Pandora

M&E Peskiri
July 2000
pandora

is an event generator for $e^+e^-$ linear collider physics processes,

intended to handle:

- beam polarization
- beamstrahlung + ISR
- spin correlations and spin asymmetries
- inclusive & arbitrary new hard processes
a general $e^+e^-$ cross section has the form:

$$\sigma = \int dx_1 \, dx_2 \, dx_3$$

$$\frac{dP(h_1)}{dx_1} \quad \frac{d\sigma(h, h_2)}{dx_2} \quad \frac{dP(h_3)}{dx_3}$$

beam \quad process \quad beam

assemble the integrand from beam ad process functions

select weight-1 events from the full distribution

modular design $\rightarrow$ C++
- functionality of pandas
- beam simulation
- event selector
- process construction
- current status
pandora is a class w. constructor

\texttt{P(beam1, beam2, process)}

and methods

\texttt{P.prepare(Events)}
\texttt{P.integral()} \rightarrow \text{returns } \sigma
\texttt{P.getEvent()} \rightarrow \text{returns a weight-1 event}

\texttt{pandora} returns parton-level events
in the \texttt{LEvent} data structure
which includes for each parton:

- 4 vecta
- ID
- parent
- final?
- color chain \leftarrow color linkage
- shower level \leftarrow order to carry out QCD showers
class LEEvent {

/* an LEEvent contains the following components:

[ LV -- an LVlist giving the 4-vectors of partons
[ IIInfo -- an IMatrix giving information on the partons
[ ID = IIInfo[n][1] -- parton ID
[ Parent = IIInfo[n][2] -- entry of parent, or 0 if top-level
[ Final = IIInfo[n][3] -- 1 if final, 0 if intermediate state
[ colorChain = IIInfo[n][4] -- entry of next state along a color-connected chain, of -1 if the final color
0 if color-singlet
-11 in this place signals a tau-L
-12 in this place signals a tau-R
-13 in this place signals a tau+L
-14 in this place signals a tau+R

[ showerLevel = IIInfo[n][5] -- order in which to carry out parton showers; the two partons with 1, 2, etc
are shower partners
0 in this place signals a nonshowering color singlet */

public:

friend class LVlist;

LEEvent(int N): LV(N), IIInfo(1,N,1,5), highestlevel(0) {};
LEEvent(const LEEvent & LE);
LEEvent(const LVlist & L);

int n() const; /* returns number of vectors in the event */

void read(int m, int id, int parent, int final,
        int chain, const LVector & V);
void readid(int m, int id, int parent, int final, int chain);

int level() const; /* return highestlevel */

void raiselevels(); /* increase all parton shower levels by 1,
and increase highestlevel by 1 */

void addshower(int i, int j);
/* add a shower between particles i and j, raising all other
showers to a higher level */
these partonic events can be hadronized
by PYTHIA

Masako Iwasaki  pandora-pythia

- insert the event in PYTHIA as external processes
- requests QCD showers
- requests hadronization according to color linkage
- decays polarized $\tau$'s w. TAUOLA
- writes final events to an external file in STFHEP format
int main(int argc, char* argv[]){

    char* outfile = argv[1];
    int nEvent = atoi(argv[2]);

    /* define the pandora event selection in this space */

double ECM = 500. ; // Center of mass energy
double Pol_e = 0. ; // Polarization for electron
ebeam b1(ECM/2.0, Pol_e, electron, electron);
ebeam b2(ECM/2.0, 0.0, positron, positron);

    b1.setup(NLC500);
b2.setup(NLC500);

eetot ttbar pr;
pandora P(b1,b2,pr);

    pandorarun PR(P, epluseminus, ECM, nEvent);

    /* end of definition */

    PR.initialize(outfile);

    PR.getevents();

    PR.terminate();
}

}
Event 4
\[ e^+e^- \rightarrow t\bar{t} \rightarrow W^+b \ W^-\bar{b} \]

Parton : 1  ID: 6  Parent: 0  Final?: 0  Chain to: -1  Shower Level: 1
Vector:  < 252.402 96.8278 -136.447 70.3312

Parton : 2  ID: 5  Parent: 1  Final?: 1  Chain to: -1  Shower Level: 3
Vector:  < 117.633 -4.74508 -96.9145 66.5024

