PS/AR Note 94-17 (Min.)

SAP-11

27 June 1994

Minutes of the Forum of

Symbolic Computing for Accelerator Physics

held on Thursday 5 May 1994

Present: Y. Alexahin, B. Autin (Chairman), J. Bosser, F. Caspers, R. Corsini, E. d'Amico, G. Dôme, W. Fischer, R. Giannini, M. Giovannozzi, J.Y. Hemery, Ch. Iselin, J. Jowett (Deputy), J.P. Koutchouk, M. Martini (Secretary), D. Manglunki, C. Metzger, J. Miles, J. Paul, F. Pedersen, W. Remmer, J.C. Schnuriger, G. Shirkov.

---

1 Running MAD via *Mathematica* for dynamic aperture calculation: Y. Alexahin

A *Mathematica* program has been written to analyse the effect of magnet misalignments on LEP dynamic aperture. Using a list of errors (`seeds`), it creates MAD input data for the various lattice configurations (`machine.latt`), and triggers MAD which produces output stored in the file `pooldump`.

Then, it creates a second MAD data set (`track.proto`) and activates MAD to perform particle tracking using the lattice defined in `pooldump` and different initial values of actions, angles and momentum errors. The results are outputted in the `print` file.

The last stage consists of analysing the tracking results. The maximum stable amplitude with the relevant input data are stored in the file `result.out` and eventually plotted in the form of dynamic aperture diagrams.

2 Generating *Mathematica* databases of MAD results, imperfect LEP2 as an example: J.M. Jowett

The information contained in the MAD PRINT files can be converted into functional databases which can be exploited for a variety of applications.

An example is given for an imperfect LEP2 with full RF system where plots and histograms are produced for positron and electron orbits, Twiss parameters, dispersion functions, tune shifts, emittances and damping partition numbers.

An upgrading version of the Notebook is currently in preparation, and will be casted in the form of a standard package.

The next meeting will be held on:

Thursday 7 July at 16.00 hr in the SL Auditorium - Prévessin, Bldg 864, 1st floor

M. Martini

Distribution list

AT, MT, PS and SL Division Leaders and Deputies.
AT, MT, PS and SL Group Leaders and Associates.
SAP list.

Running MAD via Mathematica for Dynamic Aperture Calculation

■ Introduction

Though many analytical methods exist for analysis of particle stability in circular accelerators tracking still remains the most reliable tool. Unfortunately the tracking codes have a limited flexibility not permitting to implement algorithms for purposeful search of the stability boundary (dynamic aperture) - for instance the MAD program [1] does not support nested DO loops and conditionals.

This inconvenience can be circumvented by using Mathematica [2] as a guide for MAD, preparing input for it, analyzing the output and representing the final result. Hereafter an example is given of establishing *unstructured communication* between Mathematica and MAD for statistical analysis of a misaligned accelerator stability properties.

■ Tracking with misalignments

Since the real imperfections can be determined with but a limited accuracy a number of possible error distributions (determined in MAD by the random generator **seed**) should be taken to obtain reliable results. It is particularly important for low emittance (hence highly nonlinear) lattices of the electron (positron) storage rings since small variation in phase advances between sextupoles due to imperfections destroy compensation (if any) of their kicks and substantially reduce the dynamic aperture.

```
seeds=Table[ 10000*i,{i,10}]; nSeeds=Length[ seeds ];
```

■ Auxiliary Files

The program requires two auxiliary files for MAD input preparation:

- **'jobname'.latt** specifying the lattice, misalignments type and closed orbit correction procedure to be studied, where the **'jobname'** stands for a name chosen by the user, and
- **track.proto** formulating the tracking task for MAD.

Both files are assumed to reside in the working directory, The lattice file should be provided by the user, an example can be seen below as an input echo. Output is stored in the file **'jobname'.out.**

```
jobname="lep2"; machine=StringJoin[ jobname,".latt"];
                    results=StringJoin[ jobname,".out"];
```

■ Variables and Units

3D dynamic aperture [3] can be represented as a surface in action space. For coordinates are chosen: the synchrotron oscillation amplitude in the form of fractional momentum deviation **deltap**, angle in the plane of (the square roots of) the transverse action variables (divided by π so that **angle=0** corresponds to purely horizontal and **angle=0.5** - to purely vertical motion), and the maximal stable amplitude **f** (the square root of the sum of action variables taken in $(\pi)\cdot mm\cdot mrad$).

```
deltaps=.001 {0,2.5,5}; nDelts=Length[ deltaps ];
actionPlaneAngles={0.,.25,.5}; nAngles=Length[ actionPlaneAngles ];
```

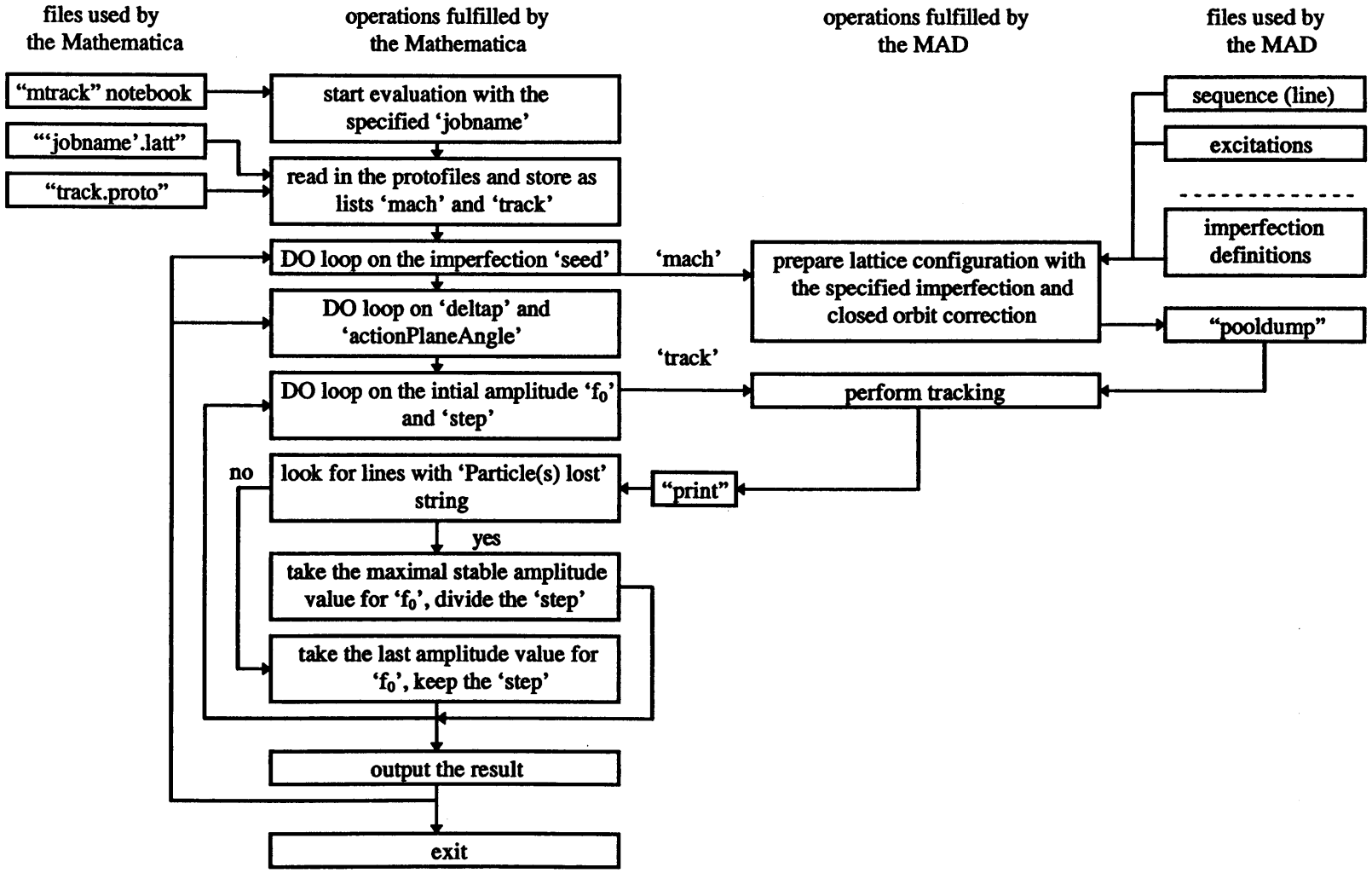


Fig. 1. Flow diagram of the dynamic aperture calculation process

■ Algorithm for finding dynamic aperture

Algorithm for finding dynamic aperture is illustrated by the flow diagram in the Fig. 1. The MAD program is called:

- to assemble the lattice, assign imperfections and correct closed orbit for each value of **seed**, saving the configuration on the file **pooldump**;
- to perform tracking with specified by the Mathematica values of **deltap**, **angle**, number of amplitudes **nampl**, the initial amplitude **f0** and **step.**

At each of the latter calls MAD tracks **nampl·nphase** particles with **nampl** amplitudes incremented by **step** from the initial value **f0**. Number of particles with the same amplitude (but different betatron phases) **nphase** corresponds to the number of the START commands in the **track.proto file.** It is set presently to 3 to avoid mistaking stable resonant islands for the dynamic aperture.

When a good guess can be made on the dynamic aperure size it can be found from the first call (especially with the fastest METHOD= TRANSPORT in the MAD TRACK routine). But with a wide scatter due to imperfections this may require too large a **nampl** (and so the CPU time) to cover the possible range of amplitudes with a **step** rendering the desired accuracy. Therefore the search could be done in a larger number of calls **ncalls** with a decreasing **step.**

```
ncalls=2; nphase=3; nampl=19;
mach=ColumnForm[ ReadList[ machine, Record]]

     option,-warn,-ECHO,-info
     !-- Load the machine configuration
     call,'/users/slath/machines/lep94/lep944.seq'
     call,'/users/slath/machines/135-56/md135-56v1.config'
     call,'/users/alexahin/madtools/track/colli.wdn'
     beam,energy=20
     !-- Switch off sextupoles for primary orbit correction
     set,KSD10 KSD1
     set,KSD20 KSD2
     set,KSD30 KSD3
     set,KSF10 KSF1
     set,KSF20 KSF2
     ksd1=0
     ksd2=0
     ksd3=0
     ksf1=0
     ksf2=0
     use,lep
     !-- Specify imperfections and orbit correction
     EOPT,seed=&
     s
     call,'/users/alexahin/lep2/error.defs'
     error.x=.0005
     error.y=.001
     correct,ncorr=32,iter=2,error=error.x,plane=x
     correct,ncorr=32,iter=2,error=error.y,plane=y
     !-- Switch on the sextupoles
     set,KSD1 KSD10
     set,KSD2 KSD20
     set,KSD3 KSD30
     set,KSF1 KSF10
     set,KSF2 KSF20
     correct,ncorr=32,iter=2,error=error.x,plane=x
     correct,ncorr=32,iter=2,error=error.y,plane=y
     !-- Switch on RF and radiation
```

```
      call,'/users/jowett/lep2/RF/rf0.lep'
      beam,bunched,radiate
      emit
      beam,ex=1.e-6,ey=1.e-6,et=1.e-6
      pooldump
      stop

track=ColumnForm[ ReadList[ "track.proto", Record]]

      option,-warn,-echo
      poolload
      ! Insert start data
      f0=&
      f
      step=&
      s
      angle=&
      a
      delt=&
      d
      set,f,f0
      set,cs,cos(angle*pi)
      set,sn,sin(angle*pi)
      track,rfcavity=aas,damp
      set,ft0,delt/beam[sige]
      do,times=&
      n
      set,f,f+step
      set,fx0,f*cs
      set,fy0,f*sn
      start,ft=ft0,fx=fx0,fy=fy0
      start,ft=ft0,fx=fx0,fy=fy0,phix=pi/4,phiy=5*pi/6
      start,ft=ft0,fx=fx0,fy=fy0,phix=5*pi/6,phiy=pi/4
      enddo
      run,turns=500
      endtrack
      stop

inQ[x_String]:=StringMatchQ[x,"*Particle(s) lost*"];

Do[ seed=seeds[[ iseed ]];
latt=OutputForm[ mach/. "s"->seed]; latt>>"!mad"; Clear[latt];
    Do[ deltap=deltaps[[ idelt ]];
        Do[ angle=actionPlaneAngles[[iang]]; step=.2; f0=0.;
            Do[ task=OutputForm[track/. ["a"->angle, "d"->deltap,
            "f"->f0, "s"->step, "n"->nampl]]; task>>"!mad";
            Clear[task]; mess=ReadList["print", Record];
            npl=Flatten[Position[Map[inQ,mess],True]];
            lnpl=Length[npl]; losp={};
                Do[ (* store lost particles numbers in losp *)
                fw=ToExpression[ First[ ReadList[
                StringToStream[mess[[npl[[i]]+2]] ],Word]]];
                If[ NumberQ[fw], AppendTo[ losp,fw], Continue[]],
                {i,lnpl}]; Clear[mess]; If[ NumberQ[Min[losp]],
            (f0=f0+Floor[ (Min[losp]-1)/nphase]*step; step/=nampl+1),
            f0=f0+nampl*step], {ncalls}];   (* f  found *)
         rest=OpenAppend[results];
         Write[rest,{ seed, deltap, angle, f0}]; Close[ rest],
         {iang,nAngles}],
     {idelt,nDelts}],
{iseed,nSeeds}];

Exit[]
```

References

[1] H.Grote, F.C.Iselin, The MAD Program. Version 8.10. User's Reference Manual. CERN/SL/90-13(AP) (Rev.3), Geneva, 1993.

