 and 13 of [2], can be adapted to utilise the STM in Equation 7. Γ_Ψ can be written in terms of $\Gamma(=\Gamma_\Phi)$ as follows $\Gamma_\Psi(j,i) = \Gamma_\Phi(j,i) \times (i-j)!$, see [6].

[3]The history in the FMP filter's case is discounted by the ratio θ per update or over time. For example the value 0.91, if done per update, represents an approximate memory length of around 30 samples. The formula $N = 2/(1-\theta)$ may be used to calculate an approximate effective memory length for the FMP filter for 0^{th} order, see Section 7.2 for higher orders.

as follows: $Z^*(t+\delta) = \Psi(\delta)Z(t)$. Note, the asterisk indicates that the newly calculated state is an estimate at $t+\delta$. The asterisk is omitted at t since t is where the last fit was calculated.

As the STM of the polynomial one can either choose the *Padé expanded* STM, $\Psi(\delta)$, or the *commonly used* STM for polynomial filtering $\Phi(\delta)$. (See [1].) Both can be used directly for prediction of states at any time, with or without denormalisation. For example in our case, as will be seen, $Z_n{}^*$ could be predicted from $Z_{\eta-\zeta}$ directly before updating the state, where $\eta = \frac{t}{\tau}$ with τ a real positive number, and $\zeta = \frac{\delta}{\tau}$.

Note that [9] compares many algorithms for computing the matrix exponential. (See the $expm$ function in [10].)

The *Padé expanded* STM is:

$$[\Psi(\delta)]_{i,j} = \begin{cases} \dfrac{\delta^{j-i}}{(j-i)!} & \{\forall\, i,j|\; 0 \le i \le j \le m\} \\ 0 & \text{elsewhere} \end{cases} \tag{7}$$

where $m = order + 1$ is the dimension of the square matrix Ψ.

This transition matrix (exactly the same as the one well-known from least squares), can be written out as follows:

$$\Psi(\delta) = \begin{bmatrix} 1 & \delta & \frac{\delta^2}{2!} & \cdots & \frac{\delta^{m-1}}{(m-1)!} & \frac{\delta^m}{m!} \\ 0 & 1 & \delta & \cdots & \frac{\delta^{m-2}}{(m-2)!} & \frac{\delta^{m-1}}{(m-1)!} \\ 0 & 0 & 1 & \cdots & \frac{\delta^{m-3}}{(m-3)!} & \frac{\delta^{m-2}}{(m-2)!} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \delta \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{8}$$

The *commonly used* STM for polynomial filtering, defined in [1], is represented as follows:

$$\Phi_{i,j}(\delta) = \begin{cases} \dbinom{j}{i}\delta^{(j-i)} & \{\forall\, i,j|\; 0 \le i \le j \le m\} \\ 0 \text{ elsewhere} \end{cases} \tag{9}$$

$$= \frac{j!}{(j-i)!i!}\delta^{(j-i)} \tag{10}$$

where $m = order + 1$ is the dimension of the square matrix Φ.

This original transition matrix, in Equation 10, can be written out as follows:

$$\Phi(\delta) = \begin{bmatrix} 1 & \delta & \delta^2 & \cdots & \frac{m-1!}{(m-2)!1!}\delta^{m-2} & \frac{m!}{(m-1)!1!}\delta^{m-1} \\ 0 & 1 & \delta & \cdots & \frac{m-1!}{(m-3)!2!}\delta^{m-3} & \frac{m!}{(m-2)!2!}\delta^{m-2} \\ 0 & 0 & 1 & \cdots & \frac{m-1!}{(m-4)!3!}\delta^{m-4} & \frac{m!}{(m-3)!3!}\delta^{m-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \delta \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{11}$$

The following discrete system results from the *commonly used* STM:

$$r(t+\delta) = \Phi(\delta))r(t) \quad \text{or in normalized form} \tag{12}$$
$$r(t+\delta) = D_\Phi(\tau)\Phi(\zeta)r(\eta) \tag{13}$$

where $D_\Phi(\tau) = diag(\begin{bmatrix} 1 & \frac{1!}{\tau} & \ldots & \frac{m!}{\tau^m} \end{bmatrix})$ is the diagonal denormalisation matrix for Φ, described in [1] and this matrix is slightly different from the $D_\Psi(\tau)$ denormalisation in Section 6.1, Equation 33.

We prefer to utilize and extend the first STM option, presented in Equation 7 and written out in Equation 8, because $\Psi(\delta)$ can easily be differentiated by simply using only one less row and column of $\Psi(\delta)$, e.g. in Matlab: `z_dot = Psi(1:N-1,1) * z'(2:N);`. The same holds for higher order derivatives. This characteristic is also applicable to the multi-dimensional case.

Altering the original EMP and FMP to use with Ψ:

The original EMP and FMP filters as derived in Chapters 9 and 13 of [2], can be adapted to utilise the STM in Equation 7.

Γ_Ψ, the weight for updating the Ψ STM based filter, is defined in terms of $\Gamma(= \Gamma_\Phi)$, the weight for updating the original Φ STM based filter, for the i^{th}-degree polynomial set for both the EMP and FMP filters, as follows:

$$\Gamma_\Psi(j,i) = \Gamma_\Phi(j,i) \times (i-j)!$$

where $j \in [0, i]$.

As an example, the change for the 4th degree Γ_Ψ is:

$$\begin{aligned} \Gamma_\Psi(4,4) &= \alpha_\Psi &&= \alpha \times 0! \\ \Gamma_\Psi(3,4) &= \beta_\Psi &&= \beta \times 1! \\ \Gamma_\Psi(2,4) &= \gamma_\Psi &&= \gamma \times 2! \\ \Gamma_\Psi(1,4) &= \delta_\Psi &&= \delta \times 3! \\ \Gamma_\Psi(0,4) &= \epsilon_\Psi &&= \epsilon \times 4! \end{aligned}$$

(See Sections 3, 6.4 and 6.6)

Furthermore, denormalisation of obtained normalised (i.e. $\tau \neq 1$) state-vectors is done by either pre-multiplying the state, by $D_\Phi(\tau)$ for Φ (see Equation 13); or by $D_\Psi(\tau)$ if using the Ψ STM choice (see Equation 36). See Section 6 for a description.

5 TWO DISCRETE ORTHONORMAL (DON) POLYNOMIAL SETS

Discrete orthogonal polynomials are encountered in discrete probability theory. The weights of the Chebyshev, Krawtchouk, and Charlier polynomials are, for example, related to the uniform, binomial and Poisson distributions, respectively. The discrete Legendre polynomials derived by Morrison [1], and [2] in 1969, and independently by Neumann and Schonbach [11] in 2000, are often used for numerical integration and interpolation.

The discrete Legendre polynomials only differ from the discrete Chebyshev polynomials by normalisation [12].

5.1 The discrete orthonormal Legendre Polynomial set

The differential equation below is referred to as the Legendre differential equation (DE) — named after the French mathematician *Adrien-Marie Legendre*:

$$\frac{d}{dx}\left[\left(1-x^2\right)\frac{d}{dx}p_j(x,n)\right]+n\,(n+1)\,p_j(x,n)=0 \quad (14)$$

Functions that are solutions to Legendre's differential equation are called Legendre functions.

An *orthonormal* basis function solution set to Equation 14 presenting an approximation $p(s)$ of the polynomial $f(s)$ of the form:

$$f(s)\approx(p(s))_n=(\beta_0)_n\varphi_0(s,n)+(\beta_1)_n\varphi_1(s,n)+ \quad (15)$$
$$\cdots+(\beta_m)_n\varphi_m(s,n) \quad (16)$$

The following properties hold:

- Any element can be written as:

$$\varphi_j(s,n)=\frac{p_j(s,n)}{c_j(n)} \quad (17)$$

- Any two elements are *orthogonal*, and orthogonality implies that if $i\neq j$ then:

$$\sum_{s=0}^{n}\varphi_i(s,n)\varphi_j(s,n)=\sum_{s=0}^{n}p_i(s)p_j(s)=0$$

- Any element is *normal*, and normality implies the sum over n if $i=j$ renders:

$$\sum_{s=0}^{n}\left(\varphi_j(s,n)\right)^2=\sum_{s=0}^{n}\left(\frac{p_j(s,n)}{c_j(n)}\right)^2=1$$

It therefore follows that for any $i,j\in[0,m]$, $\sum_{s=0}^{n}\varphi_i(s,n)\varphi_j(s,n)=\delta_{ij}$, where δ_{ij} denotes the Kronecker delta function.

