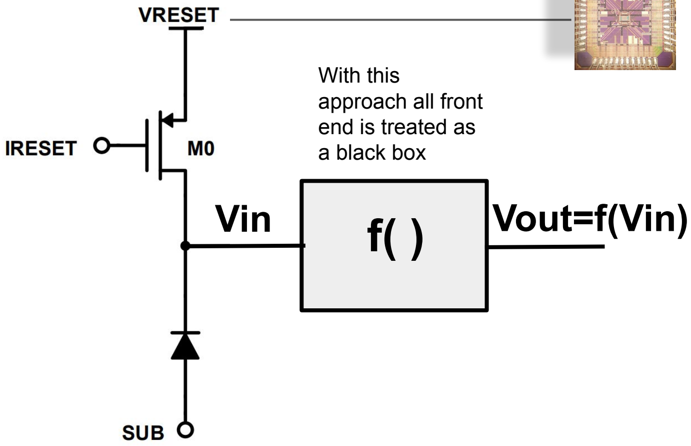
APTS OA

Gain correction methods

Considered Methods



- Three methods have been implemented and tested:
  - OUR OLD METHOD
  - SF METHOD
  - MY TB METHOD

- All three methods share the same starting hypothesis: the transfer function from input to output voltage is independent of the signal shape and frequency and it can be estimated via Baseline vs Vreset measurements:

$$Vout = f(Vin) ; Vbl = h(Vres)\ HP : f(V) = h(V)\ \forall\ V$$

- Our goal is to find Vin known Vout, therefore to invert the transfer function.

Our Old Method

- The working point has been chosen to keep the transfer function as close as possible to a linear function

- Therefore the following first order expansion should be a good approximation of f

$$f(V) = f_0 + \partial f \cdot \Delta V + o(V)$$

- The derivative has been evaluated numerically by fitting 9 point intervals of Vbl(Vres) data.

- This method allows to estimate Vin as follows:

$$\Delta Vin = \frac{\Delta Vout}{\partial f_{Vres}}$$

SF Method

- APTS SF uses a different approach:

$$Vout = f(Vin) \Rightarrow Vin = f^{-1}(Vout)$$

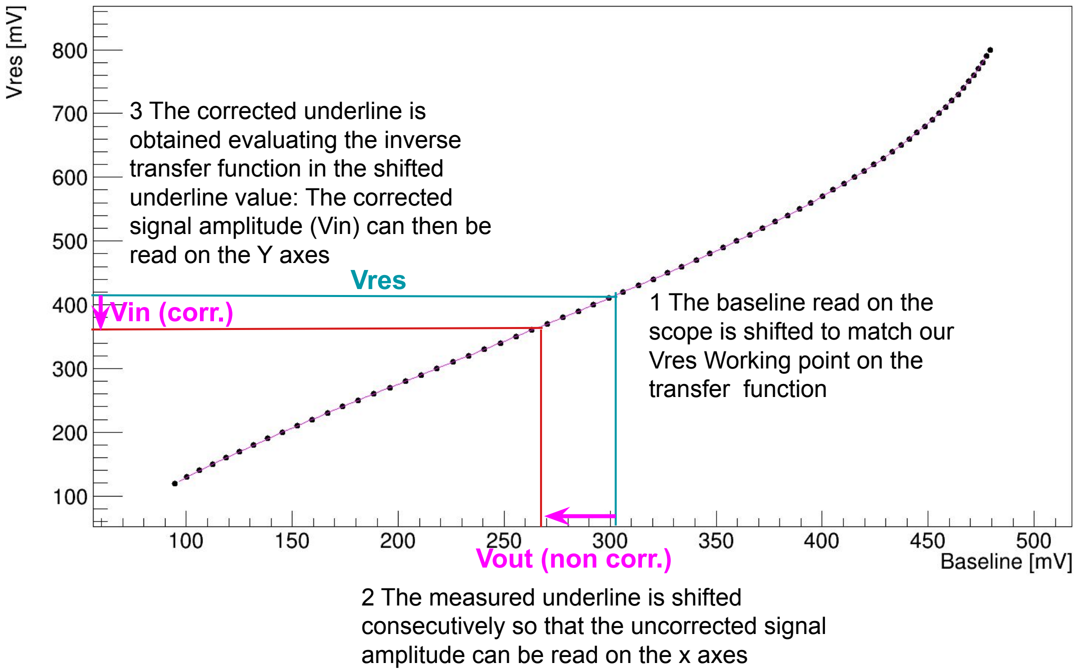
where f^{-1} is obtained via cubic interpolation of Vres(BI) data: since we measure BI(Vres) we need to invert the axes.

