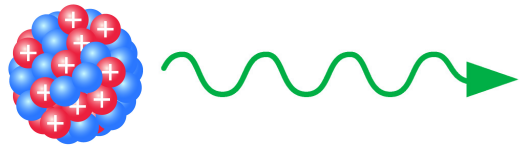
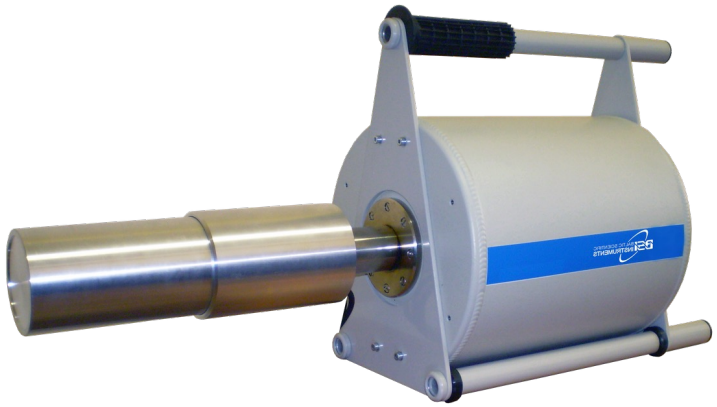
Simulations of γ-ray singles and γ-γ coincidence spectra

