Radiative Pion Decay Monte Carlo Generators for NA62

D. Protopopescu,* I. O. Skillicorn, and D. Britton
School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, United Kingdom

(Dated: October 24, 2014)

Monte Carlo generators for the $\pi \to l\nu\gamma$ process were implemented for NA62, starting from the existing implementation of the kaon decay generators and drawing on expertise from experiments like PiENu, KLOE, and from recent literature.

Keywords: Radiative pion decay; NA62; Monte Carlo

Contents

I. INTRODUCTION

The NA62 Monte Carlo simulations software is a mature package, containing code that covers all possible kaon decay scenarios and including state-of-the art theoretical results. In view of measuring the $\Gamma(K \to e\nu(\gamma))/\Gamma(\pi \to e\nu(\gamma))$ ratio as proposed in [1], one would need state of the art generators for in-beam pion decay.

In this paper we describe how Monte Carlo generators for the $\pi \to l\nu(\gamma)$ processes were implemented in Fortran for NA62, starting from the existing kaon decay generators and drawing on expertise from PiENu [2], KLOE [3] and from the most recent literature. This paper is intended for readers familiar with the NA62 code, Geant4, C++ and Fortran and its main purpose is to document and support the newly implemented generators for $\pi \to l\nu\gamma$ decays.

In Section II we describe the existing code. Section III outlines the theory and section IV compares the results of the BertI and Gatti based codes. In Section V we describe the implementation of the code and Section VI is a summary of the paper.

II. EXISTING CODE

At the time of writing, we had a set of $K \to l\nu\gamma$ generators (written in Fortran) included in the NA62 MC software package and a $\pi \to l\nu\gamma$ generator from PiENu MC.

The NA62 kaon decay modes are implemented in `kch2lnu.F`, `kch2lnug.F` and `kch2lnug_ib.F`, and encoded as in Table I. Channels 20-26 correspond to $l = e$ and 30-36 to $l = \mu$, however, both e and μ are handled by the same Fortran subroutines. The acronyms stand for: IB - inner brehmsstrahlung, SD - structure dependent, INT - interference between IB and SD components [4].

The PiENu generators for $\pi \to l\nu\gamma$ are implemented in C++ and use a customised version of the Geant4 `G4PionRadiativeDecay` class, documented in [2].

To implement our $\pi \to l\nu(\gamma)$ Monte Carlo generators we started by examining existing theoretical calculations from literature and then comparing their software implementations.

```
Kch2enu                =  20; // kch2lnu.F
Kch2enug_ib            =  21; // kch2lnug_ib.F
Kch2enug_ib_cutoff     =  22; // kch2lnug.F mode=1
Kch2enug_sdp           =  23; // kch2lnug.F mode=2
Kch2enug_sdm           =  24; // kch2lnug.F mode=3
Kch2enug_intp          =  25; // kch2lnug.F mode=4
Kch2enug_intm          =  26; // kch2lnug.F mode=5

Kch2munu               =  30;
Kch2munug_ib           =  31; // (IB)
Kch2munug_ib_cutoff    =  32; // (IB with cutoff)
Kch2munug_sdp          =  33; // (SD+)
Kch2munug_sdm          =  34; // (SD-)
Kch2munug_intp         =  35; // (INT+)
Kch2munug_intm         =  36; // (INT-)
```

TABLE I: Kaon decay modes as encoded in the existing implementation. Naming conventions are explained in text.

III. THEORY

A review of the existing literature, beyond the references given in the code source headers, was done. A summary of our findings and comparisons between various treatments are presented in this section.

The widths implemented in the original NA62 kaon decay generators (`kch2lnug.F`, aka Bertl's - see section II) are based on the equations from [6, 7]:

$$\begin{aligned}\frac{d^2\Gamma_{K\to l\nu\gamma}}{dxdy} &= \frac{\alpha}{2\pi}\Gamma_{K\to l\nu}\frac{1}{(1-r)^2}\Big\{\psi^{(0)}_{IB}(x,y)\\ &+\frac{m_K^2}{4rf_K^2}\left[(F_V+F_A)^2\psi^{(0)}_{SD+}(x,y)+(F_V-F_A)^2\psi^{(0)}_{SD-}(x,y)\right]\\ &+\frac{m_K}{f_K}\left[(F_V+F_A)\psi^{(0)}_{INT+}(x,y)+(F_V-F_A)\psi^{(0)}_{INT-}(x,y)\right]\Big\}\end{aligned} \tag{1}$$

with the notations

$$\begin{aligned}\psi^{(0)}_{IB}(x,y) &= \frac{1-y+r}{x^2(x+y-1-r)}\left[x^2+2(1-x)(1-r)-\frac{2xr(1-r)}{x+y-1-r}\right]\\ \psi^{(0)}_{SD+}(x,y) &= (x+y-1-r)\left[(x+y-1)(1-x)-r\right]\\ \psi^{(0)}_{SD-}(x,y) &= (1-y+r)\left[(1-y)(1-x)+r\right]\\ \psi^{(0)}_{INT+}(x,y) &= \frac{1-y+r}{x(x+y-1-r)}\left[(1-x)(1-x-y)+r\right]\\ \psi^{(0)}_{INT-}(x,y) &= \frac{1-y+r}{x(x+y-1-r)}\left[x^2-(1-x)(1-x-y)-r\right]\end{aligned} \tag{2}$$

where $x = 2E_\gamma/m_K$, $y = 2E_l/m_K$ and $r = (m_l/m_K)^2$. F_V and F_A are the vector and axial vector form factors, respectively, that can be taken from [8].

In the case of the pion, there is a kinematic dependence of the vector form factor F_V:

$$F_V^\pi(s) = F_V^\pi(0)(1+as),$$

with $s = (1-2E_\gamma/m_\pi) = 1-x$ which should be taken into account [9]. The slope parameter a can be taken from [8].

The integration limits (i.e. physical range for quantities x and y) are

$$0 \le x \le 1-r \qquad 1-x+\frac{r}{1-x} \le y \le 1+r \tag{3}$$

These intervals are sampled uniformly, however, a $10\,MeV$ infrared cutoff is applied, which will restrict the sampling range of x to

$$x_0 \le x \le 1-r \qquad \text{where} \qquad x_0 = 2E_{0\gamma}/m_K, \quad E_{0\gamma} = 10\,MeV \tag{4}$$

Such a cutoff is necessary because experimentally it is not possible to measure arbitrarily low energy gammas. However, this cutoff should not be understood as being there to prevent $\psi^{(0)}_{IB}(x,y) \to 0$ when $x \to 0$. This does not happen in practice because, when x approaches zero, the width of the phase-space is zero, i.e. from Eq.(3)

$$x \to 0 \quad \Rightarrow \quad 1+r \le y \le 1+r \tag{5}$$

This has been tested numerically and, indeed, x never touches zero during phase-space sampling (see Fig.2).

The PiENu implementation [5] follows the same decomposition in IB, SD$\pm$ and INT$\pm$ terms, but the approximation $r = 0$ is applied [13], with m_K replaced everywhere by m_π, such that the above equations become [2, 4]:

$$\begin{aligned}\psi^{(0)}_{IB}(x,y) &= \frac{(1-y)(1+(1-x)^2)}{x^2(x+y-1)}\\ \psi^{(0)}_{SD+}(x,y) &= (1-x)(x+y-1)^2\\ \psi^{(0)}_{SD-}(x,y) &= (1-x)(1-y)^2\\ \psi^{(0)}_{INT+}(x,y) &= -\frac{(1-x)(1-y)}{x}\\ \psi^{(0)}_{INT-}(x,y) &= \frac{x(1-y)}{x+y-1}+\frac{(1-x)(1-y)}{x}\end{aligned} \tag{6}$$

The integration limits

$$2\sqrt{r} \le y \le 1+r \qquad 1-\frac{1}{2}\left[y+\sqrt{y^2-4r}\right] \le x \le 1-\frac{1}{2}\left[y-\sqrt{y^2-4r}\right] \tag{7}$$

are equivalent to Eq.(3). The x and y sampling is done as explained in [2]. However, we found that the x and y distributions obtained this were exhibiting abnormal peaks, which we believed were unphysical. This was corrected [5] by multiplying the matrix element for each event by $(1-\sqrt{r})^2\sqrt{y^2-4r}$.

With respect to the question about collinear divergence, we have not found any problems when using Eq.(2). We have calculated the angle with [4]:

$$\cos\theta_{e\gamma} = \frac{y(x-2)+2(1-x+r)}{x\sqrt{y^2-4r}} \tag{8}$$

Second order radiative corrections corresponding to the above $\psi^{(0)}$ representation are described in [10], from where we choose the simplified form:

$$\begin{aligned}
\psi^{(1)}_{IB}(x,y) &= \frac{1+\bar{x}^2}{x^2}\left[\frac{3}{2}\frac{\bar{y}}{z}+\frac{\bar{y}}{\bar{x}}-\frac{\bar{x}+xy}{\bar{x}^2}\ln y+2\frac{\bar{y}}{z}\ln\frac{\bar{y}}{y}-\frac{x(\bar{x}^2+y^2)}{\bar{x}^2 z}\ln\frac{x}{z}\right] \\
\psi^{(1)}_{SD+}(x,y) &= \bar{x}\left[\frac{3}{2}z^2+\frac{1-y^2}{2}+\bar{y}(y-2\bar{x})+\bar{x}(\bar{x}-2y)\ln y-\bar{x}^2\bar{y}+2z^2\ln\frac{\bar{y}}{y}\right] \\
\psi^{(1)}_{SD-}(x,y) &= \bar{x}\left[\frac{3}{2}\bar{y}^2+\frac{1-y^2}{2}+\bar{y}(y-3)+(1-2y)\ln y+2\bar{y}^2\ln\frac{\bar{y}}{y}\right] \\
\psi^{(1)}_{INT+}(x,y) &= \frac{\bar{x}}{x}\left[\frac{\bar{y}}{2}-\bar{y}\ln y-2\bar{y}\ln\frac{\bar{y}}{y}\right] \\
\psi^{(1)}_{INT-}(x,y) &= \frac{1}{x}\left[-\frac{1}{2}\bar{x}\bar{y}+\frac{3}{2}\frac{x^2\bar{y}}{z}+\bar{x}\left(\bar{y}\ln y+2\bar{y}\ln\frac{\bar{y}}{y}\right)\right. \\
&\quad \left.+x^2\left(\frac{\bar{y}}{\bar{x}}-\frac{\bar{x}+xy}{\bar{x}^2}\ln y+2\frac{\bar{y}}{z}\ln\frac{\bar{y}}{y}-\frac{x(\bar{x}^2+\bar{y}^2)}{\bar{x}^2 z}\ln\frac{x}{z}\right)\right]
\end{aligned} \tag{9}$$

where $\bar{x}=1-x,\ \bar{y}=1-y,\ z=x+y-1$ and $\psi^{(1)}_i$ is the radiative correction for term $i = IB, SD\pm, INT\pm$ from Eq.(6), e.g.

$$\psi_{IB}(x,y) = \psi^{(0)}_{IB}(x,y) + \frac{\alpha}{2\pi}(L_e-1)\psi^{(1)}_{IB}(x,y), \qquad L_e = \ln\frac{y^2}{r} \tag{10}$$

Plots showing the magnitude of the radiative corrections are shown in figures 7 to 9.

