Matrix elements comparison

numerical problems

Ola

Deutsches Elektronen-Synchrotron (DESY)

30.06.2015

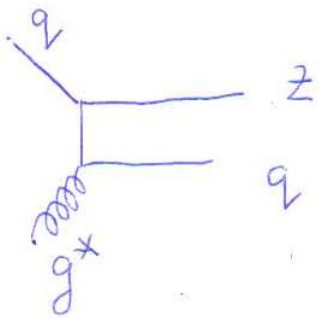
Physics & Cookies

Overview

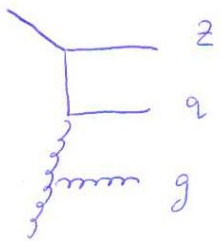
Idea of the project

We would like to compare k_T dependence of two cross sections calculated with two matrix elements:

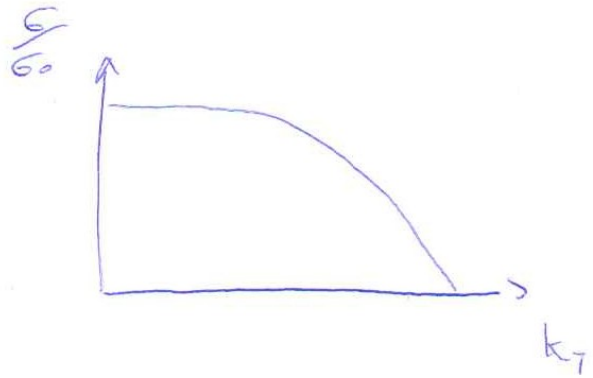
ME in of-shell case:



ME in collinear case:



we've never reached this point :(



Tools

- Fortran code:
  it uses phase space generator from Cascade (I call it fphase in this presentation) and BASES integration package
- C++ code:
  it uses TGenPhaseSpace from ROOT and gnu Vegas integration package
- C++ code:
  with fphase and BASES converted into C++ version

How the codes work in general

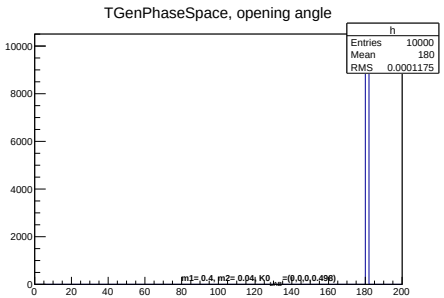
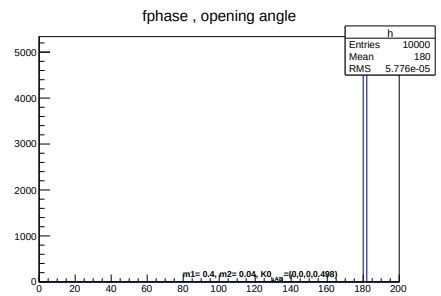
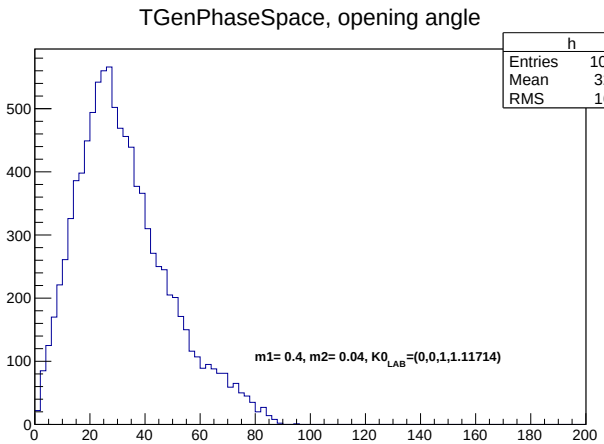
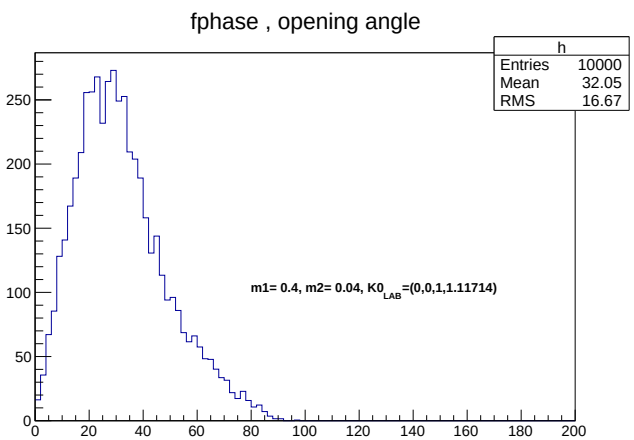
- initial particles: two protons
  $p1 = (0, 0, pz, pz)$, $p2 = (0, 0, -pz, pz)$
  pz is fixed
- 4 random numbers to generate initial partons,
  $p_{ini1} = (\sqrt{kt_1}cos(2.\pi x_2), \sqrt{kt_1}sin(2.\pi x_2), x_0 p1.Pz(), x_0 p1.E())$,
  $p_{ini2} = (\sqrt{kt_2}cos(2.\pi x_3), \sqrt{kt_2}sin(2.\pi x_3), x_1 p2.Pz(), x_1 p2.E())$
  $kt_1 = 0$, kt_2- we loop over this (off-shell case)
- phase space generation- we generate final states using phase space generator (2 random numbers needed).
  We can control the random nbs given to fphase: e.g. random nbs from RANLUX or from integration package
  We can NOT control the random nbs given to TGenPhaseSpace: it uses internal random nb generator
- we calculate matrix element from a known formulas
- we integrate ME with correct weights using integration package:
  4 DIM integral for code with TGenPhaseSpace,
  4 or 6 DIM integral for code with fphase (2 random nbs from RANLUX or integration package)