- This method provides different correction factors for signal with different amplitudes, but requires to transform both baseline and underline separately via the transfer function

SF Method 2

- To implement this method in our case a compensation for the scope data's offset had to be introduced

- The interpolation has been performed with TSpline3 Root class (vs scypy.interpol1D used by SF)



new SF method implementation

ARGUMENTS: name of root file containing data, a label with board and Vbb information, px number, Vres working point, **BASELINE and UNDERLINE** of the signal that needs correction

```
double apts_gain_corr(TString gainfile, TString label, int kpx, double Vres, double Bl, double Ul ){
  //-------open gainfile and get baseline graph------//
   TFile* gf = new TFile(gainfile.Data());
   TObjArray* arrPar=label.Tokenize("_");
   TString Flavour;
   TString Board;
   TString Vbb;
   TString VbbValue;
   if(arrPar->GetEntries()!=4){
     printf("ERROR: expect flavour_Board number_Backbias\n");
     return 0;
   }
   else{
     for(int i=0; i<arrPar->GetEntries(); i++){
       TObjString* strB =(TObjString*)arrPar->At(i);
       TString theStrB =strB->GetString();
       theStrB.ReplaceAll("\n","");
       if(i==0) Flavour=theStrB.Data();
       else if(i==1) Board=theStrB.Data();
       else if(i==2) Vbb=theStrB.Data();
       else if(i==3) VbbValue=theStrB.Data();
     }
    TGraph* gBl;
    TGraph* gBl2 = new TGraph();

    //-----------if is needed to acces scope data of pixel 6 and 9, can be removed if ADC data are interested-----//
    if(kpx == 6 || kpx == 10)
    gBl = (TGraph*)gf->Get(Form("%s:/%s/%s_W22%s/%s_%s/Px_J%ds/Baseline", gainfile.Data(), Flavour.Data(), Flavour.Data(), Board.Data(), Vbb.Data(), V
    else
    gBl = (TGraph*)gf->Get(Form("%s:/%s/%s_W22%s/%s_%s/Px_J%d/Baseline", gainfile.Data(), Flavour.Data(), Flavour.Data(), Board.Data(), Vbb.Data(),
```

ON THE LEFT:
I'm just opening the data stored into a tgraph

ON THE RIGHT:
The actual implementation of the method

```
    //-------switch axes to get Vres(baseline)-----graph//
    double bl, vr;
    for (int i=0; i<gBl->GetN()-20; i++){
        gBl->GetPoint(i+10, vr, bl);
        gBl2->SetPoint(i, bl, vr);
    }
    gBl2 = Smooth(gBl2, 2); //optional smmothing
    gBl2->GetXaxis()->SetTitle("Baseline [mV]");
    gBl2->GetYaxis()->SetTitle("Vres [mV]");
    gBl2->SetMarkerStyle(20);
    //------interpolate the bseline graph with spline3 function------//
    TSpline3 *f3 = new TSpline3("f3", gBl2);
    f3->SetLineColor(kMagenta);
    //-------correction for not fixed scope offset---------//
    double tbl = gBl->GetPointY((int)(Vres-20)/10);
    double d = tbl - Bl;
    double nUl = Ul + d;
    //--------corrects values---------//
    tbl = Vres;
    nUl = f3 -> Eval(nUl);
    //----------debug canvas---------//
   /* TCanvas* cres = new TCanvas("cres","Fit Residual Plot",1650,900);
        gBl2->Draw("AP");
        f3->Draw("same");   */
    return (Bl-Ul)/(tbl-nUl);
  }
}
```

The function now returns a correction value to apply to the input signal amplitude like our old correction to ease the implementation into our scripts

My Method

- Before knowing of SF method in november I implemented the following method to apply a different correction to different signal amplitudes:
- I start from this obvious equality

$$Vout = f(Vin) = \int \partial f(V)\, dV$$

- If I call G the derivative, this integral can be calculated in our case via the Integral Average Theorem as follows:

$$\Delta Vout = \int_{Vres - Vin}^{Vres} G(Vres - V)\, dV = \bar{G}\, \Delta Vin$$

