WINHAC++: THE OBJECT-ORIENTED MONTE CARLO FOR THE CHARGED-CURRENT DRELL–YAN PROCESS* **

Kamil Sobol

The Marian Smoluchowski Institute of Physics, Jagiellonian University
Reymonta 4, 30-059 Kraków, Poland.

(Received April 29, 2011)

This paper is devoted to the implementation of the generator WINHAC++. This is a new, object-oriented version of the Monte Carlo generator WINHAC written in Fortran, which is used to model the charged-current Drell–Yan processes, taking into account the radiative corrections by exclusive Yennie–Frautschi–Suura exponentiation. At present, the Born level and the final-state radiation (FSR) has been included into the latest stable release. The correctness of the implementation has been confirmed by a series of numerical tests. The current stage of the development includes the implementation of standard event records and the integration with parton shower and hadronization generators, such as Pythia.

DOI:10.5506/APhysPolB.42.1605
PACS numbers: 02.70.Uu, 07.05.Tp, 89.20.Ff

1. Introduction

1.1. Physical background

The process simulated by WINHAC++ is single W-boson production in hadron–hadron collisions with multiphoton radiation [1]. Currently, the generator supports the final-state radiation (FSR) including electroweak corrections, as can be seen in figure 1. The theoretical approach to the problem is to take the YFS-exponentiated cross-section [2] at the partonic level, Eq. (1), and perform a convolution with parton density functions (PDFs), Eq. (3).

* Presented at the Cracow Epiphany Conference on the First Year of the LHC, Cracow, Poland, January 10–12, 2011.

** The work is partly supported by the Programme of the French–Polish Cooperation between IN2P3 and COPIN No. 05-116.

$$\sigma_{\mathrm{YFS}} = \sum_{n=0}^{\infty} \int \frac{d^3q_l}{q_l^0} \frac{d^3q_\nu}{q_\nu^0} \rho_n^{(1)}(p_1, p_2, q_l, q_\nu, k_1, \ldots, k_n)\,, \tag{1}$$

where

$$\begin{aligned} \rho_n^{(1)} \;=\; & e^{Y(Q,q_l;k_s)} \frac{1}{n!} \prod_{i=1}^{n} \frac{d^3k_i}{k_i^0} \tilde{S}(Q, q_l, k_i)\theta\left(k_i^0 - k_s\right) \\ & \times\delta^{(4)}\left(p_1 + p_2 - q_l - q_\nu - \sum_{i=1}^{n} k_i\right) \\ & \times\left[\bar{\beta}_0^{(1)}(p_1, p_2, q_l, q_\nu) + \sum_{i=1}^{n} \frac{\bar{\beta}_1^{(1)}(p_1, p_2, q_l, q_\nu, k_i)}{\tilde{S}(Q, q_l, k_i)}\right], \end{aligned} \tag{2}$$

$$\sigma = \sum_{q,\bar{q}} \int dx_1 dx_2 \Big(f_q^A(x_1) f_{\bar{q}}^B(x_2) + (q \leftrightarrow \bar{q}) \Big) \widehat{\sigma}_{q\bar{q}\to l\bar{\nu}}\,. \tag{3}$$

Details, developed by Płaczek and Jadach, can be found in the paper [3].

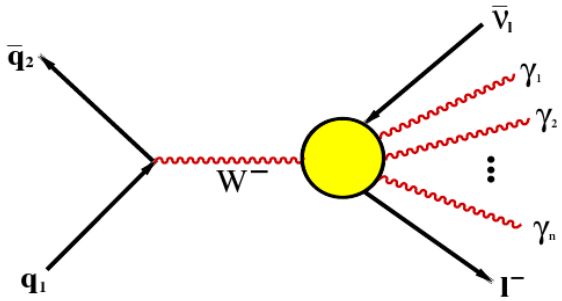


Fig. 1. Multiphoton radiation in W decay [3].

1.2. Motivation

From the physical point of view the modelling of the charged-current Drell–Yan (DY) process is very well motivated. First of all, this phenomenon is very well visible at the LHC. It is a background for new physics searches. It is also considered as a "standard candle" for other processes, this means that other distributions can be normalized using DY and escape from large luminosity errors, *etc.* As will be seen later, taking into account radiative correction has a visible effect on distributions of observables, such as lepton's transverse momentum, W's transverse mass, *etc.* The second reason is a

possibility to improve W mass prediction, which allows to check better the consistency of the Standard Model and obtain a better constraint on the Higgs-boson mass. The third reason is the investigation of the new physics, *e.g.* by extending simulated model with W', KK or some other resonances.

