SLAC – PUB – 4549
March 1988
(T/E)

The Noodle Method*

CARL JUNG-CHOON IM

Stanford Linear Accelerator Center
Stanford University, Stanford, California, 94309

and

Department of Physics
Stanford University, Stanford, CA 94309

ABSTRACT

The Noodle method allows a fast and efficient sampling of the region of integration during Monte Carlo calculations. The method works by automatically optimizing the weight distribution and by replacing analytic results by exact numerical calculations. The method allows integrations of certain functions that have been heretofore considered infeasible.

Submitted to *Journal of Computational Physics*

*Work supported in part by the Department of Energy, contract DE–AC03–76SF00515

1. *Introduction*

The traditional Monte Carlo method[1] is unacceptably inefficient for certain commonly occurring integrands and requires various analytic formulae in addition to the integrand. The Noodle method eliminates these complications. The remainder of this section contains a critical description of the traditional Monte Carlo method, section 2 describes the Noodle Method, section 3 contains a detailed example of an application of the Noodle method, namely a numerical computation of a physical cross-section, and section 4 compares the Noodle method to the traditional Monte Carlo method.

Consider the integral

$$I = \int_0^1 dx \int_0^1 dy \int_0^1 dz \; f(x,y,z) \tag{1}$$

The Monte Carlo method is based on the following equation:

$$\frac{Vol(S)}{N} \sum_{i=1}^{i=N} f(x_i) = \int_S f \pm \sqrt{\frac{Var(f)}{N}} \tag{2}$$

where each x_i is a randomly and uniformly chosen point in the space S.

Consequently, to estimate I accurately for small values of N, I is first rewritten as

$$\int_0^1 dx \int_0^1 dy \int_0^{G(x,y)} dG \; \frac{f(x,y,z(G))}{g(x,y,z(G))}, \; where \;\; G(x,y) = \int_0^1 dz \; g(x,y,z) \tag{3}$$

with the corresponding Monte Carlo sum

$$\frac{Vol(\tilde{S})}{N} \sum_{i=1}^{i=N} \frac{f(x_i,y_i,z(G_i))}{g(x_i,y_i,z(G_i))} \pm \sqrt{\frac{Var(\frac{f}{g})}{N}} \tag{4}$$

where each (x_i, y_i, G_i) is chosen randomly and uniformly from $\tilde{S}(figure\ 1)$. Then

g is chosen so that $Var(\frac{f}{g})$ is small.

For this method to work, g has to satisfy two mutually antagonistic criteria. It is crucial that g has a rather complicated form so that it approximates f well. Yet at the same time, g has to have a simple form so that it can be integrated analytically. Choosing such a g is an art form and is the complication with the traditional Monte Carlo method that impedes a person from applying the method.

Once g has been chosen, the Monte Carlo sum is performed by first generating a $(x_i, y_i, z(G_i))$ triplet, then by evaluating the summand. The usual algorithm for generating (x_i, y_i, G_i)s is to generate a random point inside the smallest box that contains $\tilde{S}$ and reject the point if it lies outside $\tilde{S}$. This algorithm becomes inefficient when $\tilde{S}$ has "spikes" (*figure* 2). Moreover, it is not easy to invert

$$G_i = \int_0^{z(G_i)} dz\ g(x,y,z) \tag{5}$$

to evaluate $z(G_i)$.

2. The Noodle Method

In the Noodle method, S is decomposed into small sets over each of which f is roughly constant. Then for each set, g is defined as constant and equals f at least one point in the set. One such choice of g for a simple decomposition of S is

$$g(x,y,z) = f(\frac{i}{M}, \frac{j}{M}, 0); \frac{i}{M} < x < \frac{i+1}{M}, \frac{j}{M} < y < \frac{j+1}{M}, 0 < z < 1 \tag{6}$$

for some M and all $0 \leq i,j < M$. The corresponding $\tilde{S}$ is shown on *figure* 3a. The decomposition of S induces a decomposition of $\tilde{S}$ via the change of

coordinates z into G:

$$\frac{i}{M} < x < \frac{i+1}{M}, \frac{j}{M} < y < \frac{j+1}{M}, 0 < G < G_{i,j}; G_{i,j} = \frac{1}{M^2}\, g(\frac{i}{M}, \frac{j}{M}).$$

Each of these sets is called a noodle. These noodles are then arbitrarily ordered from 1 up to the total number of noodles.

By construction, g approximates f, and, since g is constant over each noodle, g can be integrated exactly. This solves the problem of finding a suitable g.

Moreover, defining g locally constant allows a fast and efficient generation of the $(x_i, y_i, z(G_i))$ triplets. This is possible because of the simple relation that **the uniform distribution in $\tilde{S}$ is equivalent to the distribution g in S.** Consider $\rho(0 < x - x_0 < \delta x, 0 < y - y_0 < \delta y, 0 < G - G_0 < \delta G) = \delta x \delta y \delta G$, which is the uniform distribution in $\tilde{S}$. Since $G - G_0 = g(x, y, z_0)(z - z_0)$, we have $\rho(0 < x - x_0 < \delta x, 0 < y - y_0 < \delta y, 0 < z - z_0 < \frac{\delta G}{g}) = \delta x \delta y \delta G$. Since δG is small but otherwise arbitrary, replacing δG by $g\delta z$ yields the desired result. Using this relationship, we directly generate (x_i, y_i, z_i) with the distribution g thus eliminating the need for complicated inversion of functions. First, we arrange the noodles from end to end to make a long noodle by defining an array as follows:

NOODLE(0)= 0

NOODLE(i)= NOODLE($i-1$)+volume of the i^{th} noodle

the length of the long noodle = NOODLE(number of the noodles)

This is shown in *figure* 3b. Finally, in order to generate a (x_i, y_i, z_i), generate a random number between 0 and the length of the long noodle. This picks out a noodle uniformly in $\tilde{S}$ and a set according to the distribution g in S. Once a noodle is chosen, since g is constant within the set, any point within the set is

equally likely to be chosen. Accordingly, we simply pick a random point (x, y, z) within the set. It is fast because each generation of (x, y, z) involves only a binary search algorithm for choosing a box and three random number generations. It is efficient in the sense that no point that is generated is rejected.

For this method to work better than the traditional Monte Carlo method, it is crucial to partition S into small sets so that f is indeed approximately constant over each set. One way to achieve this is to choose M large enough so that, over S, $\frac{f}{g}$ takes values in a small interval around 1, say $(1-\epsilon, 1+\epsilon)$. Consequently, in our example, $\frac{1}{M}$ is the maximum scale in $x-y$ space over which the fractional change in f is less than ϵ. Thus, if the graph of f has a tall but narrow spike at a point but is otherwise flat, the necessary value of M dictated by the width of the spike is clearly an overestimate for the rest of the region of integration.

In the spirit of replacing nontrivial analytical calculations by exact numerical calculations, it would be pleasing if there existed a numerical algorithm for finding an acceptable partition of S. The following algorithm is only a partial solution to this problem that presumes a knowledge of the general profile of the integrand. The completely general numerical solution is usually unnecessary and seems to be very complicated. It should be mentioned that there are integration routines, VEGAS for example, that solve this problem in general, but their generality is usually compensated by their slow speed.

procedure bakery;

index:= 1;

while (index $\leq$ length of the noodle) do

begin

```
    while is_wild(index^th noodle) do
    begin
      partition the noodle into smaller noodles;
      let one of the noodles be the index^th noodle;
      put the rest of the new noodles at the end of the ordering;
      increase the number of noodles by the number of new noodles;
    end;
    noodle(index):= noodle(index-1)+ volume of the index^th noodle;
    index:= index+1;
  end;
end;
function is_wild(noodle):boolean;
  is_wild:= (f fluctuates unacceptably inside the box);
end;
```

The user has to define the *wild* function. Because the Noodle method automatically samples S, a list of points in S that are associated with unacceptable weights can readily be generated. This furnishes two pieces of information: Unacceptable weights indicate that the *wild* function is insufficiently defined, and the list indicate the regions of S over which the integrand has unexpected structures. Using this information, the *wild* function can be redefined so that the unacceptable weights do not occur.

