

Muon Emittance Exchange with a Potato Slicer

D. J. Summers, T. L. Hart, J. G. Acosta, L. M. Cremaldi,
S. J. Oliveros, and L. P. Perera

University of Mississippi - Oxford, University, MS 38677 USA

D. V. Neuffer

Fermilab, Batavia, IL 60510 USA

MAP-DOC-4405, Apr 2015

Abstract

We propose a novel scheme for final muon ionization cooling with quadrupole doublets followed by emittance exchange in vacuum to achieve the small beam sizes needed by a muon collider. A flat muon beam with a series of quadrupole doublet half cells appears to provide the strong focusing required for final cooling. Each quadrupole doublet has a low beta region occupied by a dense, low Z absorber. After final cooling, normalized transverse, longitudinal, and angular momentum emittances of 0.100, 2.5, and 0.200 mm-rad are exchanged into 0.025, 70, and 0.0 mm-rad. A skew quadrupole triplet transforms a round muon bunch with modest angular momentum into a flat bunch with no angular momentum. Thin electrostatic septa efficiently slice the flat bunch into 17 parts. The 17 bunches are interleaved into a 3.7 meter long train with RF deflector cavities. Snap bunch coalescence combines the muon bunch train longitudinally in a 21 GeV ring in $55 \mu\text{s}$, one quarter of a synchrotron oscillation period. A linear long wavelength RF bucket gives each bunch a different energy causing the bunches to drift in the ring until they merge into one bunch and can be captured in a short wavelength RF bucket with a 13% muon decay loss and a packing fraction as high as 87%.

Introduction

Due to s-channel production, a muon collider [1] may be ideal for the examination of H/A Higgs scalars which could be at the $1.5 \text{ TeV}/c^2$ mass scale and are required in supersymmetric

Table 1: Helical and Rectilinear Cooling Channel normalized 6D emittances from simulations and the normalized 6D emittance needed for muon collider. The channels cool by over five orders of magnitude and need less than a factor of 10 more for a muon collider. The 21 bunches present after phase rotation are also merged into one bunch during the 6D cooling.

	ϵ_x (mm)	ϵ_y (mm)	ϵ_z (mm)	ϵ_{6D} (mm ³)	Ref.
Emittance after Phase Rotation	48.6	48.6	17.0	40,200	[4]
Helical Cooling Channel	0.523	0.523	1.54	0.421	[3]
Rectilinear Cooling Channel	0.28	0.28	1.57	0.123	[4]
Muon Collider	0.025	0.025	70	0.044	[1]

Table 2: Rectilinear cooling channel final cell beta function [5]. $p = 204 \text{ MeV}/c$, $\beta \gamma = p/mc = 204/105.7 = 1.93$, $\epsilon_{x,y}^N = 0.280 \text{ mm}$, $\sigma_{x,y} = \sqrt{\beta_{x,y} \epsilon_{x,y}^N / (\beta \gamma)}$, and $\theta_{x,y} = \sqrt{\epsilon_{x,y}^N / (\beta_{x,y} \beta \gamma)}$. A bore diameter of $8 \times 7.86 \text{ mm} = 62.9 \text{ mm}$ contains $\pm 4 \sigma_{x,y}$ when $\beta_{x,y} = 42.64 \text{ cm}$. The regions where β is near 3 cm are short.

$z(\text{m})$	0.000	0.016	0.030	0.092	0.183	0.402	0.625	0.717	0.779	0.793	0.806
$\beta_{x,y}(\text{cm})$	3.08	3.93	5.92	26.16	42.64	33.75	42.64	26.16	5.92	3.93	3.10
$\sigma_{x,y}(\text{mm})$	2.11	2.39	2.93	6.16	7.86	7.00	7.86	6.16	2.93	2.39	2.12
$\theta_{x,y}(\text{mrad})$	68.6	60.7	49.5	23.5	18.4	20.7	18.4	23.5	49.5	60.7	68.4

models [2]. But what is the status of muon cooling? As noted in Table 1, more than five orders of magnitude of muon cooling have been shown in two simulated designs [3, 4] but not quite the six orders of magnitude needed for a high luminosity muon collider. Also as noted in Table 1, some of the longitudinal cooling needs to be exchanged for lower transverse emittance.

The breakdown of RF cavities operating in strong magnetic fields is an issue [6]. The Helical Cooling Channel inhibits breakdown with high pressure hydrogen [7]. Hydrogen at moderate pressures, lower than those used in interstate natural gas pipelines, may work for the Rectilinear Cooling Channel [8]. As seen in Table 2, the Rectilinear Cooling Channel does have some high β regions, but these only cause minimal heating if hydrogen pressure is modest [9].

An infinite solenoid [10] with a 14 Tesla magnetic field and a 200 MeV/c muon beam gives a betatron function of $\beta_{\perp} = 2p/(3.0B) = 2(200 \text{ MeV}/c)/[3.0(14 \text{ T})] = 9.5 \text{ cm}$. As noted in Table 2, the short solenoids in the final stage of the Rectilinear Cooling Channel give a

Table 3: Muon equilibrium emittance at 200 MeV/c ($\text{KE} = 121 \text{ MeV}$, $\beta = v/c = 0.88$) for hydrogen gas, lithium hydride, beryllium, boron carbide, diamond, and beryllium oxide [12, 13]. $\epsilon_{\perp} = \beta^* E_s^2 / (2g_x \beta m_{\mu} c^2 (dE/ds) L_R)$, where β^* twiss is 1 cm, E_s is 13.6 MeV, the transverse damping partition number, g_x is one with parallel absorber faces, $m_{\mu} c^2$ is 105.7 MeV, and L_R is radiation length.

Material	Density g/cm ³	L_R cm	dE/ds MeV/cm	ϵ_{\perp} (equilibrium) mm - rad
H ₂ gas	0.000084	750,000	0.00037	0.036
Li H	0.82	97	1.73	0.059
Be	1.85	35.3	3.24	0.087
B ₄ C	2.52	19.9	4.57	0.109
Diamond	3.52	12.1	6.70	0.123
BeO	3.01	13.7	5.51	0.132

betatron function of 3.1 cm which is used with lithium hydride. To possibly get to the lower betatron values of about 1 cm needed by a muon collider for final cooling [11], quadrupole doublet cells are explored in Appendix Z. Table 3 gives transverse equilibrium emittances for a number of low Z materials, particularly those with high densities.

Round to Flat Beam Transformation

Assume that a muon beam with a normalized transverse emittance of 0.100 mm is available. Further assume that the beam has some modest and smooth residual angular momentum coming out of a 6D cooling channel. The beam does pick up canonical angular momentum as it passes through absorbers in solenoids.

First, a round spinning muon beam with angular momentum is transformed to a flat non-spinning beam with a skew quadrupole triplet [14] as shown in Fig. 1. This should make slicing easier. The x to y emittance ratio of the flat beam is $(\sqrt{\epsilon^2 + L^2} + L)/(\sqrt{\epsilon^2 + L^2} - L)$, where ϵ is the intrinsic normalized transverse emittance, $L = eB\sigma_{x,y}^2/2mc$ is the quadrature emittance contribution of the canonical angular momentum, B is the solenoidal field strength that the beam is exiting, and $\sigma_{x,y}$ is the round beam radius. L is chosen to give an emittance ratio of 16 for a beam with new nominal transverse normalized emittances of 0.0264 mm and 0.4206 mm. The muon momentum is 200 ± 10 MeV/ c . Higher momentum spreads dilute performance, but beam preconditioning may help [15].

Slice the Flat Beam with 16 Septa

Slice [16] the flat muon beam into 17 pieces. The slice width is chosen to give a horizontal emittance of 0.025 mm-rad and includes 91% of the muons; 9% of the muons are in the tails and lost. A $w = 0.1$ mm wide electrostatic septa was used with 98% efficiency at the Fermilab Tevatron fixed target program for multiturn extraction. At the Tevatron the fractional loss was given by

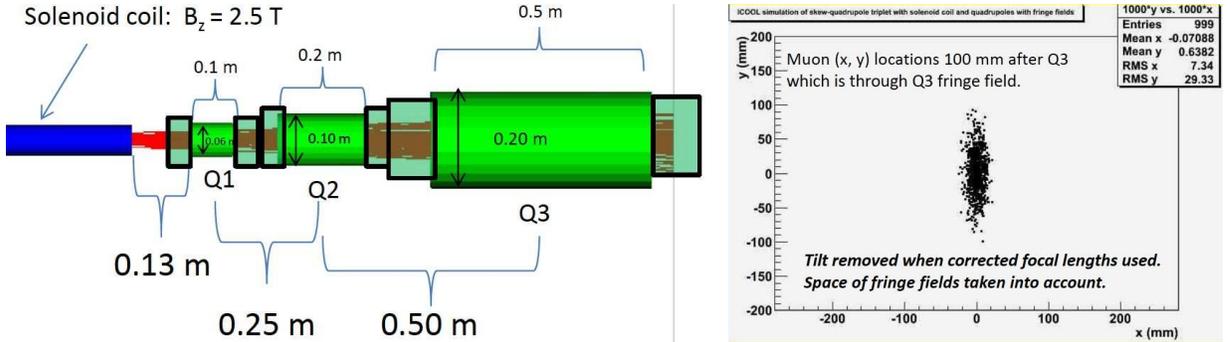


Figure 1: A round, spinning 200 ± 10 MeV/ c muon beam with a normalized transverse emittance of 0.100 mm is transformed to a flat, non-spinning muon beam with new normalized transverse emittances of 0.0264 mm and 0.4206 mm [14]. Inside the 2.5 T solenoid $\sigma_{x,y} = 7.5$ mm. The skew quadrupole pole tip fields and focal lengths are (0.77, 0.45, 0.12) T and (0.259, -0.369, 1.073) m, respectively. The calculated thin quadrupole parameters must be adjusted somewhat to make finite length quadrupoles work properly. Fringe fields are included (see Appendix A). The simulation was done with ICOOL [17].

$$\frac{4\sqrt{2}w}{x_{\max}\sqrt{\beta_s/\beta_0}} = \frac{4\sqrt{2}\times 0.1\text{ mm}}{20\text{ mm}\sqrt{2.3}} = 0.02, \quad (1)$$

where x_{\max} is the beam size with the usual β_0 of the lattice and a larger β_{\perp} function, β_s , is used in the extraction region to make the beam bigger. The physical width of the muon beam is

$$\sqrt{\frac{\epsilon_{N,x}\beta_x}{\beta\gamma}} = \sqrt{\frac{(0.4206\text{ mm})(1,000\text{ mm})}{(0.93)(2.72)}} = 13\text{ mm}, \quad (2)$$

giving a modest 3% slicing loss. The geometrical width of the beam needs to be much larger than the width of the electrostatic septa for efficient slicing.

