
CERN

Benchmarking PyHEADTAIL against HEADTAIL

Kevin Li

PDM – 20. 03. 2015

Context

- PSB simulations gave inconclusive results
- PS simulations of the injection oscillations with indirect space charge wakes (low β) need to be redone with direct space charge added
- Can we use PyHEADTAIL?

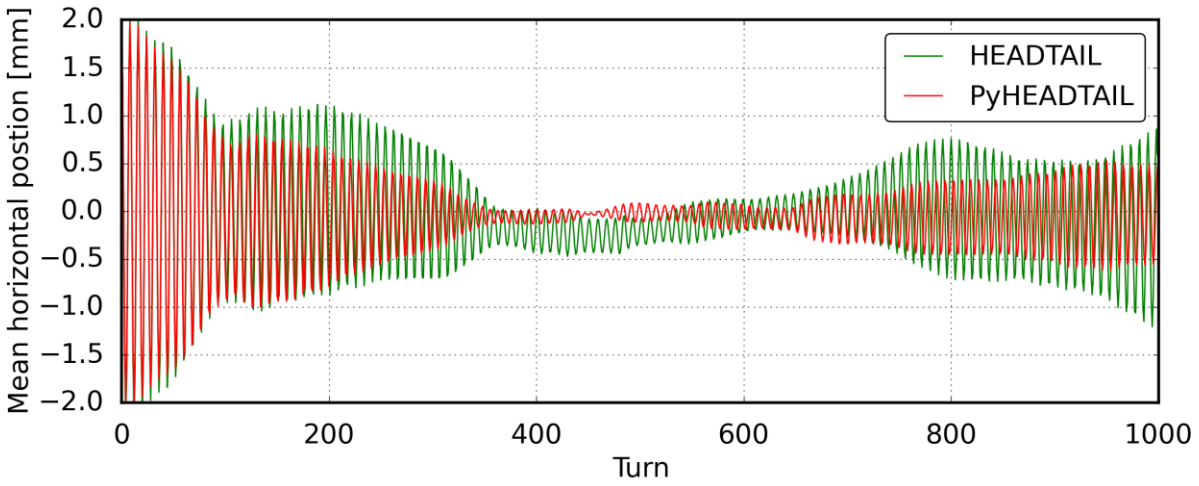
Context

- PSB simulations gave inconclusive results
- PS simulations of the injection oscillations with indirect space charge wakes (low β) need to be redone with direct space charge added
- Can we use PyHEADTAIL?

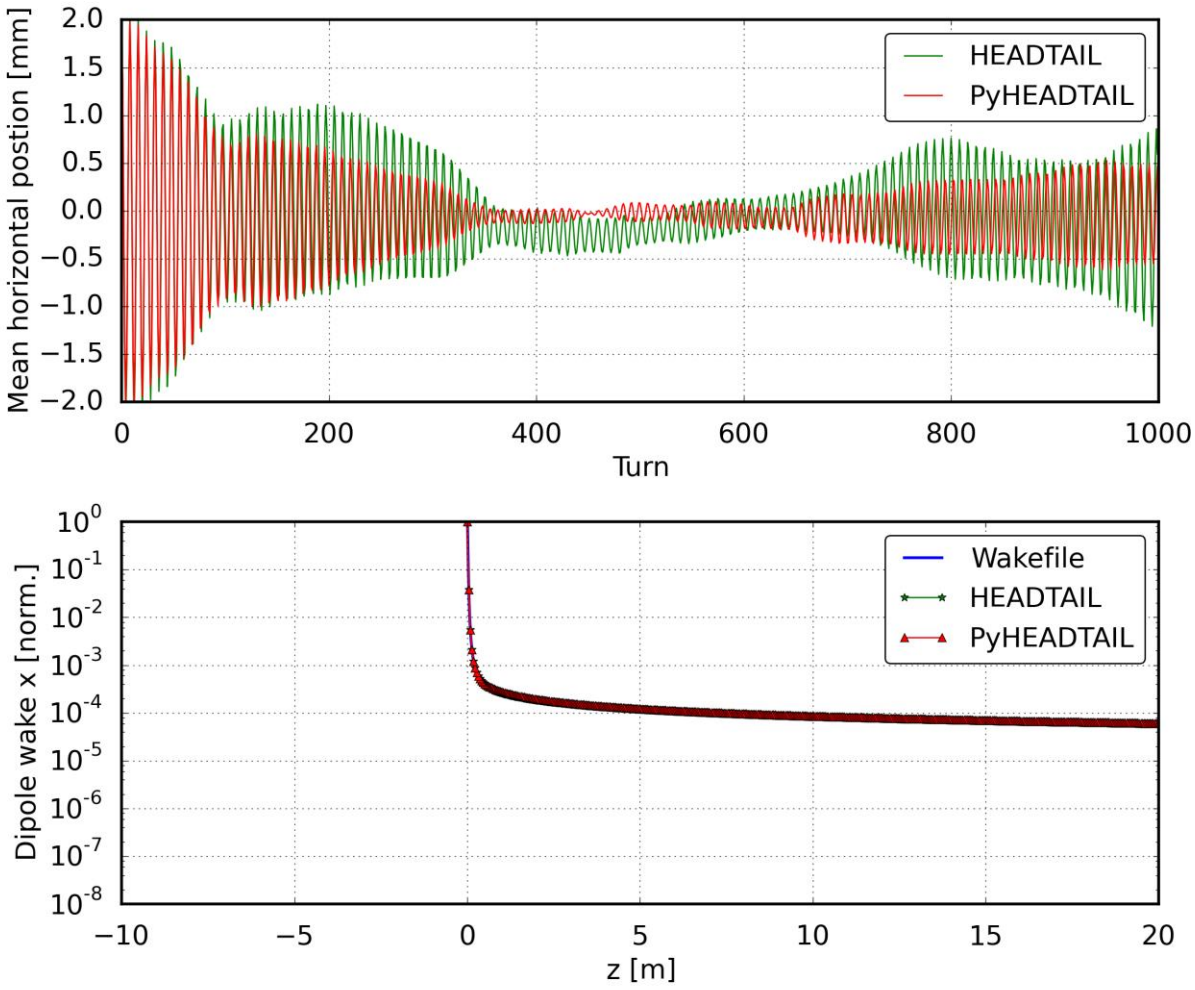
→ Only if PyHEADTAIL passes the benchmark tests against HEADTAIL!