Parton : 3  ID: 24 Parent: 1  Final?: 0  Chain to: 0  Shower Level: 3

Parton : 4  ID: -1 Parent: 3  Final?: 1  Chain to: 5  Shower Level: 4
Vector:  < 76.7714 63.0539 -16.5043 40.5668

Parton : 5  ID: 2  Parent: 3  Final?: 1  Chain to: -1  Shower Level: 4
Vector:  < 57.9974 38.519 -23.0282 -36.7381

Parton : 6  ID: -6 Parent: 0  Final?: 0  Chain to: 1  Shower Level: 1
Vector:  < 247.598 -96.8278 -136.447 -70.3312

Parton : 7  ID: -5 Parent: 6  Final?: 1  Chain to: 2  Shower Level: 2
Vector:  < 51.6563 13.0274 38.5341 31.8399

Parton : 8  ID: -24 Parent: 6  Final?: 0  Chain to: 0  Shower Level: 2

Parton : 9  ID: 15 Parent: 8  Final?: 1  Chain to: -11  Shower Level: 0
Vector:  < 66.8357 7.28161 33.6148 -57.3065

Parton : 10 ID: -16 Parent: 8  Final?: 1  Chain to: 0  Shower Level: 0
Vector:  < 129.106 -102.574 64.2982 -44.8646
Event in $e^+e^- \to t\bar{t} \to W^+b \, W^-\bar{b}$

<table>
<thead>
<tr>
<th>Parton</th>
<th>ID</th>
<th>Parent</th>
<th>Final?</th>
<th>Chain to</th>
<th>Shower Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>121.226</td>
</tr>
<tr>
<td>Vector:</td>
<td>249.857</td>
<td>-86.9575</td>
<td>-98.2031</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-29.3998</td>
</tr>
<tr>
<td>Vector:</td>
<td>62.9771</td>
<td>-43.2992</td>
<td>-35.0278</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>150.626</td>
</tr>
<tr>
<td>Vector:</td>
<td>186.88</td>
<td>-43.6583</td>
<td>-63.1753</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>107.897</td>
</tr>
<tr>
<td>Vector:</td>
<td>138.207</td>
<td>-29.05</td>
<td>-81.3364</td>
<td>42.7287</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td>18.1611</td>
</tr>
<tr>
<td>Vector:</td>
<td>48.6721</td>
<td>-14.6083</td>
<td>18.1611</td>
<td>42.7287</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>121.226</td>
</tr>
<tr>
<td>Vector:</td>
<td>250.143</td>
<td>86.9575</td>
<td>98.2031</td>
<td>-121.226</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-5</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>55.7517</td>
</tr>
<tr>
<td>Vector:</td>
<td>55.7517</td>
<td>-43.0487</td>
<td>17.6867</td>
<td>-30.6959</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-24</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>194.392</td>
</tr>
<tr>
<td>Vector:</td>
<td>194.392</td>
<td>130.006</td>
<td>80.5164</td>
<td>-90.5302</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>-1</td>
<td>121.393</td>
</tr>
<tr>
<td>Vector:</td>
<td>121.393</td>
<td>105.81</td>
<td>21.1496</td>
<td>-55.6167</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-2</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>72.9989</td>
</tr>
<tr>
<td>Vector:</td>
<td>72.9989</td>
<td>24.1966</td>
<td>59.3668</td>
<td>-34.9135</td>
<td></td>
</tr>
</tbody>
</table>
Example of pandora parton-level output:

\[ e^+ e^- \rightarrow W^+ W^- \]

W mass
W \& \nu energy

\[ e^+ e^- \rightarrow t \bar{t} \]

W, t mass
W \& \bar{\nu} energies

\[ e^+ e^- \rightarrow h^0 Z^0 \]

h^0 Z^0 energies
WZ masses in \( h^0 \rightarrow WW^*, ZZ^* \)
h^0 BR's
$e^+e^- \rightarrow W^+W^-$
$e^+e^- \rightarrow t\bar{t}$

![Graph showing $m(t)$ and $m(W)$](image)
$e^+e^- \rightarrow t\bar{t}$

beamstrahlung + ISR effect
$e^+ e^- \rightarrow t \bar{t}$
beamstrahlung + ISR on
$e^+e^- \rightarrow \zeta^0 \eta^0$
$e^+ e^- \rightarrow Z^0 h^0$
$e^+e^- \rightarrow Z^0 h^0$

