Cap. 6 Conclusioni

Sono state implementate le ricostruzioni delle tracce mediante algoritmi genetici e reti neurali. Confrontando i risultati con quelli ottenuti tramite gli algoritmi di ricostruzione classici, già studiati e implementati nelle collaborazioni NEMO ed ANTARES, si osserva che le reti neurali danno prestazioni confrontabili con quelle del prefit, e possono essere utilizzate per un filtraggio veloce delle tracce up-going.

Gli algoritmi genetici permettono di raggiungere prestazioni molto simili a quelle della minimizzazione diretta dell'errore quadratico, anche se con un tempo di elaborazione più lungo di circa 50 volte. In Figura 102 e Figura 103 sono riportati i confronti della mediana di ricostruzione e dell'area efficace.

L'algoritmo genetico descritto nel Cap. 5 è più appropriato nello studio di sistemi fisici particolarmente complessi. Nel nostro caso viene usato per minimizzare una funzione (χ^2) abbastanza semplice. Il risultato ottenuto è comunque apprezzabile visto che, con un metodo totalmente indipendente dalle tecniche di correlazioni causali spazio-temporali riesce ad ottenere risoluzioni spaziali confrontabili.

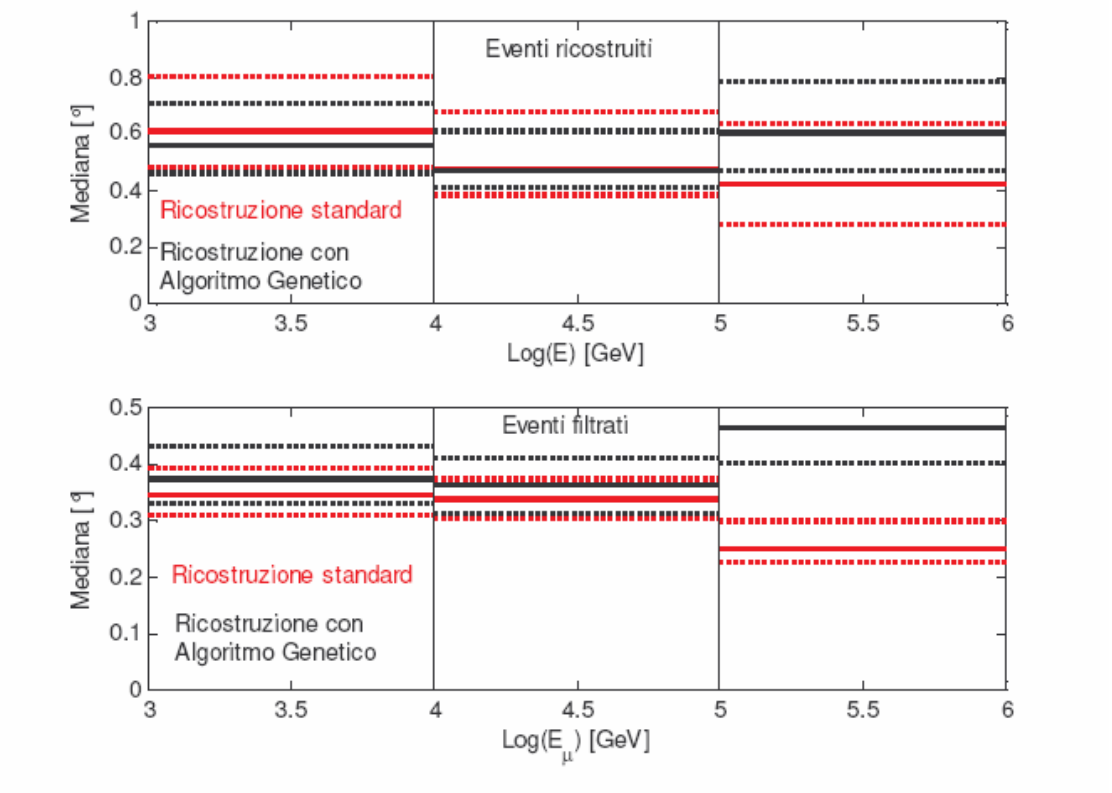

Figura 102 - Confronto tra le mediane di ricostruzione dell'algoritmo standard e dell'algoritmo genetico.

