`Circe` Version 1.02β: Beam Spectra for Simulating Linear Collider Physics*

Thorsten Ohl†

University of Würzburg
Am Hubland D-97089 Würzburg
Germany

IKDA 96/13-rev
hep-ph/9607454-rev
`prelude.nw 66 2002-03-28 17:00:10Z ohl`
`circe.nw 67 2002-03-28 17:13:06Z ohl`
`minuit.nw 58 1999-04-15 08:42:06Z ohl`
`postlude.nw 42 1996-08-08 10:38:00Z ohl`
July 1996 (expanded September 1996)

Abstract

I describe parameterizations of realistic $e^{\pm}$- and γ-beam spectra at future linear e^+e^--colliders. Emphasis is put on simplicity and reproducibility of the parameterizations, supporting reproducible physics simulations. The parameterizations are implemented in a library of distribution functions and event generators.

Contents

---

*Supported by Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie, Germany.

†e-mail: `ohl@physik.uni-wuerzburg.de`

Program Summary:

- **Title of program:** `Circe`, Version 1.02β (September 1996)
- **Program obtainable** by anonymous `ftp` from the host `crunch.ikp.physik.th-darmstadt.de` in the directory `pub/ohl/circe`.
- **Licensing provisions:** Free software under the GNU General Public License.
- **Programming language used:** Fortran77
- **Number of program lines in distributed program, including test data, etc.:** ≈ 1100 (excluding comments)
- **Computer/Operating System:** Any with a Fortran77 programming environment.
- **Memory required to execute with typical data:** Negligible on the scale of typical applications calling the library.
- **Typical running time:** A small fraction (typically a few percent) of the running time of applications calling the library.
- **Purpose of program:** Provide simple and reproducible, yet realistic, parameterizations of the $e^{\pm}$- and γ-beam spectra for linear colliders.
- **Nature of physical problem:** The intricate beam dynamics in the interaction region of a high luminosity linear collider at $\sqrt{s} = 500\text{GeV}$ result in non-trivial energy spectra of the scattering electrons, positrons and photons. Physics simulations require simple and reproducible, yet realistic, parameterizations of these spectra.
- **Method of solution:** Parameterization, curve fitting, Monte Carlo event generation.
- **Keywords:** Event generation, beamstrahlung, linear colliders.

1 Introduction

Despite the enormous quantitative success of the electro-weak standard model up to energies of 200GeV, neither the nature of electro-weak symmetry breaking (EWSB) nor the origin of mass are understood.

From theoretical considerations, we know that clues to the answer of these open questions are hidden in the energy range below $\Lambda_{\text{EWSB}} = 4\pi v \approx 3.1\text{TeV}$. Either we will discover a Higgs particle in this energy range or signatures for a strongly interacting EWSB sector will be found. Experiments at CERN's Large Hadron Collider (LHC) will shed a first light on this regime in the next decade. In the past is has been very fruitful to complement experiments at high energy hadron colliders with experiments at e^+e^--colliders. The simpler initial state allows more precise measurements with smaller theoretical errors. Lucid expositions of the physics opportunities of high energy e^+e^- colliders with references to the literature can be found in [1].

However, the power emitted by circular storage rings in form of synchrotron radiation scales like $(E/m)^4/R^2$ with the energy and mass of the particle and the radius of the ring. This cost becomes prohibitive after LEP2 and a Linear Collider (LC) has to be built instead.

Unfortunately, the "interesting" hard cross sections scale like $1/s$ with the square of the center of mass energy and a LC will have to operate at extremely high luminosities in excess of $10^{33}\text{cm}^{-2}\text{s}^{-1}$. To achieve such luminosities, the bunches of electrons and positrons have to be very dense. Under these conditions, the electrons undergo acceleration from strong electromagnetic forces from the positron bunch (and vice versa). The resulting synchrotron radiation is called *beamstrahlung* [2] and has a strong effect on the energy spectrum $D(x_1, x_2)$ of the colliding particles. This changes the observable e^+e^- cross sections

$$\frac{d\sigma_0^{e^+e^-}}{d\Omega}(s) \to \frac{d\sigma^{e^+e^-}}{d\Omega}(s) = \int_0^1 dx_1 dx_2\, D_{e^+e^-}(x_1, x_2; \sqrt{s}) J(\Omega', \Omega) \frac{d\sigma_0^{e^+e^-}}{d\Omega'}(x_1 x_2 s) \tag{1a}$$

and produces luminosity for $e^\pm\gamma$ and $\gamma\gamma$ collisions:

$$\frac{d\sigma^{e^\pm\gamma}}{d\Omega}(s) = \int_0^1 dx_1 dx_2\, D_{e^\pm\gamma}(x_1, x_2; \sqrt{s}) J(\Omega', \Omega) \frac{d\sigma_0^{e^\pm\gamma}}{d\Omega'}(x_1 x_2 s) \tag{1b}$$

$$\frac{d\sigma^{\gamma\gamma}}{d\Omega}(s) = \int_0^1 dx_1 dx_2\, D_{\gamma\gamma}(x_1, x_2; \sqrt{s}) J(\Omega', \Omega) \frac{d\sigma_0^{\gamma\gamma}}{d\Omega'}(x_1 x_2 s) \tag{1c}$$

Therefore, simulations of the physics expected at a LC need to know the spectra of the $e^\pm$ and γ beams precisely.

Microscopic simulations of the beam dynamics are available (e.g. `ABEL`[3], `CAIN`[4] and `Guinea-Pig`[5]) and their predictions are compatible with each other. But they require too much computer time and memory for direct use in physics programs. `Circe` provides a fast and simple parameterization of the results from these simulations. Furthermore, even if the computational cost of the simulations would be negligible, the input parameters for microscopic simulations are not convenient for particle physics applications. Due to the highly

	`SBAND`	`TESLA`	`XBAND`	`SBAND`	`TESLA`	`XBAND`
E/GeV	250	250	250	500	500	500
$N_{\mathrm{particles}}/10^{10}$	1.1	3.63	0.65	2.9	1.8	0.95
$\epsilon_x/10^{-6}\mathrm{mrad}$	5	14	5	10	14	5
$\epsilon_y/10^{-6}\mathrm{mrad}$	0.25	0.25	0.08	0.1	0.06	0.1
β_x^*/mm	10.98	24.95	8.00	32	25	10.00
β_y^*/mm	0.45	0.70	0.13	0.8	0.7	0.12
σ_x/nm	335	845	286	571.87	598.08	226
σ_y/nm	15.1	18.9	4.52	9.04	6.55	3.57
$\sigma_z/\mu\mathrm{m}$	300	700	100	500	500	125
f_{rep}	50	5	180	50	5	180
n_{bunch}	333	1135	90	125	2270	90

Table 1: Accelerator parameters for three typical designs at $\sqrt{s} = 500\mathrm{GeV}$ and $\sqrt{s} = 1\mathrm{TeV}$. The resulting distributions are shown in figure 1. The design efforts are currently concentrated on a 350GeV-800GeV LC. Therefore the Tesla parameters for 1TeV are slightly out of date.

non-linear beam dynamics, the optimization of LC designs is a subtle art [6], that is best practiced by the experts. Furthermore, particle physics applications need benchmarking and easily reproducible parameterizations are required for this purpose.

The parameterizations in `Circe` are not based on approximate solutions (cf. [7]) of the beamstrahlung dynamics. Instead, they provide a "phenomenological" description of the results from full simulations. The parameterizations are as simple as possible while remaining consistent with basic physical principles:

1. *positivity:* the distribution functions $D(x_1, x_2)$ *must not* be negative in the physical region $[0,1] \times [0,1]$.
2. *integrability:* the definite integral of the distribution functions over the physical region $[0,1] \times [0,1]$ *must* exist, even though the distributions can have singularities.

This paper is organized as follows: I start in section 2 with a discussion of the input for the microscopic simulations. In section 3 I describe the usage of the `Circe` library and in section 4 I discuss some technical details of the implementation. After discussing the parameterizations available in version 1.02β in section 5, I conclude in section 8.

2 Parameters

The microscopic simulation program `Guinea-Pig` [5] used for the current version of the parameterizations in `Circe` simulates the passage of electrons through a

Figure 1: Version 1, revision 1996 09 02 of the factorized $e^{\pm}$- and γ-distributions at $\sqrt{s} = 500\text{GeV}$ and $\sqrt{s} = 1\text{TeV}$ in a doubly logarithmic plot. The accelerator parameters are taken from table 1.

	`TESLA`	`TESLA`	`TESLA`
E/GeV	175	250	400
$N_{\text{particles}}/10^{10}$	3.63	3.63	3.63
$\epsilon_x/10^{-6}\text{mrad}$	14	14	14
$\epsilon_y/10^{-6}\text{mrad}$	0.25	0.25	0.1
β_x^*/mm	25.00	24.95	15.00
β_y^*/mm	0.70	0.70	0.70
σ_x/nm	1010.94	845	668.67
σ_y/nm	22.6	18.9	9.46
$\sigma_z/\mu\text{m}$	700	700	700
f_{rep}	5	5	5
n_{bunch}	1135	1135	1135

Table 2: Accelerator parameters for the Tesla design at three planned [8] energies. The resulting distributions are shown in figure 2.

	High-$\mathcal{L}$	Low-$\mathcal{L}$	Low-ϵ_y
E/GeV	400	400	400
$N_{\text{particles}}/10^{10}$	3.63	3.63	1.800
$\epsilon_x/10^{-6}\text{mrad}$	14	14	12
$\epsilon_y/10^{-6}\text{mrad}$	0.1	0.25	0.025
β_x^*/mm	15.00	25.00	25.00
β_y^*/mm	0.70	0.70	0.50
σ_x/nm	668.67	700.00	
σ_y/nm	9.46		
$\sigma_z/\mu\text{m}$	700	700	500
f_{rep}	5	5	3
n_{bunch}	1135	1135	2260

Table 3: Variant accelerator parameters for the Tesla design at 800 Gev.

Figure 2: Version 1, revision 1996 09 02 of the factorized $e^{\pm}$- and γ-distributions for Tesla in a doubly logarithmic plot. The accelerator parameters are taken from table 2.

	`TESLA`	`TESLA`
E/GeV	250	400
$N_{\mathrm{particles}}/10^{10}$	2	1.40
$\epsilon_x/10^{-6}\mathrm{m\ rad}$	10	8
$\epsilon_y/10^{-6}\mathrm{m\ rad}$	0.03	0.01
β_x^*/mm	15.00	15.00
β_y^*/mm	0.40	0.30
σ_x/nm	553	391
σ_y/nm	5	2
$\sigma_z/\mu\mathrm{m}$	400	300
f_{rep}	5	3
n_{bunch}	2820	4500

Table 4: Accelerator parameters for a high luminosity Tesla design at two planned [8] energies. The resulting distributions are shown in figure 3.

Figure 3: Version 5, revision 1998 05 05 of the factorized $e^{\pm}$- and γ-distributions for a high luminosity Tesla in a doubly logarithmic plot. The accelerator parameters are taken from table 4.

Figure 4: *Experimental:* Version 1, revision 0 of the factorized e^{-}- and γ-distributions for Tesla-$e^{-}e^{-}$ in a doubly logarithmic plot. The accelerator parameters are taken from table 2 and have *not* been endorsed for use in an $e^{-}e^{-}$-machine yet!.

bunch of electrons (and vice versa). It takes the following accelerator parameters as input:

E : the energy of the particles before the beam-beam interaction.

$N_{\textbf{particles}}$: the number of particles per bunch.

$\epsilon_{x,y}$: the normalized horizontal and vertical emittances.

$\beta^*_{x,y}$: the horizontal and vertical beta functions.

$\sigma_{x,y,z}$: the horizontal, vertical and longitudinal beam size. A Gaussian shape is used for the charge distribution in the bunches.

$f_{\textbf{rep}}$: the repetition rate.

$n_{\textbf{bunch}}$: the number of bunches per train.

The transversal beam sizes, beta functions and normalized emittances for relativistic particles are related by

$$\beta^*_{x,y} = \frac{\sigma^2_{x,y}}{\epsilon_{x,y}} \frac{E}{m_e} \tag{2}$$

The parameters used in the most recent revision of the parameterizations are collected in tables 1 and 2. The resulting factorized electron/positron and photon distributions in version 1 of the parameterizations are depicted in figures 1 and 2.

The most important purpose of `Circe` is to map the manifold of possible beam spectra for the NLC to a *finite* number of *reproducible* parameterizations. The distributions

$$D^{\alpha\nu\rho}_{p_1 p_2}(x_1, x_2; \sqrt{s}) \tag{3}$$

provided by `Circe` are indexed by three integers

α : the *accelerator design class:* currently there are three options: S-band [9], Tesla [8], X-band [10, 11]. More variety will be added later, in particular the $e^- e^-$ mode and the $e^-\gamma$ and $\gamma\gamma$ laser backscattering modes of these designs.

ν : the *version of the parameterization:* over the years, the form of the parameterizations can change, either because better approximations are found or because new simulation programs become available. All versions will remain available in order to be able to reproduce calculations.

ρ : the *revision date for the parameterization:* a particular parameterization can contain bugs, which will be fixed in subsequent revisions. While only the most recent revision should be used for new calculations, old revisions will remain available in order to be able to reproduce calculations.

The continuous parameter $\sqrt{s}$ in (3) is misleading, because accelerator parameters have been optimized for discrete values of the energy. Therefore the distributions are not available for all values of $\sqrt{s}$.

The usage of the distributions in application programs is discussed in section 3.1. `Circe` provides for each of the distributions a non-uniform random variate generator, that generates energy fractions according to the distributions. The usage of these generators is discussed in section 3.2.

3 Usage

3.1 Distributions

A generic interface to all distributions $D_{p_1p_2}(x_1,x_2)$ is given by the `circe` function

9a ⟨*API documentation* 9a⟩≡

```
double precision circe, d, x1, x2
integer p1, p2
d = circe (x1, x2, p1, p2)
```

Uses `circe` 24d.

where the energy fractions are specified by $x_{1,2}$ and the particles $p_{1,2}$ are identified by their standard Monte Carlo codes:[13]

9b ⟨*Particle codes* 9b⟩≡

```
integer ELECTR, POSITR, PHOTON
parameter (ELECTR =  11)
parameter (POSITR = -11)
parameter (PHOTON =  22)
```

Defines:
  `ELECTR`, used in chunks 18b, 24d, 63a, 67b, and 68a.
  `PHOTON`, used in chunks 24d, 63a, 67b, 68a, and 74.
  `POSITR`, used in chunks 18c and 68a.

The distributions can have integrable singularities at the end points, therefore the calling functions *must not* evaluate them at the endpoints 0 and 1. This is usually not a problem, since standard mapping techniques (cf. (10) below) will have to be used to take care of the singularity anyway. Nevertheless, all applications should favor open quadrature formulae (i.e. formulae not involving the endpoints) over closed formulae. The distributions are guaranteed to vanish unless $0 < x_{1,2} < 1$, with two exceptions. Firstly, the value -1 allows to pick up the integral of the continuum contribution:

$$D_{p_1p_2}(-1,x_2) = \lim_{\epsilon\to+0}\int_\epsilon^{1-\epsilon} dx_1\, D_{p_1p_2}(x_1,x_2) \tag{4a}$$

$$D_{p_1p_2}(x_1,-1) = \lim_{\epsilon\to+0}\int_\epsilon^{1-\epsilon} dx_2\, D_{p_1p_2}(x_1,x_2) \tag{4b}$$

$$D_{p_1p_2}(-1,-1) = \lim_{\epsilon\to+0}\int_\epsilon^{1-\epsilon} dx_1 dx_2\, D_{p_1p_2}(x_1,x_2) \tag{4c}$$

The other exception is that the strength of δ-function contributions at the endpoint can be picked up from the value at this endpoint:

$$D_{e^+e^-}(x_1,x_2) = D_{e^+e^-}(1,1)\delta(1-x_1)\delta(1-x_2) + \text{smooth and single } \delta \tag{5a}$$

$$D_{e^\pm\gamma}(x_1,x_2) = D_{e^\pm\gamma}(1,x_2)\delta(1-x_1) + \text{smooth} \tag{5b}$$

$$D_{\gamma e^\pm}(x_1,x_2) = D_{\gamma e^\pm}(x_1,1)\delta(1-x_2) + \text{smooth} \tag{5c}$$

The use of these special values is demonstrated in an example in section 3.1.1 below.

The distributions are normalized such that

$$\lim_{\epsilon\to+0}\int_{-\epsilon}^{1+\epsilon} dx_1 dx_2\, D_{e^+e^-}(x_1,x_2) = 1. \tag{6}$$

and the nominal e^+e^--luminosity of the currently active accelerator design can be retrieved from the database with the subroutine `circel`. The value is given in units of

$$\mathrm{fb}^{-1}\upsilon^{-1} = 10^{32}\mathrm{cm}^{-2}\mathrm{sec}^{-1} \tag{7}$$

where $\upsilon = 10^7\mathrm{sec} \approx \mathrm{year}/\pi$ is an "effective year" of running with about 30% up-time.

10a ⟨*API documentation* 9a⟩+≡

```
        double precision lumi
        call circel (lumi)
```

Uses `circel` 34b.

A particular parameterization is selected by the `circes` function:

10b ⟨*API documentation* 9a⟩+≡

```
        double precision x1m, x2m, roots
        integer acc, ver, rev, chat
        call circes (x1m, x2m, roots, acc, ver, rev, chat)
```

Uses `circes` 25a.

The parameter `roots` corresponds to the nominal center of mass energy $\sqrt{s}/\mathrm{GeV}$ of the collider. Currently $\sqrt{s} = 350\mathrm{GeV}, 500\mathrm{GeV}, 800\mathrm{GeV}, 1\mathrm{TeV}$ (i.e. `350D0`, `500D0`, `800D0` and `1000D0`) are supported. Application programs can *not* assume that energy values are interpolated. For convenience, e.g. in top threshold scans around 350GeV, a small interval around the supported values will be accepted as synonymous with the central value, but a warning will be printed. Section 5 should be consulted for the discrete values supported by a particular version of the parameterizations. Negative values of `roots` will keep the currently active value for $\sqrt{s}$.

The parameters `x1m` and `x2m` will set thresholds $x_{1,\mathrm{min}}$ and $x_{2,\mathrm{min}}$ for the event generation in the routines described in section 3.2.

The parameter `acc` selects the accelerator design. Currently the following accelerator codes are recognized:

10c ⟨*Accelerator codes* 10c⟩≡

```
        integer SBAND, TESLA, XBAND
        parameter (SBAND  =  1, TESLA  =  2, XBAND  =  3)
        integer JLCNLC
        parameter (JLCNLC =  3)
        integer SBNDEE, TESLEE, XBNDEE
        parameter (SBNDEE =  4, TESLEE =  5, XBNDEE =  6)
        integer NLCH
        parameter (NLCH =  7)
        integer NACC
        parameter (NACC = 7)
```

Defines:
`NACC`, used in chunks 28, 33, 34a, 37–39, 44b, 46c, 48c, 51c, 54c, and 58c.
`SBAND`, used in chunks 28c, 33, 36–40, 42, 78b, 91a, and 92a.
`TESLA`, used in chunks 26f, 28c, 33, 36–38, 40, 42–50, 52, 54d, 55a, 58d, 59b, 78, and 91–93.
`XBAND`, used in chunks 33, 36–40, 42, 50, 91a, and 92a.

The `ver` parameter is used to determine the version as follows:

`ver` > 0 : a frozen version which is documented in section 5. For example, version 1 is a family of factorized Beta distributions: $D(x_1, x_2) \propto x_1^{a_1}(1-x_1)^{b_1}x_2^{a_2}(1-x_2)^{b_2}$.

`ver` $= 0$: the latest experimental version, which is usually not documented and can change at any time without announcement.

`ver` < 0 : keep the currently active version.

The `rev` parameter is used to determine the revision of a version as follows:

`rev` > 0 : a frozen revision which is documented in section 5. The integer `rev` is constructed from the date as follows: $\texttt{rev} = 10^4 \cdot \mathrm{year} + 10^2 \cdot \mathrm{month} + \mathrm{day}$, where the year is greater than 1995. Since Fortran77 ignores whitespace, it can be written like `1996 07 11` for readability. If there is no exact match, the most recent revision before the specified date is chosen.

`rev` $= 0$: the most recent revision.

`rev` < 0 : keep the currently active revision.

Finally, the parameter `chat` controls the "chattiness" of `circe`. If it is 0, only error messages are printed. If it is 1, the parameters in use are printed whenever they change. Higher values of `chat` can produce even more diagnostics.

In addition to the generic interface `circe`, there are specialized functions for particular particle distributions. Obviously

$$D^{\alpha\nu\rho}_{e^\pm\gamma}(x_1, x_2, s) = D^{\alpha\nu\rho}_{\gamma e^\pm}(x_2, x_1, s) \tag{8}$$

and there are three independent functions $D_{e^-e^+}$, $D_{e^-\gamma}$ and $D_{\gamma\gamma}$ for the e^+e^- colliders with reasonable mnemonics:

11 ⟨*API documentation* 9a⟩+≡

```
double precision circee, circeg, circgg
d = circee (x1, x2)
d = circeg (x1, x2)
d = circgg (x1, x2)
```

Uses `circee` 34c, `circeg` 35a, and `circgg` 35c.

Calling the latter three functions is marginally faster in the current implementation, but this can change in the future.

3.1.1 Example

For clarification, let me give a simple example. Imagine we want to calculate the integrated production cross section

$$\sigma_X(s) = \int dx_1 dx_2\, \sigma_{e^+e^-\to X}(x_1 x_2 s) D_{e^+e^-}(x_1, x_2, s) \tag{9}$$

Since the distributions are singular in the $x_{1,2} \to 1$ limit, we have to map away this singularity with

$$x \to t = (1-x)^{1/\eta} \tag{10a}$$

Therefore

$$\int_0^1 dx\, f(x) = \int_0^1 dt\, \eta t^{\eta-1} f(1-t^\eta) \tag{10b}$$

with η sufficiently large to give the integrand a finite limit at $x \to 1$. If f diverges like a power $f(x) \propto 1/(1-x)^\beta$, this means $\eta > 1/(1-\beta)$.

As a specific example, let us "measure" a one particle s-channel exchange cross section

$$\sigma(s) \propto \frac{1}{s} \tag{11}$$

12a ⟨`sample.f` 12a⟩≡

```
        double precision function sigma (s)
        implicit none
        double precision s
        sigma = 1d0 / s
        end
```

I will present the example code in a bottom-up fashion, which should be intuitive and is described in some more detail in appendix A. Assuming the existence of a one- and a two-dimensional Gaussian integration function `gauss1` and `gauss2`,[1] we can perform the integral as follows:

12b ⟨*Gauss integration* 12b⟩≡

```
        s = sigma (1d0) * circee (1d0, 1d0)
       $   + gauss1 (d1, 0d0, 1d0, EPS)
       $   + gauss1 (d2, 0d0, 1d0, EPS)
       $   + gauss2 (d12, 0d0, 1d0, 0d0, 1d0, EPS)
        write (*, 1000) 'delta(sigma) (Gauss) =', (s-1d0)*100d0
   1000 format (1X, A22, 1X, F6.2, '%')
```

Uses `circee` 34c, `gauss1` 76d, and `gauss2` 77a.

Note how the four combinations of continuum and δ-peak are integrated separately, where you have to use three auxiliary functions `d1`, `d2` and `d12`. The continuum contribution, including the Jacobian:

12c ⟨`sample.f` 12a⟩+≡

```
        double precision function d12 (t1, t2)
        implicit none
        double precision t1, t2, x1, x2, sigma, circee
        ⟨EPS & PWR 13b⟩
        x1 = 1d0 - t1**PWR
        x2 = 1d0 - t2**PWR
        d12 = PWR*PWR * (t1*t2)**(PWR-1d0)
       $        * sigma (x1*x2) * circee (x1, x2)
        end
```

Uses `circee` 34c.

the first product of continuum and δ-peak:

12d ⟨`sample.f` 12a⟩+≡

```
        double precision function d1 (t1)
        implicit none
```

---

[1] They are provided in the example program `sample.f`.

```
double precision t1, x1, sigma, circee
⟨EPS & PWR 13b⟩
x1 = 1d0 - t1**PWR
d1 = PWR * t1**(PWR-1d0) * sigma (x1) * circee (x1, 1d0)
end
```

Uses `circee` 34c.

and the second one:

13a ⟨`sample.f` 12a⟩+≡

```
double precision function d2 (t2)
implicit none
double precision t2, x2, sigma, circee
⟨EPS & PWR 13b⟩
x2 = 1d0 - t2**PWR
d2 = PWR * t2**(PWR-1d0) * sigma (x2) * circee (1d0, x2)
end
```

Uses `circee` 34c.

Below you will see that the power of the singularity of the e^+e^- distributions at $x \to 1$ is $\approx -2/3$. To be on the safe side, we choose the power η in (10) as 5. It is kept in the parameter `PWR`, while `EPS` is the desired accuracy of the Gaussian integration:

13b ⟨EPS & PWR 13b⟩≡

```
double precision EPS, PWR
parameter (EPS = 1d-6, PWR = 5d0)
```

The Gauss integration of the non-singular version converges to the cotrrect value only if the final bin is integrated separately:

13c ⟨*Second Gauss integration* 13c⟩≡

```
 s = gauss2 (d12a, 0d0, 1d0-KIREPS, 0d0, 1d0-KIREPS, EPS)
$  + gauss2 (d12a, 0d0, 1d0-KIREPS, 1d0-KIREPS, 1d0, EPS)
$  + gauss2 (d12a, 1d0-KIREPS, 1d0, 0d0, 1d0-KIREPS, EPS)
$  + gauss2 (d12a, 1d0-KIREPS, 1d0, 1d0-KIREPS, 1d0, EPS)
 write (*, 1000) 'delta(sigma) (Gauss) =', (s-1d0)*100d0
```

Uses `gauss2` 77a.

13d ⟨EPS & PWR 13b⟩+≡

```
double precision KIREPS
parameter (KIREPS = 1D-6)
```

13e ⟨`sample.f` 12a⟩+≡

```
double precision function d12a (x1, x2)
implicit none
double precision x1, x2, sigma, kirkee
d12a = sigma (x1*x2) * kirkee (x1, x2)
end
```

Uses `kirkee` 63b.

These code fragments can now be used in a main program that loops over energies and accelerator designs

13f ⟨`sample.f` 12a⟩+≡

```
program sample
implicit none
```

```
      ⟨Accelerator codes 10c⟩
      ⟨EPS & PWR 13b⟩
      ⟨Other variables in sample 15⟩
      integer acc, ver, i
      double precision roots(9)
      data roots / 90D0,  170D0,  250D0,  350D0,  500D0,
     $            800D0, 1000D0, 1200D0, 1500D0 /
      do 10 acc = JLCNLC, NLCH, NLCH-JLCNLC
         do 11 ver = 9, 9
            do 12 i = 1, 9
               call circes (0d0, 0d0, roots(i), acc, ver, 20020328, 1)
               ⟨Gauss integration 12b⟩
               ⟨Second Gauss integration 13c⟩
               ⟨Monte Carlo integration 16d⟩
 14            continue
 12         continue
 13         continue
 11      continue
 10   continue
      end
```

Uses `circes` 25a.

with the following result

14 ⟨*Sample output* 14⟩≡

```
circe:message: starting up ...
circe:message: $Id: prelude.nw 66 2002-03-28 17:00:10Z ohl $
circe:message: updating ‘roots’ to   90.0
circe:message: updating ‘ver’ to  7
circe:message: updating ‘rev’ to 20000501
delta(sigma) (Gauss) =  0.11%
delta(sigma) (MC)    =  0.11%
                   +/-  0.00%
circe:message: updating ‘roots’ to  170.0
circe:message: updating ‘ver’ to  7
delta(sigma) (Gauss) =  0.38%
delta(sigma) (MC)    =  0.38%
                   +/-  0.01%
circe:message: updating ‘roots’ to  350.0
circe:message: updating ‘ver’ to  7
delta(sigma) (Gauss) =  1.67%
delta(sigma) (MC)    =  1.66%
                   +/-  0.03%
circe:message: updating ‘roots’ to  500.0
circe:message: updating ‘ver’ to  7
delta(sigma) (Gauss) =  3.66%
delta(sigma) (MC)    =  3.58%
                   +/-  0.07%
circe:message: updating ‘roots’ to  800.0
circe:message: updating ‘ver’ to  7
delta(sigma) (Gauss) =  5.21%
delta(sigma) (MC)    =  5.19%
                   +/-  0.11%
circe:message: updating ‘roots’ to 1000.0
circe:message: updating ‘ver’ to  7
circe:message: energy 1000.0GeV too high, using spectrum for  800.0GeV
delta(sigma) (Gauss) =  5.21%
delta(sigma) (MC)    =  5.19%
                   +/-  0.11%
circe:message: updating ‘roots’ to   90.0
circe:message: updating ‘acc’ to JLCNLC
circe:message: updating ‘ver’ to  7
circe:message: energy   90.0GeV too low, using spectrum for  500.0GeV
delta(sigma) (Gauss) =  4.74%
delta(sigma) (MC)    =  4.75%
                   +/-  0.11%
circe:message: updating ‘roots’ to  170.0
circe:message: updating ‘ver’ to  7
circe:message: energy  170.0GeV too low, using spectrum for  500.0GeV
delta(sigma) (Gauss) =  4.74%
delta(sigma) (MC)    =  4.68%
                   +/-  0.11%
circe:message: updating ‘roots’ to  350.0
circe:message: updating ‘ver’ to  7
circe:message: energy  350.0GeV too low, using spectrum for  500.0GeV
delta(sigma) (Gauss) =  4.74%
delta(sigma) (MC)    =  4.75%
                   +/-  0.11%
circe:message: updating ‘roots’ to  500.0
circe:message: updating ‘ver’ to  7
delta(sigma) (Gauss) =  4.74%
delta(sigma) (MC)    =  4.75%
                   +/-  0.11%
circe:message: updating ‘roots’ to  800.0
circe:message: updating ‘ver’ to  7
circe:message: energy  800.0GeV interpolated between  500.0 and 1000.0GeV
delta(sigma) (Gauss) =  8.37%
delta(sigma) (MC)    =  8.39%
                   +/-  0.21%
circe:message: updating ‘roots’ to 1000.0
circe:message: updating ‘ver’ to  7
delta(sigma) (Gauss) = 15.39%
delta(sigma) (MC)    = 14.68%
                   +/-  0.33%
```

Uses `circe` 24d.

We almost forgot to declare the variables in the main program

15 ⟨*Other variables in* `sample` 15⟩≡

```
      double precision s
      double precision gauss1, gauss2, circee, sigma, d1, d2, d12, d12a
      external d1, d2, d12, d12a
```

Uses `circee` 34c, `gauss1` 76d, and `gauss2` 77a.

This concludes the integration example. It should have made it obvious how to proceed in a realistic application.