Rewriting and remodelling the existing code to `C++` and developing new features using this platform is motivated by many factors. Firstly, the High Energy Physics (HEP) community decided to migrate to the `C++` platform. There can be found rewritten packages, such as `Pythia` (version 8.x) [4], `Herwig++` [5]. Even the standard event record is implemented now in object-oriented (OO) manner in the `HepMC` package [6]. This trend can be justified by the following arguments. The code of a well-designed OO project is more readable and has smaller volume than the same done in a procedural, fortranic way. Today, it is easier to find `C++` developers who will maintain and develop the code. Nowadays hardware is no more restriction for computing and the speed of programs is not so important. This particularly applies to the Monte Carlo methods, because they easily allow to introduce the parallel computing. The optimizers of `C++` are doing a great job, causing that OO programs are almost as fast as fortranic ones. The important reason is the availability and multiplicity of tools and libraries supporting the `C++` platform, such as integrated development environments (IDE), modeling tools, debuggers, profilers, *etc.* The `C++` community has many members and one can find a solution for his/her problem or ask a question and there is a good chance that he/she will be answered. During development of `WINHAC++` we have used such tools as: `UML` — modeling, `Eclipse` — IDE, `Subversion` — a source code repository, `CMake` — automatization of build, `Doxygen` — source code documentation, `XercesC` — an XML parser library, `Boost` — a multipurpose library, `Valgrind` — a profiler.

2. Algorithm

The generator is based on a Monte Carlo algorithm. This is a result of complicated formula of Eq. (1). The form of this formula, including the sum to the infinity and the $\rho_n^{(1)}$ functions (can be found in [3]), is very unfriendly for classical numerical approaches. Also the convolution formula, Eq. (3), can be interpreted in the Monte Carlo manner giving a satisfactory algorithm.

The algorithm has the following scheme. First of all, simplified (crude) distributions are used to generate some random variables which are used to construct an event. Next, in the opposite way to the simplification, a set of correction weights is computed, as can be seen in Eq. (4). As a result, a population of events is produced and can be used to fill histograms, *etc.* Every event consists of information about generated particles (species

and four-momenta) and, in the case of weighted events, computed weights (unweighted events have just a weight equal to one)

$$w_j^{\text{event}} = \sigma^{\text{crude}} \prod_i w_i^{\text{crude}} w_j^{\text{model}} \,. \tag{4}$$

Eq. (4) describes this algorithm very well. First of all, there is a total crude cross-section (treated as a weight for convenience) which represents a simplified model. It is common for all events generated in one run. Then, there is a product of crude weights which come from quarks generation and construction of kinematics. This product is common to all weights of the event. Finally, there is a model weight which represents calculation of exact matrix elements within some chosen model. By the models I mean different theoretical schemes of obtaining exact matrix elements, *e.g.* Born, QED YFS FSR, EW YFS FSR, fixed order FSR, *etc.* In the case of unweighted events the elimination method is used to convert weighted events into weight = 1 events.

The crude cross-section is constructed in two steps. First of all, at the partonic level, the cross-section, Eq. (1), is replaced by the Breit–Wigner distribution, Eq. (5). This formula is convoluted, Eq. (3), with the chosen PDFs in the numerical way using the `FOAM` package [7, 8], which integrates it (during initialization phase) giving the total crude cross-section, and then it is used to generate the Bjorken xs of quarks in the generation phase.

$$\sigma^{\text{crude}} = \mathcal{N} \frac{\pi}{36} \left(\frac{\alpha_{\text{eff}} V_{ij}^{\text{CKM}}}{\sin^2 \theta_W} \right)^2 \frac{s}{(s - M_W^2)^2 + \gamma(s)} \,, \tag{5}$$

where

$$\gamma(s) = \begin{cases} (\Gamma_W M_W)^2 & \text{fixed width scheme}\,, \\ \left(s \frac{\Gamma_W}{M_W} \right)^2 & \text{running width scheme}\,, \end{cases} \tag{6}$$

$$\alpha_{\text{eff}} = \begin{cases} \alpha & \text{in } \alpha \text{ scheme}\,, \\ \alpha_{G_\mu} = \frac{\sqrt{2} G_\mu M_W^2 \sin^2 \theta_W}{\pi} & \text{in } \mathrm{G}_\mu\text{scheme}\,. \end{cases} \tag{7}$$

During the event generation, this stage provides two crude weights:

- w^{FOAM} — returned by `FOAM` when it generates the Bjorken xs; it corrects the internal numerical calculations performed by this package,
- w^{kin} — it is equal to 0 if some quark is out of the phase space (it happens rarely and is a result of some approximations) or 1 in the opposite case.