Problems!

These codes give different results!
The problems always started when I wanted to included ME.

Phase Space Generation

Simple code: K0 decay into two particles with masses `masses[2] = { 0.4, 0.04}`,
in CM (`K0(0.0, 0.0, 0.0,Sqrt( 0.498**2))`)
and in LAB (`K0(0.0, 0.0, 1.0,Sqrt(1**2 + 0.498**2))`)

	TGenPhaseSpace	fphase
CM:		
LAB:		

TGenPhaseSpace and fphase give the same results

http://physik.uibk.ac.at/hephy/praktikum/K0/K0Decay.C

Integration package

easy function to integrate: $\int_0^1 x_0 x_1 x_2 x_3 dx_0 dx_1 dx_2 dx_3 = \frac{1}{16} \simeq 0.06250$

	BASES	Vegas
nb of calls	20000	20000
nb of iterations	100	default $= 5$
accuracy	0.1	default
integral	$0.0624685 + -5.76404e-05$	$0.0624979 + -2.88014e-05$

BASES and Vegas give the same results

Building the code...

Now ready to add more steps to the codes:

- initial particles: two protons
  $p1 = (0, 0, pz, pz)$, $p2 = (0, 0, -pz, pz)$
  pz is fixed
- 4 random numbers to generate initial partons,
  $p_{ini1} = (\sqrt{kt_1}cos(2.\pi x_2), \sqrt{kt_1}sin(2.\pi x_2), x_0 p1.Pz(), x_0 p1.E())$,
  $p_{ini2} = (\sqrt{kt_2}cos(2.\pi x_3), \sqrt{kt_2}sin(2.\pi x_3), x_1 p2.Pz(), x_1 p2.E())$
  $kt_1 = 0$, $kt_2 = 1000$-fixed at this stage
- phase space generation- generate final states using phase space generator
- calculate matrix element from a known formulas, but it can be also put to 1.
- integrate ME with correct weights using integration package

4 codes

4 codes created:

1. fphase Vegas
2. fphase BASES
3. TGenPhaseSpace Vegas
4. TGenPhaseSpace BASES

ME=1

function to integrate: ME * weight
ME = 1
nb of calls: 20000
nb of iterations: 100
accuracy: 0.01

TGenPhaseSpace Vegas	1.56254 +− 0.00072
TGenPhaseSpace BASES	1.56188 +− 0.00015
fphase Vegas 4 DIM	1.53996 +− 0.00080
fphase BASES 4 DIM	1.53953 +− 0.00015
fphase Vegas 6 DIM	1.53829 +− 0.0011
fphase BASES 6 DIM	1.53972 +− 0.00015

Very similar results, there is also no difference if I integrate 6 or 4 DIM.