In section 3.2.1 below, I will describe a Monte Carlo method for calculating such integrals efficiently.

3.2 Generators

The function `circe` and its companions are opaque to the user. Since they will in general contain singularities, applications will *not* be able to generate corresponding samples of random numbers efficiently. To fill this gap, four random number generators are provided. The subroutine `girce` will generate particle types $p_{1,2}$ and energy fractions $x_{1,2}$ in one step, according to the selected distribution.[2] Particle p_1 will be either a positron or a photon and p_2 will be either an electron or a photon. The energy fractions are guaranteed to be above the currently active thresholds: $x_i \geq x_{i,\text{min}}$. This can be used to cut on soft events—the photon distributions are rather soft—which might not be interesting in most simulations.

16a ⟨*API documentation* 9a⟩+≡

```
call girce  (x1, x2, p1, p2, rng)
```

Uses `girce` 67b.

The output parameters of `girce` are identical to the input parameters of `circe`, with the exception of `rng`. The latter is a subroutine with a single double precision argument, which will be assigned a uniform deviate from the interval $[0, 1]$ after each call:

16b ⟨*API documentation* 9a⟩+≡

```
subroutine rng (r)
double precision r
r = ⟨uniform deviate on [0,1] (never defined)⟩
end
```

Typically, it will be just a wrapper around the standard random number generator of the application program. For studies with a definite initial state, three generator functions are available.

16c ⟨*API documentation* 9a⟩+≡

```
call gircee (x1, x2, rng)
call girceg (x1, x2, rng)
call gircgg (x1, x2, rng)
```

Uses `gircee` 69b, `girceg` 69d, and `gircgg` 70c.

3.2.1 Example

Returning to the example from section 3.2.1, I present a concise Monte Carlo algorithm for calculating the same integral:

16d ⟨*Monte Carlo integration* 16d⟩≡

```
s = 0d0
s2 = 0d0
```

---

[2] The implementation of the flavor selection with non-vanishing thresholds $x_{1,\text{min}}$ and $x_{2,\text{min}}$ is moderately inefficient at the moment. It can be improved by a factor of two.

```
      do 100 n = 1, NEVENT
         call gircee (x1, x2, random)
         w = sigma (x1*x2)
         s = s + w
         s2 = s2 + w*w
  100 continue
      s = s / dble(NEVENT)
      s2 = s2 / dble(NEVENT)
      write (*, 1000) 'delta(sigma) (MC)    =', (s-1d0)*100d0
      write (*, 1000) '                   +/-',
     $          sqrt((s2-s*s)/dble(NEVENT))*100d0
```

Uses `gircee` 69b.

17a ⟨*Other variables in* `sample` 15⟩+≡

```
      double precision w, s2, x1, x2
      external random
      integer NEVENT, n
      parameter (NEVENT = 10000)
```

Here is a simple linear congruential random number generator for the sample program. Real applications will use their more sophisticated generators instead.

17b ⟨`sample.f` 12a⟩+≡

```
      subroutine random (r)
      implicit none
      double precision r
      integer m, a, c
      parameter (M = 259200, A = 7141, C = 54773)
      integer n
      save n
      data n /0/
      n = mod(n*a+c,m)
      r = dble (n) / dble (m)
      end
```

If the cross section is slowly varying on the range where the $x_{1,2}$ distributions are non-zero, this algorithm is very efficient.

However, if this condition is not met, the explicit form of the parameterizations in section 5 should be consulted and appropriate mapping techniques should be applied. The typical example for this problem is a narrow resonance just below the nominal beam energy.

3.2.2 Event Generators

For Monte Carlo event generators that use the standard `/hepevt/` common block [14], the addition of the `Circe` library is trivial. During the initialization of the event generator, the `circes` subroutine is called to set up `Circe`'s internal state. For example:

17c ⟨*Initialize event generator* 17c⟩≡

```
      call circes (0d0, 0d0, roots, acc, ver, 1996 07 11, 1)
```

Uses `circes` 25a.

Figure 5: Architecture of `Circe`: `circes()` selects energy and accelerator and loads the parameterization. The function `circe()` calculates the values of the selected distribution function at the given energy fractions. The subroutine `girce()` generates energy fractions using a specified random number generator in accordance with the selected distribution.

During event generation, before setting up the e^+e^- initial state, the `gircee` subroutine is called with the event generator's random number generator:

18a ⟨*Event generation* 18a⟩≡

```
        call gircee (x1, x2, random)
```

Uses `gircee` 69b.

The resulting energy fractions x_1 and x_2 are now available for defining the initial state electron

18b ⟨*Event generation* 18a⟩+≡

```
        isthep(1) = 101
        idhep(1) = ELECTR
        phep(1,1) = 0d0
        phep(2,1) = 0d0
        phep(3,1) = x1 * ebeam
        phep(4,1) = x1 * ebeam
        phep(5,1) = 0d0
```

Uses `ELECTR` 9b.

and positron.

18c ⟨*Event generation* 18a⟩+≡

```
        isthep(2) = 102
        idhep(2) = POSITR
        phep(1,2) = 0d0
        phep(2,2) = 0d0
        phep(3,2) = - x2 * ebeam
        phep(4,2) = x2 * ebeam
        phep(5,2) = 0d0
```

Uses `POSITR` 9b.

Using `Circe` with other event generators should be straightforward as well.

4 Technical Notes

The structure of `Circe` is extremely simple (cf. figure 5) and is mainly a bookkeeping excercise. All that needs to be done is to maintain a database of available parameterizations and to evaluate the corresponding functions. The only non trivial algorithms are used for the efficient generation of random deviates.

I have avoided the use of initialized `common` blocks (i.e. `block data` subroutines), because the Fortran77 standard does not provide a *portable* way of ensuring that `block data` subroutines are actually executed at loading time. Instead, the `/circom/` common block is tagged by a "magic number" to check for initialization and its members are filled by the `circes` subroutine when necessary.

	SBAND	TESLA	TESLA'	XBAND
$\mathcal{L}/\mathrm{fb}^{-1}\upsilon^{-1}$	$31.38^{+0.22}_{-0.22}$	$106.25^{+0.71}_{-0.71}$	$95.24^{+0.73}_{-0.73}$	$36.39^{+0.29}_{-0.29}$
$\int d_{e^\pm}$	$0.4812^{+0.0041}_{-0.0041}$	$0.5723^{+0.0046}_{-0.0045}$	$0.3512^{+0.0048}_{-0.0048}$	$0.3487^{+0.0040}_{-0.0040}$
$x_{e^\pm}^\alpha$	$11.1534^{+0.0770}_{-0.0761}$	$15.2837^{+0.0923}_{-0.0914}$	$27.1032^{+0.3071}_{-0.3019}$	$6.9853^{+0.0733}_{-0.0718}$
$(1-x_{e^\pm})^\alpha$	$-0.6302^{+0.0013}_{-0.0012}$	$-0.6166^{+0.0011}_{-0.0011}$	$-0.6453^{+0.0017}_{-0.0017}$	$-0.6444^{+0.0017}_{-0.0017}$
$\int d_\gamma$	$0.6237^{+0.0033}_{-0.0033}$	$0.7381^{+0.0036}_{-0.0036}$	$0.3502^{+0.0034}_{-0.0034}$	$0.4149^{+0.0031}_{-0.0031}$
x_γ^α	$-0.6911^{+0.0006}_{-0.0006}$	$-0.6921^{+0.0006}_{-0.0006}$	$-0.6947^{+0.0011}_{-0.0011}$	$-0.6876^{+0.0010}_{-0.0010}$
$(1-x_\gamma)^\alpha$	$14.9355^{+0.0761}_{-0.0754}$	$24.1647^{+0.1124}_{-0.1116}$	$33.6576^{+0.3021}_{-0.2983}$	$8.3227^{+0.0659}_{-0.0649}$

Table 5: Version 1, revision 1997 04 16 of the beam spectra at 500 GeV. The rows correspond to the luminosity per effective year, the integral over the continuum and the powers in the factorized Beta distributions (12).

Figure 6: Fit of the $e^\pm$- and γ-distributions for the S-Band design at $\sqrt{s} =$ 500GeV. The open circles with error bars are the result of the `Guinea-Pig` similation. The full line is the fit.

Figure 7: Fit of the $e^\pm$- and γ-distributions for the Tesla design at $\sqrt{s} =$ 500GeV.

A more flexible method would be to replace the `data` statements by reading external files. This option causes portability problems, however, because I would have to make sure that the names of the external files are valid in all files systems of the target operating systems. More significantly, splitting the implementation into several parts forces the user to keep all files up to date. This can be a problem, because Fortran source files and data input files will typically be kept in different parts of the file system.

The option of implementing `Circe` statelessly, i.e. with pure function calls and without `common` blocks, has been dismissed. While it would have been more straightforward on the side of the library, it would have placed the burdon of maintaining state (accelerator, energy, etc.) on the application program, thereby complicating them considerably. Keeping an explicit state in `Circe` has the additional benefit of allowing to precompute certain internal variables, resulting in a more efficient implementation.

5 Parameterizations

Version 1.02β of `Circe` supports just one version of the parameterizations. Future versions will provide additional parameterizations.

	`SBAND`	`TESLA`	`TESLA'`	`XBAND`
$\mathcal{L}/\mathrm{fb}^{-1}\upsilon^{-1}$	$119.00^{+0.83}_{-0.83}$	214.33^{+0***}_{-0***}	212.22^{+0***}_{-0***}	$118.99^{+0.91}_{-0.91}$
$\int d_{e^\pm}$	$0.5604^{+0.0040}_{-0.0039}$	$0.6686^{+0.0040}_{-0.0040}$	$0.4448^{+0.0043}_{-0.0043}$	$0.5001^{+0.0038}_{-0.0038}$
$x_{e^\pm}^{\alpha}$	$4.2170^{+0.0258}_{-0.0255}$	$5.5438^{+0.0241}_{-0.0239}$	$9.6341^{+0.0814}_{-0.0803}$	$2.6184^{+0.0192}_{-0.0190}$
$(1-x_{e^\pm})^{\alpha}$	$-0.6118^{+0.0013}_{-0.0013}$	$-0.5847^{+0.0011}_{-0.0011}$	$-0.6359^{+0.0014}_{-0.0014}$	$-0.6158^{+0.0015}_{-0.0015}$
$\int d_{\gamma}$	$0.7455^{+0.0032}_{-0.0032}$	$1.0112^{+0.0033}_{-0.0033}$	$0.4771^{+0.0031}_{-0.0031}$	$0.6741^{+0.0031}_{-0.0031}$
x_{γ}^{α}	$-0.6870^{+0.0006}_{-0.0006}$	$-0.6908^{+0.0004}_{-0.0004}$	$-0.6936^{+0.0008}_{-0.0008}$	$-0.6834^{+0.0007}_{-0.0007}$
$(1-x_{\gamma})^{\alpha}$	$6.7145^{+0.0310}_{-0.0308}$	$9.9992^{+0.0342}_{-0.0340}$	$13.1607^{+0.0896}_{-0.0886}$	$3.8589^{+0.0215}_{-0.0213}$

Table 6: Version 1, revision 1997 04 17 of the beam spectra at 1 TeV.

	350 GeV	500 GeV	800 GeV	1600 GeV
$\mathcal{L}/\mathrm{fb}^{-1}\upsilon^{-1}$	$97.45^{+0.67}_{-0.67}$	$106.25^{+0.71}_{-0.71}$	170.86^{+0***}_{-0***}	340.86^{+0***}_{-0***}
$\int d_{e^\pm}$	$0.6093^{+0.0049}_{-0.0049}$	$0.5723^{+0.0046}_{-0.0045}$	$0.6398^{+0.0042}_{-0.0041}$	$0.5094^{+0.0040}_{-0.0040}$
$x_{e^\pm}^{\alpha}$	$17.6137^{+0.1065}_{-0.1055}$	$15.2837^{+0.0923}_{-0.0914}$	$7.6221^{+0.0365}_{-0.0361}$	$5.0550^{+0.0353}_{-0.0349}$
$(1-x_{e^\pm})^{\alpha}$	$-0.6061^{+0.0011}_{-0.0011}$	$-0.6166^{+0.0011}_{-0.0011}$	$-0.5944^{+0.0011}_{-0.0011}$	$-0.6187^{+0.0013}_{-0.0013}$
$\int d_{\gamma}$	$0.7729^{+0.0039}_{-0.0039}$	$0.7381^{+0.0036}_{-0.0036}$	$0.9178^{+0.0034}_{-0.0034}$	$0.5875^{+0.0031}_{-0.0031}$
x_{γ}^{α}	$-0.6949^{+0.0006}_{-0.0006}$	$-0.6921^{+0.0006}_{-0.0006}$	$-0.6908^{+0.0005}_{-0.0005}$	$-0.6892^{+0.0007}_{-0.0007}$
$(1-x_{\gamma})^{\alpha}$	$28.9399^{+0.1370}_{-0.1361}$	$24.1647^{+0.1124}_{-0.1116}$	$13.1167^{+0.0497}_{-0.0495}$	$7.5514^{+0.0428}_{-0.0424}$

Table 7: Version 1, revision 1997 04 17 of the beam spectra for TESLA.

	500 GeV	800 GeV
$\mathcal{L}/\mathrm{fb}^{-1}\upsilon^{-1}$	$339.80^{+0.83}_{-0.83}$	$359.36^{+0.93}_{-0.93}$
$\int d_{e^\pm}$	$0.5019^{+0.0016}_{-0.0016}$	$0.4125^{+0.0016}_{-0.0016}$
$x_{e^\pm}^{\alpha}$	$12.2867^{+0.0318}_{-0.0316}$	$13.3242^{+0.0442}_{-0.0440}$
$(1-x_{e^\pm})^{\alpha}$	$-0.6276^{+0.0005}_{-0.0005}$	$-0.6401^{+0.0005}_{-0.0005}$
$\int d_{\gamma}$	$0.5114^{+0.0012}_{-0.0012}$	$0.3708^{+0.0011}_{-0.0011}$
x_{γ}^{α}	$-0.6912^{+0.0003}_{-0.0003}$	$-0.6924^{+0.0004}_{-0.0004}$
$(1-x_{\gamma})^{\alpha}$	$17.0673^{+0.0375}_{-0.0375}$	$16.8145^{+0.0482}_{-0.0480}$

Table 8: Version 5, revision 1998 05 05 of the beam spectra for high luminosity TESLA.

Figure 8: Fit of the $e^{\pm}$- and γ-distributions for the X-Band design at $\sqrt{s} = 500\mathrm{GeV}$.

Figure 9: Fit of the $e^{\pm}$- and γ-distributions for the Tesla design at $\sqrt{s} = 1\text{TeV}$.

	`SBNDEE`	`TESLEE`	`XBNDEE`
$\mathcal{L}/\text{fb}^{-1}\upsilon^{-1}$	$9.29^{+0.06}_{-0.06}$	$21.62^{+0.17}_{-0.17}$	$13.97^{+0.10}_{-0.10}$
$\int d_{e^\pm}$	$.6513^{+0.0059}_{-0.0059}$	$.7282^{+0.0083}_{-0.0082}$	$.5270^{+0.0049}_{-0.0049}$
$x^{\alpha}_{e^\pm}$	$10.3040^{+0.0601}_{-0.0593}$	$14.8578^{+0.1047}_{-0.1034}$	$5.8897^{+0.0455}_{-0.0448}$
$(1-x_{e^\pm})^{\alpha}$	$-.5946^{+0.0015}_{-0.0015}$	$-.5842^{+0.0018}_{-0.0018}$	$-.6169^{+0.0016}_{-0.0015}$
$\int d_{\gamma}$	$.4727^{+0.0035}_{-0.0035}$	$.5300^{+0.0046}_{-0.0046}$	$.3746^{+0.0029}_{-0.0029}$
x^{α}_{γ}	$-.6974^{+0.0009}_{-0.0009}$	$-.7039^{+0.0009}_{-0.0009}$	$-.6892^{+0.0010}_{-0.0010}$
$(1-x_{\gamma})^{\alpha}$	$20.6447^{+0.1513}_{-0.1497}$	$36.1286^{+0.3027}_{-0.2991}$	$10.0872^{+0.0822}_{-0.0815}$

Table 9: *Experimental* Version 1, revision 0 of the beam spectra at 500 GeV. The rows correspond to the luminosity per effective year, the integral over the continuum and the powers in the factorized Beta distributions (12).

	`SBNDEE`	`TESLEE`	`XBNDEE`
$\mathcal{L}/\text{fb}^{-1}\upsilon^{-1}$	$45.59^{+0.34}_{-0.34}$	$25.47^{+0.20}_{-0.20}$	$41.06^{+0.28}_{-0.28}$
$\int d_{e^\pm}$	$.7892^{+0.0075}_{-0.0074}$	$.6271^{+0.0066}_{-0.0065}$	$.7203^{+0.0058}_{-0.0058}$
$x^{\alpha}_{e^\pm}$	$5.4407^{+0.0285}_{-0.0281}$	$8.7504^{+0.0669}_{-0.0658}$	$2.7415^{+0.0121}_{-0.0119}$
$(1-x_{e^\pm})^{\alpha}$	$-.5285^{+0.0020}_{-0.0020}$	$-.6058^{+0.0017}_{-0.0017}$	$-.5049^{+0.0020}_{-0.0020}$
$\int d_{\gamma}$	$.6403^{+0.0040}_{-0.0040}$	$.4278^{+0.0038}_{-0.0038}$	$.6222^{+0.0032}_{-0.0032}$
x^{α}_{γ}	$-.6960^{+0.0008}_{-0.0008}$	$-.6982^{+0.0010}_{-0.0010}$	$-.6795^{+0.0008}_{-0.0008}$
$(1-x_{\gamma})^{\alpha}$	$12.4803^{+0.0839}_{-0.0831}$	$18.5260^{+0.1674}_{-0.1655}$	$4.7506^{+0.0262}_{-0.0260}$

Table 10: *Experimental* Version 1, revision 0 of the beam spectra at 1 TeV.

5.1 Version 1

The first version of the parameterization uses a simple factorized *ansatz*

$$D^{\alpha 1 \rho}_{p_1 p_2}(x_1, x_2, s) = d^{\alpha 1 \rho}_{p_1}(x_1) d^{\alpha 1 \rho}_{p_2}(x_2) \tag{12a}$$

where the distributions are simple Beta distributions:

$$d^{\alpha 1 \rho}_{e^\pm}(x) = a^{\alpha\rho}_0 \delta(1-x) + a^{\alpha\rho}_1 x^{a^{\alpha\rho}_2} (1-x)^{a^{\alpha\rho}_3} \tag{12b}$$

$$d^{\alpha 1 \rho}_{\gamma}(x) = a^{\alpha\rho}_4 x^{a^{\alpha\rho}_5} (1-x)^{a^{\alpha\rho}_6} \tag{12c}$$

This form of the distributions is motivated by the observation [2] that the $e^{\pm}$ distributions diverge like a power for $x \to 1$ and vanish at $x \to 0$. The behavior of the γ distributions is similar with the borders exchanged.

	350 GeV	500 GeV	800 GeV
$\mathcal{L}/\mathrm{fb}^{-1}v^{-1}$	$15.18^{+0.13}_{-0.13}$	$21.62^{+0.17}_{-0.17}$	$43.98^{+0.38}_{-0.38}$
$\int d_{e^\pm}$	$.6691^{+0.0083}_{-0.0083}$	$.7282^{+0.0083}_{-0.0082}$	$.7701^{+0.0090}_{-0.0089}$
$x_{e^\pm}^\alpha$	$25.2753^{+0.2040}_{-0.2007}$	$14.8578^{+0.1047}_{-0.1034}$	$8.1905^{+0.0543}_{-0.0535}$
$(1-x_{e^\pm})^\alpha$	$-.5994^{+0.0017}_{-0.0017}$	$-.5842^{+0.0018}_{-0.0018}$	$-.5575^{+0.0021}_{-0.0021}$
$\int d_\gamma$	$.4464^{+0.0047}_{-0.0047}$	$.5300^{+0.0046}_{-0.0046}$	$.5839^{+0.0047}_{-0.0047}$
x_γ^α	$-.7040^{+0.0011}_{-0.0011}$	$-.7039^{+0.0009}_{-0.0009}$	$-.7046^{+0.0009}_{-0.0009}$
$(1-x_\gamma)^\alpha$	$60.1882^{+0.5882}_{-0.5797}$	$36.1286^{+0.3027}_{-0.2991}$	$19.3944^{+0.1681}_{-0.1660}$

Table 11: *Experimental* Version 1, revision 0 of the beam spectra for `TESLEE`.

5.1.1 Fitting

The parameters a_i in (12) have been obtained by a least-square fit of (12) to histograms of simulation results from `Guinea-Pig`. Some care has to taken when fitting singular distributions to histogrammed data. Obviously equidistant bins are not a good idea, because most bins will be almost empty (cf. figures 1 and 2) and consequently a lot of information will be wasted. One solution to this problem is the use of logarithmic bins. This, however, maps the compact region $[0,1] \times [0,1]$ to $[-\infty, 0] \times [-\infty, 0]$, which is inconvenient because of the missing lower bounds.

The more appropriate solution is to use two maps

$$\begin{aligned} \phi : [0,1] &\to [0,1] \\ x &\mapsto y = x^{1/\eta} \end{aligned} \tag{13}$$

where $x = x_\gamma$ or $x = 1 - x_{e^\pm}$, and to bin the result equidistantly. If η is chosen properly (cf. (10)), the bin contents will then fall off at the singularity. The fits in tables 5, 6, and 7 have been performed with $\eta = 5$ and the resulting bin contents can be read off from figures 6–9.

Using this procedure for binning the results of the simulations, the popular fitting package `MINUIT` [15] converges quickly in all cases considered. The resulting parameters are given in tables 5, 6, and 7. Plots of the corresponding distributions have been shown in figures 1 and 2. It is obvious that an *ansatz* like (12) is able to distinguish among the accelerator designs. Thus it can provide a solid basis for physics studies.

In figures 6–9 I give a graphical impression of the quality of the fit, which appears to be as good as one could reasonably expect for a simple *ansatz* like (12). Note that the histograms have non-equidistant bins and that the resulting Jacobians have not been removed. Therefore the bin contents falls off at the singularities, as discussed above.

The errors used for the least-square fit had to be taken from a Monte Carlo (MC) study. `Guinea-Pig` only provides the $\sqrt{n}$ from Poissonian statistics for each bin, but the error accumulation during tracking the particles through phase space is not available. The MC studies shows that the latter error dominates the former, but appears to be reasonably Gaussian. A complete MC study of all parameter sets is computationally expensive (more than a week of processor time on a fast SGI). From an exemplary MC study of a few parameter sets, it appears that the errors can be described reasonably well by rescaling the Poissonian error in each bin with appropriate factors for electrons/positrons and photons and for continuum and delta. This procedure has been adopted.

The χ^2/d.o.f.'s of the fits are less than $\mathcal{O}(10)$. The simple *ansatz* (12) is therefore very satisfactory. In fact, trying to improve the ad-hoc factorized Beta distributions by the better motivated approximations from [7] or [16], it turns out [17] that (12) provides a significantly better fit of the results of the simulations. The price to pay is that the parameters in (12) have no direct physical interpretation.

5.1.2 Generators

For this version of the parameterizations we need a fast generator of Beta distributions:

$$\beta^{a,b}(x) \propto x^{a-1}(1-x)^{b-1} \tag{14}$$

This problem has been studied extensively and we can use a published algorithm [18] that is guaranteed to be very fast for all a, b such that $0 < a \leq 1 \leq b$, which turns out to be always the case (cf. tables 5, 6, and 7).

5.2 Future Versions

There are two ways in which the parameterizations can be improved:

more complicated functions: the factorized fits can only be improved marginally by adding more positive semi-definite factors to (12). More improvement is possible by using sums of functions, but in this case, the best fits violate the positivity requirement and have to be discarded.

correlations: the parameterization in section 5.1 is factorized. While this is a good approximation, the simulations nevertheless show correlations among x_1 and x_2. These correlations can be included in a future version.

interpolation: the parameterization in section 5.1 is based on fitting the simulation results by simple functions. Again, this appears to be a good approximation. But such fits can not uncover any fine structure of the distributions. Therefore it will be worthwhile to study interpolations of the simulation results in the future. A proper interpolation of results with statistical errors is however far from trivial: straightforward polynomial or spline interpolations will be oscillatory and violate the positivity requirement. Smoothing algorithms have to be investigated in depth before such a parameterization can be released.

other simulations: besides [5], other simulation codes are invited to contribute their results for inclusion in the `Circe` library.

6 Implementation of `circe`

24a ⟨`circe1.f` 24a⟩≡

```
c circe1.f -- canonical beam spectra for linear collider physics
c $Id: circe.nw 67 2002-03-28 17:13:06Z ohl $
⟨Copyleft notice 24b⟩
⟨Subroutines 24d⟩
```

Uses `circe` 24d.

The following is usually not needed for scientific programs. Nobody is going to hijack such code. But let us include it anyway to spread the gospel of free software:

24b ⟨*Copyleft notice* 24b⟩≡

```
c   Copyright (C) 1996-2002 by Thorsten Ohl <ohl@hep.tu-darmstadt.de>
c
c   Circe is free software; you can redistribute it and/or modify it
c   under the terms of the GNU General Public License as published by
c   the Free Software Foundation; either version 2, or (at your option)
c   any later version.
c
c   Circe is distributed in the hope that it will be useful, but
c   WITHOUT ANY WARRANTY; without even the implied warranty of
c   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
c   GNU General Public License for more details.
c
c   You should have received a copy of the GNU General Public License
c   along with this program; if not, write to the Free Software
c   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Now we can move on to the implementation.

6.1 Symbolic Constants

The file `circe.h` contains symbolic names for various magic constants used by `Circe`:

24c ⟨`circe.h` 24c⟩≡

```
c circe.h -- canonical beam spectra for linear collider physics
c $Id: circe.nw 67 2002-03-28 17:13:06Z ohl $
⟨Copyleft notice 24b⟩
⟨Particle codes 9b⟩
⟨Accelerator codes 10c⟩
```

Uses `circe` 24d.

6.2 Distributions

6.2.1 Version 1

We start with a convenience function which dispatches over the valid particle types. The hardest part is of course to avoid typos in such trivial functions ...

24d ⟨*Subroutines* 24d⟩≡

```
      double precision function circe (x1, x2, p1, p2)
      implicit none
      double precision x1, x2
      integer p1, p2
      double precision circee, circeg, circgg
⟨Particle codes 9b⟩
⟨/circom/ 25b⟩
⟨Initialization check 26e⟩
      circe = -1.0
      if (abs(p1) .eq. ELECTR) then
         if (abs(p2) .eq. ELECTR) then
            circe = circee (x1, x2)
         elseif (p2 .eq. PHOTON) then
            circe = circeg (x1, x2)
         endif
      elseif (p1 .eq. PHOTON) then
         if (abs(p2) .eq. ELECTR) then
            circe = circeg (x2, x1)
         elseif (p2 .eq. PHOTON) then
            circe = circgg (x1, x2)
         endif
      endif
      end
```

Defines:
`circe`, used in chunks 24d, 9a, 24, 14, 24, 26f, 74, 79a, 93c, and 98b.
Uses `ELECTR` 9b, `PHOTON` 9b, `circee` 34c, `circeg` 35a, and `circgg` 35c.

25a ⟨*Subroutines* 24d⟩+≡

```
      subroutine circes (xx1m, xx2m, xroots, xacc, xver, xrev, xchat)
      implicit none
      double precision xx1m, xx2m, xroots
      double precision beta
      integer xacc, xver, xrev, xchat
⟨Accelerator codes 10c⟩
⟨/circom/ 25b⟩
⟨Local variables for circes 27a⟩
⟨Initializations for circes 28c⟩
      if (magic .ne. 1904 06 16) then
         magic = 1904 06 16
         ⟨Initialize /circom/ 26f⟩
      endif
⟨Update /circom/ 26g⟩
⟨formats for circes 31d⟩
      end
```

Defines:
`circes`, used in chunks 25a, 10b, 13f, 25a, 17c, 25, 26e, 74, and 78a.
Uses `beta` 90a.

25b ⟨`/circom/` 25b⟩≡

```
⟨parameter part of /circom/ 26d⟩
⟨8-byte aligned part of /circom/ 26a⟩
⟨4-byte aligned part of /circom/ 26b⟩
```

```
        save /circom/
```

26a ⟨*8-byte aligned part of* `/circom/` 26a⟩≡

```
        double precision x1m, x2m, roots
        common /circom/ x1m, x2m, roots
```

26b ⟨*4-byte aligned part of* `/circom/` 26b⟩≡

```
        integer acc, ver, rev, chat
        common /circom/ acc, ver, rev, chat
```

Instead of using fragile `block data` subroutines, we use a magic number to tag `/circom/` as initialized:

26c ⟨*4-byte aligned part of* `/circom/` 26b⟩+≡

```
        integer magic
        common /circom/ magic
```

26d ⟨`parameter` *part of* `/circom/` 26d⟩≡

```
        integer MAGIC0
        parameter (MAGIC0 = 1904 06 16)
```

Since negative values are no updated, we can call `circes` with all negative variables to ensure initialization:

26e ⟨*Initialization check* 26e⟩≡

```
        if (magic .ne. MAGIC0) then
           call circes (-1d0, -1d0, -1d0, -1, -1, -1, -1)
        endif
```

Uses `circes` 25a.

26f ⟨*Initialize* `/circom/` 26f⟩≡

```
        x1m = 0d0
        x2m = 0d0
        roots = 500D0
        acc = TESLA
        ver = 0
        rev = 0
        chat = 1
        if (xchat .ne. 0) then
           call circem ('MESSAGE', 'starting up ...')
           call circem ('MESSAGE',
       $        '$Id: circe.nw 67 2002-03-28 17:13:06Z ohl $')
        endif
```

Uses `TESLA` 10c and `circe` 24d.

26g ⟨*Update* `/circom/` 26g⟩≡

```
        if ((xchat .ge. 0) .and. (xchat .ne. chat)) then
           chat = xchat
           if (chat .ge. 1) then
              write (msgbuf, 1000) 'chat', chat
 1000         format ('updating `', A, ''' to ', I2)
              call circem ('MESSAGE', msgbuf)
           endif
        else
           if (chat .ge. 2) then
              write (msgbuf, 1100) ’chat’, chat
   1100       format (’keeping ‘’, A, ’’’ at ’, I2)
              call circem (’MESSAGE’, msgbuf)
           endif
        endif
```

27a ⟨*Local variables for* `circes` 27a⟩≡