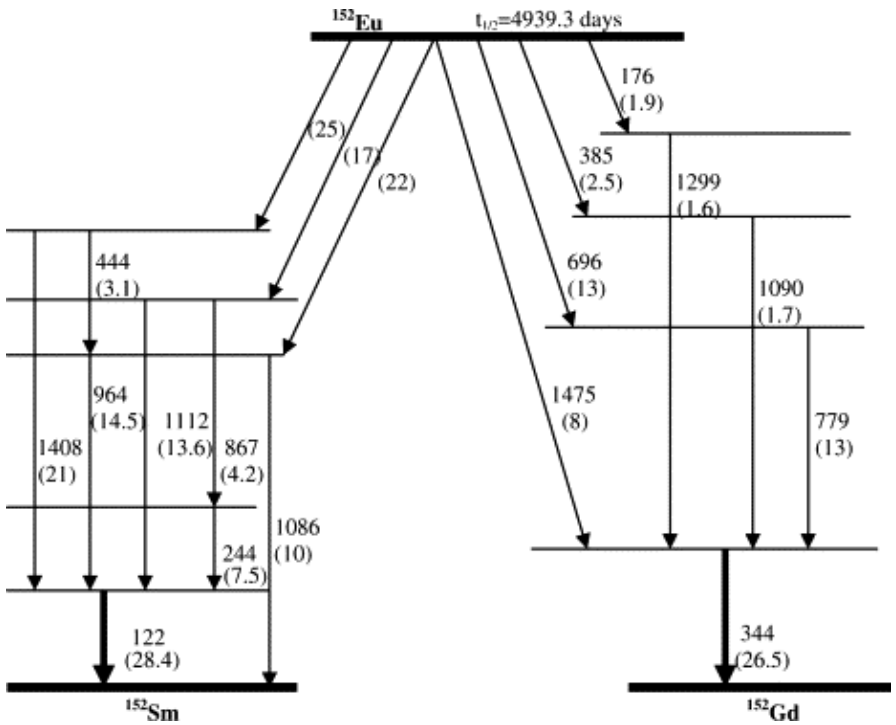
Pietro Spagnoletti

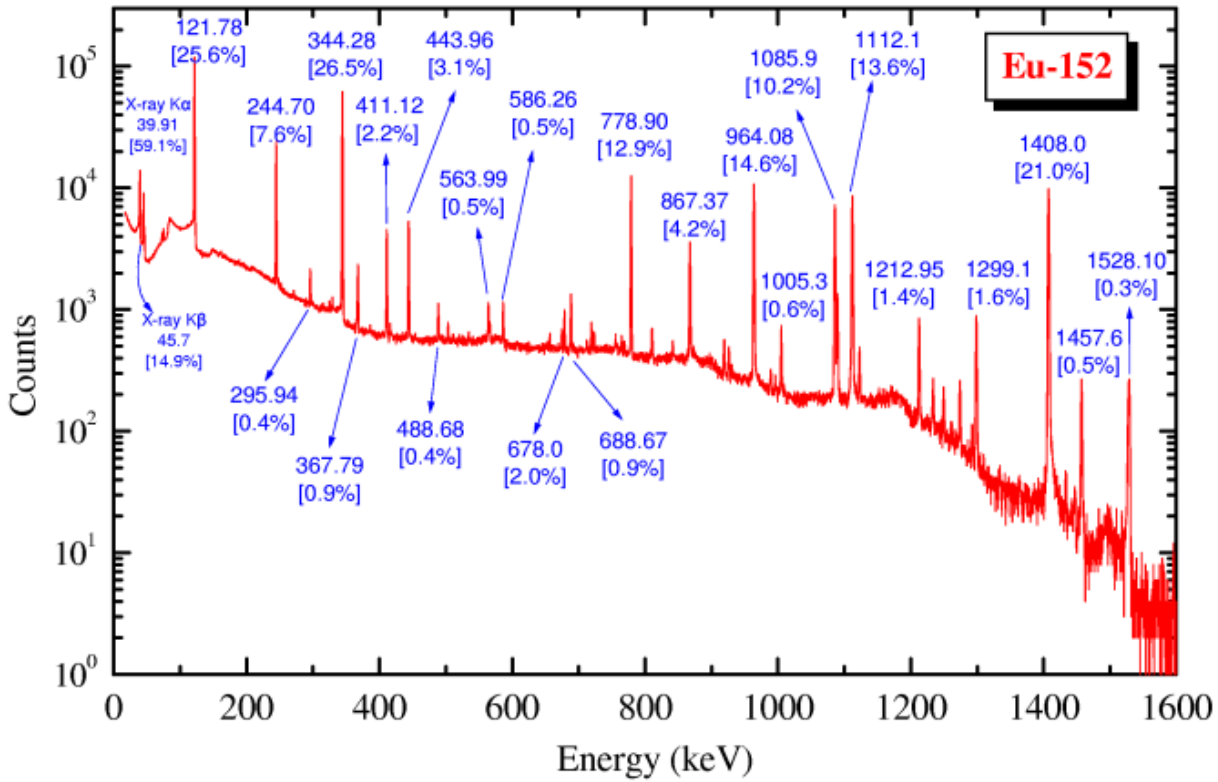
GIFFIN Analysis meeting

Singles γ-ray spectrum

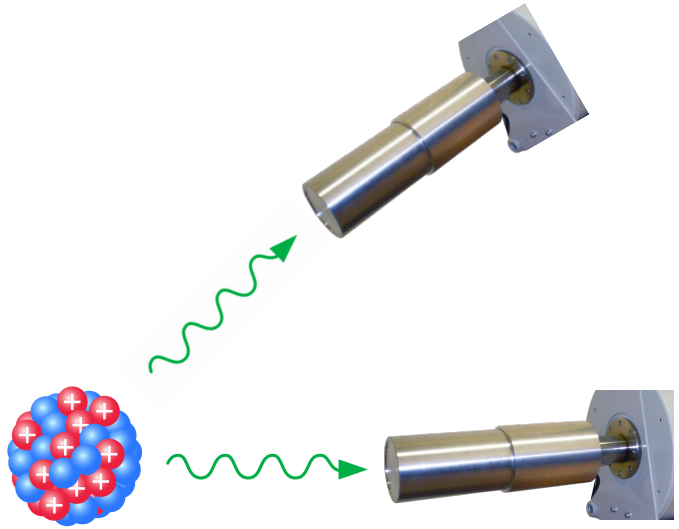




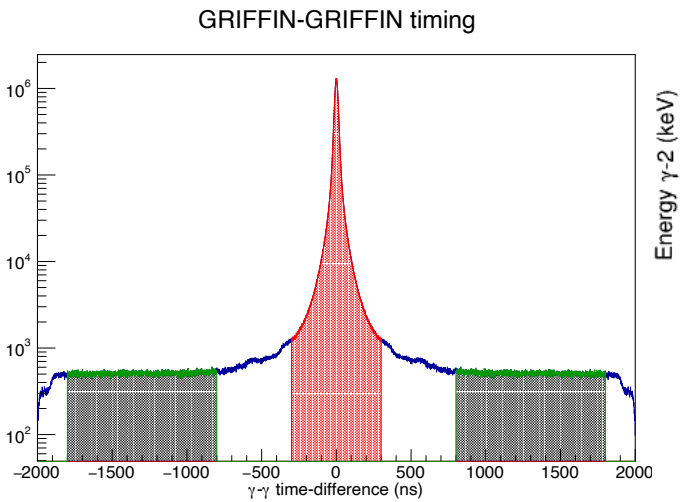


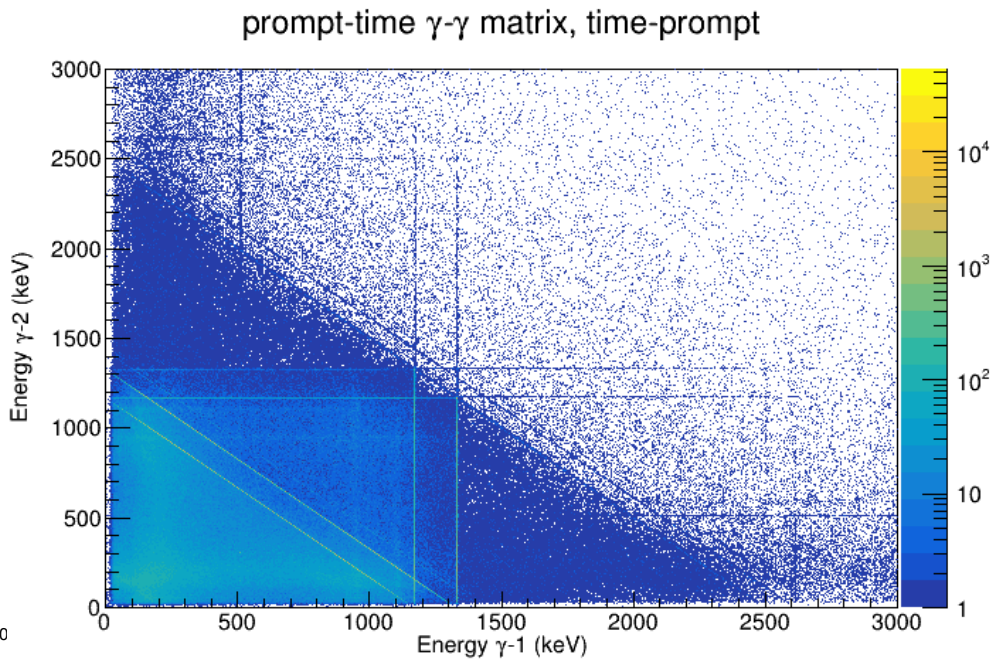


γ-γ coincidence spectroscopy

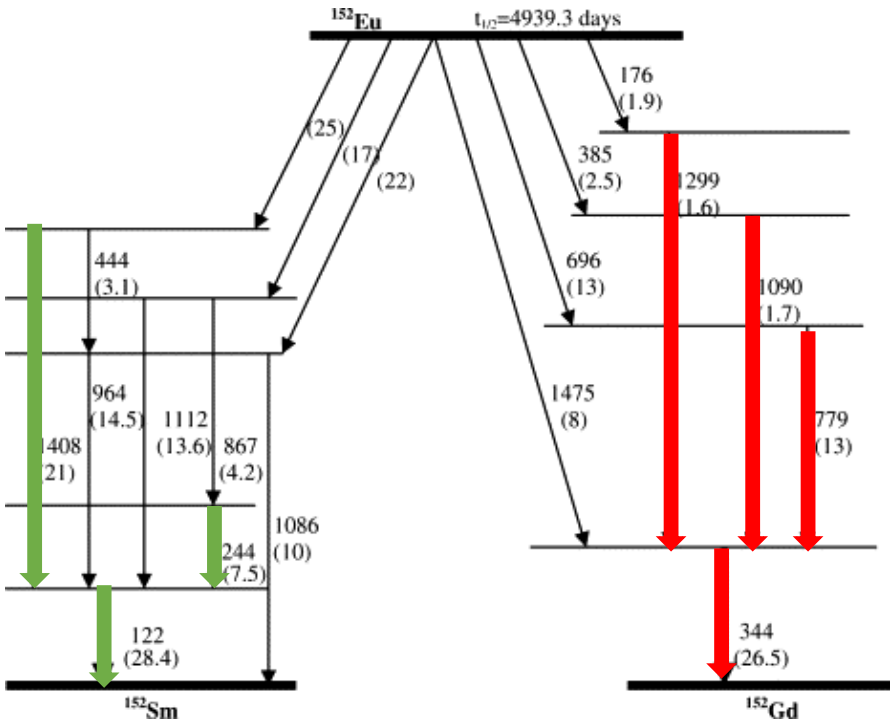


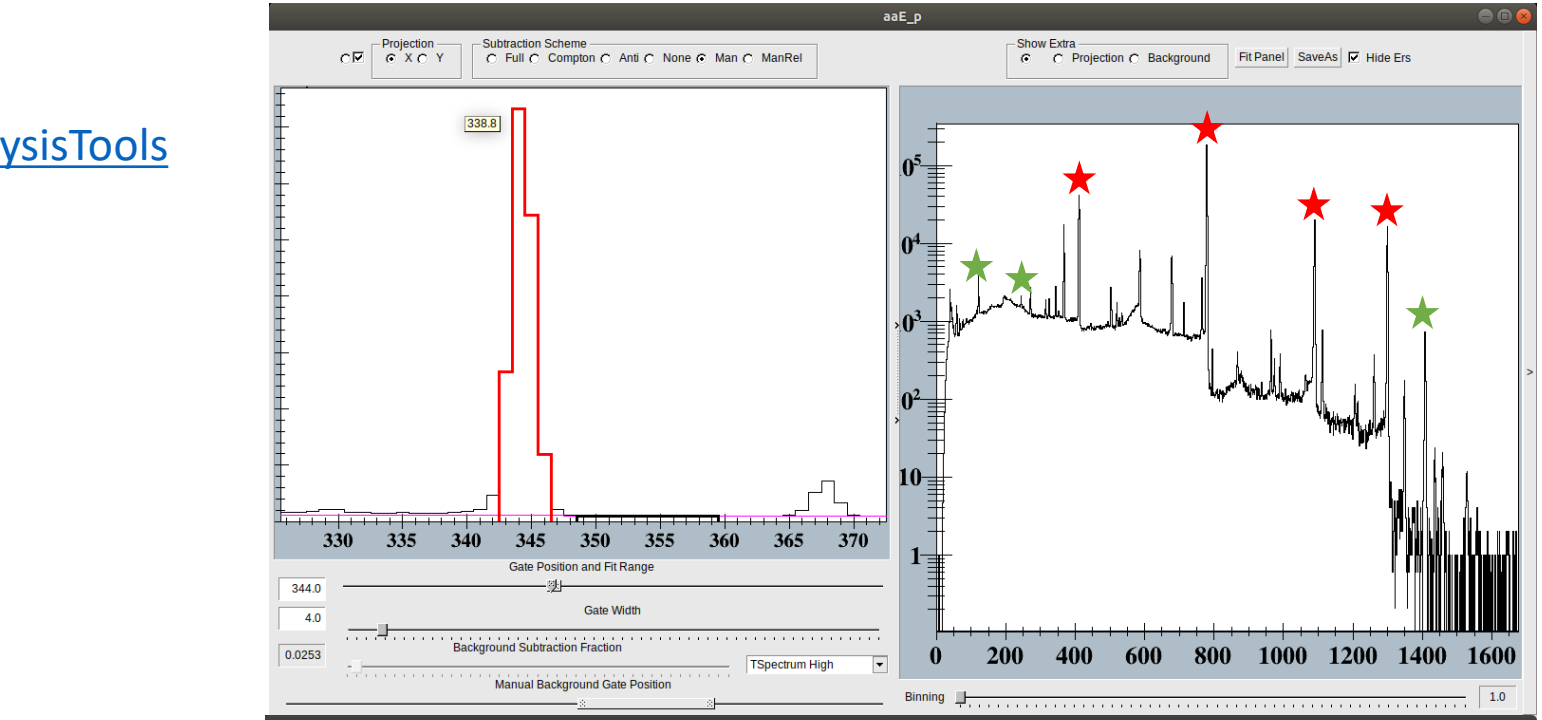
```
for(auto g = 0; g < fGrif->GetSuppressedMultiplicity(fBgo); ++g) {
    auto grif = fGrif->GetSuppressedHit(g);
    fH1.at("gE")->Fill(grif->GetEnergy()); //fill singles spectrum
    //gamma-gamma
    for(auto g2 = 0; g2 < fGrif->GetSuppressedMultiplicity(fBgo); ++g2){
        if( g == g2 ) continue; //don't compare event with itself !!!!
        auto grif2 = fGrif->GetSuppressedHit(g2);
        fH1.at("ggTd")->Fill( grif->GetTime() - grif2->GetTime() );
        if( PromptCoincidence(grif, grif2) ){
            fH2.at("ggE_p")->Fill(grif->GetEnergy(), grif2->GetEnergy()); //fill gamma-gamma coinc matrix (prompt-coincidence)
        }
        else if( TimeRandom(grif, grif2) ){
            fH2.at("ggE_r")->Fill(grif->GetEnergy(), grif2->GetEnergy()); //fill gamma-gamma coinc matrix (random-coincidence)
        }
    }
}
```

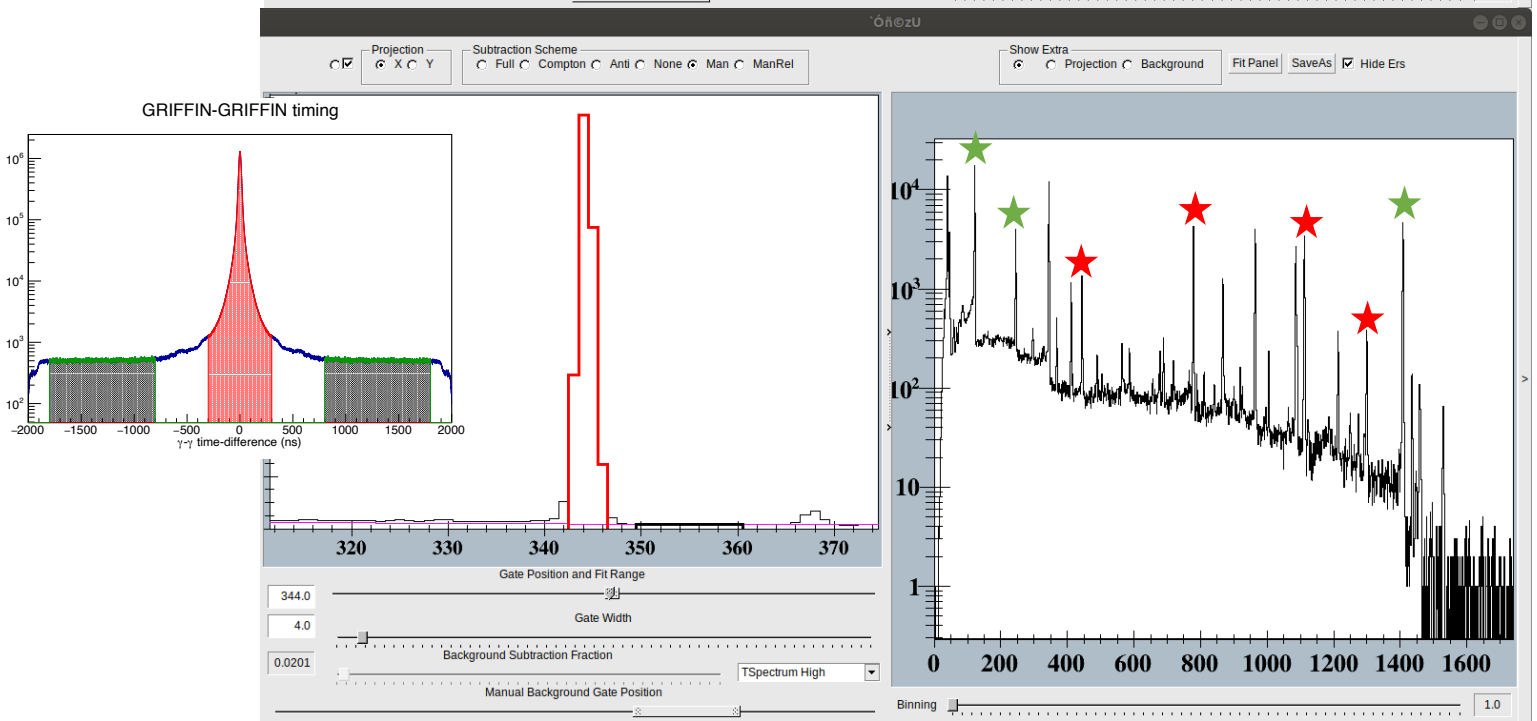


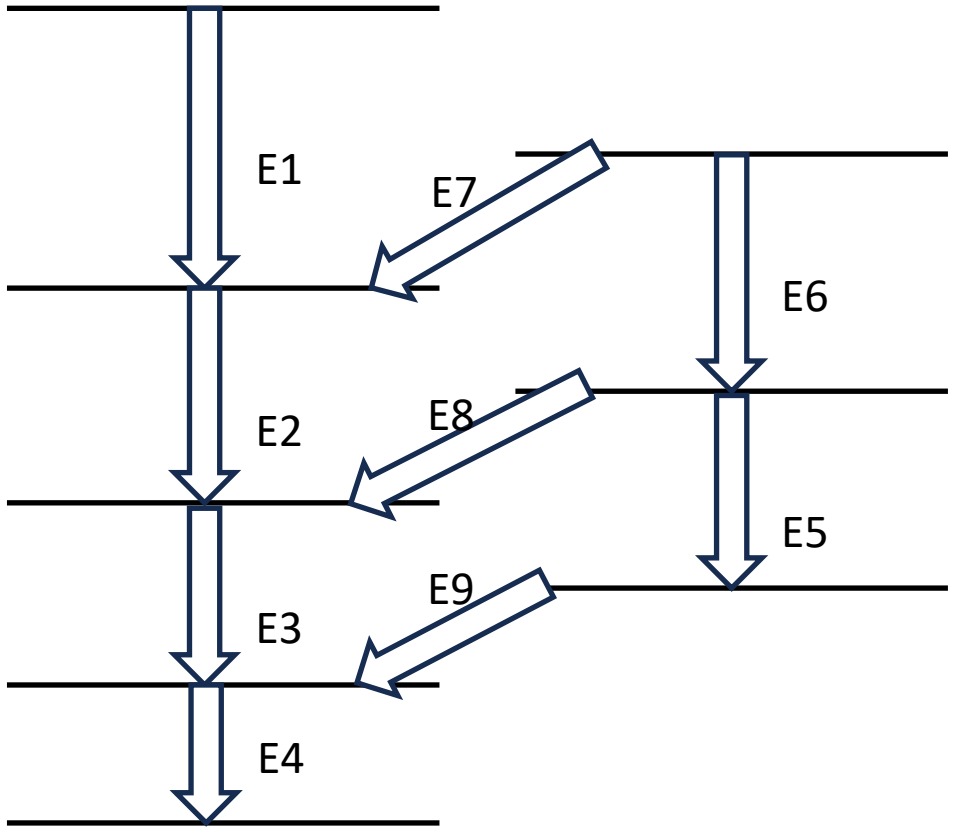


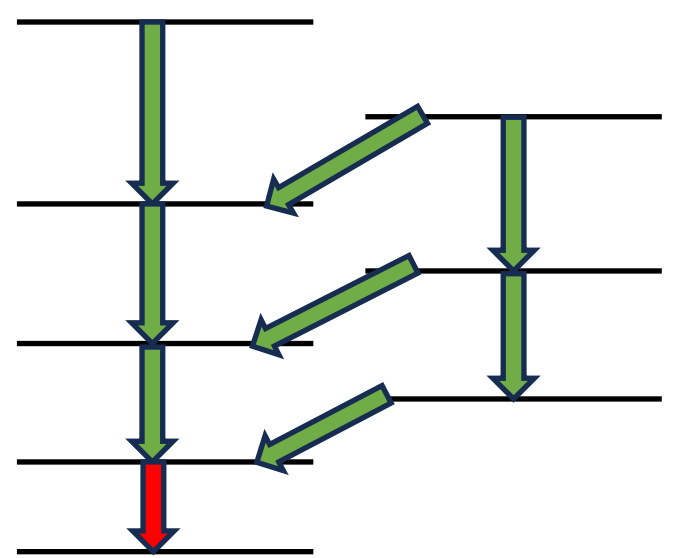
https://github.com/jsmallcombe/jRootAnalysisTools