Create a 3.7 m long bunch train with RF deflector cavities

Combine 17 bunches into a 3.7 m long train with RF deflector cavities as used in CLIC tests. Each cavity interleaves two or three bunch trains. Deflection is 4.5 mrad or zero at 300 MeV/c [18]. The final train has a 231 mm bunch spacing for acceleration by 1300 MHz RF cavities (see Table 4). Estimate the required kick [19] to inject a 300 MeV/c ($\gamma\beta = 300/105.7 = 2.84$) beam with a normalized emittance of 0.025 mm-rad and a β^* of 8000 mm. The kick must be 4x greater than the rms divergence of the beam or $4\sqrt{\epsilon/(\gamma\beta\beta^*)} = 4.2$ mrad, which matches CLIC. The $\pm 4\sigma$ beam diameter is $8\sqrt{\epsilon\beta^*/(\gamma\beta)} = 67$ mm.

Snap bunch coalesce a train of 17 bunch into one in a ring

Finally, *snap bunch coalescing* with RF is used to combine the 17 muon bunches longitudinally. In snap bunch coalescing, all bunches are partially rotated over a quarter of a synchrotron period in energy-time space with a linear long wavelength RF bucket and then the bunches drift in a ring until they merge into one bunch and can be captured in a short wavelength RF bucket. The bunches drift together because they each have a different energy set to cause the drift. Snap bunch coalescing replaced adiabatic bunch coalescing

Table 4: Combine 17 bunches into a 3.7 m long train with 10 RF Deflector Cavities. Each cavity interleaves two or three bunch trains. Deflection is ± 4.5 mrad or zero at 300 MeV/c. The final train has a 231 mm bunch spacing for acceleration by 1300 MHz RF cavities.

Tier	Number of Trains Interleaving	Number of RF Deflector Cavities	RF Frequency MHz	RF Wavelength	Output Spacing in Wavelengths	Output Bunch Spacing
1	17 \rightarrow 6	6	(9/16)1300 = 731	410 mm	9/4	923 mm
2	6 \rightarrow 2	3	(3/8)1300 = 487	616 mm	3/4	462 mm
3	2 \rightarrow 1	1	(1/2)1300 = 650	462 mm	1/2	231 mm

at the Fermilab Tevatron collider program and was used for many years [20]. Sets of fifteen bunches were combined in the Tevatron. A 21 GeV ring has been used in a simulation [21] with ESME [22] to show the coalescing of 17 muon bunches in 55 μ s. The lattice has $\gamma_t = 5.6$ [23]. The muon decay loss is 13%. The longitudinal packing fraction is as high as 87% [24]. So nominally, the initial normalized 2.5 mm longitudinal emittance is increased by a factor of 17/0.87 to become 49 mm, which is less than the 70 mm needed for a muon collider.

Conclusions

Emittance exchange with a potato slicer may be able to achieve the final normalized 0.025 mm transverse and 70 mm longitudinal emittances needed for a high luminosity muon collider,

$$L = \frac{\gamma N^2 f_0 (DC)}{4\pi \epsilon_{x,y} \beta^*} = \frac{7000 (2 \times 10^{12})^2 180,000/s (0.062)}{4\pi (0.0025 \text{ cm}) 1.0 \text{ cm}} = 1.0 \times 10^{34} \text{ cm}^{-2} \text{ s}^{-1} \quad (3)$$

where L is average luminosity, N is the initial number of muons per bunch (one positive and one negative), f_0 is the collision frequency (two detectors), DC is the duty cycle with a 15 Hz repetition rate, and β^* is the betatron function in the collision region. The initial 6D emittance must be small enough, the potato slicer does not cool muons. The longitudinal emittance is as large as can be tolerated by the $\sigma_p/p = 10^{-3}$ chromaticity requirement [25] of a 1.5 TeV/c² muon collider final focus with round 750 GeV beams, $\beta = 0.99999999$, $\gamma = E/m_\mu$, and a 10 mm long collision region.

$$\epsilon_{L,N} = (\sigma_p/p) \Delta z (\beta \gamma) = 10^{-3} 10\text{mm} 7000 = 70 \text{ mm} \quad (4)$$

Acknowledgements

Many thanks to Yuri Alexahin, Chuck Ankenbrandt, Scott Berg, Chandra Bhat, Alex Bogacz, Moses Chung, Mary Anne Cummings, Jean-Pierre Delahaye, Ben Freemire, Carol Johnstone, Trey Lyons, Bob Palmer, Mark Palmer, Philippe Piot, Tom Roberts, Robert Ryne, Hisham Sayed, Pavel Snopok, Diktys Stratakis, Mike Syphers, Yagmur Torun, and Katsuya Yonehara for many useful conversations.

Appendix A: ICOOL solenoid and skew quadrupole triplet code

for001.dat input file

```
solenoid and skew-quadrupole triplet (Dec. 31, 2014)
&cont npart=1000 / ! 1000 particles
&bmt / ! beam definition
1 2 1. 1 ! 1 mu frac gaussian
0.0 0.0 0.0 0.0 0.0 0.2 ! mean: x y z px py pz
0.0075 0.0075 0.0169 0.0014088 0.0014088 0.010 ! sigmas
0 ! no beam correlations
! particle interactions
```

```

&ints /
! histograms: make one z histogram,
! plot Bz vs. z for one muon, delta_z = 0.00342857 m, 70 z bins, 33 indicates Bz
&nhs /
&nsc /
&nzh nzhist=1 /
1 0. 0.00342857 70 0. 0. 33
&nrh /
&nem /
&ncv /
SECTION ! start problem definition
SREGION ! define region of solenoid
0.24 1 0.001 ! length, 1 radial region, radialstep
1 0.0 0.5 ! 1 radial region, min. and max.radii
SOL
4 2.5 0.32 0.0 0.035 0 0 0 0 0 0 0 0 0 0
! sheet model, central Bz,
! sheet length, sheet offset,
! sheet radius
VAC
CBLOCK
0 0 0 0 0 0 0 0 0 0
SREGION ! define region of 1st skew-quadrupole
! with 3rd-order fringe field
0.2 1 0.001 ! length, 1 radial region, radial step
1 0.0 0.5 ! 1 radial region, min. and max. radii
SQUA
2 25.74 0.1 0 0.05 0.01 0 0 0 0 0 0 0 0
! dtanh model, T/m, central length,
! 0, end length, end atten
VAC
CBLOCK
0 0 0 0 0 0 0 0 0 0
SREGION ! define region of 2nd skew-quadrupole
! with 3rd-order fringe field
0.300 1 0.001 ! length, 1 radial region, radial step
1 0.0 0.5 ! 1 radial region, min. and max. radii
SQUA
2 -9.033 0.2 0 0.075 0.015 0 0 0 0 0 0 0 0
! dtanh model, T/m, central length,
! 0, end length, end atten
VAC
CBLOCK
0 0 0 0 0 0 0 0 0 0
OUTPUT ! for009.dat output at end of 3rd

```

```

! skew-quadrupole
SREGION ! define region of 3rd skew-quadrupole
! with 3rd-order fringe field
0.700 1 0.001 ! length, 1 radial region, radial step
1 0.0 1.5 ! 1 radial region, min. and max. radii
SQUA
2 1.2426 0.5 0 0.075 0.015 0 0 0 0 0 0 0 0
! dtanh model, T/m, central length,
! 0, end length, end atten
VAC
CBLOCK
0 0 0 0 0 0 0 0 0 0
ENDSECTION

```

Appendix B: G4beamline file to set skew quadrupole focal lengths

The output of Appendix B is fed into Appendix C.

```

* Simulate muon beam starting in solenoid coil. The beam will exit
* the solenoid with angular momentum and then continue through
* three skew-quadrupoles.
*
* The output file, in one of two formats as explained later, is the particle
* information at z = 0 mm and 80 mm past the end of a solenoid.
*
* The output necessary for the C++ program which optimizes
* skew-quadrupole triplet strengths for minimum emit_x_N is Z0.txt
*
* The output necessary for running ICOOL is for009.dat which
* must be modified as instructed and renamed for003.dat
*
* February 24, 2015
*
# use TJR recommended physics routines
# no stochastic processes, particle decays turned off
physics QGSP_BERT_EMX doStochastics=0 disable=Decay list=1

# Tom Roberts bux fix
bug1021

# sets background color to white when G4Beamline is run in visualization mode
# by clicking the 'best' button for Viewer.
g4ui when=4 "/vis/viewer/set/background 1 1 1"