PS TOF beam with indirect sc wakes

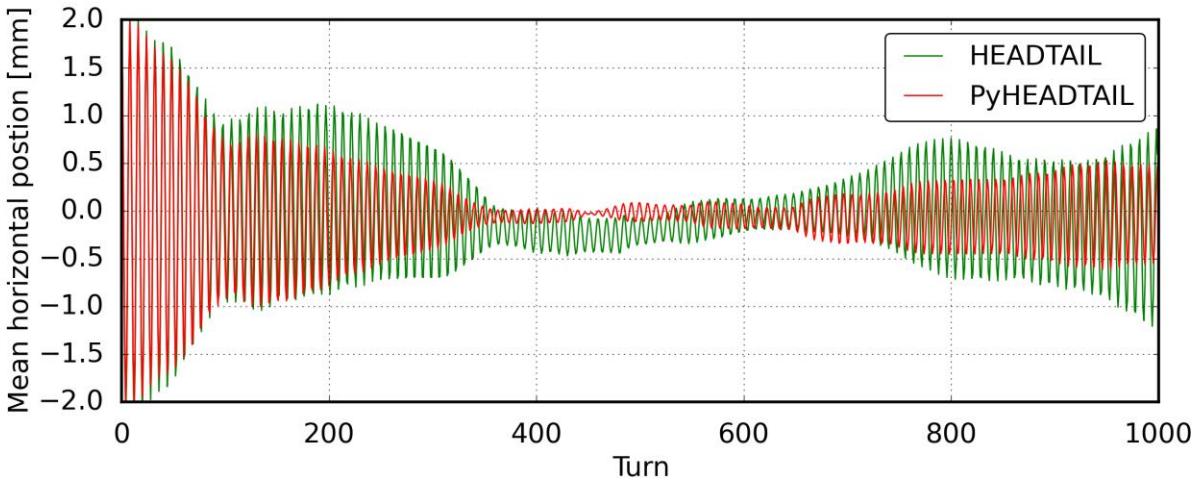
Bad agreement of coh. bunch motion

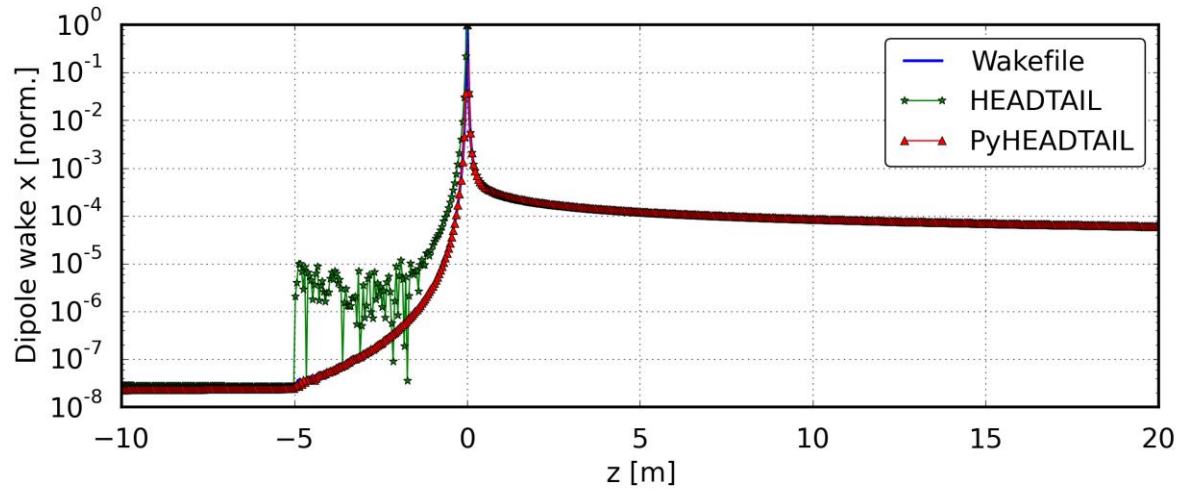


Wakes behind agree well



Wake in front is badly interpolated





Original

```
201
202 /*****************************************************************************
203  *** Subroutine     : table wake_function                                    ***
204  *** Effect         : Function that gives the wake function at a given       ***
205  ***                   location z < 0 (z is 'pos'). Wake coming from table   ***
206  *** Parameters     : pos (position behind the source - negative),           ***
207  ***                   lprov (position in zwt table of fabs(pos))           ***
208  ***                   extwake (wake table)                                 ***
209  *** Gbl var   used : zwt                                                   ***
210  *** Gbl var effect : none                                                  ***
211  *** Constants used : none                                                  ***
212  *** Subrout.  used : none                                                  ***
213  *****************************************************************************/
214
215 double wake_table (double pos, unsigned long lprov, double extwake[])
216 {
217   double wfield ;
218
219   /* previous algorithm to search for the right position in the table - not optimized
220      lprov = 0;
221      do {lprov++;}
222      //printf("%d  %d  %lf  %lf\n",imain,lprov,fabs(pos),zwt[lprov]);}
223      while (fabs(pos)>zwt[lprov]);
224   */
225   wfield = extwake[lprov-1]
226     + (extwake[lprov] - extwake[lprov-1]) / (zwt[lprov] - zwt[lprov-1])
227     * (fabs(pos) - zwt[lprov-1]);
228
229   return wfield;
230
231 }
232
233
234 /*****************************************************************************
235  *** Subroutine     : wake pretreatment                                      ***
236  *** Effect         : Function that gives zout, a table of distances z such  ***
237  ***                   between 2 sucessive z in this table, wake functions   ***
238  ***                   do not vary by more than "ratio" (in relative)        ***
239  *** Parameters     : z (initial distance for the wake sampling),            ***
240  ***                   all the wake tables in a single table (9 columns)    ***
241  ***                   ratio: maximum relative difference allowed between two***
242  ***                   output points                                        ***
243  *** Gbl var   used : none                                                  ***
244  *** Gbl var effect : none                                                  ***
245  *** Constants used : none                                                  ***
246  *** Subrout.  used : none                                                  ***
247  *****************************************************************************/
248
249 std::vector<double> wake_pretreat (double* z, double** W, unsigned long n)
250 {
251   double *mini,*maxi,eps=0.001;
252   int condition;
253   std::vector<double> zout;
```

-:**- Fields.h<2> 69% (202,0) (C/l Abbrev)

```
1022 ng theorem
1023                 // (i.e. with half the value of the wake in 0-)
1024                 kick_z = wakefac * waketZ[0] / 2. * npr1;