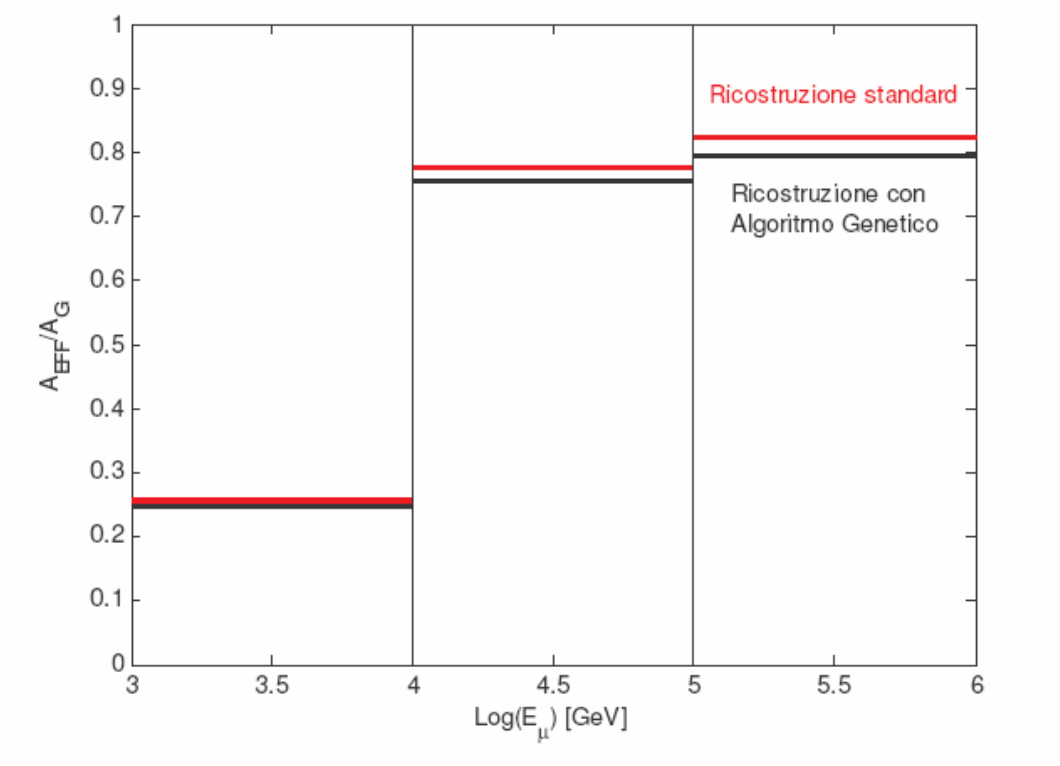

Figura 103 - Confronto tra le aree efficaci dell'algoritmo standard e dell'algoritmo genetico.

Appendice

Funzioni per l'implementazione dell'algoritmo genetico

```
      subroutine trkfit(iflag)

      integer N_POP, N_GEN
      dimension afcn(6)

      double precision tprimo,tultimo
      common /forse/ IPMfirst,KPMfirst,tprimo,IPMlast,KPMlast,tultimo
*
      include 'param81.inc'
      include 'common_geom81.inc'
      include 'common_fase181.inc'
      include 'common_evread81new.inc'
      include 'common_evdata81.inc'
      include 'common_steer.inc'
      include 'common_analysisnew.inc'

      include 'paramGA.inc'

      double precision fitness(N_MAX_POP)
      double precision fit_temp(N_MAX_POP),best_fit(N_MAX_GEN)
      double precision AAFCN(6), tempfit
      double precision sigmar_init, sigmaa_init

      double precision quX, quFI, quTH, DiamX, CentroX
      double precision best_temp, best_mc

      common/GAdata/IPOP(N_MAX_POP,5),NEW_POP(N_MAX_POP,5),
     +  fitness
      common/GAdata/fit_temp,best_fit
      common/GAdata/igene(5),itempgen(5), AAFCN, tempfit
      common/GAdata/sigmar_init, sigmaa_init, INIT_TYPE
      common/GAdim/N_POP, N_GEN, quX, quFI, quTH, DiamX, CentroX
```

```
      double precision rangte,rangfi,xbar,ybar,zbar
      common /resultprefit/ rangte,rangfi,xbar,ybar,zbar
      real RANDOM, x, y(N_MAX_POP), xi,yi,zi,fii,thi, pig

      double precision X_OPT, Y_OPT, Z_OPT, FI_OPT, TH_OPT
      double precision x0,y0,z0,thfit,phfit,r,cx,cy,cz
      double precision argtest,argcos,tcosfit,tcosmc
      double precision cxtrue,cytrue,cztrue,anglfp
      double precision xx0, yy0, zz0

      integer gene_mc(5)
      logical CONT

*============================================================
      N_POP=300
      N_GEN=300
      quX=0.025
      quFI=0.0000958738
      quTH=0.0000479369
      DiamX=1638.4
      CentroX=819.2
      sigmar_init=200.0
      sigmaa_init=0.1
      pig    = 3.141593
      INIT_TYPE=1
      CONT=.true.