A different formulation is presented in [3] and encoded in `kch2lnug_ib.F` (aka Gatti's, or KLOE code - see section II), where the infrared and collinear divergences are explicitly factored out to improve the efficiency of the x, y sampling:

$$\begin{aligned}
\psi_G(x,y) &\propto \left(\frac{E_\gamma}{E_{CM}}\right)^{b-1}\frac{1}{E_l-p_l\cos\theta_{l\gamma}}\,G(k,\cos\theta_{l\gamma}) \qquad \text{with} \\
b &= -\frac{2\alpha}{\pi}\left[1-\frac{1-\beta^2}{2\beta}\ln\frac{1+\beta}{1-\beta}\right], \qquad 1-\beta^2 = \left(\frac{2m_K m_l}{m_K^2+m_l^2}\right)^2
\end{aligned} \tag{11}$$

which we can put in the form:

$$\begin{aligned}
G(x,y) &= G_{IB}(x,y)+\frac{m_K^2}{4rf_K^2}\left[(F_V+F_A)^2 G_{SD+}(x,y)+(F_V-F_A)^2 G_{SD-}(x,y)\right] \\
&+ \frac{m_K}{f_K}\left[(F_V+F_A)G_{INT+}(x,y)+(F_V-F_A)G_{INT-}(x,y)\right]
\end{aligned} \tag{12}$$

where we've separated the terms

$$\begin{aligned} G_{IB}(x,y) &= (1-y)\left[x^2+2(1-r)\left(1-x-\frac{r}{y}\right)\right] \\ G_{SD+}(x,y) &= -x^4y^2(r-y+xy) \\ G_{SD-}(x,y) &= -x^4y(y-1)(r-y+xy-x+1) \\ G_{INT+}(x,y) &= x^2(y-1)(r-y+xy) \\ G_{INT-}(x,y) &= x^2(y-1)(r-y+xy-x) \end{aligned} \tag{13}$$

Here x is the same as before, but $y=(E_l-p_l\cos\theta_{l\gamma})/m_K$, and sampling is done via the inverse-transform method

$$x=(1-r)p^{1/b} \qquad \text{and} \qquad y=r^{1-p} \tag{14}$$

with p sampled uniformly between 0 and 1 while $-1\leq\cos\theta_{l\gamma}\leq 1$. As a note, we have tested whether using the same x,y sampling as in Bertl's implementation with the aid of a Jacobian gives the same results, and it did.

The KLOE kaon decay generator is used solely to produce the IB term without the IR cutoff Eq.(4) - see section II. For that, both form factors are set to zero ($F_V=F_A=0$) and thus only the first term in Eq.(12) is calculated.

IV. COMPARISON OF CODES

We have adapted the KLOE code for $\pi\to e\nu\gamma$ (see appendices VII D and VII E) including all terms, and compared calculations based on Eq.(13) with those obtained with Eq.(2).

The two results agree within 2% above the IR cutoff, as one can see in figures 4-5. If the IR cutoff is lowered to $1\,MeV$, then there is a 10% discrepancy between the two calculations (see Fig.6). If second order radiative corrections are included as in Eq.(10), the discrepancy increases.

The x,y sampling explained in [3] is very efficient hence Gatti's generator is orders of magnitude faster than Bertl's.

V. IMPLEMENTATION

The simplest way to incorporate the $\pi\to l\nu(\gamma)$ generators in the NA62MC software package was to write the code in Fortran, since all the other (former CMC[14]) generators contained therein are written in Fortran.

We've used the latest values of the form factors and slope parameter a of $F_V^\pi(x)$ from [8]. We've copied the structure, naming conventions and scaling factors from the existing kaon decay generators, but we added a `mode=6` corresponding to the full calculation (see section II). The `modes` are used to separate various contributions to the cross-section ($IB, SD\pm$, $INT\pm$) and are added together later on during *event mixing*. We've updated the `modes` as in Table II.

The x,y sampling is done uniformly over the intervals from Eq.(3 - 4) - see lines 63-70 of `pilnug.F` from appendix VII E.

In `pilnug.F`, the matrix elements are calculated with Eq.(10), *i.e.* including radiative corrections. With our algorithm compiled in ROOT 6.00, we were able to generate roughly 200 $\pi\to e\nu\gamma$ decays per second on a 2.3 GHz Intel Core i5 MacBook Pro machine running OS X Mavericks.

File	`mode`	Contribution	Term(s)	Eqs.
`pilnug.F`	1	IB	$\psi_{IB}^{(0)}+c_R\psi_{IB}^{(1)}$	(10)
	2	SD+	$c_{SD+}(\psi_{SD+}^{(0)}+c_R\psi_{SD+}^{(1)})$	
	3	SD–	$c_{SD-}(\psi_{SD-}^{(0)}+c_R\psi_{SD-}^{(1)})$	
	4	INT+	$\text{-}c_{INT+}(\psi_{INT+}^{(0)}+c_R\psi_{INT+}^{(1)})$	
	5	INT–	$c_{INT-}(\psi_{INT-}^{(0)}+c_R\psi_{INT-}^{(1)})$	
	6	ALL	$\sum_{i=1}^{5} c_i\left[\psi_i^{(0)}+c_R\psi_i^{(1)}\right]$	(2)
`pienug_gatti.F`	1	IB	G_{IB}	(13)
	2	SD+	$c_{SD+}G_{SD+}$	
	3	SD–	$c_{SD-}G_{SD-}$	
	4	INT+	$\text{-}c_{INT+}G_{INT+}$	
	5	INT–	$c_{INT-}G_{INT-}$	
	6	ALL	$\sum_{i=1}^{5} c_iG_i$	(12)

TABLE II: The `modes` as encoded in our Fortran implementations, with $c_R=(\alpha/2\pi)(L_e-1)$ from Eq.(10), $c_1=1$, and $c_{SD\pm}=(1/4r)(m_\pi/f_\pi)^2(F_V\pm F_A)^2$, $c_{INT\pm}=(m_\pi/f_\pi)(F_V\pm F_A)$ as in Eq.(2).

In `pienug_gatti.F`, we've implemented Gatti's approach Eq.(12-13), which does not explicitly include second order radiative corrections. The Gatti code is much faster, due to its efficient sampling, generating around 30k decays per second. However, we believe this will not make a significant difference when running Monte Carlo, since other parts of the simulation chain are orders of magnitude more time consuming.

VI. SUMMARY

Monte Carlo generators for the radiative pion decay $\pi\to e\nu\gamma$ process were implemented in Fortran. To do

this, a thorough comparison with the existing kaon generators and with code used by the PiENu experiment were made and the recent literature was checked.

We've investigated in detail the $\pi \to e\nu\gamma$ case, and we have updated form factors using the latest PDG data. We provide two implementations which correspond to different mathematical formulations but give similar results. Gatti's formulation is much faster due to very efficient phase-space sampling.

Second order radiative corrections account for a 15% difference at $E_\gamma = 10\,MeV$, falling to zero at high photon energies (see Fig.7).

In the case of $\pi \to \mu\nu\gamma$, the muon is more massive and radiative effects are negligible (see Fig.3), and the use of the `pilnu.F` generator might be sufficient.

Acknowledgments

Lots of thanks to A. Sergi (U. of Birmingham) for liaising with the NA62 software development group. Special thanks to E. Goudzovski (U. of Birmingham) for valuable hints and references and to A. Sher (TRIUMF) for providing the PiENu code together with explanations and comments.

[1] F. Gonnella et al., Proposal for the measurement of $\Gamma(K \to e\nu(\gamma))/\Gamma(\pi \to e\nu(\gamma))$, internal note (2012)
[2] M. Blecher, Internal PiENu Technical Note (2007)
[3] C.Gatti, Eur. Phys. J. C 45, 417-420 (2006) and KLOE Note N^o 194 (2004)
[4] D. A. Bryman, P. Depommier and C. Leroy, Phys. Reps. 88 (1982) 151
[5] A. Sher, private communication 24/09/2014
[6] Chuan-Hung Chen, Chao-Qiang Geng, Chong-Chung Lih, Phys. Rev. D 77, 014004 (2008), arXiv:0710.2971
[7] J. Bijnens, G. Ecker, J. Gasser, Nucl. Phys. B 396, 81-118 (1993), arXiv:hep-ph/9209261
[8] J. Beringer et al. (PDG), Phys. Rev. D 86, 010001 (2012) - aka Rosner
[9] K. Nakamura et al. (PDG), JPG 37, 075021 (2010) - aka Bertl
[10] Yu. M. Bystritsky, E. A. Kuraev, and E. P. Velicheva, Phys. Rev. D 69, 114004 (2004)
[11] NA62 software SVN howto:

http://sergiant.web.cern.ch/sergiant/NA62FW/html/index.html

[12] all paths are relative to NA62MC SVN directory, see [11]
[13] if the lepton is e, the ratio r for kaon is $r_{eK} = 10^{-6}$, while for pion $r_{e\pi} \approx 1.33 \times 10^{-5}$, for muons r is closer to 1.
[14] CMC was the software framework of NA48/2

VII. APPENDIX

Included here are full listings of the Fortran modules that implement our $\pi \to l\nu(\gamma)$ generators, together with a short section describing how the code is organised and how the function calls are done, plus a FAQ.

A. Files and functions

Our $\pi \to e\nu(\gamma)$ generators are implemented in three Fortran files located in the `NA62MC/Generator/` directory. `pienug_gatti.F` and `pilnug.F` implement the IB, SD$\pm$ and INT$\pm$ terms, the latter including radiative corrections, as described in section III. For the sake of completeness, `pilnu.F` implements simply the $\pi \to l\nu$ process.

Decay channel IDs are defined in `src/CMC.cc` [12] (and listed in `Generator/decay_dictionary.txt`). For each decay channel, C++ function prototypes are implemented in `src/G4CMCDecayer.cc` and the linkage to the corresponding Fortran subroutines is defined in `include/CMC.hh` and `src/CMC.cc`.

At the time of writing, the proposed channel IDs for the new decays are as follows:

```
Pich2enu                =  220;
Pich2enug               =  221;  // (Full)
Pich2enug_ib            =  222;  // (IB)
Pich2enug_sdp           =  223;  // (SD+)
Pich2enug_sdm           =  224;  // (SD-)
Pich2enug_intp          =  225;  // (INT+)
Pich2enug_intm          =  226;  // (INT-)
```

We suggest utilising `pilnug.F` for generating channels 221-226 and `pilnu.F` for channel 220.

B. FAQ

How do I get the generators ? The code for the $\pi \to l\nu(\gamma)$ generators is included in the standard NA62MC software package [11].

What to do if I find an error in the code ? If you find any mistake in the code, or have any suggestion, please contact E. Goudzovski or one of the authors of this note.

What if the actual implementation is changed/updated? We may update this note in the future. If you detect a change/mismatch between this document and the actual code, please verify whether you have the latest version of this note.

C. Figures

Plots mentioned in text are shown on pages 8 to 16.

D. Conversion of Gatti's formulae

Gatti's equation for $K \to e\nu\gamma$ is implemented in `kch2lnug_ib.F` like this

```
Ampiezza = -(-2.d+00*Eg**2*Fk*mkch**2*mlep**2*(-1.d+00 + y)*y*
             ((LH - RH)*(mlep**2 - mkch**2*y) +
             2.d+00*Eg*mkch*(LH*(-1.d+00 + y) - RH*y))
             + 2.d+00*Eg**4*mkch**4*y**2*
             (LH**2*(mlep**2 - mkch**2*(-1.d+00 + y))*(-1.d+00 + y) -
             RH**2*y*(-mlep**2 + mkch**2*y) +
             2.d+00*Eg*mkch*(LH**2*(-1.d+00 + y)**2 + RH**2*y**2))
             + Fk**2*mlep**2*(-1.d+00 + y)*
             (mlep**4 - 2.d+00*Eg*mkch**3*y + mkch**4*y +
             2.d+00*Eg*mkch*mmu**2*y -
             mkch**2*(-2.d+00*Eg**2*y + mlep**2*(1.d+00 + y))))
             /(mkch**2*y)
```

where we assume that `LH` $= (F_V - F_A)/m_K$, `RH` $= (F_V + F_A)/m_K$. In the muon case, there's an extra y at the denominator.