Through the *backward summation* theorem and Gram-Schmidt orthogonalisation for discrete sets [13], we can obtain a solution for $p_j(s)$.

$$p_j(s)=\sum_{\nu=0}^{j}(-1)^{\nu}\binom{j}{\nu}\binom{j+\nu}{\nu}\frac{s^{(\nu)}}{n^{(\nu)}} \quad (18)$$

(See [2], Chapter 2 for a proof.)

Note, that this is in perfect analogy to Rodriguez's theorem for continuous Legendre polynomials. Normalisation is achieved by writing the solution series over all samples as a Newton series and then summing by parts.

A solution to the Legendre DE is presented by the following DON Legendre function set [14, 15]:

$$p_j(s,n)=\frac{\sum_{v=0}^{j}(-1)^{v}\binom{j}{v}\binom{j+v}{v}\frac{s^{(v)}}{n^{(v)}}}{\frac{(n+j+1)^{(j+1)}}{(2j+1)n^{(j)}}} \quad (19)$$

$$=\frac{P_{ij}(s,n)}{c_j^2(n)}\text{ in matrix form, including derivatives} \quad (20)$$

where P is the upper triangular matrix: $P_{ij}=\frac{1}{i!}\frac{d^jp}{ds^j}$. Note: $n^{(3)}$ is the three-term product $n(n-1)(n-2)$ and $\binom{j}{i}$ is defined as $\frac{j!}{(j-i)!i!}$. Furthermore, $p_j(s,n)$ is normalised by $c_j^2(n)$ in Equation 19 [1], Chapter 13, where:

$$c_j^2(n)=\frac{(n+j+1)^{(j+1)}}{(2j+1)n^{(j)}}$$

The symbolic expansion of $P_{ij}=\frac{1}{i!}\frac{d^jp}{ds^j}$ matrix is done by executing the `pmatrix` function. The function listing is provided in Subsection A.1. Results could be cross-validated up to the 4^{th} degree with a table provided in [1], Chapter 13, Appendix 13.3.

The solution polynomials are given as a function of s. In our case the DE is usually a function of time. Therefore the x used in the original DE and the s in the solution can both be substituted with t.

By using the discrete Legendre polynomial set EMP filters of various degrees are realized. The polynomials form an orthonormal basis set. Therefore matrix inversions are avoided. Section 3 presented a low order example of the variable-step update, whereas the higher order updates for EMP and FMP filters can be found in Section 6.

5.2 The discrete orthonormal Laguerre Polynomial set

The following is the Laguerre differential equation:

$$x\frac{d^2}{dx^2}p_j(x,\theta)+(\nu+1-x)\frac{d}{dx}+\lambda p_j(x,\theta)=0 \quad (21)$$

An *orthonormal* basis function solution set to Equation 21 presenting an approximation $p(s)$ of the polynomial $f(s)$ is of the form:

$$f(s)\approx(p(s))_\theta=\sum_{s=0}^{\infty}\left[\begin{array}{c}(\beta_0)_\theta\varphi_0(s,\theta)\theta^s+(\beta_1)_\theta\varphi_1(s,\theta)\theta^s\\+\cdots+(\beta_m)_\theta\varphi_m(s,\theta)\theta^s\end{array}\right] \quad (22)$$

where the following properties hold.

- Any element can be written as:

$$\varphi_j(s,\theta)=\frac{f_j(s,\theta)}{a_j(\theta)} \quad (23)$$

- Any two elements are *orthogonal*, and orthogonality implies that when $i\neq j$:

$$\sum_{s=0}^{\infty}\varphi_j(s,\theta)\varphi_i(s,\theta)\theta^s=\sum_{s=0}^{\infty}f_i(s,\theta)f_j(s,\theta)\theta^s=0$$

- Any element is *normal*, and normality implies that the sum over ∞ if $i=j$ renders:

$$\sum_{s=0}^{\infty}\left(\varphi_j(s,\theta)\right)^2\theta^s=\sum_{s=0}^{\infty}\left(\frac{f_j(s,\theta)}{a_j(\theta)}\right)^2\theta^s=1$$

Again we write for any $i, j \in [0, m]$ that $\sum_{s=0}^{\infty} \varphi_i(s,\theta)\varphi_j(s,n)\theta^s = \delta_{ij}$, where δ_{ij} denotes the Kronecker delta function. A solution function set is obtainable.

A set of DON Laguerre Polynomials (see [2] chapters 12 and 13), which satisfies a discretised version of the Laguerre DE is as follows:

$$f_j(s,\theta) = \frac{\theta^{2j} \sum_{v=0}^{j} (-1)^v \binom{j}{v} \left(\frac{1-\theta}{\theta}\right)^v s^{(v)} \binom{s}{v}}{\frac{\theta^j}{1-\theta}} \quad (24)$$

and in matrix form, including derivatives: $\frac{F_{ij}(s,\theta)}{c_j^2(\theta)}$ (25)

in which F is the upper triangular matrix: $F_{ij} = \frac{-1^i}{i!}\frac{d^j f}{ds^j}$.

Note that in Equation 24, $f_j(s,\theta)$ is normalised by $c_j^2(\theta)$ [1]:

$$c_j(\theta)^2 = \frac{\theta^j}{1-\theta}$$

Matlab code for the symbolic expansion of $F_{ij} = \frac{-1^i}{i!}\frac{d^j f}{ds^j}$ can be found in Subsection A.2. The results could be cross-validated up to the 4^{th} degree with a table provided in [1], Chapter 13, Appendix 13.6.

FMP filters of various degrees are realized by using the discrete Laguerre polynomial set in a way similar to discounted least squares where the discounting factor is θ.

5.3 Why inversion is not required when using orthonormal basis-functions

Similar to the case of least squares, we want to assign β_n to minimize the sum-of-squared residuals:

$$e_n = \sum_{k=0}^{n} (y_k - p_i(k))^2$$

Substituting Equation 16 gives:

$$e_n = \sum_{k=0}^{n} \left(y_k - \sum_{j=0}^{m} (\beta_j)_n \varphi_j(k,n) \right)^2$$

Observe that s, is replaced by k which consists of $n+1$ evenly spaced samples, hence the *discrete* property of the set.

We now differentiate with respect to β_n and set equal to zero:

$$\frac{\partial e_n}{\partial(\beta_i)_n} = \sum_{k=0}^{n} \left(y_k - \sum_{j=0}^{m} (\beta_j)_n \varphi_j(k,n) \right) \varphi_i(k,n) = 0 \qquad 0 \le i \le m$$

Making the y_k terms the object and reversing the summation:

$$\sum_{k=0}^{n} \varphi_i(k,n) y_k = \sum_{j=0}^{m} (\beta_j)_n \sum_{k=0}^{n} \varphi_j(k,n)\varphi_i(k,n) \quad 0 \le i \le m$$

This becomes:

$$\sum_{k=0}^{n} \varphi_i(k,n) y_k = \sum_{j=0}^{m} (\beta_j)_n \delta_{ij} \quad 0 \le i \le m$$

Or in matrix form:

$$[\Phi_{0\ldots m}]_{\{m+1,1\}}[Y_k]_{\{1,m+1\}} = I_{m+1}[\beta_{0\ldots m}]^T$$

Clearly no inversion is needed to obtain a solution for the $[\beta_{0\ldots m}]^T$ weight vector.

5.4 The Classical Least Squares Polynomial set

We first define the approximating function $p(s)$ over n points as a series — thus a linear combination of basis functions for degree m:

$$p(s)_n = (\beta_0)_n \frac{s^0}{0!} + (\beta_1)_n \frac{s^1}{1!} + \cdots + (\beta_m)_n \frac{s^m}{m!}$$

Note that we have chosen $s \in \mathbb{R}$ as arbitrary domain variable. Furthermore in the least squares case the n samples don't need to be evenly spaced.

The basis functions can be considered to be the set:

$$S = \left\{ \frac{s^0}{0!} \frac{s^1}{1!} \cdots \frac{s^m}{m!} \right\}$$

We have to determine the optimal assignment for the $\{\beta_n\}$ coefficient set for the n data-points, i.e. we have to find the optimal linear combination:

$$\{\beta_n\} = \{(\beta_0)_n (\beta_1)_n \ldots (\beta_m)_n\}$$

We may write the original combination in matrix form as follows:

$$S_n \beta_n = Y_n$$

where $Y_n = \begin{bmatrix} y_1 & y_2 & \ldots & y_n \end{bmatrix}^T$ is observation, the concatenation of n measurement values each at a respective domain point s_n.