```

```

# set reference particle parameters:
# reference particle is positive muon set to red in visualization,
particlecolor reference=1,0,0 mu+=1,0,0
reference referenceMomentum=200 particle=mu+ \
    beamX=0.0 beamY=0.0 beamZ=-200 meanXp=0.0 meanYp=0.0 rotation=X0.0,Z0.0

# set normalized emittances and magnetic emittance
# e_TR_N = 100 mm-mrad and e_mag_N = 200 mm-mrad for
# Bz = 2.5 T and sigma(x,y) = 7.5 mm
# e_L_N = 1.6 mm
beam gaussian particle=mu+ nEvents=1000 \
    beamX=0.0 beamY=0.0 beamZ=-200 meanXp=0 meanYp=0 \
    sigmaX=7.5 sigmaY=7.5 sigmaZ=0 sigmaXp=0.007044 sigmaYp=0.007044 \
meanMomentum=200 sigmaP=10 meanT=0 sigmaT=0.06365 rotation=X0,Z0.0

# Keep only positive muons, and set maximum time to 100 ns.
trackcuts keep=mu+ maxTime=100.0

# This sets the output ntuple at z = 0 mm.
# There are two choices for the output format:
# 1) format=root
#     This embeds Z0 into g4beamline.root. In this file,
#     the C++/Root skew-quadrupole triplet focal length optimizer can
#     be run from the Z0 output in g4beamline.root.
#
# 2) format=for009
#     This format is close to the beam input format necessary for ICOOL.
#     To use this file at in ICOOL input, the output file name, Z0.txt,
#     needs to be changed to for003.dat, and the file entries have to be
#     rearranged to the proper format. The necessary changes are
#     a) Change file name from Z0.txt to for003.dat
#     b) Remove two of the first three comments lines and write a
#         desired comment for the first line.
#     c) For 2nd line to the last line:
#         i) Switch the 5th and 6th columns.
#         ii) Remove the 13th - 20th columns.

# This file has all the kinematic variables for all muons at z = 0 which is
# 80 mm past the end of the solenoid and at the start of the first quadrupole.
zntuple format=root z=0

# This file has all the kinematic variables for all muons at z = 1100 mm which is
# at the end of the third quadrupole.
zntuple format=root z=1100

```

```

# file showing paths for 10 muons
trace nTrace=10 format=root

# tracking and stepping parameters
param maxStep=0.5 SteppingVerbose=1

# Define solenoid coil
coil solenoid_coil innerRadius=25 outerRadius=35 length=320 material=Vacuum \
filename=coilname.dat

# Define solenoid consisting of coil and current in A/mm^2
solenoid beam_spinner current=203.15 color=0,0,1 kill=1 coilName=solenoid_coil

# Define three skew-quadrupoles
genericquad quad1 ironColor=0,1,0 fieldLength=100 ironLength=100 ironRadius=40 \
apertureRadius=30 \
    gradient=-(2545/100) fringe=0 kill=1

genericquad quad2 ironColor=0,1,0 fieldLength=200 ironLength=200 ironRadius=60 \
apertureRadius=50 \
    gradient=-(1667/200) fringe=0 kill=1

genericquad quad3 ironColor=0,1,0 fieldLength=500 ironLength=500 \
ironRadius=110 apertureRadius=100 \
    gradient=-(559/500) fringe=0 kill=1

# place solenoid magnet
place beam_spinner z=-240

# place three skew-quadrupoles
place quad1 x=0 y=0 z=50 rotation=Z45
place quad2 x=0 y=0 z=50+250 rotation=Z-45
place quad3 z=0 y=0 z=50+250+500 rotation=Z45

```

Appendix C: C++/Root code to set skew quadrupole focal lengths

Output from Appendix B is used.

```

#define test_cxx
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

```

```
// February 24, 2015
```

```

// g4beamline_muons_from_solenoid_coil.txt must be run on
// G4Beamline to produce the necessary Z0.txt. input for
// this routine.
//
// Directory names must be changed as indicated in later instructions.
//
// This is a C++/Root routine which determines the best set of three
// focal lengths of skew-quadrupoles that minimizes the final
// normalized x emittance of a beam after the third quadrupole. The
// input file is Z0 embedded in g4beamline.root. This file generates
// histograms of various kinematic quantities at the ends of each of the
// three quadrupoles. The screen output shows the best set of focal
// lengths as well as other quantities.
//
// Instructions for running this routine:
// 1) Open root g4beamline.root (generated by running G4Beamline on
//     g4beamline_muons_from_solenoid_coil.txt) with root.
// 2) Select Z0.txt from g4beamline.root directory tree.
// 3) Type the following commands at root prompts
//     a) .L skew-quad_triplet_matrix_optimization.C [the
//         name of this file which can be changed]
//     b) test t
//     c) t.Loop()
//
// This runs the program executing a loop and outputting text to the screen
// until the program is done.

// C++/Root declarations
class test {
public :
    TTree          *fChain;  //!pointer to the analyzed TTree or TChain
    Int_t          fCurrent; //!current Tree number in a TChain

    // Declaration of leaf types
    Float_t        x;        // add comments here describing the variable
    Float_t        y;        //
    Float_t        z;        //
    Float_t        Px;       //
    Float_t        Py;       //
    Float_t        Pz;       //
    Float_t        t;        //
    Float_t        PDGid;    //

```

```

Float_t      EventID;    //
Float_t      TrackID;    //
Float_t      ParentID;   //
Float_t      Weight;     //

// List of branches
TBranch      *b_x;      //!
TBranch      *b_y;      //!
TBranch      *b_z;      //!
TBranch      *b_Px;     //!
TBranch      *b_Py;     //!
TBranch      *b_Pz;     //!
TBranch      *b_t;      //!
TBranch      *b_PDGI;   //!
TBranch      *b_EventID; //!
TBranch      *b_TrackID; //!
TBranch      *b_ParentID; //!
TBranch      *b_Weight; //!

test(TTree *tree=0);
virtual ~test();
virtual Int_t  Cut(Long64_t entry);
virtual Int_t  GetEntry(Long64_t entry);
virtual Long64_t LoadTree(Long64_t entry);
virtual void   Init(TTree *tree);
virtual void   Loop();
virtual Bool_t  Notify();
virtual void   Show(Long64_t entry = -1);
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                                                                                              //
//  The directory names must be changed to directory containing g4beamline.root.  //
//                                                                                                                              //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
test::test(TTree *tree)
{
// if parameter tree is not specified (or zero), connect the file
// used to generate this class and read the Tree.
  if (tree == 0) {
    TFile *f = (TFile*)gROOT->GetListOfFiles()->
FindObject
("C:/Users/Terry Hart/Documents/6D muon cooling/appendices/g4beamline.root");
    if (!f) {
      f =

```

```

new TFile
("C:/Users/Terry Hart/Documents/6D muon cooling/appendices/g4beamline.root");
    f->cd
("C:/Users/Terry Hart/Documents/6D muon cooling/appendices/g4beamline.root:/NTuple");
    }
    tree = (TTree*)gDirectory->Get("Z0");

    }
    Init(tree);
}

test::~~test()
{
    if (!fChain) return;
    delete fChain->GetCurrentFile();
}

Int_t test::GetEntry(Long64_t entry)
{
// Read contents of entry.
    if (!fChain) return 0;
    return fChain->GetEntry(entry);
}

Long64_t test::LoadTree(Long64_t entry)
{
// Set the environment to read one entry
    if (!fChain) return -5;
    Long64_t centry = fChain->LoadTree(entry);
    if (centry < 0) return centry;
    if (!fChain->InheritsFrom(TChain::Class())) return centry;
    TChain *chain = (TChain*)fChain;
    if (chain->GetTreeNumber() != fCurrent) {
        fCurrent = chain->GetTreeNumber();
        Notify();
    }
    return centry;
}

void test::Init(TTree *tree)
{
// The Init() function is called when the selector needs to initialize
// a new tree or chain. Typically here the branch addresses and branch
// pointers of the tree will be set.
// It is normally not necessary to make changes to the generated

```

```

// code, but the routine can be extended by the user if needed.
// Init() will be called many times when running on PROOF
// (once per file to be processed).

// Set branch addresses and branch pointers
if (!tree) return;
fChain = tree;
fCurrent = -1;
fChain->SetMakeClass(1);

fChain->SetBranchAddress("x", &x, &b_x);
fChain->SetBranchAddress("y", &y, &b_y);
fChain->SetBranchAddress("z", &z, &b_z);
fChain->SetBranchAddress("Px", &Px, &b_Px);
fChain->SetBranchAddress("Py", &Py, &b_Py);
fChain->SetBranchAddress("Pz", &Pz, &b_Pz);
fChain->SetBranchAddress("t", &t, &b_t);
fChain->SetBranchAddress("PDGid", &PDGid, &b_PDGid);
fChain->SetBranchAddress("EventID", &EventID, &b_EventID);
fChain->SetBranchAddress("TrackID", &TrackID, &b_TrackID);
fChain->SetBranchAddress("ParentID", &ParentID, &b_ParentID);
fChain->SetBranchAddress("Weight", &Weight, &b_Weight);
Notify();
}

Bool_t test::Notify()
{
// The Notify() function is called when a new file is opened. This
// can be either for a new TTree in a TChain or when when a new TTree
// is started when using PROOF. It is normally not necessary to make changes
// to the generated code, but the routine can be extended by the
// user if needed. The return value is currently not used.

return kTRUE;
}

void test::Show(Long64_t entry)
{
// Print contents of entry.
// If entry is not specified, print current entry
if (!fChain) return;
fChain->Show(entry);
}

Int_t test::Cut(Long64_t entry)
{

```

```

// This function may be called from Loop.
// returns 1 if entry is accepted.
// returns -1 otherwise.
    return 1;
}
#endif // #ifdef test_cxx

// Code that does the loop over focal lengths determining optimum
// skew-quadrupole triplet focal lengths
void test::Loop() {

// define histograms after Q1
    TH1F *h1 =new TH1F("h1","x after Q1", 100, -200, 200);
    TH1F *h2 =new TH1F("h2","y after Q1", 100, -200, 200);

    TH1F *h3 =new TH1F("h3","Px after Q1", 100, -100, 100);
    TH1F *h4 =new TH1F("h4","Py after Q1", 100, -100, 100);

// define histograms before and after Q2
    TH1F *h5 =new TH1F("h5","x before Q2", 100, -200, 200);
    TH1F *h6 =new TH1F("h6","y before Q2", 100, -200, 200);

    TH1F *h7 =new TH1F("h7","Px before Q2", 100, -100, 100);
    TH1F *h8 =new TH1F("h8","Py before Q2", 100, -100, 100);

    TH1F *h9 =new TH1F("h9","x after Q2", 100, -200, 200);
    TH1F *h10 =new TH1F("h10","y after Q2", 100, -200, 200);

    TH1F *h11 =new TH1F("h11","Px after Q2", 100, -100, 100);
    TH1F *h12 =new TH1F("h12","Py after Q2", 100, -100, 100);

// define histograms before and after Q3
    TH1F *h13 =new TH1F("h13","x before Q3", 100, -200, 200);
    TH1F *h14 =new TH1F("h14","y before Q3", 100, -200, 200);

    TH1F *h15 =new TH1F("h15","Px before Q3", 100, -100, 100);
    TH1F *h16 =new TH1F("h16","Py before Q3", 100, -100, 100);

    TH1F *h17 =new TH1F("h17","x after Q3", 100, -200, 200);
    TH1F *h18 =new TH1F("h18","y after Q3", 100, -200, 200);

    TH1F *h19 =new TH1F("h19","Px after Q3", 100, -100, 100);
    TH1F *h20 =new TH1F("h20","Py after Q3", 100, -100, 100);