```
     character*60 msgbuf
```

27b ⟨*Update* `/circom/` 26g⟩+≡

```
     if ((xx1m .ge. 0d0) .and. (xx1m .ne. x1m)) then
        x1m = xx1m
        if (chat .ge. 1) then
           write (msgbuf, 1001) ’x1min’, x1m
1001       format (’updating ‘’, A, ’’’ to ’, E12.4)
           call circem (’MESSAGE’, msgbuf)
        endif
     else
        if (chat .ge. 2) then
           write (msgbuf, 1101) ’x1min’, x1m
1101       format (’keeping ‘’, A, ’’’ at ’, E12.4)
           call circem (’MESSAGE’, msgbuf)
        endif
     endif
```

27c ⟨*Update* `/circom/` 26g⟩+≡

```
     if ((xx2m .ge. 0d0) .and. (xx2m .ne. x2m)) then
        x2m = xx2m
        if (chat .ge. 1) then
           write (msgbuf, 1001) ’x2min’, x2m
           call circem (’MESSAGE’, msgbuf)
        endif
     else
        if (chat .ge. 2) then
           write (msgbuf, 1101) ’x2min’, x2m
           call circem (’MESSAGE’, msgbuf)
        endif
     endif
```

27d ⟨*Update* `/circom/` 26g⟩+≡

```
     if ((xroots .ge. 0d0) .and.(xroots .ne. roots)) then
        roots = xroots
        if (chat .ge. 1) then
           write (msgbuf, 1002) ’roots’, roots
1002       format (’updating ‘’, A, ’’’ to ’, F6.1)
           call circem (’MESSAGE’, msgbuf)
        endif
     else
        if (chat .ge. 2) then
           write (msgbuf, 1102) ’roots’, roots
1102       format (’keeping ‘’, A, ’’’ at ’, F6.1)
           call circem (’MESSAGE’, msgbuf)
```

```
         endif
      endif
```

28a ⟨*Update* /circom/ 26g⟩+≡

```
      if ((xacc .ge. 0) .and.(xacc .ne. acc)) then
         if ((xacc .ge. 1) .and. (xacc .le. NACC)) then
            acc = xacc
            if (chat .ge. 1) then
               write (msgbuf, 1003) 'acc', accnam(acc)
 1003          format ('updating `', A, ''' to ', A)
               call circem ('MESSAGE', msgbuf)
            endif
         else
            write (msgbuf, 1203) xacc
 1203       format ('invalid `acc'': ', I8)
            call circem ('ERROR', msgbuf)
            write (msgbuf, 1103) 'acc', accnam(acc)
 1103       format ('keeping `', A, ''' at ', A)
            call circem ('MESSAGE', msgbuf)
         endif
      else
         if (chat .ge. 2) then
            write (msgbuf, 1003) 'acc', accnam(acc)
            call circem ('MESSAGE', msgbuf)
         endif
      endif
      if ((acc .eq. SBNDEE) .or. (acc .eq. TESLEE)
     $    .or. (acc .eq. XBNDEE)) then
```

⟨*Warn that no parameter set has been endorsed for* e^-e^- *yet* 28d⟩

```
      endif
```

Uses NACC 10c.

28b ⟨*Local variables for* circes 27a⟩+≡

```
      character*6 accnam(NACC)
```

Uses NACC 10c.

28c ⟨*Initializations for* circes 28c⟩≡

```
      data accnam(SBAND)  /'SBAND'/
      data accnam(TESLA)  /'TESLA'/
      data accnam(JLCNLC) /'JLCNLC'/
      data accnam(SBNDEE) /'SBNDEE'/
      data accnam(TESLEE) /'TESLEE'/
      data accnam(XBNDEE) /'XBNDEE'/
      data accnam(NLCH) /'NLC H'/
```

Uses SBAND 10c and TESLA 10c.

28d ⟨*Warn that no parameter set has been endorsed for* e^-e^- *yet* 28d⟩≡

```
      call circem ('WARNING', '***********************************')
      call circem ('WARNING', '* The accelerator parameters have *')
      call circem ('WARNING', '* not been endorsed for use in    *')
      call circem ('WARNING', '* an e-e- collider yet!!!         *')
      call circem ('WARNING', '***********************************')
```

29a ⟨*Update* `/circom/` 26g⟩+≡

```
      if (xver .ge. 0) then
         ver = xver
         if (chat .ge. 1) then
            write (msgbuf, 1000) ’ver’, ver
            call circem (’MESSAGE’, msgbuf)
         endif
      else
         if (chat .ge. 2) then
            write (msgbuf, 1100) ’ver’, ver
            call circem (’MESSAGE’, msgbuf)
         endif
      endif
```

29b ⟨*Update* `/circom/` 26g⟩+≡

```
      if ((xrev .ge. 0) .and.(xrev .ne. rev)) then
         rev = xrev
         if (chat .ge. 1) then
            write (msgbuf, 1004) ’rev’, rev
 1004       format (’updating ‘’, A, ’’’ to ’, I8)
            call circem (’MESSAGE’, msgbuf)
         endif
      else
         if (chat .ge. 2) then
            write (msgbuf, 1104) ’rev’, rev
 1104       format (’keeping ‘’, A, ’’’ at ’, I8)
            call circem (’MESSAGE’, msgbuf)
         endif
      endif
```

Versions 3 and 4 are identical to version 1, except for TESLA at 800 GeV.

29c ⟨*Update* `/circom/` 26g⟩+≡

```
      ver34 = 0
      if ((ver .eq. 1) .or. (ver .eq. 0)) then
⟨Update version 1 derived parameters in /circom/ 30d⟩
      elseif ((ver .eq. 3) .or. (ver .eq. 4)) then
         ver34 = ver
         ver = 1
⟨Update version 3 and 4 derived parameters in /circom/ 43b⟩
      elseif (ver .eq. 5) then
         ver = 1
⟨Update version 5 derived parameters in /circom/ 45a⟩
      elseif (ver .eq. 6) then
         ver = 1
⟨Update version 6 derived parameters in /circom/ 46f⟩
      elseif (ver .eq. 7) then
         ver = 1
⟨Update version 7 derived parameters in /circom/ 49a⟩
      elseif (ver .eq. 8) then
         ver = 1
⟨Update version 8 derived parameters in /circom/ 53b⟩
```

```
      elseif (ver .eq. 9) then
         ver = 1
⟨Update version 9 derived parameters in /circom/ 55c⟩
⟨else handle invalid versions 30b⟩
```

30a ⟨*Local variables for* `circes` 27a⟩+≡

```
      integer ver34
```

30b ⟨`else` *handle invalid versions* 30b⟩≡

```
      elseif (ver .eq. 2) then
⟨Version 2 has been retired 43a⟩
      elseif (ver .gt. 9) then
         call circem ('PANIC', 'versions >9 not available yet')
         return
      else
         call circem ('PANIC', 'version must be positive')
         return
      endif
```

30c ⟨*4-byte aligned part of* `/circom/` 26b⟩+≡

```
      integer e, r, ehi, elo
      common /circom/ e, r, ehi, elo
```

30d ⟨*Update version 1 derived parameters in* `/circom/` 30d⟩≡

```
      if (rev .eq. 0) then
         r = 0
⟨Warn that this revision has not been released yet 31a⟩
      elseif (rev .ge. 1997 04 17) then
         r = 5
      elseif (rev .ge. 1996 09 02) then
         r = 4
      elseif (rev .ge. 1996 07 29) then
         r = 3
      elseif (rev .ge. 1996 07 11) then
         r = 2
      elseif (rev .ge. 1996 04 01) then
         r = 1
      elseif (rev .lt. 1996 04 01) then
         call circem ('ERROR',
     $        'no revision of version 1 before 96/04/01 available')
         call circem ('MESSAGE', 'falling back to default')
         r = 1
      endif
      if (chat .ge. 2) then
         write (msgbuf, 2000) rev, r
 2000    format ('mapping date ', I8, ' to revision index ', I2)
         call circem ('MESSAGE', msgbuf)
      endif
```

30e ⟨*Log revision mapping* 30e⟩≡

```
      if (chat .ge. 2) then
         write (msgbuf, 2000) rev, r
         call circem ('MESSAGE', msgbuf)
      endif
```

31a ⟨*Warn that this revision has not been released yet* 31a⟩≡

```
      call circem ('WARNING', '*************************************')
      call circem ('WARNING', '* This release is not official yet, *')
      call circem ('WARNING', '* do not use it in publications!    *')
      call circem ('WARNING', '*************************************')
```

31b ⟨*Update version 1 derived parameters in* `/circom/` 30d⟩+≡

  ⟨*Map* `roots` *to* `e` 31c⟩

31c ⟨*Map* `roots` *to* `e` 31c⟩≡

```
      if (roots .eq. 350d0) then
         e = GEV350
      elseif ((roots .ge. 340d0) .and. (roots .le. 370d0)) then
         write (msgbuf, 2001) roots, 350d0
         call circem ('MESSAGE', msgbuf)
         e = GEV350
```

31d ⟨`format`*s for* `circes` 31d⟩≡

```
 2001 format ('treating energy ', F6.1, 'GeV as ',  F6.1, 'GeV')
```

31e ⟨*Map* `roots` *to* `e` 31c⟩+≡

```
      elseif (roots .eq. 500d0) then
         e = GEV500
      elseif ((roots .ge. 480d0) .and. (roots .le. 520d0)) then
         write (msgbuf, 2001) roots, 500d0
         call circem ('MESSAGE', msgbuf)
         e = GEV500
      elseif (roots .eq. 800d0) then
         e = GEV800
      elseif ((roots .ge. 750d0) .and. (roots .le. 850d0)) then
         write (msgbuf, 2001) roots, 800d0
         call circem ('MESSAGE', msgbuf)
         e = GEV800
      elseif (roots .eq. 1000d0) then
         e = TEV1
      elseif ((roots .ge. 900d0) .and. (roots .le. 1100d0)) then
         write (msgbuf, 2001) roots, 1000d0
         call circem ('MESSAGE', msgbuf)
         e = TEV1
      elseif (roots .eq. 1600d0) then
         e = TEV16
      elseif ((roots .ge. 1500d0) .and. (roots .le. 1700d0)) then
         write (msgbuf, 2001) roots, 1600d0
         call circem ('MESSAGE', msgbuf)
         e = TEV16
```

31f ⟨*Map* `roots` *to* `e` 31c⟩+≡

```
      else
         call circem ('ERROR',
     $        'only ROOTS = 350, 500, 800, 1000 and 1600GeV available')
```

```
         call circem ('MESSAGE', 'falling back to 500GeV')
         e = GEV500
      endif
```

32a ⟨*Update version 1 derived parameters in* `/circom/` 30d⟩+≡

```
      if (xa1lum(e,acc,r) .lt. 0d0) then
         write (msgbuf, 2002) roots, accnam(acc), r
         call circem ('ERROR', msgbuf)
         call circem ('MESSAGE', 'falling back to 500GeV')
         e = GEV500
      endif
⟨Log energy mapping 32c⟩
```

32b ⟨`format`*s for* `circes` 31d⟩+≡

```
 2002 format ('energy ', F6.1, ' not available for ', A6,
     $        ' in revison ', I2)
```

32c ⟨*Log energy mapping* 32c⟩≡

```
      if (chat .ge. 2) then
         if (e .ge. GEV090) then
            write (msgbuf, 2003) roots, e
            call circem ('MESSAGE', msgbuf)
         else if (elo .ge. GEV090 .and. ehi .ge. GEV090) then
            write (msgbuf, 2013) roots, elo, ehi
            call circem ('MESSAGE', msgbuf)
         end if
      endif
```

32d ⟨`format`*s for* `circes` 31d⟩+≡

```
 2003 format ('mapping energy ', F6.1, ' to energy index ', I2)
 2013 format ('mapping energy ', F6.1, ' to energy indices ',
     $        I2, ' and ', I2)
```

The energies 250 GeV, 1.2 TeV and 1.5 TeV were entered late into the game by teh SLAC people.

32e ⟨*Local variables for* `circes` 27a⟩+≡

```
      integer GEV090, GEV170, GEV350, GEV500, GEV800, TEV1, TEV16
      integer GEV250, TEV12, TEV15
      parameter (GEV090 = -1, GEV170 = 0, GEV350 = 1, GEV500 = 2,
     $           GEV800 =  3, TEV1   = 4, TEV16  = 5,
     $           GEV250 =  6, TEV12  = 7, TEV15  = 8)
      integer A1NEGY, A1NREV
      parameter (A1NEGY = 5, A1NREV = 5)
      integer i
```

32f ⟨*8-byte aligned part of* `/circom/` 26a⟩+≡

```
      double precision lumi
      common /circom/ lumi
      double precision a1(0:7)
      common /circom/ a1
```

33a ⟨*Update version 1 derived parameters in* `/circom/` 30d⟩+≡

```
      lumi = xa1lum (e,acc,r)
      do 10 i = 0, 7
         a1(i) = xa1(i,e,acc,r)
   10 continue
```

33b ⟨*Local variables for* `circes` 27a⟩+≡

```
      real xa1lum(A1NEGY,NACC,0:A1NREV)
      real xa1(0:7,A1NEGY,NACC,0:A1NREV)
```

Uses `NACC` 10c.

Revision 1. The mother of all revisions.

33c ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV500,SBAND,1) /  5.212299E+01 /
      data (xa1(i,GEV500,SBAND,1),i=0,7) /
     $    .39192E+00,   .66026E+00,   .11828E+02,  -.62543E+00,
     $    .52292E+00,  -.69245E+00,   .14983E+02,   .65421E+00 /
      data xa1lum(GEV500,TESLA,1) /  6.066178E+01 /
      data (xa1(i,GEV500,TESLA,1),i=0,7) /
     $    .30196E+00,   .12249E+01,   .21423E+02,  -.57848E+00,
     $    .68766E+00,  -.69788E+00,   .23121E+02,   .78399E+00 /
      data xa1lum(GEV500,XBAND,1) /  5.884699E+01 /
      data (xa1(i,GEV500,XBAND,1),i=0,7) /
     $    .48594E+00,   .52435E+00,   .83585E+01,  -.61347E+00,
     $    .30703E+00,  -.68804E+00,   .84109E+01,   .44312E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

33d ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(TEV1,SBAND,1)   /  1.534650E+02 /
      data (xa1(i,TEV1,SBAND,1),i=0,7) /
     $    .24399E+00,   .87464E+00,   .66751E+01,  -.56808E+00,
     $    .59295E+00,  -.68921E+00,   .94232E+01,   .83351E+00 /
      data xa1lum(TEV1,TESLA,1)   /  1.253381E+03 /
      data (xa1(i,TEV1,TESLA,1),i=0,7) /
     $    .39843E+00,   .70097E+00,   .11602E+02,  -.61061E+00,
     $    .40737E+00,  -.69319E+00,   .14800E+02,   .51382E+00 /
      data xa1lum(TEV1,XBAND,1)   /  1.901783E+02 /
      data (xa1(i,TEV1,XBAND,1),i=0,7) /
     $    .32211E+00,   .61798E+00,   .28298E+01,  -.54644E+00,
     $    .45674E+00,  -.67301E+00,   .41703E+01,   .74536E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

Unavailable

33e ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(GEV350,i,1),i=1,NACC) / NACC*-1d0 /
      data (xa1lum(GEV800,i,1),i=1,NACC) / NACC*-1d0 /
```

Uses `NACC` 10c.

Unavailable as well

33f ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(GEV500,i,1),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(TEV1,i,1),i=SBNDEE,NACC) / 4*-1d0 /
```

Uses `NACC` 10c.

No 1.6TeV parameters in this revision

34a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(TEV16,i,1),i=1,NACC) / 7*-1d0 /
```

Uses `NACC` 10c.

34b ⟨*Subroutines* 24d⟩+≡

```
      subroutine circel (l)
      implicit none
      double precision l
  ⟨/circom/ 25b⟩
      l = lumi
      end
```

Defines:
`circel`, used in chunks 34b and 10a.

34c ⟨*Subroutines* 24d⟩+≡

```
      double precision function circee (x1, x2)
      implicit none
      double precision x1, x2
  ⟨/circom/ 25b⟩
      double precision d1, d2
  ⟨Initialization check 26e⟩
      circee = -1.0
      if ((ver .eq. 1) .or. (ver .eq. 0)) then
  ⟨Calculate version 1 of the e+e- distribution 34d⟩
  ⟨else handle invalid versions 30b⟩
      end
```

Defines:
`circee`, used in chunks 11–13, 15, 24d, and 34d.

The first version of the parametrization is factorized

$$D^{\alpha 1\rho}_{p_1p_2}(x_1,x_2,s) = d^{\alpha 1\rho}_{p_1}(x_1)\,d^{\alpha 1\rho}_{p_2}(x_2) \tag{15}$$

where the distributions are

$$d^{\alpha 1\rho}_{e^\pm}(x) = a_0^{\alpha\rho}\delta(1-x) + a_1^{\alpha\rho} x^{a_2^{\alpha\rho}}(1-x)^{a_3^{\alpha\rho}} \tag{16}$$

$$d_\gamma(x) = a_4^{\alpha\rho} x^{a_5^{\alpha\rho}}(1-x)^{a_6^{\alpha\rho}} \tag{17}$$

34d ⟨*Calculate version 1 of the* e^+e^- *distribution* 34d⟩≡

```
      if (x1 .eq. 1d0) then
         d1 = a1(0)
      elseif (x1 .lt. 1d0 .and. x1 .gt. 0d0) then
         d1 = a1(1) * x1**a1(2) * (1d0 - x1)**a1(3)
      elseif (x1 .eq. -1d0) then
         d1 = 1d0 - a1(0)
      else
         d1 = 0d0
      endif
      if (x2 .eq. 1d0) then
         d2 = a1(0)
      elseif (x2 .lt. 1d0 .and. x2 .gt. 0d0) then
```

```
         d2 = a1(1) * x2**a1(2) * (1d0 - x2)**a1(3)
      elseif (x2 .eq. -1d0) then
         d2 = 1d0 - a1(0)
      else
         d2 = 0d0
      endif
      circee = d1 * d2
```

Uses `circee` 34c.

35a ⟨*Subroutines* 24d⟩+≡

```
      double precision function circeg (x1, x2)
      implicit none
      double precision x1, x2
⟨/circom/ 25b⟩
      double precision d1, d2
⟨Initialization check 26e⟩
      circeg = -1.0
      if ((ver .eq. 1) .or. (ver .eq. 0)) then
⟨Calculate version 1 of the e±γ distribution 35b⟩
⟨else handle invalid versions 30b⟩
      end
```

Defines:
`circeg`, used in chunks 11, 24d, and 35b.

35b ⟨*Calculate version 1 of the* $e^{\pm}\gamma$ *distribution* 35b⟩≡

```
      if (x1 .eq. 1d0) then
         d1 = a1(0)
      elseif (x1 .lt. 1d0 .and. x1 .gt. 0d0) then
         d1 = a1(1) * x1**a1(2) * (1d0 - x1)**a1(3)
      elseif (x1 .eq. -1d0) then
         d1 = 1d0 - a1(0)
      else
         d1 = 0d0
      endif
      if (x2 .lt. 1d0 .and. x2 .gt. 0d0) then
         d2 = a1(4) * x2**a1(5) * (1d0 - x2)**a1(6)
      elseif (x2 .eq. -1d0) then
         d2 = a1(7)
      else
         d2 = 0d0
      endif
      circeg = d1 * d2
```

Uses `circeg` 35a.

35c ⟨*Subroutines* 24d⟩+≡

```
      double precision function circgg (x1, x2)
      implicit none
      double precision x1, x2
⟨/circom/ 25b⟩
      double precision d1, d2
⟨Initialization check 26e⟩
```

```
        circgg = -1.0
        if ((ver .eq. 1) .or. (ver .eq. 0)) then
  ⟨Calculate version 1 of the γγ distribution 36a⟩
  ⟨else handle invalid versions 30b⟩
        end
```

Defines:
`circgg`, used in chunks 11, 24d, 36a, 67b, and 68a.

36a ⟨*Calculate version 1 of the $\gamma\gamma$ distribution* 36a⟩≡

```
        if (x1 .lt. 1d0 .and. x1 .gt. 0d0) then
           d1 = a1(4) * x1**a1(5) * (1d0 - x1)**a1(6)
        elseif (x1 .eq. -1d0) then
           d1 = a1(7)
        else
           d1 = 0d0
        endif
        if (x2 .lt. 1d0 .and. x2 .gt. 0d0) then
           d2 = a1(4) * x2**a1(5) * (1d0 - x2)**a1(6)
        elseif (x2 .eq. -1d0) then
           d2 = a1(7)
        else
           d2 = 0d0
        endif
        circgg = d1 * d2
```

Uses `circgg` 35c.

Revision 2. New Tesla parameters, including 350 GeV and 800 GeV.

36b ⟨*Initializations for* `circes` 28c⟩+≡

```
        data xa1lum(GEV500,SBAND,2) /   .31057E+02 /
        data (xa1(i,GEV500,SBAND,2),i=0,7) /
       $    .38504E+00,   .79723E+00,   .14191E+02,  -.60456E+00,
       $    .53411E+00,  -.68873E+00,   .15105E+02,   .65151E+00 /
        data xa1lum(TEV1,SBAND,2) /   .24297E+03 /
        data (xa1(i,TEV1,SBAND,2),i=0,7) /
       $    .24374E+00,   .89466E+00,   .70242E+01,  -.56754E+00,
       $    .60910E+00,  -.68682E+00,   .96083E+01,   .83985E+00 /
        data xa1lum(GEV350,TESLA,2) /   .73369E+02 /
        data (xa1(i,GEV350,TESLA,2),i=0,7) /
       $    .36083E+00,   .12819E+01,   .37880E+02,  -.59492E+00,
       $    .69109E+00,  -.69379E+00,   .40061E+02,   .65036E+00 /
        data xa1lum(GEV500,TESLA,2) /   .10493E+03 /
        data (xa1(i,GEV500,TESLA,2),i=0,7) /
       $    .29569E+00,   .11854E+01,   .21282E+02,  -.58553E+00,
       $    .71341E+00,  -.69279E+00,   .24061E+02,   .77709E+00 /
        data xa1lum(GEV800,TESLA,2) /   .28010E+03 /
        data (xa1(i,GEV800,TESLA,2),i=0,7) /
       $    .22745E+00,   .11265E+01,   .10483E+02,  -.55711E+00,
       $    .69579E+00,  -.69068E+00,   .13093E+02,   .89605E+00 /
        data xa1lum(TEV1,TESLA,2) /   .10992E+03 /
        data (xa1(i,TEV1,TESLA,2),i=0,7) /
       $    .40969E+00,   .66105E+00,   .11972E+02,  -.62041E+00,
       $    .40463E+00,  -.69354E+00,   .14669E+02,   .51281E+00 /
        data xa1lum(GEV500,XBAND,2) /   .35689E+02 /
        data (xa1(i,GEV500,XBAND,2),i=0,7) /
       $    .48960E+00,   .46815E+00,   .75249E+01,  -.62769E+00,
       $    .30341E+00,  -.68754E+00,   .85545E+01,   .43453E+00 /
        data xa1lum(TEV1,XBAND,2) /   .11724E+03 /
        data (xa1(i,TEV1,XBAND,2),i=0,7) /
       $    .31939E+00,   .62415E+00,   .30763E+01,  -.55314E+00,
       $    .45634E+00,  -.67089E+00,   .41529E+01,   .73807E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

Unavailable

37a ⟨*Initializations for* `circes` 28c⟩+≡

```
 data xa1lum(GEV350,SBAND,2) / -1d0 /
 data xa1lum(GEV350,XBAND,2) / -1d0 /
 data xa1lum(GEV800,SBAND,2) / -1d0 /
 data xa1lum(GEV800,XBAND,2) / -1d0 /
```

Uses `SBAND` 10c and `XBAND` 10c.

Unavailable as well

37b ⟨*Initializations for* `circes` 28c⟩+≡

```
 data (xa1lum(GEV350,i,2),i=SBNDEE,NACC) / 4*-1d0 /
 data (xa1lum(GEV500,i,2),i=SBNDEE,NACC) / 4*-1d0 /
 data (xa1lum(GEV800,i,2),i=SBNDEE,NACC) / 4*-1d0 /
 data (xa1lum(TEV1,i,2),i=SBNDEE,NACC) / 4*-1d0 /
```

Uses `NACC` 10c.

No 1.6TeV parameters in this revision

37c ⟨*Initializations for* `circes` 28c⟩+≡

```
 data (xa1lum(TEV16,i,2),i=1,NACC) / 7*-1d0 /
```

Uses `NACC` 10c.

Revision 3. Features:

- improved error estimates.
- cleaner fitting procedure, including delta function pieces.

37d ⟨*Initializations for* `circes` 28c⟩+≡

```
 data xa1lum(GEV500,SBAND, 3) /   .31469E+02 /
 data (xa1(i,GEV500,SBAND, 3),i=0,7) /
$    .38299E+00,   .72035E+00,   .12618E+02,  -.61611E+00,
$    .51971E+00,  -.68960E+00,   .15066E+02,   .63784E+00 /
 data xa1lum(TEV1,  SBAND, 3) /   .24566E+03 /
 data (xa1(i,TEV1,  SBAND, 3),i=0,7) /
$    .24013E+00,   .95763E+00,   .69085E+01,  -.55151E+00,
$    .59497E+00,  -.68622E+00,   .94494E+01,   .82158E+00 /
 data xa1lum(GEV350,TESLA, 3) /   .74700E+02 /
 data (xa1(i,GEV350,TESLA, 3),i=0,7) /
$    .34689E+00,   .12484E+01,   .33720E+02,  -.59523E+00,
$    .66266E+00,  -.69524E+00,   .38488E+02,   .63775E+00 /
 data xa1lum(GEV500,TESLA, 3) /   .10608E+03 /
 data (xa1(i,GEV500,TESLA, 3),i=0,7) /
```

```
     $    .28282E+00,   .11700E+01,   .19258E+02,  -.58390E+00,
     $    .68777E+00,  -.69402E+00,   .23638E+02,   .75929E+00 /
      data xa1lum(GEV800,TESLA, 3) /   .28911E+03 /
      data (xa1(i,GEV800,TESLA, 3),i=0,7) /
     $    .21018E+00,   .12039E+01,   .96763E+01,  -.54024E+00,
     $    .67220E+00,  -.69083E+00,   .12733E+02,   .87355E+00 /
      data xa1lum(TEV1,  TESLA, 3) /   .10936E+03 /
      data (xa1(i,TEV1,  TESLA, 3),i=0,7) /
     $    .41040E+00,   .68099E+00,   .11610E+02,  -.61237E+00,
     $    .40155E+00,  -.69073E+00,   .14698E+02,   .49989E+00 /
      data xa1lum(GEV500,XBAND, 3) /   .36145E+02 /
      data (xa1(i,GEV500,XBAND, 3),i=0,7) /
     $    .51285E+00,   .45812E+00,   .75135E+01,  -.62247E+00,
     $    .30444E+00,  -.68530E+00,   .85519E+01,   .43062E+00 /
      data xa1lum(TEV1,  XBAND, 3) /   .11799E+03 /
      data (xa1(i,TEV1,  XBAND, 3),i=0,7) /
     $    .31241E+00,   .61241E+00,   .29938E+01,  -.55848E+00,
     $    .44801E+00,  -.67116E+00,   .41119E+01,   .72753E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

Still unavailable

38a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV350,SBAND,3) / -1d0 /
      data xa1lum(GEV350,XBAND,3) / -1d0 /
      data xa1lum(GEV800,SBAND,3) / -1d0 /
      data xa1lum(GEV800,XBAND,3) / -1d0 /
```

Uses `SBAND` 10c and `XBAND` 10c.

Unavailable as well

38b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(GEV350,i,3),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(GEV500,i,3),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(GEV800,i,3),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(TEV1,i,3),i=SBNDEE,NACC) / 4*-1d0 /
```

Uses `NACC` 10c.

No 1.6TeV parameters in this revision

38c ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(TEV16,i,3),i=1,NACC) / 7*-1d0 /
```

Uses `NACC` 10c.

Revision 4. Features:

- a bug in `Guinea-Pig`'s synchrotron radiation spectrum has been fixed.

38d ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV500,SBAND, 4) /   .31528E+02 /
      data (xa1(i,GEV500,SBAND, 4),i=0,7) /
     $    .38169E+00,   .73949E+00,   .12543E+02,  -.61112E+00,
     $    .51256E+00,  -.69009E+00,   .14892E+02,   .63314E+00 /
      data xa1lum(TEV1,  SBAND, 4) /   .24613E+03 /
      data (xa1(i,TEV1,  SBAND, 4),i=0,7) /
     $    .24256E+00,   .94117E+00,   .66775E+01,  -.55160E+00,
     $    .57484E+00,  -.68891E+00,   .92271E+01,   .81162E+00 /
      data xa1lum(GEV350,TESLA, 4) /   .74549E+02 /
      data (xa1(i,GEV350,TESLA, 4),i=0,7) /
     $    .34120E+00,   .12230E+01,   .32932E+02,  -.59850E+00,
     $    .65947E+00,  -.69574E+00,   .38116E+02,   .63879E+00 /
      data xa1lum(GEV500,TESLA, 4) /   .10668E+03 /
      data (xa1(i,GEV500,TESLA, 4),i=0,7) /
     $    .28082E+00,   .11074E+01,   .18399E+02,  -.59118E+00,
     $    .68880E+00,  -.69375E+00,   .23463E+02,   .76073E+00 /
      data xa1lum(GEV800,TESLA, 4) /   .29006E+03 /
      data (xa1(i,GEV800,TESLA, 4),i=0,7) /
     $    .21272E+00,   .11443E+01,   .92564E+01,  -.54657E+00,
     $    .66799E+00,  -.69137E+00,   .12498E+02,   .87571E+00 /
      data xa1lum(TEV1,  TESLA, 4) /   .11009E+03 /
      data (xa1(i,TEV1,  TESLA, 4),i=0,7) /
     $    .41058E+00,   .64745E+00,   .11271E+02,  -.61996E+00,
     $    .39801E+00,  -.69150E+00,   .14560E+02,   .49924E+00 /
      data xa1lum(GEV500,XBAND, 4) /   .36179E+02 /
      data (xa1(i,GEV500,XBAND, 4),i=0,7) /
     $    .51155E+00,   .43313E+00,   .70446E+01,  -.63003E+00,
     $    .29449E+00,  -.68747E+00,   .83489E+01,   .42458E+00 /
      data xa1lum(TEV1,  XBAND, 4) /   .11748E+03 /
      data (xa1(i,TEV1,  XBAND, 4),i=0,7) /
     $    .32917E+00,   .54322E+00,   .28493E+01,  -.57959E+00,
     $    .39266E+00,  -.68217E+00,   .38475E+01,   .68478E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

Still unavailable

39a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV350,SBAND,4) / -1d0 /
      data xa1lum(GEV350,XBAND,4) / -1d0 /
      data xa1lum(GEV800,SBAND,4) / -1d0 /
      data xa1lum(GEV800,XBAND,4) / -1d0 /
```

Uses `SBAND` 10c and `XBAND` 10c.