3. *An Example*

We illustrate the method by using the noodle method to compute the $e^+e^- \to \mu^+\mu^-$ cross section after the effects of the initial state bremmstrahlung have been taken into account. The cross section is given by[2]

$$\sigma = \int_{x_{cut}}^{1} dx_+ \int_{x_{cut}}^{1} dx_- \int d\Omega \; D(x_+)D(x_-)\frac{d\sigma_0}{d\Omega_{CM}}(\theta_{cm}, x_+x_-S)\Theta(p^{lab}_{\mu^+}, p^{lab}_{\mu^-}) \qquad (7)$$

where σ_0 is the uncorrected cross-section, and $\Theta(x) = 1$ if x is inside the phase space cut and 0, otherwise. For the sake of simplicity, we consider the case of no phase space cut.

In order to integrate this numerically, it is necessary to regulate the poles of D by a suitable change of variables. The resulting integral is

$$\int_0^{P_{cut}} dP_+ \int_0^{P_{cut}} dP_- \int_0^{2\pi} d\phi \int_{-1}^{1} dcos\theta \; w_d(P_+)w_d(P_-)\frac{d\sigma}{d\Omega}(cos\theta, x_+(P_+)x_-(P_-)S) \qquad (8)$$

where $P_+ = (1 - x_+)^{\frac{\beta}{2}}$, the singular part of the integrated electron structure function. We shall evaluate this integral for $\sqrt{S} = 94Gev$. The corresponding $\tilde{S}$ is shown on *figure* 4a. Since the Ds are always approximately 1, the only *wild* regions are when $\sqrt{x_+x_-S}$ is near the Z-pole or the *photon*-pole. Moreover, the noodles should not be too big. Hence, it is natural to cut S evenly in the x_+x_--plane and to consider the two extreme values of x_-x_+S to decide whether a particular noodle is wild.

Following the Noodle method, at the beginning of the program, we make noodles:

procedure bakery;

arrays $x_-^{min}, x_-^{max}, x_+^{min}, x_+^{max}$,NOODLE

number of noodles:= 1;

index:= 1;

x_-^{min}(index):= x_{cut};

x_+^{min}(index):= x_{cut};

x_-^{max}(index):= 1;

x_+^{max}(index):= 1;

while (index $\leq$ number of the noodles) do

while is_wild($x_-^{min}(index)x_+^{min}(index)S, x_-^{max}(index)x_+^{max}(index)S$) do

begin

x_-^{min}(number of noodles+1):= x_-^{min}(index);

x_+^{max}(number of noodles+1):= x_+^{max}(index);

x_-^{max}(number of noodles+1):= $\frac{1}{2}(x_-^{max}$(index)+x_-^{min}(index));

x_+^{min}(number of noodles+1):= $\frac{1}{2}(x_+^{max}$(index)+x_+^{min}(index));

x_-^{min}(number of noodles+2):= x_-^{max}(number of noodles+1);

x_+^{max}(number of noodles+2):= x_+^{max}(index);

x_-^{max}(number of noodles+2):= x_-^{max}(index);

x_+^{min}(number of noodles+2):= x_+^{min}(number of noodles+1);

x_-^{min}(number of noodles+3):= x_-^{min}(number of noodles+2);

x_+^{max}(number of noodles+3):= x_+^{min}(number of noodles+2);

x_-^{max}(number of noodles+3):= x_-^{max}(number of noodles+2);

x_+^{min}(number of noodles+3):= x_+^{max}(number of noodles+2);

x_+^{max}(index):= x_+^{max}(number of noodles+3);

x_-^{max}(index):= x_-^{max}(number of noodles+1);

number of noodles:= number of noodles + 3;

end;

$x_+ := \frac{1}{2}(x_+^{min} + x_{max}^+)$;

$x_- := \frac{1}{2}(x_-^{min} + x_{max}^-)$;

$P_+ := (1 - x_+)^{\frac{\beta}{2}}$;

$P_- := (1 - x_-)^{\frac{\beta}{2}}$;

$\Delta P_+ := (1 - x_{min}^+)^{\frac{\beta}{2}} - (1 - x_{max}^+)^{\frac{\beta}{2}}$;

$\Delta P_- := (1 - x_{min}^-)^{\frac{\beta}{2}} - (1 - x_{max}^-)^{\frac{\beta}{2}}$;

NOODLE(index):=

NOODLE(index-1)+$w_d(P_-)w_d(P_+)\frac{d\sigma}{d\Omega}(cos\theta, x_+x_-S)\Delta P_+ \ \Delta P_- \ \Delta cos\theta$;

end;

function is_wild(S_{min}, S_{max});

wild if $S_{min} < M_Z^2 + 9\Gamma_Z M_Z$ and $M_Z^2 - 9\Gamma_Z M_Z < S_{max}$;

wild if $S_{min} < S_\gamma$;

wild if $\sqrt{S_{max}} - \sqrt{S_{min}} \geq \Delta E$;

otherwise not wild;

end;

The sampling of the integration region proceeds in two steps. First, we pick a noodle with the relative probability of NOODLE(index). Second, we pick a random point $(P_i^+, P_i^-, cos\theta_i)$ within the noodle:

procedure generate weights;

pick a random number r between 0 and the length of the noodle;

find l such that NOODLE$(l-1) < r \leq$ NOODLE(l);

$\Delta P^+ := (1 - x_{min}^+(l))^{\frac{\beta}{2}} - (1 - x_{max}^+(l))^{\frac{\beta}{2}}$;

$\Delta P^- := (1 - x^-_{min}(l))^{\frac{\beta}{2}} - (1 - x^-_{max}(l))^{\frac{\beta}{2}}$;

r_1 and r_2 are random numbers between 0 and 1;

$P_+ := (1 - x^+_{max})^{\frac{\beta}{2}} + r_1 \Delta P^+$;

$P_- := (1 - x^-_{max})^{\frac{\beta}{2}} + r_2 \Delta P^-$;

$x_+ := 1 - P_+^{\frac{2}{\beta}}$;

$x_- := 1 - P_-^{\frac{2}{\beta}}$;

weight:= $\frac{w_d(P_-)w_d(P_+)\frac{d\sigma}{d\Omega}(cos\theta_l, x_+x_-S)\Delta P^+\Delta P^-\Delta cos\theta}{NOODLE(l)-NOODLE(l-1)}$;

end;

4. *Implementation and Comparison*

The simplicity and the efficiency of the Noodle method makes it trivial to write programs to integrate quite complicated functions and to investigate the properties of the integral in a short period of time. An implementation of the preceding algorithm appears in the *Appendix*. For 10^4 Monte Carlo samples, we reproduce the results obtained by Berends et al. at the Z-peak.[3] The resulting weight distribution is shown on *figure* 5. The results of a more complete calculation that contains the initial state radiation, the weak vertex functions, and the vector boson self-energies appears elsewhere[4] .

As discussed earlier, the Noodle method is a simplification of the Monte Carlo method in two aspects. First, it is no longer necessary to supply the approximate integrands to perform integrations using the Monte Carlo method. Second, there is no need for complicated inversion formulae. More importantly, however, the Noodle method is an improvement over the Monte Carlo method in that it can perform certain integrations that are not feasible to do by the traditional Monte Carlo method. In computations of e^+e^- to f^+f^- at energies around 1Tev, the

shape of S becomes quite singular as shown on *figures* 4b and 4c. The ratio of the volume of the smallest box that contains S to the volume of S is approximately 10^4 and in order to find the value of σ_{lab} to 1 percent, the number of samples required in the traditional Monte Carlo method is 10^8, compared to only 10^4 in the Noodle method. The associated weight distribution is shown on *figure* 6.

5. *Acknowledgement*

The author thanks Group H at SLAC/Mark II for its support and Bryan Lynn, Patricia Rankin, and especially Dallas Kennedy for valuable comments and discussions.

REFERENCES

1. F. James, Rept. Prog. Phys. 43:1145, 1980
2. O. Nicrosini and L. Trentadue, Phys. Lett. 196B:551, 1987
3. Berends, Burgers, van Neerman, Phys. Lett. 185B:395,1987
4. Kennedy, Lynn, Stuart, Im, SLAC–PUB–4128

6. Captions

Figure 1

The change of variables from z to G improve the convergence rate of the Monte Carlo sum.