```

```

// define histograms after Q3, x*Px, y*Py, x' vs. x, y' vs. y
TH1F *h21 =new TH1F("h21","x*Px after Q3", 100, -2500, 2500);
TH1F *h22 =new TH1F("h22","y*Py after Q3", 100, -2500, 2500);

TH2F *h23 =new TH2F("h23","Px vs. x after Q3", 1000, -200, 200,
                  1000, -50, 50);
TH2F *h24 =new TH2F("h24","Py vs. y after Q3", 1000, -200, 200,
                  1000, -50, 50);

TH2F *h25 =new TH2F("h25","Py vs. x after Q3", 1000, -200, 200,
                  1000, -40, 40);
TH2F *h26 =new TH2F("h26","Px vs. y after Q3", 1000, -200, 200,
                  1000, -40, 40);

TH1F *h27 =new TH1F("h27","x*Py after Q3", 100, -400, 400);
TH1F *h28 =new TH1F("h28","y*Px after Q3", 100, -400, 400);

TH1F *h29 =new TH1F("h29","x*y after Q3", 100, -2000, 2000);
TH1F *h30 =new TH1F("h30","Py*Px after Q3", 100, -100, 100);

TH2F *h31 =new TH2F("h31","y vs. x after Q3", 1000, -280.0, 280.0,
                  1000, -200.00, 200.00);
TH2F *h32 =new TH2F("h32","y vs. x after Q2", 1000, -280.0, 280.0,
                  1000, -200.00, 200.00);
TH2F *h33 =new TH2F("h33","y vs. x after Q1", 1000, -280.0, 280.0,
                  1000, -200.00, 200.00);

Long64_t nentries = fChain->GetEntriesFast();

Long64_t nbytes = 0, nb = 0;

// In a ROOT session, you can do:
//   Root > .L skew-quad_triplet_matrix_optimization.C
//   Root > test t
//   Root > t.GetEntry(12); // Fill t data members with entry number 12
//   Root > t.Show();      // Show values of entry 12
//   Root > t.Show(16);    // Read and show values of entry 16
//   Root > t.Loop();      // Loop on all entries
//
// This is the loop skeleton where:
//   jentry is the global entry number in the chain
//   ientry is the entry number in the current Tree
// Note that the argument to GetEntry must be:

```

```

//      jentry for TChain::GetEntry
//      ientry for TTree::GetEntry and TBranch::GetEntry
//
//      To read only selected branches, Insert statements like:
// METHOD1:
//      fChain->SetBranchStatus("*",0); // disable all branches
//      fChain->SetBranchStatus("branchname",1); // activate branchname
// METHOD2: replace line
//      fChain->GetEntry(jentry); //read all branches
//by   b_branchname->GetEntry(ientry); //read only this branch

    if (fChain == 0) return;

// Define constants
// L is distance between Q1 and Q2;
// M is distance between Q2 and Q3, all in mm.
    float L=250;
    float M=500;

// Lengths of (Q1, Q2, Q3) in mm
    float L1=100;
    float L2=200;
    float L3=500;

    cout << "quadrupole lengths: (" << L1 << "," << L2 << "," << L3 << ") mm"
         << endl;

// Set nominal focal lengths in mm
    float f1_nom=259;
    float f2_nom=-369;
    float f3_nom=1073;

// Adjusted focal lengths of (Q1, Q2, Q3)
    float f1;
    float f2;
    float f3;

// Pz in MeV/c
    float p_z;

// x, xp, y, yp arrays
    float x_array[251];
    float y_array[251];
    float xp_array[251];
    float yp_array[251];

```

```

// Variables for (x,x), (x,Px), (y,y), (y,Py),
// (Px,Py), (Py,Py), (x,y), (x,Py), (y,Px), (Px,Py) matrix elements
float sum_x_x;
float sum_x_xp;
float sum_y_y;
float sum_y_yp;

float sum_xp_xp;
float sum_yp_yp;

float sum_x_y;
float sum_x_yp;
float sum_xp_y;
float sum_xp_yp;

float sum_x;
float sum_y;
float sum_xp;
float sum_yp;

float cov_x_y;
float cov_x_yp;
float cov_xp_y;
float cov_xp_yp;

// set initial values
float chisq_lowest=12345678900000.0;
float chisq_best=12345678900000.0;
float emit_x_n_lowest=234567890000.01;
float emit_y_n_lowest=34567890000.012;
float emit_x_n_best=234567890000.01;
float emit_y_n_best=34567890000.012;

float f1_best=12.34;
float f2_best=23.45;
float f3_best=34.56;

float f1_best_exn=12.34;
float f2_best_exn=23.45;
float f3_best_exn=34.56;

// (a, b, c) are indices of 3-fold loop over focal lengths
float a_best;
float b_best;

```

```

float c_best;

float a_best_exn;
float b_best_exn;
float c_best_exn;

// Loop over (f1, f2, f3) variations
int a;
int b;
int c;
for (a=-2; a<3; a++) {
    for (b=-2; b<3; b++) {
        for (c=-2; c<3; c++) {

            cout << "a = " << a << ", " << " b = " << b << ", " << " c = " << c
                << endl;

            sum_x_x    = 0.0;
            sum_y_y    = 0.0;
            sum_x_xp   = 0.0;
            sum_y_yp   = 0.0;
            sum_xp_xp  = 0.0;
            sum_yp_yp  = 0.0;
            sum_x_y    = 0.0;
            sum_x_yp   = 0.0;
            sum_xp_y   = 0.0;
            sum_xp_yp  = 0.0;

            sum_x      = 0.0;
            sum_y      = 0.0;
            sum_xp     = 0.0;
            sum_yp     = 0.0;

// Define adjusted (f1, f2, f3) from nominal values
            f1 = f1_nom*(1.0+(a/100.0));
            f2 = f2_nom*(1.0+(b/100.0));
            f3 = f3_nom*(1.0+(c/100.0));

            cout << "f1 = " << f1 << ", " << " f2 = " << f2 << ", " << " f3 = "
                << f3 << endl;

// Loop over nentries
            for (Long64_t jentry=0; jentry<nentries;jentry++) {
                Long64_t ientry = LoadTree(jentry);

```

```

        if (ientry < 0) break;

        nb = fChain->GetEntry(jentry);
        nbytes += nb;

    p_z = Pz;

// Drift through 50 mm before reaching Q1
float xq1s = x + (Px/Pz)*50;
    float xpq1s = Px/Pz;
float yq1s = y + (Py/Pz)*50;
    float ypq1s = Py/Pz;

// Define variables for going through quadrupole
x_array[0] = xq1s;
y_array[0] = yq1s;
xp_array[0] = xpq1s;
yp_array[0] = ypq1s;

// Go through Q1 in 250 steps
int i = 0;
do {
    x_array[i+1] = x_array[i] + (L1/250.0)*xp_array[i];
    xp_array[i+1] = xp_array[i] + y_array[i]/(250.0*f1);

    y_array[i+1] = y_array[i] + (L1/250.0)*yp_array[i];
    yp_array[i+1] = yp_array[i] + x_array[i]/(250.0*f1);

    i = i + 1;

} while (i < 250);

float xq1e = x_array[250];
float yq1e = y_array[250];

float xpq1e = xp_array[250];
float ypq1e = yp_array[250];

// Define Q2 variables
// Drift from end of Q1 to start of Q2
float xq2s=xq1e+(L-(L1+L2)/2)*xpq1e;
float yq2s=yq1e+(L-(L1+L2)/2)*ypq1e;

```

```

// Maintain track angles from end of Q1 to start of Q2
    float xpq2s=xpq1e;
    float ypq2s=ypq1e;

// Initialize Q2 arrays for transport thorough Q2
    x_array[0] = xq2s;
    y_array[0] = yq2s;
    xp_array[0] = xpq2s;
    yp_array[0] = ypq2s;

// Go through Q2 in 250 steps
    i = 0;
    do {
        x_array[i+1] = x_array[i] + (L2/250.0)*xp_array[i];
        xp_array[i+1] = xp_array[i] + y_array[i]/(250.0*f2);

        y_array[i+1] = y_array[i] + (L2/250.0)*yp_array[i];
        yp_array[i+1] = yp_array[i] + x_array[i]/(250.0*f2);

        i = i + 1;

    } while (i < 250);

    float xq2e = x_array[250];
    float yq2e = y_array[250];

    float xpq2e = xp_array[250];
    float ypq2e = yp_array[250];

// Define Q3 variables
// Drift from end of Q2 to start of Q3
    float xq3s=xq2e+(M-(L2+L3)/2)*xpq2e;
    float yq3s=yq2e+(M-(L2+L3)/2)*ypq2e;

// Maintain track angles from end of Q2 to start of Q3
    float xpq3s=xpq2e;
    float ypq3s=ypq2e;

// Initialize Q3 arrays for transport through Q3
    x_array[0] = xq3s;
    y_array[0] = yq3s;
    xp_array[0] = xpq3s;
    yp_array[0] = ypq3s;