Unavailable as well

39b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(GEV350,i,4),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(GEV500,i,4),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(GEV800,i,4),i=SBNDEE,NACC) / 4*-1d0 /
      data (xa1lum(TEV1,i,4),i=SBNDEE,NACC) / 4*-1d0 /
```

Uses `NACC` 10c.

No 1.6TeV parameters in this revision

39c ⟨*Initializations for* `circes` 28c⟩+≡

```
      data (xa1lum(TEV16,i,4),i=1,NACC) / 7*-1d0 /
```

Uses `NACC` 10c.

Revision 5. Features:

- a bug in `Guinea-Pig` has been fixed.
- updated parameter sets

40 ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV350,SBAND, 5) /  0.21897E+02 /
      data (xa1(i,GEV350,SBAND, 5),i=0,7) /
     $   0.57183E+00,  0.53877E+00,  0.19422E+02, -0.63064E+00,
     $   0.49112E+00, -0.69109E+00,  0.24331E+02,  0.52718E+00 /
      data xa1lum(GEV500,SBAND, 5) /  0.31383E+02 /
      data (xa1(i,GEV500,SBAND, 5),i=0,7) /
     $   0.51882E+00,  0.49915E+00,  0.11153E+02, -0.63017E+00,
     $   0.50217E+00, -0.69113E+00,  0.14935E+02,  0.62373E+00 /
      data xa1lum(GEV800,SBAND, 5) /  0.95091E+02 /
      data (xa1(i,GEV800,SBAND, 5),i=0,7) /
     $   0.47137E+00,  0.46150E+00,  0.56562E+01, -0.61758E+00,
     $   0.46863E+00, -0.68897E+00,  0.85876E+01,  0.67577E+00 /
      data xa1lum(TEV1,SBAND, 5) /  0.11900E+03 /
      data (xa1(i,TEV1,SBAND, 5),i=0,7) /
     $   0.43956E+00,  0.45471E+00,  0.42170E+01, -0.61180E+00,
     $   0.48711E+00, -0.68696E+00,  0.67145E+01,  0.74551E+00 /
      data xa1lum(TEV16,SBAND, 5) /  0.11900E+03 /
      data (xa1(i,TEV16,SBAND, 5),i=0,7) /
     $   0.43956E+00,  0.45471E+00,  0.42170E+01, -0.61180E+00,
     $   0.48711E+00, -0.68696E+00,  0.67145E+01,  0.74551E+00 /
      data xa1lum(GEV350,TESLA, 5) /  0.97452E+02 /
      data (xa1(i,GEV350,TESLA, 5),i=0,7) /
     $   0.39071E+00,  0.84996E+00,  0.17614E+02, -0.60609E+00,
     $   0.73920E+00, -0.69490E+00,  0.28940E+02,  0.77286E+00 /
      data xa1lum(GEV500,TESLA, 5) /  0.10625E+03 /
      data (xa1(i,GEV500,TESLA, 5),i=0,7) /
     $   0.42770E+00,  0.71457E+00,  0.15284E+02, -0.61664E+00,
     $   0.68166E+00, -0.69208E+00,  0.24165E+02,  0.73806E+00 /
      data xa1lum(GEV800,TESLA, 5) /  0.17086E+03 /
      data (xa1(i,GEV800,TESLA, 5),i=0,7) /
     $   0.36025E+00,  0.69118E+00,  0.76221E+01, -0.59440E+00,
     $   0.71269E+00, -0.69077E+00,  0.13117E+02,  0.91780E+00 /
      data xa1lum(TEV1,TESLA, 5) /  0.21433E+03 /
      data (xa1(i,TEV1,TESLA, 5),i=0,7) /
     $   0.33145E+00,  0.67075E+00,  0.55438E+01, -0.58468E+00,
     $   0.72503E+00, -0.69084E+00,  0.99992E+01,  0.10112E+01 /
      data xa1lum(TEV16,TESLA, 5) /  0.34086E+03 /
      data (xa1(i,TEV16,TESLA, 5),i=0,7) /
     $   0.49058E+00,  0.42609E+00,  0.50550E+01, -0.61867E+00,
     $   0.39225E+00, -0.68916E+00,  0.75514E+01,  0.58754E+00 /
      data xa1lum(GEV350,XBAND, 5) /  0.31901E+02 /
      data (xa1(i,GEV350,XBAND, 5),i=0,7) /
     $   0.65349E+00,  0.31752E+00,  0.94342E+01, -0.64291E+00,
     $   0.30364E+00, -0.68989E+00,  0.11446E+02,  0.40486E+00 /
      data xa1lum(GEV500,XBAND, 5) /  0.36386E+02 /
      data (xa1(i,GEV500,XBAND, 5),i=0,7) /
     $   0.65132E+00,  0.28728E+00,  0.69853E+01, -0.64440E+00,
     $   0.28736E+00, -0.68758E+00,  0.83227E+01,  0.41492E+00 /
      data xa1lum(GEV800,XBAND, 5) /  0.10854E+03 /
      data (xa1(i,GEV800,XBAND, 5),i=0,7) /
     $   0.49478E+00,  0.36221E+00,  0.30116E+01, -0.61548E+00,
     $   0.39890E+00, -0.68418E+00,  0.45183E+01,  0.67243E+00 /
      data xa1lum(TEV1,XBAND, 5) /  0.11899E+03 /
      data (xa1(i,TEV1,XBAND, 5),i=0,7) /
     $   0.49992E+00,  0.34299E+00,  0.26184E+01, -0.61584E+00,
     $   0.38450E+00, -0.68342E+00,  0.38589E+01,  0.67408E+00 /
      data xa1lum(TEV16,XBAND, 5) /  0.13675E+03 /
      data (xa1(i,TEV16,XBAND, 5),i=0,7) /
     $   0.50580E+00,  0.30760E+00,  0.18339E+01, -0.61421E+00,
     $   0.35233E+00, -0.68315E+00,  0.26708E+01,  0.67918E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

Revision 0. Features:

- e^-e^- mode

41a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV500,SBNDEE, 0) /   .92914E+01 /
      data (xa1(i,GEV500,SBNDEE, 0),i=0,7) /
     $    .34866E+00,   .78710E+00,   .10304E+02,  -.59464E+00,
     $    .40234E+00,  -.69741E+00,   .20645E+02,   .47274E+00 /
      data xa1lum(TEV1,  SBNDEE, 0) /   .45586E+02 /
      data (xa1(i,TEV1,  SBNDEE, 0),i=0,7) /
     $    .21084E+00,   .99168E+00,   .54407E+01,  -.52851E+00,
     $    .47493E+00,  -.69595E+00,   .12480E+02,   .64027E+00 /
      data xa1lum(GEV350,TESLEE, 0) /   .15175E+02 /
      data (xa1(i,GEV350,TESLEE, 0),i=0,7) /
     $    .33093E+00,   .11137E+01,   .25275E+02,  -.59942E+00,
     $    .49623E+00,  -.70403E+00,   .60188E+02,   .44637E+00 /
      data xa1lum(GEV500,TESLEE, 0) /   .21622E+02 /
      data (xa1(i,GEV500,TESLEE, 0),i=0,7) /
     $    .27175E+00,   .10697E+01,   .14858E+02,  -.58418E+00,
     $    .50824E+00,  -.70387E+00,   .36129E+02,   .53002E+00 /
      data xa1lum(GEV800,TESLEE, 0) /   .43979E+02 /
      data (xa1(i,GEV800,TESLEE, 0),i=0,7) /
     $    .22994E+00,   .10129E+01,   .81905E+01,  -.55751E+00,
     $    .46551E+00,  -.70461E+00,   .19394E+02,   .58387E+00 /
      data xa1lum(TEV1,  TESLEE, 0) /   .25465E+02 /
      data (xa1(i,TEV1,  TESLEE, 0),i=0,7) /
     $    .37294E+00,   .67522E+00,   .87504E+01,  -.60576E+00,
     $    .35095E+00,  -.69821E+00,   .18526E+02,   .42784E+00 /
      data xa1lum(GEV500,XBNDEE, 0) /   .13970E+02 /
      data (xa1(i,GEV500,XBNDEE, 0),i=0,7) /
     $    .47296E+00,   .46800E+00,   .58897E+01,  -.61689E+00,
     $    .27181E+00,  -.68923E+00,   .10087E+02,   .37462E+00 /
      data xa1lum(TEV1,  XBNDEE, 0) /   .41056E+02 /
      data (xa1(i,TEV1,  XBNDEE, 0),i=0,7) /
     $    .27965E+00,   .74816E+00,   .27415E+01,  -.50491E+00,
     $    .38320E+00,  -.67945E+00,   .47506E+01,   .62218E+00 /
```

Still unavailable

41b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV350,SBNDEE,0) / -1d0 /
      data xa1lum(GEV350,XBNDEE,0) / -1d0 /
      data xa1lum(GEV800,SBNDEE,0) / -1d0 /
      data xa1lum(GEV800,XBNDEE,0) / -1d0 /
```

42a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV500,SBAND, 0) /   .31528E+02 /
      data (xa1(i,GEV500,SBAND, 0),i=0,7) /
     $    .38169E+00,   .73949E+00,   .12543E+02,  -.61112E+00,
     $    .51256E+00,  -.69009E+00,   .14892E+02,   .63314E+00 /
      data xa1lum(TEV1,  SBAND, 0) /   .24613E+03 /
      data (xa1(i,TEV1,  SBAND, 0),i=0,7) /
     $    .24256E+00,   .94117E+00,   .66775E+01,  -.55160E+00,
     $    .57484E+00,  -.68891E+00,   .92271E+01,   .81162E+00 /
      data xa1lum(GEV350,TESLA, 0) /   .74549E+02 /
      data (xa1(i,GEV350,TESLA, 0),i=0,7) /
     $    .34120E+00,   .12230E+01,   .32932E+02,  -.59850E+00,
     $    .65947E+00,  -.69574E+00,   .38116E+02,   .63879E+00 /
      data xa1lum(GEV500,TESLA, 0) /   .10668E+03 /
      data (xa1(i,GEV500,TESLA, 0),i=0,7) /
     $    .28082E+00,   .11074E+01,   .18399E+02,  -.59118E+00,
     $    .68880E+00,  -.69375E+00,   .23463E+02,   .76073E+00 /
      data xa1lum(GEV800,TESLA, 0) /   .29006E+03 /
      data (xa1(i,GEV800,TESLA, 0),i=0,7) /
     $    .21272E+00,   .11443E+01,   .92564E+01,  -.54657E+00,
     $    .66799E+00,  -.69137E+00,   .12498E+02,   .87571E+00 /
      data xa1lum(TEV1,  TESLA, 0) /   .11009E+03 /
      data (xa1(i,TEV1,  TESLA, 0),i=0,7) /
     $    .41058E+00,   .64745E+00,   .11271E+02,  -.61996E+00,
     $    .39801E+00,  -.69150E+00,   .14560E+02,   .49924E+00 /
      data xa1lum(GEV500,XBAND, 0) /   .36179E+02 /
      data (xa1(i,GEV500,XBAND, 0),i=0,7) /
     $    .51155E+00,   .43313E+00,   .70446E+01,  -.63003E+00,
     $    .29449E+00,  -.68747E+00,   .83489E+01,   .42458E+00 /
      data xa1lum(TEV1,  XBAND, 0) /   .11748E+03 /
      data (xa1(i,TEV1,  XBAND, 0),i=0,7) /
     $    .32917E+00,   .54322E+00,   .28493E+01,  -.57959E+00,
     $    .39266E+00,  -.68217E+00,   .38475E+01,   .68478E+00 /
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

Still unavailable

42b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa1lum(GEV350,SBAND,0) / -1d0 /
      data xa1lum(GEV350,XBAND,0) / -1d0 /
      data xa1lum(GEV800,SBAND,0) / -1d0 /
      data xa1lum(GEV800,XBAND,0) / -1d0 /
```

Uses `SBAND` 10c and `XBAND` 10c.

6.2.2 Version 2

43a ⟨*Version 2 has been retired* 43a⟩≡

```
        call circem ('PANIC', '*******************************')
        call circem ('PANIC', '* version 2 has been retired,   *')
        call circem ('PANIC', '* please use version 1 instead! *')
        call circem ('PANIC', '*******************************')
        return
```

6.2.3 Versions 3 and 4

43b ⟨*Update version 3 and 4 derived parameters in* `/circom/` 43b⟩≡

```
        if (rev .eq. 0) then
           r = 0
  ⟨Warn that this revision has not been released yet 31a⟩
        elseif (rev .ge. 1997 04 17) then
           r = 5
           if (ver34 .eq. 3) then
              call circem ('WARNING', 'version 3 retired after 97/04/17')
              call circem ('MESSAGE', 'falling back to version 4')
           endif
        elseif (rev .ge. 1996 10 22) then
           r = ver34
           if ((roots .ne. 800d0) .or. (acc .ne. TESLA)) then
              call circem ('ERROR', 'versions 3 and 4 before 97/04/17')
              call circem ('ERROR', 'apply to TESLA at 800 GeV only')
              call circem ('MESSAGE', 'falling back to TESLA at 800GeV')
              acc = TESLA
              e = GEV800
           endif
        elseif (rev .lt. 1996 10 22) then
           call circem ('ERROR',
     $     'no revision of versions 3 and 4 available before 96/10/22')
           call circem ('MESSAGE', 'falling back to default')
           r = 5
        endif
  ⟨Log revision mapping 30e⟩
```

Uses `TESLA` 10c.

43c ⟨*Update version 3 and 4 derived parameters in* `/circom/` 43b⟩+≡

```
  ⟨Map roots to e 31c⟩
        if (xa3lum(e,acc,r) .lt. 0d0) then
           write (msgbuf, 2002) roots, accnam(acc), r
           call circem ('ERROR', msgbuf)
           call circem ('MESSAGE', 'falling back to 500GeV')
           e = GEV500
        endif
  ⟨Log energy mapping 32c⟩
```

43d ⟨*Local variables for* `circes` 27a⟩+≡

```
        integer A3NEGY, A3NREV
        parameter (A3NEGY = 5, A3NREV = 5)
```

44a ⟨*Update version 3 and 4 derived parameters in* `/circom/` 43b⟩+≡

```
        lumi = xa3lum (e,acc,r)
        do 20 i = 0, 7
           a1(i) = xa3(i,e,acc,r)
     20 continue
```

44b ⟨*Local variables for* `circes` 27a⟩+≡

```
        real xa3lum(A3NEGY,NACC,0:A3NREV)
        real xa3(0:7,A3NEGY,NACC,0:A3NREV)
```

Uses `NACC` 10c.

Revisions 3 & 4. The mother of all revisions.

44c ⟨*Initializations for* `circes` 28c⟩+≡

```
        data xa3lum(GEV800,TESLA, 3) /   .17196E+03 /
        data (xa3(i,GEV800,TESLA, 3),i=0,7) /
       $    .21633E+00,   .11333E+01,   .95928E+01,  -.55095E+00,
       $    .73044E+00,  -.69101E+00,   .12868E+02,   .94737E+00 /
        data xa3lum(GEV800,TESLA, 4) /   .16408E+03 /
        data (xa3(i,GEV800,TESLA, 4),i=0,7) /
       $    .41828E+00,   .72418E+00,   .14137E+02,  -.61189E+00,
       $    .36697E+00,  -.69205E+00,   .17713E+02,   .43583E+00 /
```

Uses `TESLA` 10c.

Revision 5.

44d ⟨*Initializations for* `circes` 28c⟩+≡

```
        data xa3lum(GEV350,TESLA, 5) /  0.66447E+02 /
        data (xa3(i,GEV350,TESLA, 5),i=0,7) /
       $   0.69418E+00,  0.50553E+00,  0.48430E+02, -0.63911E+00,
       $   0.34074E+00, -0.69533E+00,  0.55502E+02,  0.29397E+00 /
        data xa3lum(GEV500,TESLA, 5) /  0.95241E+02 /
        data (xa3(i,GEV500,TESLA, 5),i=0,7) /
       $   0.64882E+00,  0.45462E+00,  0.27103E+02, -0.64535E+00,
       $   0.35101E+00, -0.69467E+00,  0.33658E+02,  0.35024E+00 /
        data xa3lum(GEV800,TESLA, 5) /  0.16974E+03 /
        data (xa3(i,GEV800,TESLA, 5),i=0,7) /
       $   0.58706E+00,  0.43771E+00,  0.13422E+02, -0.63804E+00,
       $   0.35541E+00, -0.69467E+00,  0.17528E+02,  0.43051E+00 /
        data xa3lum(TEV1,TESLA, 5) /  0.21222E+03 /
        data (xa3(i,TEV1,TESLA, 5),i=0,7) /
       $   0.55525E+00,  0.42577E+00,  0.96341E+01, -0.63587E+00,
       $   0.36448E+00, -0.69365E+00,  0.13161E+02,  0.47715E+00 /
        data xa3lum(TEV16,TESLA, 5) /  0.34086E+03 /
        data (xa3(i,TEV16,TESLA, 5),i=0,7) /
       $   0.49058E+00,  0.42609E+00,  0.50550E+01, -0.61867E+00,
       $   0.39225E+00, -0.68916E+00,  0.75514E+01,  0.58754E+00 /
```

Uses `TESLA` 10c.

Revision 0. Currently identical to revision 5.

44e ⟨*Initializations for* `circes` 28c⟩+≡

```
        data xa3lum(GEV350,TESLA, 0) /  0.66447E+02 /
```

```
      data (xa3(i,GEV350,TESLA, 0),i=0,7) /
     $   0.69418E+00,  0.50553E+00,  0.48430E+02, -0.63911E+00,
     $   0.34074E+00, -0.69533E+00,  0.55502E+02,  0.29397E+00 /
      data xa3lum(GEV500,TESLA, 0) /  0.95241E+02 /
      data (xa3(i,GEV500,TESLA, 0),i=0,7) /
     $   0.64882E+00,  0.45462E+00,  0.27103E+02, -0.64535E+00,
     $   0.35101E+00, -0.69467E+00,  0.33658E+02,  0.35024E+00 /
      data xa3lum(GEV800,TESLA, 0) /  0.16974E+03 /
      data (xa3(i,GEV800,TESLA, 0),i=0,7) /
     $   0.58706E+00,  0.43771E+00,  0.13422E+02, -0.63804E+00,
     $   0.35541E+00, -0.69467E+00,  0.17528E+02,  0.43051E+00 /
      data xa3lum(TEV1,TESLA, 0) /  0.21222E+03 /
      data (xa3(i,TEV1,TESLA, 0),i=0,7) /
     $   0.55525E+00,  0.42577E+00,  0.96341E+01, -0.63587E+00,
     $   0.36448E+00, -0.69365E+00,  0.13161E+02,  0.47715E+00 /
      data xa3lum(TEV16,TESLA, 0) /  0.34086E+03 /
      data (xa3(i,TEV16,TESLA, 0),i=0,7) /
     $   0.49058E+00,  0.42609E+00,  0.50550E+01, -0.61867E+00,
     $   0.39225E+00, -0.68916E+00,  0.75514E+01,  0.58754E+00 /
```

Uses `TESLA` 10c.

6.2.4 Version 5

45a ⟨*Update version 5 derived parameters in* `/circom/` 45a⟩≡

```
      if (rev .eq. 0) then
         r = 0
⟨Warn that this revision has not been released yet 31a⟩
      elseif (rev .ge. 1998 05 05) then
         r = 1
      elseif (rev .lt. 1998 05 05) then
         call circem ('ERROR',
     $     'no revision of version 5 available before 98/05/05')
         call circem ('MESSAGE', 'falling back to default')
         r = 1
      endif
⟨Log revision mapping 30e⟩
```

45b ⟨*Update version 5 derived parameters in* `/circom/` 45a⟩+≡

```
      if (acc .ne. TESLA) then
         call circem ('ERROR', 'versions 5 applies to TESLA only')
         acc = TESLA
      end if
⟨Map roots to e 31c⟩
      if (xa5lum(e,acc,r) .lt. 0d0) then
         write (msgbuf, 2002) roots, accnam(acc), r
         call circem ('ERROR', msgbuf)
         call circem ('MESSAGE', 'falling back to 500GeV')
         e = GEV500
      endif
⟨Log energy mapping 32c⟩
```

Uses `TESLA` 10c.

46a ⟨*Local variables for* `circes` 27a⟩+≡

```
      integer A5NEGY, A5NREV
      parameter (A5NEGY = 5, A5NREV = 1)
```

46b ⟨*Update version 5 derived parameters in* `/circom/` 45a⟩+≡

```
      lumi = xa5lum (e,acc,r)
      do 30 i = 0, 7
         a1(i) = xa5(i,e,acc,r)
   30 continue
```

46c ⟨*Local variables for* `circes` 27a⟩+≡

```
      real xa5lum(A5NEGY,NACC,0:A5NREV)
      real xa5(0:7,A5NEGY,NACC,0:A5NREV)
```

Uses `NACC` 10c.

Revision 1. The mother of all revisions. Note that $3.3980 \cdot 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1} = 2.4099\cdot 10^{34}\,\mathrm{m}^{-2}\cdot 2820\cdot 5\,\mathrm{s}^{-1}$ and $3.5936\cdot 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1} = 2.6619\cdot 10^{34}\,\mathrm{m}^{-2}\cdot 4500\cdot 3\,\mathrm{s}^{-1}$. This unit conversion is missing in *all* earlier versions, unfortunately.

46d ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa5lum(GEV350,TESLA, 1) /  -1.0 /
      data xa5lum(GEV500,TESLA, 1) /  0.33980E+03 /
      data (xa5(i,GEV500,TESLA, 1),i=0,7) /
     $   0.49808E+00,  0.54613E+00,  0.12287E+02, -0.62756E+00,
     $   0.42817E+00, -0.69120E+00,  0.17067E+02,  0.51143E+00 /
      data xa5lum(GEV800,TESLA, 1) /  0.35936E+03 /
      data (xa5(i,GEV800,TESLA, 1),i=0,7) /
     $   0.58751E+00,  0.43128E+00,  0.13324E+02, -0.64006E+00,
     $   0.30682E+00, -0.69235E+00,  0.16815E+02,  0.37078E+00 /
      data xa5lum(TEV1,  TESLA, 1) /  -1.0 /
      data xa5lum(TEV16, TESLA, 1) /  -1.0 /
```

Uses `TESLA` 10c.

Revision 0. Currently identical to revision 1.

46e ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa5lum(GEV350,TESLA, 0) /  -1.0 /
      data xa5lum(GEV500,TESLA, 0) /  0.33980E+03 /
      data (xa5(i,GEV500,TESLA, 0),i=0,7) /
     $   0.49808E+00,  0.54613E+00,  0.12287E+02, -0.62756E+00,
     $   0.42817E+00, -0.69120E+00,  0.17067E+02,  0.51143E+00 /
      data xa5lum(GEV800,TESLA, 0) /  0.35936E+03 /
      data (xa5(i,GEV800,TESLA, 0),i=0,7) /
     $   0.58751E+00,  0.43128E+00,  0.13324E+02, -0.64006E+00,
     $   0.30682E+00, -0.69235E+00,  0.16815E+02,  0.37078E+00 /
      data xa5lum(TEV1,  TESLA, 0) /  -1.0 /
      data xa5lum(TEV16, TESLA, 0) /  -1.0 /
```

Uses `TESLA` 10c.

6.2.5 Version 6

46f ⟨*Update version 6 derived parameters in* `/circom/` 46f⟩≡

```
      if (rev .eq. 0) then
         r = 0
⟨Warn that this revision has not been released yet 31a⟩
      elseif (rev .ge. 1999 04 15) then
         r = 1
      elseif (rev .lt. 1999 04 15) then
         call circem ('ERROR',
     $     'no revision of version 6 available before 1999/04/15')
         call circem ('MESSAGE', 'falling back to default')
         r = 1
      endif
⟨Log revision mapping 30e⟩
```

47a ⟨*Update version 6 derived parameters in* `/circom/` 46f⟩+≡

```
      if (acc .ne. TESLA) then
         call circem ('ERROR', 'versions 6 applies to TESLA only')
         acc = TESLA
      end if
⟨Map roots to e at low energies 47b⟩
      if (xa6lum(e,acc,r) .lt. 0d0) then
         write (msgbuf, 2002) roots, accnam(acc), r
         call circem ('ERROR', msgbuf)
         call circem ('MESSAGE', 'falling back to 500GeV')
         e = GEV500
      endif
⟨Log energy mapping 32c⟩
```

Uses `TESLA` 10c.

47b ⟨*Map* `roots` *to* `e` *at low energies* 47b⟩≡

```
      if (roots .eq.  90d0) then
         e = GEV090
      elseif ((roots .ge. 85d0) .and. (roots .le. 95d0)) then
         write (msgbuf, 2001) roots, 90d0
         call circem ('MESSAGE', msgbuf)
         e = GEV090
      elseif (roots .eq. 170d0) then
         e = GEV170
      elseif ((roots .ge. 160d0) .and. (roots .le. 180d0)) then
         write (msgbuf, 2001) roots, 170d0
         call circem ('MESSAGE', msgbuf)
         e = GEV170
      elseif (roots .eq. 350d0) then
         e = GEV350
      elseif ((roots .ge. 340d0) .and. (roots .le. 370d0)) then
         write (msgbuf, 2001) roots, 350d0
         call circem ('MESSAGE', msgbuf)
         e = GEV350
      elseif (roots .eq. 500d0) then
         e = GEV500
      elseif ((roots .ge. 480d0) .and. (roots .le. 520d0)) then
         write (msgbuf, 2001) roots, 500d0
         call circem ('MESSAGE', msgbuf)
         e = GEV500
      else
         call circem ('ERROR',
     $        'only ROOTS = 90, 170, 350, and 500GeV available')
         call circem ('MESSAGE', 'falling back to 500GeV')
         e = GEV500
      endif
```

48a ⟨*Local variables for* `circes` 27a⟩+≡

```
      integer A6NEGY, A6NREV
      parameter (A6NEGY = 2, A6NREV = 1)
```

48b ⟨*Update version 6 derived parameters in* `/circom/` 46f⟩+≡

```
      lumi = xa6lum (e,acc,r)
      do 40 i = 0, 7
         a1(i) = xa6(i,e,acc,r)
   40 continue
```

48c ⟨*Local variables for* `circes` 27a⟩+≡

```
      real xa6lum(GEV090:A6NEGY,NACC,0:A6NREV)
      real xa6(0:7,GEV090:A6NEGY,NACC,0:A6NREV)
```

Uses `NACC` 10c.

Revision 1. The mother of all revisions.

48d ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa6lum(GEV090,TESLA, 1) /  0.62408E+02 /
      data (xa6(i,GEV090,TESLA, 1),i=0,7) /
     $   0.72637E+00,  0.75534E+00,  0.18180E+03, -0.63426E+00,
     $   0.36829E+00, -0.69653E+00,  0.18908E+03,  0.22157E+00 /
      data xa6lum(GEV170,TESLA, 1) /  0.11532E+02 /
      data (xa6(i,GEV170,TESLA, 1),i=0,7) /
     $   0.65232E+00,  0.67249E+00,  0.66862E+02, -0.63315E+00,
     $   0.38470E+00, -0.69477E+00,  0.75120E+02,  0.30162E+00 /
      data xa6lum(GEV350,TESLA, 1) /  0.24641E+03 /
      data (xa6(i,GEV350,TESLA, 1),i=0,7) /
     $   0.54610E+00,  0.59105E+00,  0.20297E+02, -0.62747E+00,
     $   0.41588E+00, -0.69188E+00,  0.26345E+02,  0.43818E+00 /
      data xa6lum(GEV500,TESLA, 1) /  0.30340E+03 /
      data (xa6(i,GEV500,TESLA, 1),i=0,7) /
     $   0.52744E+00,  0.52573E+00,  0.13895E+02, -0.63145E+00,
     $   0.40824E+00, -0.69150E+00,  0.18645E+02,  0.47585E+00 /
```

Uses `TESLA` 10c.

Revision 0. Currently identical to revision 1.

48e ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa6lum(GEV090,TESLA, 0) /  0.62408E+02 /
      data (xa6(i,GEV090,TESLA, 0),i=0,7) /
     $   0.72637E+00,  0.75534E+00,  0.18180E+03, -0.63426E+00,
     $   0.36829E+00, -0.69653E+00,  0.18908E+03,  0.22157E+00 /
      data xa6lum(GEV170,TESLA, 0) /  0.11532E+02 /
      data (xa6(i,GEV170,TESLA, 0),i=0,7) /
     $   0.65232E+00,  0.67249E+00,  0.66862E+02, -0.63315E+00,
     $   0.38470E+00, -0.69477E+00,  0.75120E+02,  0.30162E+00 /
      data xa6lum(GEV350,TESLA, 0) /  0.24641E+03 /
      data (xa6(i,GEV350,TESLA, 0),i=0,7) /
     $   0.54610E+00,  0.59105E+00,  0.20297E+02, -0.62747E+00,
     $   0.41588E+00, -0.69188E+00,  0.26345E+02,  0.43818E+00 /
      data xa6lum(GEV500,TESLA, 0) /  0.30340E+03 /
      data (xa6(i,GEV500,TESLA, 0),i=0,7) /
     $   0.52744E+00,  0.52573E+00,  0.13895E+02, -0.63145E+00,
     $   0.40824E+00, -0.69150E+00,  0.18645E+02,  0.47585E+00 /
```
Uses `TESLA` 10c.

6.2.6 Version 7

49a ⟨*Update version 7 derived parameters in* `/circom/` 49a⟩≡
```
      if (rev .eq. 0) then
         r = 0
⟨Warn that this revision has not been released yet 31a⟩
      elseif (rev .ge. 2000 04 26) then
         r = 1
      elseif (rev .lt. 2000 04 26) then
         call circem ('ERROR',
     $     'no revision of version 7 available before 2000/04/26')
         call circem ('MESSAGE', 'falling back to default')
         r = 1
      endif
⟨Log revision mapping 30e⟩
```

49b ⟨*Update version 7 derived parameters in* `/circom/` 49a⟩+≡
```
      if (acc .ne. TESLA .and. acc .ne. JLCNLC) then
         call circem ('ERROR',
     $                'version 7 applies to TESLA and JLCNLC only')
         call circem ('ERROR', 'falling back to TESLA')
         acc = TESLA
      end if
⟨Linearly interpolate energies 50⟩
⟨Log energy mapping 32c⟩
```
Uses `TESLA` 10c.

49c ⟨`format`*s for* `circes` 31d⟩+≡
```
 2004 format ('energy ', F6.1, 'GeV too low, using spectrum for ',
     *                   F6.1, 'GeV')
 2005 format ('energy ', F6.1, 'GeV too high, using spectrum for ',
     *                   F6.1, 'GeV')
 2006 format ('energy ', F6.1, 'GeV interpolated between ',
     *                   F6.1, ' and ', F6.1, 'GeV')
```

49d ⟨*Local variables for* `circes` 27a⟩+≡
```
      double precision eloval, ehival
      double precision DELTAE
      parameter (DELTAE = 0.5d0)
```

The rules are as follows: `XBAND` has 500 GeV and 1 TeV, `TESLA` has 500 GeV and 800 TeV. Low energy `TESLA` will be added.