ME= $\widehat{s}$

function to integrate: $\widehat{s} * weight$
nb of calls: 20000
nb of iterations: 100
accuracy: 0.01

```
SumC=sh;

   if (SumC != SumC){SumC=0.;}
   if (SumC<0){SumC=0;}

   double ME=SumC;
   return ME;
```

code	$\widehat{s}$ from final 4momenta	$\widehat{s}$ from initial 4momenta
TGenPhaseSpace Vegas	6284000 +− 4200	6284000 +− 4200
TGenPhaseSpace BASES	6281280 +−630	6281280 +− 630
fphase Vegas 4 DIM	6271000 +− 4200	6271000 +− 4200
fphase BASES 4 DIM	6268400+− 620	6268400 +−620
fphase Vegas 6 DIM	6270000 +− 12000	6270000 +− 12000
fphase BASES 6 DIM	6267380 +− 790	6267380 +− 790

$\widehat{s}$ from initial 4momenta and $\widehat{s}$ from final 4momenta: the same results

Different codes: Very similar results, there is also no difference if I integrate 6 or 4 DIM.

$$ME = \frac{1}{1-cos(\theta_{final})}$$

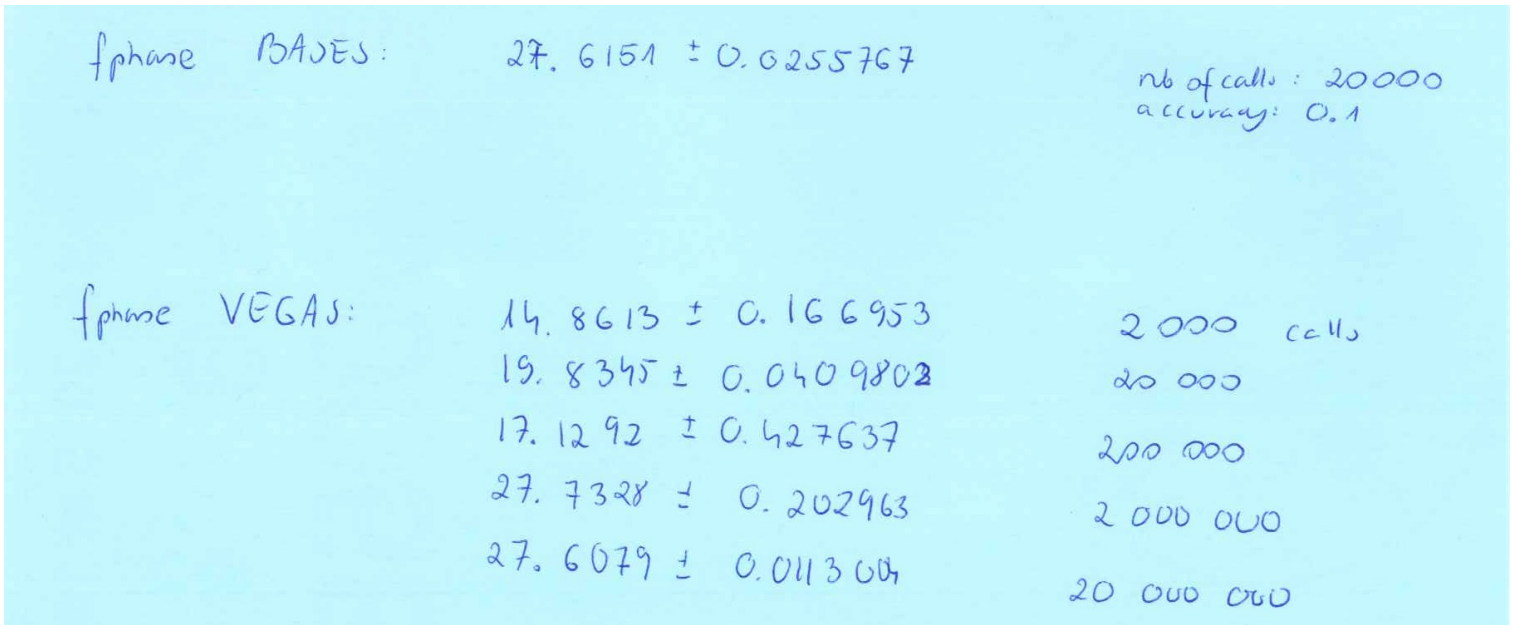
function to integrate: $\frac{1}{1-cos(\theta_{final})} * weight$
nb of calls: 20000
nb of iterations: 100
accuracy: 0.01

TGenPhaseSpace Vegas	35.66 +− 0.79
TGenPhaseSpace BASES	31.39 +− 0.47
fphase Vegas 4 DIM	27.18 +− 0.76
fphase BASES 4 DIM	25.74+−0.36
fphase Vegas 6 DIM	33.29 +− 0.40
fphase BASES 6 DIM	34.07+−0.39

Different results!