```

```

// Go through Q3 in 250 steps
    i = 0;
    do {
        x_array[i+1] = x_array[i] + (L3/250.0)*xp_array[i];
        xp_array[i+1] = xp_array[i] + y_array[i]/(250.0*f3);

        y_array[i+1] = y_array[i] + (L3/250.0)*yp_array[i];
        yp_array[i+1] = yp_array[i] + x_array[i]/(250.0*f3);

        i = i + 1;

    } while (i < 250);

    float xq3e = x_array[250];
    float yq3e = y_array[250];

    float xpq3e = xp_array[250];
    float ypq3e = yp_array[250];

// Drift through 100 mm after Q3
    xq3e = xq3e + (xpq3e)*100;
    xpq3e = xpq3e;
    yq3e = yq3e + (ypq3e)*100;
    ypq3e = ypq3e;

// Define mean x*Px and y*Py 100 mm after Q3
    float xpxq3e=xq3e*p_z*xpq3e;
    float ypyq3e=yq3e*p_z*ypq3e;

// Determine (x,x), (x,Px), (y,y), (y,Py), (Px,Px), (Py,Py),
// (x,y), (x,Py), (y,Px), (Px,Py) matrix elements
    sum_x_x = sum_x_x + xq3e*xq3e;
    sum_x_xp = sum_x_xp + xq3e*p_z*xpq3e;
    sum_y_y = sum_y_y + yq3e*yq3e;
    sum_y_yp = sum_y_yp + yq3e*p_z*ypq3e;
    sum_xp_xp = sum_xp_xp + p_z*xpq3e*p_z*xpq3e;
    sum_yp_yp = sum_yp_yp + p_z*ypq3e*p_z*ypq3e;
    sum_x_y = sum_x_y + xq3e*yq3e;
    sum_x_yp = sum_x_yp + xq3e*p_z*ypq3e;
    sum_xp_y = sum_xp_y + p_z*xpq3e*yq3e;
    sum_xp_yp = sum_xp_yp + p_z*xpq3e*p_z*ypq3e;

sum_x = sum_x + xq3e;

```

```

sum_y = sum_y + yq3e;
sum_xp = sum_xp + xpq3e;
sum_yp = sum_yp + ypq3e;

// Fill histograms only for nominal (fQ1, fQ2, fQ3) (a == b == c == 0)
if ((a == 0)&&(b == 0)&&(c == 0)) {
// Fill Q1 histograms
h1->Fill(xq1e);
h2->Fill(yq1e);

h3->Fill(p_z*xpq1e);
h4->Fill(p_z*ypq1e);

// Fill Q2 histograms
h5->Fill(xq2s);
h6->Fill(yq2s);

h7->Fill(p_z*xpq2s);
h8->Fill(p_z*ypq2s);

h9->Fill(xq2e);
h10->Fill(yq2e);

h11->Fill(p_z*xpq2e);
h12->Fill(p_z*ypq2e);

// Fill Q3 histograms
h13->Fill(xq3s);
h14->Fill(yq3s);

h15->Fill(p_z*xpq3s);
h16->Fill(p_z*ypq3s);

h17->Fill(xq3e);
h18->Fill(yq3e);

h19->Fill(p_z*xpq3e);
h20->Fill(p_z*ypq3e);

// Fill after Q3 x*Px, y*Py, Px vs. x, Py vs. y for normalized emittance
// determinations
h21->Fill(xpxq3e);
h22->Fill(ypyq3e);

h23->Fill(xq3e,p_z*xpq3e);

```

```

        h24->Fill(yq3e,p_z*ypq3e);

        h25->Fill(xq3e,p_z*ypq3e);
        h26->Fill(yq3e,p_z*xpq3e);

        h27->Fill(xq3e*p_z*ypq3e);
        h28->Fill(yq3e*p_z*xpq3e);

        h29->Fill(xq3e*yq3e);
        h30->Fill(p_z*xpq3e*p_z*ypq3e);

        h31->Fill(xq3e,yq3e);
        h32->Fill(xq2e,yq2e);
        h33->Fill(xq1e,yq1e);

    } // loop filling histograms only for
      // nominal (f1, f2, f3) (a = b = c = 0)

//      if (Cut(ientry) < 0) continue;

    } // loop over nentries

// Determine (x,x), (x,Px), (y,y), (y,Py), (Px,Px), (Py,Py),
// (x,y), (x,y'), (x',y), (x',y') matrix elements, chi-square, and
// normalized emittances
    float cov_x_x;
    float cov_x_xp;
    float cov_y_y;
    float cov_y_yp;
    float cov_xp_xp;
    float cov_yp_yp;
    float cov_x_y;
    float cov_x_yp;
    float cov_xp_y;
    float cov_xp_yp;

    float chisq;

    float emit_x_n;
    float emit_y_n;

    cov_x_x    = (sum_x_x/(float(nentries))) -
                 (sum_x/(float(nentries))*(sum_x/(float(nentries))));
    cov_xp_xp  = (sum_xp_xp/(float(nentries))) -
                 (sum_xp/(float(nentries))*(sum_xp/(float(nentries))));

```

```

cov_x_xp = (sum_x_xp/(float(nentries))) -
            (sum_x/(float(nentries)))*(sum_xp/(float(nentries)));
cov_y_p_yp = (sum_y_p_yp/(float(nentries))) -
              (sum_y_p/(float(nentries)))*(sum_yp/(float(nentries)));
cov_y_y = (sum_y_y/(float(nentries))) -
           (sum_y/(float(nentries)))*(sum_y/(float(nentries)));
cov_y_yp = (sum_y_yp/(float(nentries))) -
            (sum_y/(float(nentries)))*(sum_yp/(float(nentries)));
cov_x_y = (sum_x_y/(float(nentries))) -
           (sum_x/(float(nentries)))*(sum_y/(float(nentries)));
cov_x_yp = (sum_x_yp/(float(nentries))) -
            (sum_x/(float(nentries)))*(sum_yp/(float(nentries)));
cov_xp_y = (sum_xp_y/(float(nentries))) -
            (sum_xp/(float(nentries)))*(sum_y/(float(nentries)));
cov_xp_yp = (sum_xp_yp/(float(nentries))) -
             (sum_xp/(float(nentries)))*(sum_yp/(float(nentries)));

// Sum (x,y), (x,Py), (Px,y), (Px,Py) matrix elements to determine chi-square
chisq = cov_x_y*cov_x_y + cov_x_yp*cov_x_yp +
        cov_xp_y*cov_xp_y + cov_xp_yp*cov_xp_yp;

// Determine normalized emittances in mm-mrad
emit_x_n = (1000.0/105.66)*sqrt(cov_x_x*cov_xp_xp - cov_x_xp*cov_x_xp);
emit_y_n = (1000.0/105.66)*sqrt(cov_y_y*cov_yp_yp - cov_y_yp*cov_y_yp);

if (abs(chisq) < chisq_lowest) {
    chisq_lowest = abs(chisq);
    emit_x_n_best = emit_x_n;
    emit_y_n_best = emit_y_n;
    a_best = a;
    b_best = b;
    c_best = c;

    f1_best = f1;
    f2_best = f2;
    f3_best = f3;
}

if ((emit_x_n < emit_x_n_lowest)&&(emit_x_n > 0)&&(emit_y_n > 0)) {
    chisq_best = abs(chisq);
    emit_x_n_lowest = emit_x_n;
    emit_y_n_lowest = emit_y_n;
    a_best_exn = a;
    b_best_exn = b;
    c_best_exn = c;
}

```

```

        f1_best_exn = f1;
        f2_best_exn = f2;
        f3_best_exn = f3;
    }

    cout << " cov_x_x = " << cov_x_x << endl;
    cout << " cov_x_xp = " << cov_x_xp << endl;
    cout << " cov_x_y = " << cov_x_y << endl;
    cout << " cov_x_yp = " << cov_x_yp << endl;
    cout << " cov_xp_xp = " << cov_xp_xp << endl;
    cout << " cov_xp_y = " << cov_xp_y << endl;
    cout << " cov_xp_yp = " << cov_xp_yp << endl;
    cout << " cov_y_y = " << cov_y_y << endl;
    cout << " cov_y_yp = " << cov_y_yp << endl;
    cout << " cov_yp_yp = " << cov_yp_yp << endl;

    cout << " emit_x_n = " << emit_x_n << " mm-mrad " << endl;
    cout << " emit_y_n = " << emit_y_n << " mm-mrad " << endl;
    cout << " chisq = " << chisq << endl;
    cout << endl;

    } // loop over f3 adjustment
} // loop over f2 adjustment
} // loop over f1 adjustment

h31->Draw();

cout << "quadrupole lengths: (" << L1 << ", " << L2 << ", " << L3 << ") mm"
    << endl;

cout << "best (a,b,c) for chi-square = " << "(" << a_best << ", " << b_best
    << ", " << c_best << ")" << endl;
cout << "lowest chisq = " << chisq_lowest << " for (f1, f2, f3) = " << "("
    << f1_best << ", " << f2_best << ", " << f3_best << ")" << endl;
cout << "best emit_x_n = " << emit_x_n_best
    << " mm-mrad for (f1, f2, f3) = " << "(" << f1_best << ", " << f2_best
    << ", " << f3_best << ")" << endl;
cout << "best emit_y_n = " << emit_y_n_best
    << " mm-mrad for (f1, f2, f3) = " << "(" << f1_best << ", " << f2_best
    << ", " << f3_best << ")" << endl;
cout << endl;
cout << "best (a,b,c) for emit_x_n = " << "(" << a_best_exn << ", "
    << b_best_exn << ", " << c_best_exn << ")" << endl;
cout << "chisq for smallest emit_x_n = " << chisq_best

```

```

    << " for (f1, f2, f3) = " << "(" << f1_best_exn << "," << f2_best_exn
    << "," << f3_best_exn << ")" << endl;
cout << "lowest emit_x_n = " << emit_x_n_lowest
    << " mm-mrad for (f1, f2, f3) = " << "(" << f1_best_exn << ","
    << f2_best_exn << "," << f3_best_exn << ")" << endl;
cout << "lowest emit_y_n = " << emit_y_n_lowest
    << " mm-mrad for (f1, f2, f3) = " << "(" << f1_best_exn << ","
    << f2_best_exn << "," << f3_best_exn << ")" << endl;
} // bracket for void test::Loop()