In the next phase the construction of kinematics is performed. When radiative corrections are switched off and the Born level is set, then this stage gives no weights because it is accurate. In the opposite, more interesting case the following crude weights are computed:

- $w_{\rm Y}$ — compensates for simplifications in the YFS form factor,
- $w_{\rm S}$ — compensates for simplifications in $\tilde{S}$ — the soft-photon factor,
- $w_{\rm PS}$ — corresponds to integration of the four-momentum conservation δ-function over the phase space,
- w_m — compensates for dropped lepton-mass terms during the generation of photon angles,
- $w_{\rm kin}^{\rm fin} = 0/1$, if the event is out/in the phase space.

Then, the event is constructed. This is the time when model weights are computed. At the Born level, only one model is considered — the Born itself. The model weight is

$$w_{\rm Born}^{\rm model} = \frac{\left(\frac{d\sigma^{(0)}}{d\Omega_l}\right)}{\left(\frac{d\sigma^{\rm crude}}{d\Omega_l}\right)}\,, \tag{8}$$

where

$$\frac{d\sigma^{\rm crude}}{d\Omega_l} = \frac{2}{3}\left(\frac{\alpha_{\rm eff} V_{ij}^{\rm CKM}}{8\sin^2\theta_W}\right)^2 \frac{s(1-Q_W\cos\theta_l)^2}{(s-M_W^2)^2+\gamma(s)}\,, \tag{9}$$

$$\frac{d\sigma^{(0)}}{d\Omega_l} = \frac{1}{8(2\pi)^2 s}\frac{1}{12}\left|\mathcal{M}^{(0)}\right|^2\,. \tag{10}$$

When it comes to radiation the model weights are based on the $\bar{\beta}$ functions, which are present in Eq. (1). The following set of model weights is computed

- $w_{\rm YFS\ \mathcal{O}(\alpha)\ EW}^{\rm model} = \left(\bar{\beta}_{0({\rm EW})}^{(1)} + \bar{\beta}_1^{(1)}\right) \Big/ \left(\frac{d\sigma^{\rm crude}}{d\Omega_l}\right) \Rightarrow$ the best one,
- $w_{\rm YFS\ \mathcal{O}(1)}^{\rm model} = \bar{\beta}_0^{(0)} \Big/ \left(\frac{d\sigma^{\rm crude}}{d\Omega_l}\right)$,
- $w_{\rm YFS\ \mathcal{O}(\alpha)\ QED}^{\rm model} = \left(\bar{\beta}_{0({\rm QED})}^{(1)} + \bar{\beta}_1^{(1)}\right) \Big/ \left(\frac{d\sigma^{\rm crude}}{d\Omega_l}\right)$,
- $w_{\rm LL\ \mathcal{O}(\alpha)\ QED}^{\rm model} = \left(\bar{\beta}_{0({\rm LL})}^{(1)} + \bar{\beta}_{1({\rm LL})}^{(1)}\right) \Big/ \left(\frac{d\sigma^{\rm crude}}{d\Omega_l}\right)$,

where

- $\bar{\beta}_0^{(0)} = \frac{d\sigma^{(0)}}{d\Omega_l}$,
- $\bar{\beta}_{0(\mathrm{QED})}^{(1)} = \bar{\beta}_0^{(0)} \left(1 + \delta^{\mathrm{QED}}\right)$, QED corrections,
- $\bar{\beta}_{0(\mathrm{EW})}^{(1)} = \bar{\beta}_0^{(0)} \left(1 + \delta^{\mathrm{QED}} + \delta^{\mathrm{weak}}\right)$, the full $\mathcal{O}(\alpha)$ electroweak corrections, δ^{weak} delivered by the `SANC` package [9],
- $\bar{\beta}_{0(\mathrm{LL})}^{(1)} = \bar{\beta}_0^{(0)} \left(1 + \delta^{\mathrm{LL}}\right)$, the leading-log approximation,
- $\tilde{\beta}_1^{(1)} = \sum_{i=1}^{n} \frac{\bar{\beta}_1^{(1)}(p_1,p_2,q_l,q_\nu,k_i)}{\tilde{S}(Q,q_l,k_i)}$,
- $\tilde{\beta}_{1(\mathrm{LL})}^{(1)} = \bar{\beta}_0^{(0)} \sum\limits_{i=1}^{n} \frac{z_i^2}{2(1-z_i)}$, $\qquad z_i = \frac{2E_i^\gamma}{E_1+E_2}$.