Where $\bar{G}$ is the average of G in the integration domain. In this way we obtain an easy to invert function (linear) to get ΔVin known ΔVout as:

$$\Delta Vin = \frac{\Delta Vout}{\bar{G}}$$

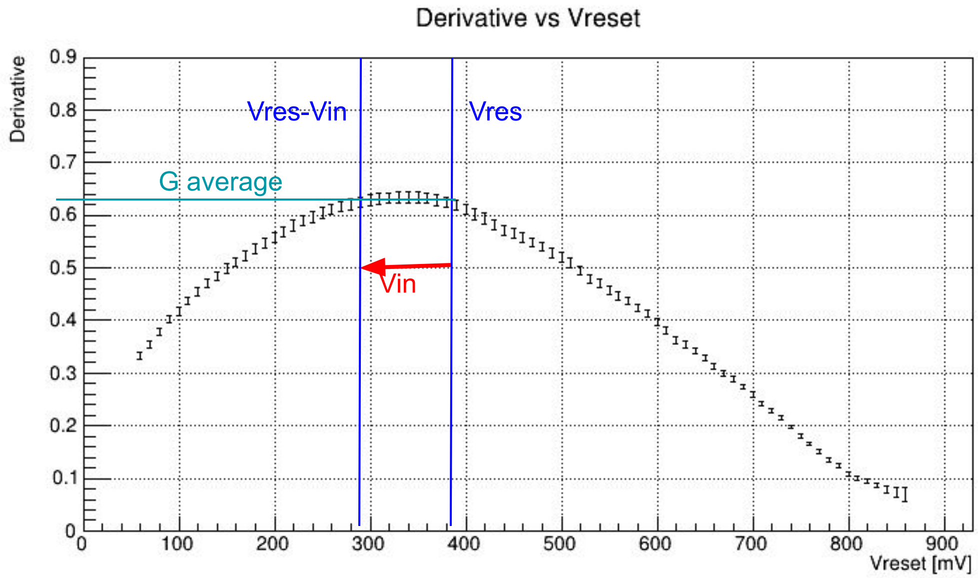
My Method 2

- $\bar{G}$ has been evaluated as the average of n points of the Derivative(Vres) plot as shown in figure

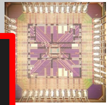
- In case Vin falls in between two points a weighted average of the two closest values is made

- Since Vin is not known a-priori but is needed to define the interval on which the average has to be calculated, it is estimated in first approximation as in our old method:

$$\Delta Vin_{range} = \frac{\Delta Vout}{G(Vres)}$$



My method new implementation



ARGUMENTS: name of root file containing data, a label with board and Vbb information, px number, Vres working point, **AMPLITUDE** of the signal that needs correction

```
double CorrGain2(TString gainfile, TString label, int kpx, double Vres, double Vout){

//-------open gainfile and get baseline graph-------//
TFile* gf = new TFile(gainfile.Data());
TObjArray* arrPar=label.Tokenize("_");
TString Flavour;
TString Board;
TString Vbb;
TString VbbValue;
if(arrPar->GetEntries()!=4){
  printf("ERROR: expect flavour_Board number_Backbias\n");
  return 0;
}
else{
  for(int i=0; i<arrPar->GetEntries(); i++){
    TObjString* strB =(TObjString*)arrPar->At(i);
    TString theStrB =strB->GetString();
    theStrB.ReplaceAll("\n","");
    if(i==0) Flavour=theStrB.Data();
    else if(i==1) Board=theStrB.Data();
    else if(i==2) Vbb=theStrB.Data();
    else if(i==3) VbbValue=theStrB.Data();
  }
  TGraphErrors* g;

  //-----------if is needed to acces scope data of pixel 6 and 9, should be removed if only ADC data are need
  if(kpx == 6 || kpx == 10)
  g = (TGraphErrors*)gf->Get(Form("%s:/%s/%s_W22%s/%s_%s/Px_J%ds/Derivative", gainfile.Data(), Flavour.Data(), Flavour.Data(), Board.Data(), Vbb.Data(), Vb
  else
  g = (TGraphErrors*)gf->Get(Form("%s:/%s/%s_W22%s/%s_%s/Px_J%d/Derivative", gainfile.Data(), Flavour.Data(), Flavour.Data(), Board.Data(), Vbb.Data(), Vbb
```