50 ⟨*Linearly interpolate energies* 50⟩≡

```
e = GEV090 - 1
elo = e
ehi = e
if (acc .eq. TESLA) then
   if (roots .lt.  90d0 - DELTAE) then
      write (msgbuf, 2004) roots, 90d0
      call circem ('MESSAGE', msgbuf)
      e = GEV090
   elseif (abs (roots-090d0) .le. DELTAE) then
      e = GEV090
   elseif (roots .lt. 170d0 - DELTAE) then
      write (msgbuf, 2005) roots, 170d0
      call circem ('MESSAGE', msgbuf)
      e = GEV170
   elseif (abs (roots-170d0) .le. DELTAE) then
      e = GEV170
   elseif (roots .lt. 350d0-DELTAE) then
      write (msgbuf, 2006) roots, 170d0, 350d0
      call circem ('MESSAGE', msgbuf)
      elo = GEV170
      ehi = GEV350
      eloval = 170d0
      ehival = 350d0
   elseif (abs (roots-350d0) .le. DELTAE) then
      e = GEV350
   elseif (roots .lt. 500d0 - DELTAE) then
      write (msgbuf, 2006) roots, 350d0, 500d0
      call circem ('MESSAGE', msgbuf)
      elo = GEV350
      ehi = GEV500
      eloval = 350d0
      ehival = 500d0
   elseif (abs (roots-500d0) .le. DELTAE) then
      e = GEV500
   elseif (roots .lt. 800d0 - DELTAE) then
      write (msgbuf, 2006) roots, 500d0, 800d0
      call circem ('MESSAGE', msgbuf)
      elo = GEV500
      ehi = GEV800
      eloval = 500d0
      ehival = 800d0
   elseif (abs (roots-800d0) .le. DELTAE) then
      e = GEV800
   else
      write (msgbuf, 2005) roots, 800d0
      call circem ('MESSAGE', msgbuf)
```

```
            e = GEV800
         endif
      elseif (acc .eq. XBAND) then
         if (roots .lt.  500d0 - DELTAE) then
            write (msgbuf, 2004) roots, 500d0
            call circem ('MESSAGE', msgbuf)
            e = GEV500
         elseif (abs (roots-500d0) .le. DELTAE) then
            e = GEV500
         elseif (roots .lt. 1000d0 - DELTAE) then
            write (msgbuf, 2006) roots, 500d0, 1000d0
            call circem ('MESSAGE', msgbuf)
            elo = GEV500
            ehi = TEV1
            eloval =  500d0
            ehival = 1000d0
         elseif (abs (roots-1000d0) .le. DELTAE) then
            e = TEV1
         else
            write (msgbuf, 2005) roots, 1000d0
            call circem ('MESSAGE', msgbuf)
            e = TEV1
         endif
      endif
```

Uses `TESLA` 10c and `XBAND` 10c.

51a ⟨*Local variables for* `circes` 27a⟩+≡

```
      integer A7NEGY, A7NREV
      parameter (A7NEGY = TEV1, A7NREV = 1)
```

Note that ew *must not* interpolate `a1(0)` and `a1(7)` because they depend non-linearly on the other parameters!

51b ⟨*Update version 7 derived parameters in* `/circom/` 49a⟩+≡

```
      if (e .ge. GEV090) then
         lumi = xa7lum(e,acc,r)
         do 50 i = 0, 7
            a1(i) = xa7(i,e,acc,r)
   50    continue
      elseif (elo .ge. GEV090 .and. ehi .ge. GEV090) then
         lumi = ((roots-eloval)*xa7lum(ehi,acc,r)
     $        + (ehival-roots)*xa7lum(elo,acc,r)) / (ehival - eloval)
         do 51 i = 1, 6
            a1(i) = ((roots-eloval)*xa7(i,ehi,acc,r)
     $           + (ehival-roots)*xa7(i,elo,acc,r)) / (ehival - eloval)
   51    continue
         a1(0) = 1d0 - a1(1) * beta(a1(2)+1d0,a1(3)+1d0)
         a1(7) = a1(4) * beta(a1(5)+1d0,a1(6)+1d0)
      endif
```

Uses `beta` 90a.

51c ⟨*Local variables for* `circes` 27a⟩+≡

```
      real xa7lum(GEV090:A7NEGY,NACC,0:A7NREV)
      real xa7(0:7,GEV090:A7NEGY,NACC,0:A7NREV)
```

Uses `NACC` 10c.

Revision 1. The mother of all revisions.

52a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa7lum(GEV090,TESLA,1) /  0.62408E+02 /
      data (xa7(i,GEV090,TESLA,1),i=0,7) /
     $   0.72637E+00,  0.75534E+00,  0.18180E+03, -0.63426E+00,
     $   0.36829E+00, -0.69653E+00,  0.18908E+03,  0.22157E+00 /
      data xa7lum(GEV170,TESLA,1) /  0.11532E+02 /
      data (xa7(i,GEV170,TESLA,1),i=0,7) /
     $   0.65232E+00,  0.67249E+00,  0.66862E+02, -0.63315E+00,
     $   0.38470E+00, -0.69477E+00,  0.75120E+02,  0.30162E+00 /
      data xa7lum(GEV350,TESLA,1) /  0.24641E+03 /
      data (xa7(i,GEV350,TESLA,1),i=0,7) /
     $   0.54610E+00,  0.59105E+00,  0.20297E+02, -0.62747E+00,
     $   0.41588E+00, -0.69188E+00,  0.26345E+02,  0.43818E+00 /
      data xa7lum(GEV500,TESLA,1) /  0.34704E+03 /
      data (xa7(i,GEV500,TESLA,1),i=0,7) /
     $   0.51288E+00,  0.49025E+00,  0.99716E+01, -0.62850E+00,
     $   0.41048E+00, -0.69065E+00,  0.13922E+02,  0.51902E+00 /
      data xa7lum(GEV800,TESLA,1) /  0.57719E+03 /
      data (xa7(i,GEV800,TESLA,1),i=0,7) /
     $   0.52490E+00,  0.42573E+00,  0.69069E+01, -0.62649E+00,
     $   0.32380E+00, -0.68958E+00,  0.93819E+01,  0.45671E+00 /
      data xa7lum(TEV1,  TESLA,1) /  -1.0 /
```

Uses `TESLA` 10c.

52b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa7lum(GEV090,JLCNLC,1) /  -1.0 /
      data xa7lum(GEV170,JLCNLC,1) /  -1.0 /
      data xa7lum(GEV350,JLCNLC,1) /  -1.0 /
      data xa7lum(GEV500,JLCNLC,1) /  0.63039E+02 /
      data (xa7(i,GEV500,JLCNLC,1),i=0,7) /
     $   0.58967E+00,  0.34035E+00,  0.63631E+01, -0.63683E+00,
     $   0.33383E+00, -0.68803E+00,  0.81005E+01,  0.48702E+00 /
      data xa7lum(TEV1,JLCNLC,1) /  0.12812E+03 /
      data (xa7(i,TEV1,JLCNLC,1),i=0,7) /
     $   0.50222E+00,  0.33773E+00,  0.25681E+01, -0.61711E+00,
     $   0.36826E+00, -0.68335E+00,  0.36746E+01,  0.65393E+00 /
```

Revision 0.

52c ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa7lum(GEV090,TESLA,0) /  0.62408E+02 /
      data (xa7(i,GEV090,TESLA,0),i=0,7) /
     $   0.72637E+00,  0.75534E+00,  0.18180E+03, -0.63426E+00,
     $   0.36829E+00, -0.69653E+00,  0.18908E+03,  0.22157E+00 /
      data xa7lum(GEV170,TESLA,0) /  0.11532E+02 /
      data (xa7(i,GEV170,TESLA,0),i=0,7) /
     $   0.65232E+00,  0.67249E+00,  0.66862E+02, -0.63315E+00,
     $   0.38470E+00, -0.69477E+00,  0.75120E+02,  0.30162E+00 /
      data xa7lum(GEV350,TESLA,0) /  0.24641E+03 /
      data (xa7(i,GEV350,TESLA,0),i=0,7) /
     $   0.54610E+00,  0.59105E+00,  0.20297E+02, -0.62747E+00,
     $   0.41588E+00, -0.69188E+00,  0.26345E+02,  0.43818E+00 /
      data xa7lum(GEV500,TESLA,0) /  0.34704E+03 /
      data (xa7(i,GEV500,TESLA,0),i=0,7) /
     $   0.51288E+00,  0.49025E+00,  0.99716E+01, -0.62850E+00,
     $   0.41048E+00, -0.69065E+00,  0.13922E+02,  0.51902E+00 /
      data xa7lum(GEV800,TESLA,0) /  0.57719E+03 /
      data (xa7(i,GEV800,TESLA,0),i=0,7) /
     $   0.52490E+00,  0.42573E+00,  0.69069E+01, -0.62649E+00,
     $   0.32380E+00, -0.68958E+00,  0.93819E+01,  0.45671E+00 /
      data xa7lum(TEV1,  TESLA,0) /  -1.0 /
```

Uses `TESLA` 10c.

53a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa7lum(GEV090,JLCNLC,0) /  -1.0 /
      data xa7lum(GEV170,JLCNLC,0) /  -1.0 /
      data xa7lum(GEV350,JLCNLC,0) /  -1.0 /
      data xa7lum(GEV500,JLCNLC,0) /  0.63039E+02 /
      data (xa7(i,GEV500,JLCNLC,0),i=0,7) /
     $   0.58967E+00,  0.34035E+00,  0.63631E+01, -0.63683E+00,
     $   0.33383E+00, -0.68803E+00,  0.81005E+01,  0.48702E+00 /
      data xa7lum(TEV1,JLCNLC,0) /  0.12812E+03 /
      data (xa7(i,TEV1,JLCNLC,0),i=0,7) /
     $   0.50222E+00,  0.33773E+00,  0.25681E+01, -0.61711E+00,
     $   0.36826E+00, -0.68335E+00,  0.36746E+01,  0.65393E+00 /
```

6.2.7 Version 8

53b ⟨*Update version 8 derived parameters in* `/circom/` 53b⟩≡

```
      if (rev .eq. 0) then
         r = 0
  ⟨Warn that this revision has not been released yet 31a⟩
      elseif (rev .ge. 2001 06 17) then
         r = 1
      elseif (rev .lt. 2001 06 17) then
         call circem (’ERROR’,
     $     ’no revision of version 8 available before 2001/06/17’)
         call circem (’MESSAGE’, ’falling back to default’)
         r = 1
      endif
  ⟨Log revision mapping 30e⟩
```

53c ⟨*Update version 8 derived parameters in* `/circom/` 53b⟩+≡

```
      if (acc .eq. NLCH) then
         acc = JLCNLC
      end if
      if (acc .ne. JLCNLC) then
         call circem (’ERROR’,
       $               'version 8 applies to JLCNLC (NLC H) only')
           call circem ('ERROR', 'falling back to JLCNLC')
           acc = JLCNLC
        end if
⟨Linearly interpolate energies 50⟩
⟨Log energy mapping 32c⟩
```

54a ⟨*Local variables for* `circes` 27a⟩+≡

```
        integer A8NEGY, A8NREV
        parameter (A8NEGY = TEV1, A8NREV = 1)
```

Note that ew *must not* interpolate `a1(0)` and `a1(7)` because they depend non-linearly on the other parameters!

54b ⟨*Update version 8 derived parameters in* `/circom/` 53b⟩+≡

```
        if (e .ge. GEV090) then
           lumi = xa8lum(e,acc,r)
           do 60 i = 0, 7
              a1(i) = xa8(i,e,acc,r)
     60    continue
        elseif (elo .ge. GEV090 .and. ehi .ge. GEV090) then
           lumi = ((roots-eloval)*xa8lum(ehi,acc,r)
       $        + (ehival-roots)*xa8lum(elo,acc,r)) / (ehival - eloval)
           do 61 i = 1, 6
              a1(i) = ((roots-eloval)*xa8(i,ehi,acc,r)
       $           + (ehival-roots)*xa8(i,elo,acc,r)) / (ehival - eloval)
     61    continue
           a1(0) = 1d0 - a1(1) * beta(a1(2)+1d0,a1(3)+1d0)
           a1(7) = a1(4) * beta(a1(5)+1d0,a1(6)+1d0)
        endif
```

Uses `beta` 90a.

54c ⟨*Local variables for* `circes` 27a⟩+≡

```
        real xa8lum(GEV090:A8NEGY,NACC,0:A8NREV)
        real xa8(0:7,GEV090:A8NEGY,NACC,0:A8NREV)
```

Uses `NACC` 10c.

Revision 1. The mother of all revisions.

54d ⟨*Initializations for* `circes` 28c⟩+≡

```
        data xa8lum(GEV090,TESLA,1) / -1.0 /
        data xa8lum(GEV170,TESLA,1) / -1.0 /
        data xa8lum(GEV350,TESLA,1) / -1.0 /
        data xa8lum(GEV500,TESLA,1) / -1.0 /
        data xa8lum(GEV800,TESLA,1) / -1.0 /
        data xa8lum(TEV1,  TESLA,1) / -1.0 /
```

Uses `TESLA` 10c.

54e ⟨*Initializations for* `circes` 28c⟩+≡

```
        data xa8lum(GEV090,JLCNLC,1) /  -1.0 /
        data xa8lum(GEV170,JLCNLC,1) /  -1.0 /
        data xa8lum(GEV350,JLCNLC,1) /  -1.0 /
        data xa8lum(GEV500,JLCNLC,1) /  0.239924E+03 /
        data (xa8(i,GEV500,JLCNLC,1),i=0,7) /
```

```
     $   0.57025E+00,  0.34004E+00,  0.52864E+01, -0.63405E+00,
     $   0.31627E+00, -0.68722E+00,  0.69629E+01,  0.47973E+00 /
      data xa8lum(TEV1,JLCNLC,1) /  0.40858E+03 /
      data (xa8(i,TEV1,JLCNLC,1),i=0,7) /
     $   0.52344E+00,  0.31536E+00,  0.25244E+01, -0.62215E+00,
     $   0.31935E+00, -0.68424E+00,  0.35877E+01,  0.57315E+00 /
```

Revision 0.

55a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa8lum(GEV090,TESLA,0) / -1.0 /
      data xa8lum(GEV170,TESLA,0) / -1.0 /
      data xa8lum(GEV350,TESLA,0) / -1.0 /
      data xa8lum(GEV500,TESLA,0) / -1.0 /
      data xa8lum(GEV800,TESLA,0) / -1.0 /
      data xa8lum(TEV1,  TESLA,0) / -1.0 /
```

Uses `TESLA` 10c.

55b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa8lum(GEV090,JLCNLC,0) /  -1.0 /
      data xa8lum(GEV170,JLCNLC,0) /  -1.0 /
      data xa8lum(GEV350,JLCNLC,0) /  -1.0 /
      data xa8lum(GEV500,JLCNLC,0) /  0.239924E+03 /
      data (xa8(i,GEV500,JLCNLC,0),i=0,7) /
     $   0.57025E+00,  0.34004E+00,  0.52864E+01, -0.63405E+00,
     $   0.31627E+00, -0.68722E+00,  0.69629E+01,  0.47973E+00 /
      data xa8lum(TEV1,JLCNLC,0) /  0.40858E+03 /
      data (xa8(i,TEV1,JLCNLC,0),i=0,7) /
     $   0.52344E+00,  0.31536E+00,  0.25244E+01, -0.62215E+00,
     $   0.31935E+00, -0.68424E+00,  0.35877E+01,  0.57315E+00 /
```

6.2.8 Version 9

55c ⟨*Update version 9 derived parameters in* `/circom/` 55c⟩≡

```
      if (rev .eq. 0) then
         r = 0
         ⟨Warn that this revision has not been released yet 31a⟩
      elseif (rev .ge. 2002 03 28) then
         r = 1
      elseif (rev .lt. 2002 03 28) then
         call circem ('ERROR',
     $     'no revision of version 9 available before 2002/03/28')
         call circem ('MESSAGE', 'falling back to default')
         r = 1
      endif
      ⟨Log revision mapping 30e⟩
```

55d ⟨*Update version 9 derived parameters in* `/circom/` 55c⟩+≡

```
      if (acc .ne. JLCNLC .and. acc .ne. NLCH) then
         call circem ('ERROR',
     $                'version 9 applies to JLCNLC and NLCH only')
         call circem ('ERROR', 'falling back to JLCNLC')
```

```
   acc = JLCNLC
end if
if (acc .eq. JLCNLC) then
   ⟨Linearly interpolate energies for JLC/NLC 2002 56⟩
else if (acc .eq. NLCH) then
   ⟨Linearly interpolate energies for NLC H 2002 57⟩
end if
⟨Log energy mapping 32c⟩
```

56 ⟨*Linearly interpolate energies for JLC/NLC 2002* 56⟩≡

```
e = GEV090 - 1
elo = e
ehi = e
if (roots .lt. 250d0 - DELTAE) then
   write (msgbuf, 2004) roots, 250d0
   call circem ('MESSAGE', msgbuf)
   e = GEV250
elseif (abs (roots-250d0) .le. DELTAE) then
   e = GEV250
elseif (roots .lt. 500d0 - DELTAE) then
   write (msgbuf, 2006) roots, 250d0, 500d0
   call circem ('MESSAGE', msgbuf)
   elo = GEV250
   ehi = GEV500
   eloval = 250d0
   ehival = 500d0
elseif (abs (roots-500d0) .le. DELTAE) then
   e = GEV500
elseif (roots .lt. 800d0 - DELTAE) then
   write (msgbuf, 2006) roots, 500d0, 800d0
   call circem ('MESSAGE', msgbuf)
   elo = GEV500
   ehi = GEV800
   eloval = 500d0
   ehival = 800d0
elseif (abs (roots-800d0) .le. DELTAE) then
   e = GEV800
elseif (roots .lt. 1000d0 - DELTAE) then
   write (msgbuf, 2006) roots, 800d0, 1000d0
   call circem ('MESSAGE', msgbuf)
   elo = GEV800
   ehi = TEV1
   eloval =  800d0
   ehival = 1000d0
elseif (abs (roots-1000d0) .le. DELTAE) then
   e = TEV1
elseif (roots .lt. 1200d0 - DELTAE) then
   write (msgbuf, 2006) roots, 1000d0, 1200d0
   call circem ('MESSAGE', msgbuf)
   elo = TEV1
   ehi = TEV12
   eloval = 1000d0
   ehival = 1200d0
elseif (abs (roots-1200d0) .le. DELTAE) then
   e = TEV12
elseif (roots .lt. 1500d0 - DELTAE) then
   write (msgbuf, 2006) roots, 1200d0, 1500d0
   call circem ('MESSAGE', msgbuf)
   elo = TEV12
   ehi = TEV15
   eloval = 1200d0
   ehival = 1500d0
elseif (abs (roots-1500d0) .le. DELTAE) then
   e = TEV15
else
   write (msgbuf, 2005) roots, 1500d0
   call circem ('MESSAGE', msgbuf)
   e = TEV15
endif
```

57 ⟨*Linearly interpolate energies for NLC H 2002* 57⟩≡

```
e = GEV090 - 1
elo = e
ehi = e
if (roots .lt. 500d0 - DELTAE) then
   write (msgbuf, 2004) roots, 500d0
   call circem ('MESSAGE', msgbuf)
   e = GEV500
elseif (abs (roots-500d0) .le. DELTAE) then
   e = GEV500
elseif (roots .lt. 1000d0 - DELTAE) then
   write (msgbuf, 2006) roots, 500d0, 1000d0
   call circem ('MESSAGE', msgbuf)
   elo = GEV500
   ehi = TEV1
   eloval =  500d0
   ehival = 1000d0
elseif (abs (roots-1000d0) .le. DELTAE) then
   e = TEV1
elseif (roots .lt. 1500d0 - DELTAE) then
   write (msgbuf, 2006) roots, 1000d0, 1500d0
   call circem ('MESSAGE', msgbuf)
   elo = TEV1
   ehi = TEV15
   eloval = 1000d0
   ehival = 1500d0
elseif (abs (roots-1500d0) .le. DELTAE) then
   e = TEV15
else
   write (msgbuf, 2005) roots, 1500d0
```

```
         call circem ('MESSAGE', msgbuf)
         e = TEV15
      endif
```

58a ⟨*Local variables for* `circes` 27a⟩+≡

```
      integer A9NEGY, A9NREV
      parameter (A9NEGY = TEV15, A9NREV = 1)
```

Note that ew *must not* interpolate `a1(0)` and `a1(7)` because they depend non-linearly on the other parameters!

58b ⟨*Update version 9 derived parameters in* `/circom/` 55c⟩+≡

```
      if (e .ge. GEV090) then
         lumi = xa9lum(e,acc,r)
         do 70 i = 0, 7
            a1(i) = xa9(i,e,acc,r)
   70    continue
      elseif (elo .ge. GEV090 .and. ehi .ge. GEV090) then
         lumi = ((roots-eloval)*xa9lum(ehi,acc,r)
     $        + (ehival-roots)*xa9lum(elo,acc,r)) / (ehival - eloval)
         do 71 i = 1, 6
            a1(i) = ((roots-eloval)*xa9(i,ehi,acc,r)
     $           + (ehival-roots)*xa9(i,elo,acc,r)) / (ehival - eloval)
   71    continue
         a1(0) = 1d0 - a1(1) * beta(a1(2)+1d0,a1(3)+1d0)
         a1(7) = a1(4) * beta(a1(5)+1d0,a1(6)+1d0)
      endif
```

Uses `beta` 90a.

58c ⟨*Local variables for* `circes` 27a⟩+≡

```
      real xa9lum(GEV090:A9NEGY,NACC,0:A9NREV)
      real xa9(0:7,GEV090:A9NEGY,NACC,0:A9NREV)
```

Uses `NACC` 10c.

Revision 1. The mother of all revisions.

58d ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa9lum(GEV090,TESLA,1) / -1.0 /
      data xa9lum(GEV170,TESLA,1) / -1.0 /
      data xa9lum(GEV350,TESLA,1) / -1.0 /
      data xa9lum(GEV500,TESLA,1) / -1.0 /
      data xa9lum(GEV800,TESLA,1) / -1.0 /
      data xa9lum(TEV1,  TESLA,1) / -1.0 /
      data xa9lum(TEV12, TESLA,1) / -1.0 /
      data xa9lum(TEV15, TESLA,1) / -1.0 /
      data xa9lum(TEV16, TESLA,1) / -1.0 /
```

Uses `TESLA` 10c.

58e ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa9lum(GEV090,JLCNLC, 1) / -1.0 /
      data xa9lum(GEV170,JLCNLC, 1) / -1.0 /
      data xa9lum(GEV250,JLCNLC, 1) / 109.886976 /
      data (xa9(i,GEV250,JLCNLC, 1),i=0,7) /
     $   0.65598E+00,  0.34993E+00,  0.13766E+02, -0.64698E+00,
     $   0.29984E+00, -0.69053E+00,  0.16444E+02,  0.36060E+00 /
      data xa9lum(GEV350,JLCNLC, 1) / -1.0 /
      data xa9lum(GEV500,JLCNLC, 1) / 220.806144 /
      data (xa9(i,GEV500,JLCNLC, 1),i=0,7) /
     $   0.57022E+00,  0.33782E+00,  0.52811E+01, -0.63540E+00,
     $   0.32035E+00, -0.68776E+00,  0.69552E+01,  0.48751E+00 /
      data xa9lum(GEV800,JLCNLC, 1) / 304.63488 /
      data (xa9(i,GEV800,JLCNLC, 1),i=0,7) /
     $   0.54839E+00,  0.31823E+00,  0.33071E+01, -0.62671E+00,
     $   0.31655E+00, -0.68468E+00,  0.45325E+01,  0.53449E+00 /
      data xa9lum(TEV1,  JLCNLC, 1) / 319.95648 /
      data (xa9(i,TEV1,  JLCNLC, 1),i=0,7) /
     $   0.56047E+00,  0.29479E+00,  0.28820E+01, -0.62856E+00,
     $   0.29827E+00, -0.68423E+00,  0.39138E+01,  0.52297E+00 /
      data xa9lum(TEV12, JLCNLC, 1) / 349.90848 /
      data (xa9(i,TEV12, JLCNLC, 1),i=0,7) /
     $   0.56102E+00,  0.28503E+00,  0.24804E+01, -0.62563E+00,
     $   0.29002E+00, -0.68376E+00,  0.33854E+01,  0.52736E+00 /
      data xa9lum(TEV15, JLCNLC, 1) / 363.15648 /
      data (xa9(i,TEV15, JLCNLC, 1),i=0,7) /
     $   0.57644E+00,  0.26570E+00,  0.22007E+01, -0.62566E+00,
     $   0.27102E+00, -0.68283E+00,  0.29719E+01,  0.50764E+00 /
      data xa9lum(TEV16, JLCNLC, 1) / -1.0 /
```

59a ⟨*Initializations for* `circes` 28c⟩+≡

```
 data xa9lum(GEV090,NLCH,   1) / -1.0 /
 data xa9lum(GEV170,NLCH,   1) / -1.0 /
 data xa9lum(GEV250,NLCH,   1) / -1.0 /
 data xa9lum(GEV350,NLCH,   1) / -1.0 /
 data xa9lum(GEV500,NLCH,   1) / 371.4624 /
 data (xa9(i,GEV500,NLCH,   1),i=0,7) /
$   0.33933E+00,  0.55165E+00,  0.29138E+01, -0.57341E+00,
$   0.54323E+00, -0.68590E+00,  0.51786E+01,  0.88956E+00 /
 data xa9lum(GEV800,NLCH,   1) / -1.0 /
 data xa9lum(TEV1,  NLCH,   1) / 516.41856 /
 data (xa9(i,TEV1,  NLCH,   1),i=0,7) /
$   0.35478E+00,  0.46474E+00,  0.17666E+01, -0.56949E+00,
$   0.49269E+00, -0.68384E+00,  0.31781E+01,  0.91121E+00 /
 data xa9lum(TEV12, NLCH,   1) / -1.0 /
 data xa9lum(TEV15, NLCH,   1) / 575.06688 /
 data (xa9(i,TEV15, NLCH,   1),i=0,7) /
$   0.38183E+00,  0.40310E+00,  0.13704E+01, -0.57742E+00,
$   0.44548E+00, -0.68341E+00,  0.24956E+01,  0.87448E+00 /
 data xa9lum(TEV16, NLCH,   1) / -1.0 /
```

Revision 0.

59b ⟨*Initializations for* `circes` 28c⟩+≡

```
 data xa9lum(GEV090,TESLA,0) / -1.0 /
 data xa9lum(GEV170,TESLA,0) / -1.0 /
 data xa9lum(GEV350,TESLA,0) / -1.0 /
 data xa9lum(GEV500,TESLA,0) / -1.0 /
```

```
      data xa9lum(GEV800,TESLA,0) / -1.0 /
      data xa9lum(TEV1,  TESLA,0) / -1.0 /
      data xa9lum(TEV12, TESLA,0) / -1.0 /
      data xa9lum(TEV15, TESLA,0) / -1.0 /
      data xa9lum(TEV16, TESLA,0) / -1.0 /
```

Uses `TESLA` 10c.