[2] S.Wolfram, Mathematica: A System for Doing Mathematics by Computer. 2nd ed. (Addison-Wesley Publishing Company, 1993)

[3] J.M.Jowett, Dynamic Aperture for LEP: Physics and Calculations, in Proc. of the Fourth Workshop on LEP Performance, Chamonix, 1994. CERN/SL/94-06(DI).

DEPICTING DYNAMIC APERTURE

```
SetDirectory["/disc/users/alexahin/dynap"];

Off[General::spell,General::spell1];
```

List machine name(s) (in double quotes):

```
mans={"121e5","1s21e5","x21v2e5","m21e5"};
```

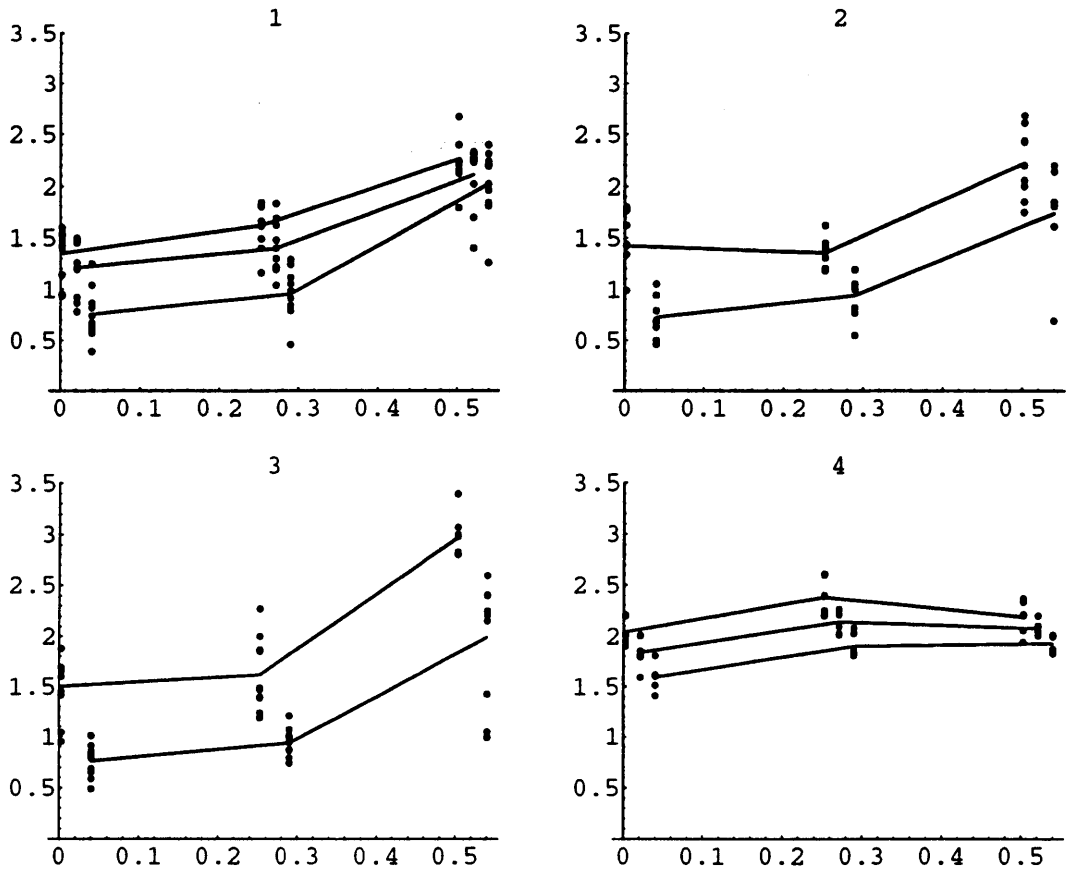
Tracking data file extension:

```
ext="res";
```

Reading data files

```
nmans=Length[mans];

fname[i_Integer]:=StringJoin[{mans[[i]],".",ext}];

data=Table[ ReadList[ fname[i]], {i,nmans}];

xpic[ang_,dpp_]:=.0025+ang+7.5 dpp;

Do[ (*cycle over machines *)
dyndl=data[[imn]]; nls1=Length[dyndl];
scatter={}; Do[ AppendTo[ scatter,
{xpic[dyndl[[i,3]],dyndl[[i,2]]],dyndl[[i,4]]} ],
{i,nls1}];
scPlot=ListPlot[scatter, Prolog->PointSize[.015]];
delts=Union[ Table[dyndl[[i,2]] ,{i,nls1}]];
angles=Union[ Table[dyndl[[i,3]] ,{i,nls1}]];
nangs=Length[angles]; ndlts=Length[delts];
Do[ mnv[idl]={}; Do[ xs=xpic[angles[[ian]],delts[[idl]]];
ns=0; ys=0; Do[ If[ scatter[[i,1]]=!=xs,
Continue[], ns=ns+1]; ys=ys+scatter[[i,2]], {i,nls1}];
If[ ns==0, Continue[], AppendTo[ mnv[idl],{xs,ys/ns}]],
{ian,nangs}],{idl,ndlts}];
Do[ lpt[i]=ListPlot[mnv[i],PlotJoined->True],{i,ndlts}];
coplt[imn]={scPlot}; Do[ AppendTo[coplt[imn],lpt[i]],
{i,ndlts}], {imn,nmans}];

Do[ shpl[imn]=Show[ coplt[imn],Prolog->PointSize[.014],
PlotRange->{0,3.5},AspectRatio->.8,
PlotLabel->ToString[imn]],{imn,nmans}];
```

```
Show[GraphicsArray[{{shpl[1],shpl[2]},
{shpl[3],shpl[4]}}]];
```



Maximal stable betatron amplitude Sqrt[(Ax+Ay)[micron]] vs.Atan[Ay/Ax]/Pi at injection for different values of synchrotron amplitude (dp/p) and seeds of misalignment. Lines join mean values over seeds for equal dp/p values: {0, 0.25%, 0.5%} for figs.[1,4] and {0, 0.5%} for [2,3]. Points corresponding to subsequent dp/p are slightly shifted to avoid overlapping.

[1] - 135/60 lattice, Qx=125.23, Qy=75.18;
[2] - the same with additional sextupoles in midarcs;
[3] - 135/56.25 lattice, Qx=125.26, Qy=71.18 with additional sextupoles;
[4] - 108/60 lattice, Qx=102.27, Qy=76.196.

In all cases quads misaligned to dx=dy=.1 mm, dpsi= .1 mrad, closed orbit corrected to errorx=.5 mm, errory=1. mm.

Extract Data from Output of MAD

John M. Jowett

Introduction

Version information

This notebook was developed bottom-up in the week preceding 4 April 1994.
The master copy of this notebook is on the author's PC: D:\math\madtomma.ma.
Copies of it may exist on Unix machines where MAD has been run. If this is a copy, check when it was made and watch out for application-specific modifications (although these should only occur in the section 'Applications').

Purpose of this notebook

This notebook provides utilities for extracting information from MAD PRINT files. It defines functions which read the main MAD output pages.
A typical application works on a set of files, all sitting in the same directory and resulting from similar MAD jobs, therefore having output of the same form. It scans all these files and extracts interesting data into vectors, one component for each job. These data are built into a database of functions which return vectors of their values.
Knowledge of the structure of the MAD job will guide you in calling them in an appropriate sequence.
The "Applications" section of this notebook contains specialised functions constructed from the generic ones to read the output generated by MAD runs for a particular applications.

This serves as a model application for others which should be kept in separate notebooks to avoid overloading this one. In general, the initialisation cells of this notebook should be evaluated before and everything else can be done in another window.

It is often useful to save the database of functions and read it back into a fresh Mathematica session.

Generic functions

This section defines the functions and other things we need. Their definitions may evolve, depending on needs and interests.

Utility functions

Parsing and string manipulation

The following function is very handy for parsing lines of MAD output and suchlike. It breaks up a string into a list words, then converts each into an expression which will often be a number or a symbol. Various other things give $Failed if their syntax is not right.

```
atomise[str_String]:=ToExpression[ReadList[StringToStream[st
```

The following inserts a string snew before the first occurrence of a substring s in a string sss. If s does not occur in sss then snew is appended to sss.

```
stringInsertBefore[sss_String,s_String,snew_String]:=
        Block[{pos,posl},
                pos=StringPosition[sss,s];
                posl=If[Length[pos]==0,-1,First[First[pos]]]
                StringInsert[sss,snew,posl ]
             ]
```

This one takes the substring before the first occurrence of a substring s in a string sss. Or the whole of sss if s does not occur.

```
stringTakeBefore[sss_String,s_String]:=
        Block[{pos},
                pos=StringPosition[sss,s];
                If[Length[pos]==0,sss,StringTake[sss,First[F
             ]
```

☐ *Functional database generation*

The following functions generate a set of function definitions using the names of objects in a list and a suffix. For each name in the list the definition of a function with the same name concatenated with the suffix is generated.
It is assumed that there will be a list with the name of the list concatenated with the suffix. Then the value of the functions will return the objects in the same place in the second list.
The purpose of the call to stringInsertBefore is to make sure the suffix is added to the names of objects in the list which happen to be functions themselves.

An example to clarify:
if vars is the list {a,b,f[x]}, a call to
accuFunctions[vars,"p"]
will generate the definitions
ap:=varsp[1]
bp:=varsp[2]
fp[x]:=varsp[3]

Something else happens while these definitions are generated. A global string variable varspSAVE (in the example) accumulates the names of the new functions separated by commas. Assuming it was initially empty (as arranged by accuFunctions), it would end up with the value ",ap ,bp, fp[x]". This string is used later by the function accuSave.

```
SetAttributes[accuFunc,HoldFirst];

accuFunc[varbuf_,n_Integer,suffix_String]:=
        Block[{acculine,bufs,newfunc,newfuncname},
                SetAttributes[ToString,HoldAll];
                newfunc=
                   stringInsertBefore[ToString[Evaluate[Part
                                      ,"[",suffix];
                newfuncname=stringTakeBefore[newfunc,"["];
                bufs=ToString[varbuf]<>suffix;
                bufsSAVE=bufs<>"SAVE";
                acculine= newfunc <> " := Part[" <> bufs <>"
                ClearAttributes[ToString,HoldAll];
                acculine=acculine <> ToString[n]<>"]";
                ToExpression[bufsSAVE<>" = " <> bufsSAVE
                                <> "<> \"," <> newfuncname <
                            ];
                ToExpression[acculine]
            ];

SetAttributes[accuFunctions,HoldFirst];
accuFunctions[varbuf_,suffix_String]:=
        Block[{},
                SetAttributes[ToString,HoldAll];
                bufs=ToString[varbuf]<>suffix;
                bufsSAVE=bufs<>"SAVE";
                ClearAttributes[ToString,HoldAll];
                ToExpression[ bufsSAVE <> "=  \" \" " ] ;
                Do[accuFunc[varbuf,n,suffix];
                  ,{n,1,Length[Evaluate[varbuf]]}
                  ]
            ]
```

☐ *Data accumulation in functional database*

Functions which take a current data vector and accumulate its values in a database labelled with a suffix.

First an initialisation function for the appropriate database.

```
SetAttributes[accuInit,HoldFirst];
accuInit[varbuf_,suffix_String]:=
        Block[{acculine},
                SetAttributes[ToString,HoldAll];
                acculine = ToString[varbuf]<>suffix<>
                        " = Table[{},{Length["
                        <>ToString[varbuf]<>"] } ]";
                ClearAttributes[ToString,HoldAll];
                ToExpression[acculine];
            ]
```

The purpose of the function accuData is to take the current set of values of varbuf and use them to extend the set of values accumulated in varbuf<>suffix.