Further details of these formulas are given in Ref. [3].

The scheme described above is implemented in the version of `WINHAC++` released in June 2010. Currently, a new weight is added to the process. When the event is made up by `WINHAC++`, then it will be passed to another generator, *e.g.* `Pythia8` [4], which will build up the initial-state QCD/QED parton shower and perform hadronization. This external generator should return a weight corresponding to the applied computations, and the events weight will be multiplied by it.

3. Numerical results

A sample of numerical test (from the first stage) can be found in this section. We have performed comparisons between `WINHAC` and `WINHAC++`. These tests were done using a PC-farm and achieving statistics samples of 2×10^9 events per run. Every run had different settings in order to cover as many cases as possible.

The total cross-sections obtained for different processes on the Born level can be found in Table I. This table contains also the relative corrections to these cross-section corresponding to FSR in the YFS exponentiation scheme (including the QED or EW correction level).

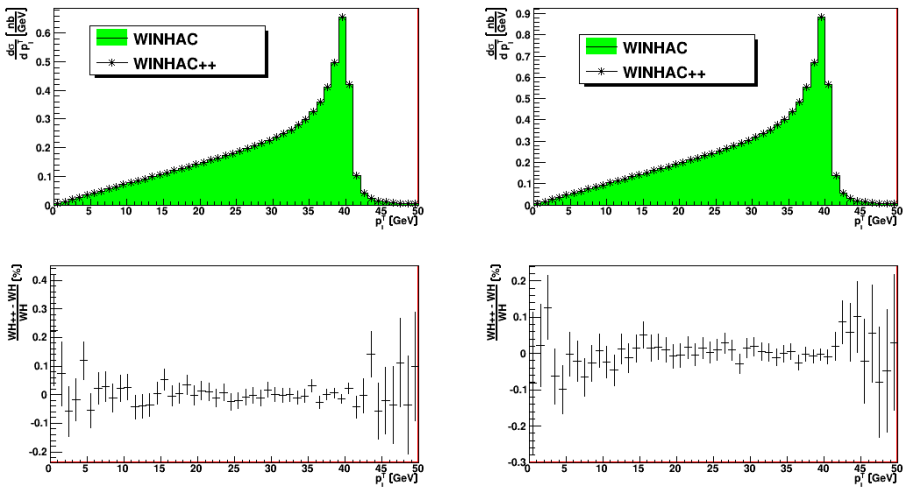
During the tests many distributions of various observables have been obtained. In Figs. 2–4, the distributions of the transverse momentum of a charged lepton are shown at the Born level and with the corrections from multiphoton radiation, the QED and weak ones.

All the tests have shown that the new implementation works well and is consistent with the original one.

TABLE I

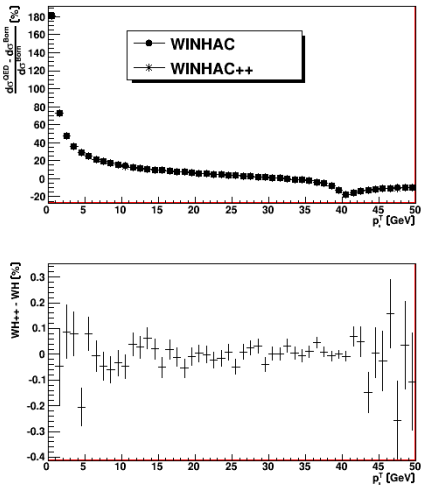
Comparisons of the total cross-sections obtained by `WINHAC` (WH) and `WINHAC++` (WH++) for 2×10^9 events.