There are two methods to solve for β_n. Firstly, by taking the partial derivative of the squared difference between the observation and approximation (the error) to β_n can be set equal to 0. Alternatively a pseudo-inverse of non-rectangular matrix S_n can be taken to determine β_n. These are mathematically equivalent.

The solution when taking the pseudo-inverse gives:

$$\begin{aligned} S_n \beta_n &= Y_n \\ S_n^T S_n \beta_n &= S_n^T Y_n \\ \beta_n &= (S_n^T S_n)^{-1} S_n^T Y_n \end{aligned}$$

It is clear that the answer involves finding an inverse of an $m+1$ by $m+1$ matrix, where m is the degree of the approximation.

6 POLYNOMIAL FILTERS

Recursive polynomial filters calculate either a least squares solution (**the expanding memory polynomial (EMP)**); or a weighted least squares solution (**the fading memory polynomial (FMP)**) with the weight ($\theta \in (0,1)$) fading the previous state estimate, i.e. $Z(t) = \theta Z(t-\delta) + (1-\theta)e_t$. For the EMP, $\Gamma(n)$ (see Section 6.4), based on orthonormal discrete Legendre polynomials, is used to update a least squares fit. In the case of FMP, $\Gamma_i(\theta)$, based on orthonormal discrete Laguerre polynomials, is used as the update weights to update a weighted least squares fit. The weight of the new datum is $(1-\theta)$. Hereby recursive autoregressive state updates are realized, as derived in [2].

Morrison [1], distinguishes between a current-estimate and a 1-step predictor. Equations 26 and 27 show the computation of the predicted state, Z^*_n and the error term e_n for both the current-estimate filter and the 1-step predictor filter.

$$Z^*{}_n = \Phi(1)Z_{n-1}, \quad \ldots \text{ (predict state } Z^*_n) \tag{26}$$
$$e_n = y_n - z^*_{0\,n}, \quad \ldots \text{ (calculate error term } e_n) \tag{27}$$

However, the formula for updating differs for the two respective cases, as shown in equations 28 and 30. Furthermore, in each case below, either $\Gamma_i(n)$ or $\Gamma_i(\theta)$ needs to be used to do the update for the EMP or FMP respectively (see Sections 6.4 and 6.6).

The current-estimate filter: Use either $\Gamma_i(n)$ or $\Gamma_i(\theta)$ and do the update:

$$Z_n = Z^*{}_{n,n-1} + \Gamma_i e_n \tag{28}$$
$$Z_n = \Phi(1)Z_{n-1} + \Gamma_i e_n \tag{29}$$

This step written out for the 2nd order current-estimator, gives:

$$\begin{aligned} z_{2n,n} &= z_{2n-1,n-1} + \Gamma_2 e_n \\ z_{1n,n} &= z_{1n-1,n-1} + 2z_{2n-1,n-1} + \Gamma_1 e_n \\ z_{0n,n} &= z_{0n-1,n-1} + z_{1n-1,n-1} + z_{2n-1,n-1} + \Gamma_0 e_n \end{aligned}$$

The 1-step predictor filter: Use either $\Gamma_i(n)$ or $\Gamma_i(\theta)$ and do the update:

$$\Phi(-1)Z_{n+1,n} = Z^*{}_n + \Gamma_i e_n \tag{30}$$

This last step, Equation 30, rewritten for the 2nd order, 1-step predictor update, gives the implementable form:

$$\begin{aligned} z_{2n+1,n} &= z_{2n,n-1} + \Gamma_2 e_n \\ z_{1n+1,n} &= z_{1n,n-1} + 2z_{2n+1,n} + \Gamma_1 e_n \\ z_{0n+1,n} &= z_{0n,n-1} + z_{1n+1,n} - z_{2n+1,n} + \Gamma_0 e_n \end{aligned}$$

6.1 Extending the Current-Estimate Polynomial Filter to a Variable-step Filter

In this section, the variable-step filter version is derived from the current-estimate filter. (See Chapter 12 of [1].) This filter is implemented in Matlab and verified on various scenarios. Two steps to switch from an integer interval to a real valued interval are:

- η is defined as the normalised time — a real number measured as a multiple of the expected update period (τ) from the start time of a track (t_0).

$$\eta = \frac{t - t_0}{\tau} \tag{31}$$

- Define $\zeta = \frac{\delta}{\tau}$ as the normalised delta-time, a real number.

$$\eta + \zeta = \frac{t + \delta - t_0}{\tau} \tag{32}$$

Note that $n = 0, 1, 2, \ldots$, the update- or batch number, as originally defined in [1], either stays unchanged or can be set equal to η. The second method was used during initial trails.

The correct time can be recovered by denormalising as follows:

$$t = \eta \times \tau + t_0 \tag{33}$$
$$t + \delta = (\eta + \zeta) \times \tau + t_0 \tag{34}$$
$$\delta = \zeta \times \tau \tag{35}$$

State denormalisation for the Ψ STM is done with $Z(t) = D_\Psi(\tau)Z(\eta)$ where:

$$D_\Psi(\tau) = diag(\begin{bmatrix} 1 & \frac{1}{\tau} & \ldots & \frac{1}{\tau^m} \end{bmatrix}) \tag{36}$$

When using the variable time-step EMP and FMP filter versions, certain normalisations should be performed in order to make them parameter identical, i.e. "hot-pluggable" with the standard 1-step predictor and current-estimate filter versions described in [1] and [2]. The two filters are normalised/scaled respectively as follows:

EMP See Equation 31.

FMP Filter parameter θ needs to be normalised to ensure that the amount of fading per update time (τ) is similar to the original FMP. This is done by either calculating the effective theta (θ_{eff}) as follows : $\theta_{eff} = \theta_0^{|\zeta|}$ or by simply updating with θ_0.

As previously discussed, in all cases normalised time $\eta(t)$ starts with 0 at the track start time (t_0) and is scaled (normalised) by the constant expected update period τ, i.e. $\eta(t) = \frac{t-t_0}{\delta}$. (See Equation 31.)

Note that when using normalised time the prediction/extrapolation/estimation formula up to batch time η becomes: $Z^*_\eta = \Psi(\zeta)Z_{\eta-\zeta}$ based on the previous fit done at the batch time $\eta - \zeta$ with $\eta, \zeta \in \mathbf{R}$.

6.2 The use of DON polynomials in variable-step polynomial filters

A definition for the variable-step filter update, applicable to both the EMP and FMP filters, is given in Equations

37 and 38 respectively:

$$t = t + \delta,\ \eta = \eta + \zeta \ldots (t,\ \eta \text{ updated})$$
$$n = \eta \text{ or, } n+1 \quad \ldots (n \text{ and optionally } \theta \text{ updated})$$
$$Z_t^* = \Psi(\delta)Z_{t-\delta} \ldots \text{(either predict by } \delta)$$
$$Z_\eta^* = \Psi(\zeta)Z_{\eta-\zeta} \ldots \text{(or predict by } \zeta)$$
$$e_n = y_n - z_{0\,t}^* \quad \ldots \text{(calculate the error at } t/\eta)$$

$$Z_t = Z_t^* + \Gamma_n e_n, \qquad \ldots \text{(do the update for EMP, or)} \quad (37)$$
$$Z_t = Z_t^* + \Gamma_\theta e_n, \qquad \ldots \text{(do the update for FMP.)} \quad (38)$$

In Equation 37, $\Gamma_n = [\Gamma(j,i)]$ for the EMP filter is obtained by simplifying $\Gamma_i(j) = p_j(i,i)$, with p_j defined in Equation 19. Similarly, $\Gamma_\theta = [\Gamma(j,i)]$ is obtained by simplifying $\Gamma_i(j) = f_j(i,\theta)$, with f_j defined in Equation 24.

If the above filter is normalised using a typical sampling interval τ, then the elements of the state estimate vector obtained from polynomial filters should be denormalised. This is done by pre-multiplying with a diagonal matrix of which the i^{th} element is $i!/\tau^i$ or $1/\tau^i$ depending on the STM choice. For details consult [1].

6.3 Making EMP filters recursive

In order to add one datum at a time the EMP update weights should further be extended.