Figure 2

The conventional Monte Carlo Method becomes inefficient for certain shapes of the space of integration, such as the one shown here.

Figure 3

In the Noodle method, the integrand is approximated by step functions. The resulting $\tilde{S}$, corresponding to the same integrand as in figure 1, is shown on the left. The blocks are then arranged into the shape of a noodle, as shown on the right.

Figure 4

The graphs of $\tilde{S}$ at $94GeV$ (a), $110GeV$ (b), $210GeV$ (c) for integrand appearing in equation 8 are shown here. Each group of blocks of the same height, the group of blocks appearing at the $(P_+, P_-) = (1,1)$ corner on figure 4a, for example, corresponds to a single noodle. This correspondence is explicitly illustrated on figure 4c. In order to show detail, only the portion of P_+ P_- in which both coordinates are greater than 0.8 is shown, and the photon pole, appearing at $(P_+, P_-) = (1,1)$ corner and the Z^0 pole, the other peak in each graph, are truncated.

Figure 5

The distribution of the ratio between the integrand in equation 8 and the approximate integrand constructed using noodles at $\sqrt{S} = 94GeV$.

Figure 6

The distribution of the ratio between the integrand in equation 8 and the approximate integrand constructed using noodles at $\sqrt{S} = 1TeV$.

7. Appendix

```
C-----------------------------------------------------------------------C
C     THE NOODLE MONTE CARLO - EXPONENTIATED INITIAL STATE BREMS-       C
C     STRAHLUNG FOR THE Z0 RESONANCE                                    C
C     E+E- STRUCTURE FUNCTIONS BY O. NICROSINI & L. TRENTADUE           C
C     MONTE CARLO METHOD AND CODING BY J. IM                            C
C     REVISIONS AND COMMENTS BY D. KENNEDY                              C
C     SLAC SLC/MARK II - 10/30/87                                       C
C     NEW IMPROVED VERSION - 2/19/88                                    C
C                                                                       C
C                                                                       C
C     CONTAINS SUBROUTINES THAT CUT NOODLES ALONG GIVEN XP AND XM       C
C     START WITH ARBITRARY PARTITION OF THE X-SPACE                     C
C     CUT NOODLES ALONG XP AND XM LINES AT UNIFORM INTERVALS            C
C     HAND IT OVER TO WILD FUNCTION                                     C
C     CONTAINS THETA CUT                                                C
C                                                                       C
C-----------------------------------------------------------------------C
      IMPLICIT REAL*8 (A-Z)
      INTEGER I,NEVENT,NN,MM,STAT,FILE,BIGRAD
      DIMENSION MUP(4),MUM(4),CUT(5)
      COMMON/COUPL/CEL,CER,CML,CMR,QE,QM
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL
C-------------------------------------------------------PARAMETERS

C: MONTE CARLO PARAMETERS: NUMBER OF EVENTS, NUMBER OF NOODLES = NN^2
      NEVENT= 20000
      NN= 5
      MM= 5
      CALL RAN11A(37561547)
C: PHYSICS CONSTANTS
      PI= 3.141592D0
      ALPHA= 1.D0/137.03602D0/(1.D0-.06D0)
      SINW2= 0.23D0
      CONST= ALPHA**2/4.D0*389385.7D0
      ME= .000511D0
      MZ= 92.D0
      GAMZ= 2.9D0
C: ELECTRON AND MUON COUPLINGS
      QE= -1.D0
      QM= -1.D0
      I3E= -.5D0
      I3M= -.5D0
      CEL= 2.D0*(I3E-SINW2*QE)
      CER= 2.D0*(-SINW2*QE)
      CML= 2.D0*(I3M-SINW2*QM)
      CMR= 2.D0*(-SINW2*QM)
C: BEAM ENERGY AND PHASE SPACE CUTS
C: ECUT IS A COMMON ENERGY CUT THAT SHOULD BE EQUAL TO OR LESS THAN
C: THE SMALLER OF THE TWO CUT(1) AND CUT(2)
      EB2= 92.D0
      DEB2= 0.5D0
      MAXEB2=91.D0
      ECUT= 5.D0
      SCUT= (45.D0)**2
      SHPOL= 7.D0*MZ*GAMZ+MZ**2
      SLPOL= -7.D0*MZ*GAMZ+MZ**2
```

```
      CUT(1)= 42.4D0
      CUT(2)= 42.4D0
      CUT(3)= 1.D0
      CUT(4)= DCOS(PI*6.07D0/180.D0)
      CUT(5)= 1.D0
      STAT= 1
      FILE= 1
C-----------------------------------------------------------------MAIN
C: LEFT POL, THEN RIGHT POL
      POL= 1.D0
   23 S= EB2**2
C: INITIALIZES AND CONSTRUCTS NOODLE FOR A GIVEN S
      CALL INIT(S)
      RADCUT= DSQRT(S*(1.D0-(BETA/(1.D0+BETA))))
   24 POL= -1.D0*POL
      CALL BAKERY(S,NN,MM,ECUT,APRCRS)
      MAXW= 0.D0
      FBASYM= 0.D0
      LRASYM= 0.D0
      WSUM= 0.D0
      WFSUM= 0.D0
      WBSUM= 0.D0
      WERR= 0.D0
      WFERR= 0.D0
      WBERR= 0.D0
      SIGERR= 0.D0
      BIGRAD= 0
C: GENERATES NEVENT EVENTS
      DO 100 I= 1,NEVENT
         IF (I-10000*(I/10000) .EQ. 0) PRINT 101,I
  101 FORMAT(' ',I10,' EVENTS PROCESSED')
         CALL GENEV(S,SP,MUP,MUM,W,DFLOAT(I)/NEVENT)
         IF (DSQRT(SP) .LT. RADCUT) THEN
            BIGRAD= BIGRAD+1
         ENDIF
         IF (POL .LT. 0.D0) CALL HISTO(W,1.D0,0.D0,8.D0,1)
         IF (POL .GT. 0.D0) CALL HISTO(W,1.D0,0.D0,8.D0,2)
         WSUM= WSUM+W
         SIGERR= SIGERR+W**2
C: BINS WEIGHTS IN WEIGHT DISTRIBUTION HISTOGRAMS AND CHECKS FOR
C: MAXIMUM WEIGHTS
         IF ((STAT .EQ. 1).AND.(W .GT. MAXW)) MAXW= W
         IF (MUM(3).GT.0.D0) THEN
            WFSUM= WFSUM+W
            WFERR= WFERR+W**2
         ELSE
            WBSUM= WBSUM+W
            WBERR= WBERR+W**2
         ENDIF
  100 CONTINUE