Divergent function

function to integrate: $\int_{1E-12}^{1} \frac{1}{x} dx = 27.631$



Nb of calls must be increased!

ME= $\frac{1}{1-cos(\theta_{final})}$, increase nb of calls

function to integrate: $\frac{1}{1-cos(\theta_{final})} * weight$
nb of calls: 200000
nb of iterations: 100
accuracy: 0.01

TGenPhaseSpace Vegas	42.7 +− 1.6
TGenPhaseSpace BASES	37.17 +− 0.40
fphase Vegas 4 DIM	32.57 +− 0.49
fphase BASES 4 DIM	33.72 +− 0.33
fphase Vegas 6 DIM	39.76 +− 0.39
fphase BASES 6 DIM	40.42 +− 0.22

Different results! And the results gets bigger with increasing nb of calls!

ME= $\frac{1}{1-cos(\theta_{final})}$, CUTS!

function to integrate: $\frac{1}{1-cos(\theta_{final})} * weight$

nb of calls: 20000

nb of iterations: 100

accuracy: 0.01

```
double sigma=0.;
if ( cos(p_pfin1.Theta()) <= 0.9 ){  sigma = weight*1./(1.-cos(p_pfin1.Theta()));}

if ( cos(p_pfin1.Theta()) >0.9 ){  sigma = 0.;}

if (sigma != sigma ) {sigma=0.;}

   return sigma;
```

TGenPhaseSpace Vegas	2.345 +− 0.019
TGenPhaseSpace BASES	2.3377 +−0.0018
fphase Vegas 4 DIM	2.307 +− 0.018
fphase BASES 4 DIM	2.3128 +− 0.0018
fphase Vegas 6 DIM	2.314 +− 0.018
fphase BASES 6 DIM	2.3099 +−0.0012

More similar results, TGenPhaseSpace and fphase a little bit different

Results similar for 400000 and 1000 nb of calls (backup)

ME=on-shell limit, 20000 calls

function to integrate: $ME * weight$
`Me= SumC=-(th/sh + sh/th + 2.*m2*uh/(sh*th))/3.` on shell limit
nb of calls: 20000
nb of iterations: 100
check: there are no events with `abs(sh), abs(th), abs(sh*th)<1E-12` accuracy: 0.01

code	$\widehat{s}$ from final states	$\widehat{s}$ from initial states
TGenPhaseSpace Vegas	5.3 +− 1.1	5.19 +− 0.74
TGenPhaseSpace BASES	4.855 +− 0.069	4.263+−0.064
fphase Vegas 4 DIM	4.81 +− 0.15	4.85 +− 0.14
fphase BASES 4 DIM	4.872 +− 0.071	4.683 +− 0.060
fphase Vegas 6 DIM	8.42 +− 0.11	8.52 +− 0.27
fphase BASES 6 DIM	8.191 +− 0.075	8.592 +− 0.067

only $\widehat{s}$ from final states: For 4DIM results the same but different than for 6 DIM

For a given code some of the results for $\widehat{s}$ from initial states different than for $\widehat{s}$ from final states

ME=on-shell limit, 2000000 calls

function to integrate: $ME * weight$
`Me= SumC=-(th/sh + sh/th + 2.*m2*uh/(sh*th))/3.` on shell limit
nb of calls: 2000000
nb of iterations: 100
check: there are no events with `abs(sh), abs(th), abs(sh*th)<1E-12` accuracy: 0.01

code	$\widehat{s}$ from final states	$\widehat{s}$ from inital states
TGenPhaseSpace Vegas	7.236 +− 0.096	7.18 +− 0.12
TGenPhaseSpace BASES	6.967 +−0.068	7.300+−0.061
fphase Vegas 4 DIM	7.07 +− 0.11	7.24 +− 0.13
fphase BASES 4 DIM	6.925+-0.067	6.981 +−0.068
fphase Vegas 6 DIM	10.47 +− 0.10	10.11 +− 0.10
fphase BASES 6 DIM	10.165 +− 0.056	10.497+−0.062

Different results for different versions of the code

and bigger than for 20000 calls!