1025                 for (n_turn=nt_wake-1; n_turn>=0; n_turn--) {
1026                   ntnb = n_turn*nbunch;
1027                   dist = zs[jmain] - (double)n_turn*circ;
1028                   if (n_turn==0) end_b=bmain;
1029                   else end_b=0;
1030                   for (n_step_b=nbunch-1; n_step_b>=end_b; n_step_b--) {
1031                     imain2 = (n_step_b + ntnb) * nbin;
1032                     zav = zave[offzave + ntnb + n_step_b];
1033                     zst = zstep[offzave + ntnb + n_step_b];
1034                     if (bunch_table[n_step_b]>0) {// do sth only if source bunch is not empty
1035                       if ((n_turn==0)&&(n_step_b==bmain)) end_sl=jmain;
1036                       else end_sl=0;
1037                       //loop over different slices within the bunch
1038                       //1) include all the slices
1039                       for (n_step_sl=nbin-1; n_step_sl>=0; n_step_sl-- ) {
1040                         //2) include only up to slice in front of current slice
1041                         //for(n_step_sl=nbin-1; n_step_sl>=end_sl+1; n_step_sl--) {
1042                         // reconstruct the position of the source slice
1043                         zstmp = zav + zst*( (double)n_step_sl+nbin2 );
1044                         // distance: distance between source and test (negative if test is behind t
he source)
1045                         distance = -zstmp + dist;
1046                         imain = n_step_sl + imain2;
1047                         if (i_pre==0) {
1048                           // not using wake pre-treatment
1049                           if (bunch_table[n_step_b]>0) {
1050                             lprov           = locate(zwt, -distance, 0, lprov);
1051                             wake_storeXdip  = -wake_table(distance, lprov, waketXdip);
1052                             wake_storeYdip  = -wake_table(distance, lprov, waketYdip);
1053                             wake_storeXquad = -wake_table(distance, lprov, waketXquad);
1054                             wake_storeYquad = -wake_table(distance, lprov, waketYquad);
1055                             wake_storeXYdip = -wake_table(distance, lprov, waketXYdip);
1056                             wake_storeXYquad= -wake_table(distance, lprov, waketXYquad);
1057                             wake_storeXcst  = -wake_table(distance, lprov, waketXcst);
1058                             wake_storeYcst  = -wake_table(distance, lprov, waketYcst);
1059                             wakez_store1    =  wake_table(distance, lprov, waketZ);
1060                           }
1061                           else {
1062                             wake_storeXdip  = 0.0;
1063                             wake_storeYdip  = 0.0;
1064                             wake_storeXquad = 0.0;
1065                             wake_storeYquad = 0.0;
1066                             wake_storeXYdip = 0.0;
1067                             wake_storeXYquad= 0.0;
1068                             wake_storeXcst  = 0.0;
1069                             wake_storeYcst  = 0.0;
1070                             wakez_store1    = 0.0;
1071                           }
1072                           tmpfact = wakefac*npr[offxs+imain];
1073                           //printf("turnnumber=%ld, nt_wake+1=%ld, bmain=%ld, bmin=%ld, jmain=%ld,
```

-:**- main.cpp<2> 44% (1023,16) (C++/l Abbrev)