```

Appendix D: Skew quadrupole triplet algebra

Define magnetic normalized emittance as

$$\epsilon_{mag,N} = \frac{eB_{solenoid} \sigma_{(x,y)}^2}{2mc}.$$

Define transverse intrinsic normalized emittance as

$$\epsilon_{TR,int,N} = \frac{p_{beam}}{mc} \sigma_{(x,y)} \sigma_{(x',y')} \text{ which assumes } \sigma_x = \sigma_y = \sigma_{(x,y)} \text{ and } \sigma_{x'} = \sigma_{y'} = \sigma_{(x',y')}.$$

When exiting solenoid, $(\epsilon_{x,N}, \epsilon_{y,N}) = (\epsilon_{TR,int,N}, \epsilon_{TR,int,N})$ is transformed to $(\epsilon_{TR,smaller,N}, \epsilon_{TR,larger,N})$

$$\text{with } \epsilon_{N,smaller} = \sqrt{\epsilon_{mag,N}^2 + \epsilon_{TR,int,N}^2} - \epsilon_{mag,N} \approx \frac{\epsilon_{TR,int,N}^2}{2\epsilon_{mag,N}} \text{ and}$$

$$\epsilon_{N,larger} = \sqrt{\epsilon_{mag,N}^2 + \epsilon_{TR,int,N}^2} + \epsilon_{mag,N} \approx 2\epsilon_{mag,N}$$

$$\text{Define } \beta_{\perp} = \frac{2p_{beam}}{eB_{solenoid}}.$$

$$f_{Q1} = \frac{\beta_{\perp}}{\sqrt{1 + \frac{\beta_{\perp}^2}{L(L+M)}}}$$

$$f_{Q2} = \frac{LM}{\beta_{\perp}} \left[1 + \sqrt{1 + \frac{\beta_{\perp}^2}{L(L+M)}} \right]$$

$$f_{Q3} = \frac{2M(L+M)}{\beta_{\perp}}$$

The relationship between the quadrupole focal length and integrated magnetic field is

$$\frac{p_{beam}}{ef_{quad}} = \int \frac{dB_y}{dx} dl = \int \frac{dB_x}{dt} dl \approx \frac{B_{pole}}{r_{pole}} l_{quad}$$

Appendix E: Skew Quadrupole Triplet for a Large Emittances.

The emittance exchange of a large 50 mm transverse emittance beam is simulated. Parameters are chosen (see Fig.2) to give an ideal emittance ratio of 16. The actual new nominal transverse normalized emittances are 13 mm and 295 mm. Round to flat beam transformations may have many uses. A beam with a low emittance in one dimension might be spread with a dipole to separate isotopes [27].

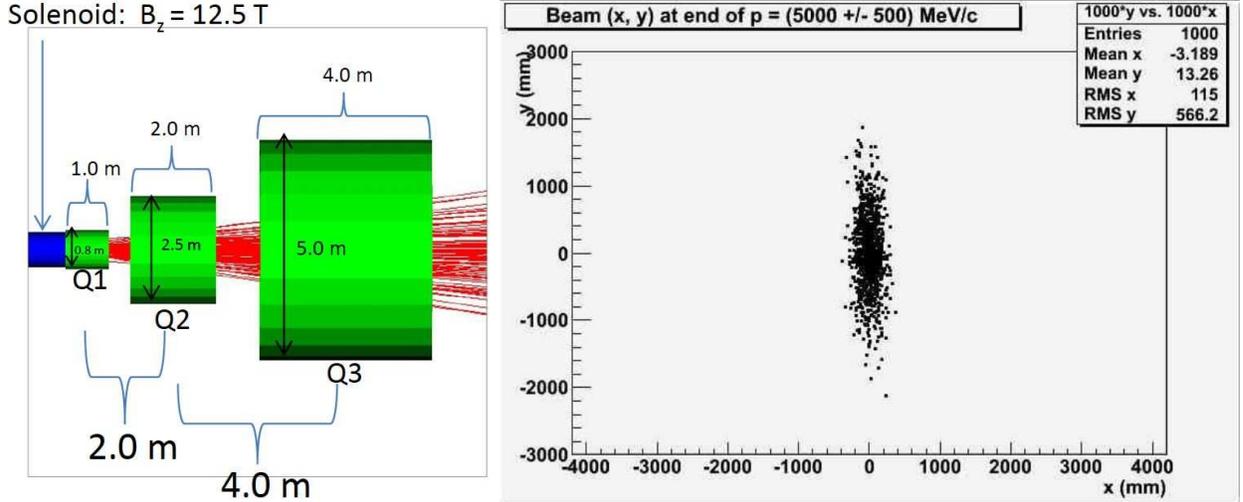


Figure 2: A round, spinning 5000 ± 500 MeV/c muon beam with a normalized transverse emittance of 50 mm is transformed to a flat, non-spinning muon beam with new normalized transverse emittances of 13 mm and 295 mm [14]. Inside the 12.5 T solenoid $\sigma_{x,y} = 75$ mm. The skew quadrupole pole tip fields are (4.49, 2.79, 0.684)m. The calculated thin quadrupole parameters must be adjusted somewhat to make finite length quadrupoles work properly. The simulation was done with G4Beamline [26] for the solenoid and ICOOL [17] which works better for quadrupoles.

Appendix F: Deflecting RF Transverse to Longitudinal Exchange

We explore the possibility of using deflecting RF cavities to transfer transverse emittance to longitudinal emittance. The desired normalized emittance transfer is $(\epsilon_{y,N}, \epsilon_{L,N}) \sim (400, 1600) \mu\text{m} \rightarrow (25, 25600) \mu\text{m}$ so that $(\epsilon_{y,N}/\epsilon_{L,N})$ is (decreased/increased) by a factor of 16. $\epsilon_{x,N}$ would be kept at $25 \mu\text{m}$. Figure 3 shows a schematic diagram of a box RF deflecting cavity. The initial muon momentum is roughly 200 ± 10 MeV/c, and the Gaussian beam position spread in y is 30 mm. Transforming the y and longitudinal normalized emittances by a factor of 16 with deflecting RF cavities involves increasing the beam momentum spread 16-fold to 160 MeV/c and establishing a correlation between y and the momentum. As this treatment is very approximate, the longitudinal quantity z' will be defined as δ_p/p_0 so that the following derivations may be missing a factor or two of β which is 0.884 for a 200 MeV/c muon.

An algorithm [28] is used for determining the optics of a triplet of deflecting RF cavities that will enact the transverse-longitudinal emittance transformation. Figure 4 shows the layout of a triplet of RF deflecting cavities.

There are an infinite number of solutions, and section VI-A of [28] includes a solution in which the three deflecting RF cavities are equally spaced and the normalized strength of the first deflecting RF cavity, b , is set equal to the correlation ratio between the energy deviation and the transverse position, $\zeta \equiv z'/y_0 = (\delta_p/p_0)/y_0$. Defining the normalized strengths of the deflecting RF cavities as k_1 , k_2 , and k_3 and the distances between the deflecting RF cavities as L :

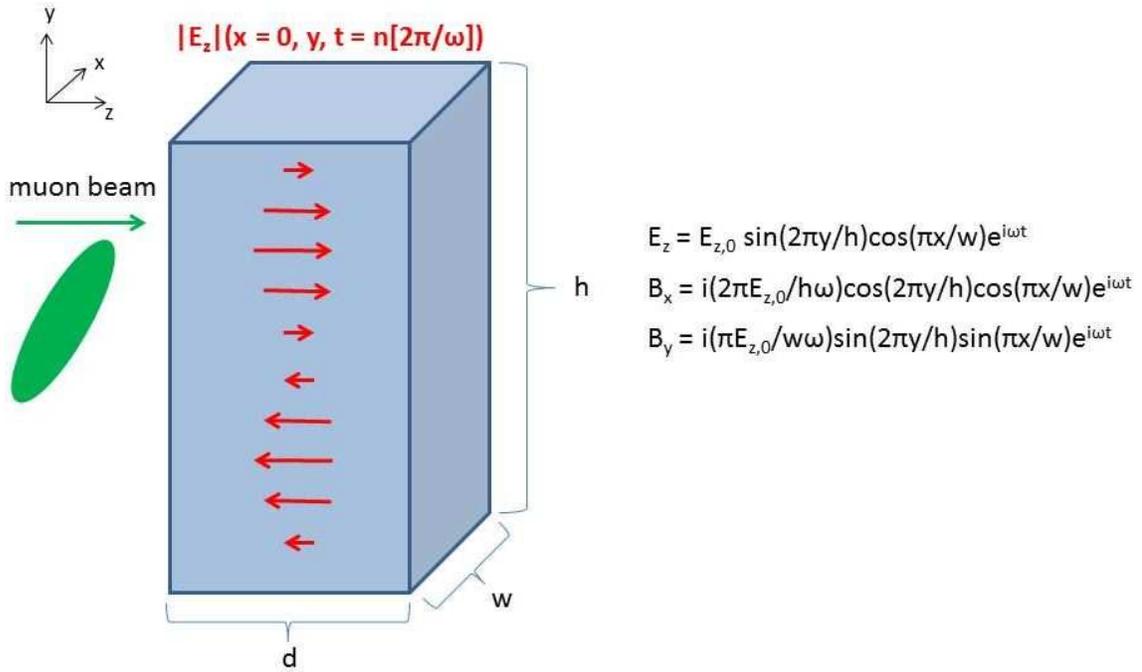
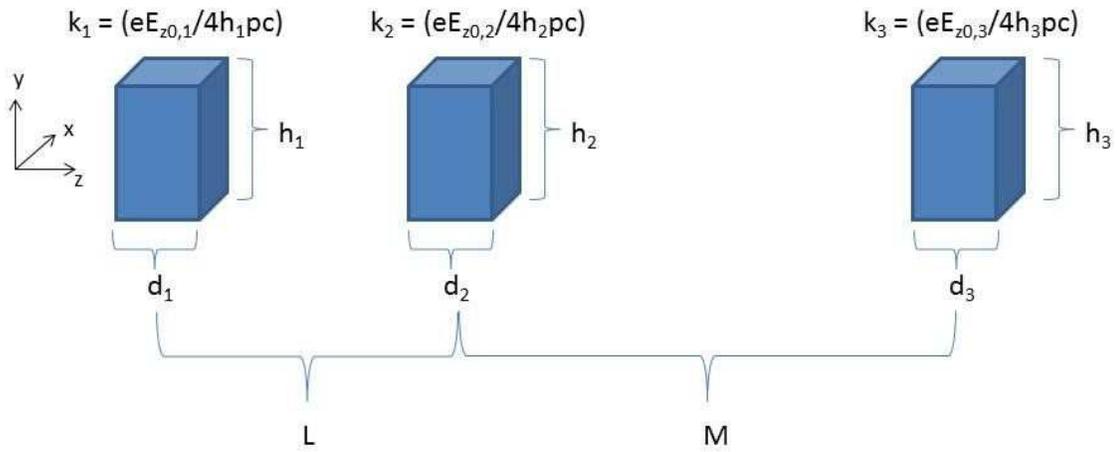


Figure 3: Schematic diagram of a box deflecting RF cavity with electromagnetic fields.