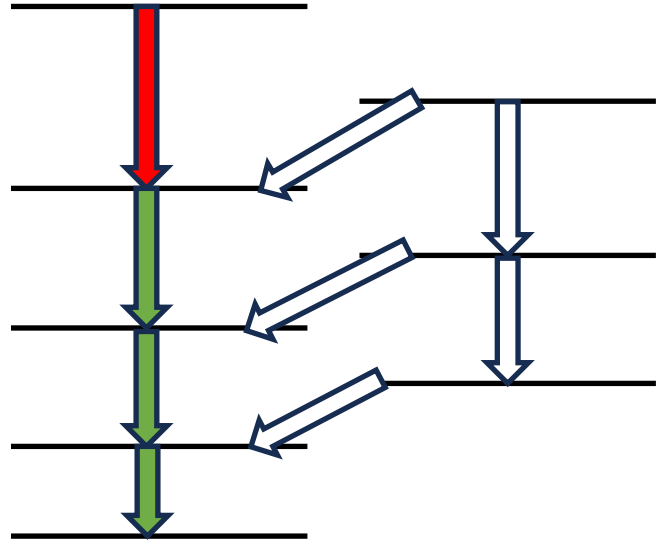


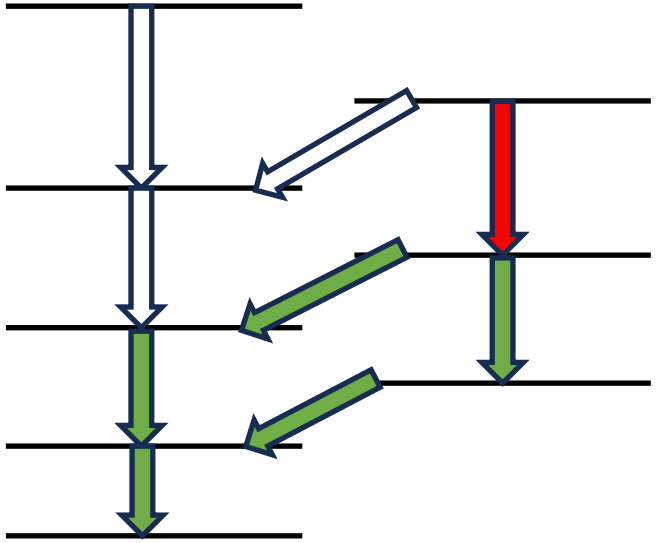


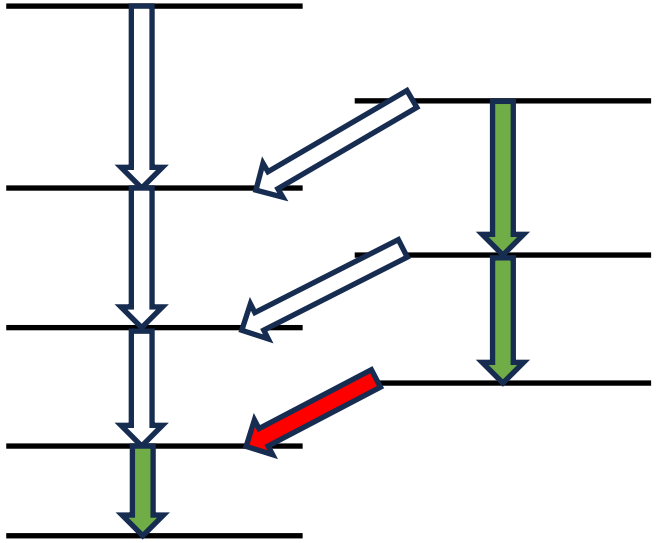












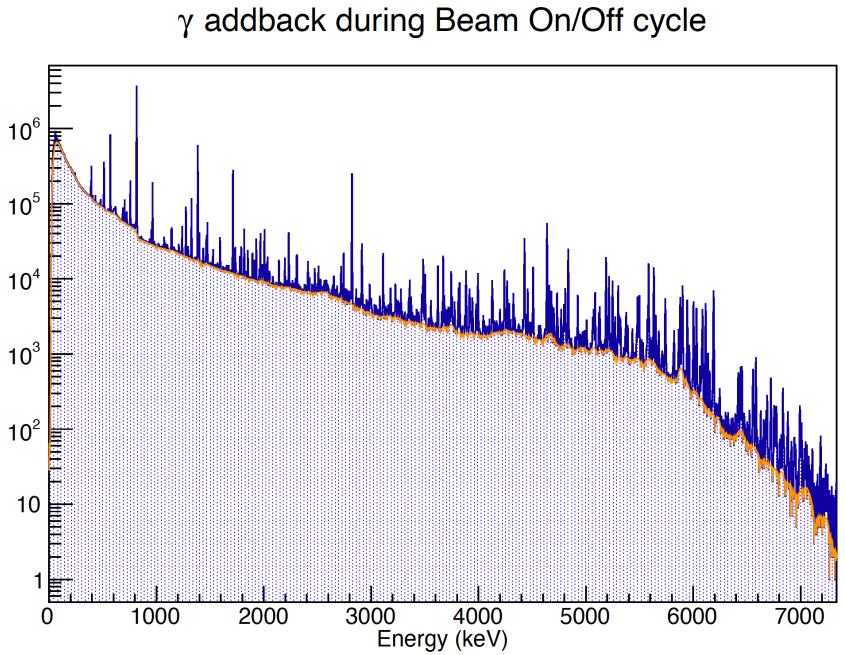
Gate

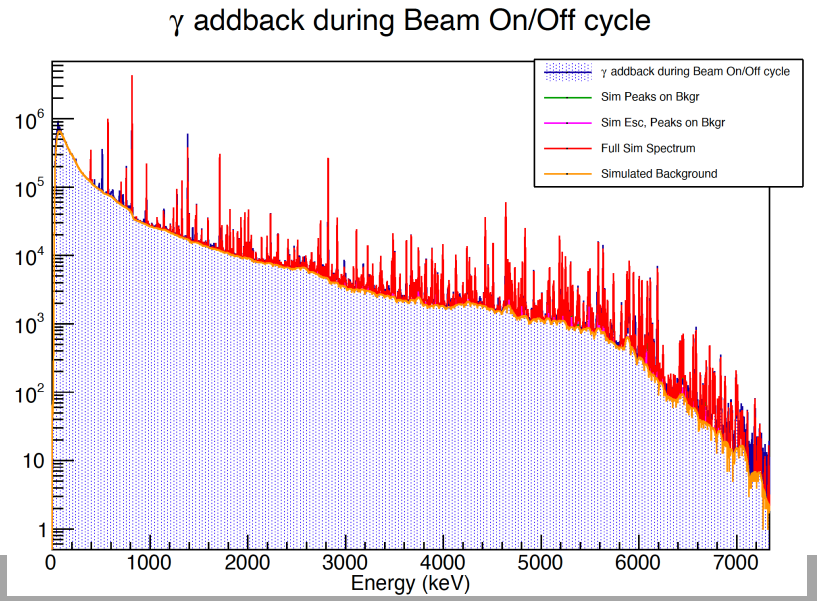
Coincident γ ray

Not Coincident γ ray

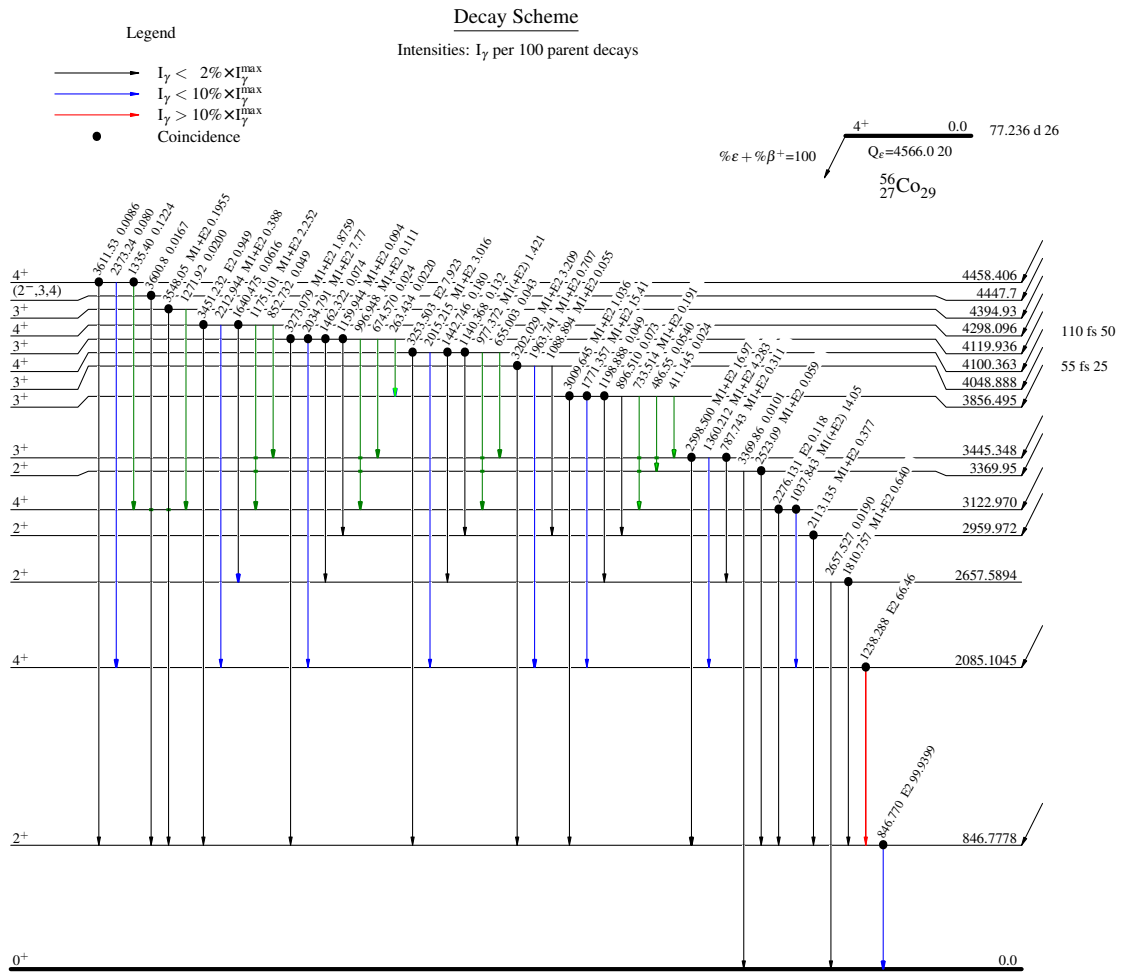
Simulation Tool

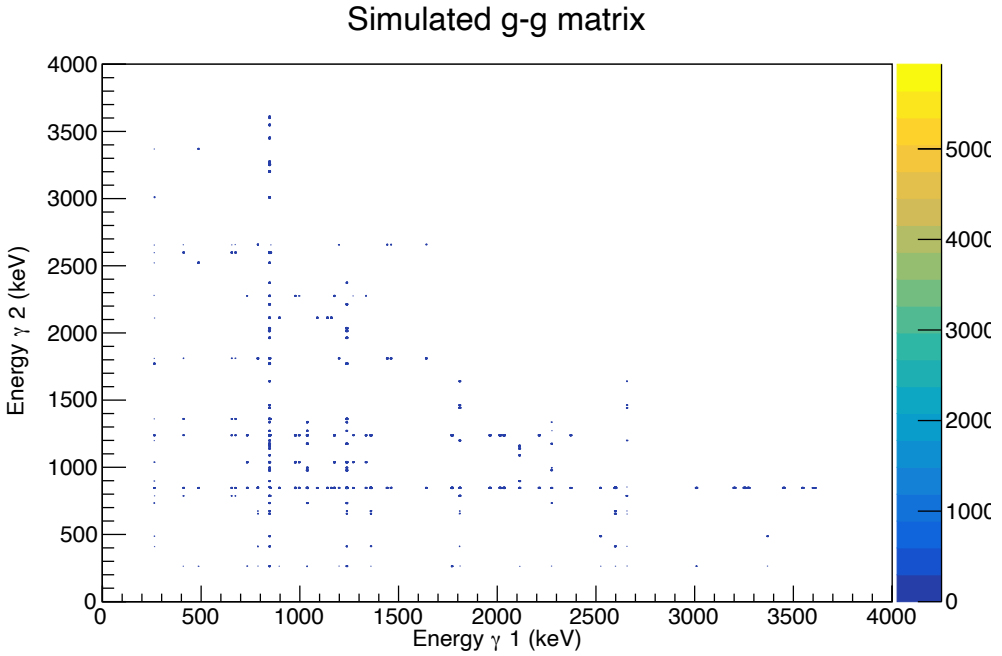
- Used to test your proposed decay scheme.
- Very basic!
  - But simple and quick!
  - Can produce singles and γ-γ coincidences
- Requires!
  - Transition information
    - $E_i, E_\gamma, E_f, I_\gamma, \delta I_\gamma$
  - Efficiency curve.
  - Peak Widths.
  - S.E.P intensities.
  - Experimental spectrum
- Limitations
  - Extracts crude background from experiment spectra.
  - No internal conversion.
  - No E0 transitions.
  - No summing corrections.
- Available on github.
  - https://github.com/Spagnole/SimpleSpectraSimulations
  - Example files provided.