Expanding the recursive polynomial EMP filter update fraction using the recursive discrete Legendre solution (, i.e. finding $\Gamma_n = P_{ij}(s,n)|_{s=n}$,) is an extremely laborious task. A Matlab function has been written to generate $\Gamma_n(j,i)$. It has been cross-validated with [1] up to the 4^{th} degree. The Matlab code for this function, rendering refined algebraic expressions for the EMP update weights up to any degree, can be found in Subsection A.3.

6.4 The resulting variable-step EMP filter

The update workflow for adding an observation, y_n, to the EMP filter is as follows:

$$\eta = \eta + \zeta,\ n = \eta \text{ or } n+1 \ \ldots (n \text{ and } \eta \text{ is updated})$$
$$Z_\eta^* = \Psi(\zeta)Z_{\eta-\zeta}, \qquad \ldots \text{(predict by } \zeta)$$
$$e_\eta = y_n - z_{0\,\eta}^*, \qquad \ldots \text{(calculate the error)}$$
$$Z_\eta = Z_\eta^* + \Gamma_n e_\eta, \qquad \ldots \text{(do the update)}$$

The update weights have been shown to be:

$$\Gamma_n(j,i) = P_{ij}|_{s=n}$$

In this subsection computational results for the EMP filter update weights generated, simplified, cross-validated up to the 4^{th} degree and refined in Matlab are provided. Note that calculating the $\Gamma_n(j,i)$ for higher degrees than 5, if needed, is just a matter of changing a constant in a `for`-loop.

$\Gamma(j,i)$ for EMP filters up to the 5^{th}-degree are as follows:

- 0^{th}**-degree**

$$z_{0\eta} = {z_{0\eta}}^* + \alpha_n e_\eta, \qquad \Gamma(0,0) = \alpha = \frac{1}{n+1}$$

- 1^{st}**-degree**

$$\begin{aligned} z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\ z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\ \Gamma(1,1) = \beta &= \frac{6}{(n+2)^{(2)}} \\ \Gamma(0,1) = \alpha &= \frac{2(2n+1)}{(n+2)^{(2)}} \end{aligned}$$

- 2^{nd}**-degree**

$$\begin{aligned} z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta \\ z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\ z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\ \Gamma(2,2) = \gamma &= \frac{30}{(n+3)^{(3)}} \\ \Gamma(1,2) = \beta &= \frac{18(2n+1)}{(n+3)^{(3)}} \\ \Gamma(0,1) = \alpha &= \frac{3(3n^2+3n+2)}{(n+3)^{(3)}} \end{aligned}$$

- 3^{rd}**-degree**

$$\begin{aligned} z_{3\eta} &= {z_{3\eta}}^* + (3!)\delta e_\eta \\ z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta \\ z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\ z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\ \Gamma(3,3) = \delta &= \frac{140}{(n+4)^{(4)}} \\ \Gamma(2,3) = \gamma &= \frac{120(2n+1)}{(n+4)^{(4)}} \\ \Gamma(1,3) = \beta &= \frac{20(6n^2+6n+5)}{(n+4)^{(4)}} \\ \Gamma(0,3) = \alpha &= \frac{8(2n^3+3n^2+7n+3)}{(n+4)^{(4)}} \end{aligned}$$

- 4^{th}**-degree**

$$\begin{aligned} z_{4\eta} &= {z_{4\eta}}^* + (4!)\epsilon e_\eta \\ z_{3\eta} &= {z_{3\eta}}^* + (3!)\delta e_\eta \\ z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta \end{aligned}$$

$$\begin{aligned}
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\
\Gamma(4,4) = \epsilon &= \frac{630}{(n+5)^{(5)}} \\
\Gamma(3,4) = \delta &= \frac{700(2n+1)}{(n+5)^{(5)}} \\
\Gamma(2,4) = \gamma &= \frac{1050(n^2+n+1)}{(n+5)^{(5)}} \\
\Gamma(1,4) = \beta &= \frac{25(12n^3+18n^2+46n+20)}{(n+5)^{(5)}} \\
\Gamma(0,4) = \alpha &= \frac{5(5n^4+10n^3+55n^2+50n+24)}{(n+5)^{(5)}}
\end{aligned}$$

- 5^{th}**-degree**

$$\begin{aligned}
z_{5\eta} &= {z_{5\eta}}^* + (5!)\zeta e_\eta \\
z_{4\eta} &= {z_{4\eta}}^* + (4!)\epsilon e_\eta \\
z_{3\eta} &= {z_{3\eta}}^* + (3!)\delta e_\eta \\
z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta \\
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\
\Gamma(5,5) = \zeta &= \frac{2772}{(n+6)^{(6)}} \\
\Gamma(4,5) = \epsilon &= \frac{3780(2n+1)}{(n+6)^{(6)}} \\
\Gamma(3,5) = \delta &= \frac{1260(6n^2+6n+7)}{(n+6)^{(6)}} \\
\Gamma(2,5) = \gamma &= \frac{420(2n+1)(4n^2+4n+15)}{(n+6)^{(6)}} \\
\Gamma(1,5) = \beta &= \frac{126(5n^4+10n^3+55n^2+50n+28)}{(n+6)^{(6)}} \\
\Gamma(0,5) = \alpha &= \frac{6(2n+1)(3n^4+6n^3+77n^2+74n+120)}{(n+6)^{(6)}}
\end{aligned}$$

6.5 Making FMP filters recursive

FMP update weights have been further extended in order to enable addition of one datum to an existing fit [2].

The expansion of the recursive polynomial FMP filter update fraction has been achieved by writing a Matlab function. Cross-validation could be carried out against previously published results [1] up to the 4th degree. The Matlab code for this function rendering simplified algebraic expressions for the FMP update weights, $\Gamma_\theta = F_{ij}(s,\theta)|_{s=1}$, up to any degree is provided in Subsection A.4.

6.6 The resulting variable-step FMP filter

The update workflow for adding an observation, y_n, to the FMP filter is as follows:

$$\begin{aligned}
\theta &= \theta_0^{|\zeta|} \text{ or } \theta_0 && \ldots (\theta \text{ optionally updated with normalised } \delta\text{-time}) \\
\eta &= \eta + \zeta && \ldots (\eta \text{ is updated}) \\
Z_\eta^* &= \Psi(\zeta) Z_{\eta-\zeta}, && \ldots (\text{predict by } \zeta) \\
e_\eta &= y_n - z_{0\,\eta}^*, && \ldots (\text{calculate the error}) \\
Z_\eta &= Z_\eta^* + \Gamma_\theta e_\eta, && \ldots (\text{do the update})
\end{aligned}$$

The update weights have been shown to be:

$$\Gamma_\theta(j,i) = F_{ij}|_{s=0}$$

In this subsection computational results of derivations for the FMP filter weights generated, simplified, cross-validated up to the 4^{th} degree and refined in Matlab are provided. Note, similarly to EMP filters, that higher degrees than 5, if needed, is just a matter of changing a constant in a `for`-loop.

$\Gamma(j,i)$ for FMP filters up to the 5^{th}-degree are as follows:

- 0^{th}**-degree**

$$z_{0\eta} = {z_{0\eta}}^* + \alpha e_\eta, \qquad \Gamma(0,0) = \alpha = 1-\theta$$

- 1^{st}**-degree**

$$\begin{aligned}
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\
\Gamma(1,1) = \beta &= (1-\theta)^2 \\
\Gamma(0,1) = \alpha &= 1-\theta^2
\end{aligned}$$

- 2^{nd}**-degree**

$$\begin{aligned}
z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta \\
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\
\Gamma(2,2) = \gamma &= \frac{1}{2}(1-\theta)^3 \\
\Gamma(1,2) = \beta &= \frac{3}{2}(1-\theta)^2(1+\theta) \\
\Gamma(0,2) = \alpha &= 1-\theta^3
\end{aligned}$$

- 3^{rd}**-degree**

$$\begin{aligned}
z_{3\eta} &= {z_{3\eta}}^* + (3!)\delta e_\eta \\
z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta \\
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta \\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta \\
\Gamma(3,3) = \delta &= \frac{1}{6}(1-\theta)^4 \\
\Gamma(2,3) = \gamma &= (1-\theta)^3(1+\theta) \\
\Gamma(1,3) = \beta &= \frac{1}{6}(1-\theta)^2(11+14\theta+11\theta^2) \\
\Gamma(0,3) = \alpha &= 1-\theta^4
\end{aligned}$$