Now let's simplify the notation to make it easier to manipulate this equation. We put `Eg` $= E$, `mkch` $= M$, `mlep` $= m$, `RH` $= R$, `LH` $= L$, `Fk` $= F$ and then it looks like this

$$\begin{aligned}
A = & - (-2E^2FM^2m^2(y-1)y((L-R)(m^2-M^2y) + && (15)\\
& \quad 2EM(L(y-1)-Ry)) && (16)\\
& + 2E^4M^4y^2(L^2(m^2-M^2(y-1))(y-1) - R^2y(-m^2+M^2y) + && (17)\\
& \quad 2EM(L^2(y-1)^2 + R^2y^2)) && (18)\\
& + F^2m^2(y-1)(m^4 - 2EM^3y + M^4y + 2EMm^2y - && (19)\\
& \quad M^2(-2E^2y + m^2(1+y))))/(M^2y) && (20)
\end{aligned}$$

or

$$\begin{aligned}
A = & - \frac{1}{M^2y}\left\{-2E^2FM^2m^2(y-1)y\left[(L-R)(m^2-M^2y) + 2EM(L(y-1)-Ry)\right]\right. && (21)\\
& + 2E^4M^4y^2\left[L^2(m^2-M^2(y-1))(y-1) - R^2y(-m^2+M^2y) + 2EM(L^2(y-1)^2 + R^2y^2)\right] && (22)\\
& + \left. F^2m^2(y-1)\left[m^4 - 2EM^3y + M^4y + 2EMm^2y - M^2(-2E^2y + m^2(1+y))\right]\right\} && (23)
\end{aligned}$$

or, using $2E = xM$,

$$\begin{aligned}
A = & - \frac{1}{M^2y}\left\{-\frac{1}{2}x^2M^4Fm^2(y-1)y\left[(L-R)(m^2-M^2y) + xM^2(L(y-1)-Ry)\right]\right. && (24)\\
& + \frac{1}{8}x^4M^8y^2\left[L^2(m^2-M^2(y-1))(y-1) - R^2y(-m^2+M^2y) + xM^2(L^2(y-1)^2 + R^2y^2)\right] && (25)\\
& + \left. F^2m^2(y-1)\left[m^4 - xM^4y + M^4y + xM^2m^2y - M^2(-\frac{1}{2}x^2M^2y + m^2(1+y))\right]\right\} && (26)
\end{aligned}$$

If we factor out M^4, and use $r = (m/M)^2$ in the last term we obtain

$$\begin{aligned}
A = & - \frac{M^2}{y}\left\{-\frac{1}{2}x^2Fm^2(y-1)y\left[(L-R)(m^2-M^2y) + xM^2(L(y-1)-Ry)\right]\right. && (27)\\
& + \frac{1}{8}x^4M^4y^2\left[L^2(m^2-M^2(y-1))(y-1) - R^2y(-m^2+M^2y) + xM^2(L^2(y-1)^2 + R^2y^2)\right] && (28)\\
& + \left. F^2m^2(y-1)\left[r^2 - xy + y + rxy + \frac{1}{2}x^2y - r(1+y)\right]\right\} && (29)
\end{aligned}$$

We can group factors with same powers of R and L

$$A = -\frac{M^2}{y}\left\{\frac{1}{2}x^2Fm^2(y-1)y\left[R(m^2-M^2y+M^2xy)-L(m^2-M^2y+M^2xy-M^2x)\right]\right. \tag{30}$$

$$+\frac{1}{8}M^4x^4y^2\left[R^2y\left(m^2+M^2y(x-1)\right)+L^2(y-1)\left(m^2+M^2(x-1)(y-1)\right)\right] \tag{31}$$

$$\left.+F^2m^2(y-1)\left[r^2-xy+y+rxy+\frac{1}{2}x^2y-r(1+y)\right]\right\} \tag{32}$$

and then factor out $F^2m^2/2$, and use $r=(m/M)^2$ in the coefficient of the SD term

$$A = -\frac{F^2m^2M^2}{2y}\left\{\frac{1}{F}x^2(y-1)y\left[R(m^2-M^2y+M^2xy)-L(m^2-M^2y+M^2xy-M^2x)\right]\right. \tag{33}$$

$$+\frac{M^2}{4rF^2}x^4y^2\left[R^2y(m^2-M^2y+M^2xy)+L^2(y-1)\left(m^2+M^2(x-1)(y-1)\right)\right] \tag{34}$$

$$\left.+(y-1)\left[x^2y+2\left(r^2-xy+y+rxy-r(1+y)\right)\right]\right\} \tag{35}$$

If we now divide by y and make use of $r=(m/M)^2$ again we get

$$A = -\frac{F^2m^2M^2}{2}\left\{\frac{M^2}{F}(y-1)x^2\left[R(r-y+xy)-L(r-y+xy-x)\right]\right. \tag{36}$$

$$+\frac{M^4}{4rF^2}x^4y\left[R^2y(r-y+xy)+L^2(y-1)\left(r+(x-1)(y-1)\right)\right] \tag{37}$$

$$\left.+(y-1)\left[x^2+\frac{2}{y}\left(r^2-xy+y+rxy-r(1+y)\right)\right]\right\} \tag{38}$$

Now we replace $F=f_\pi$, $R=(F_V+F_A)/m_\pi$, $L=(F_V-F_A)/m_\pi$ and then M,m with m_π, m_e, respectively, to revert to our notation from section III

$$A_e(x,y) = -\frac{f_\pi^2m_e^2m_\pi^2}{2}\left\{\frac{m_\pi^2}{f_\pi}(y-1)x^2\left[\frac{(F_V+F_A)}{m_\pi}(r-y+xy)-\frac{(F_V-F_A)}{m_\pi}(r-y+xy-x)\right]\right. \tag{39}$$

$$+\frac{m_\pi^4}{4rf_\pi^2}x^4y\left[\frac{(F_V+F_A)^2}{m_\pi^2}y(r-y+xy)+\frac{(F_V-F_A)^2}{m_\pi^2}(y-1)\left(r+(x-1)(y-1)\right)\right] \tag{40}$$

$$\left.+(y-1)\left[x^2+\frac{2}{y}\left(r^2-xy+y+rxy-r(1+y)\right)\right]\right\} \tag{41}$$

and after cancelling out the m_π's we regain the familliar coefficients

$$A_e(x,y) = -\frac{f_\pi^2m_e^2m_\pi^2}{2}\left\{\frac{m_\pi}{f_\pi}(y-1)x^2\left[(F_V+F_A)(r-y+xy)-(F_V-F_A)(r-y+xy-x)\right]\right. \tag{42}$$

$$+\frac{m_\pi^2}{4rf_\pi^2}x^4y\left[(F_V+F_A)^2y(r-y+xy)+(F_V-F_A)^2(y-1)\left(r+(x-1)(y-1)\right)\right] \tag{43}$$

$$\left.+(y-1)\left[x^2+\frac{2}{y}\left(r^2-xy+y+rxy-r(1+y)\right)\right]\right\} \tag{44}$$

and one can more easily identify the $INT+$ and $INT-$, $SD+$, $SD-$ and IB terms:

$$G_{IB}(x,y) = (1-y)\left[x^2+2(1-r)\left(1-x-\frac{r}{y}\right)\right] \tag{45}$$

$$G_{SD+}(x,y) = -x^4y^2(r-y+xy) \tag{46}$$

$$G_{SD-}(x,y) = -x^4y(y-1)(r-y+xy-x+1) \tag{47}$$

$$G_{INT+}(x,y) = -x^2(y-1)(r-y+xy) \tag{48}$$

$$G_{INT-}(x,y) = x^2(y-1)(r-y+xy-x) \tag{49}$$

as presented in Eq.(13) on page 4.

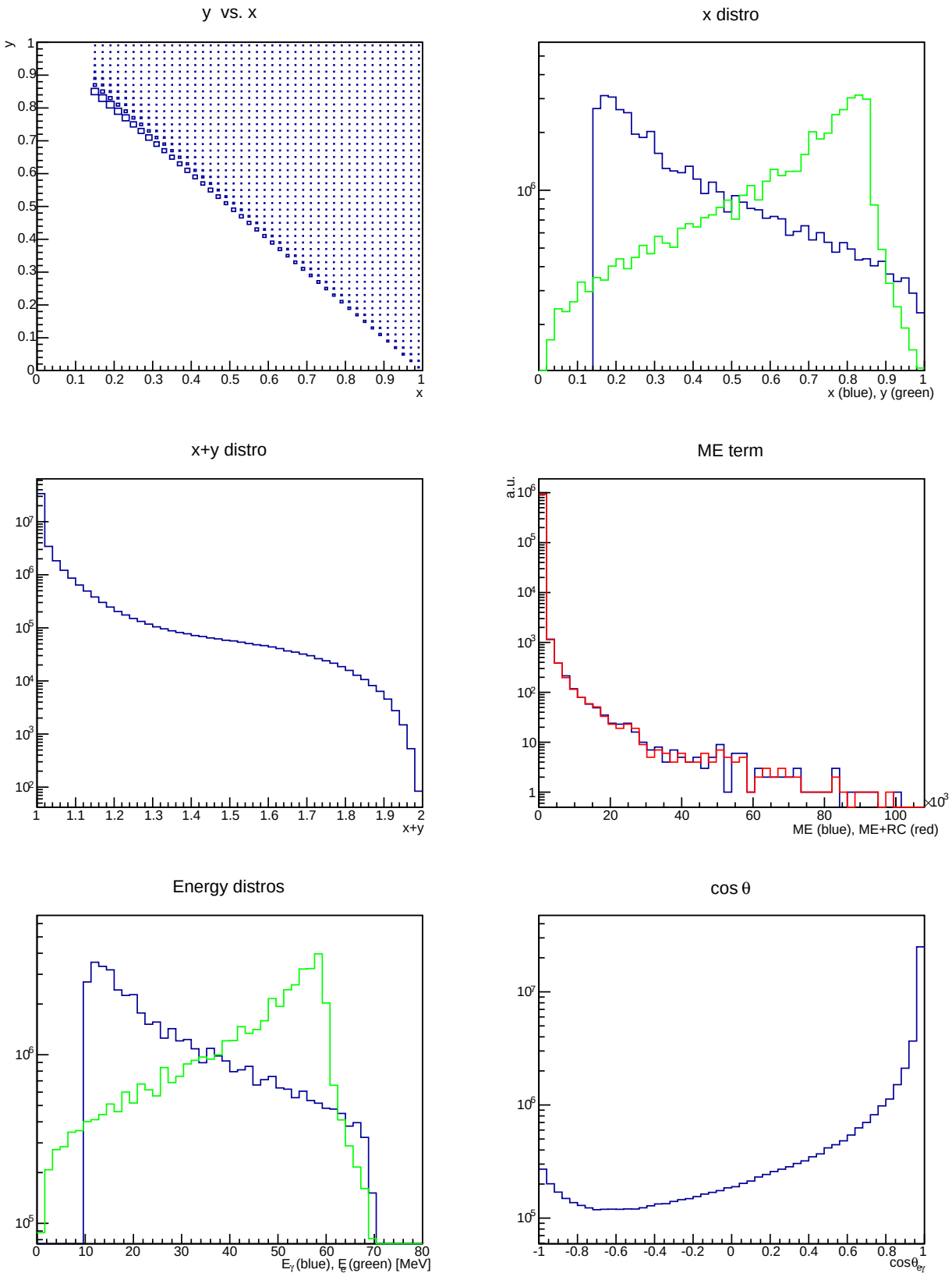

FIG. 1: Some $\pi \to e\nu\gamma$ monitoring plots for Bertl's: x versus y, x, y, and $x + y$ distributions, cross-section with and without radiative corrections, lepton and photon energies, and $\cos\theta_{e\gamma}$. The $10\,MeV$ infrared cutoff was applied.