C: MONTE CARLO INTEGRATION VOLUME, APR AND EXACT CROSS SECTIONS
      DV= APRCRS/NEVENT
      SIG= WSUM*DV
      ERR= DV*DSQRT(SIGERR-WSUM**2/NEVENT)
C: FORWARD/BACKWARD ASYMMETRY
      FBASYM= (WFSUM-WBSUM)/(WFSUM+WBSUM)
      FBAERR= WBSUM**2*WFERR+WFSUM**2*WBERR
```

```
      FBAERR= DSQRT(FBAERR)/(WFSUM+WBSUM)**2
      IF (POL .LT. 0.D0) THEN
         PRINT 1,EB2,SIG,ERR
         PRINT 2,FBASYM,FBAERR
         IF (STAT .EQ. 1) PRINT 3,WSUM/NEVENT,ERR/DV/NEVENT,MAXW,BIGRAD,
     .                            RADCUT
         IF (FILE .EQ. 1) THEN
            WRITE (24,9) EB2,SIG,ERR
            WRITE (27,9) EB2,FBASYM,FBAERR
         ENDIF
         APPL= APRCRS
         SIGL= SIG
         ERRL= ERR
         WFSUML= WFSUM*APPL
         WBSUML= WBSUM*APPL
         WFERRL= WFERR*APPL**2
         WBERRL= WBERR*APPL**2
         GOTO 24
      ELSE
         PRINT 4,EB2,SIG,ERR
         PRINT 5,FBASYM,FBAERR
         IF (STAT .EQ. 1) PRINT 6,WSUM/NEVENT,ERR/DV/NEVENT,MAXW,BIGRAD,
     .                            RADCUT
         APPR= APRCRS
         SIGR= SIG
         ERRR= ERR
         WFSUMR= WFSUM*APPR
         WBSUMR= WBSUM*APPR
         WFERRR= WFERR*APPR**2
         WBERRR= WBERR*APPR**2
         SIGTOT= 0.5D0*(SIGL+SIGR)
         ERRTOT= 0.5D0*DSQRT(ERRL**2+ERRR**2)
         LRASYM= (SIGL-SIGR)/(SIGL+SIGR)
         LRAERR= DSQRT((SIGR*ERRL)**2+(SIGL*ERRR)**2)/2.D0/SIGTOT**2
         FBATOT= (WFSUML+WFSUMR-WBSUML-WBSUMR)/
     .           (WFSUML+WFSUMR+WBSUML+WBSUMR)
         FBERRT= (WBSUML+WBSUMR)**2*(WFERRL+WFERRR)
         FBERRT= FBERRT+(WFSUML+WFSUMR)**2*(WBERRL+WBERRR)
         FBERRT= DSQRT(FBERRT)/(WFSUML+WFUMR+WBSUML+WBSUMR)**2
         PRINT 7,EB2,SIGTOT,ERRTOT,FBATOT,FBERRT
         PRINT 8,LRASYM,LRAERR
         IF (FILE. EQ. 1) THEN
            WRITE (25,9) EB2,SIGR,ERRR
            WRITE (28,9) EB2,FBASYM,FBAERR
            WRITE (23,9) EB2,SIGTOT,ERRTOT
            WRITE (26,9) EB2,FBATOT,FBERRT
            WRITE (29,9) EB2,LRASYM,LRAERR
         ENDIF
         EB2= EB2+DEB2
      ENDIF
C: OUTPUTS CROSS SECTION, L/R AND F/B ASYMMETRIES, ERRORS
    1 FORMAT(' LEFT: ROOT(S)=',F9.4,'      SIGL: ',F10.8,'+-',F6.4)
    2 FORMAT(' FBASYML: ',F10.8,'+-',F6.4)
    3 FORMAT(' STATISTICS:  SUM(WL)=',F10.4,'   SUM(WL2)=',F10.4,
     .       '   MAX(WL)=',F10.4,/,
     .       ' ',I6,' EVENTS BELOW ',F9.4,' GEV',/)
    4 FORMAT(' RIGHT: ROOT(S)=',F9.4,'      SIGR: ',F10.8,'+-',F6.4)
```

```
    5 FORMAT(' FBASYMR: ',F10.8,'+-',F6.4)
    6 FORMAT(' STATISTICS:  SUM(WR)=',F10.4,'   SUM(WR2)=',F10.4,
     .       '   MAX(WR)=',F10.4,/,
     .       ' ',I6,' EVENTS BELOW ',F9.4,' GEV',/)
    7 FORMAT(' TOTAL: ROOT(S)=',F9.4,'       SIG: ',F10.8,'+-',F6.4,/,
     .       ' FBASYM: ',F10.8,'+-',F6.4)
    8 FORMAT(' LRASYM: ',F10.8,'+-',F6.4,//)
    9 FORMAT(F10.4,'   ',F14.8,'   ',F14.8)
      IF (EB2 .LE. MAXEB2) GOTO 23
      IF ((STAT .EQ. 1).AND.(FILE .EQ. 1)) THEN
         CALL REPORT(3)
      ENDIF
      CALL REPORT(1)
      CALL REPORT(2)
      STOP
      END
C-----------------------------------------------------------------INIT
      SUBROUTINE INIT(S)
      IMPLICIT REAL*8 (A-Z)
      INTEGER I,J
      DIMENSION CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: DEFINES PARAMETERS IN TRENTADUE'S FORM FACTORS
      BETA= 2.D0*ALPHA/PI*(DLOG(S/ME**2)-1.D0)
      Z2= 1.6449340668D0
      Z3= 1.202056903159D0
      L= DLOG(S/ME**2)
      DELP2= 1.D0+BETA**2*PI**2/24.D0+ALPHA/PI
     .       *(1.5D0*L+PI**2/3.D0-2.D0)
     .       +ALPHA**2.D0/PI**2.D0*((9.D0/8.D0-2.D0*Z2)*L**2.D0
     .       +(-45.D0/16.D0+
     .       11.D0/2.D0*Z2+3.D0*Z3)*L-6.D0/5.D0*Z2**2.D0-9.D0/2.D0*Z3-
     .       6.D0*Z2*DLOG(2.D0)+3.D0/8.D0*Z2+57.D0/12.D0)
      DELP= DSQRT(DELP2)
C: SETS UP WEIGHT HISTOGRAM
      CALL CLRHST
      RETURN
      END
C-----------------------------------------------------------------BAKERY
      SUBROUTINE BAKERY(S,NN,MM,ECUT,APRCRS)
      IMPLICIT REAL*8 (A-Z)
      INTEGER A,B,I,J,K,L,DL,NN,MM,PWRO2,THETA,LENGTH
      DIMENSION NOODLE(50000),XP(50000),XM(50000),SIZP(50000),
     .          THETA(50000),SIZM(50000),CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL
      COMMON/BAKRY/NOODLE,XP,XM,SIZP,SIZM,THETA,L
C: CONSTRUCTS NOODLES - DISCRETE SPACE: APPRSIG0, APRP+, APRP-
C: NOODLES EXIST ONLY IN BAKERY AND ENTRY POINT GENWGT BELOW
      XCUT=ECUT*2.D0/DSQRT(S)
      PCUT= GETP(XCUT)
      L= 0
C: THE BIG SQURE
      DO 1 I= 0,MM-1
      L= L+1
```

```
      XP(L)= XCUT
      XM(L)= XCUT
      SIZP(L)= 1.D0-XCUT
      SIZM(L)= 1.D0-XCUT
    1 THETA(L)= I
C: THE LEFT RECTANGLE
      DO 2 I= 0,MM-1
      L= L+1
      XP(L)= XCUT**2
      XM(L)= XCUT
      SIZP(L)= XCUT
      SIZM(L)= (1.D0-XP(L))-XCUT
    2 THETA(L)= I
C: THE RIGHT RECTANGLE
      DO 3 I= 0,MM-1
      L= L+1
      XM(L)= XCUT**2
      XP(L)= XCUT
      SIZM(L)= XCUT
      SIZP(L)= (1.D0-XM(L))-XCUT
    3 THETA(L)= I
      PRINT *,L
      DO 321 I= 1,NN
      CALL CUTXP(GETX(PCUT*I/NN))
  321 CALL CUTXM(GETX(PCUT*I/NN))
      LASTND= 0.D0
      I= 1
   23 NXP= XP(I)+SIZP(I)
      NXM= XM(I)+SIZM(I)
C: FINDS WILD NOODLES - PHOTON AND Z PEAKS
      WD= WILD(S*XP(I)*XM(I),S*NXP*NXM)
