

Using BEAMLINE/MXYZPTLK class libraries to partitioned a lattice and construct its maps

Leo Michelotti
Fermilab
Beam Physics Department

Ordered Libraries of C++ Classes

First version of three core libraries was written in 1988-1989.

basic_toolkit:

Matrix, Vector, dlist, Frame, Barnacle, ...

Low level algebraic, container, and utility classes. Includes header files MathConstants.h and PhysicsConstants.h for uniformity across applications.

MXYZPTLK:

Jet, LieOperator, Mapping, ...

Algebraic functors that, among other things, implement automatic differentiation and differential algebra.

LieOperator and Mapping are crucial for doing normal form calculations.

BEAMLINE:

drift, Slot, sbend, rbend, quadrupole, ..., beamline

Classes to model electric and magnetic elements of a beamline.

[Jet]Proton, [Jet]Electron, [Jet]Muon, ...

Particle classes acted upon by the Propagators. To date, only the [Jet]Proton class has been developed fully.

FEATURE: Much of BEAMLINE is hardwired for Protons.

[Deep]BeamlineIterator, BmlVisitor

Iterators and base class for visitor pattern.

Propagators

All physics is isolated into small Propagator functors which connect Particles with beamline elements.

Ordered Libraries of C++ Classes (cont.)

Libraries of utility classes were added to make it easier to write applications.

physics_toolkit:

Sage, LattFuncSage, ClosedOrbitSage, ...

“Expert” or “wizard” at doing a particular calculation.

TuneAdjuster, Chromaticity Adjuster, ...

Classes for controlling beamline parameters.

FramePusher, DriftEliminator, FPSolver, ...

Utility classes, usually derived from BmlVisitor.

BeamlineContext

Organizer class (facade) for controlling all manipulations and calculations done on a beamline.

bmlfactory:

Factory class that constructs instances of beamline from descriptions in a MAD input file. Designed and written by Dmitri Mokhov, Oleg Krivosheev, and Francois Ostiguy using yacc.

File: boosterv6.cfg (fragments of)

```
drift*    0    = new drift( "0", 0.25 );
drift*    00   = new drift( "00", 0.30 );
drift*    000 = new drift( "000", 1.50 );
thinQuad* ED  = new thinQuad( "ED", - 0.00062639*(296.5/10.) );
thinQuad* EF  = new thinQuad( "EF", - 0.00086636*(296.5/10.) );
sbend     Dhf ( "Dhf", 0.722403/npt,           // length      [ m ]
                6.1727/10.,                  // field       [ T ]
                xpt*( 6.1727*0.722403 / 296.5 ) // bend angle [ rad ]
                );
sbend     Fhf ( "Fhf", 0.722403/npt,
                7.2588/10.,
                xpt*( 7.2588*0.722403 / 296.5 )
                );
thinQuad  QDhf( "QDhf", ( -(17.1101/10.)*0.722403 )/( npt - 1.0 ) );
thinQuad  QFhf( "QFhf", ( (16.0761/10.)*0.722403 )/( npt - 1.0 ) );

beamline* D = new beamline( "D" );
beamline* F = new beamline( "F" );
int i;
for( i = 0; i < ipt - 1; i++ )
{ D->append( Dhf );  D->append( QDhf );
  F->append( Fhf );  F->append( QFhf );
} D->append( Dhf );  F->append( Fhf );

// Build the booster cell
beamline  cell( "cell" );
  cell.append( *00  );
  cell.append( *00  );
  cell.append( *EF  );
  for( i = 0; i < 4; i++ )
  {  cell.append( *F  );
    cell.append( *EF  );
    cell.append( *0  );
    cell.append( *0  );
    cell.append( *ED  );
  for( i = 0; i < 4; i++ )
  {  cell.append( *D  );
    cell.append( *ED  );
    ...
    etc., etc.

    cell.append( *00  );
    cell.append( *00  );
```

Partitioning the booster: demo 3 (Matrices).

Step 1: Construct a model of the booster.

```
#include <iostream.h>
#include <iomanip.h>

#include "beamline.h"

int main( int argc, char* argv[] )
{
    // Process command line
    if( argc != 2 ) {
        cout << "Usage: " << argv[0]
            << " <number of markers (int)>"
            << endl;
        exit(0);
    }
    else {
        cout << "Command line: ";
        for( int i = 0; i < argc; i++ ) {
            cout << argv[i] << " ";
        }
        cout << "\n" << endl;
    }

    int number0fMarkers = atoi( argv[1] );

    // Create the booster model
    // ... Variables used by .cfg file in constructing the booster cell.
    const int ipt = 4;
    double npt      = ipt;
    double xpt      = 1.0/npt;

#include "boosterv6.cfg"

// beamline cell is created within boosterv6.cfg.
// Construct a booster model from 24 cells.
int i;
beamline booster;
for( i = 0; i < 24; i++ ) {
    booster.append( cell.Clone() );
}
```

Partitioning the booster (cont.)