- 4^{th}**-degree**

$$
\begin{aligned}
z_{4\eta} &= {z_{4\eta}}^* + (4!)\epsilon e_\eta\\
z_{3\eta} &= {z_{3\eta}}^* + (3!)\delta e_\eta\\
z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta\\
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta\\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta\\
\Gamma(4,4) &= \epsilon = \frac{1}{24}(1-\theta)^5\\
\Gamma(3,4) &= \delta = \frac{5}{12}(1-\theta)^4(1+\theta)\\
\Gamma(2,4) &= \gamma = \frac{5}{24}(1-\theta)^3(7+10\theta+7\theta^2)\\
\Gamma(1,4) &= \beta = \frac{5}{12}(1-\theta)^2(5+7\theta+7\theta^2+5\theta^3)\\
\Gamma(0,4) &= \alpha = 1-\theta^5
\end{aligned}
$$

- 5^{th}**-degree**

$$
\begin{aligned}
z_{5\eta} &= {z_{5\eta}}^* + (5!)\zeta e_\eta\\
z_{4\eta} &= {z_{4\eta}}^* + (4!)\epsilon e_\eta\\
z_{3\eta} &= {z_{3\eta}}^* + (3!)\delta e_\eta\\
z_{2\eta} &= {z_{2\eta}}^* + (2!)\gamma e_\eta\\
z_{1\eta} &= {z_{1\eta}}^* + \beta e_\eta\\
z_{0\eta} &= {z_{0\eta}}^* + \alpha e_\eta\\
\Gamma(5,5) &= \zeta = \frac{1}{120}(1-\theta)^6\\
\Gamma(4,5) &= \epsilon = \frac{1}{8}(1-\theta)^5(1+\theta)\\
\Gamma(3,5) &= \delta = \frac{1}{24}(1-\theta)^4(17+26\theta+17\theta^2)\\
\Gamma(2,5) &= \gamma = \frac{5}{8}(1-\theta)^3(1+\theta)(3+2\theta+3\theta^2)\\
\Gamma(1,5) &= \beta = \frac{1}{60}(1-\theta)^2\\
&\qquad (137+202\theta+222\theta^2+202\theta^3+137\theta^4)\\
\Gamma(0,5) &= \alpha = 1-\theta^6
\end{aligned}
$$

7 SWITCHING DEGREE, TYPE AND ADAPTING EFFECTIVE LENGTH

In this section pointers are provided for implementing a noise and degree sensing algorithm. Such an algorithm for switching degree and adapting the effective length is essential for achieving near-optimal fits in dynamically changing conditions. An example of such a scenario would be traversing a tight corner and thereafter traveling along a steady slope monotonically climbing. It is often preferable that such an algorithm should be completely automatic and should not need any tuning. Therefore a goodness-of-fit (GOF) measure is introduced as sensing mechanism to iteratively decide on length, degree and filter-type.

7.1 GOF based variable History-length

When using only the CRLB-consistent, self-initializing EMP filter, many methods can be devised to change the history-length on the fly. Methods include buffering filters in parallel and recycling the decided history through the filter upon any new datum's arrival (called a *windowed EMP*). One goodness-of-fit measure for length adjustment which empirically converges to $1-\frac{1}{\sqrt{N}}$ is:

$$
\begin{aligned}
E^2 &= \sum_{i=1}^{N}\left[y_n - \Psi(\zeta)Z_\eta\right]^2 R_\eta^{-1}\\
E^2 &= \left[Y_n - \Psi(\zeta)Z_\eta\right] R_\eta^{-1}\left[Y_n - \Psi(\zeta)Z_\eta\right]^T
\end{aligned}
$$

where R_η, the covariance matrix, is either known or may be estimated by means of the following variance estimator (on window-size N):

$$
R_\eta = diag\left(\sum_{i=2}^{N}\frac{\left[y_n - y_{n-1}\right]^2}{2N-2}\right) \tag{39}
$$

The Matlab code for implementing above RMS noise calculation is provided in the `stdd.m` function in Subsection A.5. An alternative method of achieving an *on-the-fly* variance without the above memory restriction is a recursive calculation due to Knuth (who cites Welford) [16, 17]. This algorithm is available in Python. Note that the Knuth calculation needs some tuning.

A third, well-known technique for finding a good indication of the immediate RMS noise present in a signal is to take the RMS of the residuals of a short fixed-length filter. E.g. a 10-sample windowed EMP (to the 1^{st} or 2^{nd} degree) will be suitable.

In this subsection we have briefly discussed an EMP variable-length extension. The focus of the next section is on combining the EMP and FMP filters.

7.2 EMP to FMP switching upon equal VRF cross-over

EMP and FMP filters can be used in combination and there can be seamless switching from the self-initializing EMP to an FMP of the same degree when their VRFs are equal. Note that this will always happen as the FMP has a constant VRF and the EMP's VRF shows a hyperbolic decay. In this section we determine the switch points between EMP and FMP filters for the various degrees using Matlab.

Matlab was used to generate the table for $N_s(degree)$. N_s is the length n at which the VRFs of the EMP and FMP is approximately equal.

Table 1 is the direct output of executing the `Ns_E2F` function provided in Subsection A.6.

The VRF derivation for the EMP and FMP diagonals is provided in the Listings in Subsection A.7 and the calculating of the crossing point was simply a matter of setting these results equal and making either n or θ the subject.

Table 1. N_s for switching from EMP to FMP

Degree	$N_s(\theta) = N_{effective}$	$\theta_s(n) = \theta_{effective}$
0_{th}	$2/(1-\theta)$	$1 - \frac{2}{n}$
1_{st}	$3.2/(1-\theta)$	$1 - \frac{3.2}{n}$
2_{nd}	$4.3636/(1-\theta)$	$1 - \frac{4.3636}{n}$
3_{rd}	$5.5054/(1-\theta)$	$1 - \frac{5.5054}{n}$
4_{th}	$6.6321/(1-\theta)$	$1 - \frac{6.6321}{n}$
5_{th}	$7.7478/(1-\theta)$	$1 - \frac{7.7478}{n}$

Note that two assumptions were made, firstly that n is big and, secondly that $\theta \approx 1$ leaving only the $(1-\theta)$ terms for each degree.

The provided VRF functions calculate the diagonals only.

The off-diagonal VRF matrix elements are calculated as follows:

$$S_{X*_{n+1,n}} = P_{n+1,n} C_n^2 P_{n+1,n}^T \quad \ldots \text{for the 1-step predictor EMP}$$

$$S_{X*_{n+1,n}} = F_{-1,\theta} A(\theta) F_{-1,\theta}^T \quad \ldots \text{for the 1-step predictor FMP}$$

$$S_{X*_{n,n}} = P_{n,n} C_n^2 P_{n,n}^T \quad \ldots \text{for the current estimator EMP}$$

$$S_{X*_{n,n}} = F_{0,\theta} A(\theta) F_{0,\theta}^T \quad \ldots \text{for the current estimator FMP}$$

The Matlab code for creating an example of these, rarely needed, full VRF matrices is provided in Subsection A.8.

7.3 Quick settling — degree switching

Lower order fits converge faster therefore the quick settling technique described next is relevant [1]. It is recommended to start with a 0^{th} degree self-initializing EMP filter. When the VRF drops acceptably then switch to 1^{st} degree, etc. The switch-over points depend on the update rate and noise. These should be empirically refined after a VRF inspection. Its is recommended that the next higher order filter is executed in tandem to ensure a stable higher degree coefficient before switching over to it.

7.4 Outlier rejection

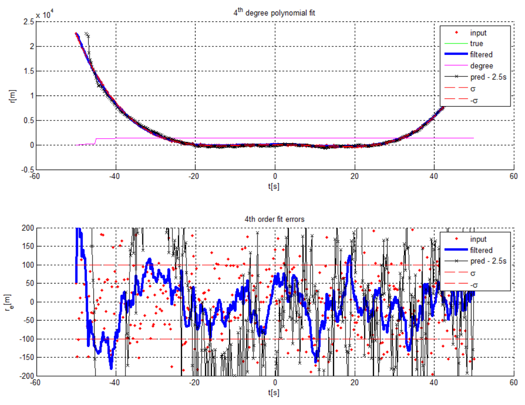
As noted in [1], the outlier rejection criteria for 2^{nd} degree fits, and for higher degrees, can be approximated by the $3\sigma_v$ criteria, i.e. on whether $|y_n - z^*| < 3\sigma_v$. Note that the estimation of z^* is done by using the prediction formula $z^*(t+\delta) = \Psi(\delta)z(t)$. We found that $|y_n - z^*| < \sigma_v \left(1 + \frac{2}{\sqrt{N}}\right)$ is well suited for an outlier detection threshold, where N is the current memory-length. Furthermore given that Nyquist holds, $\sigma_v = \sqrt{R_\eta}$ gives a good noise estimate where R_η is defined in Equation 39.