      IF ((WD.GT.0.D0).AND.(L.LE.49996)) THEN
         XP(L+1)= (XP(I)+NXP)/2.D0
         XM(L+1)= XM(I)
         THETA(L+1)= THETA(I)
         XP(L+2)= XP(L+1)
         XM(L+2)= (XM(I)+NXM)/2.D0
         THETA(L+2)= THETA(I)
         XP(L+3)= XP(I)
         XM(L+3)= XM(L+2)
         THETA(L+3)= THETA(I)
         SIZP(I)= SIZP(I)/2.D0
         SIZM(I)= SIZM(I)/2.D0
         SIZP(L+1)= SIZP(I)
         SIZM(L+1)= SIZM(I)
         SIZP(L+2)= SIZP(I)
         SIZM(L+2)= SIZM(I)
         SIZP(L+3)= SIZP(I)
         SIZM(L+3)= SIZM(I)
         L= L+3
         GOTO 23
      ENDIF
   24 XPMID= XP(I)+SIZP(I)/2.D0
      XMMID= XM(I)+SIZM(I)/2.D0
      CALL GETWD(XPMID,W1)
      CALL GETWD(XMMID,W2)
      CS= 2.D0*(THETA(I)+.5D0)/MM-1.D0
```

```
      DV= (GETP(XP(I)+SIZP(I))-GETP(XP(I)))*
     .    (GETP(XM(I)+SIZM(I))-GETP(XM(I)))*2.D0/MM
C: VECTOR NOODLE DEFINED AS CUMULATIVE SUM OF APPROXIMANT CROSS SECTIONS
      NOODLE(I)= LASTND+DSIG(S*XPMID*XMMID,CS,POL)*W1*W2*DV
      LASTND= NOODLE(I)
      I= I+1
      IF (I.LE.L) GOTO 23
      PWRO2= IDINT(DLOG(L+.5D0)/DLOG(2.D0))
      LENGTH= L
      PRINT *,LENGTH
C: FINAL ENTRY OF NOODLE = TOTAL CUMULATIVE MEGANOODLE
C:                       = APR TOT CS/2PI
      APRCRS= NOODLE(L)*2.D0*PI
      RETURN
C-----------------------------------------------------------------GENWGT
      ENTRY GENWGT(S,P,M,C,WGT,INDEX)

C: GENERATES EVENTS BY PICKING RANDOM POINT ALONG MEGA NOODLE
      RN= INDEX*NOODLE(LENGTH)
C: FINDS CORRESPONDING NOODLE THRU BINARY SEARCH
      L= 2**PWRO2
      IF (L.GT.LENGTH) L= L/2
      DL= L/2
  111 IF ((RN.GT.NOODLE(L)).AND.(L+DL.LE.LENGTH)) THEN
         L= L+DL
      ELSE IF (RN.LT.NOODLE(L-1)) THEN
         L= L-DL
      ELSE IF (L+DL.LE.LENGTH) THEN
         GOTO 400
      ENDIF
      DL= DL/2
      IF (DL.GT.0) GOTO 111
  400 SIZEP= GETP(XP(L))-GETP(XP(L)+SIZP(L))
      SIZEM= GETP(XM(L))-GETP(XM(L)+SIZM(L))
C: GENERATES EVENT: X+, X-, ANGLE, WEIGHT
      P= GETX(GETP(XP(L))-RANDOM(SIZEP))
      M= GETX(GETP(XM(L))-RANDOM(SIZEM))
      CALL GETWD(P,WP)
      CALL GETWD(M,WM)
      C= 2.D0*THETA(L)/MM-1.D0+RANDOM(2.D0/MM)
      WGT= WP*WM*DSIG(S*P*M,C,POL)*SIZEP*SIZEM*2.D0/MM/
     .     (NOODLE(L)-NOODLE(L-1))
C: BINS S/PRIME DISTRIBUTION
      RETURN
      END
C-----------------------------------------------------------------CUTXP
      SUBROUTINE CUTXP(X)
      IMPLICIT REAL*8 (A-Z)
      INTEGER I,L,THETA
      DIMENSION NOODLE(50000),XP(50000),XM(50000),SIZP(50000),
     .          THETA(50000),SIZM(50000),CUT(5)
      COMMON/BAKRY/NOODLE,XP,XM,SIZP,SIZM,THETA,L

      DO 124 I=1,L
         IF ((XP(I).LT.X).AND.(XP(I)+SIZP(I).GT.X)) THEN
            L= L+1
            XP(L)= X
```

```
            XM(L)= XM(I)
            SIZP(L)= XP(I)+SIZP(I)-X
            SIZM(L)= SIZM(I)
            THETA(L)= THETA(I)
            SIZP(I)= X-XP(I)
         ENDIF
  124 CONTINUE
      RETURN
      END
C-----------------------------------------------------------------CUTXM
      SUBROUTINE CUTXM(Y)
      IMPLICIT REAL*8 (A-Z)
      INTEGER I,L,THETA
      DIMENSION NOODLE(50000),XP(50000),XM(50000),SIZP(50000),
     .          THETA(50000),SIZM(50000),CUT(5)
      COMMON/BAKRY/NOODLE,XP,XM,SIZP,SIZM,THETA,L

      DO 124 I=1,L
         IF ((XM(I).LT.Y).AND.(XM(I)+SIZM(I).GT.Y)) THEN
            L= L+1
            XM(L)= Y
            XP(L)= XP(I)
            SIZM(L)= XM(I)+SIZM(I)-Y
            SIZP(L)= SIZP(I)
            THETA(L)= THETA(I)
            SIZM(I)= Y-XM(I)
         ENDIF
  124 CONTINUE
      RETURN
      END
C-----------------------------------------------------------------GETP
      FUNCTION GETP(X)
      IMPLICIT REAL*8 (A-Z)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: COMPUTES FUNCTION P GIVEN X
      IF (1.D0-X .LT. 0.2D0**(2.D0/BETA)) THEN
         GETP= 0.D0
      ELSE
         GETP= (1.D0-X)**(BETA/2.D0)
      ENDIF
      RETURN
      END
C-----------------------------------------------------------------WILD
      FUNCTION WILD(SL,SH)
      IMPLICIT REAL*8 (A-Z)
      DIMENSION CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: DEFINES WILD NOODLES AT PHOTON AND Z PEAKS
      WILD= -1.D0
      DE= DSQRT(SH)-DSQRT(SL)
      IF (((SL.LT.SCUT).AND.(DE.GT.4.D0)).OR.
     .    ((SL.LE.SHPOL).AND.(SH.GE.SLPOL).AND.(DE.GT.GAMZ/2.D0)).OR.
     .    (DE.GT.50.D0)) WILD= 1.D0
```

```
      RETURN
      END
C-----------------------------------------------------------------RANDOM
      FUNCTION RANDOM(X)
      REAL*8 X
      REAL*4 RAN11

      RANDOM= X*RAN11(0)
      RETURN
      END
C-----------------------------------------------------------------GENEV
      SUBROUTINE GENEV(S,SPRIME,MUP,MUM,WGT,INDEX)
      IMPLICIT REAL*8 (A-Z)
      DIMENSION MUP(4),MUM(4)
      DIMENSION CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: GENERATES EVENT: X+, X-, APR DIFF CS
      CALL GENWGT(S,XP,XM,COSTHP,WGT,INDEX)
      IF (WGT .EQ. 0.D0) THEN
         W= 0.D0
      ELSE
C: DEFINES C.M. (PRIMED) FRAME KINEMATICS: RANDOMLY DISTRIBUTED
C: POLAR AND AZIMUTHAL ANGLES
         SPRIME= S*XP*XM
         SINTHP= DSQRT(1.D0-COSTHP**2)
         COSPHP= DCOS(RANDOM(2.D0*PI))
         SINPHP= DSQRT(1.D0-COSPHP**2)
         BOOST= (XM-XP)/(XM+XP)