Summary

- TGenPhaseSpace and fphase give the same results in easy code with K0 decay
- Vegas and BASES give the same result when I integrate function like $x_1 x_2 x_3 x_4$
- Vegas and BASES give the same result when I integrate divergent function like $\int_{1E-12}^{1} \frac{1}{x} dx$, but I need very big nb of calls for Vegas
- 4 codes gives the same result for integrating over weight, $\widehat{s}$ (from initial and final momenta), $\frac{1}{1-cos(\theta_{final})}$ (with cuts), for 4 and 6 DIM

BUT

- 4 codes gives different result for on-shell limit, for 4 and 6 DIM, different nb of calls, depending on $\widehat{s}$ (if calculated from initial or final momenta) ...
- nb of calls increase $\rightarrow$ result increase

It looks like the problems starts when the integral is divergent
How can I know which result is correct?

Summary

- TGenPhaseSpace and fphase give the same results in easy code with K0 decay
- Vegas and BASES give the same result when I integrate function like $x_1 x_2 x_3 x_4$
- Vegas and BASES give the same result when I integrate divergent function like $\int_{1E-12}^{1} \frac{1}{x} dx$, but I need very big nb of calls for Vegas
- 4 codes gives the same result for integrating over weight, $\widehat{s}$ (from initial and final momenta), $\frac{1}{1-cos(\theta_{final})}$ (with cuts), for 4 and 6 DIM

BUT

- 4 codes gives different result for on-shell limit, for 4 and 6 DIM, different nb of calls, depending on $\widehat{s}$ (if calculated from initial or final momenta) ...
- nb of calls increase $\rightarrow$ result increase

It looks like the problems starts when the integral is divergent
How can I know which result is correct?

Please, help!

Thank you!

differences between TGenPhaseSpace and fphase

fphase:

- We can control the random nbs given to fphase: e.g. random nbs from RANLUX or from integration package
- The way it is called:
  `CASCADE::fphase( np, et, xm, pcm, weight) ;`
  `np`- nb of generated particles, `et`- $\sqrt{\hat{s}}$, `xm`-array with masses of generated particles, `pcm`-array of the returned 4-momenta, e.g. `double pcm[8]`, `weight`- returned weight,
- generates particles in CM $\rightarrow$ in our case boost to LAB needed
- weight: continuos distribution between 0 and $\frac{\pi}{2}$

differences between TGenPhaseSpace and fphase

TGenPhaseSpace:

- We can NOT control the random nbs given to TGenPhaseSpace: it uses internal random nb generator
- The way it is called:

```
TGenPhaseSpace event;
event.SetDecay(p_parton_total, 2., mass);
Double_t weight = event.Generate();
TLorentzVector *p_Nf1 = event.GetDecay(0);
TLorentzVector *p_Nf2 = event.GetDecay(1);
p_pfin1 = *p_Nf1; p_pfin2 = *p_Nf2;
```

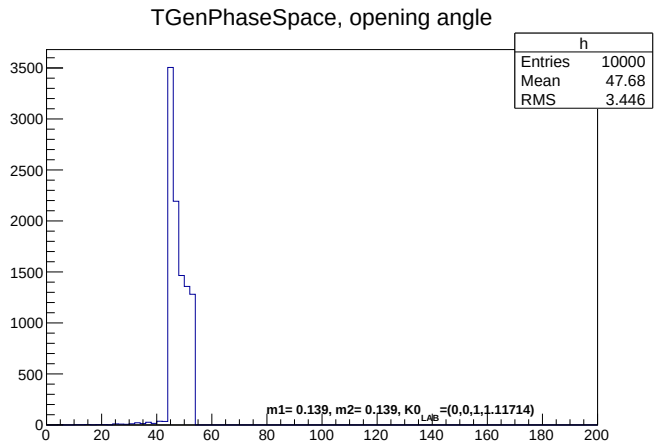
  `p_parton_total`- sum of 4momenta of incoming particles
- generetes particles in the frame p_parton_total was calculated (in our case LAB).
- weight = 0 or 1. To compensate with fphase: $weight = weight * 3.14/2$

