Authors

Theo Alexopoulos, Gianluigi de Geronimo , George Iakovidis, Venetios Polychronakos

The VMM Shaper

The VMM "semi-Gaussian" shaper responds to an event with an analog pulse, the peak amplitude of which is proportional to the event charge. The time needed to return to baseline after the peak, depends on the the time constants and the configuration of poles. The VMM facilitates a 3$^{\text{rd}}$ order c-shaper with the combination of one real and two conjugate poles. The transfer function $T(s)$ for such shaper is given by the following expression:

$$T(s) = \frac{1}{(s+p_1)\prod_{i=2}^{(n+1)/2}\left[(s+r_i)^2+c_i^2\right]} = \frac{1}{(s+p_1)\left[(s+r_2)^2+c_2^2\right]}, \quad n=3$$

where n is the order of the shaper, and r_i, c_i are the real and imaging parts. The roots are:

$$(s+r_2)^2 + c_2^2 = 0 \Rightarrow s + r_2 = \pm jc_2 \Rightarrow s = -r_2 \pm jc_2$$

so the transfer function can be written with the simple fractions like :

$$T(s) = \frac{K_1}{(s+p_1)} + \frac{K_2}{(s+r_2-jc_2)} + \frac{K_3}{(s+r_2+jc_2)} \tag{1}$$

where one real pole, $\text{pole}_\emptyset = -p_1$ and the two complex poles, $\text{pole}_1 = -r_2 + jc_2$ and $\text{pole}_2 = -r_2 - jc_2 = p_1^*$, $\Re\text{pole}_1 = -r_2$, $\Im\text{pole}_1 = c_2$. The coefficients K_i are :

$$\begin{aligned}
K_1 &= \frac{1}{(s+r_2-jc_2)(s+r_2+jc_2)}\bigg|_{s=-p_1} \\
&= \frac{1}{(-p_1+r_2-jc_2)(-p_1+r_2+jc_2)} = \frac{1}{(r_2-p_1)^2+c_2^2}, \quad \in \Re \\
K_2 &= \frac{1}{(s+p_1)(s+r_2+jc_2)}\bigg|_{s=-r_2+jc_2} \\
&= \frac{1}{(-r_2-jc_2+p_1)(\cancel{-r_2}+jc_2+\cancel{r_2}+jc_2)} \\
&= \frac{1}{2jc_2(p_1-r_2+jc_2)} = |K_2|e^{j\phi}, \quad \phi = \angle K_2, \quad \in \mathbb{C} \\
K_3 &= \frac{1}{(s+p_1)(s+r_2-jc_2)}\bigg|_{s=-r_2-jc_2} \\
&= \frac{1}{(-r_2-jc_2+p_1)(\cancel{-r_2}+jc_2+\cancel{r_2}-jc_2)} \\
&= \frac{1}{-2jc_2(p_1-r_2-jc_2)} = K_2^*, \quad \in \mathbb{C}
\end{aligned} \tag{2}$$

From Eq. (2) the Eq. (1) will be:

$$T(s) = \frac{K_1}{(s+p_1)} + \frac{K_2}{(s+r_2-jc_2)} + \frac{K_2^*}{(s+r_2+jc_2)} \tag{3}$$

The time-domain representations (apart from the amplitude factor) of the shapers in Eq. (3) can be calculated as the inverse Laplace transform $T(s) \overset{\mathcal{L}^{-1}}{\longleftrightarrow} f(t)$:

$$\begin{aligned} f(t) &= K_1 e^{-p_1 t} + K_2 e^{(-r_2+jc_2)t} + K_2^* e^{(-r_2-jc_2)t} \\ &= K_1 e^{-p_1 t} + e^{-r_2 t}\left[K_2 e^{jc_2 t} + K_2^* e^{-jc_2 t}\right] \\ &= K_1 e^{-p_1 t} + 2e^{-r_2 t} 2\Re\left(K_2 e^{jc_2 t}\right) = K_1 e^{-p_1 t} + e^{-r_2 t} 2\Re\left(|K_2| e^{j\phi} e^{jc_2 t}\right) \\ &= K_1 e^{-p_1 t} + 2|K_2| e^{-r_2 t}\cos\left(c_2 t + \phi\right), \quad \text{where } \phi = \angle K_2 \\ &= K_1 e^{\text{pole}_\emptyset t} + 2|K_2| e^{\Re\text{pole}_1 t}\cos\left(\Im\text{pole}_1 t + \angle K_2\right) \end{aligned}$$

where

$$|K_2| = \frac{1}{2c_2\sqrt{\left(p_1 - r_2\right)^2 + c_2^2}} \quad \overset{(2)}{\Longrightarrow} 4c_2^2|K_2|^2 = K_1$$

$$K_1 = \frac{1}{\left(\text{pole}_\emptyset - \Re\text{pole}_1\right)^2 + \Im\text{pole}_1^2}$$