Fixed

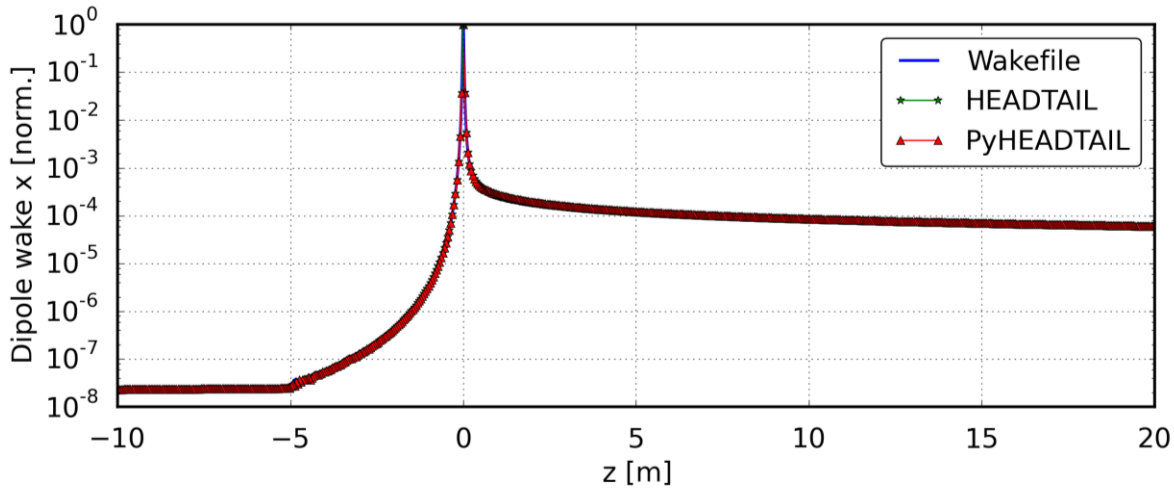
```
201
202 /*****************************************************************************
203  *** Subroutine     : table wake_function                                    ***
204  *** Effect         : Function that gives the wake function at a given       ***
205  ***                   location z < 0 (z is 'pos'). Wake coming from table   ***
206  *** Parameters     : pos (position behind the source - negative),           ***
207  ***                   lprov (position in zwt table of fabs(pos))           ***
208  ***                   extwake (wake table)                                 ***
209  *** Gbl var   used : zwt                                                   ***
210  *** Gbl var effect : none                                                  ***
211  *** Constants used : none                                                  ***
212  *** Subrout.  used : none                                                  ***
213  *****************************************************************************/
214
215 double wake_table (double pos, unsigned long lprov, double extwake[])
216 {
217   double wfield ;
218
219   /* previous algorithm to search for the right position in the table - not optimized
220      lprov = 0;
221      do {lprov++;}
222      //printf("%d  %d  %lf  %lf\n",imain,lprov,fabs(pos),zwt[lprov]);}
223      while (fabs(pos)>zwt[lprov]);
224   */
225   wfield = extwake[lprov-1]
226     + (extwake[lprov] - extwake[lprov-1]) / (zwt[lprov] - zwt[lprov-1])
227     * ((pos) - zwt[lprov-1]);
228
229   return wfield;
230
231 }
232
233
234 /*****************************************************************************
235  *** Subroutine     : wake pretreatment                                      ***
236  *** Effect         : Function that gives zout, a table of distances z such  ***
237  ***                   between 2 sucessive z in this table, wake functions   ***
238  ***                   do not vary by more than "ratio" (in relative)        ***
239  *** Parameters     : z (initial distance for the wake sampling),            ***
240  ***                   all the wake tables in a single table (9 columns)     ***
241  ***                   ratio: maximum relative difference allowed between two***
242  ***                   output points                                         ***
243  *** Gbl var   used : none                                                  ***
244  *** Gbl var effect : none                                                  ***
245  *** Constants used : none                                                  ***
246  *** Subrout.  used : none                                                  ***
247  *****************************************************************************/
248
249 std::vector<double> wake_pretreat (double* z, double** W, unsigned long n)
250 {
251   double *mini,*maxi,eps=0.001;
252   int condition;
253   std::vector<double> zout;
```

-:**- Fields.h<2> 69% (202,0) (C/l Abbrev)