Example: if vars was defined through vars:={a,b,f[x]} and currently a=1, b=2, f[x]=3, then a call to

accuInit[vars,"p"]

would define varsp={{},{},{}}. Then a call

accuData[vars,"p"]

would set varsp={{1},{2},{3}}. Then, if the values meanwhile change to a=4, b=5, f[x]=6, a further call to

accuData[vars,"p"]

would set varsp={{1,4},{2,5},{3,6}}.

```
accuDatavar[accuvar_,var_]:=
        If[Length[accuvar]==Length[var],
           MapThread[Append,{accuvar,var}],
           accuDatavar::mismatch="Mismatched data not accumu
           Message[accuDatavar::mismatch];accuvar
        ]
SetAttributes[accuData,HoldFirst];
accuData[varbuf_,suffix_String]:=
        Block[{acculine},
                SetAttributes[ToString,HoldAll];
                acculine = ToString[varbuf]<>suffix<>
                        " = accuDatavar[" <>
                        ToString[varbuf]<>suffix <>"," <>
                        ToString[varbuf] <> "]";
                ClearAttributes[ToString,HoldAll];
                ToExpression[acculine];
            ]
```

☐ *Saving the functional database*

This function uses the string of names of functions defined as a side-effect of accuFunctions to save the functions and everything they depend on in a file.

```
SetAttributes[accuSave,HoldRest];
accuSave[filename_String,varbuf_,suffix_String]:=
        Block[{bufs,bufsSAVE},
                SetAttributes[ToString,HoldAll];
                bufs=ToString[varbuf]<>suffix;
                bufsSAVE=bufs<>"SAVE";
                saveline="Save[\"" <> filename <> "\""<>
                        ToExpression[bufsSAVE]<>"]";
                ClearAttributes[ToString,HoldAll];
                ToExpression[saveline]
            ]
```

Reading EMIT output of global machine parameters

Here is a list of quantities provided by MAD's EMIT command in which we are interested. Evaluating it provides a buffer with the current values. Immediately, we should generate a list of functions which allow the recovery of lists of these quantities from the accumulation of the above list (see below).

```
Off[General::spell,General::spelll];
varEMIT:=[Ebeam,U0,alphac,Q1,Q2,Q3,gammapt,J1,J2,J3,emittl,e
On[General::spell,General::spelll]
```

Here is a function whose job is to get these quantities from the next EMIT output page for the particle type specified. Normally this will be a string "POSITRONS" or "ELECTRONS".
Along the way we get the gamma_pt which we can use to convert longitudinal emittance to energy spread with the formula sige=Sqrt[gamma_t Es].
The transverse emittances are converted to nm.

We are careful to set all the global variables to Null in cases where the EMIT page is not found.
Note also the use of —— (the SameQ function) rather than just — to test that strings are really the same. Using — sometimes returns neither True nor False since the relation is treated algebraically.
The value of this function is not important. What matters are its side effects of assigning values to a set of global variables. This is not terribly object-oriented, I know, but is a convenient and fairly efficient way to program the problem. It is up to the user to do something with the values after each call of the function.

```
readEMIT[prstr_InputStream,particle_String]:=
If[Find[prstr,"Global parameters for "<>particle]===EndOfFil
  ,Block[{},
         {Ebeam,U0,alphac,Q1,Q2,Q3,gammapt,J1,J2,J3,emittl,e
                =Table[Null,{Length[varEMIT]}];
         readEMIT::eof="EMIT data not found.";
         Message[readEMIT::eof]
        ]
  ,Block[{},
        Off[ToExpression::esntx];
        Find[prstr,"f0"];
        {junk,alphac}=Read[prstr,{Word,Real}]; (* number in
        Find[prstr,"Bcurrent"];
        Ebeam=First[Take[atomise[Read[prstr,Record]],{2,2}]]
        U0=First[Take[atomise[Find[prstr," U0 "]],{2,2}]] Me
        Find[prstr,"Fractional tunes"];
        {Q1,Q2,Q3}=Take[atomise[Read[prstr,Record]],{2,4}];
        Find[prstr,"gamma(max) [1/m]"];
        gammapt=First[Take[atomise[Find[prstr,"pt"]],{4}]];
        {J1,J2,J3}=Take[atomise[Find[prstr,"Damping partitic
                                ],{4,6}];
        Off[General::spell,General::spelll];
        {emittl,emitt2,emitt3}=Take[atomise[Find[prstr,"Emit
        On[General::spell,General::spelll];
        On[ToExpression::esntx];
        sige=Sqrt[gammapt emitt3 10^-6];
        emittl=10^3 emittl nm;emitt2=10^3 emitt2 nm;
        Print["EMIT output for ",particle," on ",First[prstr
       ]
 ]
```

Reading TWISS3 output

First it is necessary to define the marker points where we will look.

```
markers={"IP2","IP4","IP6","IP8"};
Off[General::spell,General::spelll];
Clear[x,y,pt,betxl,bety2,mul,mu2,mu3];
varTWISS3:=Flatten[{Map[x,markers],Map[y,markers],Map[pt,mar
                Map[betxl,markers],Map[bety2,markers],
                Map[mul,markers],Map[mu2,markers],Map[mu3,ma
                  }]
On[General::spell,General::spelll];
```

This function reads the next Twiss3 output of Mais-Ripken functions. As argument, it takes a list of elements which should have been selected (by the PRINT command) in MAD for output of the values. It then sets up a Mathematica database of these values by implementing values for the functions tw1, tw2 and tw3. Note that these values change each time the function is called.

```
readTWISS3[prstr_InputStream,markers_List]:=
If[Find[prstr,"Mais-Ripken"]===EndOfFile
  ,Block[{},readTWISS3::eof="TWISS3 data not found.";
        Message[readTWISS3::eof];
        Do[orbit[ markers[[i]] ]=Table[Null,{i,5,10}];
           tw1[ markers[[i]] ]=Table[Null,{i,2,11}];
           tw2[ markers[[i]] ]=Table[Null,{i,2,11}];
           tw3[ markers[[i]] ]=Table[Null,{i,2,11}];
           ,{i,Length[markers]}
          ]
        ]
  ,Block[{posTwiss3},posTwiss3=StreamPosition[prstr];
        Do[SetStreamPosition[prstr,posTwiss3];
           orbit[ markers[[i]] ]=Take[atomise[Find[prstr,mar
           tw1[ markers[[i]] ]=Take[atomise[Read[prstr,Recor
           tw2[ markers[[i]] ]=Take[atomise[Read[prstr,Recor
           tw3[ markers[[i]] ]=Take[atomise[Read[prstr,Recor
           ,{i,Length[markers]}
          ]
        ];
        Print["TWISS3 output on ",First[prstr]," read succes
  ]
```

☐ *Functions which extract information from the Twiss3 database.*

Actual execution of these definitions is suspended until all calls to accuFunctions are done with since, if these definitions were already active, substitutions would be made in varTWISS3.

```
Off[General::spell,General::spell1];
dbTWISS3:=
Block[{},
        (*components of the orbit with appropriate units*)
        x[marker_String]:=orbit[marker] [[1]] mm;
        px[marker_String]:=orbit[marker] [[2]]10^-3;
        y[marker_String]:=orbit[marker] [[3]] mm;
        py[marker_String]:=orbit[marker] [[4]]10^-3;
        t[marker_String]:=orbit[marker] [[5]] mm;
        pt[marker_String]:=orbit[marker] [[6]]10^-3;
        (*phase advances of the three normal modes*)
        mu1[marker_String]:=First[tw1[marker]];
        mu2[marker_String]:=First[tw2[marker]];
        mu3[marker_String]:=First[tw3[marker]];
        (*diagonal beta-functions.
          We do not yet implement the other 9-3=6 components
        betx1[marker_String]:=tw1[marker][[2]] m;
        bety2[marker_String]:=tw2[marker][[5]] m;
        bett3[marker_String]:=tw3[marker][[8]] m;
      ];
On[General::spell,General::spell1];
```

Reading TWISS output

```
markers={"IP2","IP4","IP6","IP8"};
Off[General::spell,General::spelll];
Clear[Dx,Dy];
varTWISS:=Flatten[
{Qx,Qy,deltas,chromx,chromy,betaxmax,betaymax,Dxmax,Dymax,Dx
 xcomax,ycomax,xcorms,ycorms,
 Map[Dx,markers],Map[Dy,markers]
}];
On[General::spell,General::spelll];
```

This function is like readTWISS3 but a bit simpler since there is only one line per element. However it also picks up the global parameters at the end of the TWISS output.

```
readTWISS[prstr_InputStream,markers_List]:=
If[Find[prstr,"Linear lattice functions.    TWISS"]===EndOfF
  ,Block[{},readTWISS::eof="TWISS data not found.";
        Message[readTWISS::eof];
        Do[tw[ markers[[i]] ]=Table[Null,{i,5,18}];
           ,{i,Length[markers]}
          ];
        {Qx,Qy,deltas,chromx,chromy,
         betaxmax,betaymax,Dxmax,Dymax,Dxrms,Dyrms,
         xcomax,ycomax,xcorms,ycorms
        }=Table[Null,{15}];
        ]
  ,Block[{posTwiss,tuneline, chromline, alfaline, gammline,
          dxrmline, xmaxline, xrmsline}
        ,posTwiss=StreamPosition[prstr];
        Do[SetStreamPosition[prstr,posTwiss];
           tw[ markers[[i]] ]=Take[atomise[Find[prstr,marker
           ,{i,Length[markers]}
          ];
        Off[ToExpression::esntx];
        tuneline=atomise[Find[prstr,"total length"]];
        chromline=atomise[Find[prstr,"delta(s)"]];
        alfaline=atomise[Find[prstr,"alfa          ="]];
        gammline=atomise[Find[prstr,"gamma(tr)"]];
        dxrmline=atomise[Find[prstr,"Dx(r.m.s.)"]];
        xmaxline=atomise[Find[prstr,"xco(max)"]];
        xrmsline=atomise[Find[prstr,"xco(r.m.s.)"]];
        On[ToExpression::esntx];
        Qx=tuneline[[7]];Qy=tuneline[[10]];
        deltas=chromline[[3]];chromx=chromline[[7]];chromy=c
        betaxmax=alfaline[[6]] m;betaymax=alfaline[[9]] m;
        Dxmax=gammline[[6]] m;Dymax=gammline[[9]] m;
        Dxrms=dxrmline[[3]] m;Dyrms=dxrmline[[6]] m;
        xcomax=xmaxline[[3]] mm;ycomax=xmaxline[[6]] mm;
        xcorms=xrmsline[[3]] mm;ycorms=xrmsline[[6]] mm;
        Print["Twiss output on ",First[prstr]," read success
        ]
  ]
```

☐ *Functions which extract information from the Twiss database.*

First the components of the orbit with appropriate units. See notes preceding definition of dbTWISS3.

```
Off[General::spell,General::spelll];
dbTWISS:=
Block[{},
        betx[marker_String]:=tw[marker][[1]] m;
        alfx[marker_String]:=tw[marker][[2]];
        mux[marker_String]:=tw[marker][[3]];
        xco[marker_String]:=tw[marker] [[4]] mm;
        pxco[marker_String]:=tw[marker] [[5]] 10^-3;
        Dx[marker_String]:=tw[marker][[6]] m;
        Dpx[marker_String]:=tw[marker][[7]];
        bety[marker_String]:=tw[marker][[8]] m;
        alfy[marker_String]:=tw[marker][[9]];
        muy[marker_String]:=tw[marker][[10]];
        yco[marker_String]:=tw[marker] [[11]] mm;
        pyco[marker_String]:=tw[marker] [[12]] 10^-3;
        Dy[marker_String]:=tw[marker] [[13]] m;
        Dpy[marker_String]:=tw[marker] [[14]];
    ];
On[General::spell,General::spelll];
```

■ A Model Application

An applications of the generic functions defined in the preceding section. A good way to construct other applications may be to copy this section into another notebook and modify it.

■ Simulation of imperfect LEP and separations at the IPs

This first example of a specialised data accumulation function was designed for the MAD jobs arising from the file
romeo:/users/jowett/lep2/ipsep.mad.
It should serve as a model for other applications.

☐ *Function definitions for this application*

The form of the functions in this section depends strongly on the original input to MAD. It needs to be rewritten for different applications. The most important functions opens a MAD print file and reads data (it will often apply the generic output page-reading functions in an appropriate order). For each file, it appends the interesting data to lists which are initialised in the preceding subsection. Other functions perform appropriate initialisations and save the results.