$H^0$ BR's

decay mode id

$\bar{b}b$

$W^+ W^*$

$Z Z^*$

$\gamma \gamma$

$\gamma Z$

invis.
beam class:

$e^+ e^-$ beams have

ISR           Fadin - Kuraev
+ beamstrahlung 'consistent Yokoya - Chen'

input

a standard design

<table>
<thead>
<tr>
<th>NLC / JLC</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESLA</td>
<td>500</td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>CLIC</td>
<td>500</td>
<td>1000</td>
<td>3000</td>
</tr>
</tbody>
</table>

a machine parameter

$N \beta_x \beta_y \sigma_x \sigma_y \sigma_z$
class beam {

protected:

double Ebeam, polarization; /* basic beam parameters: energy and polarization */

int from, to; /* particle ID's:
from -> particle nominally entering the collision
to   -> particle making the hard reaction */

public:

int n; /* number of integration variables */

DVector distrib; /* placeholder for the distribution functions for each helicity
generated by the beam */

inline double getEbeam(){return Ebeam;}
inline double getPolarization(){return polarization;}

beam(double EB, double Pol, int From, int To, int N): Ebeam(EB),
polarization(Pol), n(N), from(From), to(To), distrib(-1,1){};

virtual double computeX(DVector & Z) = 0;
/* return the value of the longitudinal fraction x determined
by the integration variables Z,
and write the needed intermediate fractions to class variables */

virtual void distribution() = 0;
/* use the results stored in computeX to compute the distribution function f(x) for each helicity
at the position determined by the integration variables Z
the distribution is normalized to: sum(helicity, to) int f(x) dx = */

}
Figure 1: Comparison of Guinea Pig simulation data on the electron energy spectrum with various analytic approximations for the NLC-500B design parameters: blue-P1, green-P2, red-YC1 over YC0, magenta-C2.

Guinea Pig simulation data from Kathy Thompson!
Figure 2: Figure 1 with a logarithmic scale, showing the low-energy, low-\(x\) tail of the distribution.
Figure 3: Blowup of Figure 1 concentrating on the highest-\( x \) bins.
event selection

weight - 1 is an approximation after \( \frac{d\sigma}{dx} \) is given

use VEGAS algorithm or in BASES/SPRING

model factor by adjusting a coordinate grid

find max. weight

keep pts w. probability

\[
\left( \frac{\text{weight}}{\text{max. weight}} \right)
\]

standard VEGAS chooses grid to minimize variance

if instead minimize max. weight, factn 4-10 speedup!

event selection times:  

\( e^+e^- \rightarrow \tau^+\tau^- \quad e^+e^- \rightarrow \ell^+\ell^- \quad e^+e^- \rightarrow 2\gamma \)

2 msec. \( \cdots \) 10 msec.
Is there a better algorithm?

Actually,

pandora - Vegas MC - MonteCarlo (abstract class)

pandora uses any methods (interface)
of MonteCarlo

so any other subclass of MonteCarlo can be
freely substituted for VegasMC

Jadach: Foam
Y. Chen: (another fractal model)
class MonteCarlo{

    public:

    int N; /* number of variables integrated over [0,1] */
    MonteCarlo(int N); /* initializes MonteCarlo variables */
    virtual double surface(DVector & X)=0; /* function integrated */

    virtual void prepare(int nEvents, int nseed = 1)=0;
    /* adaptation; nEvents is the number of events to be used in the adaptation step */

    virtual DVector getPoint()=0;
    virtual DVector getPoint(double & weight) = 0;

    void reset(int nseed = 1);
    /* resets the Monte Carlo counters, and resets integration results to 0 */

    void resetMC(int nseed = 1);
    /* reset the Monte Carlo counters only */

    double integral(double & sd);
    /* returns the accumulated value of the integral, and the error */
    /* data for computation of the integral */

    double Ip, Np, I2p;

    /* running estimates of the integral */
    integral = Ip/Np;
    sd = sqrt(I2p - Ip*Ip/Np)/Np

    double maxweight, threshold, maxratio;
    /* data of the Monte Carlo selection */

    int presented, accepted, bad;
    /* Monte Carlo counters: */
    /* presented = events sampled */
    /* accepted = events accepted */
    /* bad = events with weight > threshold */
    /* by default, threshold is set at 2.0 * maxweight, but this can be adjusted by setThreshold below */
    maxratio = highest weight found

    void setThreshold(double x);
    /* resets threshold = x * maxweight */

    void printIntegral();

    void printStatistics();
}
process class

must implement the operations:

- test if \( \bar{X} \) is in allowed phase space
- compute differential cross section
- construct partonic final state

more specifically →
class process {

public:

    int n;        /* number of integration variables X[i] in [0,1]
                   needed to specify the final state of the process */

    DMatrix cs;

    process(int N): n(N), cs(-1,1,-1,1){}

    virtual int validEvent(DVector & X, double s, double beta) = 0;
    /* determine whether the LEEvent, boosted longitudinally by beta,
       satisfies the kinematic cuts defining the cross section */

    virtual double computeKinematics(DVector & X, double s) = 0;
    /* compute the kinematic variables which determine the final-state
       momenta, and write the answers into appropriate class variables
       of the process.
       the function should return J = the Jacobian of the transformation
       to the variables X_i from the usual variables for
       expressing differential cross sections. For example, if there
       is one integration variable x1 = (1 + cos theta)/2, then
       J = d cos theta/ dx1 = 2
       This form allows crosssection above to return the usual formula for
       the differential cross section */

    virtual void crosssection() = 0;
    /* compute the cross section from the kinematic variables
       fixed by computeKinematics, without consideration of cuts.
       The cross section should be returned by filling the class variable
       DMatrix cs(-1,1,-1,1), a 3 x 3 matrix which can hold the
       cross sections for various initial helicities
       from [-1,-1] to [1,1] */

    virtual LEEvent buildEvent() = 0;
    /* return the parton-level LEEvent determined by the kinematic variables
       fixed by computeKinematics */

    virtual LVlist buildVectors() = 0;
    /* use the results of computeKinematics to
       compute the list of 4-vectors determined by the kinematic variables
       fixed by computeKinematics */
#ifndef EETOTTBAR_H
#define EETOTTBAR_H

#include "pandora.h"
#include "WZtdecay.h"
#include "processes.h"

class eetottbar: public twototwomm {

/* integration variables: */

    cos theta = 2 X[1] - 1 in the CM

    m(t) =
        sqrt( M^2 + M*Gamma * tan (PI - 2 Gamma/M) * (X[2] - 1/2))

    m(tbar) =
        sqrt( M^2 + M*Gamma * tan (PI - 2 Gamma/M) * (X[3] - 1/2))

    cos chi = 2 X[4] - 1 top decay angles

    psi = 2 PI X[5]

    cos chib = 2 X[6] - 1 tbar decay angles

    psib = 2 PI X[7]

    m(W+) =
        sqrt( M^2 + M*Gamma * tan (PI - 2 Gamma/M) * (X[8] - 1/2))

    cos chiW = 2 X[9] - 1 W+ decay angles

    psiW = 2 PI X[10]

    m(W-) =
        sqrt( M^2 + M*Gamma * tan (PI - 2 Gamma/M) * (X[11] - 1/2))

    cos chiWb = 2 X[12] - 1 W- decay angles

    psiWb = 2 PI X[13]

*/

public:

eetottbar();

int validEvent(DVector & X, double s, double beta);
double computeKinematics(DVector & X, double s);
void crosssection();
LVlist buildVectors();
LEvent buildEvent();

double simplecrosssection(double cost, double s);
    /* returns the unpolarized differential cross section as a
     * function of cos theta    */

/* buildEvent creates an LEvent with 10 LVectors with the following
   IDs:

    1  ->  t
    +  2  ->  b
    +  3  ->  W+
    +  4  ->  W+ decay l+ or qbar
    +  5  ->  W+ decay nu or q
    +  6  ->  tbar
    +  7  ->  bbar
    +  8  ->  W-
    +  9  ->  W- decay l- or q
    +  10  ->  W- decay nubar or qbar

*/
How does one write a process class?