If someone defines the normalized: $\overline{H}\left(f\right) = H\left(f\right)/\tau$ which allows in our calculations, to take into account the proportionality of $H\left(f\right)$ to τ (so that the integral, a measure of the amplitude, is independent of τ). Through the normalization, the VMM shaper constants are :

$$\begin{aligned} \alpha &= 10^{-8} \\ \text{pole}_\emptyset &= \frac{1.263}{\alpha} \\ \text{pole}_1 &= (1.149 - j0.789)\frac{1}{\alpha} \\ K_1 &= 1.584 \\ K_2 &= -0.792 - 0.115j \\ t_{peak} &= 1.5\alpha \end{aligned}$$

and the final function can be written in a computational form:

$$f(t) = \alpha^3|\text{pole}_\emptyset|\left(|\text{pole}_1|\right)^2\left[K_1 e^{-t\text{pole}_\emptyset} + 2|K_2| e^{-t\Re\text{pole}_1}\cos\left(-t\Im\text{pole}_1 + \angle K_2\right)\right]$$

The Code implementation in ROOT

```
double vmmResponse(vector<double> electronsTime,
                   vector<double> electronsGain, double &slope,
                   double electronicsThreshold, double &amplitudeFirstPeak){

    TH1D *response = new TH1D("response","VMM response",5000,0,500);

response->Reset();
double PeakingTime = 25.;
double thresh = 0.05;
double a = (PeakingTime*(10^-9))/1.5;
double pole0 = 1.263/a;  // real pole
double Re_pole1 = 1.149/a;
```

```
double Im_pole1 = -0.786/a; //complex pole
double K0 = 1.584;
double Re_K1 = -0.792;
double Im_K1 = -0.115;
double pole1_square = Re_pole1*Re_pole1 + Im_pole1*Im_pole1;
double K1_abs = TMath::Sqrt(Re_K1*Re_K1 + Im_K1*Im_K1);
double argK1 = TMath::ATan2(Im_K1, Re_K1);
if(debugElx)
 cout<<"#Starting VMM responce with size of electrons: "<<electronsTime.size()
 <<endl;
double t,st,timeStep;
timeStep=0.1;
for (Int_t i=0; i<electronsTime.size(); i++) {
 if(debugElx)
  cout<<"#electrons with time: "<<electronsTime[i]<<", and gain: "
  <<electronsGain[i]<<endl;
   for(double ti=electronsTime[i];ti<=500;ti=ti+timeStep){
    t = (ti-electronsTime[i])*(10^-9);
    st = electronsGain[i]* TMath::Power(a,3)*pole0*pole1_square*
    ((K0*TMath::Exp(-t*pole0))+(2.*K1_abs*TMath::Exp(-t*Re_pole1)
    *cos(-t*Im_pole1+argK1)));
          response->Fill(ti,st);
   }
}
    double timeAboveThr = response->GetXaxis()->
    GetBinCenter(response->FindFirstBinAbove(electronicsThreshold));

    TF1 *polFit = new TF1("polFit","pol1",0,500);
    polFit->SetLineColor(kRed);
    response->Fit("polFit","QR","",timeAboveThr,timeAboveThr+2.);
    slope = polFit->GetParameter(1);
    if(debug)
        cout<<"          Threshold found: "<< timeAboveThr <<endl;


    // peak finder
    TSpectrum *s = new TSpectrum(10,1); //Spectrum for the peaks detection
    double tFromFirstPeak = 50000;
    Int_t nfound = s->Search(response,1,"nobackground,goff");
    Float_t *xpeaks = s->GetPositionX();
    for(int j = 0;j<nfound;j++){
        if(response->GetBinContent(response->
        FindBin(xpeaks[j]))<electronicsThreshold){
            continue;
        }
        if(tFromFirstPeak>xpeaks[j])
            tFromFirstPeak = xpeaks[j];
    }
    int binFromFirstPeak = response->FindBin(tFromFirstPeak);
    amplitudeFirstPeak = response->GetBinContent(binFromFirstPeak);

    if(debugElx){
```

```
        TCanvas *responseCanvas = new TCanvas("responseCanvas",
                                  "Response",600,600);
        responseCanvas->cd();
        response->Draw("C");
        cout<<"Plotting histogram with first peak:"<<tFromFirstPeak
        <<", slope: "<<slope <<", amplitude at first peak: "<<
        amplitudeFirstPeak<<", Hit enter"<<endl;
        responseCanvas->Update();
        getchar();
        delete responseCanvas;
    }
    delete response,polFit;
    return timeAboveThr;
}
```