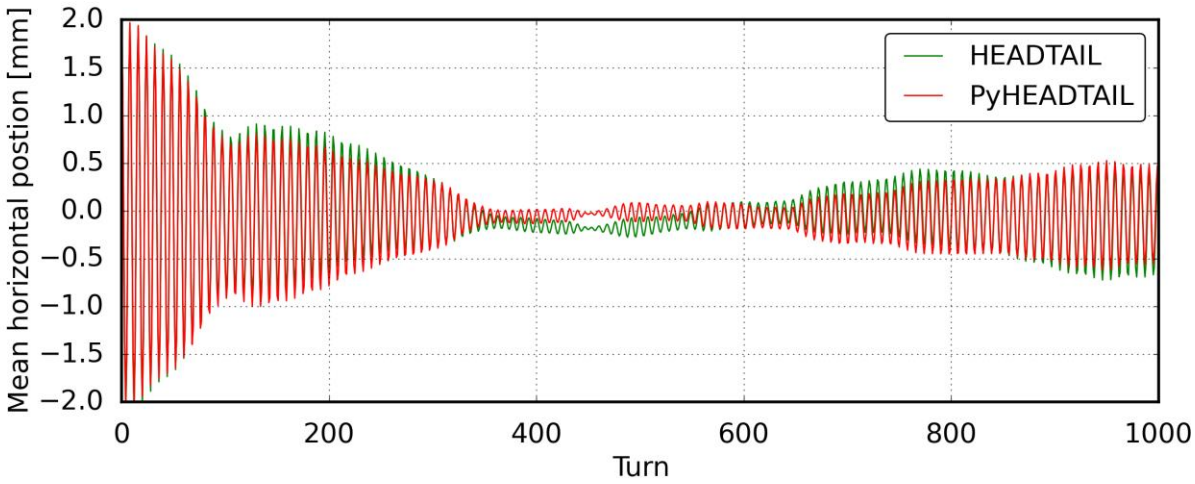
```
1022 ng theorem
1023                 // (i.e. with half the value of the wake in 0-)
1024                 kick_z = wakefac * waketZ[0] / 2. * npr1;
1025                 for (n_turn=nt_wake-1; n_turn>=0; n_turn--) {
1026                   ntnb = n_turn*nbunch;
1027                   dist = zs[jmain] - (double)n_turn*circ;
1028                   if (n_turn==0) end_b=bmain;
1029                   else end_b=0;
1030                   for (n_step_b=nbunch-1; n_step_b>=end_b; n_step_b--) {
1031                     imain2 = (n_step_b + ntnb) * nbin;
1032                     zav = zave[offzave + ntnb + n_step_b];
1033                     zst = zstep[offzave + ntnb + n_step_b];
1034                     if (bunch_table[n_step_b]>0) {// do sth only if source bunch is not empty
1035                       if ((n_turn==0)&&(n_step_b==bmain)) end_sl=jmain;
1036                       else end_sl=0;
1037                       //loop over different slices within the bunch
1038                       //1) include all the slices
1039                       for (n_step_sl=nbin-1; n_step_sl>=0; n_step_sl-- ) {
1040                         //2) include only up to slice in front of current slice
1041                         //for(n_step_sl=nbin-1; n_step_sl>=end_sl+1; n_step_sl--) {
1042                         // reconstruct the position of the source slice
1043                         zstmp = zav + zst*( (double)n_step_sl+nbin2 );
1044                         // distance: distance between source and test (negative if test is behind t
he source)
1045                         distance = -zstmp + dist;
1046                         imain = n_step_sl + imain2;
1047                         if (i_pre==0) {
1048                           // not using wake pre-treatment
1049                           if (bunch_table[n_step_b]>0) {
1050                             lprov            = locate(zwt, -distance, 0, lprov);
1051                             wake_storeXdip   = -wake_table(-distance, lprov, waketXdip);
1052                             wake_storeYdip   = -wake_table(-distance, lprov, waketYdip);
1053                             wake_storeXquad  = -wake_table(-distance, lprov, waketXquad);
1054                             wake_storeYquad  = -wake_table(-distance, lprov, waketYquad);
1055                             wake_storeXYdip  = -wake_table(-distance, lprov, waketXYdip);
1056                             wake_storeXYquad = -wake_table(-distance, lprov, waketXYquad);
1057                             wake_storeXcst   = -wake_table(-distance, lprov, waketXcst);
1058                             wake_storeYcst   = -wake_table(-distance, lprov, waketYcst);
1059                             wakez_store1     =  wake_table(-distance, lprov, waketZ);
1060                           }
1061                           else {
1062                             wake_storeXdip   = 0.0;
1063                             wake_storeYdip   = 0.0;
1064                             wake_storeXquad  = 0.0;
1065                             wake_storeYquad  = 0.0;
1066                             wake_storeXYdip  = 0.0;
1067                             wake_storeXYquad = 0.0;
1068                             wake_storeXcst   = 0.0;
1069                             wake_storeYcst   = 0.0;
1070                             wakez_store1     = 0.0;
1071                           }
1072                           tmpfact = wakefac*npr[offxs+imain];
1073                           //printf("turnnumber=%ld, nt_wake+1=%ld, bmain=%ld, bmin=%ld, jmain=%ld,
```

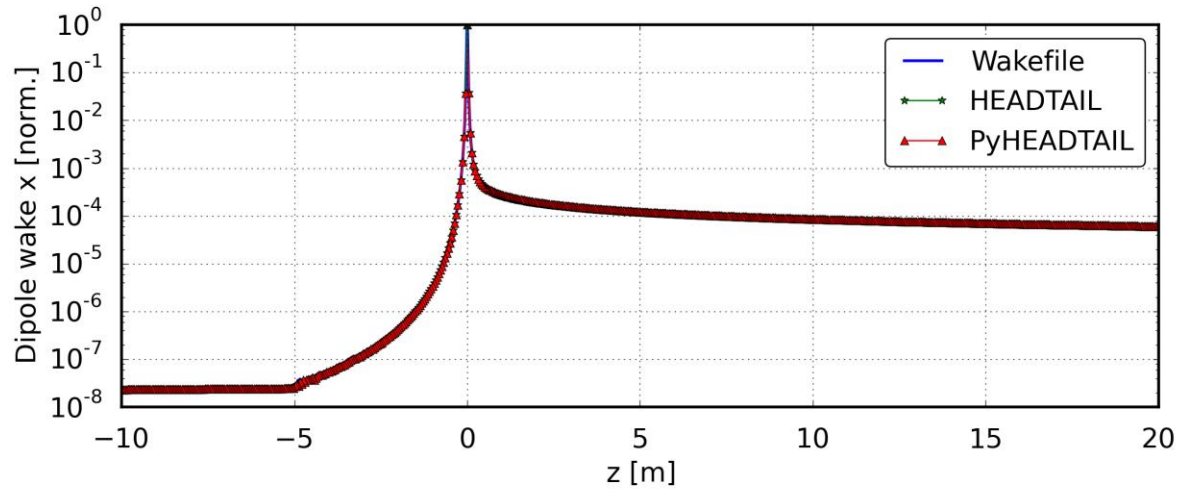
-:**- main.cpp<2> 44% (1023,16) (C++/l Abbrev)