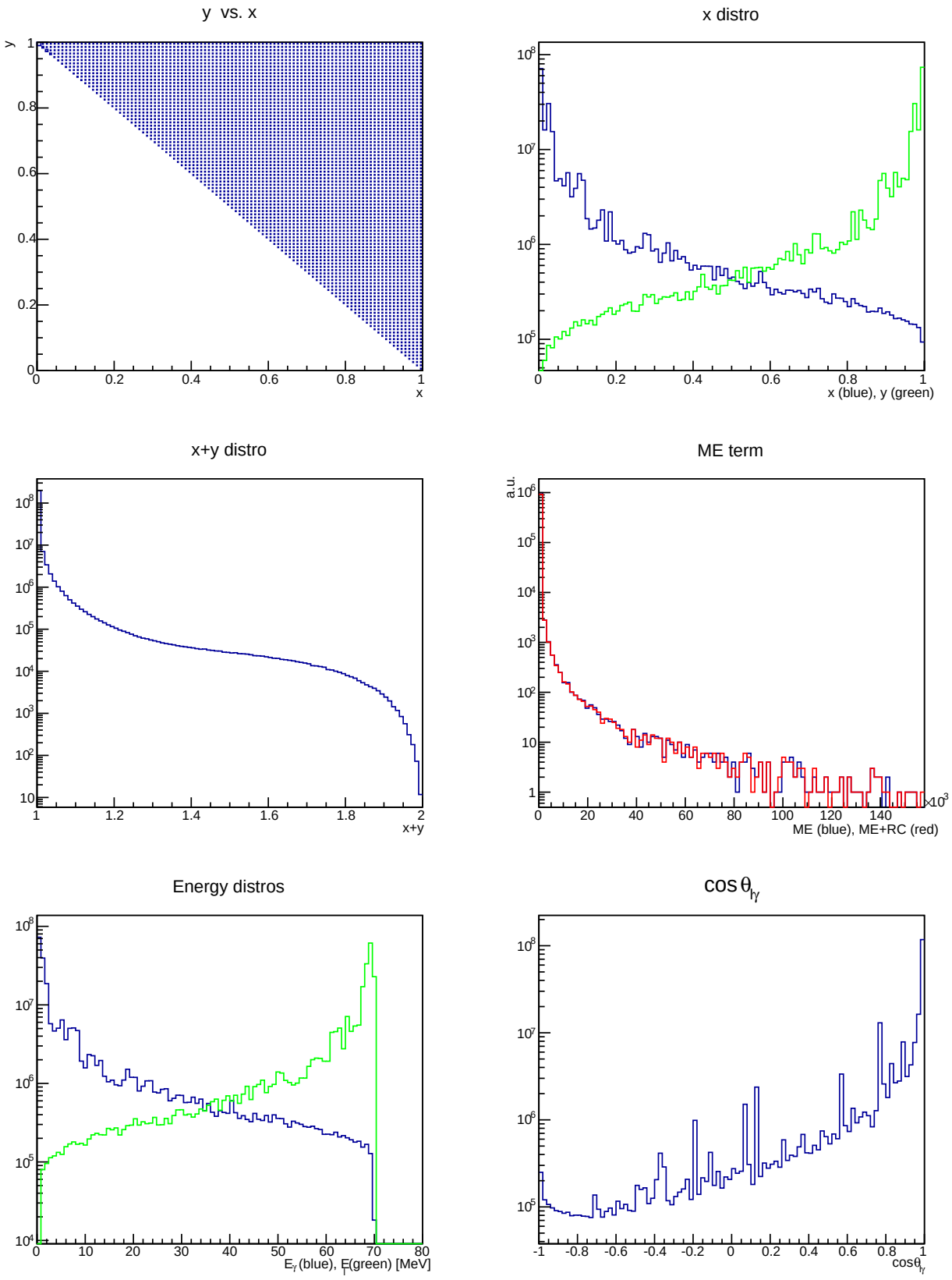

FIG. 2: Same $\pi \to e\nu\gamma$ monitoring plots: x versus y, x, y, and $x + y$ distributions as in Fig.1, but without the $E_\gamma > 10$ MeV cutoff.

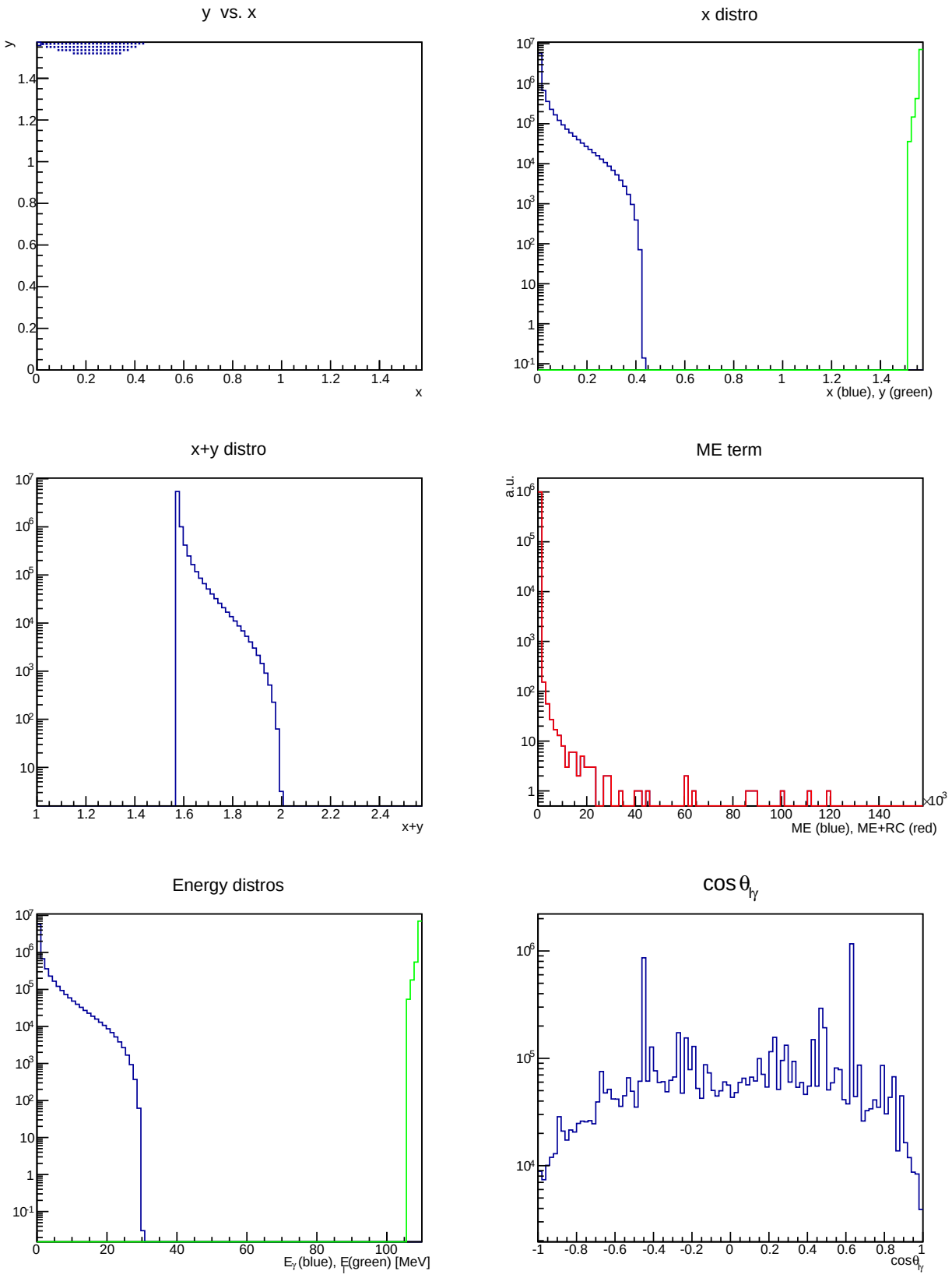

FIG. 3: Some $\pi \to \mu\nu\gamma$ monitoring plots (Bertl's code): x versus y, x, y, and $x + y$ distributions, unnormalised cross-section with and without radiative corrections, lepton and photon energies, and $\cos\theta_{e\gamma}$. Note the changed x and y distributions, and the absence of the $\theta_{l\gamma} \to 0$ divergence.

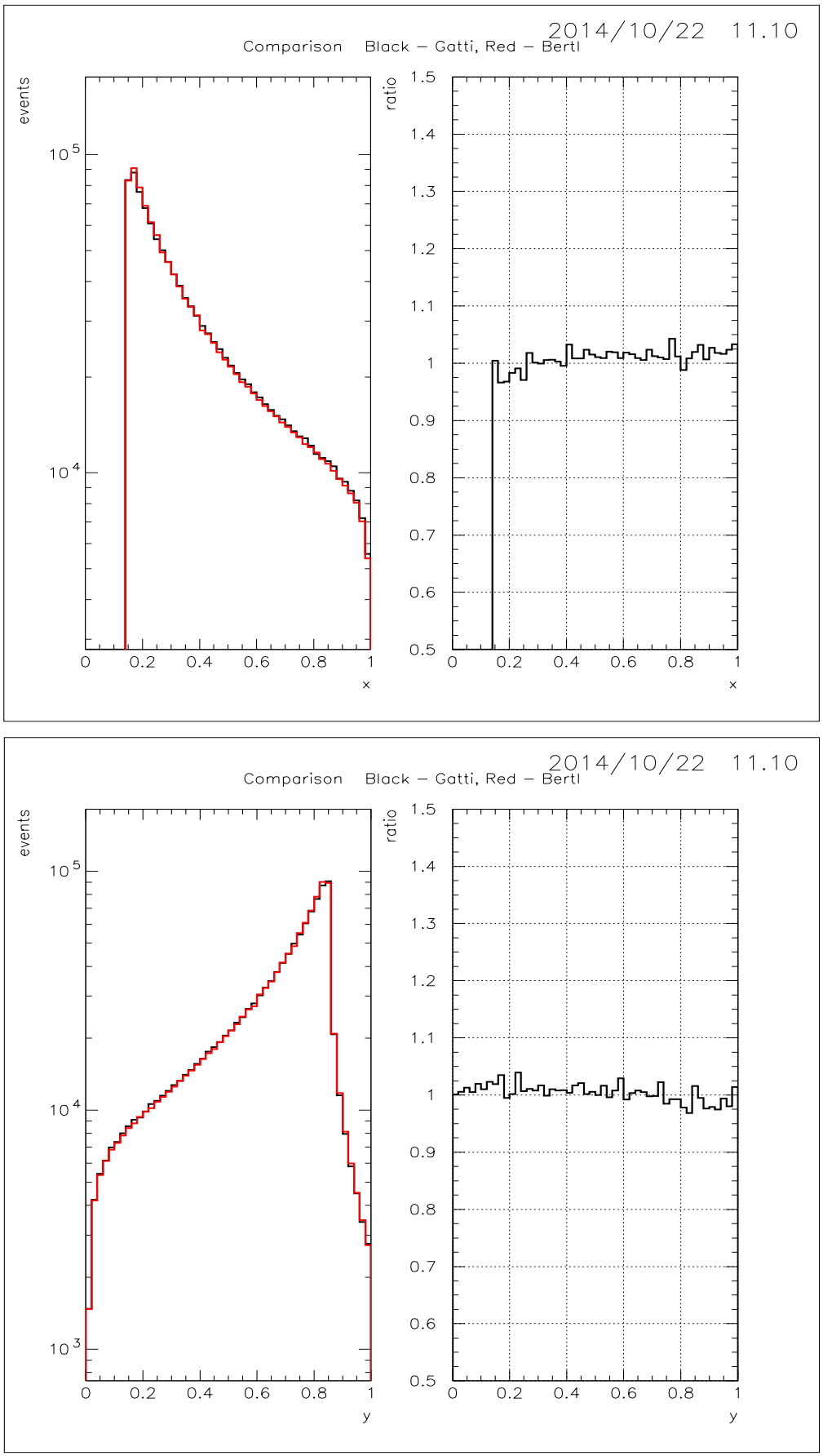

FIG. 4: Comparison between x and y distributions obtained with Bertl's [9] (red) and Gatti's [3] (black) formulae. An $E_\gamma > 10\,MeV$ cutoff is applied.