60a ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa9lum(GEV090,JLCNLC, 0) / -1.0 /
      data xa9lum(GEV170,JLCNLC, 0) / -1.0 /
      data xa9lum(GEV250,JLCNLC, 0) / 109.886976 /
      data (xa9(i,GEV250,JLCNLC, 0),i=0,7) /
     $   0.65598E+00,  0.34993E+00,  0.13766E+02, -0.64698E+00,
     $   0.29984E+00, -0.69053E+00,  0.16444E+02,  0.36060E+00 /
      data xa9lum(GEV350,JLCNLC, 0) / -1.0 /
      data xa9lum(GEV500,JLCNLC, 0) / 220.806144 /
      data (xa9(i,GEV500,JLCNLC, 0),i=0,7) /
     $   0.57022E+00,  0.33782E+00,  0.52811E+01, -0.63540E+00,
     $   0.32035E+00, -0.68776E+00,  0.69552E+01,  0.48751E+00 /
      data xa9lum(GEV800,JLCNLC, 0) / 304.63488 /
      data (xa9(i,GEV800,JLCNLC, 0),i=0,7) /
     $   0.54839E+00,  0.31823E+00,  0.33071E+01, -0.62671E+00,
     $   0.31655E+00, -0.68468E+00,  0.45325E+01,  0.53449E+00 /
      data xa9lum(TEV1,  JLCNLC, 0) / 319.95648 /
      data (xa9(i,TEV1,  JLCNLC, 0),i=0,7) /
     $   0.56047E+00,  0.29479E+00,  0.28820E+01, -0.62856E+00,
     $   0.29827E+00, -0.68423E+00,  0.39138E+01,  0.52297E+00 /
      data xa9lum(TEV12, JLCNLC, 0) / 349.90848 /
      data (xa9(i,TEV12, JLCNLC, 0),i=0,7) /
     $   0.56102E+00,  0.28503E+00,  0.24804E+01, -0.62563E+00,
     $   0.29002E+00, -0.68376E+00,  0.33854E+01,  0.52736E+00 /
      data xa9lum(TEV15, JLCNLC, 0) / 363.15648 /
      data (xa9(i,TEV15, JLCNLC, 0),i=0,7) /
     $   0.57644E+00,  0.26570E+00,  0.22007E+01, -0.62566E+00,
     $   0.27102E+00, -0.68283E+00,  0.29719E+01,  0.50764E+00 /
      data xa9lum(TEV16, JLCNLC, 0) / -1.0 /
```

60b ⟨*Initializations for* `circes` 28c⟩+≡

```
      data xa9lum(GEV090,NLCH,   0) / -1.0 /
      data xa9lum(GEV170,NLCH,   0) / -1.0 /
      data xa9lum(GEV250,NLCH,   0) / -1.0 /
      data xa9lum(GEV350,NLCH,   0) / -1.0 /
      data xa9lum(GEV500,NLCH,   0) / 371.4624 /
      data (xa9(i,GEV500,NLCH,   0),i=0,7) /
     $   0.33933E+00,  0.55165E+00,  0.29138E+01, -0.57341E+00,
     $   0.54323E+00, -0.68590E+00,  0.51786E+01,  0.88956E+00 /
      data xa9lum(GEV800,NLCH,   0) / -1.0 /
      data xa9lum(TEV1,  NLCH,   0) / 516.41856 /
      data (xa9(i,TEV1,  NLCH,   0),i=0,7) /
     $   0.35478E+00,  0.46474E+00,  0.17666E+01, -0.56949E+00,
     $   0.49269E+00, -0.68384E+00,  0.31781E+01,  0.91121E+00 /
```

```
      data xa9lum(TEV12, NLCH,   0) / -1.0 /
      data xa9lum(TEV15, NLCH,   0) / 575.06688 /
      data (xa9(i,TEV15, NLCH,   0),i=0,7) /
     $   0.38183E+00,  0.40310E+00,  0.13704E+01, -0.57742E+00,
     $   0.44548E+00, -0.68341E+00,  0.24956E+01,  0.87448E+00 /
      data xa9lum(TEV16, NLCH,   0) / -1.0 /
```

6.3 Special Functions

61a ⟨*Subroutines* 24d⟩+≡

```
      double precision function beta (a, b)
      implicit none
      double precision a, b
      double precision dlogam
      beta = exp (dlogam(a) + dlogam(b) - dlogam(a+b))
      end
```

Uses `beta` 90a.

61b ⟨*Subroutines* 24d⟩+≡

```
CERNLIB C304
      DOUBLE PRECISION FUNCTION DLOGAM(X)
      IMPLICIT NONE
      DOUBLE PRECISION P1(7),Q1(7),P2(7),Q2(7),P3(7),Q3(7),C(5),XL(5)
      DOUBLE PRECISION X,Y,ZERO,ONE,TWO,HALF,AP,AQ
      INTEGER I
      DATA ZERO /0.0D0/, ONE /1.0D0/, TWO /2.0D0/, HALF /0.5D0/
      DATA XL /0.0D0,0.5D0,1.5D0,4.0D0,12.0D0/
      DATA P1
     1/+3.84287 36567 460D+0, +5.27068 93753 010D+1,
     2 +5.55840 45723 515D+1, -2.15135 13573 726D+2,
     3 -2.45872 61722 292D+2, -5.75008 93603 041D+1,
     4 -2.33590 98949 513D+0/
      DATA Q1
     1/+1.00000 00000 000D+0, +3.37330 47907 071D+1,
     2 +1.93877 84034 377D+2, +3.08829 54973 424D+2,
     3 +1.50068 39064 891D+2, +2.01068 51344 334D+1,
     4 +4.57174 20282 503D-1/
      DATA P2
     1/+4.87402 01396 839D+0, +2.48845 25168 574D+2,
     2 +2.17973 66058 896D+3, +3.79751 24011 525D+3,
     3 -1.97780 70769 842D+3, -3.69298 34005 591D+3,
     4 -5.60177 73537 804D+2/
      DATA Q2
     1/+1.00000 00000 000D+0, +9.50999 17418 209D+1,
     2 +1.56120 45277 929D+3, +7.23400 87928 948D+3,
     3 +1.04595 76594 059D+4, +4.16994 15153 200D+3,
     4 +2.76785 83623 804D+2/
      DATA P3
     1/-6.88062 40094 594D+3, -4.30699 69819 571D+5,
     2 -4.75045 94653 440D+6, -2.94234 45930 322D+6,
     3 +3.63218 04931 543D+7, -3.35677 82814 546D+6,
     4 -2.48043 69488 286D+7/
      DATA Q3
     1/+1.00000 00000 000D+0, -1.42168 29839 651D+3,
     2 -1.55528 90280 854D+5, -3.41525 17108 011D+6,
     3 -2.09696 23255 804D+7, -3.45441 75093 344D+7,
     4 -9.16055 82863 713D+6/
      DATA C
     1/ 1.12249 21356 561D-1,  7.95916 92961 204D-2,
     1 -1.70877 94611 020D-3,  9.18938 53320 467D-1,
     2  1.34699 05627 879D+0/
      IF(X .LE. XL(1)) THEN
       print *, 'ERROR: DLOGAM non positive argument: ', X
       DLOGAM=ZERO
      ENDIF
      IF(X .LE. XL(2)) THEN
       Y=X+ONE
       AP=P1(1)
       AQ=Q1(1)
       DO 2 I = 2,7
          AP=P1(I)+Y*AP
    2     AQ=Q1(I)+Y*AQ
       Y=-LOG(X)+X*AP/AQ
      ELSEIF(X .LE. XL(3)) THEN
       AP=P1(1)
       AQ=Q1(1)
       DO 3 I = 2,7
          AP=P1(I)+X*AP
    3     AQ=Q1(I)+X*AQ
       Y=(X-ONE)*AP/AQ
      ELSEIF(X .LE. XL(4)) THEN
       AP=P2(1)
       AQ=Q2(1)
       DO 4 I = 2,7
          AP=P2(I)+X*AP
    4     AQ=Q2(I)+X*AQ
       Y=(X-TWO)*AP/AQ
      ELSEIF(X .LE. XL(5)) THEN
       AP=P3(1)
       AQ=Q3(1)
       DO 5 I = 2,7
          AP=P3(I)+X*AP
    5     AQ=Q3(I)+X*AQ
       Y=AP/AQ
      ELSE
       Y=ONE/X**2
       Y=(X-HALF)*LOG(X)-X+C(4)+(C(1)+Y*(C(2)+Y*C(3)))/
     1                                        ((C(5)+Y)*X)
      ENDIF
      DLOGAM=Y
```

```
        END
```

6.4 Non-Singular Distributions

63a ⟨*Subroutines* 24d⟩+≡

```
        double precision function kirke (x1, x2, p1, p2)
        implicit none
        double precision x1, x2
        integer p1, p2
        double precision kirkee, kirkeg, kirkgg
        ⟨Particle codes 9b⟩
        ⟨/circom/ 25b⟩
        ⟨Initialization check 26e⟩
        kirke = -1.0
        if (abs(p1) .eq. ELECTR) then
           if (abs(p2) .eq. ELECTR) then
              kirke = kirkee (x1, x2)
           elseif (p2 .eq. PHOTON) then
              kirke = kirkeg (x1, x2)
           endif
        elseif (p1 .eq. PHOTON) then
           if (abs(p2) .eq. ELECTR) then
              kirke = kirkeg (x2, x1)
           elseif (p2 .eq. PHOTON) then
              kirke = kirkgg (x1, x2)
           endif
        endif
        end
```

Defines:
  `kirke`, never used.
Uses `ELECTR` 9b, `PHOTON` 9b, `kirkee` 63b, `kirkeg` 65c, and `kirkgg` 66a.

63b ⟨*Subroutines* 24d⟩+≡

```
        double precision function kirkee (x1, x2)
        implicit none
        double precision x1, x2
        ⟨/circom/ 25b⟩
        double precision d1, d2
        ⟨Initialization check 26e⟩
        kirkee = -1.0
        if ((ver .eq. 1) .or. (ver .eq. 0)) then
           ⟨Calculate version 1 of the non-singular e+e- distribution 64e⟩
        ⟨else handle invalid versions 30b⟩
        end
```

Defines:
  `kirkee`, used in chunks 13e, 63a, and 64e.

63c ⟨`parameter` *part of* `/circom/` 26d⟩+≡

```
        double precision KIREPS
        parameter (KIREPS = 1D-6)
```

64a ⟨*8-byte aligned part of* `/circom/` 26a⟩+≡

```
      double precision elect0, gamma0
      common /circom/ elect0, gamma0
```

$$\int_{1-\epsilon}^{1^+} \mathrm{d}x\, d_{e^\pm}^{\alpha 1\rho}(x) = a_0^{\alpha\rho} + a_1^{\alpha\rho} \int_{1-\epsilon}^{1^-} \mathrm{d}x\, x^{a_2^{\alpha\rho}} (1-x)^{a_3^{\alpha\rho}} \tag{18}$$

Approximately

$$\int_{1-\epsilon}^{1^+} \mathrm{d}x\, d_{e^\pm}^{\alpha 1\rho}(x) = a_0^{\alpha\rho} + a_1^{\alpha\rho} \int_{1-\epsilon}^{1^-} \mathrm{d}x\, (1-x)^{a_3^{\alpha\rho}} = a_0^{\alpha\rho} + a_1^{\alpha\rho} \int_{0^+}^{\epsilon} \mathrm{d}\xi\, \xi^{a_3^{\alpha\rho}} \tag{19}$$

and therefore

$$\int_{1-\epsilon}^{1^+} \mathrm{d}x\, d_{e^\pm}^{\alpha 1\rho}(x) = a_0^{\alpha\rho} + a_1^{\alpha\rho} \frac{1-\epsilon^{a_3^{\alpha\rho}+1}}{a_3^{\alpha\rho}+1} \tag{20}$$

This simple approximation is good enough

64b ⟨*Update* `/circom/` 26g⟩+≡

```
      elect0 = a1(0) + a1(1) * KIREPS**(a1(3)+1) / (a1(3)+1)
      elect0 = elect0 / KIREPS
      gamma0 = a1(4) * KIREPS**(a1(5)+1) / (a1(5)+1)
      gamma0 = gamma0 / KIREPS
```

but we can also use incomplete Beta functions for the exact result:

64c ⟨*Alternative: Update* `/circom/` 64c⟩≡

```
      elect0 = a1(0) + a1(1) * beta (a1(2)+1, a1(3)+1)
     $               * (1d0 - betinc (a1(2)+1, a1(3)+1, 1d0 - KIREPS))
      elect0 = elect0 / KIREPS
      gamma0 = a1(7) + a1(4) * beta (a1(5)+1, a1(6)+1)
     $               * betinc (a1(5)+1, a1(6)+1, KIREPS)
      gamma0 = gamma0 / KIREPS
```

Uses `beta` 90a.

64d ⟨*Local variables for* `circes` 27a⟩+≡

```
      double precision betinc
      external betinc
```

64e ⟨*Calculate version 1 of the non-singular e^+e^- distribution* 64e⟩≡

```
      if (x1 .gt. 1d0) then
         d1 = 0d0
      elseif (x1 .ge. (1d0 - KIREPS)) then
         d1 = elect0
      elseif (x1 .ge. 0d0) then
         d1 = a1(1) * x1**a1(2) * (1d0 - x1)**a1(3)
      else
         d1 = 0d0
      endif
      if (x2 .gt. 1d0) then
         d2 = 0d0
      elseif (x2 .ge. (1d0 - KIREPS)) then
         d2 = elect0
      elseif (x2 .ge. 0d0) then
         d2 = a1(1) * x2**a1(2) * (1d0 - x2)**a1(3)
```

```
          else
             d2 = 0d0
          endif
          kirkee = d1 * d2
```

Uses `kirkee` 63b.

65a ⟨*Calculate version 1 of the non-singular $e^{\pm}\gamma$ distribution* 65a⟩≡

```
          if (x1 .gt. 1d0) then
             d1 = 0d0
          elseif (x1 .ge. (1d0 - KIREPS)) then
             d1 = elect0
          elseif (x1 .ge. 0d0) then
             d1 = a1(1) * x1**a1(2) * (1d0 - x1)**a1(3)
          else
             d1 = 0d0
          endif
          if (x2 .gt. 1d0) then
             d2 = 0d0
          elseif (x2 .gt. KIREPS) then
             d2 = a1(4) * x2**a1(5) * (1d0 - x2)**a1(6)
          elseif (x2 .ge. 0d0) then
             d2 = gamma0
          else
             d2 = 0d0
          endif
          kirkeg = d1 * d2
```

Uses `kirkeg` 65c.

65b ⟨*Calculate version 1 of the non-singular $\gamma\gamma$ distribution* 65b⟩≡

```
          if (x1 .gt. 1d0) then
             d1 = 0d0
          elseif (x1 .gt. KIREPS) then
             d1 = a1(4) * x1**a1(5) * (1d0 - x1)**a1(6)
          elseif (x1 .ge. 0d0) then
             d1 = gamma0
          else
             d1 = 0d0
          endif
          if (x2 .gt. 1d0) then
             d2 = 0d0
          elseif (x2 .gt. KIREPS) then
             d2 = a1(4) * x2**a1(5) * (1d0 - x2)**a1(6)
          elseif (x2 .ge. 0d0) then
             d2 = gamma0
          else
             d2 = 0d0
          endif
          kirkgg = d1 * d2
```

Uses `kirkgg` 66a.

65c ⟨*Subroutines* 24d⟩+≡

```
 double precision function kirkeg (x1, x2)
 implicit none
 double precision x1, x2
 ⟨/circom/ 25b⟩
 double precision d1, d2
 ⟨Initialization check 26e⟩
 kirkeg = -1.0
 if ((ver .eq. 1) .or. (ver .eq. 0)) then
    ⟨Calculate version 1 of the non-singular e±γ distribution 65a⟩
 ⟨else handle invalid versions 30b⟩
 end
```

Defines:
`kirkeg`, used in chunks 63a and 65a.

66a ⟨*Subroutines* 24d⟩+≡

```
 double precision function kirkgg (x1, x2)
 implicit none
 double precision x1, x2
 ⟨/circom/ 25b⟩
 double precision d1, d2
 ⟨Initialization check 26e⟩
 kirkgg = -1.0
 if ((ver .eq. 1) .or. (ver .eq. 0)) then
    ⟨Calculate version 1 of the non-singular γγ distribution 65b⟩
 ⟨else handle invalid versions 30b⟩
 end
```

Defines:
`kirkgg`, used in chunks 63a and 65b.

66b ⟨*Alternative: Subroutines* 66b⟩≡

```
 double precision function betinc (a, b, x)
 implicit none
 double precision x, a, b
 double precision bt, betacf, dlogam
 external betacf, dlogam
 if (x .lt. 0d0 .or. x .gt. 1d0) then
    betinc = 0d0
 else
    if (x .eq. 0d0 .or. x .eq. 1d0) then
       bt = 0d0
    else
       bt = exp(dlogam(a+b)-dlogam(a)-dlogam(b)
$                + a*log(x) + b*log(1d0-x))
    endif
    if (x .lt. (a+1d0)/ (a+b+2d0)) then
       betinc = bt*betacf (a, b, x) / a
    else
       betinc = 1d0 - bt*betacf (b, a, 1d0-x) / b
    endif
 endif
 end
```

67a ⟨*Alternative: Subroutines* 66b⟩+≡

```
      double precision function betacf (a, b, x)
      implicit none
      double precision x, a, b
      integer ITMAX
      double precision EPS
      parameter (ITMAX = 100, EPS = 3D-7)
      double precision am, bm, curr, prev, qab, qap, qam, bz,
     $    ap, bp, app, bpp, em, tem, d
      integer m
      am = 1d0
      bm = 1d0
      curr = 1d0
      qab = a + b
      qap = a + 1d0
      qam = a - 1d0
      bz = 1d0 - qab * x / qap
      do 10 m = 1, ITMAX
        em = m
        tem = 2*em
        d = em * (b - m) * x / ((qam + tem) * (a + tem))
        ap = curr + d*am
        bp = bz + d*bm
        d = - (a + em) * (qab + em) * x / ((a + tem) * (qap + tem))
        app = ap + d * curr
        bpp = bp + d * bz
        prev = curr
        am = ap / bpp
        bm = bp / bpp
        curr = app / bpp
        bz = 1d0
        if (abs (curr - prev) .lt. EPS * abs (curr)) then
           betacf = curr
           return
        endif
   10 continue
      print *, 'betacf: failed to converge'
      betacf = 0d0
      end
```

6.5 Generators

6.5.1 Version 1

Beta distributions have the practical advantage that they have been popular among mathematicians.[**?**]

67b ⟨*Subroutines* 24d⟩+≡

```
      subroutine girce (x1, x2, p1, p2, rng)
      implicit none
```

```
        double precision x1, x2
        integer p1, p2
        external rng
        ⟨/circom/ 25b⟩
        double precision u, w, circgg
        ⟨Particle codes 9b⟩
        ⟨Initialization check 26e⟩
        ⟨x1m, x2m kludge, part 1 68b⟩
        ⟨Select particles p1 and p2 68a⟩
        if (abs(p1) .eq. ELECTR) then
           if (abs(p2) .eq. ELECTR) then
              call gircee (x1, x2, rng)
           elseif (p2 .eq. PHOTON) then
              call girceg (x1, x2, rng)
           endif
        elseif (p1 .eq. PHOTON) then
           if (abs(p2) .eq. ELECTR) then
              call girceg (x2, x1, rng)
           elseif (p2 .eq. PHOTON) then
              call gircgg (x1, x2, rng)
           endif
        endif
        ⟨x1m, x2m kludge, part 2 69a⟩
        end
```

Defines:
`girce`, used in chunks 67b, 16a, and 67.
Uses `ELECTR` 9b, `PHOTON` 9b, `circgg` 35c, `gircee` 69b, `girceg` 69d, and `gircgg` 70c.

68a ⟨*Select particles* `p1` *and* `p2` 68a⟩≡

```
        w = 1d0 / (1d0 + circgg (-1d0, -1d0))
        call rng (u)
        if (u*u .le. w) then
           p1 = POSITR
        else
           p1 = PHOTON
        endif
        call rng (u)
        if (u*u .le. w) then
           p2 = ELECTR
        else
           p2 = PHOTON
        endif
```

Uses `ELECTR` 9b, `PHOTON` 9b, `POSITR` 9b, and `circgg` 35c.

The flavor selection is incorrect, because the relative weights depend on the minimum energy fractions. We resort to a moderately inefficient kludge, because we don't have the distribution functions available yet. We'll have to implement incomplete Beta functions and other horrible things for this. Fortunately, the efficiency can not drop below the relative contribution of e^+e^-.

68b ⟨`x1m`, `x2m` *kludge, part 1* 68b⟩≡

```
   99 continue
```

Crude rejection:

69a ⟨`x1m`, `x2m` *kludge, part 2* 69a⟩≡

```
        if ((x1 .lt. x1m) .or. (x2 .lt. x2m)) goto 99
```

69b ⟨*Subroutines* 24d⟩+≡

```
        subroutine gircee (x1, x2, rng)
        implicit none
        double precision x1, x2
        external rng
        ⟨/circom/ 25b⟩
        double precision u, girceb
        ⟨Initialization check 26e⟩
        if ((ver .eq. 1) .or. (ver .eq. 0)) then
           ⟨Generate version 1 of the e+e- distribution 69c⟩
        ⟨else handle invalid versions 30b⟩
        end
```

Defines:
  `gircee`, used in chunks 16, 69b, 18a, and 67b.
Uses `girceb` 71a.

For version 1 of the parametrizations we rely on `girceb`, a fast generator of β-distribitions:

$$\beta^{a,b}_{x_{\min},x_{\max}}(x) = x^{a-1}(1-x)^{b-1} \cdot \frac{\Theta(x_{\max}-x)\Theta(x-x_{\min})}{I(x_{\min},a,b)-I(x_{\max},a,b)} \tag{21}$$

$$I(x,a,b) = \int_x^1 d\xi\,\xi^{a-1}(1-\xi)^{b-1} \tag{22}$$

69c ⟨*Generate version 1 of the* e^+e^- *distribution* 69c⟩≡

```
        call rng (u)
        if (u .le. a1(0)) then
           x1 = 1d0
        else
           x1 = 1d0 - girceb (0d0, 1d0-x1m, a1(3)+1d0, a1(2)+1d0, rng)
        endif
        call rng (u)
        if (u .le. a1(0)) then
           x2 = 1d0
        else
           x2 = 1d0 - girceb (0d0, 1d0-x2m, a1(3)+1d0, a1(2)+1d0, rng)
        endif
```

Uses `girceb` 71a.

69d ⟨*Subroutines* 24d⟩+≡

```
        subroutine girceg (x1, x2, rng)
        implicit none
        double precision x1, x2
        external rng
        ⟨/circom/ 25b⟩
        double precision u, girceb
        ⟨Initialization check 26e⟩
        if ((ver .eq. 1) .or. (ver .eq. 0)) then
```

```
   ⟨Generate version 1 of the e±γ distribution 70a⟩
⟨else handle invalid versions 30b⟩
end
```

Defines:
  `girceg`, used in chunks 16c and 67b.
Uses `girceb` 71a.

70a ⟨*Generate version 1 of the* $e^{\pm}\gamma$ *distribution* 70a⟩≡

```
call rng (u)
if (u .le. a1(0)) then
   x1 = 1d0
else
   x1 = 1d0 - girceb (0d0, 1d0-x1m, a1(3)+1d0, a1(2)+1d0, rng)
endif
x2 = girceb (x2m, 1d0, a1(5)+1d0, a1(6)+1d0, rng)
```

Uses `girceb` 71a.

70b ⟨*Subroutines* 24d⟩+≡

```
subroutine gircgg (x1, x2, rng)
implicit none
double precision x1, x2
external rng
⟨/circom/ 25b⟩
double precision girceb
⟨Initialization check 26e⟩
if ((ver .eq. 1) .or. (ver .eq. 0)) then
   ⟨Generate version 1 of the γγ distribution 70c⟩
⟨else handle invalid versions 30b⟩
end
```

Uses `girceb` 71a and `gircgg` 70c.

70c ⟨*Generate version 1 of the* $\gamma\gamma$ *distribution* 70c⟩≡

```
x1 = girceb (x1m, 1d0, a1(5)+1d0, a1(6)+1d0, rng)
x2 = girceb (x2m, 1d0, a1(5)+1d0, a1(6)+1d0, rng)
```

Defines:
  `gircgg`, used in chunks 16c, 67b, and 70b.
Uses `girceb` 71a.

6.5.2 Version 2

Retired.

6.5.3 Version 3 and 4

Identical to version 1.

6.6 Utilities

For version 1 of the parametrizations we need a fast generator of β-distribitions:

$$\beta^{a,b}_{x_{\mathrm{min}},x_{\mathrm{max}}}(x) = x^{a-1}(1-x)^{b-1}\cdot\frac{\Theta(x_{\mathrm{max}}-x)\Theta(x-x_{\mathrm{min}})}{I(x_{\mathrm{min}},a,b)-I(x_{\mathrm{max}},a,b)} \tag{23}$$

with the *incomplete Beta-function I:*

$$
\begin{aligned}
I(x,a,b) &= \int_x^1 d\xi\, \xi^{a-1}(1-\xi)^{b-1} && (24) \\
B(a,b) &= I(0,a,b) && (25)
\end{aligned}
$$

This problem has been studied extensively [?] and we can use an algorithm [18] that is very fast for $0 < a \leq 1 \leq b$, which turns out to be the case in our application.

71a ⟨*Subroutines* 24d⟩+≡

```
        double precision function girceb (xmin, xmax, a, b, rng)
        implicit none
        double precision xmin, xmax, a, b
        external rng
        double precision t, p, u, umin, umax, x, w
        ⟨Check a and b 71b⟩
        ⟨Set up girceb parameters 71c⟩
     10 continue
           ⟨Generate a trial x and calculate its weight w 72a⟩
           call rng (u)
        if (w .le. u) goto 10
        girceb = x
        end
```

Defines:
`girceb`, used in chunks 69b, 71a, 69–71, and 73b.

In fact, this algorithm works for $0 < a \leq 1 \leq b$ only:

71b ⟨*Check* `a` *and* `b` 71b⟩≡

```
        if ((a .gt. 1d0) .or. (b .lt. 1d0)) then
           girceb = -1d0
           call circem ('ERROR', 'beta-distribution expects a<=1<=b')
           return
        endif
```

Uses `girceb` 71a.

The trick is to split the interval $[0,1]$ into two parts $[0,t]$ and $[t,1]$. In these intervals we obviously have

$$
x^{a-1}(1-x)^{b-1} \leq \begin{cases} x^{a-1} & \text{for } x \leq t \\ t^{a-1}(1-x)^{b-1} & \text{for } x \geq t \end{cases} \tag{26}
$$

because we have assumed that $0 < a \leq 1 \leq b$. The integrals of the two dominating distributions are t^a/a and $t^{a-1}(1-t)^b/b$ respectively and therefore the probability for picking a random number from the first interval is

$$
P(x \leq t) = \frac{bt}{bt + a(1-t)^b} \tag{27}
$$

We postpone the discussion of the choice of t until later:

71c ⟨*Set up* `girceb` *parameters* 71c⟩≡

```
        ⟨Set up best value for t 73c⟩
        p = b*t / (b*t + a * (1d0 - t)**b)
```

The dominating distributions can be generated by simple mappings

$$\begin{aligned}
\phi : [0,1] &\to [0,1] && (28)\\
u &\mapsto \begin{cases} t\left(\frac{u}{p}\right)^{\frac{1}{a}} & < t \text{ for } u < p \\ t & = t \text{ for } u = p \\ 1-(1-t)\left(\frac{1-u}{1-p}\right)^{\frac{1}{b}} & > t \text{ for } u > p \end{cases} && (29)
\end{aligned}$$

The beauty of the algorithm is that we can use a single uniform deviate u for both intervals:

72a ⟨*Generate a trial* `x` *and calculate its weight* `w` 72a⟩≡

```
call rng (u)
u = umin + (umax - umin) * u
if (u .le. p) then
   x = t * (u/p)**(1d0/a)
   w = (1d0 - x)**(b-1d0)
else
   x = 1d0 - (1d0 - t) * ((1d0 - u)/(1d0 - p))**(1d0/b)
   w = (x/t)**(a-1d0)
endif
```

The weights that are derived by dividing the distribution by the dominating distributions are already normalized correctly:

$$\begin{aligned}
w : [0,1] &\to [0,1] && (30)\\
x &\mapsto \begin{cases} (1-x)^{b-1} & \in [(1-t)^{b-1}, 1] \text{ for } x \le t \\ \left(\frac{x}{t}\right)^{a-1} & \in [t^{1-a}, 1] \text{ for } x \ge t \end{cases} && (31)
\end{aligned}$$

To derive $u_{\text{min,max}}$ from $x_{\text{min,max}}$ we can use ϕ^{-1}:

$$\begin{aligned}
\phi^{-1} : [0,1] &\to [0,1] && (32)\\
x &\mapsto \begin{cases} p\left(\frac{x}{t}\right)^{a} & < p \text{ for } x < t \\ p & = p \text{ for } x = t \\ 1-(1-p)\left(\frac{1-x}{1-t}\right)^{b} & > p \text{ for } x > t \end{cases} && (33)
\end{aligned}$$

We start with u_{min}. For efficiency, we handle the most common cases (small x_{min}) first:

72b ⟨*Set up* `girceb` *parameters* 71c⟩+≡

```
if (xmin .le. 0d0) then
   umin = 0d0
elseif (xmin .lt. t) then
   umin = p * (xmin/t)**a
elseif (xmin .eq. t) then
   umin = p
elseif (xmin .lt. 1d0) then
   umin = 1d0 - (1d0 - p) * ((1d0 - xmin)/(1d0 - t))**b
else
   umin = 1d0
endif
```

Same procedure for $u_{\max}$; again, handle the most common cases (large $x_{\max}$) first:

73a ⟨*Set up* `girceb` *parameters* 71c⟩+≡

```
if (xmax .ge. 1d0) then
   umax = 1d0
elseif (xmax .gt. t) then
   umax = 1d0 - (1d0 - p) * ((1d0 - xmax)/(1d0 - t))**b
elseif (xmax .eq. t) then
   umax = p
elseif (xmax .gt. 0d0) then
   umax = p * (xmax/t)**a
else
   umax = 0d0
endif
```

Check for absurd cases.

73b ⟨*Set up* `girceb` *parameters* 71c⟩+≡

```
if (umax .lt. umin) then
   girceb = -1d0
   return
endif
```

Uses `girceb` 71a.

It remains to choose he best value for t. The rejection efficiency ϵ of the algorithm is given by the ratio of the dominating distribution and the distribution

$$\frac{1}{\epsilon(t)} = \frac{B(a,b)}{ab}\left(bt^a + at^{a-1}(1-t)^b\right). \tag{34}$$

It is maximized for

$$bt - bt(1-t)^{b-1} + (a-1)(1-t)^b = 0 \tag{35}$$

This equation has a solution which can be determined numerically. While this determination is far too expensive compared to a moderate loss in efficiency, we could perform it once after fitting the coefficients a, b. Nevertheless, it has been shown,[18] that

$$t = \frac{1-a}{b+1-a} \tag{36}$$

results in non-vanishing efficiency for all values $1 < a \leq 1 \leq b$. Empirically we have found efficiencies of at least 80% for this choice, which is enough for our needs.

73c ⟨*Set up best value for t* 73c⟩≡

```
t = (1d0 - a) / (b + 1d0 - a)
```

73d ⟨*Subroutines* 24d⟩+≡

```
subroutine circem (errlvl, errmsg)
implicit none
character*(*) errlvl, errmsg
⟨/circom/ 25b⟩
integer errcnt
save errcnt
```

```
data errcnt /0/
if (errlvl .eq. 'MESSAGE') then
   print *, 'circe:message: ', errmsg
elseif (errlvl .eq. 'WARNING') then
   if (errcnt .lt. 100) then
      errcnt = errcnt + 1
      print *, 'circe:warning: ', errmsg
   elseif (errcnt .eq. 100) then
      errcnt = errcnt + 1
      print *, 'circe:message: more than 100 messages'
      print *, 'circe:message: turning warnings off'
   endif
elseif (errlvl .eq. 'ERROR') then
   if (errcnt .lt. 200) then
      errcnt = errcnt + 1
      print *, 'circe:error:   ', errmsg
   elseif (errcnt .eq. 200) then
      errcnt = errcnt + 1
      print *, 'circe:message: more than 200 messages'
      print *, 'circe:message: turning error messages off'
   endif
elseif (errlvl .eq. 'PANIC') then
   if (errcnt .lt. 300) then
      errcnt = errcnt + 1
      print *, 'circe:panic:   ', errmsg
   elseif (errcnt .eq. 300) then
      errcnt = errcnt + 1
      print *, 'circe:message: more than 300 messages'
      print *, 'circe:message: turning panic messages off'
   endif
else
   print *, 'circe:panic:    invalid error code ', errlvl
endif
end
```

6.7 Examples

6.7.1 Distributions

74 ⟨cplot.f 74⟩≡

```
program cplot
implicit none
⟨Particle codes 9b⟩
double precision xmin, xmax, y, roots
integer xory, nstep, p1, p2, acc, ver, rev
double precision x, logx, d, circe
read *, xory, xmin, xmax, nstep, y, p1, p2, roots, acc, ver, rev
call circes (0d0, 0d0, roots, acc, ver, rev, 0)
do 10 logx = log (xmin), log (xmax), log (xmax/xmin) / nstep
```

```
         x = exp (logx)
         d = 0d0
         if (xory .eq. 1) then
            if (p1 .eq. PHOTON) then
               d = circe (x, y, p1, p2)
            else
               d = circe (1d0 - x, y, p1, p2)
            endif
         elseif (xory .eq. 2) then
            if (p1 .eq. PHOTON) then
               d = circe (y, x, p1, p2)
            else
               d = circe (y, 1d0 - x, p1, p2)
            endif
         endif
         if (d .gt. 1d-4) print *, x, d
   10 continue
      end
```

Uses `PHOTON` 9b, `circe` 24d, and `circes` 25a.

6.7.2 Library functions

If Fortran77 only had first class functions, then the following cruft would not be necessary. OK, here's the outline of the adaptive Gauss integration routine from CERNLIB:

75a ⟨*Part one of Gaussian integration* 75a⟩≡