      best_mc=1000000000.0
      imc=0
      MC=1

      if(iflag.eq.1) then
*--------1---------2---------3---------4---------5---------6---------7-*
*                                                                      *
*                          IFLAG = 1                                   *
*                                                                      *
*--------1---------2---------3---------4---------5---------6---------7-*
*
      elseif(iflag.eq.2) then
*--------1---------2---------3---------4---------5---------6---------7-*
*                                                                      *
*                          IFLAG = 2                                   *
*                                                                      *
*--------1---------2---------3---------4---------5---------6---------7-*
*
      nt=0
      do i=1,npmhit
        if (flagPMT_HIT(i,3).eq.0) then
          nt=nt+1
```

```
      endif
    end do
    if (nt.ge.5) then
      do while (CONT)

        if (nt.gt.200) then
          MC=3
        elseif (nt.gt.100) then
          MC=5
        elseif (nt.gt.30) then
          MC=10
        elseif (nt.gt.10) then
          MC=15
        elseif (nt.gt.5) then
          MC=20
        else
          MC=0
        endif

     if (INIT_TYPE.eq.1) then
* inizializzazione pilotata dai risultati del prefit
        cxpre    = sin(rangte)*cos(rangfi)
        cypre    = sin(rangte)*sin(rangfi)
        czpre    = cos(rangte)
        aa       = 1.
        bb       = -2.*(cxpre*xbar + cypre*ybar + czpre*zbar)
        cc       = xbar**2 + ybar**2 + zbar**2 - CentroX**2
* utilizzo un raggio della sfera di 800m, più vicino al primo hit della traccia
        test = bb**2 - 4.*aa*cc
        if(test.lt.0.) then
          solm     = (-bb)/(2.*aa)
          solp     = (-bb)/(2.*aa)
        else
          solm     = (-bb - sqrt(bb**2 - 4.*aa*cc))/(2.*aa)
          solp     = (-bb + sqrt(bb**2 - 4.*aa*cc))/(2.*aa)
        endif

        xx0      = xbar - cxpre * solp
        yy0      = ybar - cypre * solp
        zz0      = zbar - czpre * solp
*
              do i=1,N_POP
* un punto random intorno al punto trovato sopra, con larghezza sigmar_init
                CALL NORRAN(RANDOM)
                      xi=sigmar_init*RANDOM+xx0
                      CALL NORRAN(RANDOM)
                      yi=sigmar_init*RANDOM+yy0
                      CALL NORRAN(RANDOM)
                      zi=sigmar_init*RANDOM+zz0
```

```
* il punto non deve essere esterno al dominio di esistenza + o - CentroX
                  xi=max(xi,-CentroX)
                  yi=max(yi,-CentroX)
                  zi=max(zi,-CentroX)
                  xi=min(xi,CentroX)
                  yi=min(yi,CentroX)
                  zi=min(zi,CentroX)
* angoli random intorno agli angoli del prefit, con larghezza sigmaa_init (in radianti)
                  CALL NORRAN(RANDOM)
           if (rangfi.lt.0.0) then
                        fii=sigmaa_init*RANDOM+rangfi+2*pig
           else
             fii=sigmaa_init*RANDOM+rangfi
           end if
                  CALL NORRAN(RANDOM)
                        thi=sigmaa_init*RANDOM+rangte
* anche gli angoli devono essere limitati tra 0 e 360 fi e tra 0 e 180 teta
                  fii=amax1(fii,0.0)
                  fii=amin1(fii,2*pig)
                  thi=amax1(thi,0.0)
                  thi=amin1(thi,pig)
* POP contiene i geni codificati a 16 bit (tra o e 65535)
                  IPOP(i,1)=32767*(xi+CentroX)/CentroX
                  IPOP(i,2)=32767*(yi+CentroX)/CentroX
                  IPOP(i,3)=32767*(zi+CentroX)/CentroX
                  IPOP(i,4)=65535*fii/(2*pig)
                  IPOP(i,5)=65535*thi/pig
                end do
              elseif (INIT_TYPE.eq.2) then
* inizializzazione random
                 do i=1,N_POP
           call RANLUX(y,5)
           do j=1,5
             IPOP(i,j)=65535*y(j)
           end do
                 end do
              elseif (INIT_TYPE.eq.3) then
* inizializzazione costante da un punto
         do i=1,N_POP
           IPOP(i,1)=32768
           IPOP(i,2)=32768
           IPOP(i,3)=0
           IPOP(i,4)=0
           IPOP(i,5)=0
         end do
       end if

            do igen=1,(N_GEN-1)
*-------------------------------------------------------
```

```
* ciclo sulle generazioni

                call FITNEMO
          best_temp=1000000000.0
                do ip=1,N_POP
                  fitness(ip)=fit_temp(ip)
             if (fitness(ip).lt.best_temp) best_temp=fitness(ip)
                end do
                best_fit(igen)=best_temp
* calcola la fitness di tutti gli individui e trova il migliore
*---------------------------------------------------------------


*---------------------------------------------------------------
                call SELECTION
* trascrive gli individui migliori e seleziona quelli per il CO
*---------------------------------------------------------------

*-----------------------------
                call CROSSOVER
* attua il CO
*-----------------------------

*-----------------------------
                call NEW_ENTRIES
* attua la MUTAZIONE
*-----------------------------

                do m=1,N_POP
             do k=1,5
               IPOP(m,k)=NEW_POP(m,k)
             end do
          end do

        end do
C    fine ciclo sulle generazioni
C_____________________________________________________

*-----------------------------------------
* ULTIMA VALUTAZIONE e SCELTA DEL MIGLIORE
               call FITNEMO
        best_temp=1000000000.0
               do ip=1,N_POP
                 fitness(ip)=fit_temp(ip)
           if (fitness(ip).lt.best_temp) then
             best_temp=fitness(ip)
             IND_MIN=ip
           endif
               end do
```

```
            best_fit(igen+1)=best_temp
          write(41,*) iev, best_temp
          if (best_temp.lt.best_mc) then
            best_mc=best_temp
            do k=1,5
              gene_mc(k)=IPOP(IND_MIN,k)
            end do
          endif

          if (best_mc.eq.0) CONT=.false.