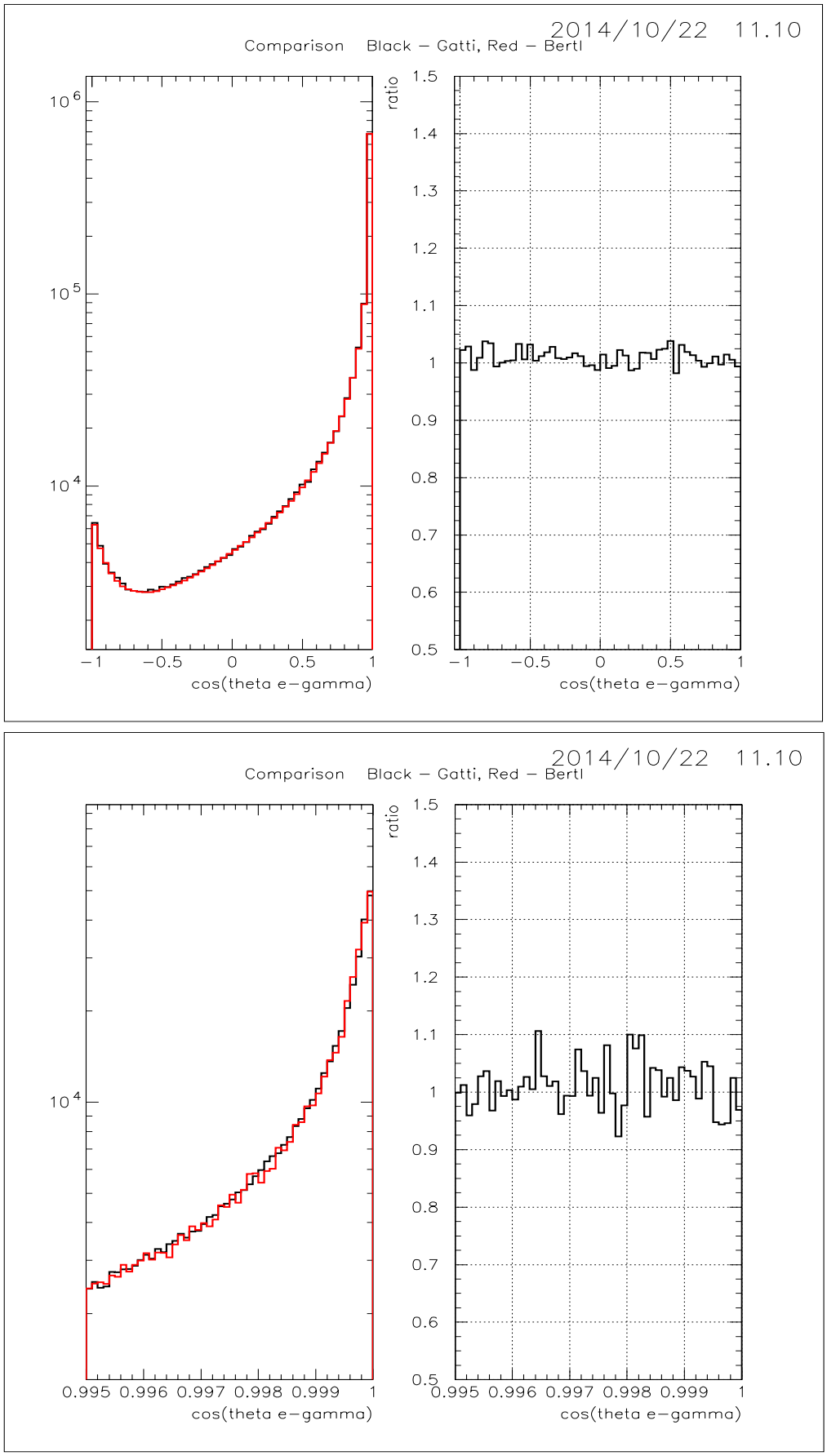

FIG. 5: Comparison between $\cos\theta_{e\gamma}$ distributions obtained with Bertl's [9] (red) and Gatti's [3] (black) formulae: top - full range, bottom - detail at very small $\theta_{e\gamma}$. See Eq.(8). An $E_\gamma > 10\,MeV$ cutoff is applied.

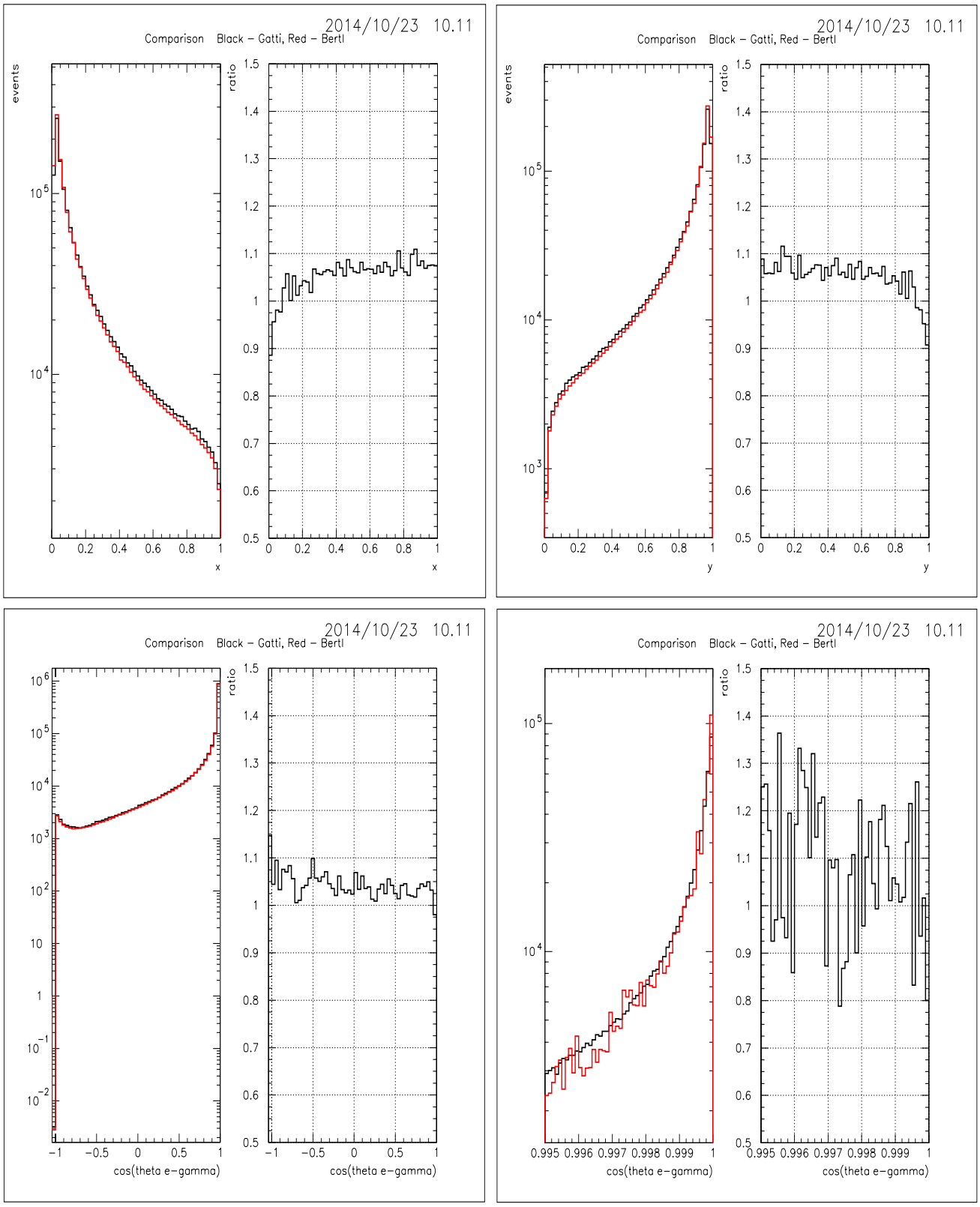

FIG. 6: Same comparison between Bertl's [9] (red) and Gatti's [3] (black) formulae, this time with a 1 MeV cutoff.

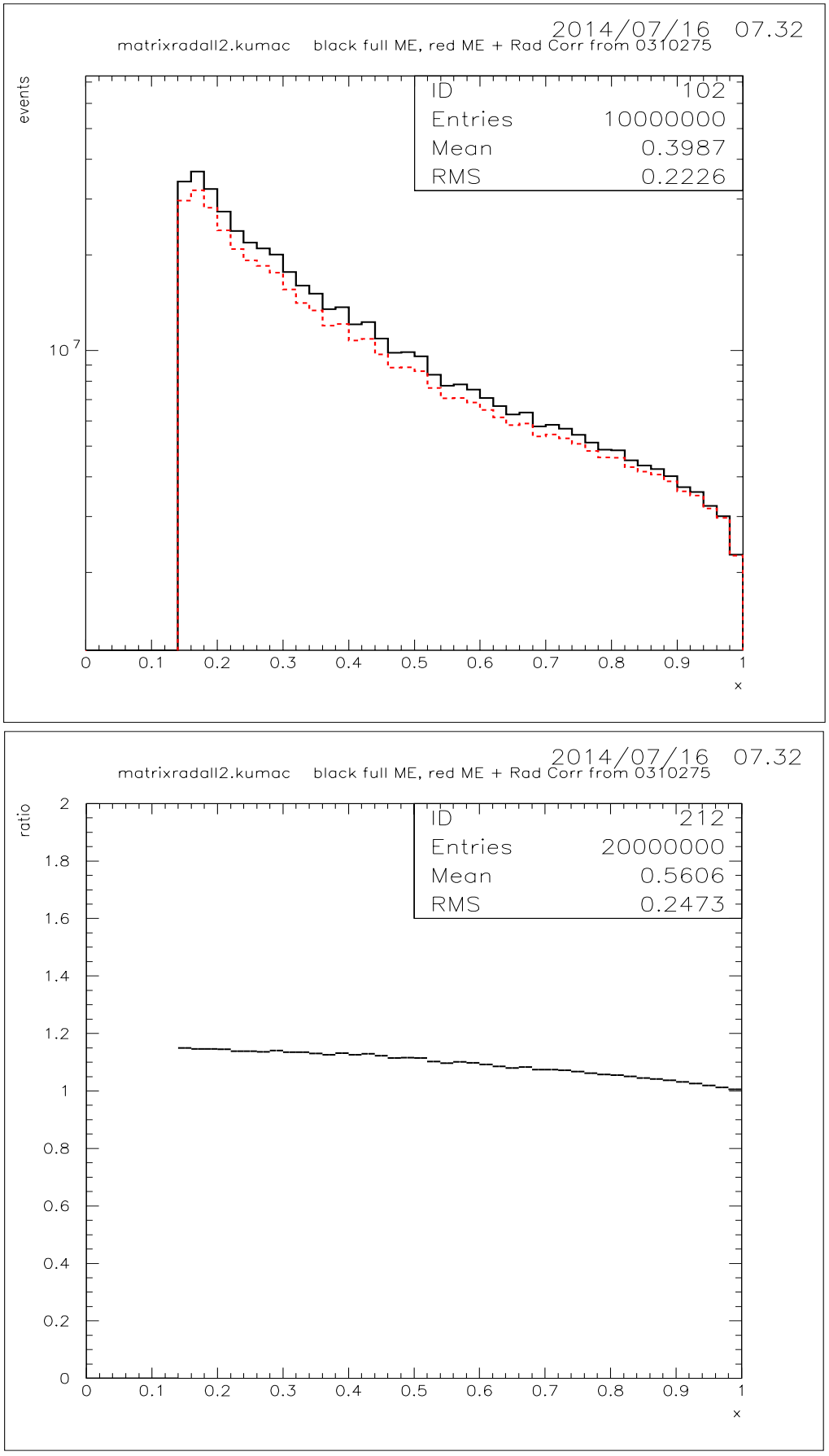

FIG. 7: Comparison between $x = 2E_\gamma/m_\pi$ distributions with (red) and without (black) radiative corrections, obtained with Eq.2 and Eq.9. Bottom panel shows the ratio. At the lowest x plotted radiative corrections account for a 15% difference.