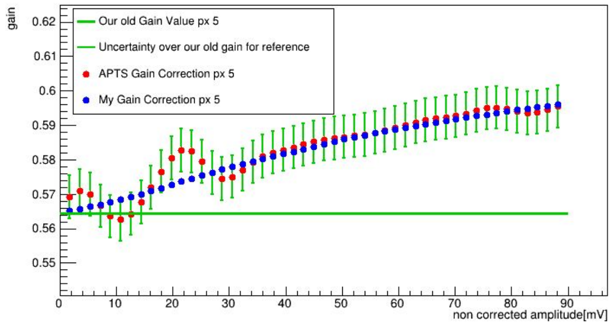
ON THE LEFT:
I'm just opening the data stored into a tgraph

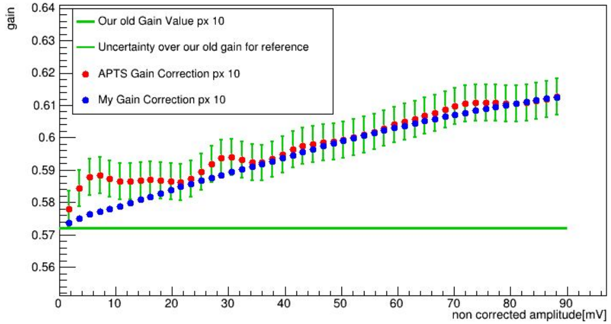
ON THE RIGHT:
The actual implementation of the method

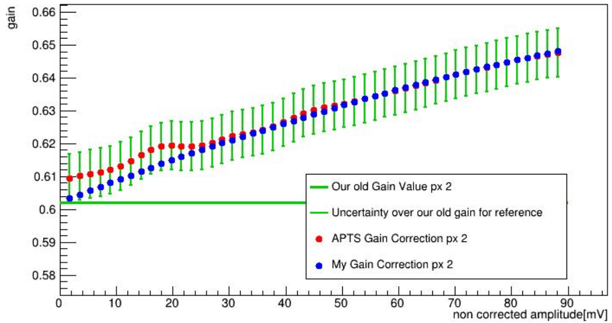
The function returns a correction value to apply to the input signal amplitude like our old correction to ease the implementation into our scripts

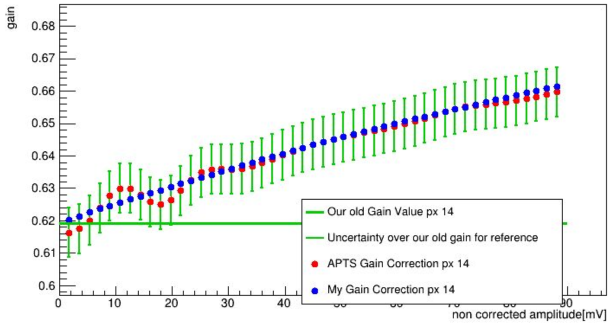
```
double cgl,cgu, vin, gp;
double nn, s;
int n, neff1, neff2;
//----estimate Vin to define average range----//
vin=Vout/g->GetPointY((int)(Vres-20)/10);
nn=vin/10+1;
n=(int)nn; //number of points to average
s=nn-n; // weight for weighted average
cgl=0;
cgu=0;
neff1=0;
neff2=0;
for (int i=0; i<n; i++) { //lower gain value
  gp=g->GetPointY((int)(Vres-20)/10-i);
  if(gp<0.2) continue;
  neff1++;
  cgl+=gp;
}
for (int i=0; i<n+1; i++) {//higher gain value
  gp=g->GetPointY((int)(Vres-20)/10-i);
  if(gp<0.2) continue;
  neff2++;
  cgu+=gp;
}
cgl=cgl/neff1;
cgu=cgu/neff2;
return cgl*(1-s)+cgu*s; //final weighted average
}
```