- Make $h_1, h_2, h_3 = 0.2$ m to contain incoming beam ($\sigma_y \sim 30$ mm).
- One solution for transforming $(y, y', z, \Delta p/p_0)$ from $(y_0, 0, z_0 + \lambda y_0, \zeta y_0)$ before 1st cavity $\rightarrow (0, 0, z_{larger}, (\Delta p/p_0)_{larger})$ after 3rd cavity
 - Set $L = M = 1/2\zeta$
 - Set $k_1 = \zeta, k_2 = -2\zeta, k_3 = 2\zeta$

Figure 4: RF deflecting cavity triplet that transforms transverse to longitudinal emittance.

$$\begin{aligned}
k_1 &= \zeta \text{ (first deflecting RF cavity)} \\
k_2 &= -2\zeta \text{ (second deflecting RF cavity)} \\
k_3 &= 2\zeta \text{ (third deflecting RF cavity)} \\
L &= 1/2\zeta \text{ (distances between RF cavities)}
\end{aligned}$$

Distances and strengths of the RF cavities will be now be worked out, which has $\zeta \sim (\delta_p/p_0)/y_0 \sim (160/200)/(0.03 \text{ m}) = 26.7/\text{m}$. Then, $L = \text{distance between cavities} = 1/(2 \times 26.7/\text{m}) = 0.01875 \text{ m}$. To capture the beam with $\sigma_y = 0.03 \text{ m}$, the deflecting RF cavity should be about $h = \text{total height} = 0.2 \text{ m}$ tall. The longitudinal electric field in a box deflecting RF cavity with total height h (along y) and total width w (along x) is $E_z = E_{z,0} \sin(2\pi y/h) \cos(\pi x/w)$.

Also, the total length, d , of deflecting RF cavities should be a fraction of the separation between the cavities, so that we can set $d = \text{total cavity length} = 0.01 \text{ m}$. Also, $d = \lambda_{RF}/2$ so that $\lambda_{RF} = 0.02 \text{ m}$ and $f_{RF} = 15 \text{ GHz}$. Finally, the normalized strength, k of a deflecting RF cavity is related to $E_{z,0}$ through $k = (eE_{z,0}d)/(4hp_0c)$ where d and h are the total length and total width of the deflecting RF cavity respectively. For $d = 0.01 \text{ m}$ and $h = 0.2 \text{ m}$,

$$\begin{aligned}
E_{z,0} &= 432,000 \text{ MV/m for the first deflecting RF cavity,} \\
E_{z,0} &= -864,000 \text{ MV/m for the second deflecting RF cavity,} \\
E_{z,0} &= 864,000 \text{ MV/m for the third deflecting RF cavity.}
\end{aligned}$$

The electric fields gradients required appear to be vastly too high to perform a transverse to longitudinal exchange for large emittances.

Appendix Z: Quadrupole Doublet Focusing for Final Muon Cooling

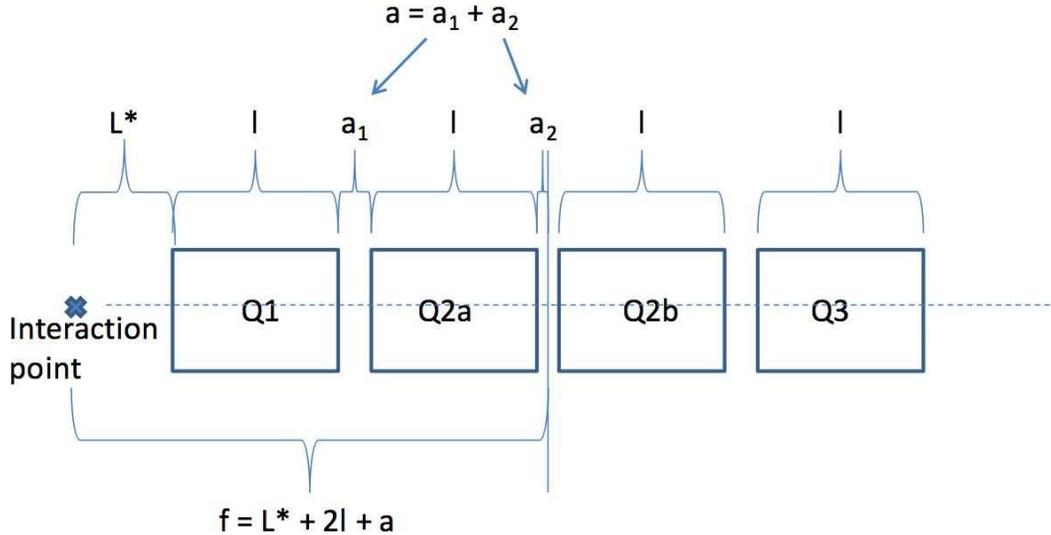


Figure 5: Geometry of a quadrupole triplet focusing system [29]. Each quadrupole length is ℓ . L^* is the distance from the interaction point to the front of the first quadrupole and a is an additional length for magnet interconnections. $f = L_f = L^* + 2\ell + a$ is focal length. Magnet length times gradient is proportional to beam momentum divided by focal length.

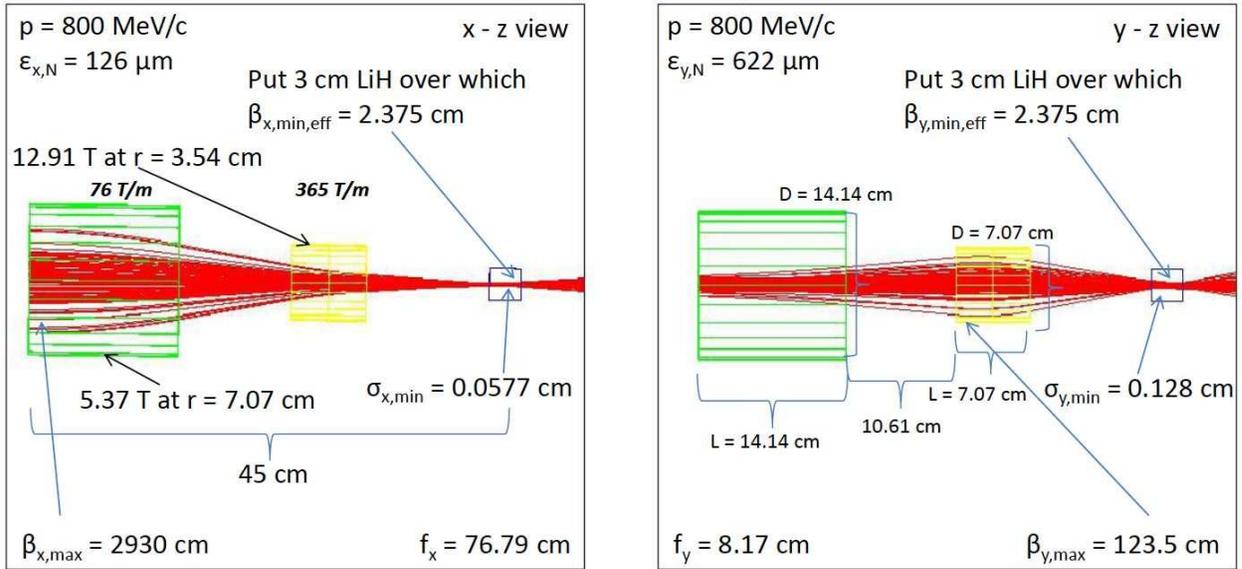


Figure 6: Quadrupole doublet half cell for final muon cooling with a flat beam, minimum $\beta_{x,y} = 2$ cm, and a 3 cm long LiH absorber. The G4beamline transmission through one half cell is 998/1000 and the coverage for quadrupoles is at least $\pm 3.2\sigma$.

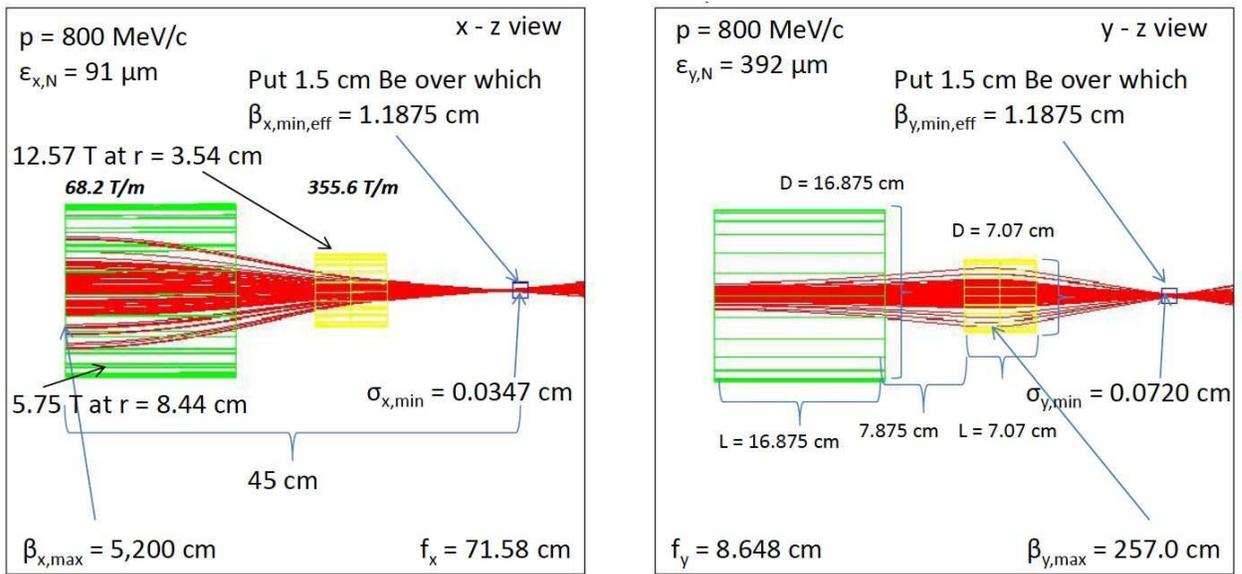


Figure 7: Quadrupole doublet half cell for final muon cooling with a flat beam, minimum $\beta_{x,y} = 1$ cm, and a 1.5 cm long beryllium absorber. The G4beamline transmission through one half cell is 999/1000 and the coverage for quadrupoles is at least $\pm 3.06\sigma$.