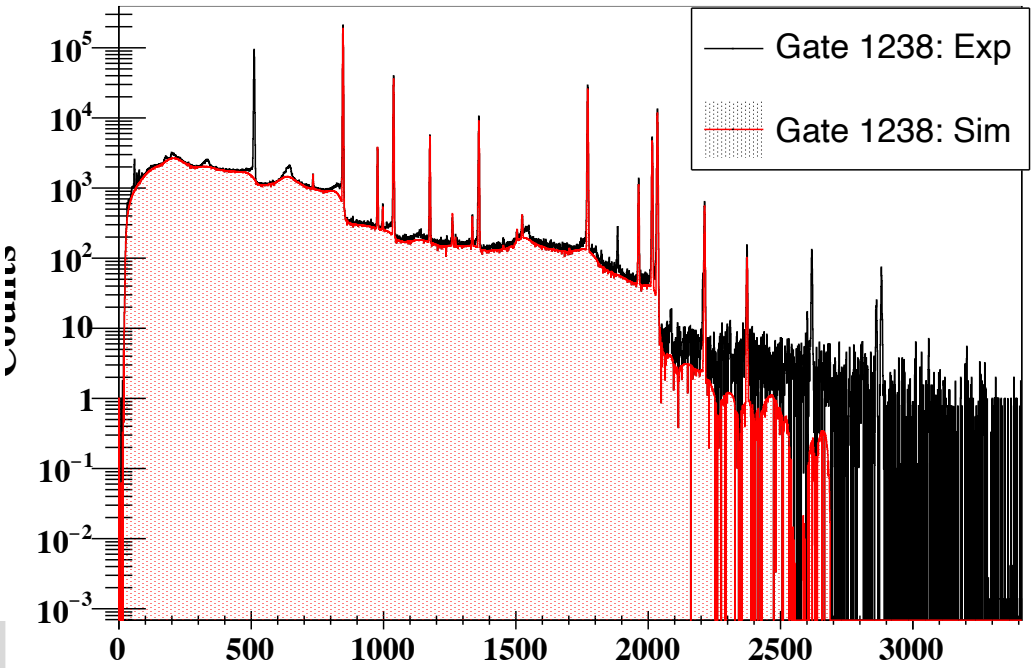




Simulated Coincidence ^{56}Co decay







ReadInputData.cc

- First of three codes!
  - SimulateSingles.cc and SimulateCoincidences.cc access this script!
- Functions
  - ReadDecayScheme(”decay-data.dat”)
    - Can enter multiple decay schemes!
  - GetEfficiency(“efficiencies.dat”)
  - GetPeakWidth(“peak_widths.dat”)
  - ReadEscapePeaks(“esc-peaks.dat”)
  - GetRealSpectra(TFile, TH1D)
  - GetSpectrumBackground(TH1D)

ReadDecayScheme("decay-data.dat")

- `vector<tuple<double, double, double, double, double, double >> AssignedTransition[NSources];`
  - $E_i, E_\gamma, E_f, I_\gamma, \delta I_\gamma, I_{E_i}, BR_\gamma$
- `Input file reads`
  - $E_i, E_\gamma, E_f, I_\gamma, \delta I_\gamma$
- I_{E_i} and BR_γ `are not read in but calculated later for coincidence sim.`
- Singles simulation
  - Only E_γ and I_γ are used!
- Coincidence Simulation
  - List must be ordered by E_i
  - E_i for the same level must match!
  - E_f needs to be close to correct level energy

```
void ReadDecayScheme(string filename = "TransitionList.dat", string enter_source_name = Form("source_%d",used_sources)){
    ifstream input( filename.c_str() );
    if( !input.is_open() ){
        cout << filename << " is not open!" << endl;
        return;
    }
    double a[5];
    input >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
    while( !input.eof() ){
        AssignedTransition[used_sources].push_back(make_tuple(a[0], a[1], a[2], a[3], 0, 0 ));
        if(PrintReadData) cout << a[0] << "\t" <<  a[1] << "\t" <<  a[2] << "\t" <<  a[3] << "\t" <<  a[4] << endl;
        input >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
    }
    used_sources++;
    source_name.push_back(enter_source_name);
}
```

```
0   0   0   0   0
344.2789    344.2785    0   87847224.5  878472.2
615.415 271.08  344.2789    236309.0    2363.1
755.396 411.1165    344.2789    7390587.0   73905.9
930.546 174.8   755.396 4568.1  45.7
930.546 315.1   615.415 131770.8    1317.7
930.546 586.2648    344.2789    1502187.5   15021.9
930.546 930.59  0   240701.4    2407.0
1047.81 703.55  344.2789    8784.7  87.8
1109.2  354.16  755.396 3162.5  31.6
1109.2  493.78  615.415 32239.9 322.4
1109.2  764.88  344.2789    625472.2    6254.7
1109.2  1109.18 0   622836.8    6228.4
1123.1855   192.6   930.546 22488.9 224.9
1123.1855   367.7891    755.396 2839222.3   28392.2
1123.1855   778.9045    344.2789    42711320.6  427113.2
1282.24 172.1   1109.2  1405.6  14.1
1282.24 351.66  930.546 35138.9 351.4
1282.24 526.88  755.396 39531.3 395.3
1314.64 970.22  344.2789    3953.1  39.5
1314.64 1314.6  0   6325.0  63.3
```

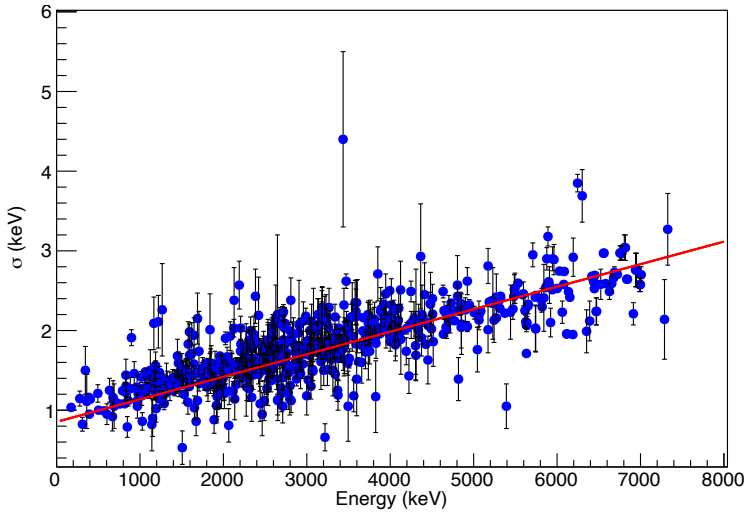
GetEfficiency("efficiencies.dat")

```
//The GetEfficiency() function is used to read in your gamma-ray efficiency
//and add produce a graph of efficiency as a function of energy
//this function reads in a two column text file
//the first column is energy in keV
//the second column is the efficiency at that energy
//This efficiency graph is used to correct intensities when filling the simulated spectra
void GetEfficiency(string eff_filename = "MyExpEffnew.dat"){
    ifstream myfitresult( eff_filename.c_str() );
    if( !myfitresult.is_open() ){
        cout << eff_filename << " is not open!" << endl;
        return;
    }
    double a[2];
    myfitresult >> a[0] >> a[1];
    while( !myfitresult.eof() ){
        gEff->SetPoint( gEff->GetN(), a[0], a[1]);
        myfitresult >> a[0] >> a[1];
    }
}
```

E_γ	Eff_γ
52	0.1097934
53	0.1112056
54	0.1125988
55	0.1139716
56	0.115321
57	0.1166456
58	0.1179442
59	0.1192162
60	0.1204598
61	0.1216748
62	0.1228612
63	0.1240182
64	0.1251456
65	0.126243
66	0.1273112
67	0.1283496
68	0.1293582
69	0.130338
70	0.1312882

GetPeakWidth("peak_widths.dat")

```
//the GetPeakWidth() function is used to set the peak width as a function of energy for your simulated spectra
//This file requires a four column text file, the input should be the following
//gamma-ray energy ---> g.-ray energy error ---> peak width ---> peak width error
//please use term sigma! NOT FWHM!!!!
//this function fills a graph with peak widths as a function of energy
//a linear function is then fit to the data
//this function is used to get the peak widths for the simulated spectra
void GetPeakWidth(string peak_widths_filename = "PeakWidths.dat"){

    gSigma->SetName("gSigma");
    gSigma->SetMarkerStyle(20);
    gSigma->SetMarkerColor(kBlue);

    ifstream input( peak_widths_filename.c_str() );
    if( !input.is_open() ){
        cout << peak_widths_filename << " is not open!" << endl;
        return;
    }
    double a[4];
    input >> a[0] >> a[1] >> a[2] >> a[3];
    while( !input.eof() ){
        if( a[2] > a[3] ){
            gSigma->SetPoint( gSigma->GetN(), a[0], a[2]);
            gSigma->SetPointError( gSigma->GetN()-1,  a[1], a[3]);
            if(PrintPeakWidths) cout << a[0] << "\t" << a[1] << "\t" << a[2] << "\t" << a[3] << endl;
        }
            input >> a[0] >> a[1] >> a[2] >> a[3];

    }
    fWidth = new TF1("fWidth","[0]+[1]*x",0,8000);
    fWidth->SetParameters(9.57477e-01, 2.59267e-04);
    gSigma->Draw("AP");
    gSigma->Fit("fWidth");
    fWidth->Draw("same");
}
```



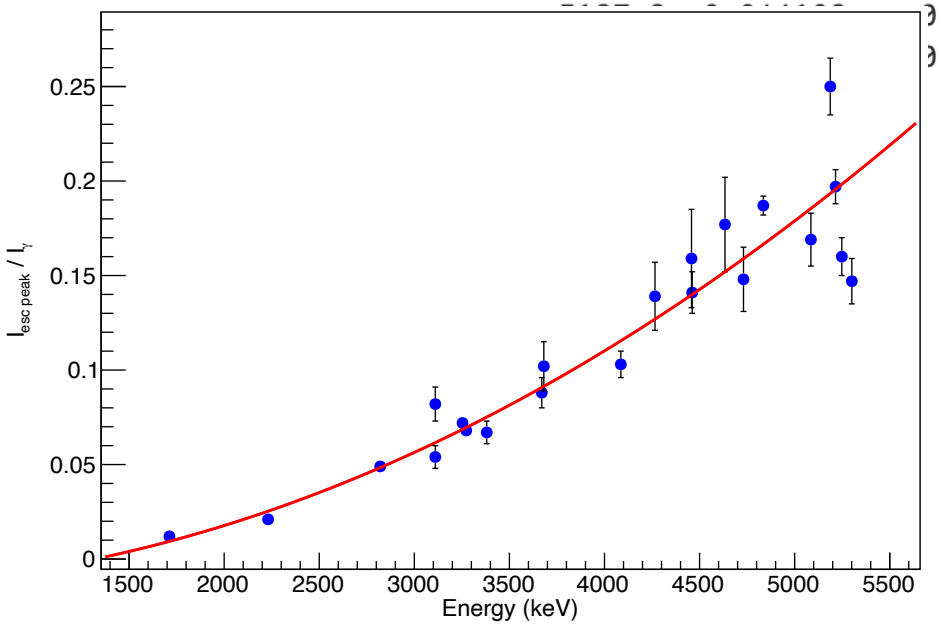
E_γ	δE_γ	σ_γ	$\delta\sigma_\gamma$
814.94	0.0032707	1.08079	0.00065
569.79	0.0014402	0.992	0.0024
858.81	0.086875	1.122	0.071
393.43	0.0059702	0.95	0.011
963.35	0.0038484	1.1141	0.0059
1778	0.018445	1.465	0.014
174.28	0.073384	1.035	0.051
278.69	0.096025	1.146	0.094
1137.7	0.041717	1.289	0.015
669.8	0.12943	0.92	0.18
1239.1	0.027019	1.278	0.014
310.13	0.1002	0.824	0.087
703.82	0.013942	1.104	0.027
1273.6	0.018352	1.197	0.024
361.98	0.35037	1.11	0.34
756.13	0.0097213	1.078	0.024
1326.1	0.020243	1.288	0.01
2140.5	0.032142	1.544	0.014
386.33	0.020216	1.161	0.014
1142.5	0.033478	0.82	0.33
1712.2	0.0067236	1.3348	0.0078

ReadEscapePeaks("esc-peaks.dat")

$$R = {}^{I_{S.E.P}}/_{I_{F.E.P}}$$

```
//the ReadEscapePeaks() function is used to get escape peak intensites relative to the full energy peak as a function of energy
//This file requires a four column text file, the input should be the following
//gamma-ray energy ---> g.-ray energy error ---> Esc.-Peak Int. / Full-Energy Peak Int  ---> uncertainty (Esc.-Peak Int. / Full-Energy Peak Int)
//this function is required for the simulation of escape peak intensities
void ReadEscapePeaks(string EscPeaksFilename = "EscapePeaks.dat"){

    gEscPeaks->SetName("gSigma");
    gEscPeaks->SetMarkerStyle(20);
    gEscPeaks->SetMarkerColor(kBlue);

    ifstream input( EscPeaksFilename.c_str() );
    if( !input.is_open() ){
        cout << EscPeaksFilename << " is not open!" << endl;
        return;
    }
    double a[4];
    input >> a[0] >> a[1] >> a[2] >> a[3];
    while( !input.eof() ){
        if( a[2] > a[3] ){
            gEscPeaks->SetPoint( gEscPeaks->GetN(), a[0], a[2]);
            gEscPeaks->SetPointError( gEscPeaks->GetN()-1,  a[1], a[3]);
            if(PrintEscPeakData) cout << a[0] << "\t" << a[1] << "\t" << a[2] << "\t" << a[3] << endl;
        }
            input >> a[0] >> a[1] >> a[2] >> a[3];
    }

    fEscPeak = new TF1("fEscPeak","[0]+[1]*x+[2]*x*x",0,8000);
    fEscPeak->SetParameters(-1.45859e-02, 1.11648e-06, 7.51546e-09);
    gEscPeaks->Draw("AP");
    gEscPeaks->Fit("fEscPeak");
    fEscPeak->Draw("same");
}
```