Generate definitions of functions with suffixes for this application. The following cell MUST BE EXECUTED before any call to functions such as readEMIT or the database function definitions contained in dbTWISS3. Since these assign values to the names in the lists like varEMIT they can only be executed afterwards.
It follows that it can be executed only once per Mathematica session.

```
Off[General::spell,General::spell1];
accuFunctions[varEMIT,"p"];accuInit[varEMIT,"p"];
accuFunctions[varEMIT,"e"];accuInit[varEMIT,"e"];
accuFunctions[varTWISS3,"p"];accuInit[varTWISS3,"p"];
accuFunctions[varTWISS3,"e"];accuInit[varTWISS3,"e"];
accuFunctions[varTWISS,"p"];accuInit[varTWISS,"p"];
accuFunctions[varTWISS,"e"];accuInit[varTWISS,"e"];
dbTWISS3;dbTWISS;
On[General::spell,General::spell1];
initialiseIPSEPdata:=
  Block[{},
                dirIPSEP=Directory[];
                fileIPSEP={};
                accuInit[varEMIT,"e"];
                accuInit[varEMIT,"p"];
                accuInit[varTWISS,"p"];
                accuInit[varTWISS,"e"];
                accuInit[varTWISS3,"p"];
                accuInit[varTWISS3,"e"];
        ]
accumulateIPSEPdata[filename_String]:=
   Block[{prstr},prstr=OpenRead[filename];
         markers={"IP2","IP4","IP6","IP8"};
         Skip[prstr,Character,1];MADTitle=Read[prstr,Record]
         Print[MADTitle];

         AppendTo[fileIPSEP,First[prstr]];

         readEMIT[prstr,"POSITRONS"];accuData[varEMIT,"p"];
         readTWISS3[prstr,markers];accuData[varTWISS3,"p"];
         readTWISS[prstr,markers];accuData[varTWISS,"p"];

         readEMIT[prstr,"ELECTRONS"];accuData[varEMIT,"e"];
         readTWISS3[prstr,markers];accuData[varTWISS3,"e"];
         readTWISS[prstr,markers];accuData[varTWISS,"e"];

         Close[prstr];
        ]
```

Saving the accumulated data: This appends all the interesting accumulated data in a file whose name is given as argument. The lists saved can later be restored with Get[filename]. Include the functions which extract data from the main arrays.

```
saveIPSEPdata[filename_String]:=
        Block[{},
                Save[filename,dirIPSEP,fileIPSEP,markers];

                accuSave[filename,varEMIT,"p"];
                accuSave[filename,varTWISS3,"p"];
                accuSave[filename,varTWISS,"p"];
                accuSave[filename,varEMIT,"e"];
                accuSave[filename,varTWISS3,"e"];
                accuSave[filename,varTWISS,"e"];
            ]
```

☐ *Define working directory [SYSTEM-DEPENDENT]*

PC version (after suitable ftp to change file names):

```
SetDirectory["G:\home\jowett\lep2\ipsep\case6"];
prtFiles=FileNames["*.prt"]
```

SL-AP group HP network version:

```
SetDirectory["/users/jowett/lep2/ipsep/case6"];
prtFiles=FileNames["print.*"]
```

PaRC version:

```
SetDirectory["/home/si/jowett/lep2/ipsep/case6"];
prtFiles=FileNames["print.*"]
```

☐ *Extract data from a set of files*

First we should initialise our lists.

```
initialiseIPSEPdata
```

```
Part::partw: Part 6 of tw[IP2] does not exist.
Part::partw: Part 6 of tw[IP4] does not exist.
Part::partw: Part 6 of tw[IP6] does not exist.
General::stop:
   Further output of Part::partw
     will be suppressed during this calculation.
```

Now look through all (or a selection of) the files in this project, extracting the data from each one and accumulating it in lists.

```
Scan[accumulateIPSEPdata,prtFiles]
```

```
General::spell1:
   Possible spelling error: new symbol name "gamma"
     is similar to existing symbol "Gamma".
General::spell:
   Possible spelling error: new symbol name "beta"
     is similar to existing symbols {Beta, betx, bety}.
General::spell1:
   Possible spelling error: new symbol name "partition"
     is similar to existing symbol "Partition".
General::spell1:
   Possible spelling error: new symbol name "length"
     is similar to existing symbol "Length".
General::stop:
   Further output of General::spell1
     will be suppressed during this calculation.
General::spell:
   Possible spelling error: new symbol name "alfa"
     is similar to existing symbols {alfx, alfy}.
General::spell:
   Possible spelling error: new symbol name "betax"
     is similar to existing symbols {beta, betx}.
General::stop:
   Further output of General::spell
     will be suppressed during this calculation.
```

Save the database (after deleting old version, if any, otherwise you would accumulate more definitions in the same file).

```
DeleteFile["ipsepd.m"]
```

```
saveIPSEPdata["ipsepd.m"]
```

Look at the contents of the database (tends to be long!):

```
!!ipsepd.m
```

Imperfect LEP2 with Full RF System

John M. Jowett

Case to be treated

Set up database

```
SetDirectory["D:\lep2\ipsep\case6"]
```

```
D:\LEP2\IPSEP\CASE6
```

```
!!readme.txt
```

```
                                                              .LOG
 ------------------------------------------------------------
    17:12:20 DFT, Monday 04/04/94 [Day 094, Week 14 of 1994]
                      Current directory and name of this file:
                              /home/si/jowett/lep2/ipsep/case6
                                                        README

   First batch of imperfect machines with ideal rf0.lep and
                                                    no pretzel
                  Correcting orbit with sextupoles off first.

 ------------------------------------------------------------
12:19:25 METDST, Tuesday 05/04/94 [Day 095, Week 14 of 1994]
                      Current directory and name of this file:
                           /disc/users/jowett/lep2/ipsep/case6
                         /users/jowett/lep2/ipsep/case6/README


  Just transferred these from PaRC for Mma processing.  I keep
ıning up against a stupid CPU limit for interactive commands on
                                                    the PaRC.

                                                   12:50  05/04/94
a database created on romeo using madtomma.ma, now transferred
                                                 to PC for analysis.
```

```
FileDate["ipsepd.m"]
```

```
{1994, 4, 5, 22, 57, 24}
```

Read in the database:

```
<<ipsepd.m
```

Directory in which this database was created:

```
dirIPSEP
```

```
/tmp_mnt/nfs/romeo/users/jowett/lep2/ipsep/case6
```

Function Definitions

Useful functions, loads of standard packages, option settings etc. for this and similar notebooks are in this file.

The command should be executed only once per session.

```
Get["D:\lep2\ipsep\ipsep.m"]
```

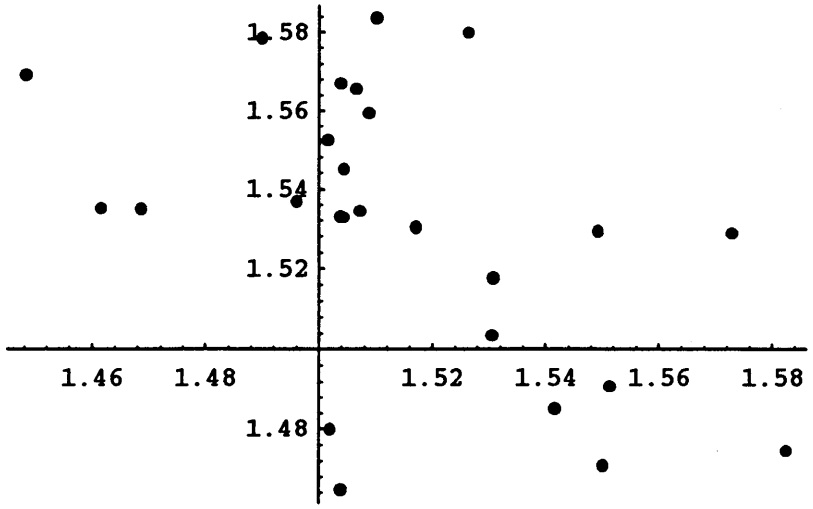
In the following sections, we make plots of a anumber of quantities from the database.
In this application, each data point corresponds to an initial random SEED value in MAD and which therefore generates a different random machine. For each such machine, quantities corresponding to both electrons and positrons are computed and compared with each other.

■ Global Orbits and Dispersion

□ Scatter Plots

Horizontal global RMS orbits for positrons and electrons.

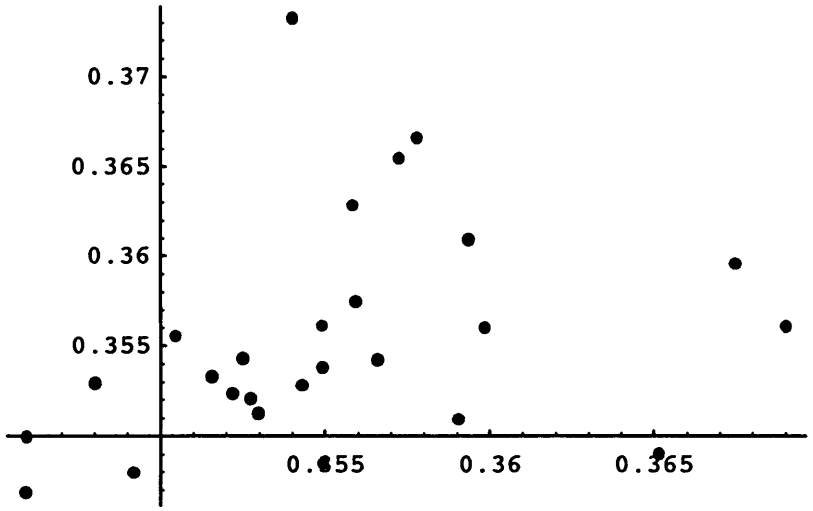
```
corrPlot[xcormsp/mm, xcormse/mm]
```




```
-Graphics-
```

Vertical global RMS orbits for positron and electron.

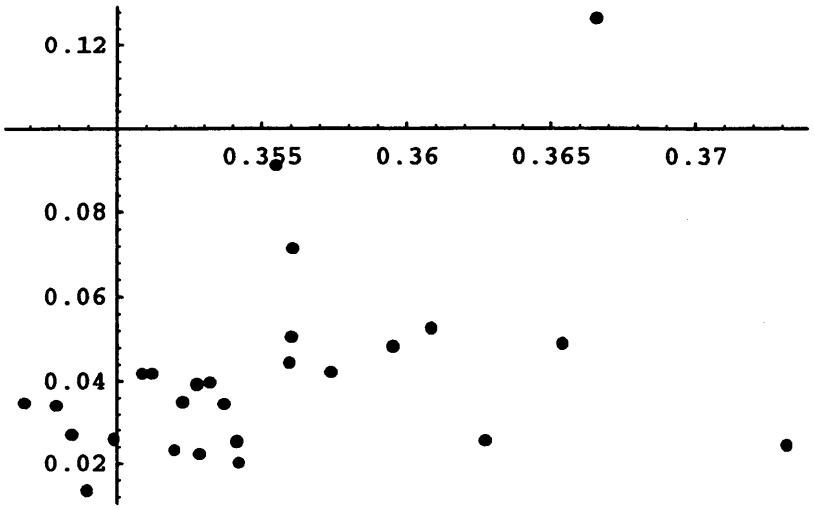
```
corrPlot[ycormsp/mm, ycormse/mm]
```




```
-Graphics-
```

No very obvious correlation between vertical orbit and dispersion.

```
corrPlot[Dyrmsp/m,ycormsp/mm]
```



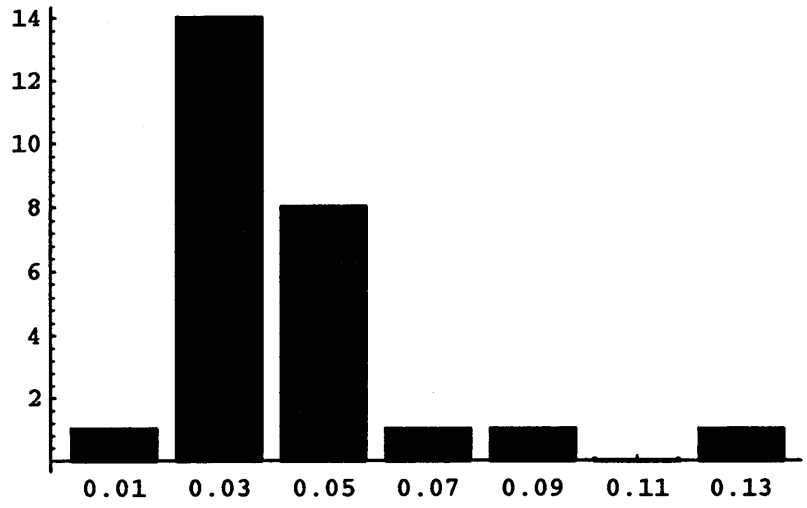

```
-Graphics-
```

☐ Statistical distributions

Distribution of RMS dispersion over ensemble of machines.

```
histogram[Dyrmsp/m,0.02]
```

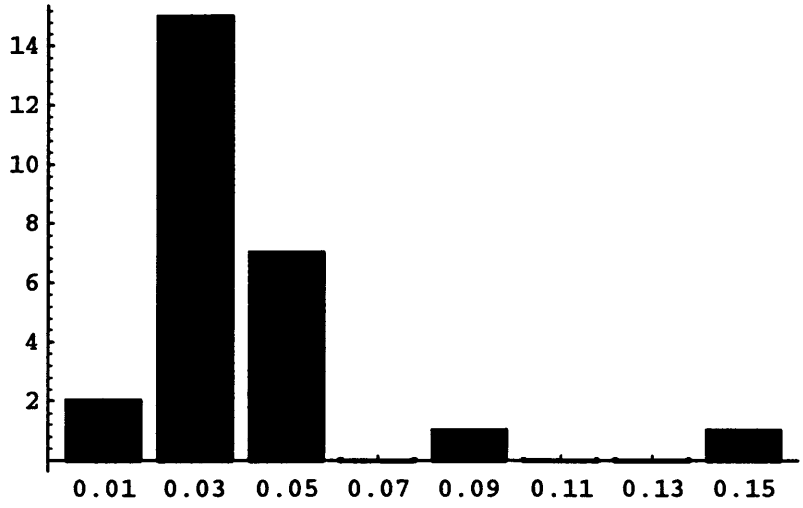
```
                                    Mean= 0.0415077
                    Standard Deviation (MLE) =0.0234531
```




```
-Graphics-
```

```
histogram[Dyrmse/m,0.02]
```