Process	Gen.	σ_{Born} [nb]	δ_{QED} [%]	δ_{weak} [%]
$u\bar{d} \to W^+ + X \to e^+\nu_e\,, \mu^+\nu_\mu + X$	WH	15.00753135(2)	−0.0349(5)	−0.2837(7)
	WH++	15.00753132(3)	−0.0345(5)	−0.2837(7)
$pp \to W + X \to e\nu_e\,, \mu\nu_\mu + X$	WH	35.54376(5)	−0.0368(5)	−0.2837(7)
	WH++	35.54384(5)	−0.0390(5)	−0.2830(7)
$pp \to W^- + X \to e^-\bar{\nu}_e + X$	WH	7.574157(12)	−0.0352(6)	−0.2794(8)
	WH++	7.574149(12)	−0.0344(6)	−0.2809(8)
$pp \to W^+ + X \to e^+\nu_e + X$	WH	10.197775(16)	−0.0349(6)	−0.2794(8)
	WH++	10.197802(15)	−0.0369(6)	−0.2794(8)
$pp \to W^- + X \to \mu^-\bar{\nu}_\mu + X$	WH	7.574150(12)	−0.0326(5)	−0.2880(7)
	WH++	7.574116(12)	−0.0317(5)	−0.2887(7)
$pp \to W^+ + X \to \mu^+\nu_\mu + X$	WH	10.197765(16)	−0.0328(5)	−0.2880(7)
	WH++	10.197749(15)	−0.0323(5)	−0.2885(7)
$pp \to W^- + X \to \tau^-\bar{\nu}_\tau + X$	WH	7.572035(12)	−0.0793(5)	−0.2928(7)
	WH++	7.572021(12)	−0.0796(5)	−0.2926(7)
$pp \to W^+ + X \to \tau^+\nu_\tau + X$	WH	10.194957(16)	−0.0799(5)	−0.2928(7)
	WH++	10.194945(15)	−0.0799(5)	−0.2938(7)

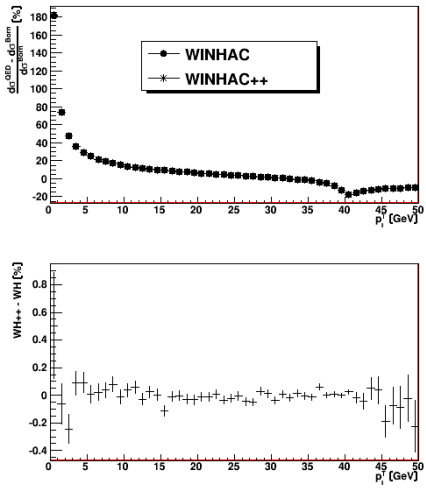


(a) $pp \to W^- + X \to e^-\bar{\nu}_e + X$ (b) $pp \to W^+ + X \to e^+\nu_e + X$

Fig. 2. Distributions of the transverse momentum of the electron (a) and the positron (b) from W decays at the Born level.

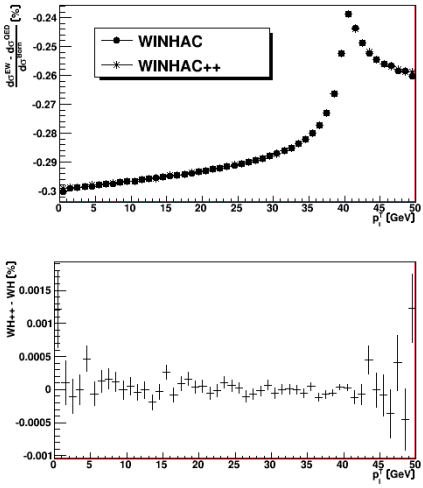


(a) $pp \to W^- + X \to e^- \bar{\nu}_e + X$

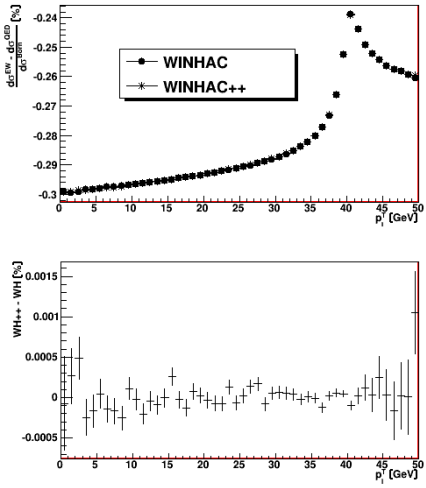


(b) $pp \to W^+ + X \to e^+ \nu_e + X$

Fig. 3. Distributions of the transverse momentum of the electron (a) and the positron (b) from W decays for the relative QED corrections.