          imc=imc+1
          print *, 'imc ', imc
          if (imc.ge.MC) CONT=.false.
          end do

          sumn=best_mc

                call DECODE_GENE(gene_mc,X_OPT,Y_OPT,Z_OPT,FI_OPT,TH_OPT)

                x0=X_OPT
                y0=Y_OPT
                z0=Z_OPT
                cx=sin(TH_OPT)*cos(FI_OPT)
                cy=sin(TH_OPT)*sin(FI_OPT)
                cz=cos(TH_OPT)
          thfit=TH_OPT*180/pig
          phfit=FI_OPT*180/pig
          if (phfit.gt.180) then
            phfit=phfit-360.
          endif
          print *,'Delta theta ', thfit, thmu, thfit-thmu
          print *,'Delta fi' , phfit, phimu, phfit-phimu

*******************************************************
          xmc0   = vxmu
          ymc0   = vymu
          zmc0   = vzmu
          cxmc   = pxmu/sqrt(pxmu**2+pymu**2+pzmu**2)
          cymc   = pymu/sqrt(pxmu**2+pymu**2+pzmu**2)
          czmc   = pzmu/sqrt(pxmu**2+pymu**2+pzmu**2)
          cxtrue = dble( cxmc )
          cytrue = dble( cymc )
          cztrue = dble( czmc )
          anglfp = 2.*pig
          if((cx**2+cy**2+cz**2).ne.dble( 0.).
     +      and.(cxtrue**2+cytrue**2+cztrue**2).ne.dble( 0.)) then
            if((dabs(cx*cxtrue+cy*cytrue+cz*cztrue)).lt.dble(1.01))then
              if( (cx*cxtrue+cy*cytrue+cz*cztrue) .gt.dble(1.01))
     +            anglfp = dacos(dble(1.))
```

```
            if( (cx*cxtrue+cy*cytrue+cz*cztrue) .lt.dble(-1.01))
     +           anglfp = dacos(dble(-1.))
            else
              write(16,*)' cannot evaluate anglfp !!!',
     ,                cx,cy,cz,cxtrue,cytrue,cztrue
            print *,' attention !, cx,cy,cz,cxmc,cymc,czmc',
     ,        cx,cy,cz,ccxmc,cymc,czmc
            print *,' and the argument of acos :',
     ,           cx*cxtrue+cy*cytrue+cz*cztrue
            endif
          endif
          argtest = cx*cxtrue+cy*cytrue+cz*cztrue
          argcos  = ( cx*cxtrue + cy*cytrue + cz*cztrue )/
     +        (dsqrt(cx**2+cy**2+cz**2)*
     +        dsqrt(cxtrue**2+cytrue**2+cztrue**2))
          tcosfit = cx**2+cy**2+cz**2
          tcosmc  = cxtrue**2+cytrue**2+cztrue**2
          call hfill(3010,sngl(argtest),0.,1.)
          call hfill(3011,sngl(argcos ),0.,1.)
          call hfill(3012,sngl(tcosfit),0.,1.)
          call hfill(3013,sngl(tcosmc ),0.,1.)
          if((dabs(cx*cxtrue+cy*cytrue+cz*cztrue)).le.dble(1.))then
            anglfp = dacos( cx*cxtrue + cy*cytrue + cz*cztrue )/
     +        (dsqrt(cx**2+cy**2+cz**2)*
     +        dsqrt(cxtrue**2+cytrue**2+cztrue**2))
            if(iprlvl.ge.3) then
              print *,' in FCNTRK cx,cy,cz from fit and from MC',
     +          cx,cy,cz,cxmc,cymc,czmc
            endif
          endif

          write(16,*) sngl(x0),sngl(y0),sngl(z0),sngl(thfit),
     +   sngl(phfit),thmu,phimu,anglfp,sumn
          thepre = 180.*rangte/pig
          phipre = 180.*rangfi/pig
          write(16,2002) thepre,phipre
 2002 format(' theta, phi prefit',2f10.2)
          deltang = sngl(anglfp)*180./pig
          emul = log10(emu/1000.)


* chiamate per la gestione degli istogrammi
            call hfill(3000,sngl(thfit )-thmu ,0.,1.)
*     ...................
*     ...................
*     ...................