Phase Space Generation - 2 identical particles in LAB

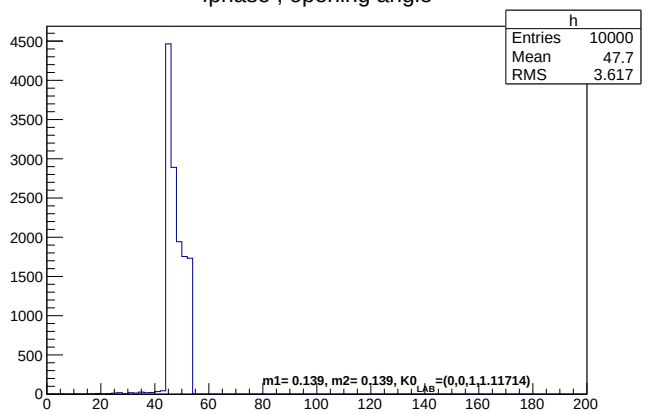
Simple code: K0 decay into two particles with masses `masses[2] = { 0.139, 0.139}`,
in LAB (`K0(0.0, 0.0, 1.0,Sqrt(1**2 + 0.498**2))`)

LAB:

TGenPhaseSpace



fphase



TGenPhaseSpace and fphase give the same results

Example with boost: how to perform a boost of K0 4-momentum from LAB to CM and back:

```
TLorentzVector K0(0.0, 0.0, 1.0, sqrt(1 + 0.498 * 0.498)); //LAB
TLorentzVector forBoostBack = K0; // save a vector to boost back to lab
frame
K0.Boost( -K0.BoostVector() ); // boost to K0 rest frame
forBoostBack.SetPxPyPzE(-forBoostBack[0],-forBoostBack[1],-forBoostBack[2],
forBoostBack[3]);//boost Back to lab frame by reversing momentum of
"forBoostBack" vector
K0.Boost( -forBoostBack.BoostVector() ); //K0 back to LAB
```

How to call integration package

Integrate easy function with BASES and Vegas:

```
double fun (double *x){ (BASES)
or
 double fun (double *x, size_t dim, void *params){ (Vegas)
double result=x[0]*x[1]*x[2]*x[3];
return result;}
```

BASES

```
int ii;
CASCADE::fbsinit() ;
bparm1.ndimb=4;
bparm1.nwildb=4;
bparm1.ncallb=20000;
bparm2.itmx1b=30;
bparm2.acc1b= 0.5;
bparm2.itmx2b = 100;
bparm2.acc2b  = 0.1;
for (ii=0;ii<4;ii++) { bparm1.ig[ii]=1; bparm1.xlb[ii]=1e-12; bparm1.xub[ii]=1.;}
double sigma, err, Sigma_test;
double ctime;
double ffxn;
int it1, it2;

CASCADE::fbases(fun, sigma, err, ctime, it1, it2 ) ;
cout<<"sigma ="<<sigma<<"+-"<<err<<endl;
```

Vegas

```
double xl[4]={1E-12, 1E-12,1E-12 ,1E-12  };
double xu[4]={1., 1., 1., 1., };
const gsl_rng_type *T;
gsl_rng *r;
gsl_rng_env_setup ();
T = gsl_rng_default;
r = gsl_rng_alloc (T);
gsl_monte_function G = {&fun, 4.,0};
gsl_monte_vegas_state *s=gsl_monte_vegas_alloc (4);
gsl_monte_vegas_init (s) ;
gsl_monte_vegas_integrate (&G , xl, xu, 4., 20000., r, s, &sigma, &err );
gsl_monte_vegas_free (s);


cout<<"sigma ="<<sigma<<"+-"<<err<<endl;
```

integrate over constant function

function to integrate: 1
nb of calls: 20000
nb of iterations: 100
accuracy: 0.01

TGenPhaseSpace Vegas	0.99525 +− 0.0004555
TGenPhaseSpace BASES	0.994862 +− 9.92992e-05
fphase Vegas 4 DIM	0.99525 +− 0.0004555
fphase BASES 4 DIM	0.994862 +− 9.92992e-05
fphase Vegas 6 DIM	0.99456787 +− 0.0005538
fphase BASES 6 DIM	0.995073 +− 9.84061e-05

Very similar results, there is also no difference if I integrate 6 or 4 DIM.