After fix – wake fields agree



After fix – problem with dispersion left





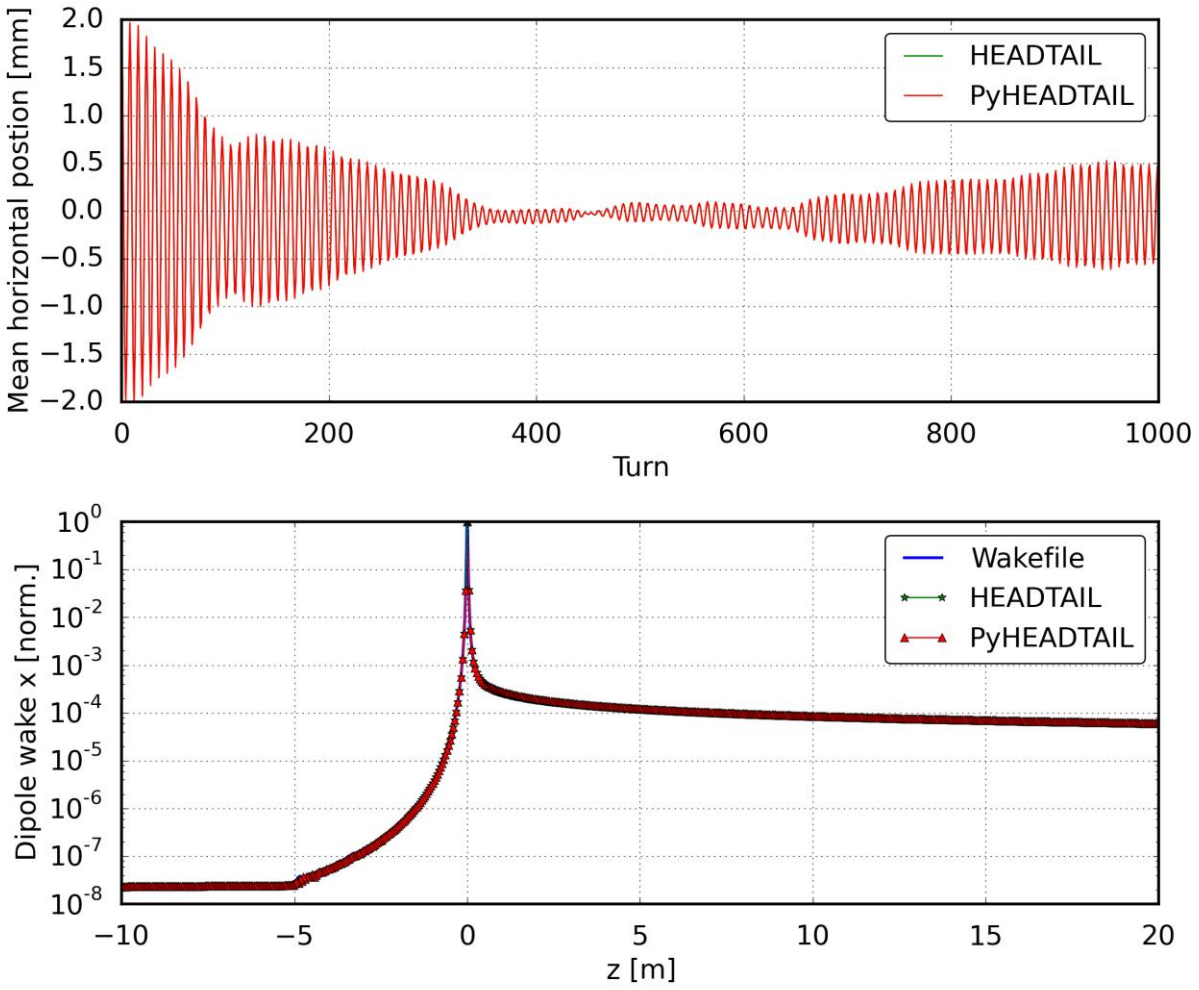
Return to HEADTAIL – fix re-binning

- Dipolar wake kicks based on slice first moments
- Slice first moments calculated without dispersion
- We need to re-bin after adding dispersion → slice first moment with dispersion