        elseif(iflag.eq.3) then
*--------1---------2---------3---------4---------5---------6---------7-*
```

```
*                                                     *
*                         IFLAG = 3                         *
*                                                     *
*--------1---------2---------3---------4---------5---------6---------7-*
*
      elseif(iflag.eq.4) then
*--------1---------2---------3---------4---------5---------6---------7-*
*                                                     *
*                         IFLAG = 4                         *
*--------1---------2---------3---------4---------5---------6---------7-*
*
      endif
      return
 30   FORMAT(A30)
      end

*--------1---------2---------3---------4---------5---------6---------7-*
      subroutine DECODE_GENE(igene_p, XDEC, YDEC, ZDEC, FIDEC, THDEC)
* dato un gene creo i valori x, y, z, fi e theta corrispondenti
* i valori di distanza sono in metri, i valori angolari sono in radianti
      implicit double precision(a-h,o-z)
      integer igene_p(5)
      include 'paramGA.inc'
      include 'common_GAdim.inc'

      common/GAdata/IPOP(N_MAX_POP,5),NEW_POP(N_MAX_POP,5),
     +  fitness(N_MAX_POP)
      common/GAdata/fit_temp(N_MAX_POP),best_fit(N_MAX_GEN)
      common/GAdata/igene(5),itempgen(5), AAFCN(6), tempfit
      common/GAdata/sigmar_init, sigmaa_init, INIT_TYPE


      XDEC=igene_p(1)*quX-CentroX
      YDEC=igene_p(2)*quX-CentroX
      ZDEC=igene_p(3)*quX-CentroX
      FIDEC=igene_p(4)*quFI
      THDEC=igene_p(5)*quTH

      end

*--------1---------2---------3---------4---------5---------6---------7-*
      subroutine FITNEMO
* calcola l'errore quadratico dei tempi delle tracce codificate nei geni della popolazione

      include 'param81.inc'
      include 'paramGA.inc'

      include 'common_geom81.inc'
      include 'common_fase181.inc'
```

```
      include 'common_evread81new.inc'
      include 'common_evdata81.inc'
      include 'common_steer.inc'
      include 'common_analysisnew.inc'

      double precision fitness(N_MAX_POP)
      double precision fit_temp(N_MAX_POP),best_fit(N_MAX_GEN)
      double precision AAFCN(6), tempfit
      double precision sigmar_init, sigmaa_init
      double precision xx, yy, zz, ffi, tth
      double precision quX, quFI, quTH, DiamX, CentroX
      double precision t_th, s, ST, out, tme, delt

      common/GAdata/IPOP(N_MAX_POP,5),NEW_POP(N_MAX_POP,5),
     +  fitness
      common/GAdata/fit_temp,best_fit
      common/GAdata/igene(5),itempgen(5), AAFCN, tempfit
      common/GAdata/sigmar_init, sigmaa_init, INIT_TYPE
      common/GAdim/N_POP, N_GEN, quX, quFI, quTH, DiamX, CentroX

      nt=0
      do i=1,npmhit
        if (flagPMT_HIT(i,3).eq.0) nt=nt+1
      end do
      do i=1,N_POP
                do k=1,5
                  igene(k)=IPOP(i,k)
                end do
                call DECODE_GENE(igene,xx,yy,zz,ffi,tth)
                AAFCN(1)=xx
                AAFCN(2)=yy
                AAFCN(3)=zz
                AAFCN(4)=cos(ffi)*sin(tth)
                AAFCN(5)=sin(ffi)*sin(tth)
                AAFCN(6)=cos(tth)

                out=0
          if (nt.ge.5) then

                  do jj=1,npmhit

               if (flagPMT_HIT(jj,3).eq.0) then
                  ip=idPMT_HIT(jj)
                        call tpmcal(ip,t_th,AAFCN,s)
                  ST=dble(pmtdata(ip,8))

                  tme=timePMT_HIT(jj,2)

                  delt=tme-t_th
```

```
                    out=out+delt**2/ST**2

               endif
                  end do
                  fit_temp(i)=log10(out/nt)

                  if (fit_temp(i).lt.0) fit_temp(i)=0
          else
            fit_temp(i)=1000000000.0
          endif
             end do

             end

*--------1---------2---------3---------4---------5---------6---------7-*
      subroutine SELECTION
* copia nella NEW_POP i migliori (N_BEST) e seleziona con il metodo
* della roulette wheel quelli destinati al CrossOver
      implicit double precision(a-h,o-z)
      include 'paramGA.inc'
      include 'common_GAdim.inc'
      dimension temp_sel(N_MAX_POP)