Mean= 0.041363
Standard Deviation (MLE) =0.0259913

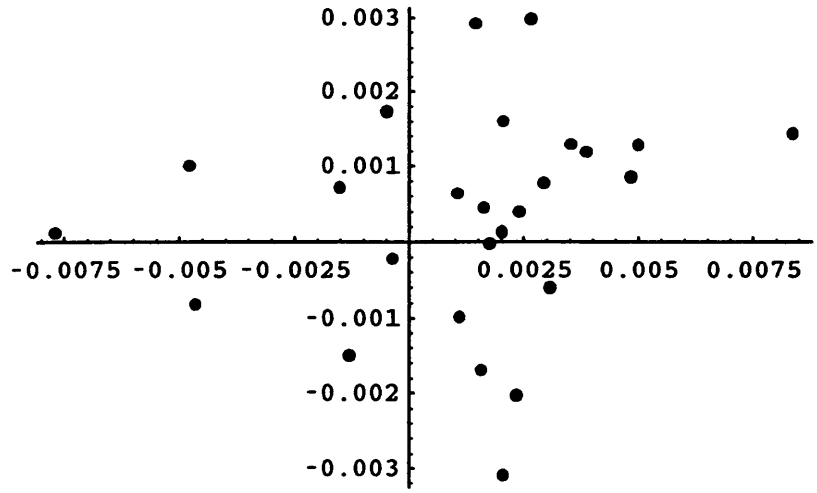


-Graphics-

Tunes and tune-splits

Positron-electron tune splits for the first two normal modes (corresponding approximately to horizontal and vertical betatron motion).

```
corrPlot[Q2p-Q2e,Q1p-Q1e]
```

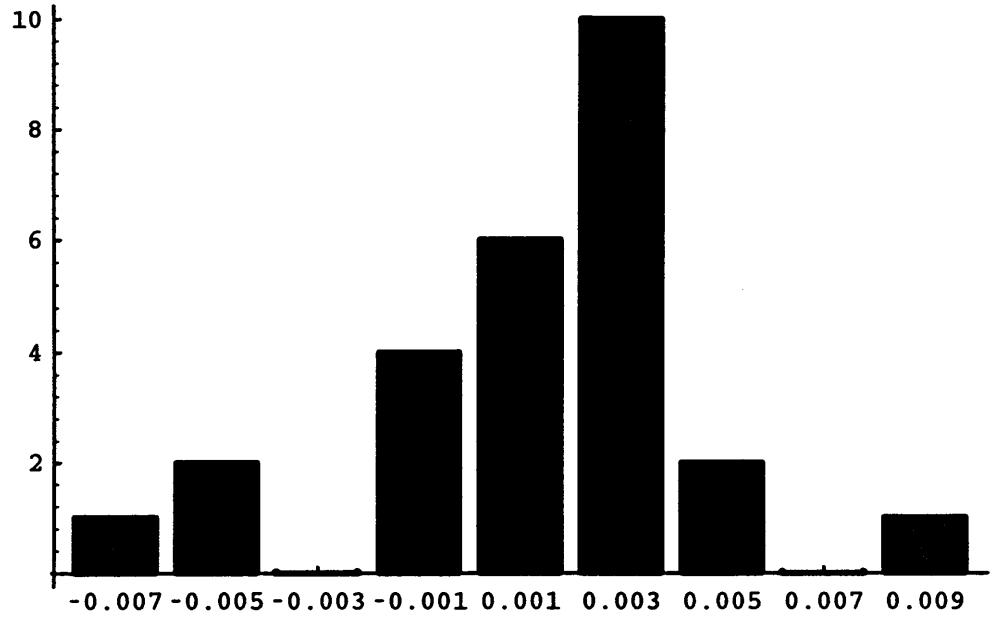



```
-Graphics-
```

Distribution of tune splits

```
histogram[Q1p-Q1e,0.002]
```

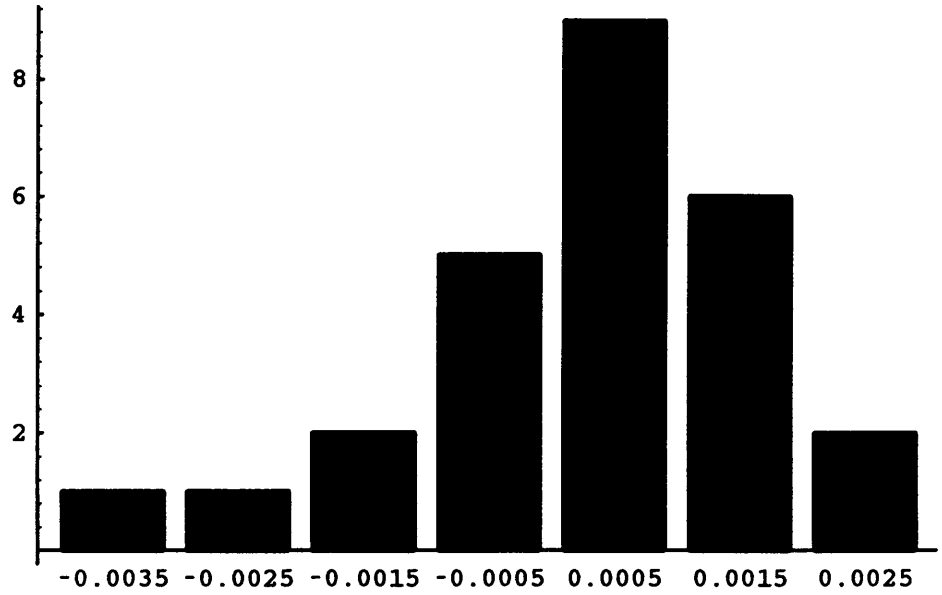
```
                                    Mean= 0.00127494
                    Standard Deviation (MLE) =0.00324407
```




```
-Graphics-
```

```
histogram[Q2p-Q2e,0.001]
```

```
                                    Mean= 0.000319108
                    Standard Deviation (MLE) =0.00140771
```

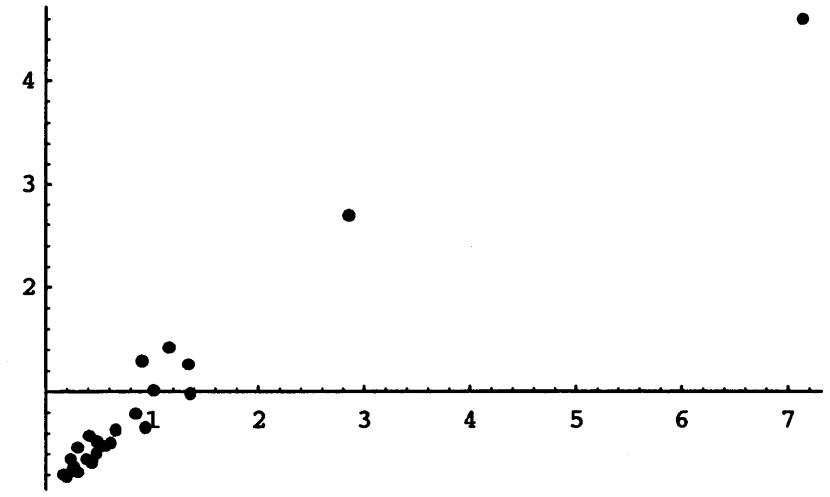



```
-Graphics-
```

Emittances and damping partitions

Vertical emittances of positron and electron. Note that these are machines without solenoids or skew quads.

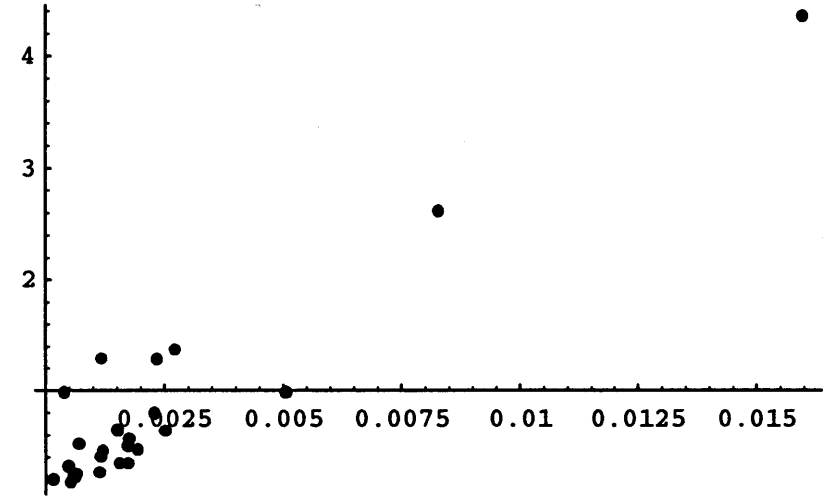
```
corrPlot[emitt2p/nm,emitt2e/nm]
```




```
-Graphics-
```

Correlation between vertical emittance times damping partition, i.e., the vertical quantum excitation, and mean-square dispersion looks quite good:

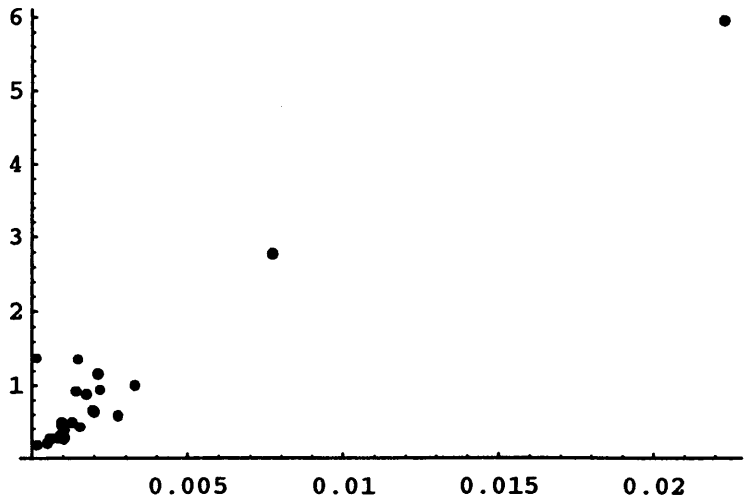
```
corrPlot[emitt2p/nm J2p,(Dyrmsp/m)^2 ]
```




```
-Graphics-
```

Same thing for the electrons:

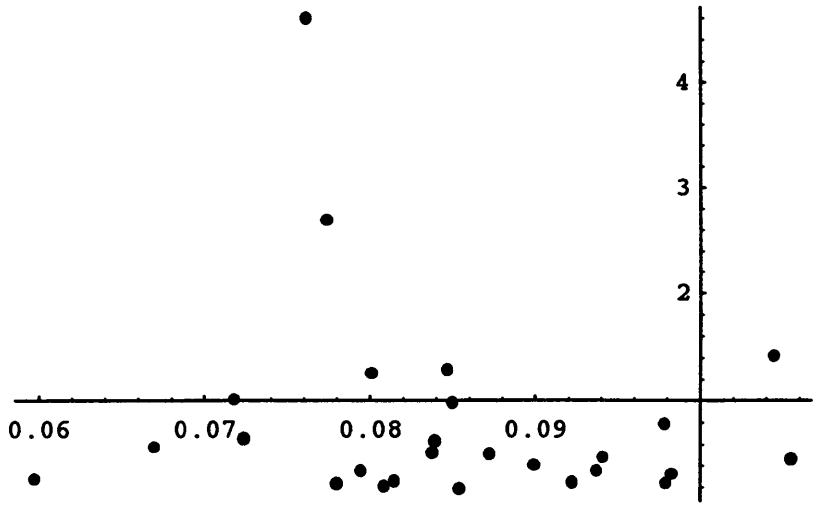
```
corrPlot[emitt2e/nm J2e,(Dyrmse/m)^2 ]
```




```
-Graphics-
```

On the other hand the correlation with the distance from the coupling resonance is not. We are too far away for that to be relevant.

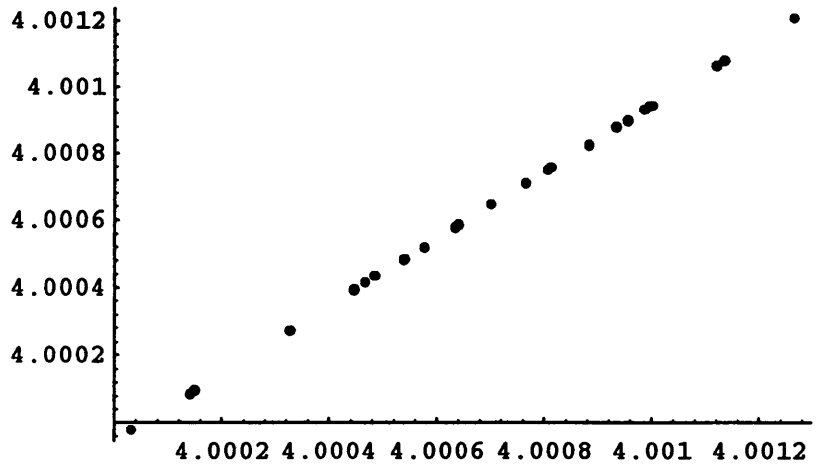
```
corrPlot[emitt2p/nm,Q1p-Q2p]
```




```
-Graphics-
```

Check how well Robinson's theorem is satisfied for the two beams. All the points should fallbang on on {4,4}.