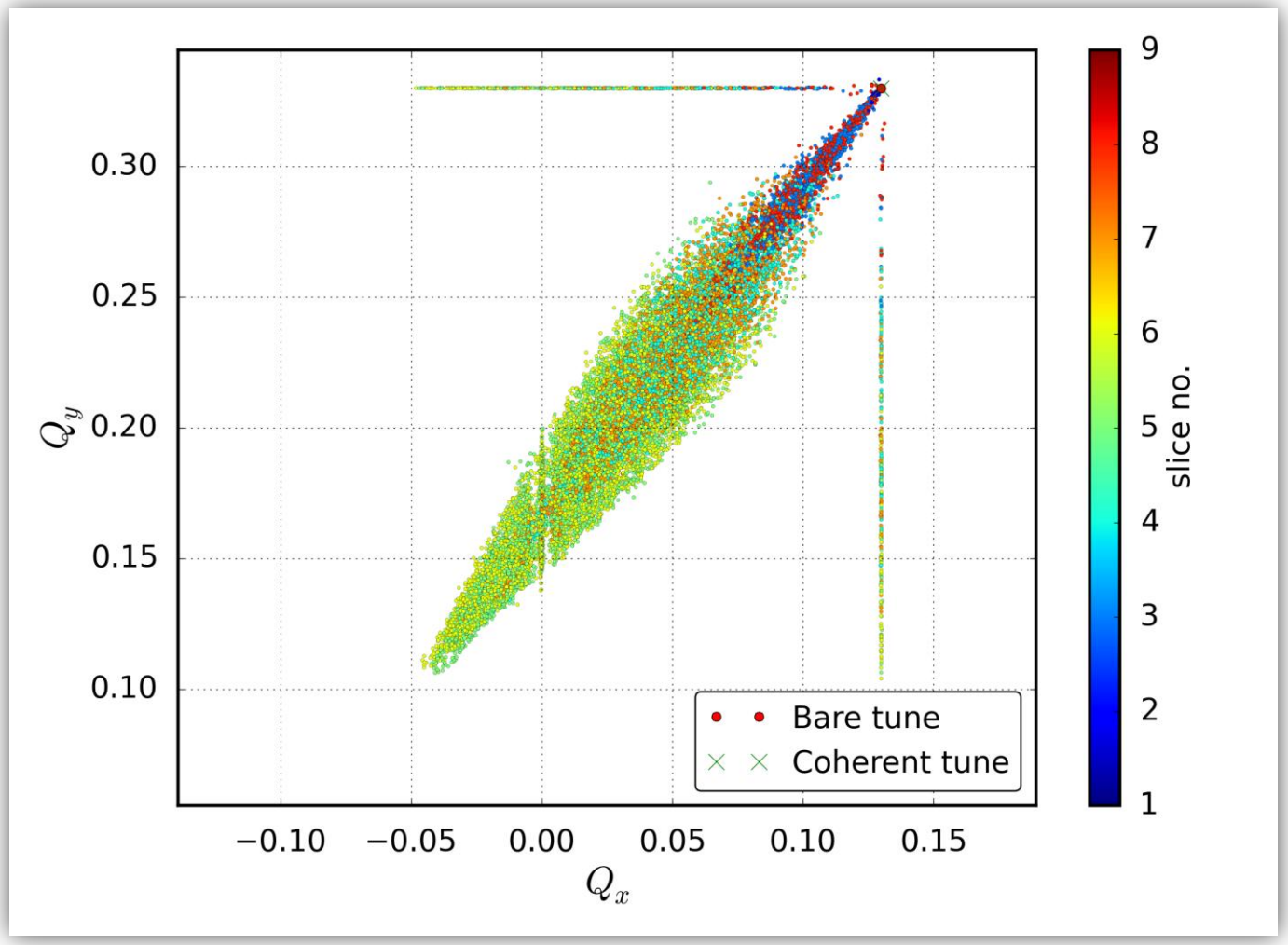
  Thanks Giovanni!

```
741          //}
742
743          /* dispersion is added before interaction bunch-impedance */
744          if(dispx>0.1) {
745            for(jmain=fbunch[bmain_me]+npr_eff[bmain_me]; jmain<fbunch[bmain_me]+nprbunch[bmain_me]; jmain++)
746              xpr[jmain] += dispx * dp[jmain];
747
748            // Re-binning
749            binning(zavetmp, szsize, bmain_me);
750
751            // NM Feb 2011 (MPI): if the number of proc. (nprocs) is not a divider of the number of bunches
752            // (nb_nonempty), then xs,ys, etc. of the last bunches treated should be also broadcasted
753            // to the processors outside commplus
754            for(iproc=0; iproc<1; iproc++) {
755              bmain2=bmain;
756              zavetmp2=zavetmp;
757              zsteptmp2=zsteptmp;
758              for(i=0; i<nbin; i++) {
759                xstmp2[i]=xstmp[i];
760                ystmp2[i]=ystmp[i];
761                nprtmp2[i]=nprtmp[i];
762              }
763 #ifdef USEMPI
764              MPI_Bcast(&bmain2, 1, MPI_LONG, iproc, MPI_COMM_WORLD);
765              MPI_Bcast(&zavetmp2, 1, MPI_DOUBLE, iproc, MPI_COMM_WORLD);
766              MPI_Bcast(&zsteptmp2, 1, MPI_DOUBLE, iproc, MPI_COMM_WORLD);
767              MPI_Bcast(xstmp2, nbin, MPI_DOUBLE, iproc, MPI_COMM_WORLD);
768              MPI_Bcast(ystmp2, nbin, MPI_DOUBLE, iproc, MPI_COMM_WORLD);
769              MPI_Bcast(nprtmp2, nbin, MPI_DOUBLE, iproc, MPI_COMM_WORLD);
770 #endif
771              zave[offzave+bmain2]=zavetmp2;
772              zstep[offzave+bmain2]=zsteptmp2;
773              for(i=0; i<nbin; i++) {
774                xs[offxs+nbin*bmain2+i]=xstmp2[i];
775                ys[offxs+nbin*bmain2+i]=ystmp2[i];
776                npr[offxs+nbin*bmain2+i]=nprtmp2[i];
777              }
778            }
779          }
780
781
782          dpturn = 0.;
783
784          if (i_space == 1) {
785            vprx=new double[NPR1];
786            vpry=new double[NPR1];
787            vprxp=new double[NPR1];
788            vpryp=new double[NPR1];
789          }
790
791
792          /* offsets and rms sizes of the slices are printed to file.
793             NM Feb 2011, for the MPI version: created separate loops on the slices. */
-:---  main.cpp       29% (793,0)     (C++/l Abbrev)
```