      real RANDOM
      common/GAdata/IPOP(N_MAX_POP,5),NEW_POP(N_MAX_POP,5),
     +  fitness(N_MAX_POP)
      common/GAdata/fit_temp(N_MAX_POP),best_fit(N_MAX_GEN)
      common/GAdata/igene(5),itempgen(5), AAFCN(6), tempfit
      common/GAdata/sigmar_init, sigmaa_init, INIT_TYPE
      real x, y(N_MAX_POP)
      double precision MIN_SEL
      newe=1
      N_NEW=int(N_POP/10)+1
      N_BEST=int(N_POP/30)+1
      if (newe.gt.0) then
        N_CO_S=N_POP-N_BEST-N_NEW
      else
        N_CO_S=N_POP-N_BEST
      endif

      do j=1,N_POP
        temp_sel(j)=fitness(j)
      end do

      do i=1,N_BEST
        tempfit=1000000000.0
        IND_MIN=0
        do j=1,N_POP
          if (temp_sel(j).lt.tempfit) then
```

```
            tempfit=temp_sel(j)
            IND_MIN=j
          end if
        end do
        do k=1,5
          NEW_POP(i,k)=IPOP(IND_MIN,k)
          temp_sel(IND_MIN)=1000000000.0
             end do
      end do
      do i=1,N_CO_S
        call RANLUX(y,N_POP)
        do k=1,N_POP
               temp_sel(k)=fitness(k)*y(k)
             end do
             MIN_SEL=1000000000.0
             IND_MIN=0
             do k=1,N_POP
               if (temp_sel(k).lt.MIN_SEL) then
                 MIN_SEL=temp_sel(k)
                 IND_MIN=k
               endif
             end do
             do k=1,5
               NEW_POP(N_BEST+i,k)=IPOP(IND_MIN,k)
             end do
      end do
      end

*--------1---------2---------3---------4---------5---------6---------7-*
      subroutine CROSSOVER
* procedura che implementa il CROSS OVER a due a due sui geni selezionati
      implicit double precision(a-h,o-z)
      include 'paramGA.inc'
      include 'common_GAdim.inc'

      common/GAdata/IPOP(N_MAX_POP,5),NEW_POP(N_MAX_POP,5),
     +  fitness(N_MAX_POP)
      common/GAdata/fit_temp(N_MAX_POP),best_fit(N_MAX_GEN)
      common/GAdata/igene(5),itempgen(5), AAFCN(6), tempfit
      common/GAdata/sigmar_init, sigmaa_init, INIT_TYPE

      integer gene1(5), gene2(5), newgene1(5), newgene2(5)
      integer jj, CO_TYPE, CUT_POINT, INDW_CUT, INDK_CUT
      integer w11, w12, w21, w22, w(5), wn(5)
      double precision cromo1, cromo2, ncrom1, ncrom2
      real x,y(N_MAX_POP)

      newe=1
      cop=1
```

```
      call RANLUX(x,1)
      if (x.lt.cop) then

      N_NEW=int(N_POP/10)+1
      N_BEST=int(N_POP/30)+1
      if (newe.gt.0) then
        N_CO=N_POP-N_NEW
      else
        N_CO=N_POP
      endif
            do i=(N_BEST+1),N_CO,2
              do k=1,5
                gene1(k)=NEW_POP(i,k)
                gene2(k)=NEW_POP(i+1,k)
              end do

              CO_TYPE=2
              if (CO_TYPE.eq.1) then
* Crossover ad un taglio
           call RANLUX(y,1)
                CUT_POINT=NINT(y(1)*78+1)
* punto di taglio random da 1 a 79 (lunghezza gene 80)
                INDW_CUT=INT(CUT_POINT/16)+1
* parola selezionata per il taglio
                do jj=1,INDW_CUT-1
                  newgene1(jj)=gene1(jj)
                  newgene2(jj)=gene2(jj)
                end do
                do jj=INDW_CUT+1,5
                  newgene1(jj)=gene2(jj)
                  newgene2(jj)=gene1(jj)
                end do
* ho copiato le parole non tagliate, quelle prima del taglio in ordine,
* quelle dopo il taglio in ordine inverso
                INDK_CUT=mod(CUT_POINT,16)
*            print *, 'cut ', INDW_CUT, INDK_CUT
* taglio all'interno della parola selezionata
                MASK1=0
* parte più significativa
                do kk=0,INDK_CUT
                  MASK1=MASK1+2**(15-kk)
                end do
                MASK2=2**16-1-MASK1
*    parte meno significativa
                newgene1(INDW_CUT)=IOR(IAND(gene1(INDW_CUT),MASK1),
     +       IAND(gene2(INDW_CUT),MASK2))
                newgene2(INDW_CUT)=IOR(IAND(gene1(INDW_CUT),MASK2),
     +       IAND(gene2(INDW_CUT),MASK1))
                do k=1,5
```

```
                    NEW_POP(i,k)=newgene1(k)
                    NEW_POP(i+1,k)=newgene2(k)
                  end do
                elseif (CO_TYPE.eq.2) then
*       crossover2 è con due punti di taglio.