Results B6, Vbb=0V

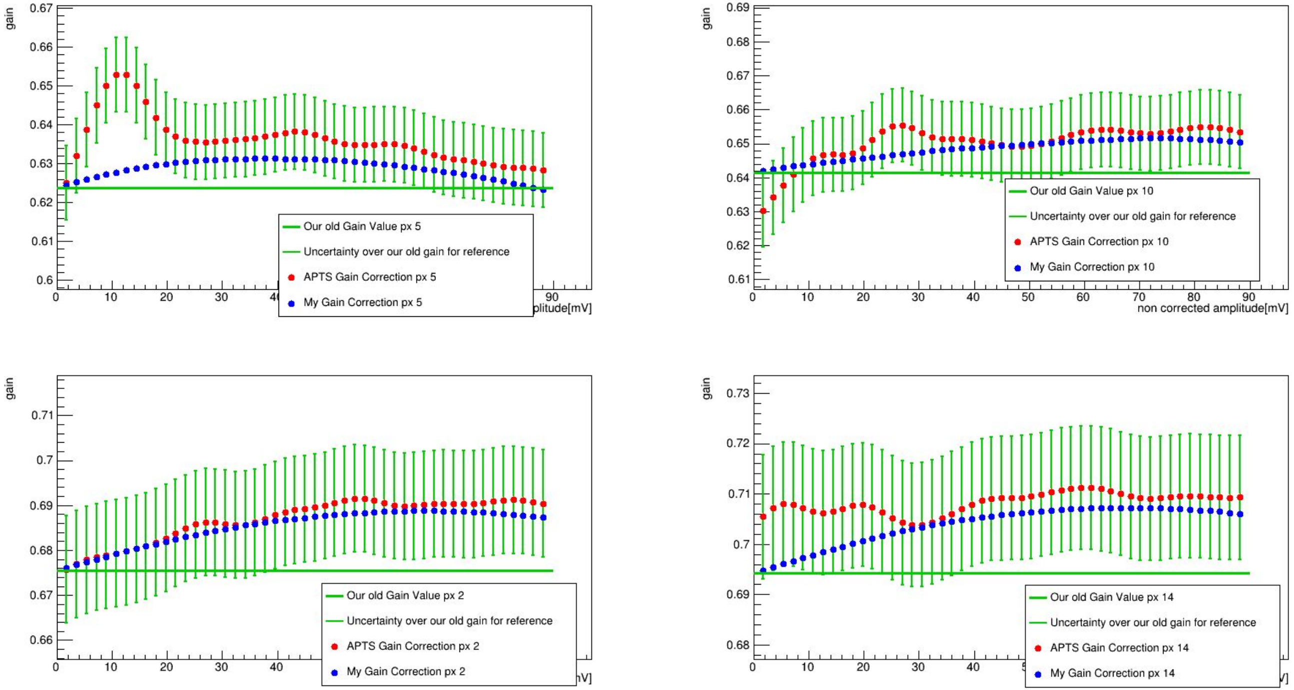




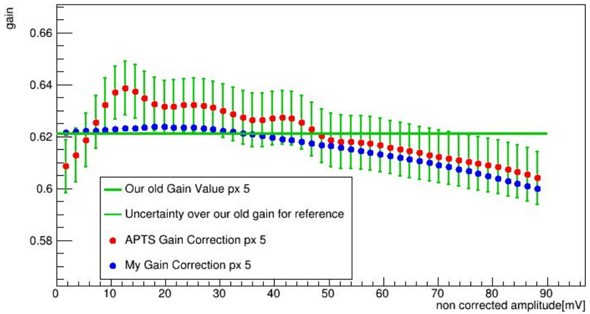


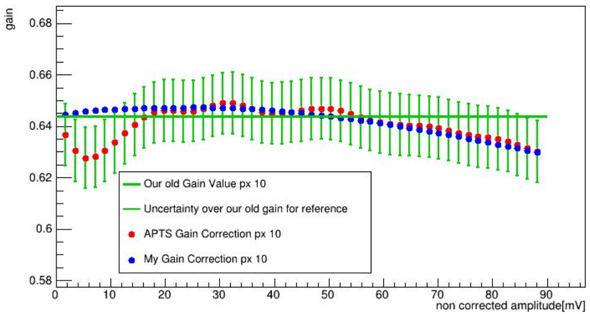


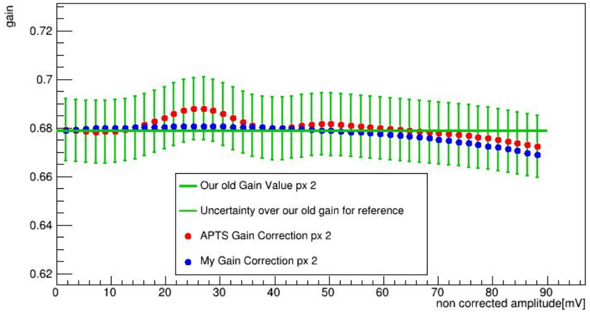
Results B6, Vbb=2.4V

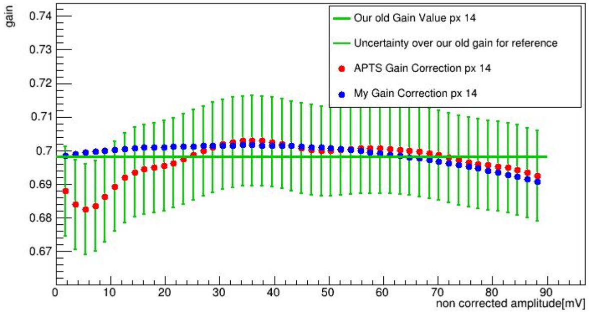


Results B6, Vbb=4.8V

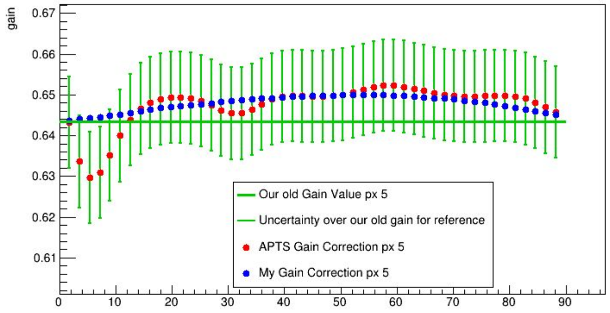


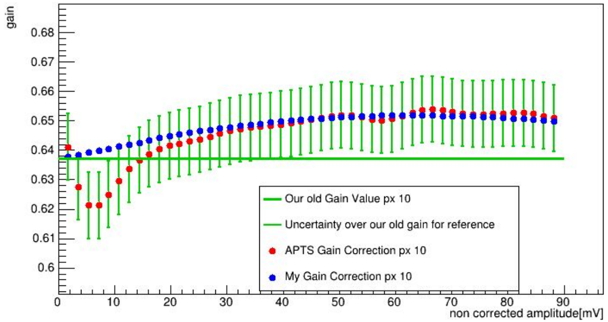


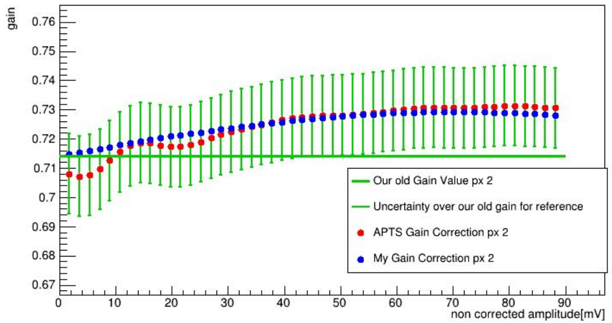


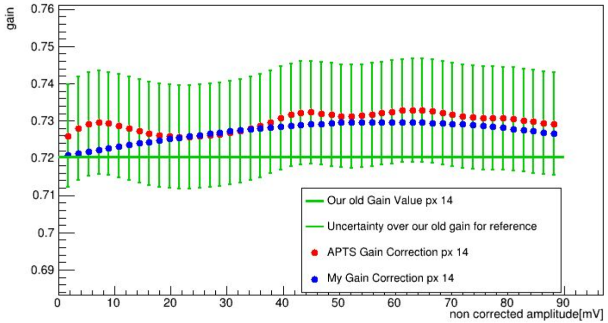


Results B13, Vbb=4.8V

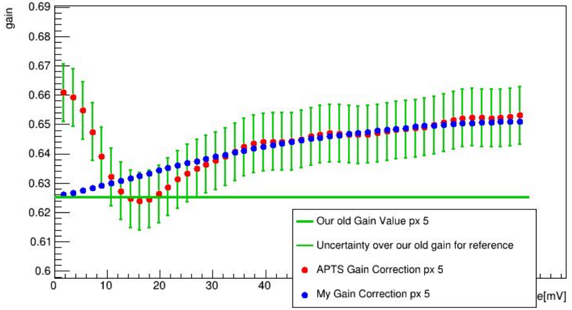


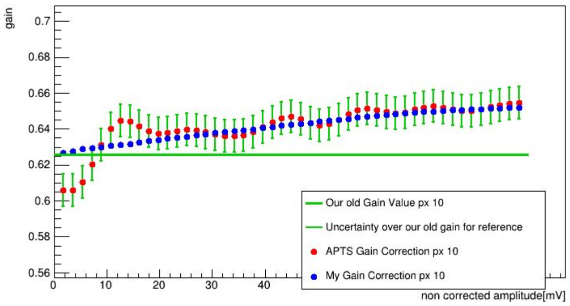


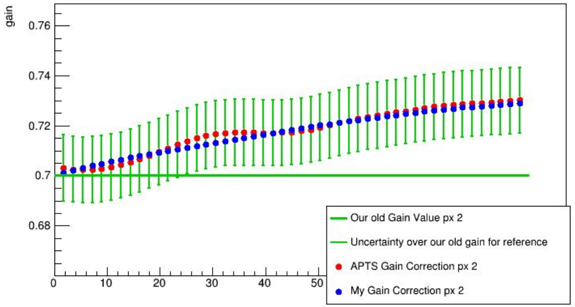


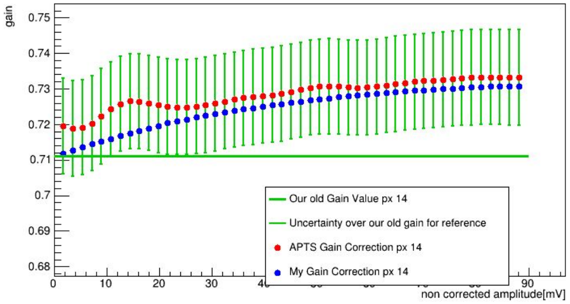