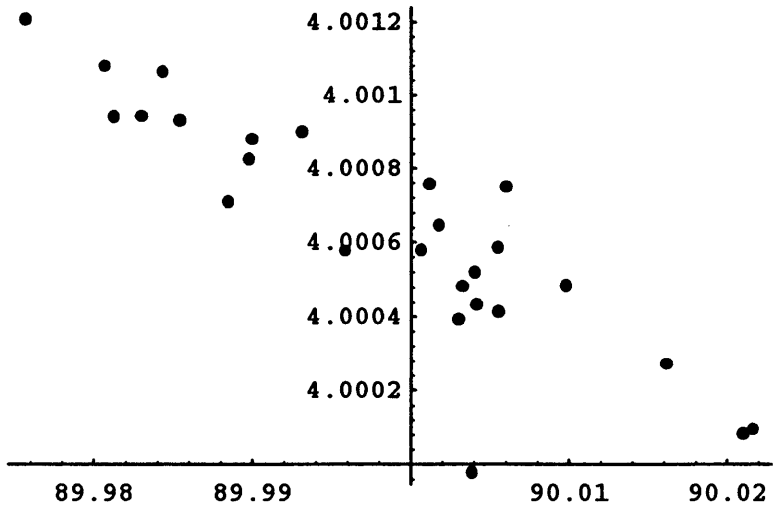
```
corrPlot[J1p+J2p+J3p,J1e+J2e+J3e]
```




```
-Graphics-
```

There is an anti-correlation between the sum of damping rates and the actual average beam energy (shown here in GeV), sugggesting that the definition of damping partition in MAD needs a slight adjustment.

```
corrPlot[(J1p+J2p+J3p),(1+(ptp[IP4]+pte[IP4]))*90]
```

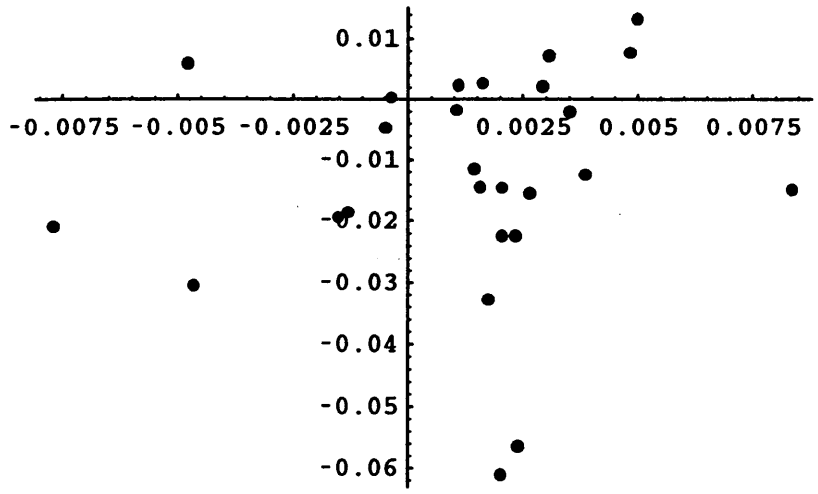



```
-Graphics-
```

Separations and energy sawtooth at the IPS

Separations at the IPs in mm, plotted against horizontal tune-split.

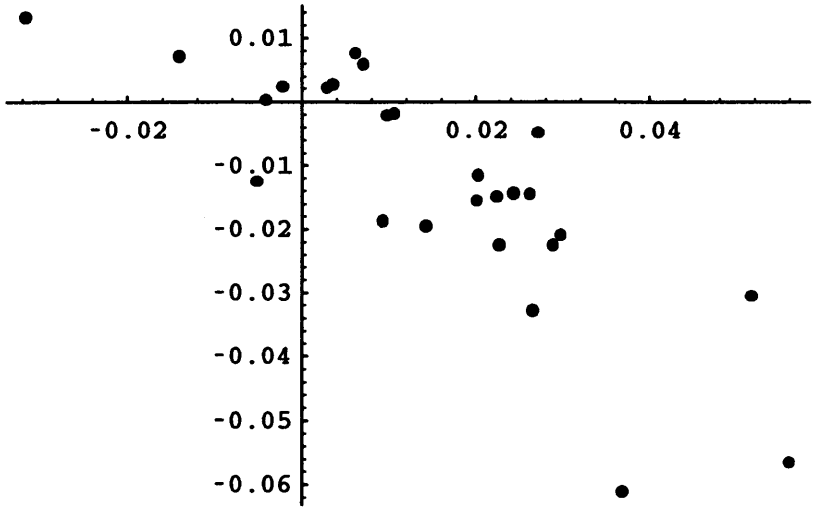
```
corrPlot[(xp[IP2]-xe[IP2])/mm,Q1p-Q1e]
```




```
-Graphics-
```

Correlation between separations at two IPs

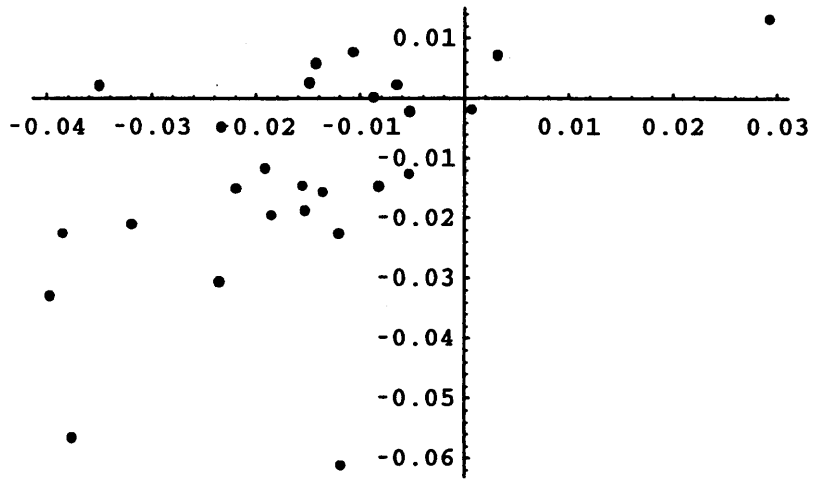
```
corrPlot[(xp[IP2]-xe[IP2])/mm,(xp[IP4]-xe[IP4])/mm]
```




```
-Graphics-
```

And another two IPs:

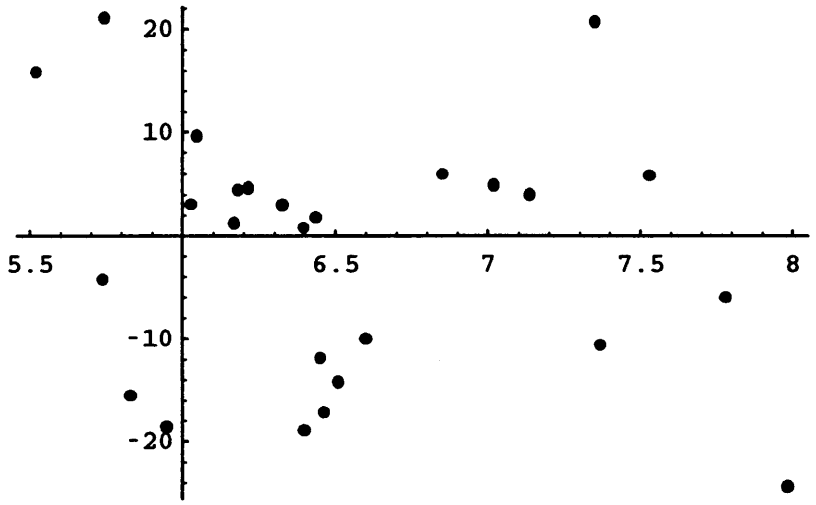
```
corrPlot[(xp[IP2]-xe[IP2])/mm,(xp[IP6]-xe[IP6])/mm]
```




```
-Graphics-
```

Centre of mass energy correlated with xcorms for e+

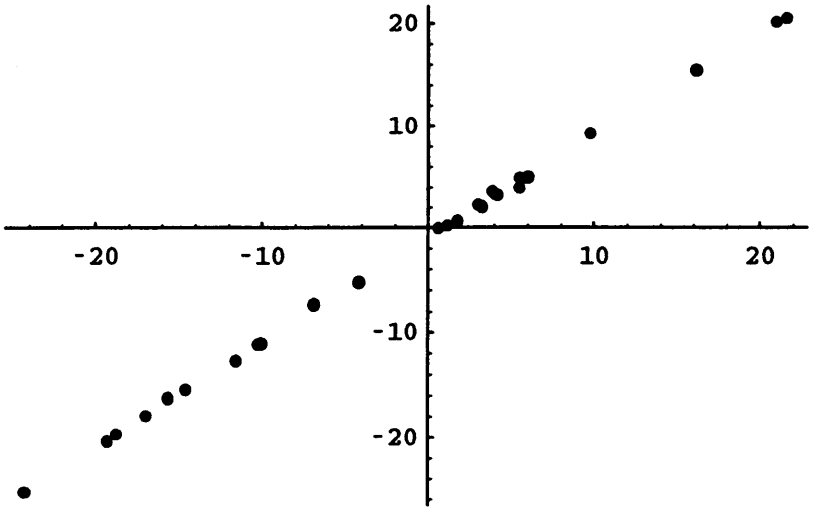
```
corrPlot[ (ptp[IP8]+pte[IP8]) 90 GeV (1000 MeV/GeV)/MeV,
          xcomaxp/mm]
```




```
-Graphics-
```

Deviations of centre of mass energy at two different IPs arenearly the same (thankfully!). They are shown here in MeV. However there can be quite a variation in centre-of-mass energy with different imperfect machines.
It would be nice to try to correlate these with the corrector strengths in use.

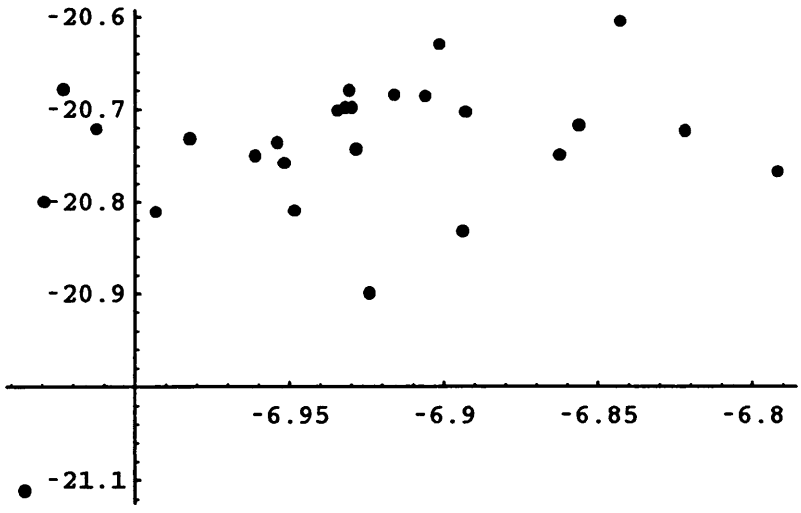
```
corrPlot[ (ptp[IP2]+pte[IP2]) 90 GeV (1000 MeV/GeV)/MeV,
          (ptp[IP4]+pte[IP4]) 90 GeV (1000 MeV/GeV)/MeV]
```




```
-Graphics-
```

Differences between the energies of e+ and e- at the IPs.

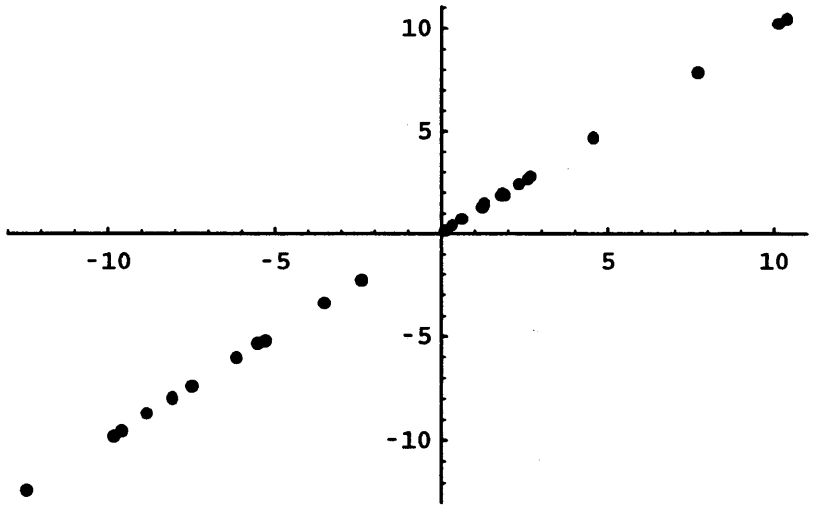
```
corrPlot[ (ptp[IP2]-pte[IP2]) 90 GeV (1000 MeV/GeV)/MeV,
          (ptp[IP4]-pte[IP4]) 90 GeV (1000 MeV/GeV)/MeV]
```




```
-Graphics-
```

Differences in the average energies of e+ and e- around the ring:

```
corrPlot[ (ptp[IP2]+ptp[IP4]+ptp[IP6]+ptp[IP8])/4 90 GeV (1000 Me
          (pte[IP2]+pte[IP4]+pte[IP6]+pte[IP8])/4 90 GeV (1000 Me
```




```
-Graphics-
```

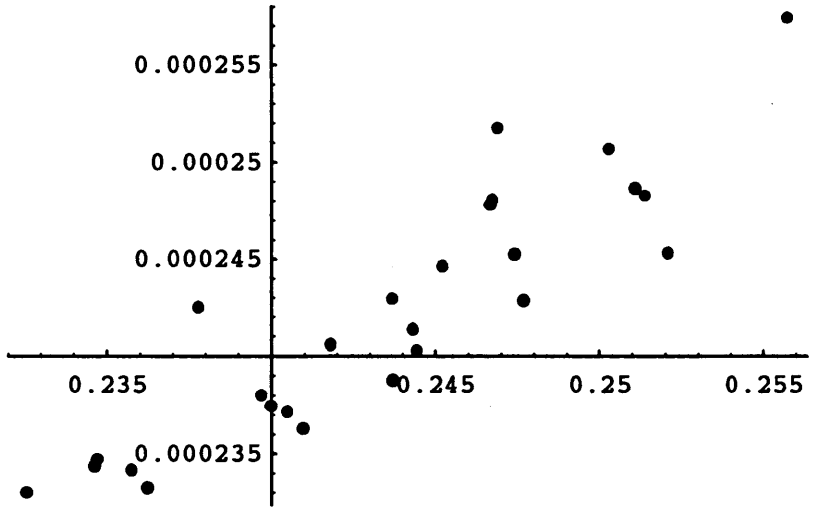
I.e. these are perfectly equal but there is a spread from one machine to another.