*       g1 e g2 sono i due geni che partecipano al cross over
*       sono due vettori riga di 5 elementi x y z fi teta
*       ogni elemento è un uint16

*         due indici random (1..62; ind1+16..79) danno i punti di taglio
          call RANLUX(y,2)
          ind1=anint(61*y(1))+1
          ind2=anint((63-ind1)*y(2))+ind1+16
          k1=int(ind1/16)+1
          k2=int(ind2/16)+1

          do jj=1,(k1-1)
            newgene1(jj)=gene1(jj)
            newgene2(jj)=gene2(jj)
          end do
          do jj=(k1+1),(k2-1)
            newgene1(jj)=gene2(jj)
            newgene2(jj)=gene1(jj)
          end do
          do jj=(k2+1),5
            newgene1(jj)=gene1(jj)
            newgene2(jj)=gene2(jj)
          end do
          indk1=mod(ind1,16)
          indk2=mod(ind2,16)
* indk va da 0 a 15
*------------------
          MASK11=0
*   % parte più significativa
          do kk=0,indk1
            MASK11=MASK11+2**(15-kk)
          end do
          MASK21=2**16-1-MASK11
*  % parte meno significativa
          newgene1(k1)=IOR(IAND(gene1(k1),MASK11),
     +       IAND(gene2(k1),MASK21))
          newgene2(k1)=IOR(IAND(gene1(k1),MASK21),
     +       IAND(gene2(k1),MASK11))
          MASK12=0
*   % parte più significativa
          do kk=0,indk2
            MASK12=MASK12+2**(15-kk);
          end do
          MASK22=2**16-1-MASK12
```

```
*   % parte meno significativa
            newgene1(k2)=IOR(IAND(gene1(k2),MASK22),
     +       IAND(gene2(k2),MASK12))
            newgene2(k2)=IOR(IAND(gene1(k2),MASK12),
     +       IAND(gene2(k2),MASK22))
                do k=1,5
                  NEW_POP(i,k)=newgene1(k)
                  NEW_POP(i+1,k)=newgene2(k)
                end do
              else if (CO_TYPE.eq.3) then
* % crossover_u è il crossover uniforme
* % cioè ogni bit di g1 passa a g2 con probabilità 50%
* % g1 e g2 sono i due geni che partecipano al cross over
* % sono due vettori riga di 5 elementi x y z fi teta
* % ogni elemento è un uint16

            call RANLUX(y,5)
            do k=1,5
              w(k)=int(65535*y(k))
              wn(k)=65536-w(k)
              newgene1(k)=IOR(IAND(gene1(k),wn(k)),
     +         IAND(gene2(k),w(k)))
              newgene2(k)=IOR(IAND(gene1(k),w(k)),
     +         IAND(gene2(k),wn(k)))
              NEW_POP(i,k)=newgene1(k)
                  NEW_POP(i+1,k)=newgene2(k)
            end do
               else if (CO_TYPE.eq.4) then
*            % crossover4 è il crossover generalizzato Coli-Palazzari
*% ma agisce su due parole da 16bit
*% g1 e g2 sono i due geni che partecipano al cross over
*% sono due vettori riga di 5 elementi x y z fi teta
*% ogni elemento è un uint16

*% parametri del CO
            b=1.05;
            M=int((31*log10(2.0))/log10(b))+1

*% parole su cui si opera
            call RANLUX(y,1)
            j=anint(3*y(1))+1
*% da 1 a 4, perché prende 2 parole
*% cromo sono parole da 32 bit
            cromo1=gene1(j)*65536.0+gene1(j+1)
            cromo2=gene2(j)*65536.0+gene2(j+1)

            call RANLUX(y,1)
            k=anint((M-1)*y(1))+1
            cr=b**k
```

```
            r1=dmod(cromo1,cr)
            r2=dmod(cromo2,cr)
            ncrom1=cromo1-r1+r2
            ncrom2=cromo2-r2+r1
            aa1=ncrom1/65536
            aa2=ncrom2/65536

            w11=anint(aa1)
            w12=mod(ncrom1,65536)
            w21=anint(aa2)
            w22=mod(ncrom2,65536)

            do jj=1,5
              newgene1(jj)=gene1(jj)
              newgene2(jj)=gene2(jj)
            end do

            newgene1(j)=w11
            newgene1(j+1)=w12
            newgene2(j)=w21
            newgene2(j+1)=w22

                do jj=1,5
                  NEW_POP(i,jj)=newgene1(jj)
                  NEW_POP(i+1,jj)=newgene2(jj)
                end do
              end if
            end do
        endif
              end

*--------1---------2---------3---------4---------5---------6---------7-*
      subroutine NEW_ENTRIES
* procedura che genera nuovi arrivi nella popolazione
* sostituisce la MUTAZIONE
* implementa se la flag apposita è alzata, la eliminazione dei cloni
      implicit double precision(a-h,o-z)
      include 'paramGA.inc'
      include 'common_GAdim.inc'

      common/GAdata/IPOP(N_MAX_POP,5),NEW_POP(N_MAX_POP,5),
     +  fitness(N_MAX_POP)
      common/GAdata/fit_temp(N_MAX_POP),best_fit(N_MAX_GEN)
      common/GAdata/igene(5),itempgen(5), AAFCN(6), tempfit
      common/GAdata/sigmar_init, sigmaa_init, INIT_TYPE

      real x, y(5)
      logical NO_CLONE, equa
      real PMUT, pmutemp(N_MAX_POP), indtemp(2)
```

```
integer wordmut, wordgen, indmuw, indmub
newe=1
N_NEW=int(N_POP/10)+1
NO_CLONE=.false.