Results B13, Vbb=2.4V





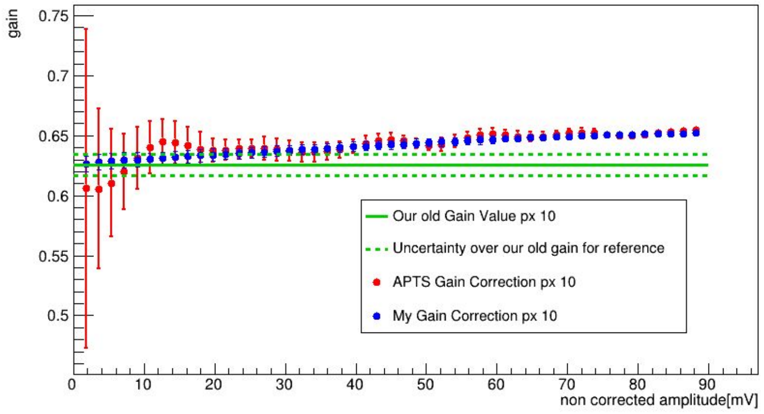




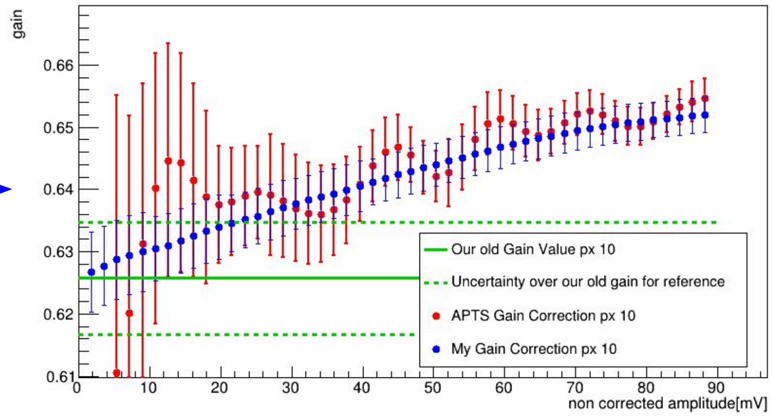
Error estimation

- Since last version of this presentation I implemented uncertainty evaluation for both SF and my method propagating the uncertainty over the baseline in the first case, over it's derivative in the second.

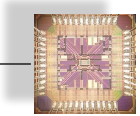
- The error bars describe well the fluctuations observed and further prove the perfect convergence of the two new correction methods for higher signal amplitudes.



Zoom over Y



Conclusions



- All methods tested give coherente results between each other for any given Vbb, working point, board type both for scope and ADC data

- The two methods that allow for a variable gain are in accordance with each other in all conditions as expected

- The SF method seems to have larger fluctuations deriving from our baseline uncertainty: smoothing the input data gives some benefit

- My method, although it is mathematically equivalent to the SF one (as demonstrated in the first part of this presentation), is less susceptible to our data fluctuations since both the derivative and integration operations introduce some smoothing, since an average over some points is being performed