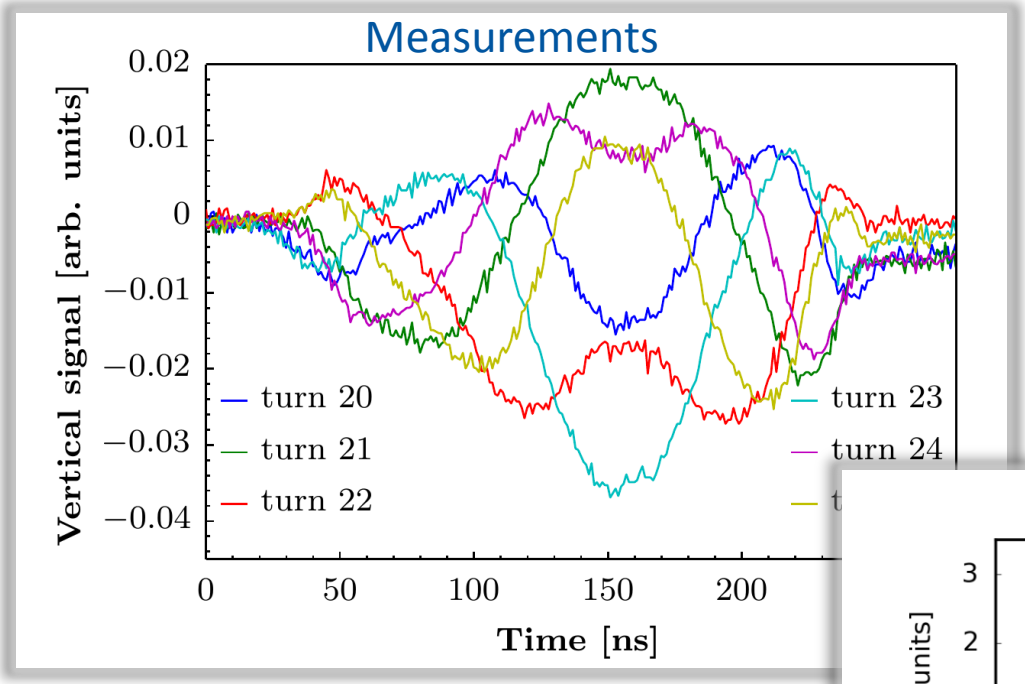
Finally ist seems like PyHEADTAIL works



… also with direct space charge

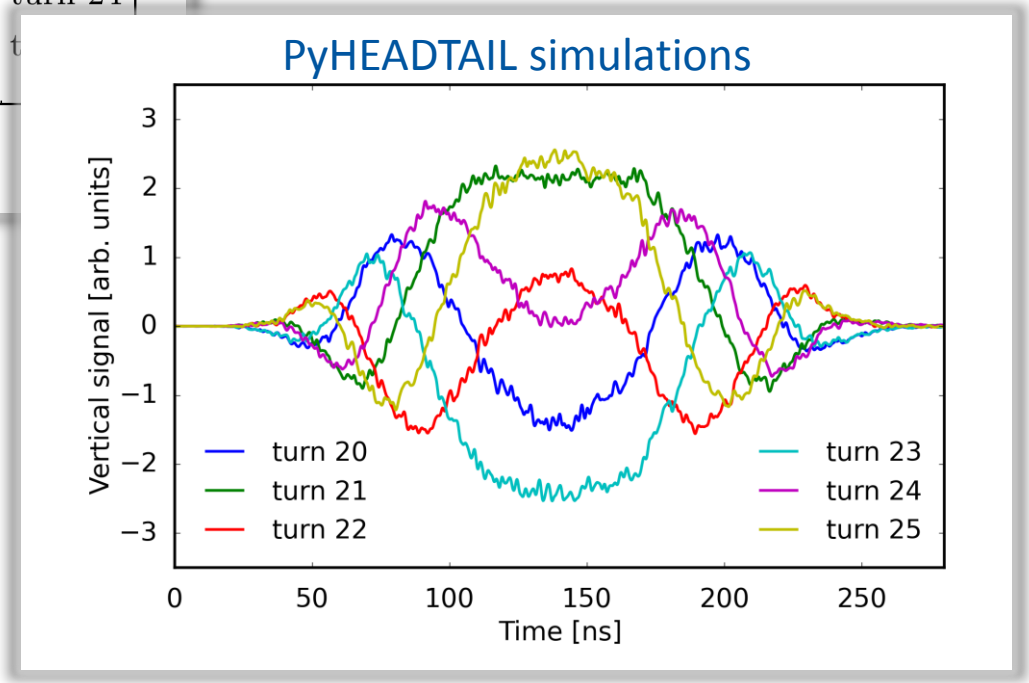


PyHEADTAIL – benchmarks (PS)



Unfortunately, with all these „corrections“ the resemblence with measurements is no longer that striking!

A. Huschauer et al.



Conclusions

- PyHEADTAIL successfully benchmarked against HEADTAIL for PS low β wakes.
- Wakes in front treated correctly
- Dispersion treated correctly
- PS simulations with direct space charge launched
- PSB simulations still pending… include multi-turn wakes

www.cern.ch