E_γ	δE_γ	R	δR
1712.2	0.023241	0.012	0.002
2231.1	0.020926	0.021	0.002
2820.7	0.039899	0.049	0.003
3110.2	0.045491	0.054	0.006
3110.3	0.069746	0.082	0.009
3670.3	0.031300	0.088	0.008
4265.7	0.066639	0.139	0.018
4731	0.066964	0.148	0.017
3681	0.083115	0.102	0.013
4633.8	0.105590	0.177	0.025
4457.6	0.055457	0.159	0.026
4835.3	0.030967	0.187	0.005
5085.6	0.069284	0.169	0.014
		0.250	0.015
		0.197	0.009



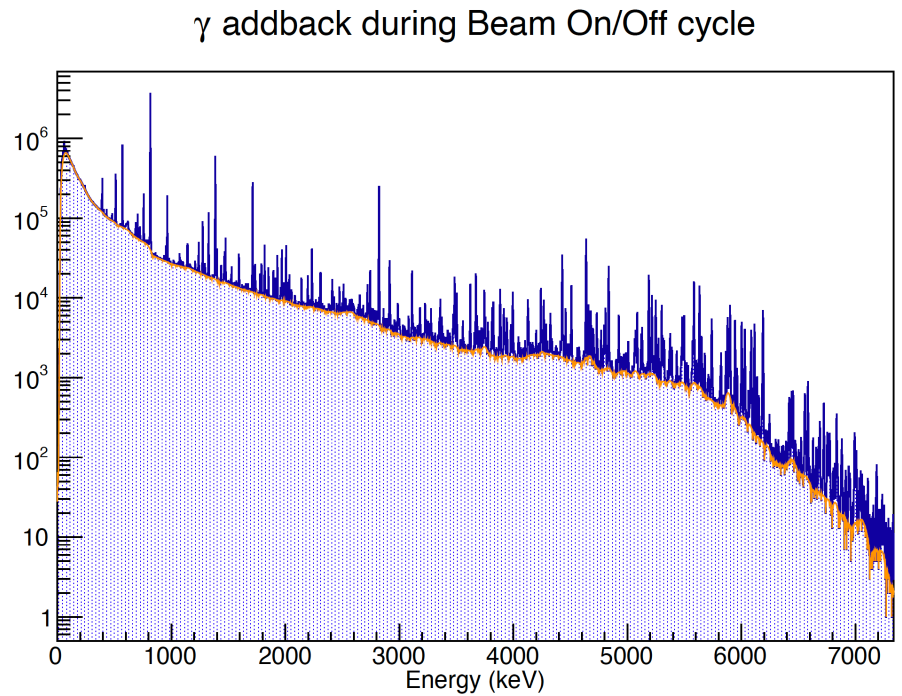
GetRealSpectra(TFile, TH1D)

```
//this function is used to get your real experimental spectra
//user should provide the name of the root file and the name of the spectrum
void GetRealSpectra(string rootfilename = "ExampleFile.root", string RealHistName = "hgE_56Co"){

    //open root file
    if(gSystem->AccessPathName(rootfilename.c_str())){ //check to see root file exists!
        cout << "The file " << rootfilename << " doesn't exists\n";
        return;
    }
    else fRealData = TFile::Open( rootfilename.c_str() ); //opening root file

    if( !fRealData->GetListOfKeys()->Contains( RealHistName.c_str() )){ //check to see histogram exists!
        cout << "The histogram " << RealHistName << " doesn't exists\n";
        return;
    }
    else{ //getting root spectrum
        hRealSpectra = (TH1D*)fRealData->Get( RealHistName.c_str() );
        hRealSpectra->Draw("hist");
        hRealSpectra->GetXaxis()->SetTitle("Energy (keV)");
        hRealSpectra->GetXaxis()->CenterTitle();
    }
}
```

GetSpectrumBackground(TH1D)

```
//this function is used to extract a background from your experimntal spectrum
//the default values of the function will extract a rather crude background
//the user is free to play with these parameters to try and improve the extracted background
void GetSpectrumBackground(TH1D *h, int iterations = 50, int decreasewindow = 1, int backorder = 2, bool smoothing = false, int smoothwindow = 3, bool compton = 0){

    //extract binning information from the experimental spectrum
    //we want to use the same binning to compare
    const int Nbins = h->GetXaxis()->GetNbins();
    double x_low = h->GetXaxis()->GetBinLowEdge(1);
    double x_max = h->GetXaxis()->GetBinUpEdge(Nbins);
    double source[Nbins];

    hBkgr = new TH1D("hBkgr","Simulated Background",Nbins,x_low,x_max);

    TSpectrum *s = new TSpectrum();
    for (int i = 0; i < Nbins; i++) source[i]=h->GetBinContent(i + 1);
    //s->Background(source,Nbins,75,TSpectrum::kBackDecreasingWindow, TSpectrum::kBackOrder2,kFALSE, TSpectrum::kBackSmoothing3,kFALSE);
    s->Background(source,Nbins,iterations,decreasewindow,backorder,smoothing,smoothwindow,compton);
    for (int i = 0; i < Nbins; i++) hBkgr->SetBinContent(i + 1,source[i]);
    h->Draw("hist");
    hBkgr->SetLineColor(kOrange+1);
    hBkgr->Draw("SAME L");

}
```

Built from simple root tutorial Background_Compton.C
User can play with the inputs of
s->Background() to get the best result possible
Default options work pretty well.



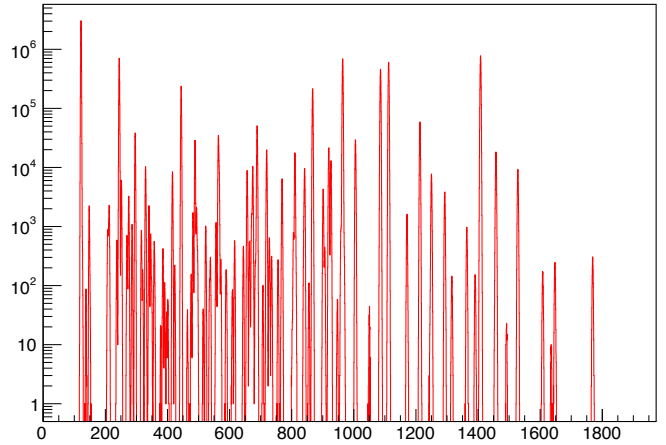
SimulateSingles.cc

- Used to simulate singles spectrum!
- Only has three functions.
- `FillSimulation(int source_number, double scale_int = 1.0)`
  - Fills the simulated peak intensities.
- `FillEscapePeaks(int source_number, double scale_int = 1.0)`
  - Fills the simulated single escape peaks.
- `BuildSimuledSpectra()`
  - Adds the histograms together.

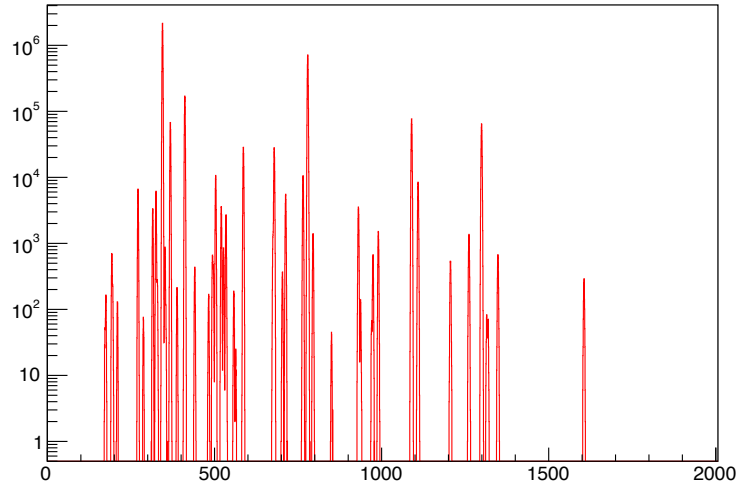
FillSimulation(int source_number, double scale_int = 1.0)

```
//this function fills the simulated peak intensities
//this function requires the output from the ReadDecayScheme(), GetEfficiency() and GetPeakWidth()
//scaling paramter is used to fix for using relative efficiency and/or relative intensities
//for absolute efficiency and absolute intensities just use the default value
void FillSimulation(int source_number, double scale_int = 1.0){

    if( AssignedTransition[source_number].size() == 0 ){
        cout << "No data in decay list!\nHave you read in list of gamma-rays?" << endl;
        return;
    }

    const int Nbins = hBkgr->GetXaxis()->GetNbins();
    double x_low = hBkgr->GetXaxis()->GetBinLowEdge(1);
    double x_max = hBkgr->GetXaxis()->GetBinUpEdge(Nbins);
    hSimPeaks[source_number] = new TH1D(Form("hPeaks_%s", source_name.at(source_number).c_str()),
                                Form("Simulated Source Peaks: %s",  source_name.at(source_number).c_str() ),
                                Nbins, x_low, x_max);
    double peak_int;
    double energy;
    double width;
    for(int i = 0; i < AssignedTransition[source_number].size(); i++){
        energy = get<1>(AssignedTransition[source_number].at(i));   //get gamma-ray energy
        peak_int = get<3>(AssignedTransition[source_number].at(i)) * gEff->Eval(energy) * scale_int;    //calculate peak intensity
        width = fWidth->Eval(energy);   //get peak width
        for(int j = 0; j < peak_int; j++){
            hSimPeaks[source_number]->Fill( gRandom->Gaus(energy,width) );  //fill spectrum using simple gaussian distribution
        }
    }
    hSimPeaks[source_number]->SetLineColor(kRed);
}
```

Simulated Source Peaks: 152Sm

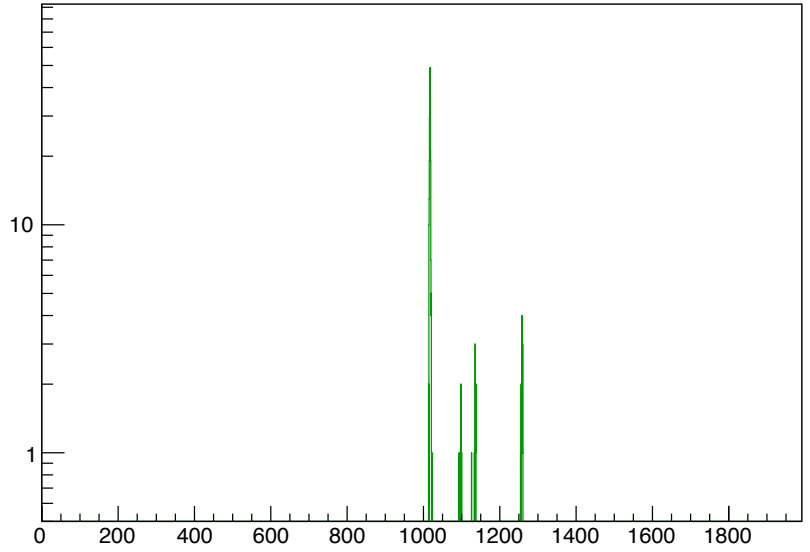


Simulated Source Peaks: 152Gd



FillEscapePeaks(int source_number, double scale_int = 1.0)