8 TESTING

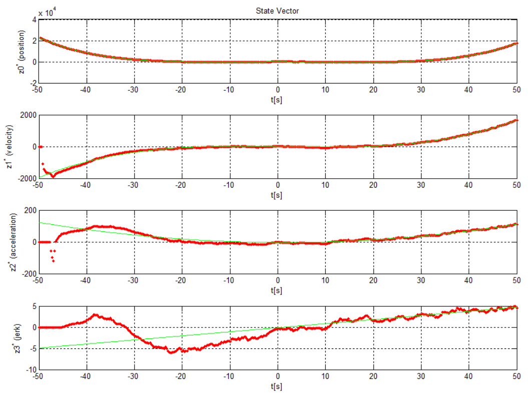
The Matlab code for an implementation of the EMP and FMP filters is extremely simple and available from the corresponding author or from *www.c-develop.co.za*.
Testing was done on the following scenarios.

8.1 The polynomial filter in a noisy environment

Here the original and improvised versions of the polynomial filters are compared using known simulated polynomials of different orders. The simulation parameters were $P_D = 0.5$, in the presence of $\sigma = 100.0$ additive noise, for an update period of $\tau = 0.25s$. The resulting graphs when approximating a 4^{th}-degree polynomial with a 4^{th}-order filter model are shown in Fig. 1. The filter parameter θ was set to 0.95.



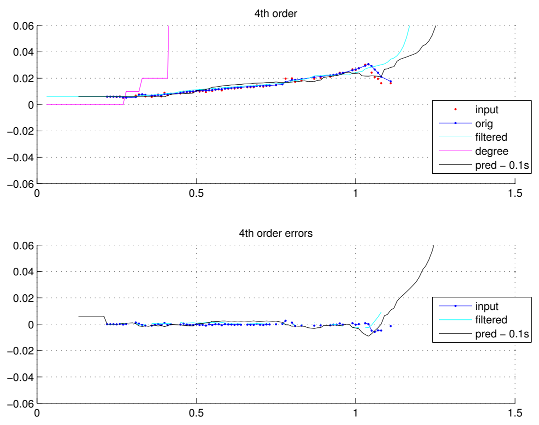
(a) Position and position-errors



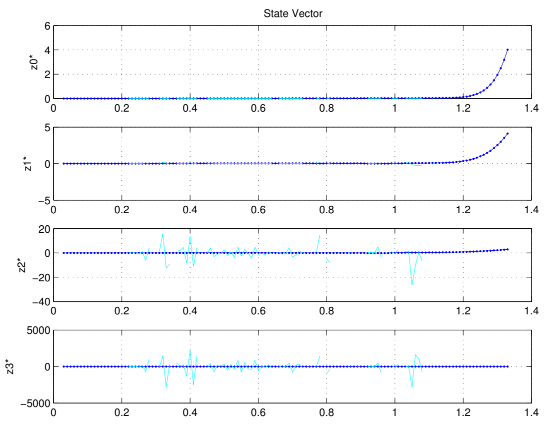
(b) 4^{th}-degree state vector

Fig. 1. 4^{th}-degree polynomial function

Note, when choosing a constant update rate the variable-step and original polynomial filters render near identical state vector estimates. Thus when the derived variable-step filter was compared to both the original filters the results were $\Sigma \left(Z_{vstep} - Z_{original}\right)^2 < 10^{-10}$ over the total track-time of $100s$ ($\tau = 1s$). During the comparison to the 1-step predictor a "one batch" prediction was necessary on the variable-step filter, i.e $Z^*_{\eta+1} = \Psi(1)Z_\eta$, before the mentioned sum-of-square subtraction could be done.

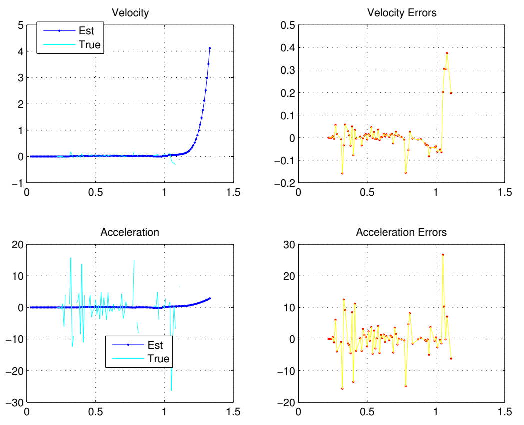


(a) Position and position-errors

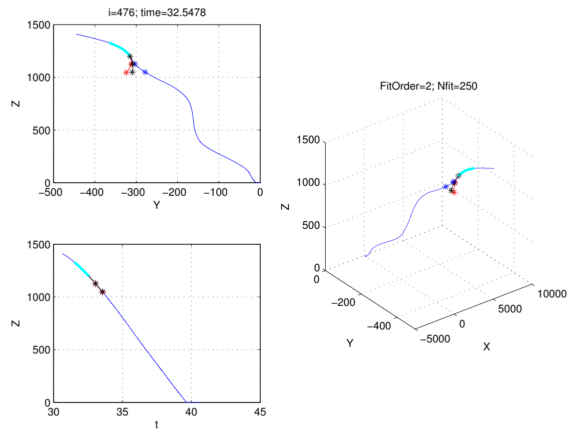


(b) 3^{rd}-degree state vector

Fig. 2. 3^{rd}-degree polynomial filter on a very noisy image feature, detected area



(a) Velocity, acceleration and their errors



(b) 3^{rd}-degree polynomial filter (in black) compared to the LS(100)(red)

Fig. 3. Max 3^{rd}-degree polynomial compared to Least-Squares with history of 100 in a 3D Missile on Missile encounter

8.2 Results from smoothing very noisy flight data that verify smoothing and prediction capability

Fig. 2 displays the smoothing achieved on “detected area in pixels”, an irregularly detected feature, extracted from noisy flight data. The data-set parameters were as follows: $P_D \approx 0.25$, in the presence of non-linearly changing additive noise σ as a function of range, and, for an irregular update period of around $\tau = 0.01s$.

8.3 Results from smoothing a flight profile in 3-D

Fig. 3 presents results from smoothing a flight profile in 3-D. This verifies the smoothing and prediction capability in a Missile on Missile encounter.

9 CONCLUSION

This article extends the fading- and expanding memory polynomial filters devised in [1, 2].

In addition to all the characteristics of the originally devised polynomial filters, the modified polynomial filters have the following improved capabilities:

- The time-interval of the modified (variable-step) polynomial filters do not have to be evenly spaced.
- Missed detections need not be cycled (updated) in the modified polynomial filters.
- Denormalisation is only required if the modified polynomial filter should be hot-pluggable with the original. (See Section 6.1.)

- The state transition matrix differs slightly from the polynomial filters originally devised in [2]. This difference is proposed to simplify predictions and derivatives of predictions. (See Section 4.) The i^{th} order derivation can be calculated with the following sub-matrix formula: $\frac{d^i Z(t)}{dt^i} = \Psi_{1:N-i,1} \times Z(t)_{1+i:N}$

Recursive polynomial filters are simple, elegant and fast. It is becoming feasible to implement banks of these filters in firmware with *type* (either EMP or FMP), θ and *order* as the only tunable parameters. These parameters can all be changed on the fly. Since weights as well as the VRFs for these filters can now be derived up to *any* degree, future research initiatives include the determination of an algorithm that automatically adjusts the filter parameters and does order sensing in high dynamic scenarios.

APPENDIX A MATLAB CODE

The presented MATLAB code has been optimized with respect to the clarity of the procedure being implemented. The main automated derivations of this article can be verified by running a main m-file without parameters.