         GAMMA= 1.D0/DSQRT(1.D0-BOOST**2)
         EP= DSQRT(SPRIME)/2.D0
C: FIRST DEFINES MUM AND MUP MOMENTA IN C.M. (PRIMED) FRAME
         MUM(1)= EP*SINTHP*COSPHP
         MUM(2)= EP*SINTHP*SINPHP
         MUM(3)= EP*COSTHP
         MUM(4)= EP
         MUP(1)= -MUM(1)
         MUP(2)= -MUM(2)
         MUP(3)= -MUM(3)
         MUP(4)=  MUM(4)
C: BOOSTS BACK TO LAB FRAME
         MP3= GAMMA*(MUP(3)+BOOST*MUP(4))
         MP4= GAMMA*(MUP(4)+BOOST*MUP(3))
         MM3= GAMMA*(MUM(3)+BOOST*MUM(4))
         MM4= GAMMA*(MUM(4)+BOOST*MUM(3))
C: THEN REDEFINES MUM AND MUP MOMENTA IN LAB (UNPRIMED) FRAME
         MUP(3)= MP3
         MUM(3)= MM3
         MUP(4)= MP4
         MUM(4)= MM4
C: DEFINES TOTAL LEFT AND RIGHT WEIGHTS WITH PHASE SPACE CUTS
       WGT= WGT*PHCUT(MUM,MUP)
      END IF
      RETURN
      END
C-----------------------------------------------------------------GETX
```

```
      FUNCTION GETX(P)
      IMPLICIT REAL*8 (A-Z)
      DIMENSION CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL
C: COMPUTES X GIVEN FUNCTION P
      IF (P .EQ. 1.D0) THEN
         GETX= 0.D0
      ELSE IF (P .LT. 0.2D0) THEN
         GETX= 1.D0
      ELSE
         GETX= 1.D0-P**(2.D0/BETA)
      END IF
      RETURN
      END
C-----------------------------------------------------------------GETWD
      SUBROUTINE GETWD(X,W)
      IMPLICIT REAL*8 (A-Z)
      DIMENSION CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: COMPUTES BREMSSTRAHLUNG WEIGHT = EXACT D/APR D
C: IF X = 0, THROW EVENT WAY
      IF (X .EQ. 0.D0) THEN
         W= 0.D0
C: IF P < 0.1, X ESSENTIALLY ONE
      ELSE IF (X .GT. .9999999D0) THEN
         W= DELP
C: OTHERWISE...
      ELSE
         W= DELP-.5D0*(1.D0+X)*(1.D0-X)**(1.D0-BETA/2.D0)+BETA/16.D0*
     .       ((X+1.D0)*(-4.D0*DLOG(1.D0-X)+3.D0*DLOG(X))-4.D0/(1.D0-X)
     .       *DLOG(X)-5.D0-X)*(1.D0-X)**(1.D0-BETA/2.D0)
      END IF
      RETURN
      END
C-----------------------------------------------------------------PHCUT
      FUNCTION PHCUT(MUP,MUM)
      IMPLICIT REAL*8 (A-Z)
      DIMENSION MUP(4),MUM(4),CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

      IF (CUT(5).EQ.0.D0) THEN
         PHCUT= 1.D0
         RETURN
      ENDIF
C: MIN ENERGY CUT
      ECUT= 1.D0
      IF ((MUM(4).LT.CUT(1)).AND.(MUP(4).GE.CUT(2))) ECUT=0.D0
C: ENDCAP ANGLE CUT
      COSTH= MUM(3)/MUM(4)
      THCUT= 1.D0
      IF (DABS(COSTH).GT.CUT(3)) THCUT= 0.D0
C: ACOLLINEARITY CUT
      DOT= MUM(1)*MUP(1)+MUM(2)*MUP(2)+MUM(3)*MUP(3)
```

```
      COSXI= -DOT/MUM(4)/MUP(4)
      XICUT= 1.D0
      IF (COSXI.LT.CUT(4)) XICUT= 0.D0
C: TOTAL CUT
      PHCUT= ECUT*THCUT*XICUT
      RETURN
      END
C-----------------------------------------------------------------CLRHST
      SUBROUTINE CLRHST
      INTEGER I,J,N
      REAL*8 HST,MIN,MAX,X,W,XMIN,XMAX
      DIMENSION HST(500,5),MIN(5),MAX(5)

C: CLEARS COMMON HISTOGRAM
      DO 1 I= 1,500
      DO 1 J= 1,5
    1 HST(I,J)= 0.D0
      RETURN
C-----------------------------------------------------------------HISTO
      ENTRY HISTO(X,W,XMIN,XMAX,N)

C: BINS WEIGHTS IN HISTOGRAM
      MIN(N)= XMIN
      MAX(N)= XMAX
      I= (X-XMIN)/(XMAX-XMIN)*500
      IF ((I .LE. 500) .AND. (I .GT. 0)) HST(I,N)= HST(I,N)+W
      RETURN
C-----------------------------------------------------------------REPORT
      ENTRY REPORT(N)

C: OUTPUTS HISTOGRAMS
      DO 2 I=1,500
    2 WRITE (30+N,*) I*(MAX(N)-MIN(N))/500.D0+MIN(N),HST(I,N)
      RETURN
      END
C-----------------------------------------------------------------DSIG
      FUNCTION DSIG(SP,X,POL)
      IMPLICIT REAL*8 (A-Z)

C: DEFINES CROSS SECTION DEPENDING ON POLARIZATION
      IF (POL .EQ. -1.D0) DSIG= DSIGL(SP,X)
      IF (POL .EQ.  1.D0) DSIG= DSIGR(SP,X)
      RETURN
      END
C-----------------------------------------------------------------DSIGL
      FUNCTION DSIGL(SP,X)
      IMPLICIT REAL*8 (A-Z)
      COMPLEX*16 R
      DIMENSION CUT(5)
      COMMON/COUPL/CEL,CER,CML,CMR,QE,QM
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: LEFT-HANDED CROSS SECTION - X = COS OF POLAR ANGLE
      DSIGL=CONST/SP*((1.D0+X)**2*(CDABS(QE*QM+R(SP)*CEL*CML))**2.D0
     .               +(1.D0-X)**2*(CDABS(QE*QM+R(SP)*CEL*CMR))**2.D0)
      DSIGL=DSIGL/2.D0
```

```
      RETURN
      END
C-----------------------------------------------------------------------DSIGR
      FUNCTION DSIGR(SP,X)
      IMPLICIT REAL*8 (A-Z)
      COMPLEX*16 R
      DIMENSION CUT(5)
      COMMON/COUPL/CEL,CER,CML,CMR,QE,QM
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: RIGHT-HANDED CROSS SECTION - X = COS OF POLAR ANGLE
      DSIGR=CONST/SP*((1.D0-X)**2*(CDABS(QE*QM+R(SP)*CER*CML))**2.D0
     .               +(1.D0+X)**2*(CDABS(QE*QM+R(SP)*CER*CMR))**2.D0)
      DSIGR=DSIGR/2.D0
      RETURN
      END
C-------------------------------------------------------------------------R
      FUNCTION R(S)
      IMPLICIT REAL*8 (A-Z)
      COMPLEX*16 R
      DIMENSION CUT(5)
      COMMON/PARAM/BETA,ALPHA,PI,ME,MZ,GAMZ,SINW2,CONST,DELP,CUT,POL,
     .             SCUT,SLPOL,SHPOL

C: NORMALIZED Z0 RESONANCE FACTOR
      R= S/4.D0/SINW2/(1.D0-SINW2)/DCMPLX(S-MZ**2,S*GAMZ/MZ)
      RETURN
      END
```