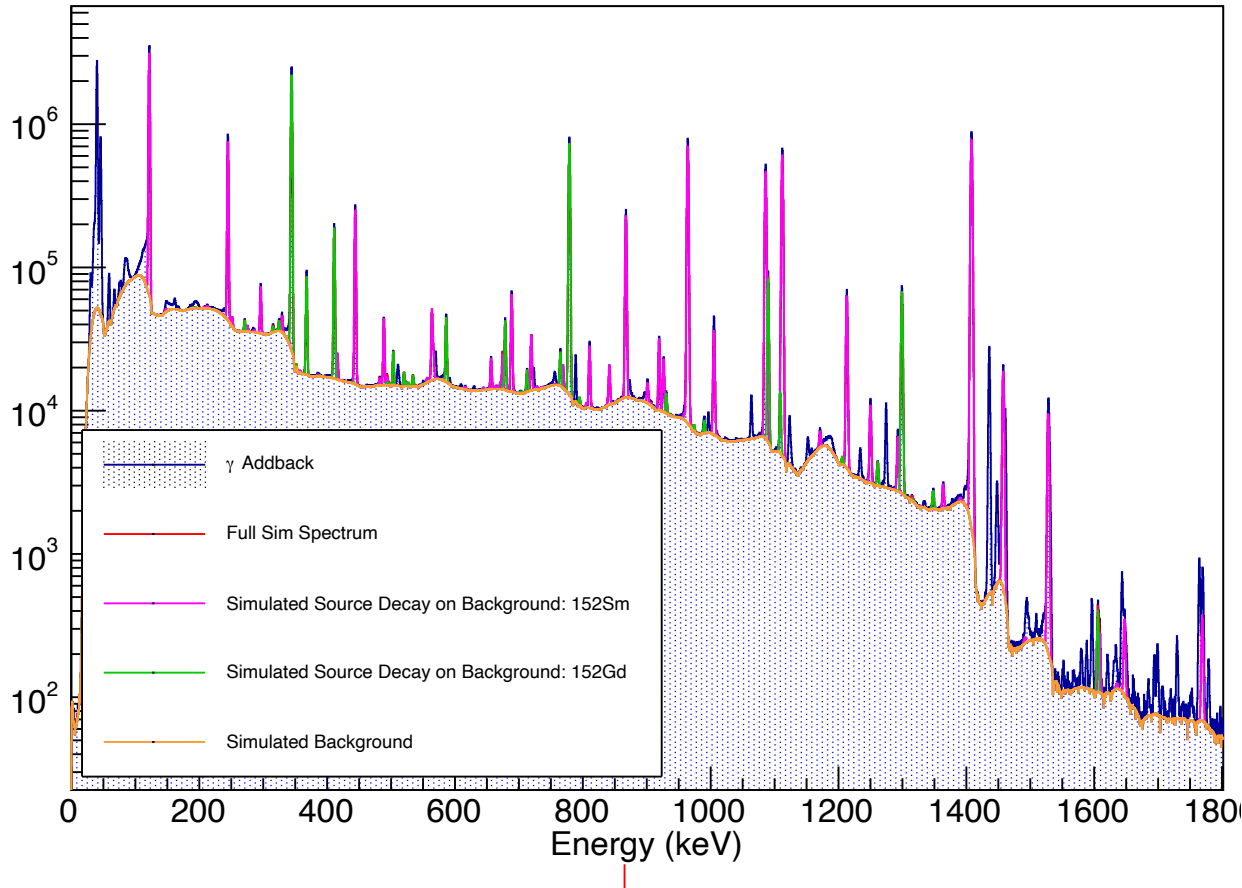
Simulated Source Single-Escape Peaks: 152Sm




```
//this function fills the simulated single-escape peak intensities
//this function requires the output from the ReadEscapePeaks() function
//!!!USE SAME SCALING THAT WAS USED BY FillSimulation()
void FillEscapePeaks(int source_number, double scale_int = 1.0){

    const int Nbins = hBkgr->GetXaxis()->GetNbins();
    double x_low = hBkgr->GetXaxis()->GetBinLowEdge(1);
    double x_max = hBkgr->GetXaxis()->GetBinUpEdge(Nbins);
    hEscPeaks[source_number] = new TH1D(Form("hEscPeaks_%s", source_name.at(source_number).c_str()),
                                    Form("Simulated Source Single-Escape Peaks: %s",  source_name.at(source_number).c_str() ),
                                        Nbins, x_low, x_max);
    hEscPeaks[source_number]->SetLineColor(kGreen+2);

    double peak_int;
    double energy;
    double width;
    for(int i = 0; i < AssignedTransition[source_number].size(); i++){
        if( get<1>(AssignedTransition[source_number].at(i)) < 1500. ) continue;
        energy = get<1>(AssignedTransition[source_number].at(i));   //get gamma-ray energy
        peak_int = get<3>(AssignedTransition[source_number].at(i))*gEff->Eval(energy)*fEscPeak->Eval(energy) * scale_int;   //calculate escape-peak intensity
        width = 1.1*fWidth->Eval(energy);   //get peak width, S.E.P width is increased by 10 %
        for(int j = 0; j < peak_int; j++){
            hEscPeaks[source_number]->Fill( gRandom->Gaus(energy-511.,width) ); //fill spectrum using simple gaussian distribution
        }
    }
}
```

BuildSimuledSpectra()

```
//this function is used to produce the full simulated spectrum
//this function reads in the simulated Full energy peaks, single escape peaks and the background extracted from the real spectrum
//this function creates three new spectra
//One adds the full-energy peaks, single esc, peaks and the background, this is the full simulated spectrum
//Another adds the full energy peaks to the background
//The third adds the escape peaks to the background
void BuildSimuledSpectra(){


    const int Nbins = hBkgr->GetXaxis()->GetNbins();
    double x_low = hBkgr->GetXaxis()->GetBinLowEdge(1);
    double x_max = hBkgr->GetXaxis()->GetBinUpEdge(Nbins);

    hFullSim = new TH1D("hFullSim","Full Sim Spectrum",Nbins, x_low, x_max);
    hFullSim->SetLineColor(kRed);
    hFullSim->Add(hBkgr);

    int hist_colors[] = {6, 417, 1, 900-4, 432, 801, 880, 861, 625, 416};
    for(int i = 0; i < used_sources; i++){
        cout << i << endl;
        hSimSource[i] = new TH1D( Form("hSim_%s", source_name.at(i).c_str() ),
                                  Form("Simulated Source Decay on Background: %s",  source_name.at(i).c_str() ),
                                  Nbins, x_low, x_max);
        hSimSource[i]->SetLineColor( hist_colors[i] );
        hSimSource[i]->Add(hBkgr);
        hSimSource[i]->Add(hSimPeaks[i]);
        hSimSource[i]->Add(hEscPeaks[i]);
        hFullSim->Add(hSimPeaks[i]);
        hFullSim->Add(hEscPeaks[i]);
    }


    hRealSpectra->SetFillColor(kBlue);
    hRealSpectra->SetFillStyle(3003);
    hRealSpectra->Draw("hist");
    hFullSim->Draw("histsame");
    for(int i = 0; i < used_sources; i++){
        hSimSource[i]->Draw("histsame");
    }
    hBkgr->Draw("histsame");
}
```



SimulateCoincidences.cc

- `GetLevelList(int source_number)`
- `FixTransitionList(int source_number)`
- `CalcLevelFeedingAndGammaBR(int source_number)`
- `FindCoincidences(int source_number)`
- `ReduceGammaGammaList(int source_number)`
- `FillCoincMatrix(int source_number)`
- `GateOnSimMat(int source_number, int low, int up)`
- `FillEscPeakSpec(int source_number)`
- `BuildSimuledSpectra()`

GetLevelList(int source_number)

```
//this function gets a list of levels from the 'AssignedTransition' vector from the ReadInputData.cc file
//this function requires that ReadDecayScheme() function has been utilized
void GetLevelList(int source_number){

        if( AssignedTransition[source_number].size() == 0 ){
            cout << "No data in decay list!\nHave you read in list of gamma-rays?" << endl;
            return;
        }
        double prev_energy = -1;
        int counter = 0;
        for(int i = 0; i < AssignedTransition[source_number].size(); i++){
            if( get<0>(AssignedTransition[source_number].at(i)) != prev_energy ){
                LevelEnergy[source_number].push_back( make_tuple(counter, get<0>(AssignedTransition[source_number].at(i)), 0) );
                counter++;
            }
            prev_energy = get<0>(AssignedTransition[source_number].at(i));
        }
        for(int i = 0; i < LevelEnergy[source_number].size(); i++){
            if(PrintLevelList) cout << get<0>(LevelEnergy[source_number].at(i)) << "\t" << get<1>(LevelEnergy[source_number].at(i)) <<endl;
        }
}
```

FixTransitionList(int source_number)

```
//this function is used to correct the 'AssignedTransition' list
//this correction is required if your final level energies do not exactly match the level energies in the 'LevelEnergy' vector
void FixTransitionList(int source_number){

        if( AssignedTransition[source_number].size() == 0 ){
            cout << "No data in decay list!\nHave you read in list of gamma-rays?" << endl;
            return;
        }

        double min_diff;
        double difference;
        double new_level_energy;
        for(int i = 0; i <  AssignedTransition[source_number].size(); i++){
            min_diff = 10000;
            for(int j = 0; j < LevelEnergy[source_number].size(); j++){
                difference = TMath::Abs( get<2>(AssignedTransition[source_number].at(i)) - get<1>(LevelEnergy[source_number].at(j)) );
                if( difference < min_diff ){
                    new_level_energy = get<1>(LevelEnergy[source_number].at(j));
                    min_diff = difference;
                }
            }
            get<2>(AssignedTransition[source_number].at(i)) = new_level_energy;
        }

}
```

CalcLevelFeedingAndGammaBR(int source_number)

```
//this function calculates the level population of each state and the gamma-ray branching ratio of each transition
//this is function is required to properly calculate the expected gamma-gamma coincidence intensity
void CalcLevelFeedingAndGammaBR(int source_number){

        double sum_temp;
        for(int i = 0; i < LevelEnergy[source_number].size(); i++){
            sum_temp = 0;
            for(int j = 0; j <  AssignedTransition[source_number].size(); j++){
                if(  get<1>(LevelEnergy[source_number].at(i)) == get<0>(AssignedTransition[source_number].at(j))){
                    sum_temp += get<3>(AssignedTransition[source_number].at(j));
                }
            }
            get<2>(LevelEnergy[source_number].at(i)) = sum_temp;
        }

        //calculating gamma-ray branching ratios
        for(int i = 0; i <  AssignedTransition[source_number].size(); i++){
            if( i == 0 ) continue;
            for(int j = 0; j < LevelEnergy[source_number].size(); j++){
                if(  get<1>(LevelEnergy[source_number].at(j)) == get<0>(AssignedTransition[source_number].at(i))){
                    get<4>(AssignedTransition[source_number].at(i)) = get<2>(LevelEnergy[source_number].at(j));
                    get<5>(AssignedTransition[source_number].at(i)) = get<3>(AssignedTransition[source_number].at(i)) / get<2>(LevelEnergy[source_number].at(j));
                }
            }
        }

        //calculate level population
        for(int i = 0; i <  AssignedTransition[source_number].size(); i++){
            if( i == 0 ) continue;
            for(int j = 0; j < AssignedTransition[source_number].size(); j++){
                if(  get<2>(AssignedTransition[source_number].at(j)) == get<0>(AssignedTransition[source_number].at(i))){
                    get<4>(AssignedTransition[source_number].at(i)) = get<4>(AssignedTransition[source_number].at(i)) - get<3>(AssignedTransition[source_number].at(j));
                }
            }
        }
}
```

FindCoincidences(int source_number)

```
//this function calculates the expected number of gamma-gamma coincidences
//this function uses a large number of nested loops (Should be replaced by some recursive loop?)
//if a gamma-gamma coincidence is separated by about 5 intermediate gamma-rays it will not be added to the list of coincidences
//if you require such coincidences ---> Add more nested loops
void FindCoincidences(int source_number){

  if( AssignedTransition[source_number].size() == 0 ){
    cout << "No data in decay list!\nHave you read in list of gamma-rays?" << endl;
    return;
  }

  double NDecays, Ngg[10];

  gg_coinc[source_number].clear();

  for(int i = AssignedTransition[source_number].size()-1; i >= 0 ; i--){
    if( get<2>(AssignedTransition[source_number].at(i))  == 0 ) continue;
    NDecays = get<4>(AssignedTransition[source_number].at(i))*get<5>(AssignedTransition[source_number].at(i));
    for(int j = i-1; j >= 0 ; j--){
      if( get<2>(AssignedTransition[source_number].at(i)) == get<0>(AssignedTransition[source_number].at(j))){
        Ngg[0] = NDecays*get<5>(AssignedTransition[source_number].at(j));
        AddToCoincidenceList(source_number,{i},j, Ngg[0] );
        if( get<2>(AssignedTransition[source_number].at(j))  == 0 ) continue;
        for(int k = j-1; k >= 0 ; k--){
          if( get<2>(AssignedTransition[source_number].at(j)) == get<0>(AssignedTransition[source_number].at(k))){
            Ngg[1] = Ngg[0]*get<5>(AssignedTransition[source_number].at(k));
            AddToCoincidenceList(source_number,{i,j},k, Ngg[1] );
            if( get<2>(AssignedTransition[source_number].at(k))  == 0 ) continue;
            for(int l = k-1; l >= 0 ; l--){
              if( get<2>(AssignedTransition[source_number].at(k)) == get<0>(AssignedTransition[source_number].at(l))){
                Ngg[2] = Ngg[1]*get<5>(AssignedTransition[source_number].at(l));
                AddToCoincidenceList(source_number,{i,j,k},l, Ngg[2] );
                if( get<2>(AssignedTransition[source_number].at(l))  == 0 ) continue;
                for(int m = l-1; m >= 0 ; m--){
                  if( get<2>(AssignedTransition[source_number].at(l)) == get<0>(AssignedTransition[source_number].at(m))){
                    Ngg[3] = Ngg[2]*get<5>(AssignedTransition[source_number].at(m));
                    AddToCoincidenceList(source_number,{i,j,k,l},m, Ngg[3] );
                    if( get<2>(AssignedTransition[source_number].at(m))  == 0 ) continue;
                    for(int n = m-1; n >= 0 ; n--){
                      if( get<2>(AssignedTransition[source_number].at(m)) == get<0>(AssignedTransition[source_number].at(n))){
                        Ngg[4] = Ngg[3]*get<5>(AssignedTransition[source_number].at(n));
                        AddToCoincidenceList(source_number,{i,j,k,l,m},n, Ngg[4] );
                        if( get<2>(AssignedTransition[source_number].at(n))  == 0 ) continue;
                        for(int p = n-1; p >= 0 ; p--){
                          if( get<2>(AssignedTransition[source_number].at(n)) == get<0>(AssignedTransition[source_number].at(p))){
                            Ngg[5] = Ngg[4]*get<5>(AssignedTransition[source_number].at(p));
                            AddToCoincidenceList(source_number,{i,j,k,l,m,n},p, Ngg[5] );
                            if( get<2>(AssignedTransition[source_number].at(p)) !=0) PrintUnfinishedCascade(source_number,{i,j,k,l,m,n});
                          }
                        }
                      }
```

```
void AddToCoincidenceList(int source_number, vector<int> feeder_index, int decay_index, double intensity ){

  double decay_energy = get<1>(AssignedTransition[source_number].at(decay_index));
  for(int i = 0; i < feeder_index.size(); i++){
    double feeder_energy = get<1>(AssignedTransition[source_number].at( feeder_index.at(i) ) );
    gg_coinc[source_number].push_back( make_tuple(feeder_energy,decay_energy, intensity) );
  }
}
```