A.1 Generate solution for : $P_{ij} = \frac{1}{i!}\frac{d^j p}{ds^j}$

The `pmatrix` function (can be executed without input parameters):

```
function [ dipjs ] = pmatrix( i, j, n, s, ss )
% Generates the i^th degree (col) j^th element (row)
% for the legendre solution matrix. Note, n and s
% passed for uniqueness.

display_pretty = 0;
if (nargin < 5)
    ss = n+1; % for 1step, ss=n; for current estimator
end

if (nargin < 1)
    syms s n;
    i = 4;
    j = 4;
    ss = n;
    display_pretty = 1;
end

pij = legendre_polynomial(j, n, s);
if ((i>0) &&(~isa(pij,'double')))
    dipj = 1/factorial(i)*diff(pij,'s',i);
else if (i>0)
        dipj = 0;
    else
        dipj = 1/factorial(i)*pij;
    end
end
dipjs = subs(dipj,s,ss,0);


if (display_pretty)
    display(['']);
    display(['------------------------------']);
    display(['P Matrix (' num2str(i) ', ' ...
                          num2str(j) ' )']);
    display(['------------------------------']);
    if(~isa(dipjs,'double'))
        pretty(simple(dipjs));
    else
        display(num2str(dipjs));
    end
end

end
```

Normalized with the `legendre_norm`, $c_j(n)^2$:

```
function [cjn2] = legendre_norm (j, n, s)

    if (nargin<1)
        syms n s;
        j=4;
    end
    cjn2 = (n+j+1)/(2*j+1)*M(n+j,j)/M(n,j);
end

function [p] = M(z,v)
    ii=0;
    p =1;
    for(i=1:v)
        p=p*(z-ii);
        ii = ii+1;
    end;
end
```

A.2 Generate symbolic expressions for: $F_{ij} = \frac{-1^i}{i!}\frac{d^j f}{ds^j}$

The `fmatrix` function (can be executed without input parameters):

```
function [ dipjs ] = fmatrix( i, j, The, s , ss )
% Generates the i^th degree (col) j^th element (row)
% for the leguerre solution matrix. Note, The and s
% passed for uniqueness.

display_pretty = 0;
if (nargin < 5)
    ss = -1; % for 1step, ss=0; for current estimator
end

if (nargin < 1)
    syms The s;
    i = 4;
    j = 4;
    display_pretty = 1;
end

pij = ((-1)^i)*laguerre_polynomial(j, The, s);
if ((i>0) &&(~isa(pij,'double')))
    dipj = diff(pij,'s',i);
else if (i>0)
        dipj = 0;
    else
        dipj = pij;
    end
end
dipjs = dipj/factorial(i);
dipjs = subs(dipjs,{s},{ss},0);


if (display_pretty)
    display(['']);
    display(['------------------------------']);
    display(['F Matrix (' num2str(i) ', ' ...
                          num2str(j) ' )']);
    display(['------------------------------']);
    if(~isa(dipjs,'double'))
        pretty(simple(dipjs));
    else
        display(num2str(dipjs));
    end
end

end
```

Normalized with the `laguerre_norm`, $c_j(\theta)^2$:

```
function [cjn2] = laguerre_norm(j, The, s)

    if (nargin<1)
        syms The s;
        j=4;
    end

    cjn2 = The^j/(1-The);
end
```

A.3 Filter weights for EMP filters up to any degree

The m-code is provided for the cross-validated `gamma_EMP_polynomial` function with only one user definable input n, the filter degree. This function renders algebraically refined expressions for EMP update weights up to any degree.

The code can be executed without input parameters:

```
function all_legendre()

    % Run in Matlab to generate EMP filter
    % weights up to the 5th degree
    clear all;
    clc;
    syms n s;
    for (j=0:5)
        display([' ']);
        display(['---------------------------------------']);
        display(['Expanding Memory filter for degree : ' ...
                        num2str(j )]);
        display(['---------------------------------------']);
        i = j;
        pj = gamma_EMP_polynomial(i,j,n,s);
        if(~isa(pj,'double'))
            pretty(simple(pj));
        else
            display(num2str(pj));
        end
        for (i=j-1:-1:0)
            pj = gamma_EMP_polynomial(i,j,n,s);
            if(~isa(pj,'double'))
                pretty(simple(pj));
            else
                display(num2str(pj));
            end
        end
    end

end

function [ Gamma ] = gamma_EMP_polynomial( i, j, n, s )

    dipj = 0;
    for (jj = j:-1:0)
        pij = ((-1)^jj)*legendre_polynomial(jj, n, s);
        if (i>0)
            if (~isa(pij,'double'))
                dipj = dipj+diff(pij,'s',i)/...
                         legendre_norm(jj, n, s);
            end
        else
            dipj = dipj+pij/legendre_norm(jj, n, s);
        end
    end

    dipj = dipj/factorial(i);
    Gamma = subs(dipj,s,n,0);

end
```

A.4 Filter weights for FMP filters up to any degree

The m-code is provided for the cross-validated `gamma_FMP_polynomial` function with only one user definable input n, the filter degree. The function renders simplified expressions for FMP update weights up to any degree.
The code can be executed without input parameters:

```
function all_laguerre()

    % Run in Matlab to generate FMP filter
    % weights up to the 5th degree
    clear all;
    clc;
    syms The s;
    for (j=0:5)
        display([' ']);
        display(['---------------------------------------']);
        display(['Fading Memory filter for degree : '...
                        num2str(j )]);
        display(['---------------------------------------']);
        i = j;
        pj = gamma_FMP_polynomial(i,j,The,s);
        if(~isa(pj,'double'))
            pretty(simple(pj));
        else
            display(num2str(pj));
        end
        for (i=j-1:-1:0)
            pj = gamma_FMP_polynomial(i,j,The,s);
            if(~isa(pj,'double'))
                pretty(simple(pj));
            else
                display(num2str(pj));
            end
        end
    end

end
```

```
...
function [ Gamma ] = gamma_FMP_polynomial( i, j, ...
                                           The, s )
    dipj = 0;
    for (jj = j:-1:0)
        pij = ((-1)^i)*(laguerre_polynomial(jj, The, s))...
            *The^(1*jj);
        if (i>0)
            if (~isa(pij,'double'))
                dipj = dipj+diff(pij,'s',i)/...
                         laguerre_norm(jj, The, s);
            end
        else
            dipj = dipj+pij/laguerre_norm(jj, The, s);
        end
    end
    dipj = dipj/factorial(i);
    Gamma = subs(dipj,s,0);

end
```

A.5 $\sqrt{R_\eta}$, an *on-the-fly* standard deviation

The `stdd.m` function, an *on-the-fly* difference based RMS noise algorithm with no requirement for prior knowledge about the underlying truth:

```
function [sout] = stdd(data_in, dim)

if (nargin<1)
    data_in = randn(1000,1)+[1:1000];
    dim = 1;
end

sz = size(data_in);
if (max(sz)<=1)
    sout = 0;
    return;
end

if nargin==1,
  % Determine which dimension STDD will use
  dim = min(find(size(data_in)~=1));
  if isempty(dim), dim = 1; end

  sout = std(diff(data_in))./sqrt(2);
else
  sout = std(diff(data_in,dim))./sqrt(2);
end
```

And a windowed version, called `stddw.m`. This function calculates the moving window (`winsize=20`) of root-mean-square (RMS) noise values and returns a signal of the original length:

```
function [sout] = stddw(s_in, winsize, dim)

if (nargin<2)
    winsize = 20;
end

ida = 1:length(s_in);
for (ii = ida)
    idx =   (ida>=ii-floor(winsize/2))&...
            (ida<ii-floor(winsize/2)+winsize);
    if (nargin<3)
        sout(ii) = stdd(s_in(idx));
    else
        sout(ii) = stdd(s_in(idx),dim);
    end
end
```

A.6 The variance reduction factor diagonals for the EMP and FMP filters

The `Ns_E2F.m` function calculates the optimal switch point between EMP and FMP filters for a given degree. Note that the function calls the VRF functions provided in Subsection A.7 in order to obtain the relevant denormalised symbolic VRF expressions.