Step 2: Partition the booster model.

```
// Insert equally spaced markers throughout
// the booster model.

double momentum = PH_CNV_brho_to_p*( 296.5/10. );
double energy    = sqrt( PH_NORM_mp*PH_NORM_mp + momentum*momentum );

double boosterLength = 0.0;

bmlnElmnt* q;
DeepBeamlineIterator dbi( booster );
while(( q = dbi++ )) {
    boosterLength += q->Length();
}
dbi.reset();

marker spaceCharge( "Space Charge Marker" );

double markerInterval = boosterLength / ((double) number0fMarkers);
double insertionPoint = markerInterval;
InsertionList insl( momentum );
for( i = 0; i < number0fMarkers-1; i++ ) {
    insl.Append( new InsertionListElement( insertionPoint, &spaceCharge ) );
    insertionPoint += markerInterval;
}

double s_0 = 0.0;
slist removedElements;
booster.InsertElementsFromList( s_0, insl, removedElements );
booster.append( spaceCharge );

// Display the removed elements
cout << "Removed elements\n";
slist_iterator fembril( removedElements );
bmlnElmnt* qq;
while(( qq = (bmlnElmnt*) fembril() )) {
    cout << qq->Type() << " " << qq->Name() << "\n";
}
cout << endl;
```

Partitioning the booster (cont..)

Step 3. Construct the matrices.

```
// Prepare a Jet environment for use.  
Jet::BeginEnvironment( 1 );  
coord u(0.0), v(0.0), w(0.0),  
      U(0.0), V(0.0), W(0.0);  
Jet::EndEnvironment();  
  
// Array of matrices to store the maps.  
// If higher order maps were needed, this would  
// be an array of instances of Mapping.  
MatrixD linearMap[numberOfMarkers];  
  
// Store particle indices, for later convenience.  
int x = Particle::_x();  
int y = Particle::_y();  
int xp = Particle::_xp();  
int yp = Particle::_yp();  
  
// Partition the booster.  
dbi.reset();  
for( i = 0; i < numberOfMarkers; i++ ) {  
    JetProton jp( energy );  
    q = dbi++;  
    while(( q != &spaceCharge )) {  
        q->propagate( jp );  
        if( 0 == (q = dbi++) ) {  
            cerr << "Whoops!" << endl;  
            exit(1);  
        }  
    }  
    linearMap[i] = jp.State().Jacobian();
```

Partitioning the booster (cont...)

```
// ... a little output along the way
cout << "\n\n --- i = " << i << endl;
cout << linearMap[i];

cout << "      Horizontal:           Vertical:" << "\n"
<< setprecision(8) << setw(12) << linearMap[i](x,x)
<< setprecision(8) << setw(12) << linearMap[i](x,xp)
<< " "
<< setprecision(8) << setw(12) << linearMap[i](y,y)
<< setprecision(8) << setw(12) << linearMap[i](y,yp)
<< "\n"
<< setprecision(8) << setw(12) << linearMap[i](xp,x)
<< setprecision(8) << setw(12) << linearMap[i](xp,xp)
<< " "
<< setprecision(8) << setw(12) << linearMap[i](yp,y)
<< setprecision(8) << setw(12) << linearMap[i](yp,yp)
<< endl;
}
dbi.reset();

return 0;
}
```

Partitioning the booster: output

```
Command line: boosterMapDemo.3 5

Removed elements
drift 0
drift 000
drift 000
drift 0

--- i = 0
( -0.53123, 0, 0, 16.463, 0, 4.06278, )
( 0, -0.755966, 0, 0, 7.1688, 0, )
( 0.105421, 0, 1, 4.42333, 0, 1.72284, )
( -0.131631, 0, 0, 2.19687, 0, 0.809348, )
( 0, 0.0277093, 0, 0, -1.58558, 0, )
( 0, 0, 0, 0, 0, 1, )

    Horizontal:           Vertical:
-0.53123027   16.462975   -0.75596628   7.1688024
-0.13163122   2.1968656    0.02770926  -1.5855763

--- i = 1
( -2.1318779, 0, 0, 9.2733731, 0, 3.2573578, )
( 0, 0.89966809, 0, 0, 10.421103, 0, )
( -0.3618033, 0, 1, 3.1102116, 0, 1.7837183, )
( -0.018605403, 0, 0, -0.38813909, 0, 0.19720117, )
( 0, -0.14388674, 0, 0, -0.55515868, 0, )
( 0, 0, 0, 0, 0, 1, )

    Horizontal:           Vertical:
-2.1318779   9.2733731    0.89966809   10.421103
-0.018605403 -0.38813909  -0.14388674  -0.55515868
```