Uses a number of nested loops!
Should be replaced using recursion?
Cascades should end at a final level energy 0.
If not, unfinished cascade is printed to terminal.
If required, user can add more nested loops.

ReduceGammaGammaList(int source_number)

```
//this function is used to reduce the size of the gamma-gamma coincidence list
//if a given coincidence is given a number of times, this function will reduce the entries of that coincidence to one instance but adds all of the intensities
//strictly speaking this function is not absolutely required for the tool to function! BUT YOU SHOULD STILL USE IT!
void ReduceGammaGammaList(int source_number){

    for(int i = 0; i < gg_coinc[source_number].size();i++){
        for(int j = 0; j < gg_coinc[source_number].size(); j++){
            if(i==j) continue;
            if( get<0>(gg_coinc[source_number].at(i)) == get<0>(gg_coinc[source_number].at(j)) && get<1>(gg_coinc[source_number].at(i)) == get<1>(gg_coinc[source_number].at(j)) ){
                get<2>(gg_coinc[source_number].at(i)) =  get<2>(gg_coinc[source_number].at(i)) + get<2>(gg_coinc[source_number].at(j));
                gg_coinc[source_number].erase(gg_coinc[source_number].begin()+j);
                j=j-1;
            }
        }
    }
}
```

A given coincidence a,b can be added to the coincidence list multiple times if the coincidence is produced from different initially populated levels.

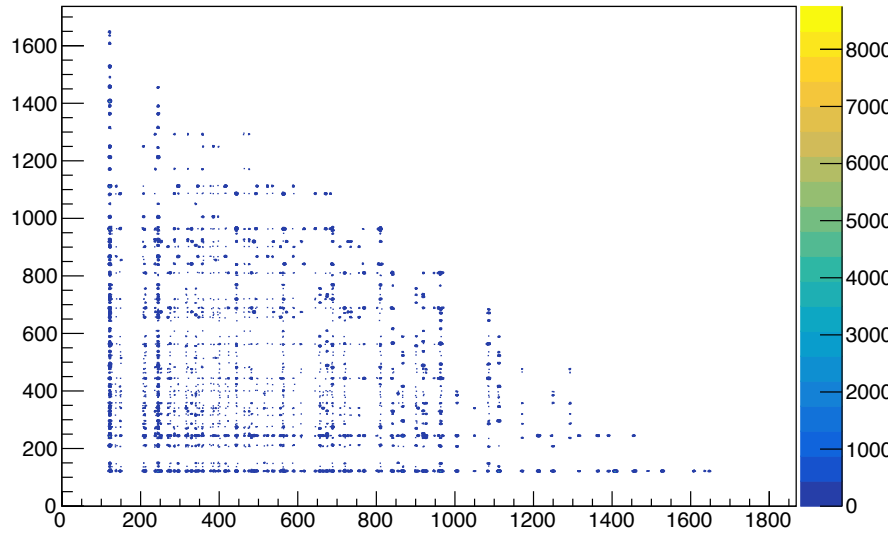
e.g. $c \rightarrow b \rightarrow a$, $d \rightarrow b \rightarrow a$

This function sums the intensities of all occurances of a,b to a single value

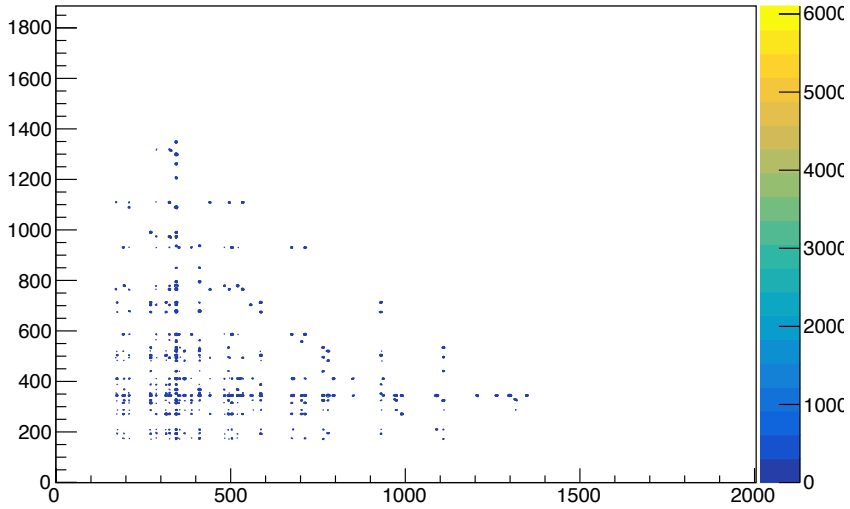
FillCoincMatrix(int source_number)

```
//this function fills a simulated gamma-gamma coincidence matrix
//user can provide specified binning of the matrix
//this function requires that the list of coincidences has been produced
//as well as reading in the gamma-ray efficiency and peak widths
void FillCoincMatrix(int source_number, int NBins = -1, double low = -1, double upp = -1){

    if( NBins == -1){
        NBins = hRealSpectra->GetXaxis()->GetNbins();
        low = hRealSpectra->GetXaxis()->GetBinLowEdge(1);
        upp = hRealSpectra->GetXaxis()->GetBinUpEdge(NBins);
    }

    sim_gg_mat[source_number] = new TH2D(Form("sim_gg_mat_%s", source_name.at(source_number).c_str() ),
                                Form("Simulated g-g matrix Source: %s",source_name.at(source_number).c_str() ),
                                NBins,low,upp,NBins,low,upp);
    double gamma1,gamma2,sigma1,sigma2;
    double counts;
    for(int i = 0; i < gg_coinc[source_number].size(); i++){
        gamma1 = get<0>(gg_coinc[source_number].at(i));
        gamma2 = get<1>(gg_coinc[source_number].at(i));
        counts = (14./15) * get<2>(gg_coinc[source_number].at(i)) * gEff->Eval(gamma1) * gEff->Eval(gamma2); //user may require
        sigma1 = fWidth->Eval(gamma1);
        sigma2 = fWidth->Eval(gamma2);
        for(int j = 0; j < counts; j++){
            sim_gg_mat[source_number]->Fill( gRandom->Gaus(gamma1,sigma1), gRandom->Gaus(gamma2,sigma2));
            sim_gg_mat[source_number]->Fill( gRandom->Gaus(gamma2,sigma2), gRandom->Gaus(gamma1,sigma1));
        }
    }

}
```

Simulated g-g matrix Source: 152Sm

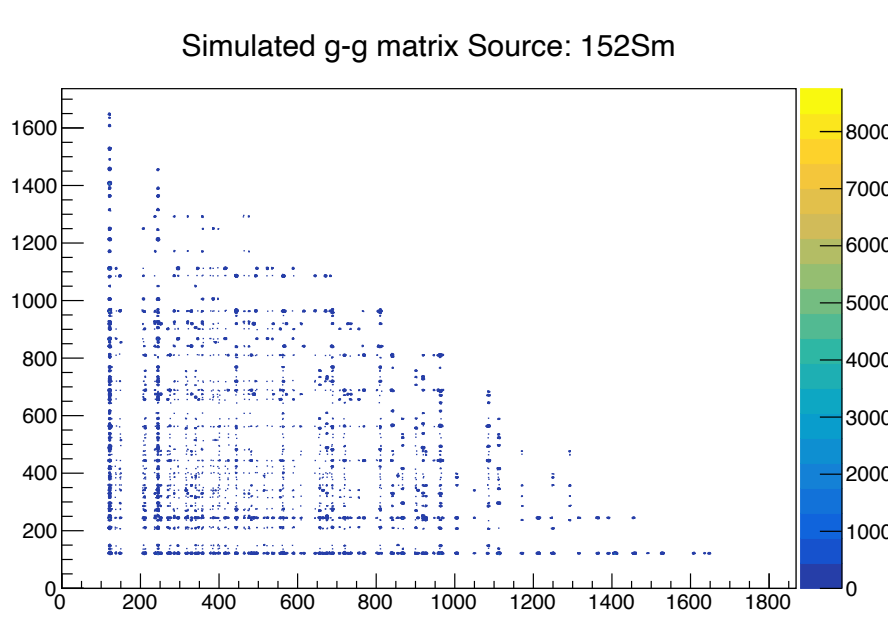


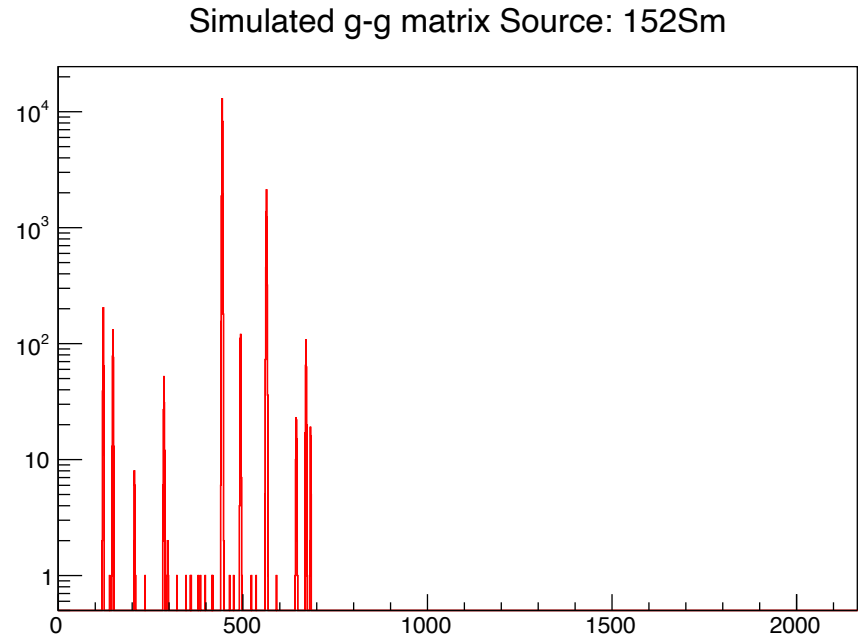
Simulated g-g matrix Source: 152Gd



GateOnSimMat(int source_number, int low, int up)