ME= $\hat{s}$ calculated from initial 4momenta, more cuts

function to integrate: $\hat{s} * weight$
nb of calls: 20000
nb of iterations: 100
accuracy: 0.01

```
 if (abs(sh)<m2){SumC=0.;}
 if (abs(th)<1E-11){SumC=0.;}
 if (abs(sh*th)<1E-11){SumC=0.;}


if (SumC != SumC){SumC=0.;}
if (SumC<0){SumC=0;}

double pt2f1=p_NF1.Px()*p_NF1.Px()+p_NF1.Py()*p_NF1.Py();
double pt2f2=p_NF2.Px()*p_NF2.Px()+p_NF2.Py()*p_NF2.Py();
if (pt2f1<pt2cut){SumC=0.;}
if (pt2f2<pt2cut){SumC=0.;}



double ME=SumC;

return ME;
```

TGenPhaseSpace Vegas	6284017.5 +−4200
TGenPhaseSpace BASES	6281277.9 +− 627
fphase Vegas 4 DIM	6271019.9 +− 4200
fphase BASES 4 DIM	6268403.2 +− 621
fphase Vegas 6 DIM	6270055.9 +− 11630
fphase BASES 6 DIM	6267378.1 +− 790

Very similar results, there is also no difference if I integrate 6 or 4 DIM.

ME$= \hat{s}$ calculated from final 4momenta, more cuts

function to integrate: $\hat{s} * weight$
nb of calls: 20000
nb of iterations: 100
accuracy: 0.01

```
 if (abs(sh)<m2){SumC=0.;}
 if (abs(th)<1E-11){SumC=0.;}
 if (abs(sh*th)<1E-11){SumC=0.;}


if (SumC != SumC){SumC=0.;}
if (SumC<0){SumC=0;}

double pt2f1=p_NF1.Px()*p_NF1.Px()+p_NF1.Py()*p_NF1.Py();
double pt2f2=p_NF2.Px()*p_NF2.Px()+p_NF2.Py()*p_NF2.Py();
if (pt2f1<pt2cut){SumC=0.;}
if (pt2f2<pt2cut){SumC=0.;}



double ME=SumC;

return ME;
```

TGenPhaseSpace Vegas	6284017.5 +− 4200
TGenPhaseSpace BASES	6281277.9 +− 626
fphase Vegas 4 DIM	6271019.9 +− 4200
fphase BASES 4 DIM	6267379.8 +− 790
fphase Vegas 6 DIM	6270055.9 +− 11630
fphase BASES 6 DIM	6267378.1 +− 790

Very similar results, there is also no difference if I integrate 6 or 4 DIM.

ME= $\frac{1}{1-cos(\theta_{final})}$, CUTS!, increase nb of calls

function to integrate: $\frac{1}{1-cos(\theta_{final})} * weight$
nb of calls: 400000
nb of iterations: 100
accuracy: 0.01

TGenPhaseSpace Vegas	2.3323406 +− 0.004238
TGenPhaseSpace BASES	2.33764+−0.000399752
fphase Vegas 4 DIM	2.3097765 +− 0.004202
fphase BASES 4 DIM	2.31254+−0.000394527
fphase Vegas 6 DIM	2.3158837 +− 0.00344
fphase BASES 6 DIM	2.31266+−0.000262554

ME= $\frac{1}{1-cos(\theta_{final})}$, CUTS!, decrease nb of calls

function to integrate: $\frac{1}{1-cos(\theta_{final})} * weight$
nb of calls: 1000
nb of iterations: 100
accuracy: 0.01

TGenPhaseSpace Vegas	2.334236 +− 0.02299
TGenPhaseSpace BASES	2.32738 +− 0.00964999
fphase Vegas 4 DIM	2.2584442 +− 0.02204
fphase BASES 4 DIM	2.31916 +− 0.00955651
fphase Vegas 6 DIM	2.3044853 +− 0.02279
fphase BASES 6 DIM	2.30684 +− 0.00559735