```
      double precision f, a, b, eps
      external f
      double precision Z1, HF, CST
      parameter (Z1 = 1, HF = Z1/2, CST = 5*Z1/1000)
      integer i
      double precision h, const, aa, bb, c1, c2, s8, s16, u
      ⟨Gaussian weights 77b⟩
      h = 0
      if (b .eq. a) go to 99
      const = CST / dabs(b-a)
      bb = a
    1 continue
         aa = bb
         bb = b
    2 continue
         c1 = HF*(bb+aa)
         c2 = HF*(bb-aa)
         s8 = 0
         do 3 i = 1, 4
            u = c2*x(i)
```

Here are now the first two function calls that we have to fill in later in various ways:

75b ⟨*Function call stub* 75b⟩≡

```
               s8 = s8 + w(i) * (f (c1+u) + f (c1-u))
```

Continuing

76a ⟨*Part two of Gaussian integration* 76a⟩≡

```
    3    continue
         s16 = 0
         do 4 i = 5, 12
            u = c2*x(i)
```

And here are the other two function calls:

76b ⟨*Function call stub* 75b⟩+≡

```
               s16 = s16 + w(i) * (f (c1+u) + f (c1-u))
```

Terminating:

76c ⟨*Part three of Gaussian integration* 76c⟩≡

```
    4    continue
         s16 = c2*s16
      if (dabs(s16-c2*s8) .le. eps*(1+dabs(s16))) then
         h = h + s16
         if (bb .ne. b) go to 1
      else
         bb = c1
         if (1 + const*dabs(c2) .ne. 1) go to 2
         h = 0
         print *, 'gauss: too high accuracy required'
         go to 99
      end if
   99 continue
```

This one is still reasonably straightforward

$$\texttt{gauss1} : (f, a, b) \mapsto \int_a^b dx\, f(x) \tag{37}$$

76d ⟨`sample.f` 12a⟩+≡

```
      double precision function gauss1 (f, a, b, eps)
      implicit none
      ⟨Part one of Gaussian integration 75a⟩
      s8 = s8 + w(i) * (f (c1+u) + f (c1-u))
      ⟨Part two of Gaussian integration 76a⟩
      s16 = s16 + w(i) * (f (c1+u) + f (c1-u))
      ⟨Part three of Gaussian integration 76c⟩
      gauss1 = h
      end
```

Defines:
`gauss1`, used in chunks 76d, 12b, 15, and 76.

But this almost identical repeat

$$\texttt{gaussx} : (f, a, b) \mapsto \left(y \mapsto \int_a^b dx\, f(y, x) \right) \tag{38}$$

would not be necassary in a modern programming language with currying:

76e ⟨`sample.f` 12a⟩+≡

```
      double precision function gaussx (f, y, a, b, eps)
      implicit none
      double precision y
      ⟨Part one of Gaussian integration 75a⟩
      s8 = s8 + w(i) * (f (y, c1+u) + f (y, c1-u))
      ⟨Part two of Gaussian integration 76a⟩
      s16 = s16 + w(i) * (f (y, c1+u) + f (y, c1-u))
      ⟨Part three of Gaussian integration 76c⟩
      gaussx = h
      end
```

Defines:
`gaussx`, used in chunks 76 and 77a.

Fortunately, this is the last one we need

$$\texttt{gauss2} : (f, a, b, a_1, b_1) \mapsto \int_a^b dx \int_{a_1}^{b_1} dy\, f(x,y) \\ = \texttt{gauss1}\left(\texttt{gaussx}(f, a, b), a_1, b_1\right) \quad (39)$$

77a ⟨`sample.f` 12a⟩+≡

```
      double precision function gauss2 (f, a, b, a1, b1, eps)
      implicit none
      double precision a1, b1, gaussx
      ⟨Part one of Gaussian integration 75a⟩
      s8 = s8 + w(i) * (gaussx (f, c1+u, a1, b1, eps)
     $                   + gaussx (f, c1-u, a1, b1, eps))
      ⟨Part two of Gaussian integration 76a⟩
      s16 = s16 + w(i) * (gaussx (f, c1+u, a1, b1, eps)
     $                     + gaussx (f, c1-u, a1, b1, eps))
      ⟨Part three of Gaussian integration 76c⟩
      gauss2 = h
      end
```

Defines:
`gauss2`, used in chunks 77a, 12b, 13c, 15, and 77a.
Uses `gaussx` 76e.

77b ⟨*Gaussian weights* 77b⟩≡

```
      double precision w(12), x(12)
      data x( 1) /9.6028985649753623d-1/, w( 1) /1.0122853629037626d-1/
      data x( 2) /7.9666647741362674d-1/, w( 2) /2.2238103445337447d-1/
      data x( 3) /5.2553240991632899d-1/, w( 3) /3.1370664587788729d-1/
      data x( 4) /1.8343464249564980d-1/, w( 4) /3.6268378337836198d-1/
      data x( 5) /9.8940093499164993d-1/, w( 5) /2.7152459411754095d-2/
      data x( 6) /9.4457502307323258d-1/, w( 6) /6.2253523938647893d-2/
      data x( 7) /8.6563120238783174d-1/, w( 7) /9.5158511682492785d-2/
      data x( 8) /7.5540440835500303d-1/, w( 8) /1.2462897125553387d-1/
      data x( 9) /6.1787624440264375d-1/, w( 9) /1.4959598881657673d-1/
      data x(10) /4.5801677765722739d-1/, w(10) /1.6915651939500254d-1/
      data x(11) /2.8160355077925891d-1/, w(11) /1.8260341504492359d-1/
      data x(12) /9.5012509837637440d-2/, w(12) /1.8945061045506850d-1/
```

6.7.3 Generators

6.8 Dumping Parameters

78a ⟨`params.f` 78a⟩≡

```
      program params
      implicit none
      ⟨Accelerator codes 10c⟩
      integer acc, ver, i
      double precision roots(7)
      data roots / 90D0, 170D0, 350D0, 500D0, 800D0, 1000D0, 1500D0 /
      do 10 ver = 7, 8
         print *, 'VERSION ', ver
         do 11 acc = TESLA, XBNDEE
            do 12 i = 1, 7
               print *, '=============================='
               call circes (0d0, 0d0, roots(i), acc, ver, 20020307, 0)
               call dump ()
   12       continue
   11    continue
   10 continue
      end
```

Uses `TESLA` 10c and `circes` 25a.

78b ⟨`params.f` 78a⟩+≡

```
      subroutine dump
      implicit none
      ⟨/circom/ 25b⟩
      ⟨Accelerator codes 10c⟩
      character*9 name
      if (acc .eq. SBAND) then
        name = 'SBAND'
      else if (acc .eq. TESLA) then
        name = 'TESLA'
      else if (acc .eq. JLCNLC) then
        name = 'JLCNLC'
      else if (acc .eq. SBNDEE) then
        name = 'SBAND/EE'
      else if (acc .eq. TESLEE) then
        name = 'TESLA/EE'
      else if (acc .eq. XBNDEE) then
        name = 'JLCNLC/EE'
      end if
      write (*, 1000) name, roots
      write (*, 1001) 'e^+ e^-', lumi
      write (*, 1002) 'e^+/e^-', a1(0)
      write (*, 1003) 'e^+/e^-', 1 - a1(0)
      write (*, 1004) 'e^+/e^-', a1(1), a1(2), a1(3)
      write (*, 1003) 'gamma', a1(7)
      write (*, 1004) 'gamma', a1(4), a1(5), a1(6)
```

```
 1000 format (A9, ' @ ', F5.0, ' GeV')
 1001 format (4X, A7, ' lumi            = ', F7.2,
     $        ' * 10^32 cm^-2 sec^-1')
 1002 format (4X, A7, ' delta strength  = ', F9.5)
 1003 format (4X, A7, ' integral(cont.) = ', F9.5)
 1004 format (4X, A7, ' distribution    = ',
    $         F9.5, ' * x^{', F9.5, '} * (1-x)^{', F9.5, '}')
      end
```

Uses `SBAND` 10c and `TESLA` 10c.

7 Fitting

7.1 Version 1: Factorized Beta Distributions

79a ⟨`fit_v1.f90` 79a⟩≡

```
c fit_v1.f90 -- fitting for circe
⟨Copyleft notice 24b⟩
      program fit
      implicit none
      external fct
      integer i, rcode
      ⟨Declare NPARAM 79b⟩
      ⟨Declare parameters 80a⟩
      ⟨Declare arguments 80b⟩
      ⟨Initialize parameters for fit_v1.f90 80d⟩
      call mninit (5, 6, 7)
      ⟨Load parameters 80c⟩
      call mnseti ('CIRCE: fit version 1          ')
      argv(1) = 1
      call mnexcm (fct, 'SET PRINTOUT              ', argv, 1, rcode, 0d0)
      argv(1) = 1
      call mnexcm (fct, 'CALL FCT                  ', argv, 1, rcode, 0d0)
      call mnexcm (fct, 'MIGRAD                    ', argv, 0, rcode, 0d0)
      call mnexcm (fct, 'MINOS                     ', argv, 0, rcode, 0d0)
      argv(1) = 3
      call mnexcm (fct, 'CALL FCT                  ', argv, 1, rcode, 0d0)
      call mnexcm (fct, 'STOP                      ', argv, 0, rcode, 0d0)
      end
```

Defines:
`fit`, used in chunks 94b and 98a.
Uses `circe` 24d and `fct` 80e.

79b ⟨*Declare* `NPARAM` 79b⟩≡

```
      integer NPARAM
      parameter (NPARAM = 6)
```

Defines:
`NPARAM`, used in chunks 80 and 86b.

80a ⟨*Declare parameters* 80a⟩≡

```
      integer pnum(NPARAM)
      character*10 pname(NPARAM)
      double precision pstart(NPARAM), pstep(NPARAM)
```

Uses `NPARAM` 79b.

80b ⟨*Declare arguments* 80b⟩≡

```
      integer ARGC
      parameter (ARGC = 10)
      double precision argv(ARGC)
```

80c ⟨*Load parameters* 80c⟩≡

```
      do 10 i = 1, NPARAM
         call mnparm (pnum(i), pname(i), pstart(i), pstep (i),
     $                0d0, 0d0, rcode)
         if (rcode .ne. 0) then
            print *, 'fit: MINUIT won''t accept parameter ', pnum(i)
            stop
         endif
 10   continue
```

Uses `NPARAM` 79b.

80d ⟨*Initialize parameters for* `fit_v1.f90` 80d⟩≡

```
      data pnum   /     1,     2,       3,     4,     5,       6 /
      data pname  / '1_e', 'x_e', '1-x_e', '1_g', 'x_g', '1-x_g' /
      data pstart / -1.00, 20.00,    0.20, -1.00,  0.20,   20.00 /
      data pstep  /  0.01,  0.01,    0.01,  0.01,  0.01,    0.01 /
```

80e ⟨`fit_v1.f90` 79a⟩+≡

```
      subroutine fct (nx, df, f, a, mode, g)
      implicit none
      integer nx, mode
      double precision f, df(*), a(*), g
      ⟨Local variables for fct (v1) 81b⟩
      external scale
      if (mode .eq. 1) then
        ⟨Read input data (v1) 80f⟩
      else if (mode .eq. 2) then
        ⟨Calculate ∇f 84a⟩
      endif
      ⟨Calculate f (v1) 84b⟩
 999  continue
      if (mode .eq. 3) then
         ⟨Write output (v1) 86a⟩
      endif
      end
```

Defines:
`fct`, used in chunks 79a and 94.
Uses `scale` 83e.

80f ⟨*Read input data (v1)* 80f⟩≡

```
      ⟨Read data from file 81a⟩
      ⟨Fixup errors 82c⟩
```

⟨*Normalize* 83a⟩

81a ⟨*Read data from file* 81a⟩≡
```
        call gethst ('ee', NDATA, xee, fee, dfee, see, tee, pwr)
        call gethst ('eg', NDATA, xeg, feg, dfeg, seg, teg, pwr)
        call gethst ('ge', NDATA, xge, fge, dfge, sge, tge, pwr)
        call gethst ('gg', NDATA, xgg, fgg, dfgg, sgg, tgg, pwr)
```
Uses `gethst` 81c.

81b ⟨*Local variables for* `fct` *(v1)* 81b⟩≡
```
        integer NDATA
        parameter (NDATA = 20)
        double precision see, tee, dtee, xee(2,0:NDATA+1,0:NDATA+1),
       $     fee(0:NDATA+1,0:NDATA+1), dfee(0:NDATA+1,0:NDATA+1)
        double precision seg, teg, dteg, xeg(2,0:NDATA+1,0:NDATA+1),
       $     feg(0:NDATA+1,0:NDATA+1), dfeg(0:NDATA+1,0:NDATA+1)
        double precision sge, tge, dtge, xge(2,0:NDATA+1,0:NDATA+1),
       $     fge(0:NDATA+1,0:NDATA+1), dfge(0:NDATA+1,0:NDATA+1)
        double precision sgg, tgg, dtgg, xgg(2,0:NDATA+1,0:NDATA+1),
       $     fgg(0:NDATA+1,0:NDATA+1), dfgg(0:NDATA+1,0:NDATA+1)
        double precision pwr
```

81c ⟨`fit_v1.f90` 79a⟩+≡
```
        subroutine gethst (tag, ndata, x, f, df, s, t, pwr)
        implicit none
        character*(2) tag
        integer ndata
        double precision s, t, pwr, x(2,0:ndata+1,0:ndata+1)
        double precision f(0:ndata+1,0:ndata+1), df(0:ndata+1,0:ndata+1)
        integer i, j
        open (10, file = 'lumidiff-'//tag//'.dat')
        read (10, *) pwr
        s = 0d0
        ⟨Read continuum, summing in s 81d⟩
        t = s
        ⟨Read single δ, summing in t 81e⟩
        ⟨Read double δ, summing in t 82b⟩
        close (10)
        end
```
Defines:
`gethst`, used in chunk 81a.

81d ⟨*Read continuum, summing in* `s` 81d⟩≡
```
        do 10 i = 1, ndata
           do 11 j = 1, ndata
              read (10, *) x(1,i,j), x(2,i,j), f(i,j), df(i,j)
              s = s + f(i,j)
   11      continue
   10   continue
```

81e ⟨*Read single δ, summing in* `t` 81e⟩≡
```
        do 20 i = 1, ndata
```

```
            read (10, *) x(1,i,0), f(i,0), df(i,0),
     $                             f(i,ndata+1), df(i,ndata+1)
            x(1,i,ndata+1) = x(1,i,0)
            t = t + f(i,0) + f(i,ndata+1)
   20    continue
```

82a ⟨*Read single δ, summing in* `t` 81e⟩+≡

```
         do 21 i = 1, ndata
            read (10, *) x(2,0,i), f(0,i), df(0,i),
     $                             f(ndata+1,i), df(ndata+1,i)
            x(2,ndata+1,i) = x(2,0,i)
            t = t + f(0,i) + f(ndata+1,i)
   21    continue
```

82b ⟨*Read double δ, summing in* `t` 82b⟩≡

```
         read (10, *) f(0,0), df(0,0), f(0,ndata+1), df(0,ndata+1)
         t = t + f(0,0) + f(0,ndata+1)
         read (10, *) f(ndata+1,0), df(ndata+1,0),
     $             f(ndata+1,ndata+1), df(ndata+1,ndata+1)
         t = t + f(ndata+1,0) + f(ndata+1,ndata+1)
```

`Guinea-Pig` does not provide the full error. A Monte Carlo study shows that it is a reasonable approximation to rescale the bin error by suitable factors. These factors are different for eahc distribution and the factors for the δ-pieces are bigger than those for the continuum parts. The follows factors are for the `slow` parameter set.

82c ⟨*Fixup errors* 82c⟩≡

```
         call fixerr (NDATA, dfee, 20d0, 30d0, 40d0)
         call fixerr (NDATA, dfeg, 15d0, 20d0,  0d0)
         call fixerr (NDATA, dfge, 15d0, 20d0,  0d0)
         call fixerr (NDATA, dfgg, 10d0,  0d0,  0d0)
```

Uses `fixerr` 82d.

82d ⟨`fit_v1.f90` 79a⟩+≡

```
         subroutine fixerr (ndata, df, c, sd, dd)
         implicit none
         integer ndata
         double precision df(0:ndata+1,0:ndata+1), c, sd, dd
         integer i, j
         do 1 i = 1, NDATA
            do 2 j = 1, NDATA
               df(i,j) = c * df(i,j)
    2       continue
    1    continue
         do 3 i = 1, NDATA
            df(0,i) = sd * df(0,i)
            df(i,0) = sd * df(i,0)
            df(ndata+1,i) = sd * df(ndata+1,i)
            df(i,ndata+1) = sd * df(i,ndata+1)
    3    continue
         df(0,0) = dd * df(0,0)
         df(ndata+1,0) = dd * df(ndata+1,0)
```

```
        df(0,ndata+1) = dd * df(0,ndata+1)
        df(ndata+1,ndata+1) = dd * df(ndata+1,ndata+1)
        end
```

Defines:
`fixerr`, used in chunk 82c.

The error on the integrated luminosity is obtained from adding the error in channels in quadrature.

83a ⟨*Normalize* 83a⟩≡

```
        dtee = sumsqu (NDATA, dfee)
        dteg = sumsqu (NDATA, dfeg)
        dtge = sumsqu (NDATA, dfge)
        dtgg = sumsqu (NDATA, dfgg)
```

Uses `sumsqu` 83c.

83b ⟨*Local variables for* `fct` *(v1)* 81b⟩+≡

```
        double precision sumsqu
        external sumsqu
```

Uses `sumsqu` 83c.

83c ⟨`fit_v1.f90` 79a⟩+≡

```
        double precision function sumsqu (ndata, f)
        implicit none
        integer ndata
        double precision f(0:ndata+1,0:ndata+1)
        integer i, j
        double precision s2
        s2 = 0
        do 100 i = 0, NDATA+1
           do 101 j = 0, NDATA+1
              s2 = s2 + f(i,j)*f(i,j)
  101      continue
  100   continue
        sumsqu = sqrt (s2)
        end
```

Defines:
`sumsqu`, used in chunk 83.

83d ⟨*Normalize* 83a⟩+≡

```
        call scale (NDATA, 1d0/tee, fee)
        call scale (NDATA, 1d0/tee, dfee)
        call scale (NDATA, 1d0/tee, feg)
        call scale (NDATA, 1d0/tee, dfeg)
        call scale (NDATA, 1d0/tee, fge)
        call scale (NDATA, 1d0/tee, dfge)
        call scale (NDATA, 1d0/tee, fgg)
        call scale (NDATA, 1d0/tee, dfgg)
```

Uses `scale` 83e.

83e ⟨`fit_v1.f90` 79a⟩+≡

```
        subroutine scale (ndata, s, f)
        implicit none
        integer ndata
```

```
      double precision s, f(0:ndata+1,0:ndata+1)
      integer i, j
      do 100 i = 0, NDATA+1
         do 101 j = 0, NDATA+1
            f(i,j) = s * f(i,j)
 101     continue
 100  continue
      end
```

Defines:
`scale`, used in chunks 80e and 83d.

84a ⟨*Calculate* ∇f 84a⟩≡

```
      print *, 'ERROR: $\nabla f$ n.a.'
      stop
```

Log-likelihood won't fly, because we can't normalize the likelihood function for an unbounded parameter range. Let's use good ole least-squares instead.

84b ⟨*Calculate f (v1)* 84b⟩≡

```
      f = 0d0
      do 10 i = 1, NDATA
         do 11 j = 1, NDATA
            if (dfee(i,j) .gt. 0d0) then
               f = f + ((phie(xee(1,i,j),a) * phie(xee(2,i,j),a)
     $                   - fee(i,j)) / dfee(i,j))**2
            endif
            if (dfeg(i,j) .gt. 0d0) then
               f = f + ((phie(xeg(1,i,j),a) * phig(xeg(2,i,j),a)
     $                   - feg(i,j)) / dfeg(i,j))**2
            endif
            if (dfge(i,j) .gt. 0d0) then
               f = f + ((phig(xge(1,i,j),a) * phie(xge(2,i,j),a)
     $                   - fge(i,j)) / dfge(i,j))**2
            endif
            if (dfgg(i,j) .gt. 0d0) then
               f = f + ((phig(xgg(1,i,j),a) * phig(xgg(2,i,j),a)
     $                   - fgg(i,j)) / dfgg(i,j))**2
            endif
 11      continue
 10   continue
```

Uses `phie` 85d and `phig` 85e.

84c ⟨*Local variables for* `fct` *(v1)* 81b⟩+≡

```
      integer i, j
      double precision phie, phig, delta
      external phie, phig
```

Uses `phie` 85d and `phig` 85e.

84d ⟨*Calculate f (v1)* 84b⟩+≡

```
      if ((a(2) .le. -1d0) .or. (a(3) .le. -1d0/pwr)) then
         print *, 'warning: discarding out-of-range a2/3: ', a(2), a(3)
         ⟨Give up on f 85a⟩
      endif
      delta = 1d0 - exp(a(1)) * beta(a(2)+1d0,a(3)+1d0/pwr)
     $                        * dble(NDATA) / pwr
      if (delta .lt. 0d0) then
         print *, 'warnimg: delta forced to 0 from ', delta
         delta = 0d0
      endif
```

Uses `beta` 90a.

85a ⟨*Give up on f* 85a⟩≡

```
      f = 1d100
      goto 999
```

85b ⟨*Calculate f (v1)* 84b⟩+≡

```
      do 12 i = 1, NDATA
         if (dfee(ndata+1,i) .gt. 0d0) then
            f = f + ((delta*phie(xee(2,ndata+1,i),a)
     $                - fee(ndata+1,i)) / dfee(ndata+1,i))**2
         endif
         if (dfeg(ndata+1,i) .gt. 0d0) then
            f = f + ((delta*phig(xeg(2,ndata+1,i),a)
     $                - feg(ndata+1,i)) / dfeg(ndata+1,i))**2
         endif
         if (dfee(i,ndata+1) .gt. 0d0) then
            f = f + ((delta*phie(xee(1,i,ndata+1),a)
     $                - fee(i,ndata+1)) / dfee(i,ndata+1))**2
         endif
         if (dfge(i,ndata+1) .gt. 0d0) then
            f = f + ((delta*phig(xge(1,i,ndata+1),a)
     $                - fge(i,ndata+1)) / dfge(i,ndata+1))**2
         endif
 12   continue
```

Uses `phie` 85d and `phig` 85e.

85c ⟨*Calculate f (v1)* 84b⟩+≡

```
      if (dfee(ndata+1,ndata+1) .gt. 0d0) then
         f = f + ((delta*delta
     $             - fee(ndata+1,ndata+1)) / dfee(ndata+1,ndata+1))**2
      endif
```

85d ⟨`fit_v1.f90` 79a⟩+≡

```
      double precision function phie (x, a)
      implicit none
      double precision x, a(6)
      phie = exp (a(1) + a(2)*log(x) + a(3)*log(1d0-x))
      end
```

Defines:
  `phie`, used in chunks 84, 85b, and 88b.

85e ⟨`fit_v1.f90` 79a⟩+≡

```
      double precision function phig (x, a)
      implicit none
      double precision x, a(6)
      phig = exp (a(4) + a(5)*log(x) + a(6)*log(1d0-x))
      end
```

Defines:
`phig`, used in chunks 84, 85b, and 88b.

86a ⟨*Write output (v1)* 86a⟩≡

```
      a1(1) = exp(a(1)) * dble(NDATA) / pwr
      a1(2) = a(2)
      a1(3) = a(3) - 1d0 + 1d0/pwr
      a1(4) = exp(a(4)) * dble(NDATA) / pwr
      a1(5) = a(5) - 1d0 + 1d0/pwr
      a1(6) = a(6)
      open (10, file = 'Parameters')
      write (10, 1000) REV, tee / 1D32
 1000 format ('      data xa5lum(@ENERGY@,@ACC@,', I2, ') / ',
     $        E12.5, ' /')
      write (10, 1001) REV,
     $     1d0 - a1(1) * beta(a1(2)+1d0,a1(3)+1d0),
     $     a1(1), a1(2), a1(3), a1(4), a1(5), a1(6),
     $     a1(4) * beta(a1(5)+1d0,a1(6)+1d0)
 1001 format ('      data (xa5(i,@ENERGY@,@ACC@,', I2 ,'),i=0,7) /', /,
     $        '     $  ', 4(E12.5,', '), /,
     $        '     $  ', 3(E12.5,', '), E12.5, ' /')
      close (10)
```

Uses `beta` 90a.

86b ⟨*Local variables for* `fct` *(v1)* 81b⟩+≡

```
      ⟨Declare NPARAM 79b⟩
      double precision beta, a1(NPARAM)
      integer REV
      parameter (REV = 1)
```

Uses `NPARAM` 79b and `beta` 90a.

The average elektron energy in the continuum can be calculated analytically:

$$\begin{aligned}\langle E_{e^\pm}\rangle_{\text{cont}} = E_{\text{beam}}\,\langle x_{e^\pm}\rangle_{\text{cont}} &= E_{\text{beam}}\frac{\int dx\, x^{a_2}(1-x)^{a_3}x}{B(a_2,a_3)}\\ &= E_{\text{beam}}\frac{B(a_2+1,a_3)}{B(a_2,a_3)} = E_{\text{beam}}\frac{a_2+1}{a_2+a_3+2}\end{aligned} \tag{40}$$

86c ⟨*Write output (v1)* 86a⟩+≡

```
      delta = 1d0 - a1(1) * beta(a1(2)+1d0,a1(3)+1d0)
      print *, '< x_e > = ',
     $         delta + (1d0-delta)*(a1(2)+1d0)/(a1(2)+a1(3)+2d0)
```

Uses `beta` 90a.

similarly:

$$\langle E_\gamma\rangle = E_{\text{beam}}\frac{a_5+1}{a_5+a_6+2} \tag{41}$$

86d ⟨*Write output (v1)* 86a⟩+≡

```
      print *, '< x_g > = ',
     $         (a1(5)+1d0)/(a1(5)+a1(6)+2d0)
```

Count the degrees of freedom in `ndof`:

87a ⟨*Write output (v1)* 86a⟩+≡

```
      ndof = 0
      do 40 i = 0, ndata+1
         do 41 j = 0, ndata+1
            if (dfee(i,j) .gt. 0d0) ndof = ndof + 1
            if (dfeg(i,j) .gt. 0d0) ndof = ndof + 1
            if (dfge(i,j) .gt. 0d0) ndof = ndof + 1
            if (dfgg(i,j) .gt. 0d0) ndof = ndof + 1
 41      continue
 40   continue
      print *, 'CHI2 = ', f / ndof
```

87b ⟨*Local variables for* `fct` *(v1)* 81b⟩+≡

```
      integer ndof
```

The error on the luminosity is just the (possibly rescaled) counting error:

87c ⟨*Write output (v1)* 86a⟩+≡

```
      open (10, file = 'Errors.tex')
      write (10, 1099) tee / 1d32, dtee / 1d32, dtee / 1d32
 1099 format ('$', F8.2, '_{-', F4.2, '}^{+', F4.2, '}$')
```

After retrieving the error from `MINUIT`, we have to take care of the mapping of the parameters

$$a'_{1/4} = e^{a_{1/4}} B(a_{2/5}+1, a_{3/6}+1) N_{\text{bins}} \eta^{-1} \Longrightarrow \delta a'_{1/4} = a'_{1/4} \delta a_{1/4} \tag{42}$$

ignoring the errors in the integral (i.e. the Beta function).

87d ⟨*Write output (v1)* 86a⟩+≡

```
      call mnerrs (1,  eplus, eminus, epara, corr)
      ab = a1(1) * beta(a1(2)+1d0,a1(3)+1d0)
      write (10, 1100) ab, abs (ab*eminus), abs (ab*eplus)
 1100 format ('$', F8.4, '_{-', F6.4, '}^{+', F6.4, '}$')
```

Uses `beta` 90a.

87e ⟨*Local variables for* `fct` *(v1)* 81b⟩+≡

```
      double precision ab
```

The other mappings are even more trivial:

$$a'_{2/6} = a_{2/6} - 1 + \eta^{-1} \Longrightarrow \delta a'_{2/6} = \delta a_{2/6} \quad a'_{3/5} = a_{3/5} - 1 + \eta^{-1} \Longrightarrow \delta a'_{3/5} = \delta a_{3/5} \tag{43}$$

87f ⟨*Write output (v1)* 86a⟩+≡

```
      do 110 i = 2, 3
         call mnerrs (i,  eplus, eminus, epara, corr)
         write (10, 1100) a1(i), abs (eminus), abs (eplus)
 110  continue
      call mnerrs (4,  eplus, eminus, epara, corr)
      ab = a1(4) * beta(a1(5)+1d0,a1(6)+1d0)
      write (10, 1100) ab, abs (ab*eminus), abs (ab*eplus)
      do 111 i = 5, 6
         call mnerrs (i,  eplus, eminus, epara, corr)
         write (10, 1100) a1(i), abs (eminus), abs (eplus)
 111  continue
      close (10)
```

Uses `beta` 90a.

88a ⟨*Local variables for* `fct` *(v1)* 81b⟩+≡

```
      double precision eplus, eminus, epara, corr
      integer n
```

88b ⟨*Write output (v1)* 86a⟩+≡

```
      do 30 n = 1, 10
         call pslice ('ee','x',n,NDATA,xee,fee,dfee,phie,phie,a)
         call pslice ('eg','x',n,NDATA,xeg,feg,dfeg,phie,phig,a)
         call pslice ('ge','x',n,NDATA,xge,fge,dfge,phig,phie,a)
         call pslice ('gg','x',n,NDATA,xgg,fgg,dfgg,phig,phig,a)
         call pslice ('ee','y',n,NDATA,xee,fee,dfee,phie,phie,a)
         call pslice ('eg','y',n,NDATA,xeg,feg,dfeg,phie,phig,a)
         call pslice ('ge','y',n,NDATA,xge,fge,dfge,phig,phie,a)
         call pslice ('gg','y',n,NDATA,xgg,fgg,dfgg,phig,phig,a)
 30   continue
      call pslice ('ee','x',21,NDATA,xee,fee,dfee,phie,phie,a)
      call pslice ('eg','x',21,NDATA,xeg,feg,dfeg,phie,phig,a)
      call pslice ('ee','y',21,NDATA,xee,fee,dfee,phie,phie,a)
      call pslice ('ge','y',21,NDATA,xge,fge,dfge,phig,phie,a)
```

Uses `phie` 85d, `phig` 85e, and `pslice` 88d.

UNIX Fortran compiler want backslashes escaped:

88c ⟨*Write output (v1)* 86a⟩+≡

```
      open (10, file = 'Slices.mp4')
      write (10,*) 'picture eslice[], gslice[];'
      do 31 n = 1, NDATA
         write (10,*) 'eslice[', n, '] := ',
     $        'btex $x_{e^\\pm} = ', xee(1,n,1), '$ etex;'
         write (10,*) 'gslice[', n, '] := ',
     $        'btex $x_\\gamma = ', xgg(1,n,1), '$ etex;'
 31   continue
      close (10)
```

88d ⟨`fit_v1.f90` 79a⟩+≡