1. Compute helicity amplitudes for the process.

Pandora's conventions: View processes in the event plane (works up to $2 \rightarrow 3$)

For a vector boson in $+3$ direction

$$\xi_{+1}^\mu = \frac{1}{2}(0, 1, i, 0) \quad \xi_{0}^\mu = \left(\frac{k}{m}, 0, 0, \frac{E}{m}\right) \quad \xi_{-1}^\mu = \frac{1}{2}(0, 1, -i, 0)$$

For a massive fermion in $+3$ direction

(Ass 2-copmut notation!)

$$U_{+\frac{1}{2}} = \sqrt{E - p} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad U_{-\frac{1}{2}} = \sqrt{E + p} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$U_{+\frac{1}{2}} = \sqrt{E + p} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad U_{-\frac{1}{2}} = \sqrt{E - p} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

+ Notate in plane $1$ other configurations.
(2) Call standard decay amplitudes for mesonic particles
decay classes one writes for

\[ WZ \rightarrow t \bar{t} h^0 \] (SM Higgs)

(It is crucial to adhere to the common conventions.)
class Wplusdecay: public decaytotwoz 

/*
   chi is the angle from the W direction to the right-handed fermion
   (ie, 1+)
   psi is the rotation angle about the W direction of motion */

public:

Wplusdecay();

DVector BRs(); /* returns the BR's to (u,c,e, mu, tau)*/

void decayVamp(double coschi, double psi);
/* fill the vector Camp[Whelicity] */

CVector Camp;

double divideby;
/* for an on-shell W decay,
   integrate the squared amplitude over d coschi d phi;
   then divide by divideby */

/* Wplusdecay inherits the method: */

LVlist buildDecayVectors(double m, double coschi, double psi)

returns a list of three LVectors in the order
1  W+
2  l+ or qbar
3  nu or q

the kinematics of the decay are based on m, not on the argument of
the constructor */

void placeIDs(LEvent & LE, int i, int parent);
/* appropriately fills in 3 rows of an LEvent, beginning with
   row i, with the ID information for a real W+
   decay to quark or lepton pairs
   Pass the row corresponding to the parent of the W+ in parent */

void placeIDs(LEvent & LE, int i, int parent, int ID);
/* appropriately fills in 3 rows of an LEvent, beginning with
row i, with the ID information for the products of a
virtual W decay to quark or lepton pairs.
Pass the row corresponding to the parent of the decaying particle
in parent, and pass the ID of this particle in ID */
class Higgsdecay : public generaldecay {

public:

    Higgsdecay(double mh);

    /* The Higgs decays to final states containing different numbers of
    final particles and different numbers of integration
    variables. The value x passed to the Higgsdecay functions
    keeps track of this. Based on the value of x, the Higgsdecay
    class method choosescase will choose one of the following channe
    and the other decay functions will act accordingly:

    1 b bbar
    2 c cbar
    3 t tbar
    4 tau+ tau-
    5 g g
    6 W W (including W W*)
    7 Z Z (including Z Z*)
    8 gamma gamma
    9 Z gamma
    10 invisible final states
        (coded in the LEvent as nu nubar )

    The default is to take the partial widths given in the Minimal
    Standard Model. However, the following commands allow one
    to modify these widths:
    */

    void newWidth(int channel, double Width);
    /* resets Gamma(h-> channel) to the indicated value,
       and recomputes the branching ratios */
    void onlyDecay(int channel);
    /* sets all decay widths except Gamma(h-> channel) to zero */

    DVector partialwidths();
    /* returns the partial widths as a DVector(1,10)*/
    DVector BRs();
    /* returns the BRs as a DVector(1,10) */
    DVector SMwidths();
    /* returns the Standard Model partial widths as a DVector(1,10)*/

    void newmhmass(double mh);
    /* set the Higgs mass to the new value, and sets
       partial widths to the new Standard Model values */
    double mass(){ return mh; }

    int validDecay(DVector & X, int i, double MH);
    /* returns 0 if X does not correspond to a physically correct
       decay configuration; returns 1 otherwise
       this function uses the 13 doubles in X beginning with X[i]
       and the mass MH of the h computed by the production class
Inherent from classes with compute react × kinematics
these include finite width effects
and cut off singularities (e.g. at $s \approx 0$)
treatment of finite width:

Breit Wigner about $M \rightarrow m_1, m_2 + \sqrt{s}$

use $E_{\text{ia}} = (p^2 + m_i^2)^{1/2}$ for kinematics

$E = (p^2 + M^2)^{1/2}$ to compute amplitudes,
class twototwomm : public process {
    /* integration variables:
    \[
    \cos \theta = 2.0 \cdot X[1] - 1.0 \quad \text{in the CM}
    \]
    \[
    \text{mass}_1 = \sqrt{M_1 \cdot M_1 + M_1 \cdot \Gamma_1 \cdot \tan \left( \left( \pi - 2 \frac{\Gamma_1}{M} \right) \cdot (X[2] - 1/2) \right)}
    \]
    \[
    \text{mass}_2 = \sqrt{M_2 \cdot M_2 + M_2 \cdot \Gamma_2 \cdot \tan \left( \left( \pi - 2 \frac{\Gamma_2}{M} \right) \cdot (X[3] - 1/2) \right)}
    \]

public:

twototwomm(int N, double M1, double G1, double M2, double G2);

int validEvent(DVector & X, double s, double beta);

double computeKinematics(DVector & X, double s);
LVlist buildVectors();

protected:

double M1, G1, M2, G2;
double m1, m2;
double s, p, E1a, E2a, E1, E2, cost, sint, phi;
    /* center of mass momentum and energy and cos theta, sin theta
    \[
    E_{1a} = (p \cdot p + m_1 \cdot m_1), \quad E_2 = (p \cdot p + m_2 \cdot m_2),
    \]
    \[
    E_1 = (p \cdot p + M_1 \cdot M_1), \quad E_2 = (p \cdot p + M_2 \cdot M_2),
    \]*/
class twototwomzt : public process {

/* integration variables:
   cos theta = tanh( (2 * X[1]-1) * 10)
   to give the cross section in the CM up to 10^-5 rad
   mass = sqrt( M*M + M*Gamma * tan [(PI - 2 Gamma/M)* (X[2] - 1/2)]) */

public:

   twototwomzt(int N, double M, double G);
   twototwomzt(int N, double M, double G, double thetamin,
               double ptmin, double Emin);
   twototwomzt(int N, double M, double G, double thetamin, double thetamax,
               double ptmin, double ptmax, double Emin, double Emax);

/* the second of these computes the cross section over the region
   in which the massless particle (usually a photon) obeys,
   in the lab frame, | theta | > thetamin, E > Emin, |pt| > ptmin.
   The default values are: thetamin = 10 mrad, ptmin = 0, Emin = 2 GeV.
   These default values
   are implemented in the constructor with no arguments.
   The third constructor allows one to produce disjoint event samples. */

   int validEvent(DVector & X, double s, double beta);

   double computeKinematics(DVector & X, double s);
   LVlist buildVectors();
4. Code the g.m. helicity amplitudes

5. Construct the final state parton momenta using bosons + jets from 4-vector lists from decay classes

6. Add ID's etc. to LEvent using sections from decay classes
LVlist eetottbar::buildVectors()
   /* prepare t decay products */
   LVlist Lt = TD.buildDecayVectors(m1, mW, coschi, psi, coschiW, psiW);
   Lt.boost(p/E1a);
   /* prepare tbar decay products */
   LVlist Ltb = TBD.buildDecayVectors(m2, mWb, coschib, psib, coschiWb, psiWb);
   Ltb.boost(p/E2a);
   Ltb.reverseinplane();
   LVlist LL = merge(Lt, Ltb);
   LL.rotateinplane(cost);
   LL.rotate(\phi);
   return LL;
}

LEvent eetottbar::buildEvent()
   LEvent LE(buildVectors());
   TD.placeIDs(LE, 1, 0);
   TBD.placeIDs(LE, 6, 0, 1, 2);
   LE.addshower(1, 6);
   return LE;
current status of included processes:

$e^+e^-, e^-e^-$ beam classes

$e^+e^- \rightarrow l^+l^-, e^+e^-, \gamma\gamma, t\bar{t}$

$W^+W^-, Z\gamma, Z\gamma$, $Z^0\gamma$

$e^+e^- \rightarrow t\bar{t}$ w. nonstandard couplings (C. Ferretti)

$\gamma\gamma \rightarrow l^+l^- e^+e^- t\bar{t}$

goal for root distribution (by end of July 2000)

$\gamma\gamma$ beam class

$\gamma\gamma \rightarrow \gamma\gamma e^-Z^0 \nu\bar{\nu}$

$\gamma\gamma \rightarrow h^0$