Functions to return the horizontal beam size:

```
sigxp[marker_]:=PowerExpand[ Sqrt[10^-9 m/nm emittlp betxlp[marke
                        Dxp[marker]^2 sigep^2]
                    ]
sigxe[marker_]:=PowerExpand[ Sqrt[10^-9 m/nm emittle betxle[marke
                        Dxe[marker]^2 sigee^2]
                    ]
```

Beam sizes in mm at an IP:

```
corrPlot[sigxp[IP2]/m,sigxe[IP2]/m 10^3]
```



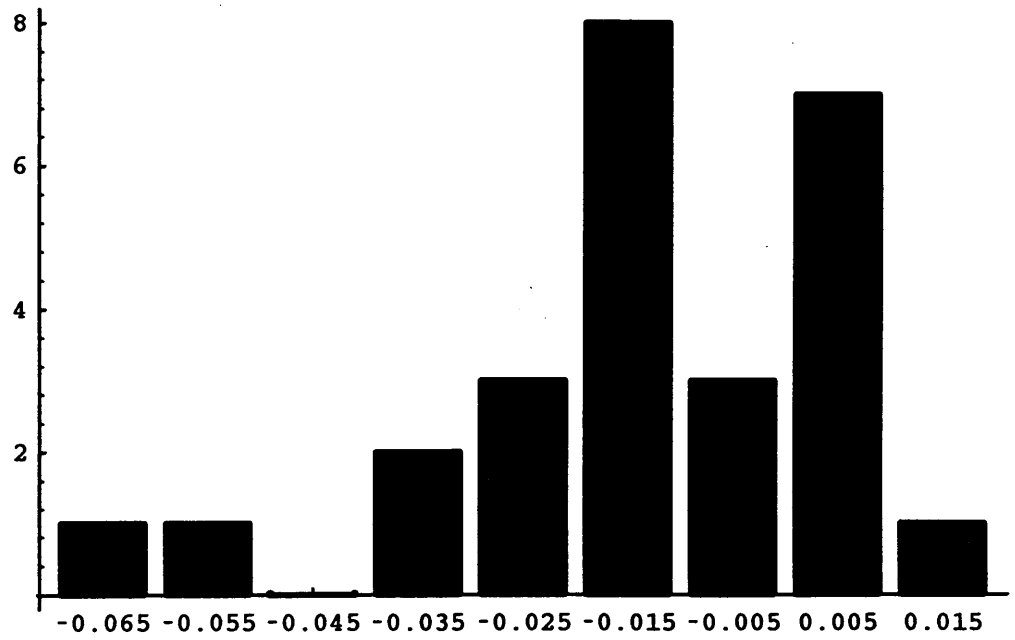

```
-Graphics-
```

Separation between e+ and e-:

```
xsep[marker_]:=xp[marker]-xe[marker]
```

```
histogram[xsep[IP2]/mm ,0.01]
```

```
                                    Mean= -0.0130242
                    Standard Deviation (MLE) =0.017895
```

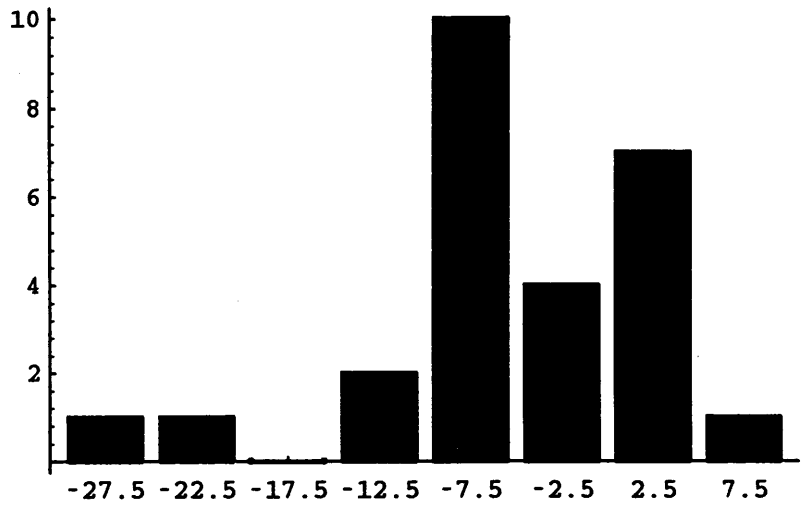



```
-Graphics-
```

Separations at the IP expressed as a PERCENTAGE of the horizontal betatron beam size.

```
histogram[ 100 10^-3 xsep[IP2]/mm /( sigxp[IP2]/m),5]
```

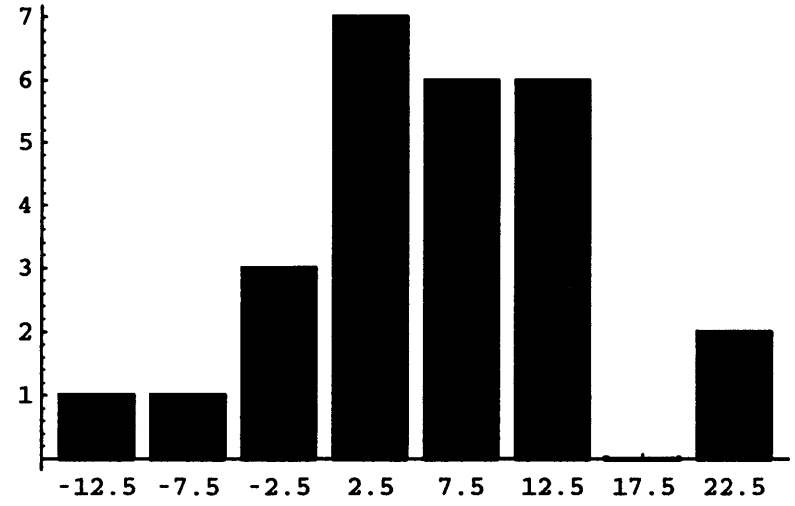
```
                                    Mean= -5.36229
                    Standard Deviation (MLE) =7.39713
```




```
-Graphics-
```

```
histogram[ 100 10^-3 xsep[IP4]/mm /( sigxp[IP4]/m),5]
```

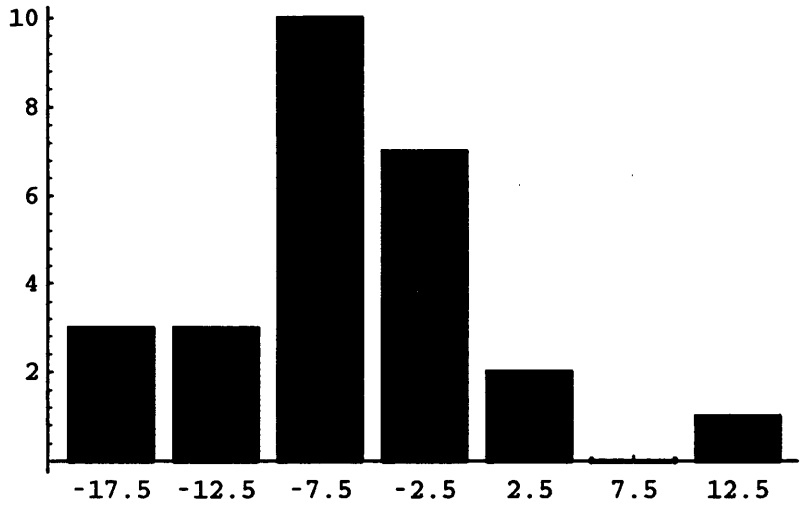
Mean= 6.23864
Standard Deviation (MLE) =7.45929



-Graphics-

```
histogram[ 100 10^-3 xsep[IP6]/mm /( sigxp[IP6]/m),5]
```

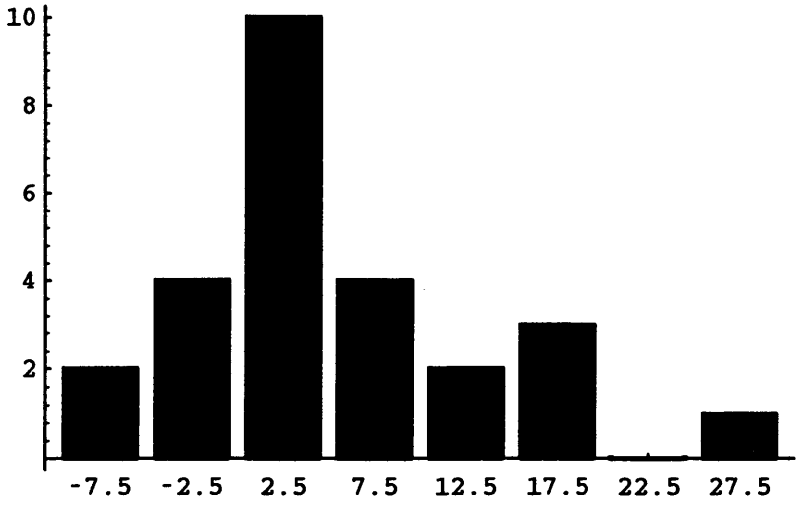
Mean= -6.33431
Standard Deviation (MLE) =6.13445



-Graphics-

```
histogram[ 100 10^-3 xsep[IP8]/mm /( sigxp[IP8]/m),5]
```

```
                                    Mean= 5.42332
                    Standard Deviation (MLE) =7.8975
```




```
-Graphics-
```

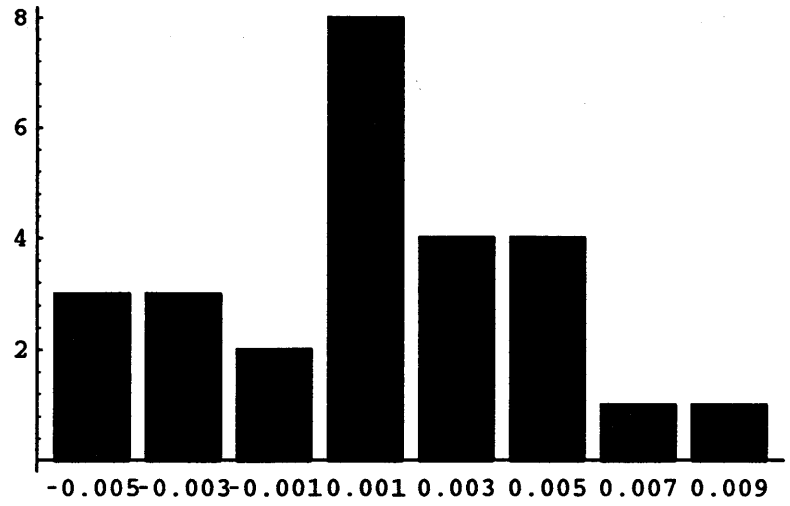
Dispersion and beta functions at the IPs

There is a problem with accuracy of the numbers quoted by MAD's TWISS command here. But we can still get some gross statistical information.