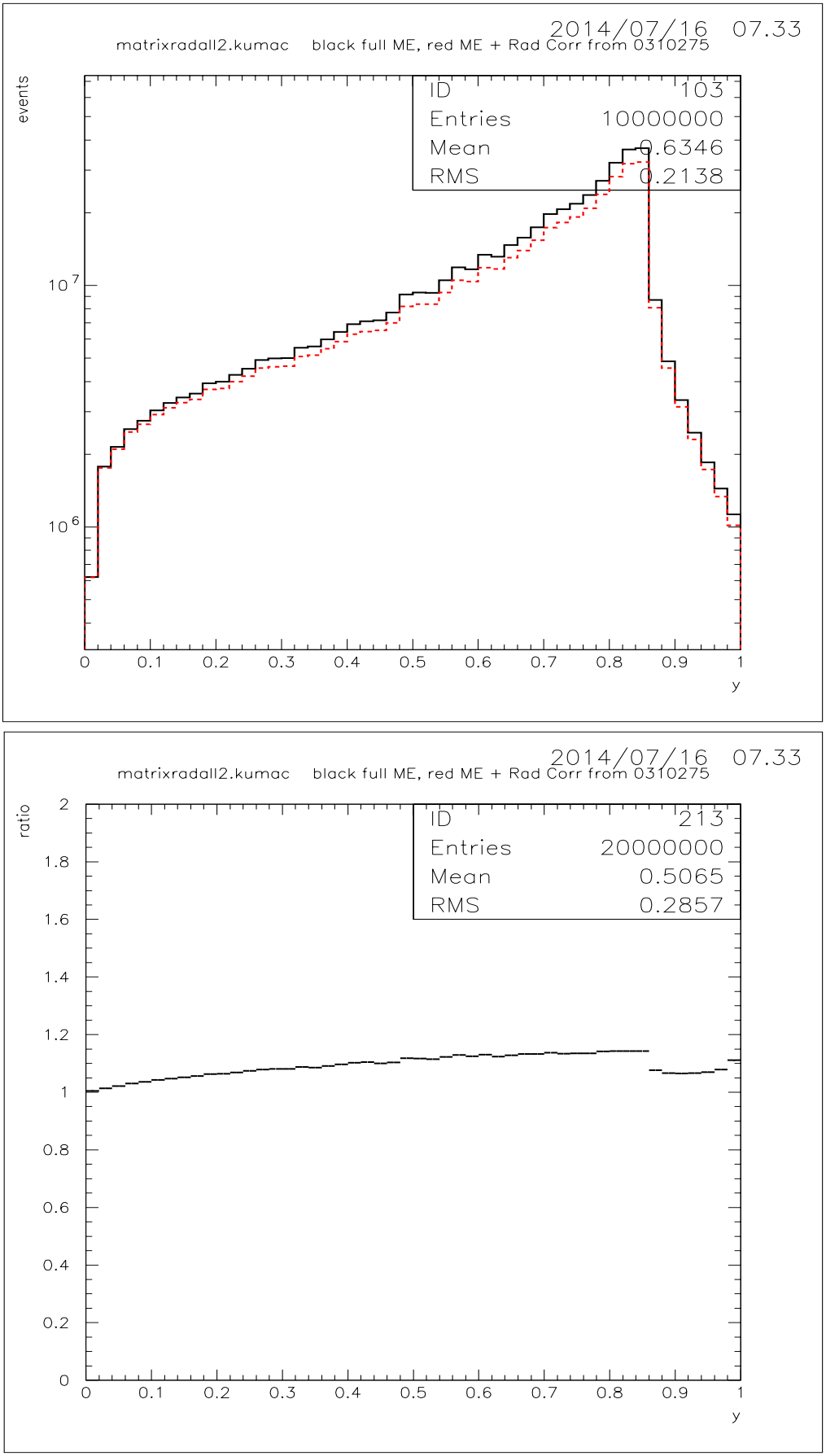

FIG. 8: Comparison between $y = 2E_e/m_\pi$ distributions with (red) and without (black) radiative corrections, obtained with Eq.2 and Eq.9. Bottom panel shows the ratio.

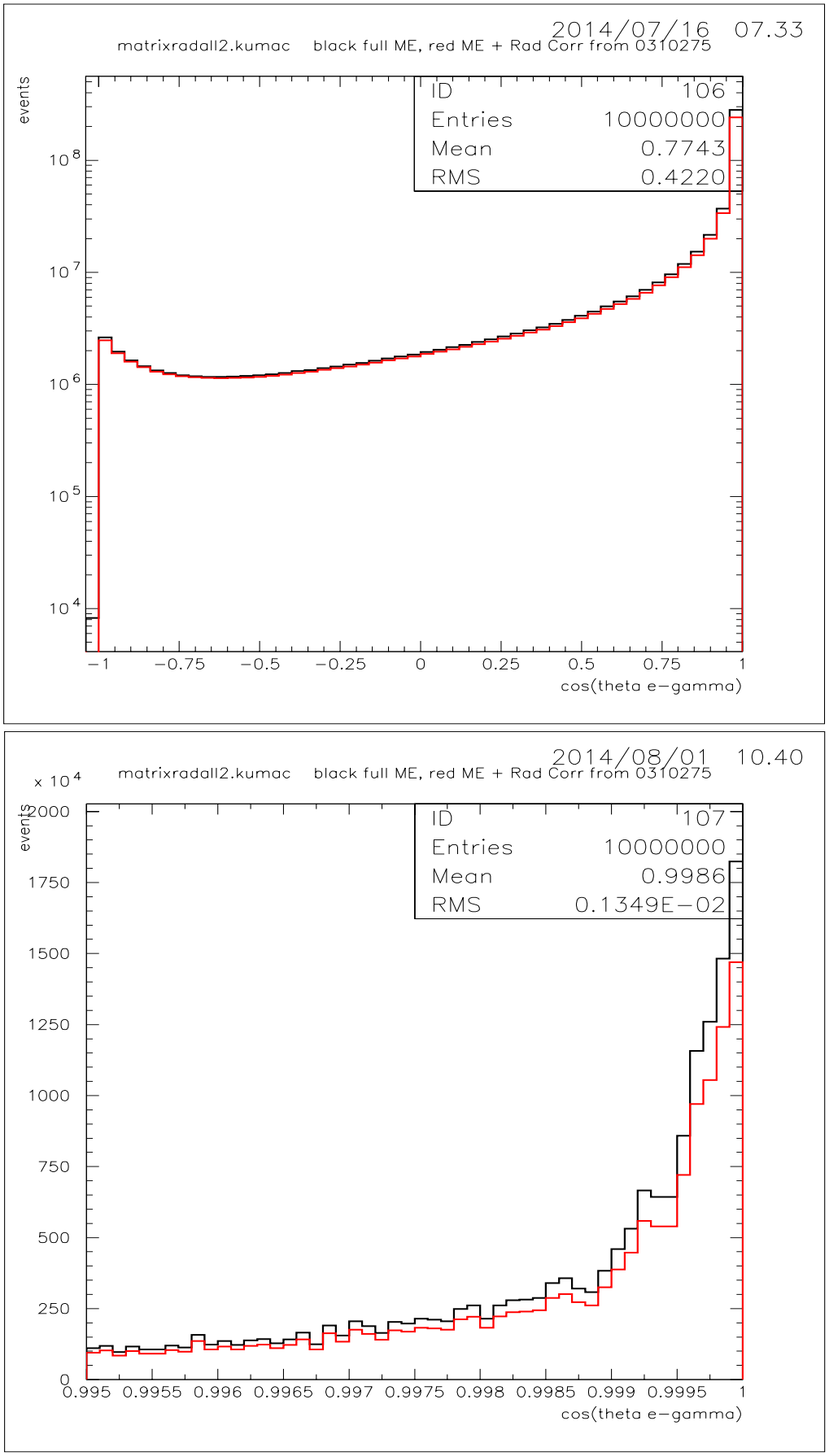

FIG. 9: Comparison between $\cos\theta_{e\gamma}$ distributions with (red) and without (black) radiative corrections, obtained with Eq.2 and Eq.9. Bottom panel is a detail view of the $\cos\theta_{e\gamma} > 0.995$ region.