Following Feher and Strait [29] and their paper on hadron collider final focus quadrupole triplet design in 1996 with a β^* of 50 cm, we look into a short quadrupole doublet for final muon cooling with a β^* of 1 cm. Focal lengths in x and y are not the same in the doublet. The

LHC inner triplet is composed of four identical 5.5 m long quadrupoles as shown in Figure 5. The outer two quadrupoles are focusing in the first transverse dimension and defocusing in the second transverse dimension. The inner two quadrupoles are focusing in the second transverse dimension and defocusing in the first transverse dimension. Add 0.3 m for trim coils and take a quadrupole length, ℓ , of 5.8 m. The focusing strength to be proportional to the lengths of the four quadrupoles 4ℓ , times field gradient G in T/m, times focal length ($L_f = L^* + 2\ell + a$) in meters, divided by beam momentum p in TeV/c. L^* is the distance from the interaction point to the front of the first quadrupole and a is additional length for magnet interconnections. The focal length is just the distance from the interaction point to the center of the quadrupole triplet as shown in Figures 6 and 7. The relation between β functions and the focal length is given by $\beta_{\max} = bL_f^2/\beta^*$, where b is a fudge factor equal to 1.65 for the LHC.

A short length of low Z absorber is placed at the focus of each quadrupole doublet as shown in Figures 6 and 7. Flat beams are used with the $\sin(2\theta)$ quadrupole doublets which do not exceed 14 T as in the LHC Nb₃Sn LARP quadrupoles [30]. Note that $\beta(s) = \beta^* + s^2/\beta^*$. As β^* becomes smaller, the absorber must become thinner in the beam direction s , so one may want to employ beryllium or diamond. The fringe fields of the magnet fall off as the cube of distance [31] and may be small enough to not cause breakdown even in vacuum RF. The beam power of 4×10^{12} 800 MeV/c muons (KE = 701 MeV) arriving at 15 Hz is 6700 watts of which only a tiny fraction would heat the superconductor even in the absence of shielding.

In summary, quadrupole doublets and dense, low Z absorbers are being examined to cool the current outputs given either by the Helical [3] or Rectilinear [4] 6D muon cooling channels and prepare the input for the potato slicer which reduces the normalized transverse beam emittance to the $25 \mu\text{m}$ size required by a high luminosity muon collider. A more complete simulation of a tapered quadrupole doublet channel is in progress.

References

- [1] G. Budker, Conf.Proc. C690827 (1969) 33;
 A. N. Skrinsky, Morges 1971, AIP Conf. Proc. **352** (1996) 6;
 D. V. Neuffer and R. B. Palmer, EPAC 94, BNL-61267;
 D. J. Summers, Bull. Am. Phys. Soc. **39** (1994) 1818;
 R. Palmer *et al.*, Nucl. Phys. Proc. Suppl. **51A** (1996) 61;
 C. M. Ankenbrandt *et al.*, Phys. Rev. ST Accel. Beams **2** (1999) 081001;
 M. M. Alsharo'a *et al.*, Phys. Rev. ST Accel. Beams **6** (2003) 081001;
 R. B. Palmer *et al.*, PAC 07, arXiv:0711.4275.
- [2] E. Eichten and A. Martin, Phys. Lett. **B728** (2014) 125.
- [3] C. Yoshikawa *et al.*, IPAC-2014-TUPME016;
 Y. Derbenev and R. P. Johnson, Phys. Rev. ST Accel. Beams **8** (2005) 041002.
- [4] D. Stratakis *et al.*, IPAC-2014-TUPME020;
 D. Stratakis *et al.*, Phys. Rev. ST Accel. Beams **18** (2015) 044201;

- D. Stratakis and R. B. Palmer, Phys. Rev. ST Accel. Beams **18** (2015) 031003;
V. Balbekov, MAP-DOC-4365 (2013);
D. Stratakis *et al.*, Phys. Rev. ST Accel. Beams **16** (2013) 091001;
R. B. Palmer *et al.*, Phys. Rev. ST Accel. Beams **8** (2005) 061003.
- [5] Diktys Stratakis, private communication.
- [6] A. Moretti *et al.*, LINAC04, Conf.Proc. C0408164 (2004) 271.
- [7] M. Chung *et al.*, Phys. Rev. Lett. **111** (2013) 184802;
B. Freemire *et al.*, IPAC-2014-THPRI064;
D. Neuffer and K. Paul, EPAC-2006-WEPLS012.
- [8] D. Stratakis, IPAC-2014-TUPME024;
J. C. Gallardo and M. S. Zisman, AIP Conf. Proc. **1222** (2010) 308.
- [9] K. Yonehara, “Emittance growth by gas in a hybrid channel (preliminary),”
13 Jan 2015, MAP D&S Meeting,
<https://indico.fnal.gov/conferenceDisplay.py?confId=9267>
- [10] J. C. Gallardo *et al.*, Snowmass 96, BNL-52503, pp. 245-250.
- [11] H. K. Sayed, IPAC-2014-TUPME019;
D. Neuffer, Advanced Accelerator Concepts 2014, arXiv:1502.02709;
D. Neuffer, NuFact 2014, MAP-DOC-4403;
D. V. Neuffer *et al.*, IPAC-2015-TUBD2;
D. Summers *et al.*, IPAC-2015-TUPWI044;
D. Stratakis *et al.*, IPAC-2015-THPF153.
- [12] David Neuffer, arXiv:1312.1266;
D. Neuffer, Nucl. Instrum. Meth. **A532** (2004) 26;
David Neuffer, Part. Accel. **14** (1983) 75;
A. N. Skrinsky and V. V. Parkhomchuk, Sov. J. Part. Nucl. **12** (1981) 223.
- [13] <http://pdg.lbl.gov/2014/AtomicNuclearProperties/>
- [14] R. Brinkmann, Y. Derbenev, and K. Flöttmann, Phys. Rev. ST Accel. Beams **4** (2001) 053501; B. E. Carlsten and K. E. Bishofberger, New J. Phys. **8** (2006) 286.
- [15] J. Zhu, P. Piot, D. Mihalcea, and C. R. Prokop, Phys. Rev. ST Accel. Beams **17** (2014) 084401.
- [16] D. Edwards and M. Syphers, “An Introduction to the Physics of High Energy Accelerators,” (1993) p. 126.
- [17] R. C. Fernow, PAC 2005, Conf. Proc. C0505161 (2005) 2651.
- [18] M. Aicheler *et al.*, “A Multi-TeV Linear Collider Based on CLIC Technology : CLIC Conceptual Design Report,” CERN-2012-007, p. 32;
Robert Corsini *et al.*, Phys. Rev. ST Accel. Beams **7** (2004) 040101.

- [19] David Neuffer, Nucl. Instrum. Meth. **A503** (2003) 374;
G. Schröder, “Fast Pulsed Magnet Systems,” CERN-SL-98-17-BT (1998).
- [20] I. Kourbanis, G. P. Jackson, and X. Lu, Conf. Proc. C930517 (1993) 3799;
G. W. Foster, FERMILAB-TM-1902 (1994).
- [21] R. P. Johnson, C. Ankenbrandt, C. Bhat, M. Popovic, S. A. Bogacz, and Y. Derbenev,
“Muon Bunch Coalescing,” PAC 07-THPMN095.
- [22] S. Stahl and J. MacLachlan, FERMILAB-TM-1650 (1990).
- [23] Alex Bogacz, “Lattices for Bunch Coalescing,” Low Emittance Muon Collider Workshop,
Fermilab, 6-10 Feb 2006, MAP-DOC-4406,
<http://map-docdb.fnal.gov:8080/cgi-bin/RetrieveFile?docid=4406>.
- [24] Chandra Bhat, private communication.
- [25] Y. Alexahin *et al.*, IPAC-2010, arXiv:1202.0198.
- [26] T. J. Roberts *et al.*, EPAC 08, Conf. Proc. C0806233 (2008) WEPP120.
- [27] P. Bertrand *et al.*, “Flat beams and application to the mass separation of radioactive
beams,” Conf. Proc. C060626 (2006) 1687.
- [28] B. E. Carlsten, K. A. Bishofberger, L. D. Duffy, S. J. Russell, R. D. Ryne, N. A. Yampol-
sky, and A. J. Dragt, “Arbitrary emittance partitioning between any two dimensions for
electron beams,” Phys. Rev. ST Accel. Beams **14** (2011) 050706.
- [29] S. Feher and J. Strait, “Estimated Inner Triplet Quadrupole Length and Aperture for
Really Large Hadron Colliders of $E_{\text{beam}} = 30, 60$ and 100 TeV,” Snowmass-1996-ACC042.
- [30] F. Borgnolutti *et al.*, “Fabrication of a Second-Generation of Nb₃Sn Coils for the LARP
HQ02 Quadrupole Magnet,” IEEE Trans. Appl. Supercond. **24** (2014) 4003005.
- [31] C. Johnstone, M. Berz, D. Errede, and K. Makino, “Muon beam ionization cooling in a
linear quadrupole channel” (Fig. 5 on page 479), Nucl. Instrum. Meth. **A519** (2004) 472.