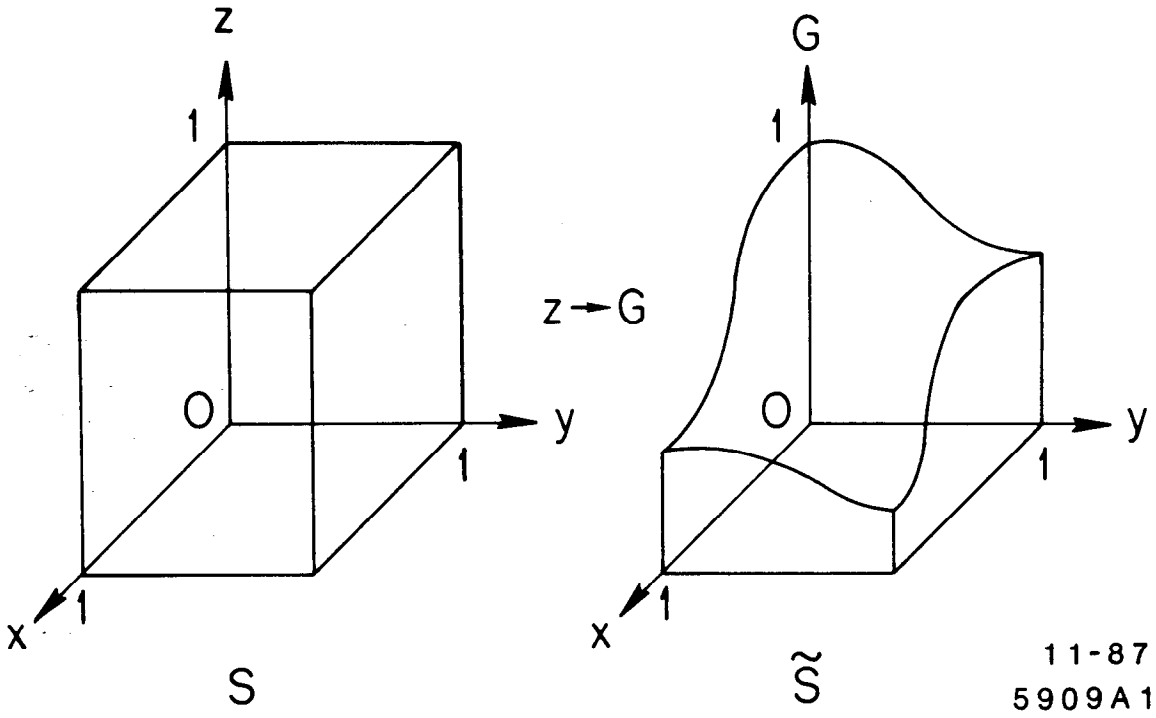


Fig. 1

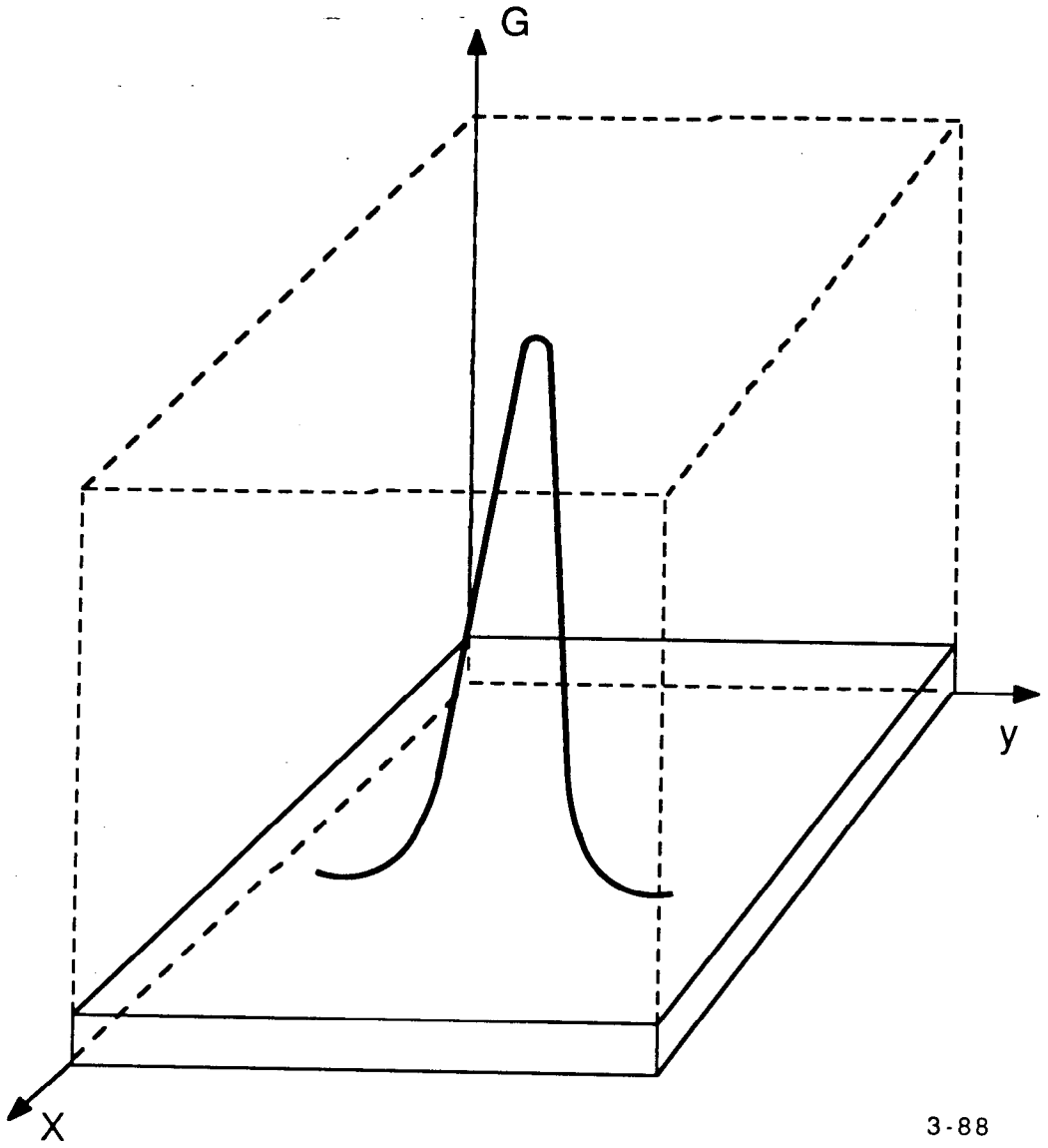


Fig. 2

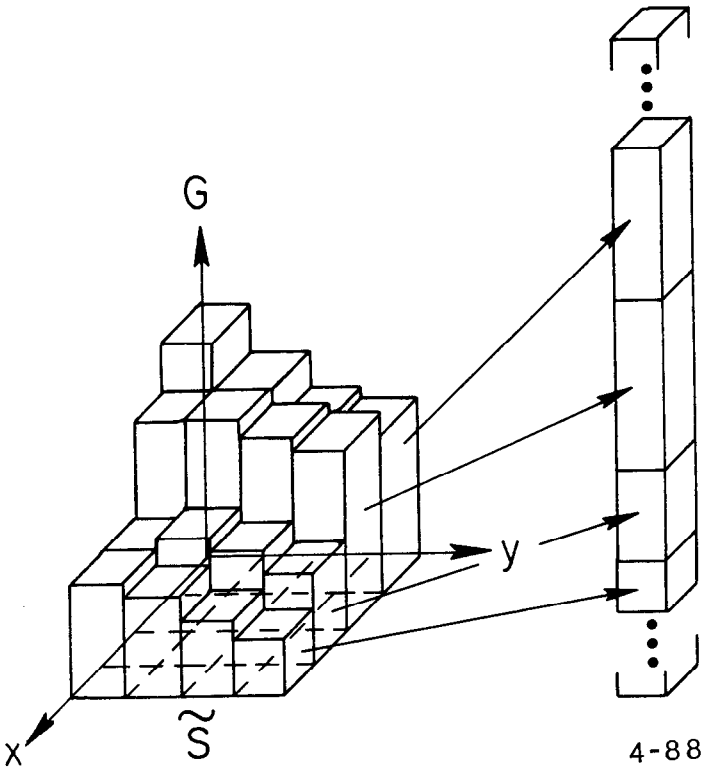


Fig. 3

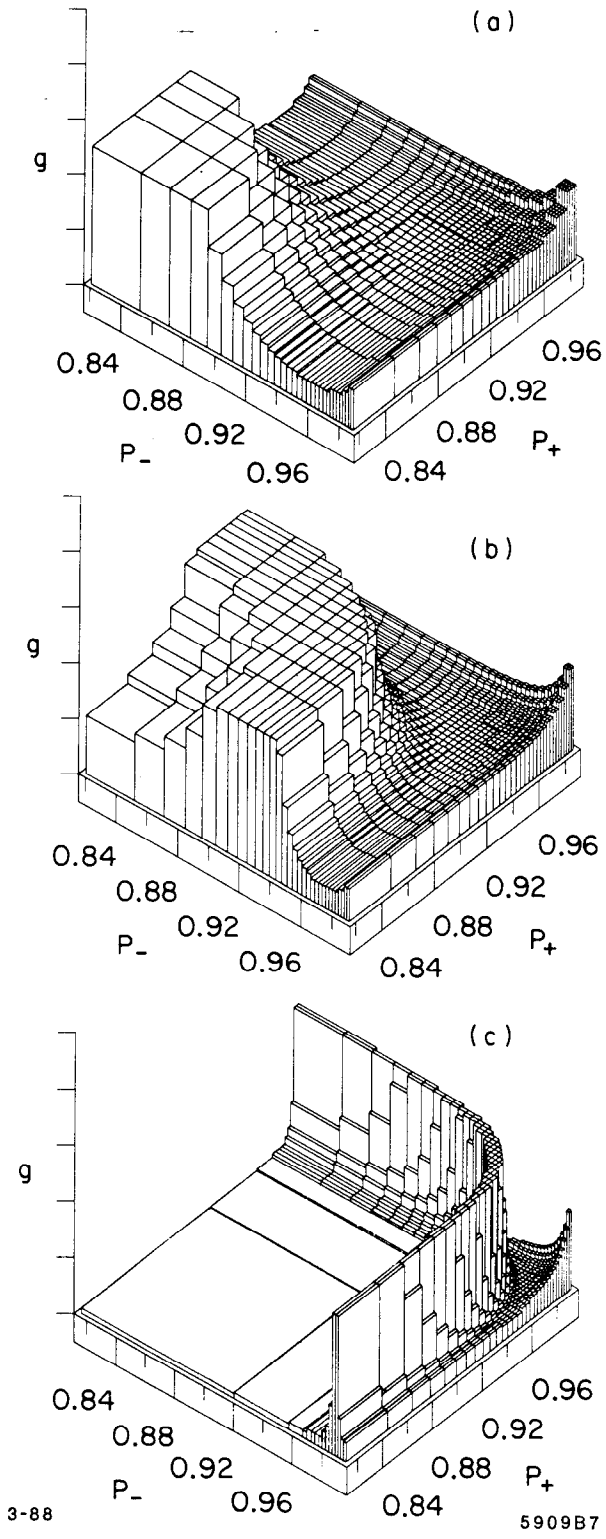
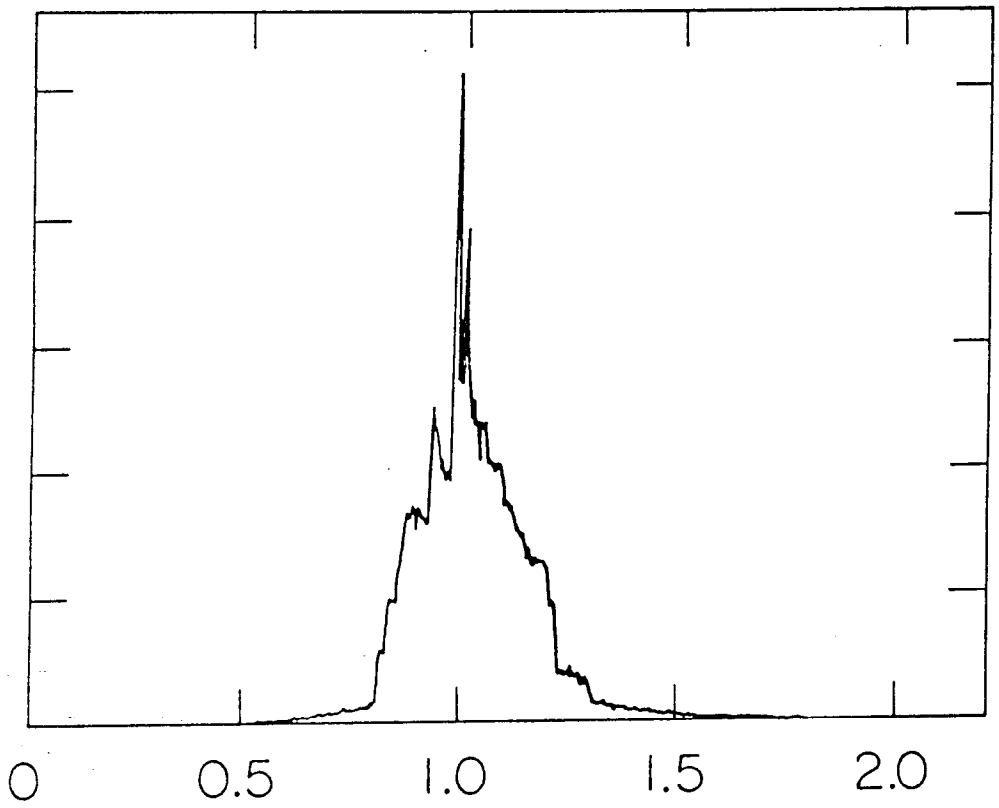

(a)