E. Code listings

This appendix lists our Fortran implementation of $\pi \to l\nu(\gamma)$ generators. Below is the contents of `pilnug.F`, the main module that produces the IB, SD$\pm$ and INT$\pm$ terms and includes an $E_\gamma > 10\,MeV$ infrared cutoff:

```
      SUBROUTINE PILNUG(JPION, LTYPE, MODE)
C-------------------------------------------------------------C
C pi+- -> L NU GAMMA decay                                    C
C                                                             C
C LTYPE:  1=electron, 2=muon                                  C
C MODE:   1=IB, 2=SD+, 3=SD-, 4=INT+, 5=INT-, 6=ALL           C
C                                                             C
C Evgueni.Goudzovski@cern.ch                                  C
C 02/09/2007: original version                                C
C 13/02/2008: form-factor x-dependence introduced             C
C 25/02/2011: INT terms added                                 C
C 03/02/2011: adapted for NA62MC                              C
C                                                             C
C Adapted for pion decay by D.Protopopescu & I.Skillicorn     C
C Dan.Protopopescu@cern.ch  30/06/2014                        C
C For details, see internal note NA62-14-10                   C
C-------------------------------------------------------------C

#include "common_blocks.f"
#include "masses.f"

      INTEGER LTYPE, MODE

      INTEGER i, JPION, JLEP, JGAM, JNU, IDLEP, ffmode
      REAL*8 vec(3), P1(4,4), P2(4,4), P3(4,4), P4(4,4)
      REAL*8 PHI, SINPHI, COSPHI, COSTHE, SINTHE, COSPSI, SINPSI
      REAL*8 MLEP, Eg, Ee, Pe, En, Pgx, Pgy
      REAL*8 plep(4), pgam(4), pnu(4)
      REAL*8 x, y, ymin, rl, f1, f2, wtcomp, wtmax
      REAL*8 Fa, Fv, Fva, Fv0, alpha, fPI
      REAL*8 f_ib, f_sd_p, f_sd_m, f_int_p, f_int_m, f_here, f_tot
      REAL*8 c_sd_p, c_sd_m, c_int_p, c_int_m, c_g, xb, yb, z, rcf
      REAL*8 rc_ib, rc_sd_p, rc_sd_m, rc_int_p, rc_int_m, rc_tot
      REAL*8 pi
      Parameter (pi = 3.141592653589793)

c ........................................

      if (mode.lt.1.or.mode.gt.6)     mode  = 6
      if (ltype.ne.1.and.ltype.ne.2) ltype = 1

      if (ltype.eq.1) MLEP = MEL
      if (ltype.eq.2) MLEP = MMU
      rl = MLEP*MLEP/SQMPI

      if (ltype.eq.1.and.mode.eq.1) wtmax = 5.00e5 !  IB
      if (ltype.eq.1.and.mode.eq.2) wtmax = 4.610  !  SD+
      if (ltype.eq.1.and.mode.eq.3) wtmax = 0.650  !  SD-
      if (ltype.eq.1.and.mode.eq.4) wtmax = 0.036  !  INT+
      if (ltype.eq.1.and.mode.eq.5) wtmax = 275.0  !  INT-
      if (ltype.eq.1.and.mode.eq.6) wtmax = 5.00e5 !  FULL

      if (ltype.eq.2.and.mode.eq.1) wtmax = 1.09e6 !  IB
      if (ltype.eq.2.and.mode.eq.2) wtmax = 0.0001 !  SD+
      if (ltype.eq.2.and.mode.eq.3) wtmax = 0.0001 !  SD-
      if (ltype.eq.2.and.mode.eq.4) wtmax = 0.0025 !  INT+
      if (ltype.eq.2.and.mode.eq.5) wtmax = 0.0013 !  INT-
      if (ltype.eq.2.and.mode.eq.6) wtmax = 1.09e6 !  FULL

c ... Generate (x,y) uniformly over their physical allowed regions
c ... Where x=2Eg/MPI , y=2El/MPI, rl=(MLEP/MPI)^2      ! See PDG reference
c
 1    CONTINUE
      x    = ranf() * (1-rl)
      y    = ranf() * (1+rl)
```

```
      ymin = 1-x+rl/(1-x)
      if (y.lt.ymin) goto 1

      Eg = 0.5 * MPI * x
      if (Eg.lt.0.010) goto 1  ! infrared cutoff Eg > 0.01 GeV optional

c  ------  FORM-FACTORS etc. ------
c
c  Form-factors taken from the PDG Review of Particle Physics
c  http://journals.aps.org/prd/pdf/10.1103/PhysRevD.86.010001, page 34
c
      Fv0 = 0.0254  ! +/- 0.0017
      Fva = 0.10    ! +/- 0.06        - Fv slope parameter
      Fv  = Fv0*(1 + Fva*(1-x))     ! - vector form factor
      Fa  = 0.0119  ! +/- 0.0001      - axial-vector form factor
      fPI = 0.13041 ! from J. Rosner et al. (2012)
      alpha = 1./137.036

c  -------  MATRIX ELEMENT  -------
c
      f1      = 1-y+rl
      f2      = x+y-1-rl                                  ! Notations from PDG reference:
      f_ib    = f1/(x*x*f2)*(x*x+2*(1-x)*(1-rl)-2*x*rl*(1-rl)/f2) ! IB(x,y)
      f_sd_p  = f2*((x+y-1)*(1-x)-rl)                     ! SD+(x,y)
      f_sd_m  = (1-y+rl)*((1-x)*(1-y)+rl)                 ! SD-(x,y)
      f_int_p = f1/x/f2*((1-x)*(1-x-y)+rl)                ! INT+(x,y)
      f_int_m = f1/x/f2*(x*x-(1-x)*(1-x-y)-rl)            ! INT-(x,y)

c ----------  COEFFICIENTS  ----------
c
      c_sd_p = (MPI/fPI)*(MPI/fPI)/4/rl*(Fv+Fa)*(Fv+Fa)
      c_sd_m = (MPI/fPI)*(MPI/fPI)/4/rl*(Fv-Fa)*(Fv-Fa)
      c_int_p = MPI/fPI*(Fv+Fa)
      c_int_m = MPI/fPI*(Fv-Fa)
c     c_g = alpha/(2.*Pi)/(1-rl)/(1-rl)

      f_tot = f_ib + c_sd_p*f_sd_p + c_sd_m*f_sd_m
     >      + c_int_p*f_int_p + c_int_m*f_int_m          ! psi^(0) terms

c  -------  RADIATIVE CORRECTIONS  -------
c See Bystritsky et al. arXiv:hep-ph/0310275v3 (2004-2013), page 14
c
      xb = 1.-x
      yb = 1.-y
      z = x+y-1.
      if(xb.lt.0.00001) xb = 0.00001
      if(yb.lt.0.00001) yb = 0.00001
      if( z.lt.0.00001)  z = 0.00001

      rcf = alpha/(2.*Pi)*(log(y*y/rl) - 1.)
      rc_ib = ((1+xb*xb)/x*x)*(3/2*yb/z+yb/xb -(xb+x*y)/xb**2*log(y)
     >        + 2.*yb/z*log(yb/y)-x*(xb*xb+y**2)/(xb*xb*z)*log(x/z))
      rc_sd_p = xb*(3*z*z/2 + (1-y*y)/2 + yb*(y-2*xb)
     >        + xb*(xb-2*y)*log(y) - xb*xb*yb + 2*z*z*log(yb/y))
      rc_sd_m = xb*(3*yb*yb/2 + (1-y*y)/2. + yb*(y-3.) + (1-2*y)*log(y)
     >        + 2*yb*yb*log(yb/y))
      rc_int_p = (xb/x)*(yb/2 - yb*log(y) - 2*yb*log(yb/y));
      rc_int_m = (1./x)*(-xb*yb/2. + 3.*x*x*yb/(z*2)
     >        + xb*(yb*log(y) + 2*yb*log(yb/y))
     >        + x*x*(yb/x - (xb+x*y)/(xb*xb)*log(y) + 2*yb/z*log(yb/y)
     >        - x*(xb*xb + y*y)/(xb*xb*z)*log(x/z)))

      rc_tot = rcf*(rc_ib + c_sd_p*rc_sd_p + c_sd_m*rc_sd_m
     >        + c_int_p*rc_int_p + c_int_m*rc_int_m)        ! psi^(1) terms

      if (mode.eq.1) f_here = f_ib + rcf*rc_ib
      if (mode.eq.2) f_here = c_sd_p*(f_sd_p + rcf*rc_sd_p)
      if (mode.eq.3) f_here = c_sd_m*(f_sd_m + rcf*rc_sd_m)
      if (mode.eq.4) f_here = c_int_p*(f_int_p + rcf*rc_int_p)
      if (mode.eq.5) f_here = -c_int_m*(f_int_m + rcf*rc_int_m) ! changed sign
```

```
      if (mode.eq.6) f_here = f_tot + rc_tot                ! full calculation

      wtcomp = ranf()*wtmax
      if (wtcomp.gt.f_here) goto 1

c ---- Transform (x,y) into 4-momenta in pion rest frame
c ... Lepton momentum is aligned along the X axis
      Eg  = 0.5 * MPI * x
      Ee  = 0.5 * MPI * y
      En  = MPI - Eg - Ee
      Pe  = sqrt(Ee*Ee - MLEP*MLEP)
      Pgx = -0.5 * (En*En - Eg*Eg - Pe*Pe) / Pe
      Pgx = sqrt(Eg*Eg - Pgx*Pgx)

      plep(1) = Pe
      plep(2) = 0.0
      plep(3) = 0.0
      plep(4) = Ee
      pgam(1) = Pgx
      pgam(2) = Pgy
      pgam(3) = 0.0
      pgam(4) = Eg
      pnu(1)  = -Pe-Pgx
      pnu(2)  = -Pgy
      pnu(3)  = 0.0
      pnu(4)  = En

c ---- Finally, perform a rotation

c ---- For rotation of momenta into a random direction, let us use
c ---- the Euler angles: rotations around z, unrotated x, unrotated z

      DO I = 1, 4
         P1(1, I) = Plep(I)
         P1(2, I) = Pgam(I)
         P1(3, I) = Pnu(I)
      ENDDO

c ---- a) Counterclockwise rotation around Z axis (PHI)
      PHI    = RANF()*2.0*PI
      SINPHI = DSIN(PHI)
      COSPHI = DCOS(PHI)
      DO I = 1, 3
         P2(I,1) =  P1(I,1)*COSPHI + P1(I,2)*SINPHI
         P2(I,2) = -P1(I,1)*SINPHI + P1(I,2)*COSPHI
         P2(I,3) =  P1(I,3)
         P2(I,4) =  P1(I,4)
      ENDDO

c ---- b) Generate uniformly the new direction of Z axis,
c ----    define the corresponding Euler angles THETA, PSI
      CALL GENSPH(VEC)
      COSTHE = VEC(3)/SQRT(VEC(1)**2+VEC(2)**2+VEC(3)**2)
      SINTHE = SQRT((VEC(1)**2+VEC(2)**2)/
     >               (VEC(1)**2+VEC(2)**2+VEC(3)**2))
      COSPSI = VEC(2)/SQRT(VEC(1)**2+VEC(2)**2)
      SINPSI = VEC(1)/SQRT(VEC(1)**2+VEC(2)**2)

c ---- c) Clockwise rotation around X axis (THETA)
      DO I = 1, 3
         P3(I,1) = P2(I,1)
         P3(I,2) = P2(I,2)*COSTHE - P2(I,3)*SINTHE
         P3(I,3) = P2(I,2)*SINTHE + P2(I,3)*COSTHE
         P3(I,4) = P2(I,4)
      ENDDO

c ---- d) Counterclockwise rotation around Z axis (PSI)
      DO I = 1, 3
         P4(I,1) = P3(I,1)*COSPSI - P3(I,2)*SINPSI
         P4(I,2) = P3(I,1)*SINPSI + P3(I,2)*COSPSI
```

```
            P4(I,3) = P3(I,3)
            P4(I,4) = P3(I,4)
         ENDDO

c ---- put the results back into the old vectors
         do i = 1, 4
            Plep(i) = P4(1, I)
            Pgam(i) = P4(2, I)
            Pnu(i)  = P4(3, I)
         enddo

c-----------------------------------------------------------------

c ---- FILL MC PARTICLE LIST
         if (ltype.eq.1) IDLEP = IDELEP
         if (ltype.eq.2) IDLEP = IDMUP
         JLEP = MCADD4GEN(JPION, IDLEP, plep, 0)
         JGAM = MCADD4GEN(JPION, IDGAM, pgam, 0)
         JNU  = MCADD4GEN(JPION, IDNU,  pnu,  0)

c ---- BOOST TO THE LAB-SYSTEM
         CALL DBOOST(P4INI(1,JPION),MPI,plep,plep)
         CALL DBOOST(P4INI(1,JPION),MPI,pgam,pgam)

c ---- FILL MC PARTICLE LIST
         JLEP = MCADD4(JPION, IDLEP, plep)
         JGAM = MCADD4(JPION, IDGAM, pgam)

         RETURN
         END
```