```
      subroutine pslice (pp, xy, n, ndata, x, f, df, phi1, phi2, a)
      implicit none
      character*2 pp
      character*1 xy
      integer n, ndata
      double precision x(2,0:ndata+1,0:ndata+1)
      double precision f(0:ndata+1,0:ndata+1), df(0:ndata+1,0:ndata+1)
      double precision a(6)
      double precision z
      double precision phi1, phi2, d, delta, pwr, beta
      external phi1, phi2, beta
      integer i
      character*2 digits
      write (digits, '(I2.2)') n
      open (10, file = 'lumidiff-'//pp//xy//digits//'.dat')
      open (11, file = 'lumidiff-'//pp//xy//digits//'.fit')
      open (12, file = 'lumidiff-'//pp//xy//digits//'.chi')
      if (n .eq. ndata+1) then
         pwr = 5d0
         delta = 1d0 - exp(a(1))*beta(a(2)+1d0,a(3)+1d0/pwr)
     $                   * dble(NDATA) / pwr
      else
         delta = 0
      endif
      if (xy .eq. 'x') then
         do 10 i = 1, ndata
            if (df(n,i) .gt. 0d0) then
               if (pp(2:2) .eq. 'g') then
                  z = x(2,n,i)
               else
                  z = 1d0 - x(2,n,i)
               endif
               if (n .eq. ndata+1) then
                  d = delta*phi2(x(2,n,i),a)
               else
                  d = phi1(x(1,n,i),a)*phi2(x(2,n,i),a)
               endif
               write (10,*) z, f(n,i), df(n,i)
               write (11,*) z, d
               write (12,*) z, (f(n,i) - d) / df(n,i)
            endif
 10      continue
      else if (xy .eq. 'y') then
         do 11 i = 1, ndata
            if (df(i,n) .gt. 0d0) then
               if (pp(1:1) .eq. 'g') then
                  z = x(1,i,n)
               else
                  z = 1d0 - x(1,i,n)
               endif
               if (n .eq. ndata+1) then
                  d = phi1(x(1,i,n),a)*delta
               else
                  d = phi1(x(1,i,n),a)*phi2(x(2,i,n),a)
               endif
               write (10,*) z, f(i,n), df(i,n)
               write (11,*) z, d
               write (12,*) z, (f(i,n) - d) / df(i,n)
            endif
 11      continue
      endif
      close (10)
      close (11)
      close (12)
      end
```

Defines:
`pslice`, used in chunk 88b.
Uses `beta` 90a.

90a ⟨`fit_v1.f90` 79a⟩+≡

```
      double precision function beta (a, b)
      implicit none
      double precision a, b
      double precision dlgama
      beta = exp (dlgama(a) + dlgama(b) - dlgama(a+b))
      end
```

Defines:
`beta`, used in chunks 25a, 51b, 54b, 58b, 61a, 64c, 84d, 86–88, and 91c.

90b ⟨`fit_v1` 90b⟩≡

```
#! /bin/sh
# mode=${2-slow}
mode=${2-fast}
root=‘pwd‘
indir=${root}/${3-input}
tmpdir=${root}/tmp
outdir=${root}/output
acc="${1-sband350 sband500 sband800 sband1000 sband1600
         tesla350 tesla500 tesla800 tesla1000 tesla1600
         tesla350-low tesla500-low tesla800-low tesla1000-low tesla1600-low
         xband350 xband500 xband800 xband1000 xband1600}"
```

90c ⟨`fit_v1` 90b⟩+≡

```
xmkdir () {
  for d in "$@"; do
    mkdir $d 2>/dev/null || true
  done
}
rm -fr ${tmpdir}
xmkdir ${outdir} ${tmpdir}
```

90d ⟨`fit_v1` 90b⟩+≡

```
cd ${tmpdir}
cat /dev/null >${outdir}/Params.f90
for a in $acc; do
  case "$a" in
    *1600*) energy=TEV16;;
    *1000*) energy=TEV1;;
     *800*) energy=GEV800;;
     *500*) energy=GEV500;;
  *3[56]0*) energy=GEV350;;
     *170*) energy=GEV170;;
      *90*) energy=GEV090;;
         *) energy=GEV500;;
  esac
  cp ${indir}/${a}_${mode}/lumidiff-??.dat .
  ${root}/fit_v1.bin
  rm -fr ${outdir}/${a}_${mode}
  mkdir ${outdir}/${a}_${mode}
  cp Slices.mp4 ${outdir}
  cp Errors.tex lumidiff-??x[0-9][0-9].??? ${outdir}/${a}_${mode}
  sed -e "s/@ENERGY@/$energy/g" \
      -e "s/@ACC@/`echo $a | tr a-z A-Z | tr -cd A-Z`/g" Parameters \
    >>${outdir}/Params.f90
done
cd ${root}
rm -fr ${tmpdir}
```

91a ⟨fit_v1 90b⟩+≡

```
cat >${outdir}/Params.tex <<'END'
\begin{table}
  \begin{center}
    \renewcommand{\arraystretch}{1.3}
    \begin{tabular}{|c||c|c|c|c|}\hline
      & \texttt{SBAND} & \texttt{TESLA} & \texttt{TESLA'} & \texttt{XBAND}
      \\\hline\hline
END
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

91b ⟨fit_v1 90b⟩+≡

```
line () {
  for a in $acc; do
    case $a in
      *350* | *800* | *1000* | *1600*)
          ;;
      *)  echo -n ' & '
          sed -n $1p ${outdir}/${a}_${mode}/Errors.tex
          ;;
    esac
  done
  echo '\\\hline'
}
(echo '$\mathcal{L}/\text{fb}^{-1}\upsilon^{-1}$'; line 1
 echo '$\int d_{e^\pm}$';                         line 2
 echo '$x_{e^\pm}^\alpha$';                       line 3
 echo '$(1-x_{e^\pm})^\alpha$';                   line 4
 echo '$\int d_\gamma$';                          line 5
 echo '$x_\gamma^\alpha$';                        line 6
 echo '$(1-x_\gamma)^\alpha$';                    line 7
) >>${outdir}/Params.tex
```

91c ⟨fit_v1 90b⟩+≡

```
cat >>${outdir}/Params.tex <<'END'
    \end{tabular}
  \end{center}
  \caption{\label{tab:param}%
```

```
    Version 1, revision 1997 04 16 of the beam spectra at 500 GeV.
    The rows correspond to the luminosity per effective year, the
    integral over the continuum and the powers in the factorized Beta
    distributions~(\ref{eq:beta}).}
\end{table}
END
```

Uses `beta` 90a.

92a ⟨fit_v1 90b⟩+≡

```
cat >>${outdir}/Params.tex <<'END'
\begin{table}
  \begin{center}
    \renewcommand{\arraystretch}{1.3}
    \begin{tabular}{|c||c|c|c|c|}\hline
      & \texttt{SBAND} & \texttt{TESLA} & \texttt{TESLA'} & \texttt{XBAND}
      \\\hline\hline
END
```

Uses `SBAND` 10c, `TESLA` 10c, and `XBAND` 10c.

92b ⟨fit_v1 90b⟩+≡

```
line () {
  for a in $acc; do
    case $a in
      *1000*)
        echo -n ' & '
        sed -n $1p ${outdir}/${a}_${mode}/Errors.tex
        ;;
    esac
  done
  echo '\\\hline'
}
(echo '$\mathcal{L}/\text{fb}^{-1}\upsilon^{-1}$'; line 1
 echo '$\int d_{e^\pm}$';                          line 2
 echo '$x_{e^\pm}^\alpha$';                        line 3
 echo '$(1-x_{e^\pm})^\alpha$';                    line 4
 echo '$\int d_\gamma$';                           line 5
 echo '$x_\gamma^\alpha$';                         line 6
 echo '$(1-x_\gamma)^\alpha$';                     line 7
) >>${outdir}/Params.tex
```

92c ⟨fit_v1 90b⟩+≡

```
cat >>${outdir}/Params.tex <<'END'
     \end{tabular}
  \end{center}
  \caption{\label{tab:param/TeV}%
    Version 1, revision 1997 04 17 of the beam spectra at 1 TeV.}
\end{table}
END
```

92d ⟨fit_v1 90b⟩+≡

```
cat >>${outdir}/Params.tex <<'END'
\begin{table}
```

```
    \begin{center}
      \renewcommand{\arraystretch}{1.3}
      \begin{tabular}{|c||c|c|c|c|}\hline
        & 350 GeV & 500 GeV & 800 GeV & 1600 GeV
        \\\hline\hline
  END
```

93a ⟨fit_v1 90b⟩+≡

```
line () {
  for a in $acc; do
    case $a in
      tesla*-low)
        ;;
      tesla1000)
        ;;
      tesla*)
        echo -n ’ & ’
        sed -n $1p ${outdir}/${a}_${mode}/Errors.tex
        ;;
    esac
  done
  echo ’\\\hline’
}
(echo ’$\mathcal{L}/\text{fb}^{-1}\upsilon^{-1}$’; line 1
 echo ’$\int d_{e^\pm}$’;                           line 2
 echo ’$x_{e^\pm}^\alpha$’;                         line 3
 echo ’$(1-x_{e^\pm})^\alpha$’;                     line 4
 echo ’$\int d_\gamma$’;                            line 5
 echo ’$x_\gamma^\alpha$’;                          line 6
 echo ’$(1-x_\gamma)^\alpha$’;                      line 7
) >>${outdir}/Params.tex
```

93b ⟨fit_v1 90b⟩+≡

```
cat >>${outdir}/Params.tex <<’END’
    \end{tabular}
  \end{center}
  \caption{\label{tab:param/Tesla}%
    Version 1, revision 1997 04 17 of the beam spectra for TESLA.}
\end{table}
END
exit 0
```

Uses `TESLA` 10c.

7.2 Experimental

7.2.1 Quasi One Dimensional

93c ⟨minuit1.f90 93c⟩≡

```
c minuit1.f90 -- fitting for circe
⟨Copyleft notice 24b⟩
```

Uses `circe` 24d.

We're utilizing the familiar "`MINUIT`" package [15].

94a ⟨`minuit1.f90` 93c⟩+≡
  ⟨*Minuit main program* 94b⟩
  ⟨*Function to minimize* 94c⟩

94b ⟨*Minuit main program* 94b⟩≡

```
      program fit
      implicit none
      external fct
      call minuit (fct, 0d0)
      end
```

Uses `fct` 80e and `fit` 79a.

94c ⟨*Function to minimize* 94c⟩≡

```
      subroutine fct (nx, df, f, a, mode, g)
      implicit none
      integer nx, mode
      double precision f, df(*), a(*), g
      ⟨Local variables for fct 94e⟩
      if (mode .eq. 1) then
        ⟨Read input data 94d⟩
      else if (mode .eq. 2) then
        ⟨Calculate ∇f 84a⟩
      endif
      ⟨Calculate f 95a⟩
      if (mode .eq. 3) then
         ⟨Write output 95b⟩
      endif
      end
```

Uses `fct` 80e.

94d ⟨*Read input data* 94d⟩≡

```
      open (10, file = 'minuit.data')
      do 10 i = 1, NDATA
         do 11 j = 1, NDATA
            read (10, *) xi(1,i,j), xi(2,i,j), fi(i,j), dfi(i,j)
            fi(i,j) = fi(i,j)/1d30
            dfi(i,j) = dfi(i,j)/1d30
 11      continue
 10   continue
      close (10)
```

94e ⟨*Local variables for* `fct` 94e⟩≡

```
      integer NDATA
      parameter (NDATA = 20)
      double precision xi(2,NDATA,NDATA),
     $     fi(NDATA,NDATA), dfi(NDATA,NDATA)
      integer i, j, n
      double precision phi, chi, chi2
```

95a ⟨*Calculate f* 95a⟩≡

```
      f = 0d0
      do 110 i = 1, NDATA
         do 111 j = 1, NDATA
            if (dfi(i,j).gt.0d0) then
               f = f + ((phi(xi(1,i,j),xi(2,i,j),a)
     $                    - fi(i,j)) / dfi(i,j))**2
            endif
 111     continue
 110  continue
```

95b ⟨*Write output* 95b⟩≡

```
      chi2 = 0d0
      n = 0
      open (10, file = 'minuit.fit')
      do 210 i = 1, NDATA
         do 211 j = 1, NDATA
            if (dfi(i,j).gt.0d0) then
               chi = (phi(xi(1,i,j),xi(2,i,j),a)-fi(i,j))/dfi(i,j)
               write (10,*) xi(1,i,j), xi(2,i,j),
     $                      1d30 * phi(xi(1,i,j),xi(2,i,j),a),
     $                      1d30 * fi(i,j),
     $                      chi
               chi2 = chi2 + chi**2
               n = n + 1
            else
               write (10,*) xi(1,i,j), xi(2,i,j),
     $                      1d30 * phi(xi(1,i,j),xi(2,i,j),a),
     $                      1d30 * fi(i,j)
            endif
 211     continue
 210  continue
      close (10)
      print *, 'CHI2 = ', chi2/n
```

95c ⟨`minuit1.f90` 93c⟩+≡

```
      double precision function phi (e1, e2, a)
      implicit none
      double precision e1, e2, a(17)
      double precision y1, y2
      y1 = e1 / 250d0
      y2 = e2 / 250d0
      phi = exp (
     $     + a( 1) * 1d0
     $     + a( 2) * log(y1)
     $     + a( 3) * log(1d0-y1)
     $     + a( 4) * log(-log(y1))
     $     + a( 5) * log(-log(1d0-y1))
     $     + a( 6) * y1
     $     + a( 7) * log(y1)**2
     $     + a( 8) * log(1d0-y1)**2
     $     + a( 9) * log(-log(y1))**2
     $     + a(10) * log(-log(1d0-y1))**2
     $     + a(11) * y1**2
     $     + a(12) / log(y1)
     $     + a(13) / log(1d0-y1)
     $     + a(14) / log(-log(y1))
     $     + a(15) / log(-log(1d0-y1))
     $     + a(16) / y1
     $     + a(17) / (1d0-y1)
     $     + a( 2) * log(y2)
     $     + a( 3) * log(1d0-y2)
     $     + a( 4) * log(-log(y2))
     $     + a( 5) * log(-log(1d0-y2))
     $     + a( 6) * y2
     $     + a( 7) * log(y2)**2
     $     + a( 8) * log(1d0-y2)**2
     $     + a( 9) * log(-log(y2))**2
     $     + a(10) * log(-log(1d0-y2))**2
     $     + a(11) * y2**2
     $     + a(12) / log(y2)
     $     + a(13) / log(1d0-y2)
     $     + a(14) / log(-log(y2))
     $     + a(15) / log(-log(1d0-y2))
     $     + a(16) / y2
     $     + a(17) / (1d0-y2)
     $     )
      end
```

96a ⟨`minuit1.sh` 96a⟩≡

```
#! /bin/sh
minuit_bin=`pwd`/minuit1.bin
⟨Process arguments 96b⟩
(
  ⟨Define parameters 97b⟩
  ⟨Fix parameters 97c⟩
  ⟨Fix strategy 97d⟩
  ⟨Run Minuit 97e⟩
) | eval "$minuit_bin $filter"
⟨Maybe plot results 98a⟩
exit 0
```

96b ⟨*Process arguments* 96b⟩≡

```
tmp="$IFS"
  IFS=:
  args=":$*:"
IFS="$tmp"
```

96c ⟨*Process arguments* 96b⟩+≡

```
filter="| \
  awk '/STATUS=(CONVERGED|CALL LIMIT|FAILED)/ { p=1; print }; \
       /@.* \.00000 *fixed/ { next }; \
       /EDM=|CHI2|@/ && p { print }' "
```

97a ⟨*Process arguments* 96b⟩+≡

```
case "$args" in
  *:v:*) filter=;;
esac
```

97b ⟨*Define parameters* 97b⟩≡

```
cat <<END
set title
CIRCE
parameters
1  ’@ 1        ’ 0.00 0.01
2  ’@ lx       ’ 0.20 0.01
3  ’@ l(1-x)   ’ 0.20 0.01
4  ’@ llx      ’ 0.00 0.01
5  ’@ ll(1-x)  ’ 0.00 0.01
6  ’@ x        ’ 0.00 0.01
7  ’@ lx^2     ’ 0.00 0.01
8  ’@ l(1-x)^2 ’ 0.00 0.01
9  ’@ llx^2    ’ 0.00 0.01
10 ’@ ll(1-x)^2’ 0.00 0.01
11 ’@ x^2      ’ 0.00 0.01
12 ’@ 1/lx     ’ 0.00 0.01
13 ’@ 1/l(1-x) ’ 0.00 0.01
14 ’@ 1/llx    ’ 0.00 0.01
15 ’@ 1/ll(1-x)’ 0.00 0.01
16 ’@ 1/x      ’ 0.00 0.01
17 ’@ 1/(1-x)  ’ 0.00 0.01

END
```

97c ⟨*Fix parameters* 97c⟩≡

```
for p in 1  2  3  4  5  6  7  8  9 10 \
        11 12 13 14 15 16 17; do
  case "$args" in
    *:$p=*:*) val=‘echo "$args" | sed ’s/.*:’"$p"’=\\([0-9.-]*\\):.*/\\1/’‘;
              echo set parameter $p $val;
              echo fix $p;;
      *:$p:*) ;;
           *) echo fix $p;;
  esac
done
```

97d ⟨*Fix strategy* 97d⟩≡

```
case "$args" in
  *:S0:*) echo set strategy 0;;
  *:S1:*) echo set strategy 1;;
  *:S2:*) echo set strategy 2;;
esac
```

97e ⟨*Run Minuit* 97e⟩≡

```
cat <<END
migrat 10000 0.01
```

```
  stop
  END
```

98a ⟨*Maybe plot results* 98a⟩≡

```
  case "$args" in
    *:p:*) awk '$5 != "" { print $1, $2, $5 }' minuit.fit > chi2
           awk '$5 != "" { print $1, $5 }' minuit.fit > chix
           awk '$5 != "" { print $2, $5 }' minuit.fit > chiy
           gnuplot -geometry -0+0 plot2 >/dev/null 2>&1
  esac
```

Uses `fit` 79a.

7.2.2 Quasi Two Dimensional

98b ⟨`minuit2.f90` 98b⟩≡

```
  c minuit2.f90 -- fitting for circe
  ⟨Copyleft notice 24b⟩
```

Uses `circe` 24d.

98c ⟨`minuit2.f90` 98b⟩+≡

```
  ⟨Minuit main program 94b⟩
  ⟨Function to minimize 94c⟩
```

98d ⟨`minuit2.f90` 98b⟩+≡

```
        double precision function phi (e1, e2, a)
        implicit none
        double precision e1, e2, a(33)
        double precision y1, y2
        y1 = e1 / 250d0
        y2 = e2 / 250d0
        phi = exp (
       $     + a( 1) * 1d0
       $     + a( 2) * log(y1)
       $     + a( 3) * log(1d0-y1)
       $     + a( 4) * log(-log(y1))
       $     + a( 5) * log(-log(1d0-y1))
       $     + a( 6) * y1
       $     + a( 7) * log(y1)**2
       $     + a( 8) * log(1d0-y1)**2
       $     + a( 9) * log(-log(y1))**2
       $     + a(10) * log(-log(1d0-y1))**2
       $     + a(11) * y1**2
       $     + a(12) / log(y1)
       $     + a(13) / log(1d0-y1)
       $     + a(14) / log(-log(y1))
       $     + a(15) / log(-log(1d0-y1))
       $     + a(16) / y1
       $     + a(17) / (1d0-y1)
       $     + a(18) * log(y2)
       $     + a(19) * log(1d0-y2)
       $     + a(20) * log(-log(y2))
```

```
     $     + a(21) * log(-log(1d0-y2))
     $     + a(22) * y2
     $     + a(23) * log(y2)**2
     $     + a(24) * log(1d0-y2)**2
     $     + a(25) * log(-log(y2))**2
     $     + a(26) * log(-log(1d0-y2))**2
     $     + a(27) * y2**2
     $     + a(28) / log(y2)
     $     + a(29) / log(1d0-y2)
     $     + a(30) / log(-log(y2))
     $     + a(31) / log(-log(1d0-y2))
     $     + a(32) / y2
     $     + a(33) / (1d0-y2)
     $     )
      end
```

99a ⟨`minuit2.sh` 99a⟩≡

```
#! /bin/sh
minuit_bin=`pwd`/minuit2.bin
⟨Process arguments 96b⟩
(
  ⟨Define parameters (2dim) 99b⟩
  ⟨Fix parameters (2dim) 100⟩
  ⟨Fix strategy 97d⟩
  ⟨Run Minuit 97e⟩
) | eval "$minuit_bin $filter"
⟨Maybe plot results 98a⟩
exit 0
```

99b ⟨*Define parameters (2dim)* 99b⟩≡

```
cat <<END
set title
CIRCE
parameters
1  '@ 1        ' 0.00 0.01
2  '@ lx       ' 0.20 0.01
3  '@ l(1-x)   ' 0.20 0.01
4  '@ llx      ' 0.00 0.01
5  '@ ll(1-x)  ' 0.00 0.01
6  '@ x        ' 0.00 0.01
7  '@ lx^2     ' 0.00 0.01
8  '@ l(1-x)^2 ' 0.00 0.01
9  '@ llx^2    ' 0.00 0.01
10 '@ ll(1-x)^2' 0.00 0.01
11 '@ x^2      ' 0.00 0.01
12 '@ 1/lx     ' 0.00 0.01
13 '@ 1/l(1-x) ' 0.00 0.01
14 '@ 1/llx    ' 0.00 0.01
15 '@ 1/ll(1-x)' 0.00 0.01
16 '@ 1/x      ' 0.00 0.01
17 '@ 1/(1-x)  ' 0.00 0.01
```

```
18 ’@ ly       ’ 0.20 0.01
19 ’@ l(1-y)   ’ 0.20 0.01
20 ’@ lly      ’ 0.00 0.01
21 ’@ ll(1-y)  ’ 0.00 0.01
22 ’@ y        ’ 0.00 0.01
23 ’@ ly^2     ’ 0.00 0.01
24 ’@ l(1-y)^2 ’ 0.00 0.01
25 ’@ lly^2    ’ 0.00 0.01
26 ’@ ll(1-y)^2’ 0.00 0.01
27 ’@ y^2      ’ 0.00 0.01
28 ’@ 1/ly     ’ 0.00 0.01
29 ’@ 1/l(1-y) ’ 0.00 0.01
30 ’@ 1/lly    ’ 0.00 0.01
31 ’@ 1/ll(1-y)’ 0.00 0.01
32 ’@ 1/y      ’ 0.00 0.01
33 ’@ 1/(1-y)  ’ 0.00 0.01

END
```

100 ⟨*Fix parameters (2dim)* 100⟩≡

```
for p in 1  2  3  4  5  6  7  8  9 10 \
        11 12 13 14 15 16 17 18 19 20 \
        21 22 23 24 25 26 27 28 29 30 \
        31 32 33; do
  case "$args" in
    *:$p=*:*) val=‘echo "$args" | sed ’s/.*:’"$p"’=\\([0-9.-]*\\):.*/\\1/’‘;
              echo set parameter $p $val;
              echo fix $p;;
      *:$p:*) ;;
           *) echo fix $p;;
  esac
done
```

7.3 Version 2

8 Conclusions

I have presented a library of simple parameterizations of realistic $e^{\pm}$- and γ-beam spectra at future linear e^+e^--colliders. The library can be used for integration and event generation. Emphasis is put on simplicity and reproducibility of the parameterizations for supporting reproducible physics simulations.

Acknowledgements

Daniel Schulte made his simulation code `Guinea-Pig` available and answered questions. Harald Anlauf and Torbjörn Sjöstrand have contributed useful suggestions. The Tesla group at DESY/Zeuthen made error estimates feasible by donating time on the multi-headed number cruncher `Hydra`. The 1996

ECFA/Desy Linear Collider Workshop got me started and provided support. Thanks to all of them.

Identifiers

Refinements

Index

References

[1] H. Murayama and M. E. Peskin, SLAC-PUB-7149, to appear in *Ann. Rev. Nucl. Part. Sci.*; P. Zerwas, DESY 94-001-REV.

[2] P. Chen and R. J. Noble, SLAC-PUB-4050; M. Bell and J. S. Bell, Part. Accl. **24**, 1 (1988); R. Blankenbecler and S. D. Drell, Phys. Rev. Lett. **61**, 2324 (1988); P. Chen and K. Yokoya, Phys. Rev. Lett. **61**, 1101 (1988); M. Jacob and T. T. Wu, Nucl. Phys. **B303**, 389 (1988); V. N. Baier, V. M. Katkov, and V. M. Strakovenkov, Nucl. Phys. **B328**, 387 (1989); R. Blankenbecler, S. D. Drell, and N. Kroll, Phys. Rev. **D40**, 2462 (1989); P. Chen and V. L. Telnov, Phys. Rev. Lett. **63**, 1796 (1989).

[3] K. Yokoya, KEK 85-9, KEK.

[4] P. Chen *et al.*, Nucl. Inst. Meth. **A355**, 107 (1995).

[5] D. Schulte, Ph.D. thesis, in preparation.

[6] R. B. Palmer, Ann. Rev. Nucl. Part. Sci. **40**, 529 (1990).

[7] P. Chen, Phys. Rev. **D46**, 1186 (1992).

[8] Tesla Collaboration, Conceptual Design Report, in preparation.

[9] Desy-Darmstadt Linear Collider Collaboration, Conceptual Design Report, in preparation.

[10] JLC Group, KEK Report 92-16.

[11] NLC ZDR Design Group, SLAC-Report-474.

[12] NLC ZDR Design Group and NLC Physics Working Groups, SLAC-Report-485.

[13] Particle Data Group, Phys. Rev. **D50**, 1173 (1994).

[14] G. Altarelli, R. Kleiss, and C. Verzegnassi, CERN Yellow Report 89-08.

[15] F. James and M. Roos, *MINUIT, Function Minimization and Error Analysis, Release 89.12j*, CERN, Geneva, 1989.

[16] H. Anlauf, IKDA 96/6.

[17] H. Anlauf, private communication.

[18] A. Atkinson and J. Whittaker, Appl. Stat. **28**, 90 (1979).

[19] D. E. Knuth, *Literate Programming*, Vol. 27 of *CSLI Lecture Notes* (Center for the Study of Language and Information, Leland Stanford Junior University, Stanford, CA, 1991).

[20] D. E. Knuth, *TEX: The Program*, Vol. B of *Computers & Typesetting* (Addison-Wesley, Reading, Mass., 1986).

[21] D. E. Knuth, *METAFONT: The Program*, Vol. D of *Computers & Typesetting* (Addison-Wesley, Reading, Mass., 1986).

[22] N. Ramsey, IEEE Software **11**, 97 (1994).

A Literate Programming

A.1 Paradigm

I have presented the sample code in this paper using the *literate programming* paradigm. This paradigm has been introduced by Donald Knuth [19] and his programs TEX [20] and METAFONT [21] provide excellent examples of the virtues of literate programming. Knuth summarized his intention as follows ([19], p. 99)

> "Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do."

Usually, literate programming uses two utility programs to produce two kinds of files from the source

`tangle` produces the computer program that is acceptable to an "illiterate" (Fortran, C, etc.) compiler. This process consists of stripping documentation and reordering code. Therefore it frees the author from having to present the code in the particular order enforced by a compiler for purely technical reasons. Instead, the author can present the code in the order that is most comprehensible.

`weave` produces a documents that describes the program. Extensive cross referencing of the code sections is usually provided, which has been suppressed in this paper. If a powerful typesetting system (such a TEX) is used, the document can present the algorithms in clear mathematical notation alongside the code. These features improve readability and maintainability of scientific code immensely.

A.2 Practice

`Circe` uses the `noweb` [22] system. This system has the advantage to work with any traditional programming language and support the essential features described in section A.1 with minimal effort. `noweb`'s `tangle` program only reorders the code sections, but does not reformat them. Therefore its output can be used just like any other "illiterate" program.

The examples above should be almost self-explaining, but in order to avoid any ambiguities, I give another example:

107a ⟨*Literate programming example* 107a⟩≡
  ⟨*Code that has to be at the top* 107c⟩
  ⟨*Other code* 107b⟩

I can start the presentation with the first line of the "other code":

107b ⟨*Other code* 107b⟩≡

```
line 1 of the other code
```

If appropriate, the first line of the code that has to appear *before* the other code can be presented later:

107c ⟨*Code that has to be at the top* 107c⟩≡

```
line 1 of the code at the top
```

Now I can augment the sections:

108a ⟨*Other code* 107b⟩+≡
```
  line 2 of the other code
```

108b ⟨*Code that has to be at the top* 107c⟩+≡
```
  line 2 of the code at the top
```

The complete "program" will be presented to the compiler as

```
line 1 of the code at the top
line 2 of the code at the top
line 1 of the other code
line 2 of the other code
```

The examples in section 3.1.1 show that this reordering is particularly useful for declaring variables when they are first used (rather than at the beginning) and for zooming in on code inside of loops.

B Fortran Name Space

In addition to the ten procedures and one `common` block discussed in section 3

- `circe`, `circee`, `circeg`, `circgg`,
- `girce`, `gircee`, `girceg`, `gircgg`,
- `circes`, `circel`, `/circom/`,

there are two more globally visible functions which are used internally:

- `circem`: error message handler,
- `girceb`: efficient Beta distribution generator.

Even if the `/circom/` is globally visible, application programs *must not* manipulate it directly. The `circes`, subroutine is provided for this purpose and updates some internal parameters as well.

With features from the current Fortran standard (Fortran90), I could have kept the last two functions and the `common` block private. But since Fortran90 has only been adopted by a small fraction of the high energy physics community, I have decided to remain in the confines of Fortran77 (except for the ubiquitous `implicit none`).

Application programs wishing to remain compatible with future versions of `Circe` must not use `common` blocks or procedures starting with `circe` or `girce`.

C Updates

Information about updates can be obtained

- on the World Wide Web:

  `http://crunch.ikp.physik.th-darmstadt.de/nlc/beam.html`

- by internet FTP:

host: `crunch.ikp.physik.th-darmstadt.de`

user: `anonymous`

password: your email address

directory: `pub/ohl/circe`

- from mailing lists:

  `circe-announce@crunch.ikp.physik.th-darmstadt.de`

  `circe-bugs@crunch.ikp.physik.th-darmstadt.de`

  `circe-discuss@crunch.ikp.physik.th-darmstadt.de`

  Subscriptions are available from

  `majordomo@crunch.ikp.physik.th-darmstadt.de`

Contributions of results from other simulation programs and updated accelerator designs are welcome at

`Thorsten.Ohl@Physik.TH-Darmstadt.de`