g
0.84
0.88
0.92
0.96
P-
0.84
0.88
0.92
0.96
P+
(b)
g
0.84
0.88
0.92
0.96
P-
0.84
0.88
0.92
0.96
P+
(c)
g
0.84
0.88
0.92
0.96
P-
0.84
0.88
0.92
0.96
P+
3-88
5909B7

Fig. 4



5909A6

Fig. 5

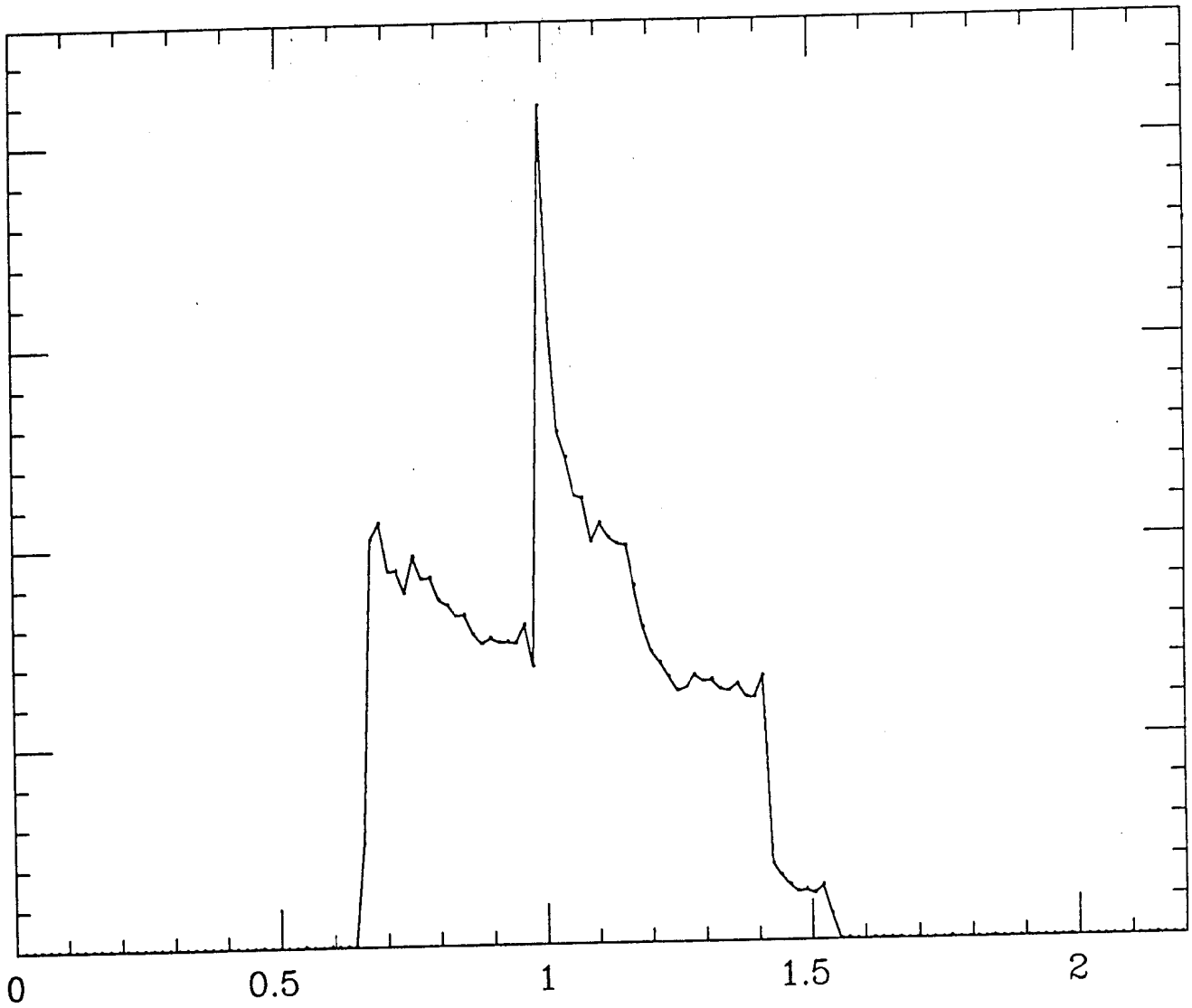


Fig. 6