PMUT=0.02

if (NO_CLONE) then
  do i=1,(N_POP-1)
    do j=(i+1),N_POP
      equa=.true.
      do k=1,5
        if (NEW_POP(i,k).ne.NEW_POP(j,k)) equa=.false.
      end do
      if (equa) then
        call RANLUX(y,5)
        do k=1,5
              NEW_POP(j,k)=65535*y(k)
        end do
      endif
    end do
  end do
endif

if (newe.gt.0) then
  N_BEST=int(N_POP/30)+1
  N_CO=N_POP-N_NEW
  do i=(N_CO+1),N_POP
    call RANLUX(y,5)
    do j=1,5
      NEW_POP(i,j)=65535*y(j)
    end do
  end do

endif
if (pmut.gt.0.0)then
  call RANLUX(pmutemp,N_POP)
  do i=2,N_POP
    if (pmutemp(i).lt.PMUT) then
      call RANLUX(indtemp,2)
      indmuw=nint(4*indtemp(1))+1
      indmub=nint(15*indtemp(2))+1
      wordgen=NEW_POP(i,indmuw)
      wordmut=2**(indmub-1)
      NEW_POP(i,indmuw)=IEOR(wordgen,wordmut)
    endif
  end do
end if
end
```

Funzioni per l'implementazione delle reti neurali

Genera i file di ingresso alla rete neurale, partendo dai file degli eventi

```
load GEOM
n=1.355;
th_C=acos(1/n);
c=0.29979;

load NH_45BF
load GEOM

for k_nhh=1:1000
   if NH_45BF(k_nhh)>19
      evnum=k_nhh;
               eval(['load data/ev_' num2str(evnum)]);
               eval(['load data/T_ev_' num2str(evnum) 'BF']);
               eval(['load data/PMT_ev_' num2str(evnum) 'BF']);
               eval(['ev=ev_' num2str(evnum) ';']);
               eval(['Tempi=T_ev_' num2str(evnum) 'BF;']);
               eval(['PMT=PMT_ev_' num2str(evnum) 'BF;']);

      fiv=ev(4);
      thv=ev(5);

       ind_ev=1;
       P=zeros(80,1);
       T=zeros(2,1);
       N_PMT=length(Tempi);
       n_ev_ev=N_PMT-19;
         for jk=1:n_ev_ev
            for jkk=0:19
               P(4*jkk+1,ind_ev)=(GEOM(PMT(jk+jkk),1)+800)/1600;
               P(4*jkk+2,ind_ev)=(GEOM(PMT(jk+jkk),2)+800)/1600;
               P(4*jkk+3,ind_ev)=(GEOM(PMT(jk+jkk),3)+800)/1600;
               P(4*jkk+4,ind_ev)=(Tempi(jk+jkk)-min(Tempi))/10000;
            end
            T(:,ind_ev)=[fiv/360; thv/180];
            ind_ev=ind_ev+1;
         end
         eval(['P_' num2str(evnum) 'BF=P; save P_' num2str(evnum) 'BF P_' num2str(evnum) 'BF;']);
         eval(['T_' num2str(evnum) 'BF=T; save T_' num2str(evnum) 'BF T_' num2str(evnum) 'BF;']);

   end
end
```

Genera i file di addestramento della rete

```
load NH_45BF
load P_1BF
load T_1BF
Pt=P_1BF;
Tt=T_1BF;

for i=2:500
   if NH_45BF(i)>19
      eval(['load P_' num2str(i) 'BF; load T_' num2str(i) 'BF;'])
      eval(['newP=P_' num2str(i) 'BF; newT=T_' num2str(i) 'BF;'])
      Pt=[Pt newP];
      Tt=[Tt newT];
   end
end
```

Addestramento della rete

```
clear
R2=(ones(80,1)*[0 1]);
SIZ=[25 2];
net=newff(R2, SIZ, {'tansig','satlin'}, 'trainscg');

load Pt
load Tt
P=Pt;
T=Tt;

net.trainParam.epochs=50000;
[nett, TR, YY, EE]=train(net,P,T);
```

Test della rete

```
clear
load NH_45BF

load net45_20
netw=net45_20;

k=1;
for n=501:1000
   if NH_45BF(n)>19
   evnum=n;
eval(['load P_' num2str(evnum) 'BF; P=P_' num2str(evnum) 'BF;']);
eval(['load T_' num2str(evnum) 'BF; T=T_' num2str(evnum) 'BF;']);
Y=sim(netw, P);
E=Y-T;
```