Listed below is `pienug_gatti.F`, which adapts the KLOE code for $\pi \to e\nu\gamma$. This module is roughly 100 times faster than `pilnug.F` because of very efficient x, y sampling:

```
      SUBROUTINE PIENUG_GATTI(JPION, MODE)
C-----------------------------------------------------------------C
C PI+- -> E NU GAMMA DECAY                                        C
C                                                                 C
C IB matrix element: Bijnens,Ecker,Gasser,hep-ph/9209261          C
C Higher-order corrections: C.Gatti, EPJC45 (2006) 417            C
C This wrapper just calls the KLOE generators,                    C
C boosts daughters into lab frame & interfaces to GEANT           C
C E.Goudzovski 3/08/2009, 28/06/2011                              C
C                                                                 C
C modified to pass muon polarization to GEANT4                    C
C by: M.Koval, 14/8/2013, michal.koval@cern.ch                    C
C For details, see internal note NA62-13-09.                      C
C                                                                 C
C Original function: KCH2LNUG_IB(JKAON, LTYPE)                    C
C Adapted for pion decay by D.Protopopescu & I.Skillicorn         C
C Dan.Protopopescu@cern.ch 22/10/2014                             C
C For details, see CERN internal note NA62-14-10                  C
C                                                                 C
C MODE: 1=IB, 2=SD+, 3=SD-, 4=INT+, 5=INT-, 6=ALL                 C
C-----------------------------------------------------------------C

#include "common_blocks.f"
#include "masses.f"

      INTEGER JPION, JELEC, JGAMMA, istat, i
      real*8 PPCM(4,3)
      REAL*8 p4e(4), p4g(4), x, y

      INTEGER PIE2G_GATTI

      if (mode.lt.1.or.mode.gt.6) mode = 6

      istat = PIE2G_GATTI (PPCM, MODE)

      do i=1,4
         p4g(i) = PPCM(i,1)
         p4e(i) = PPCM(i,2)
      enddo

      x = 2.0*p4g(4)/MPI
      y = 2.0*p4e(4)/MPI

C --- FILL MC PARTICLE LIST
      JELEC  = MCADD4GEN(JPION, IDELEP, p4e, 0)
      JGAMMA = MCADD4GEN(JPION, IDGAM, p4g, 0)

C --- BOOST TO THE LAB-SYSTEM
      CALL DBOOST(P4INI(1,JPION),MPI,p4e,p4e)
      if (x.gt.1.0e-10) CALL DBOOST(P4INI(1,JPION),MPI,p4g,p4g)

C --- FILL Pie2 MC PARTICLE LIST
      JELEC = MCADD4(JPION, IDELEP, p4e)
      if (x.gt.1.0e-10) JGAMMA = MCADD4(JPION, IDGAM, p4g)

      RETURN
      END

C -----------------------------------------------------------------

      Function PIE2G_GATTI(PCM, MODE)
C-----------------------------------------------------------------C
C PI+- --> E NU (GAMMA) DECAY                                     C
C Includes IB consistently the with RK definition                 C
C IB matrix element: Bijnens,Ecker,Gasser,hep-ph/9209261          C
C Higher-order corrections: C.Gatti, EPJC45 (2006) 417            C
C Imported from the KLOE library with minimal changes             C
```

```
C (thanks to Tommaso Spadaro)                                    C
C E.Goudzovski 3/08/2009, 28/06/2011                             C
C                                                                C
C Original function: KE2G_IB_KLOE(PCM)                           C
C Adapted for pion decay by D.Protopopescu & I.Skillicorn        C
C Dan.Protopopescu@cern.ch 22/10/2014                            C
C For details, see CERN internal note NA62-14-10                 C
C-----------------------------------------------------------------C

      IMPLICIT NONE

#include "masses.f"

      real*8 PCM(4,3), Amp, Amax
      integer status, PIE2G_GATTI

      real*8 Fa, Fv, Fva, Fv0
      real*8 betae, b, rl
      real*8 x, y, Ctheta, Eg, El
      real*8 rando(2), pb(1), angles(3)
      real*8 g_ib, g_sd_p, g_sd_m, g_int_p, g_int_m
      real*8 c_sd_p, c_sd_m, c_int_p, c_int_m
      real*8 ctg, stg, cpg, spg, cpl, spl
      real*8 RCM(3,2)

C     Parameters
      real*8    pi, alpha, fPI
      Parameter (pi = 3.1415927d+00)
      Parameter (alpha=1.d+00/137.03599968d+00)
      Parameter (fPI = 0.13041)   ! from J. Rosner et al. (2012)
C
C  Form-factors taken from the PDG Review of Particle Physics
C  http://journals.aps.org/prd/pdf/10.1103/PhysRevD.86.010001, page 34
C
      Fv0 = 0.0254  ! +/- 0.0017
      Fva = 0.10    ! +/- 0.06    - Fv slope parameter
C     Fv  = Fv0*(1 + Fva*(1-x)) ! - calculated within loop
      Fa  = 0.0119  ! +/- 0.0001  - axial-vector form factor

C  These need to be calculated for all modes if Amp is rescaled
      if (mode.eq.1) Amax = 1.985   ! IB
      if (mode.eq.2) Amax = 2.418   ! SD+
      if (mode.eq.3) Amax = 0.051   ! SD-
      if (mode.eq.4) Amax = 0.002   ! INT+
      if (mode.eq.5) Amax = 0.014   ! INT-
      if (mode.eq.6) Amax = 2.421   ! Full

C  Lepton mass and max value of the non-peaking factor
      rl = mel*mel/SQMPI

C  Bond factor
      betae= dsqrt(1.d+00-(2.d+00*MPI*mel/(mel**2+MPI**2))**2)
      b = - 2.d+00*alpha/pi*( 1.d+00-dlog((1.d+00+betae)/(1.d+00-betae))
     >     /(2.d+00*betae))

      status = 0
      Do while (status.eq.0)

C  Energy Distribution + y

         CALL RANLUX(rando,2)

C  Extraction of y = (El - Pl*Ctheta)/m_K
         y = rl**(1.d+00 - dble(rando(2)))

C  Photon Energy
         x = (1.d+00 - rl)*dble(rando(1))**(1.d+00/b)
         Eg = x*MPI/2.d+00

C  x-dependant parms
```

```
          Fv  = Fv0*(1 + Fva*(1-x))

C  Lepton energy
          El = (MPI**2+mel**2+2.d+00*MPI*Eg*(y-1.d+00))/(2.d+00*MPI)

C  Ctheta
          if (El.gt.mel) then
             Ctheta = (El -MPI* y)/(dsqrt(El**2-mel**2))
          Else
             goto 654
          Endif

          If (Ctheta.lt.-1.d+00.or.Ctheta.gt.1.d+00) goto 654
C
C  Amplitude split into IB, SD+/-, INT+/- terms
C
          g_ib    = (1.d+00 - y)*(x*x + 2.d+00*(1.d+00 - rl)*
     >         (1.d+00 - x - r/y))
          g_sd_p  = -x*x*x*x*y*y*(rl - y + x*y)
          g_sd_m  = -x*x*x*x*y*(y - 1.d+00)*
     >         (rl - y + x*y - x + 1.d+00)
          g_int_p = -x*x*(y - 1.d+00)*(rl - y + x*y)
          g_int_m = x*x*(y - 1.d+00)*(rl - y + x*y - x)
C
C  Coefficients
C
          c_sd_p = (MPI/fPI)*(MPI/fPI)/4./rl*(Fv+Fa)*(Fv+Fa)
          c_sd_m = (MPI/fPI)*(MPI/fPI)/4./rl*(Fv-Fa)*(Fv-Fa)
          c_int_p = MPI/fPI*(Fv+Fa)
          c_int_m = MPI/fPI*(Fv-Fa)

          if (mode.eq.1) Amp = g_ib
          if (mode.eq.2) Amp = c_sd_p*g_sd_p
          if (mode.eq.3) Amp = c_sd_m*g_sd_m
          if (mode.eq.4) Amp = -c_int_p*g_int_p ! changed sign
          if (mode.eq.5) Amp = c_int_m*g_int_m

          if (mode.eq.6) Amp = g_ib + c_sd_p*g_sd_p
     >                         + c_sd_m*g_sd_m
     >                         + c_int_p*g_int_p
     >                         + c_int_m*g_int_m  ! all terms

C ---- To match original KLOE implementation we would need
C      Amp = (fPI*fPI*MPI*MPI*mel*mel/2) * Amp / 1.0d-9
C      with all Amax parameters recalculated

          If (Amp.gt.Amax) then
             write (*,*) '@ piG:ERROR prob>1', Amp
          Endif

          CALL RANLUX(pb,1)       ! hit or miss
          If (dble(pb(1))*Amax.le.Amp) then
             Status = 1
          Endif

 654      continue
       enddo ! end of "do while"

       CALL RANLUX(angles,3)

       ctg=dble(angles(1))*2.d+00-1.d+00
       stg=dsqrt(1.d+00-ctg**2)
       cpg=dcos(dble(angles(2))*2.d+00*pi)
       spg=dsin(dble(angles(2))*2.d+00*pi)
       cpl=dcos(dble(angles(3))*2.d+00*pi)
       spl=dsin(dble(angles(3))*2.d+00*pi)

C      photon
       RCM(1,1) = 0.d+00
       RCM(2,1) = 0.d+00
```

```
      RCM(3,1) = Eg
209
C     lepton
211   RCM(1,2) = dsqrt(El**2-mel**2)*dsqrt(1.d+00-Ctheta**2)*cpl
      RCM(2,2) = dsqrt(El**2-mel**2)*dsqrt(1.d+00-Ctheta**2)*spl
213   RCM(3,2) = dsqrt(El**2-mel**2)*Ctheta

215 C Rotation

217 C     photon
      PCM(1,1) = cpg*stg*RCM(3,1)
219   PCM(2,1) = spg*stg*RCM(3,1)
      PCM(3,1) = ctg*RCM(3,1)
221   PCM(4,1) = Eg

223 C     lepton
      PCM(1,2) = cpg*ctg*RCM(1,2)-spg*RCM(2,2)+cpg*stg*RCM(3,2)
225   PCM(2,2) = spg*ctg*RCM(1,2)+cpg*RCM(2,2)+spg*stg*RCM(3,2)
      PCM(3,2) = -stg*RCM(1,2)+ctg*RCM(3,2)
227   PCM(4,2) = El

229 C     neutrino
      PCM(1,3) = - PCM(1,1) - PCM(1,2)
231   PCM(2,3) = - PCM(2,1) - PCM(2,2)
      PCM(3,3) = - PCM(3,1) - PCM(3,2)
233   PCM(4,3) = MPI - Eg - El

235   PIE2G_GATTI = 0
      RETURN
237   END
```

Listed below is `pilnu.F`, which implements $\pi \to l\nu$ for the sake of completeness:

```
      SUBROUTINE PILNU(JPION,LTYPE)
C-------------------------------PION-------------------------------C
C                                                                   C
C     Two body decay generator: pi+- --> l+- nu                     C
C     Original code: kch2lnu.F by E.Goudzovski & M.Koval            C
C     Pion version: D.Protopopescu 24/04/2014                       C
C     For details, see internal note NA62-14-08                     C
C                                                                   C
C     Input: LTYPE=1 for e decay, LTYPE=2 for mu decay              C
C-------------------------------------------------------------------C

#include "common_blocks.f"
#include "masses.f"

      INTEGER JPION, LTYPE, IDlep, JLEP, J
      REAL*8  Mlep, Elep, Eneu, Plep, EL(4), VEC(3), POL(3)
      REAL*8  Scale1, Scale2, P0(4)

      if (ltype.eq.1) then
         Mlep  = MEL
         IDlep = IDELEP
      else
         Mlep  = MMU
         IDlep = IDMUP
      endif
      Elep = (SQMPI + Mlep**2) / (2.0*MPI)
      Eneu = (SQMPI - Mlep**2) / (2.0*MPI)
      Plep = sqrt(Elep**2 - Mlep**2)

c --- POSITRON 4-MOMENTUM
      CALL GENSPH(VEC)
      EL(1) = Plep * VEC(1)
      EL(2) = Plep * VEC(2)
      EL(3) = Plep * VEC(3)
      EL(4) = Elep

c --- FILL MC PARTICLE LIST
      JLEP = MCADD4GEN(JPION, IDlep, EL, 0)

c --- BOOST FROM PION FRAME INTO LAB FRAME
      CALL DBOOST(P4INI(1,JPION),MPI,EL,EL)

      if (ltype.eq.1) then
c        --- FILL MC PARTICLE LIST
         JLEP = MCADD4(JPION, IDlep, EL)
      else
c        --- CALCULATE MU+ POLARIZATION IN ITS REST FRAME
         DO J = 1, 4
            P0(J) = P4INI(J,JPION)
         ENDDO

         Scale1 = 2*Mlep/(SQMPI - Mlep**2)
         Scale2 = (-1/(EL(4)+Mlep))*(1+Scale1*(P0(4)+Mlep))

         POL(1) = Scale1*P0(1)+Scale2*EL(1)
         POL(2) = Scale1*P0(2)+Scale2*EL(2)
         POL(3) = Scale1*P0(3)+Scale2*EL(3)

c        --- FILL MC PARTICLE LIST
         JLEP = MCADD4POL3(JPION, IDlep, EL, POL)
      endif

      RETURN
      END
```