Distribution of horizontal dispersion in mm.

```
histogram[Dxp[IP2]/m,0.002]
```

```
                                    Mean= 0.00169231
                    Standard Deviation (MLE) =0.00366658
```

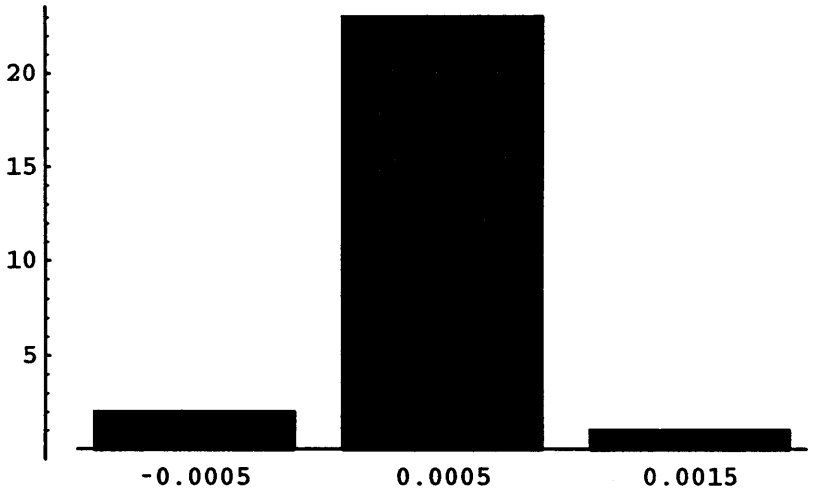



```
-Graphics-
```

Distribution of vertical dispersion in mm.

```
histogram[Dyp[IP2]/m ,0.001]
```

```
                                    Mean= 0.000423077
                    Standard Deviation (MLE) =0.000689095
```

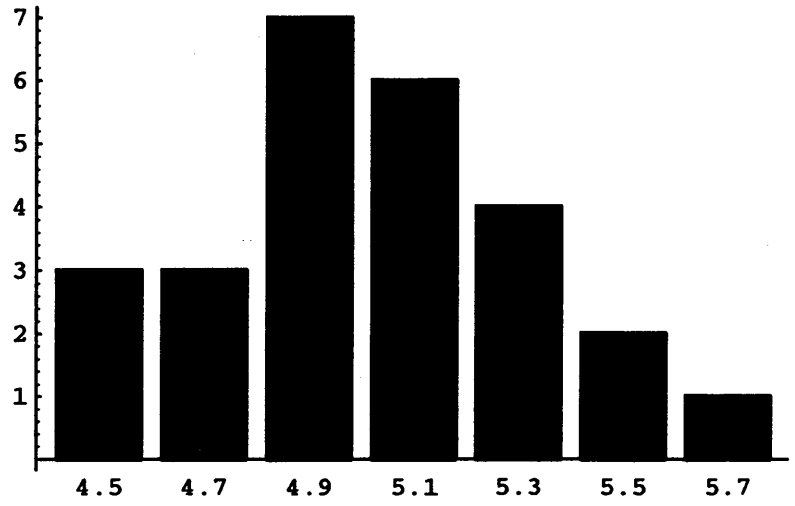



```
-Graphics-
```

Distribution ofvertical beta-function in cm.

```
histogram[100  bety2p[IP2]/m,.2]
```

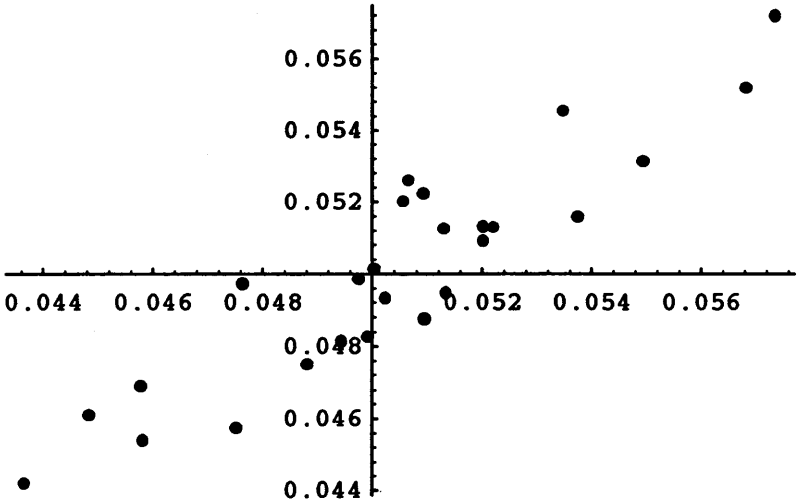
```
Mean= 5.00881
Standard Deviation (MLE) =0.307053
```




```
-Graphics-
```

Correlation of beta-functions for the two beams

```
corrPlot[bety2p[IP2]/m,bety2e[IP2]/m]
```




```
-Graphics-
```

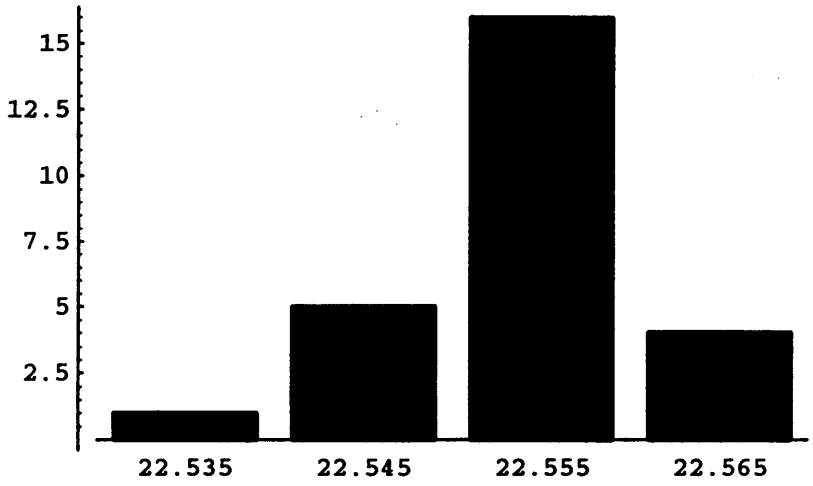
Phase advances between IPs

Remember all mu s are in units of 2Pi.

Look at differences between IPs.

```
histogram[ mu1p[IP4]-mu1p[IP2],0.01]
```

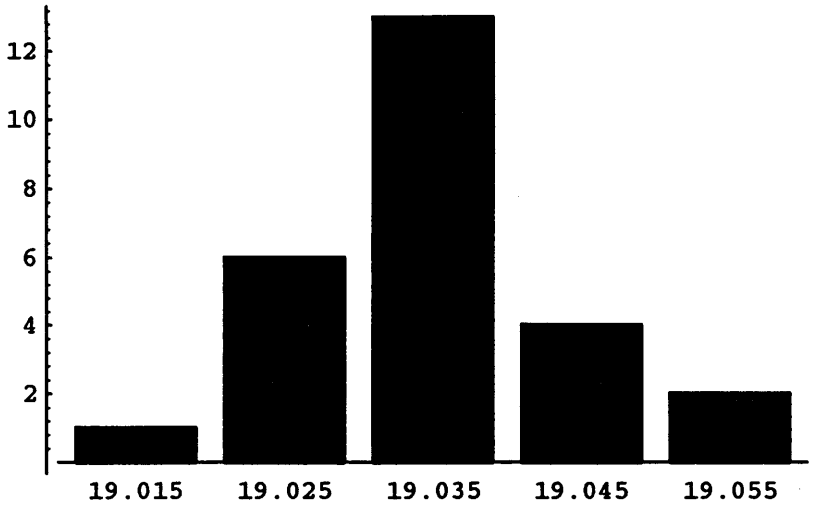
```
                                    Mean= 22.5528
                    Standard Deviation (MLE) =0.00637168
```




```
-Graphics-
```

```
histogram[ mu2p[IP4]-mu2p[IP2] ,0.01]
```

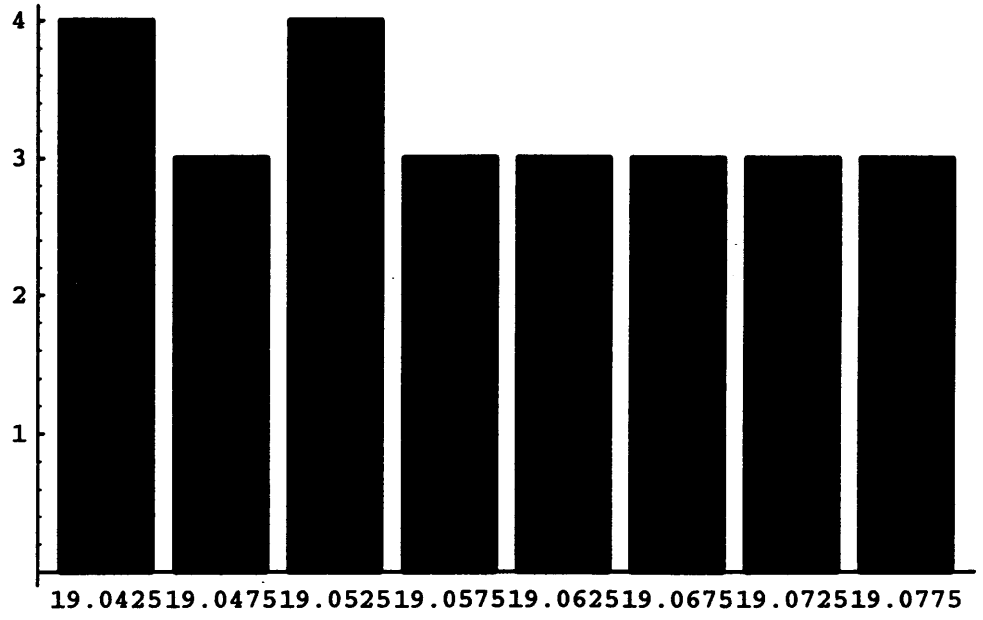
```
                                    Mean= 19.0344
                    Standard Deviation (MLE) =0.00851681
```




```
-Graphics-
```

```
histogram[ mu2p[IP6]-mu2p[IP4],.005]
```

```
                                    Mean= 19.059
                    Standard Deviation (MLE) =0.0114557
```

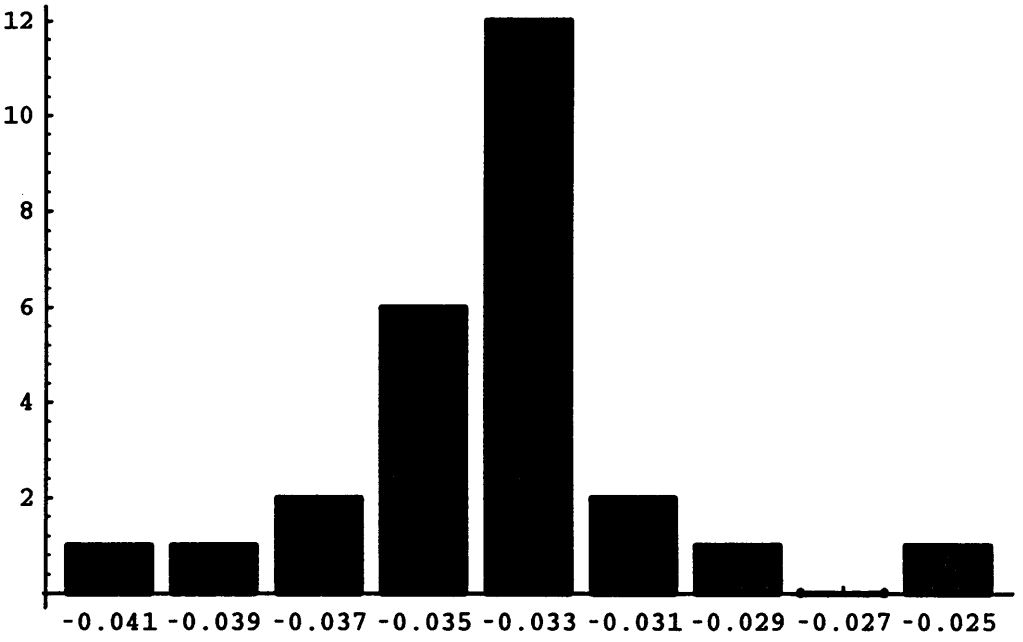



```
-Graphics-
```

Now look at phase differences between particles, remembering the sequence is reversed for electrons.

```
histogram[ ((mulp[IP4]-mulp[IP2])-(mule[IP2]-mule[IP4])
           ),0.002]
```

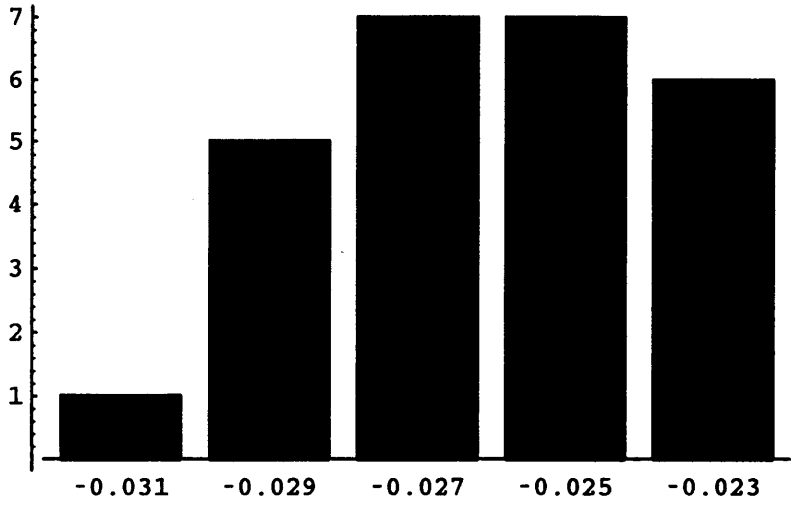
```
                                    Mean= -0.0334781
                    Standard Deviation (MLE) =0.0028076
```




```
-Graphics-
```

```
histogram[ ((mu2p[IP4]-mu2p[IP2])-(mu2e[IP2]-mu2e[IP4])
                          ) ,0.002]
```

```
                                    Mean= -0.0261962
                    Standard Deviation (MLE) =0.00218855
```




```
-Graphics-
```