The `Ns_E2F.m` function can be executed without input parameters:

```
function Ns_E2F( )

    syms The s tau n;

    for (j = 0:5)
        i = 0;
        p_Evrf = simple(vrf_EMP_polynomial(i,j,n,s,tau));
        p_Fvrf = simple(vrf_FMP_polynomial(i,j,The,s,tau));

        % assumption one
        p_Evrf = p_Evrf*n;
        p_Evrf = limit(p_Evrf,n,Inf)/n;

        % assumption two
        p_Fvrf = expand((p_Fvrf)/(1-The));
        p_Fvrf = simple(p_Fvrf);
        p_Fvrf = subs(p_Fvrf,'The',1)*(1-The);

        Ns = eval(solve(p_Evrf-p_Fvrf/(1-The),'n'));

        display( [' Ns(' num2str(j) ') = ' ...
                  num2str(Ns) '/(1-The)'] );
    end
end
```

A.7 The variance reduction factor (VRF) diagonals for EMP and FMP filters

The function for achieving expressions for the denormalised VRF matrix diagonal for EMP filters.

```
function vrf_diag_emp()

clear all; close all
clc;
syms n s tau;

for (j=0:5)
    display([' ']);
    display(['---------------------------------------']);
    display(['Expanding Memory VRF for degree : ' num2str(j)]);
    display(['---------------------------------------']);
    pjnumi0 = pseries(n+1,j+1,n);
    pjnumij = pseries(n+j+1,j*2+1,n);
    for (i=j:-1:0)
        pj = vrf_EMP_polynomial(i,j,n,s,tau);

        if(~isa(pj,'double'))
            if (i>0)
                pretty(simple(pj*pjnumij)/pjnumij);
            else
                pretty(simple(pj*pjnumi0)/pjnumi0);
            end
        else
            display(num2str(pj));
        end
    end
end

end

function [p] = pseries(in,pn,n)
    pi = in; p = pi; pn = pn-1;
    while (pn>0)
       p = p*(pi-1);
       pi = pi-1;
       pn = pn-1;
    end
end

function [ Sm ] = vrf_EMP_polynomial( i, j, n, s, tau )

    ss = n+1; % for 1step, ss=n; for current estimator
    Sm = pmatrix(i, 0, n, s, ss)^2/legendre_norm(0, n, s);
    for (k=1:j)
       Sm = Sm + pmatrix(i, k, n, s, ss)^2/...
            legendre_norm(k, n, s);
    end
    Sm = (factorial(i)/tau^i)^2*Sm;

end
```

The function for achieving expressions for the denormalised VRF matrix diagonals for FMP filters:

```
function vrf_diag_fmp()

clear all;
clc;
syms The s tau;
```

```
...
for (j=0:5)
    display([' ']);
    display(['---------------------------------------']);
    display(['Fading Memory VRF for degree : ' num2str(j )]);
    display(['---------------------------------------']);
    for (i=j:-1:0)
        pj = vrf_FMP_polynomial(i,j,The,s,tau);
        if(~isa(pj,'double'))
            pretty(simple(pj));
        else
            display(num2str(pj));
        end
    end
end
end

function [ Sm ] = vrf_FMP_polynomial( i, j, The, s, tau )

    ss = 1; % for 1step, ss=0; for current estimator
    % calc row i, start at column j=i and stop at j=j (k)
    Sm = 0;
    for (k=i:j)
      Am = 0;
      F_1stepik = fmatrix(i,k,The,s,ss);
      % calc columm j, start at row j=i, stop at j=j (o)
      for (o=i:j)
          F_1stepio = fmatrix(i,o,The,s,ss);
          Am = Am + F_1stepik*A(k, o, The, s)*...
               F_1stepio;
      end
      Sm = Sm + Am;
    end
    Sm = (factorial(i)/tau^i)^2*Sm;
end

function [ aij ] = A(i, j, The, s)
    aij = factorial(i+j)/factorial(j)/factorial(i)*(1-The)...
        /(1+The)^(i+j+1);
end
```

A.8 Generating the full VRF matrix (including the off- diagonals)

The m-code that generates a complete VRF matrix for the 1-step EMP filter is shown. (Here, the result is still in normalised form.)

```
syms n s;
tau = 1;

%% All vrf elements
P = [
     pmatrix( 0, 0, n, s, n+1 ) pmatrix( 0, 1, n, s, n+1 )
     pmatrix( 1, 0, n, s, n+1 ) pmatrix( 1, 1, n, s, n+1 )
];

C2 = [
        1/legendre_norm( 0, n, s ) 0
        0 1/legendre_norm( 1, n, s )
];

Svrf = P*C2*transpose(P);
pretty(simple(Svrf));

%% Vrf Diagonals only
S_diag = [
    vrf_EMP_polynomial( 0, 1, n, s, tau )
    vrf_EMP_polynomial( 1, 1, n, s, tau )
];

eval(S_diag(1)-Svrf(1,1))
eval(S_diag(2)-Svrf(2,2))
```

REFERENCES

[1] N. Morrison, *Tracking Filter Engineering: The Gauss-Newton and Polynomial Filters*. IET, 2012.

[2] N. Morrison, *Introduction to Sequential Smoothing and Prediction*. McGraw-Hill, New York, 1969.

[3] H. Cramér, *Mathematical Methods of Statistics*. Princeton Univ. Press, 1st ed., 1946.

[4] C. R. Rao, "Information and the accuracy attainable in the estimation of statistical parameters," *Bulletin of Calcutta Mathematical Society*, vol. 37, 1945.

[5] Z. Long, N. Ruixin, and P. Varshney, "A sensor selection approach for target tracking in sensor networks with quantized measurements," *ICASSP 2008*, March 2008.

[6] P. V. Reyneke, N. Morrison, D. Kourie, and C. de Ridder, "Smoothing Irregular data using Polynomial Filters," *Proceedings Elmar-2010*, pp. 393–397, 2010.

[7] T. Chihara, *An introduction to Orthogonal Polynomials*. Gordon and Breach Science Publishers, New York-London-Paris, 1978.

[8] P. Gibbs and S. Gragert, "What is the term used for the third derivative of position?," tech. rep., Usenet Physics FAQ, November 1998.

[9] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 2nd ed., 1989.

[10] Mathworks, "Matrix exponential function — `expm`." http://www.mathworks.com/access/helpdesk/help/techdoc /ref/expm.html, 2009.

[11] C. P. Neuman and D. I. Schonbach, "Discrete (Legendre) orthogonal polynomials - a survey," *International Journal for Numerical Methods in Engineering*, vol. 8, pp. 743–770, Jun 2005.

[12] A. den Brinker and M. Bastiaans, "Modern Signal Transformations," tech. rep., Technishe Universiteit Eindhoven, April 2000.

[13] D. Pollock, *A Handbook of Time-Series Analysis, Signal Processing and Dynamics*, pp. 227–291. No. v. 1 in Signal Processing and its Applications, Academic, 1999.

[14] M. F. Aburdene and J. E. Dorband, "On the Computation of Discrete Legendre Polynomial Coefficients," *Multidimentionsl Systems and Signal Processing*, no. 4, pp. 181–186, 1993.

[15] E. Brookner, *Tracking and Kalman Filtering made Easy*. Wiley & Sons, 1st ed., 1998.

[16] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, p. 232. Addison-Wesley, Boston, 3 ed., 1998.

[17] B. P. Welford, "A note on a Method for Calculating Corrected Sums of Squares and Products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

Pieter V. Reyneke a masters student at the Department of Computer Science, University of Pretoria (UP), completed a BSc in Electronic Engineering at UP in 1991. He is currently practicing as a Algorithm Engineer at Denel Dynamics, SA.

Norman Morrison lectures and mentors at the University of Cape Town, and enjoys sharing his experiences in short courses, workshops and seminars, all over. He worked at various institutions internationally including AT&T. He has worked on discrete filtering techniques including Polynomial Filtering. His most recent book entitled: "Tracking Filter Engineering" will be published by the IET in 2012. Norman is currently involved in the development of radar tracking algorithms for TWS- and pulse-doppler Radar Systems. This entails a combination of Gauss-Aitken and iterative non-linear Gauss-Newton state-estimation filters.

Derrick G. Kourie a professor in the Computer Science Department, University of Pretoria, completed a PhD in operations research in 1975 at Lancaster, UK.

Corné de Ridder is a PhD student at the University of Pretoria (UP), South Africa. Her research interests include the development of pattern matching algorithms to solve computationally difficult biological problems. She is a lecturer at the School of Computing, University of South Africa (UNISA).

AUTHORS' ADDRESSES

P. V. Reyneke
Fastar Research Group
Dept. Radar and Imaging Systems,
Denel Dynamics, Nellmapius Rd,
Irene, 0157, Centurion, SA
email: pieter.reyneke@deneldynamics.co.za

Norman Morrison, Ph.D.
Department of Electrical Eng.,
University of Cape Town,
7700, Cape Town, SA
email: norman.morrison@uct.ac.za

Prof Derrick G. Kourie, Ph.D.
Dept. of Computer Science,
University of Pretoria,
Roper Str, 0002, Pretoria, SA
email: dkourie@cs.up.ac.za

Corné de Ridder, MSc.
School of Computing,
University of South Africa,
Preller Str 1, 0001, Pretoria, SA
email: driddc@unisa.ac.za

Received: 2012-04-10
Accepted: 2012-09-08