(a) $pp \to W^- + X \to e^- \bar{\nu}_e + X$



(b) $pp \to W^+ + X \to e^+ \nu_e + X$

Fig. 4. Distributions of the transverse momentum of the electron (a) and the positron (b) from W decays for the relative electroweak corrections.

4. Summary and outlook

The object-oriented Monte Carlo event generator `WINHAC++` is under development. This process is divided into four stages. The first stable release took place in June 2010 as a result of the first stage. It has the following features: the Born level, multiphoton final-state radiation within the $\mathcal{O}(\alpha)$ YFS exclusive exponentiation scheme, the $\mathcal{O}(\alpha)$ electroweak corrections in FSR, the interface to parton distribution functions (PDFs) from the `LHAPDF` library (with the help of the `FOAM` package). This implementation was checked by high-statistics numerical tests (examples in the previous section).

Currently, at the second stage, the integration with the standard event records (`Les Houches Event Accord` [10] and `HepMC` [6]) has been done. This was immediately followed by applying the QCD/QED parton shower and hadronization (by `Pythia8` [4]), using the above interfaces. The other work was done in the field of optimization (thanks to profiler results delivered by P. Stecko) and currently the speed factor of 1.4 (`C++` *vs.* `Fortran`) has been achieved.

At the third stage we wish to include QED interferences with the initial-state radiation (ISR), the full $\mathcal{O}(\alpha)$ electroweak corrections. The feature of polarized W bosons will be also included at this stage. After finishing this part of development, `WINHAC++` will cover all the functionalities of its base program — `WINHAC`.

The last, fourth stage will involve some new research. We plan to develop $\mathcal{O}(\alpha^2)$ QED FSR, add some new resonances (W', KK, ...), try to use the `KRKMC` package (parton shower) developed by Jadach *et al.* [11]. This stage is open for new ideas and could change in time.

The second stage is almost ready for testing. After completing tests, we plan to issue a new stable release.

`WINHAC++` has a dedicated Web page [12]. One can find there the releases and other resources, such as documentation or related papers. The current sources and its history (since June 2010) can be found in the `SVN` repository located on the `Google Code` site [13].

I would like to thank people involved in the `WINHAC++` project: Wiesław Płaczek, Andrzej Siódmok and Piotr Stecko. Many thanks for your contribution during design sessions and your research which is connected with the project. Especially I wish to thank Wiesław Płaczek for supervising the project and gathering the team together.

REFERENCES

[1] S.D. Drell, T.-M. Yan, *Phys. Rev. Lett.* **25**, 316 (1970).

[2] D. Yennie, S. Frautschi, H. Suura, *Ann. Phys. (NY)* **13**, 379 (1961).

[3] W. Płaczek, S. Jadach, *Eur. Phys. J.* **C29**, 325 (2003).

[4] T. Sjostrand, S. Mrenna, P.Z. Skands, *Comput. Phys. Commun.* **178**, 852 (2008) [arXiv:0710.3820v1 [hep-ph]].

[5] S. Gieseke *et al.*, *J. High Energy Phys.* **0402**, 005 (2004) [arXiv:hep-ph/0311208v2].

[6] M. Dobbs, J.B. Hansen, *Comput. Phys. Commun.* **134**, 41 (2001).

[7] S. Jadach, *Comput. Phys. Commun.* **152**, 55 (2003).

[8] S. Jadach, P. Sawicki, *Comput. Phys. Commun.* **177**, 441 (2007).

[9] D. Bardin *et al.*, *Acta Phys. Pol. B* **40**, 75 (2009).

[10] J. Alwall *et al.*, *Comput. Phys. Commun.* **176**, 300 (2007), [arXiv:hep-ph/0609017v1].

[11] S. Jadach, M. Skrzypek, A. Kusina, M. Slawinska, *PoS* **RADCOR2009**, 069 (2010) [arXiv:1002.0010v1 [hep-ph]], and refs. therein.

[12] K. Sobol, WINHAC++ Home Page, http://th.if.uj.edu.pl/~sobol/index.php?action=winhac

[13] K. Sobol, WINHAC++ @ Google Code, http://code.google.com/p/winhacplusplus/