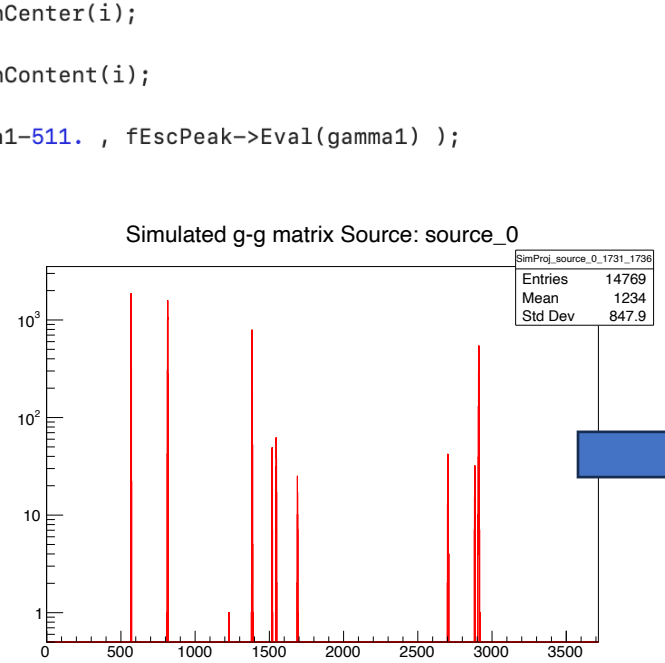
```
//this function gates on the simulated gamma-gamma matrix to produce a coincidence spectrum
//used should use the same gate they used for their experimental efficiency curve
void GateOnSimMat(int source_number, int low, int up){

    hSimPeaks[source_number]  = sim_gg_mat[source_number]->ProjectionY(Form("SimProj_%s_%d_%d",
                                source_name.at(source_number).c_str(),low,up),low,up);
    hSimPeaks[source_number]->Draw("hist");
    hSimPeaks[source_number]->SetLineColor(kRed);
    hSimPeaks[source_number]->SetFillColor(kRed);
    hSimPeaks[source_number]->SetFillStyle(3000);

}
```

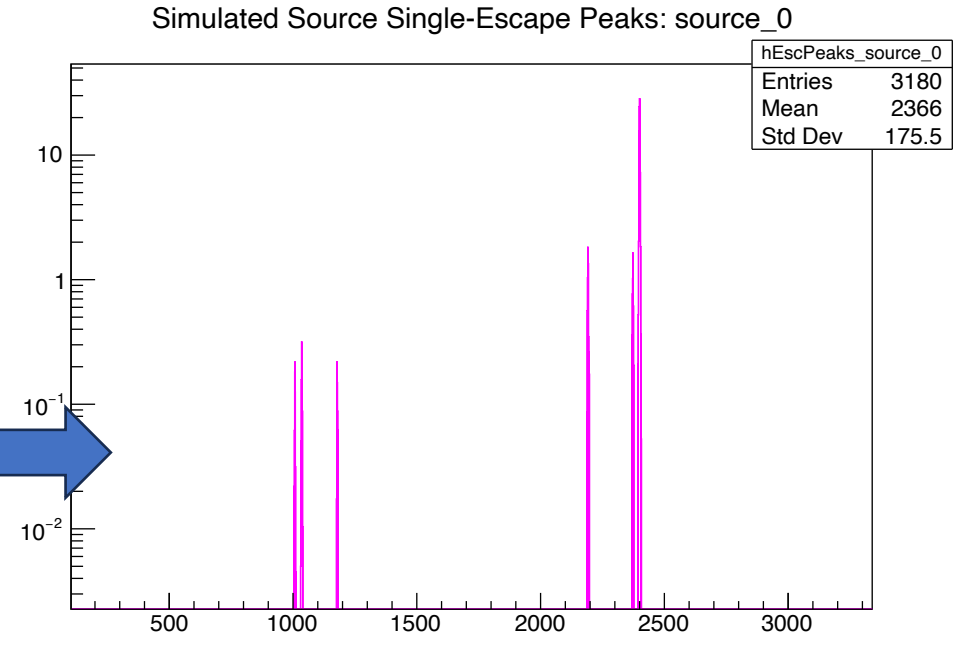




FillEscPeakSpec(int source_number)

```
//this function fills the simulated escape peaks
//this is done by using the output TH1D from the GateOnSimMat() function
void FillEscPeakSpec(int source_number){

    int Nbins = hSimPeaks[source_number]->GetXaxis()->GetNbins();
    double low = hSimPeaks[source_number]->GetXaxis()->GetBinLowEdge(1);
    double upp = hSimPeaks[source_number]->GetXaxis()->GetBinUpEdge(Nbins);

    hEscPeaks[source_number] = new TH1D(Form("hEscPeaks_%s", source_name.at(source_number).c_str()),
            Form("Simulated Source Single-Escape Peaks: %s",  source_name.at(source_number).c_str() ),Nbins, low, upp);
    hEscPeaks[source_number]->SetLineColor(6);
    hEscPeaks[source_number]->SetFillColor(6);
    hEscPeaks[source_number]->SetFillStyle(3000);
    double gamma1,sigma1;
    double counts, scale;

    for(int i = 1; i <= Nbins; i++){
        gamma1 = hSimPeaks[source_number]->GetBinCenter(i);
        if(gamma1 < 1500. ) continue;
        counts = hSimPeaks[source_number]->GetBinContent(i);
        for(int j = 0; j < counts; j++){
            hEscPeaks[source_number]->Fill( gamma1-511. , fEscPeak->Eval(gamma1) );
        }
    }
}
```



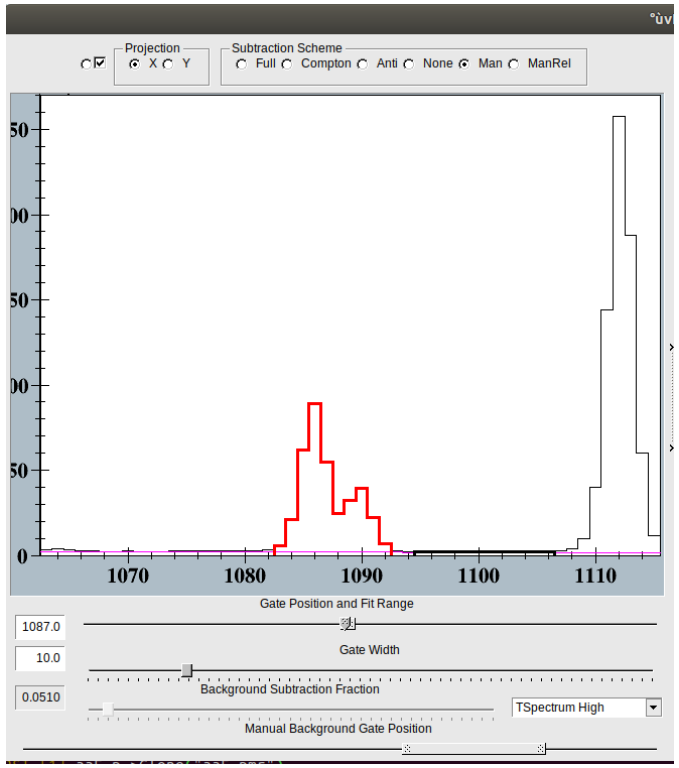


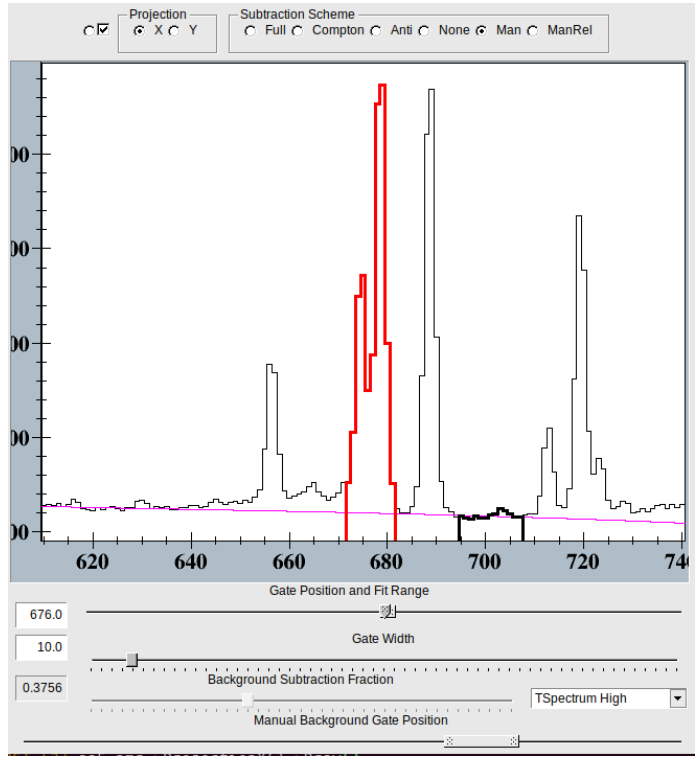
BuildSimuledSpectra()

- The same function as in SimulateSingles.cc
- This should probably be moved to avoid duplication.

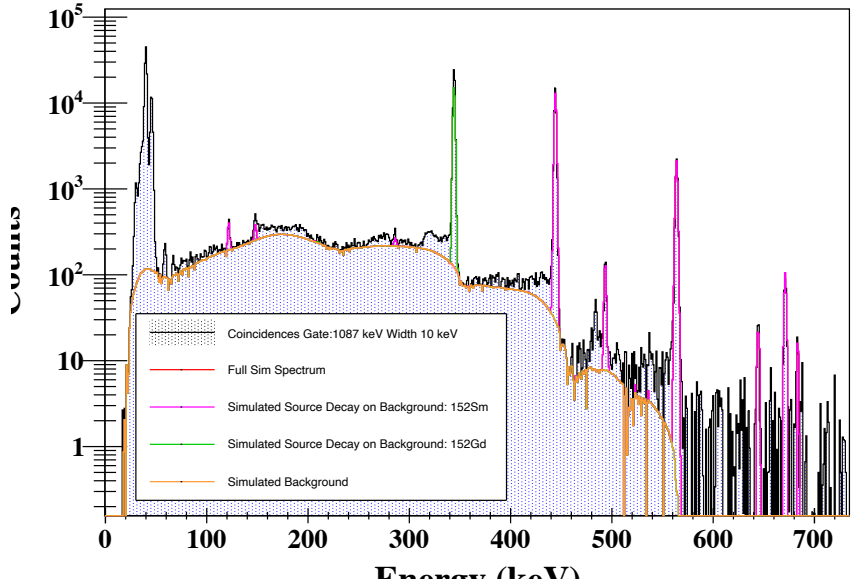
RunSimulateCoincidenceExample.cc

```
void RunSim_Gate1087(){

    ReadDecayScheme("example-152Eu/152Sm.dat","152Sm");
    ReadDecayScheme("example-152Eu/152Gd.dat","152Gd");
    GetEfficiency("MyExpEffnew.dat");
    GetPeakWidth("PeakWidths.dat");
    ReadEscapePeaks("EscapePeaks.dat");
    GetRealSpectra("ExampleFile.root","hggE_Gate1087");
    GetSpectrumBackground(hRealSpectra,50);

    for(int i = 0; i < used_sources; i++){
        GetLevelList(i);
        FixTransitionList(i);
        CalcLevelFeedingAndGammaBR(i);
        FindCoincidences(i);
        ReduceGammaGammaList(i);
        FillCoincMatrix(i);
        GateOnSimMat(i, 1087-5,1087+4);
        FillEscPeakSpec(i);
    }
    BuildSimuledSpectra();

}
```

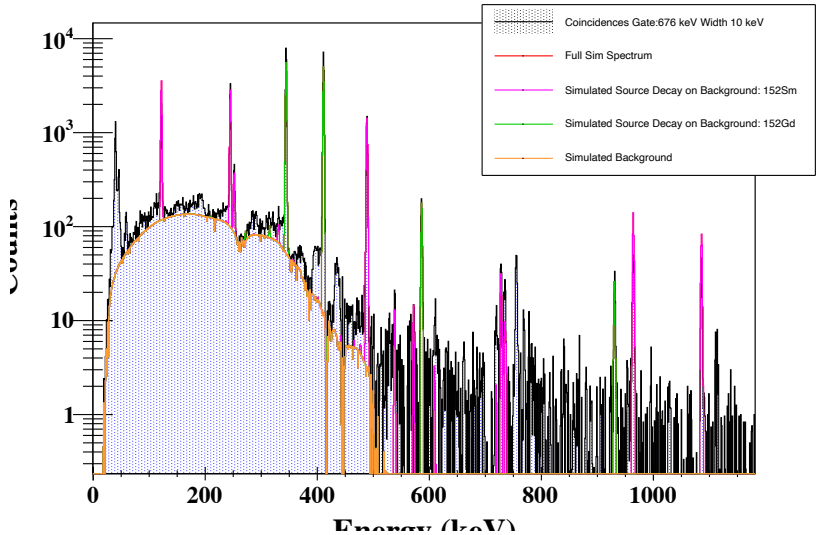




Coincidences Gate:1087 keV Width 10 keV



Coincidences Gate:676 keV Width 10 keV



Summary

- This is a simple tool for simulating gamma-ray singles and gated coincidence spectra.
- It can be used to test your proposed decay schemes.
  - Do you have missing intensities or too much?
  - Do you have missing coincidences or false coincidences?
- Available on github!
  - https://github.com/Spagnole/SpectraSimulationsMultipleSources
  - Example using 152Eu decay data.
- Things not implemened!
  - Double escape peaks!
    - Simple to do but I lack data.
  - E0 transitions and internal conversion.
    - Can effect gamma-gamma coincidence intensities.