Partitioning the booster: output (cont.)

```
--- i = 2
( -0.64014983, 0, 0, 6.3214936, 0, 2.9900934, )
( 0, -0.83665909, 0, 0, 12.764491, 0, )
( 0.17115629, 0, 1, 3.0067052, 0, 1.9543001, )
( -0.093365309, 0, 0, -0.64014983, 0, 0.17021067, )
( 0, -0.023502823, 0, 0, -0.83665909, 0, )
( 0, 0, 0, 0, 1, )
```

Horizontal:	Vertical:
-0.64014983	6.3214936
-0.093365309	-0.64014983
-0.023502823	-0.83665909

```
--- i = 3
( -0.38813909, 0, 0, 9.2733731, 0, 3.0930279, )
( 0, -0.55515868, 0, 0, 10.421103, 0, )
( 0.19829674, 0, 1, 3.2754544, 0, 1.7837183, )
( -0.018605403, 0, 0, -2.1318779, 0, -0.35980436, )
( 0, -0.14388674, 0, 0, 0.89966809, 0, )
( 0, 0, 0, 0, 1, )
```

Horizontal:	Vertical:
-0.38813909	9.2733731
-0.018605403	-2.1318779
-0.14388674	0.89966809

```
--- i = 4
( 2.1968656, 0, 0, 16.462975, 0, 4.3988866, )
( 0, -1.5855763, 0, 0, 7.1688024, 0, )
( 0.81384437, 0, 1, 4.0853542, 0, 1.7228428, )
( -0.13163122, 0, 0, -0.53123027, 0, 0.10483897, )
( 0, 0.02770926, 0, 0, -0.75596628, 0, )
( 0, 0, 0, 0, 0, 1, )
```

Horizontal:	Vertical:
2.1968656	16.462975
-0.13163122	-0.53123027
0.02770926	0.02770926
-0.75596628	-0.75596628

Partitioning the booster: demo 4 (Mappings).

Step 1: Construct a model of the booster.

```
#include <iostream.h>
#include <iomanip.h>

#include "beamline.h"

int main( int argc, char* argv[] )
{
    // Process command line
    if( argc != 3 ) {
        cout << "Usage: " << argv[0]
            << " <number of markers (int)> <order (int)>"
            << endl;
        exit(0);
    }
    else {
        cout << "Command line: ";
        for( int i = 0; i < argc; i++ ) {
            cout << argv[i] << " ";
        }
        cout << "\n" << endl;
    }

    int number0fMarkers = atoi( argv[1] );
    int order          = atoi( argv[2] );

    // Create the booster model
    // ... Variables used by .cfg file in constructing the booster cell.
    const int ipt = 4;
    double npt   = ipt;
    double xpt   = 1.0/npt;

    #include "boosterv6.cfg"

    // beamline cell is created within boosterv6.cfg.
    // Construct a booster model from 24 cells.
    int i;
    beamline booster;
    for( i = 0; i < 24; i++ ) {
        booster.append( cell.Clone() );
    }
```

Partitioning the booster: (cont.)

Step 2: Partition the booster model.

```
// Insert equally spaced markers throughout
// the booster model.
const double momentum = PH_CNV_brho_to_p*( 296.5/10. );
const double energy    = sqrt( PH_NORM_mp*PH_NORM_mp + momentum*momentum );

double boosterLength = 0.0;

bmlnElmnt* q;
DeepBeamlineIterator dbi( booster );
while(( q = dbi++ )) {
    boosterLength += q->Length();
}
dbi.reset();

marker spaceCharge( "Space Charge Marker" );

double markerInterval = boosterLength / ((double) number0fMarkers);
double insertionPoint = markerInterval;
InsertionList insl( momentum );
for( i = 0; i < number0fMarkers-1; i++ ) {
    insl.Append( new InsertionListElement( insertionPoint, &spaceCharge ) );
    insertionPoint += markerInterval;
}

double s_0 = 0.0;
slist removedElements;
booster.InsertElementsFromList( s_0, insl, removedElements );
booster.append( spaceCharge );

// Display the removed elements
cout << "Removed elements\n";
slist_iterator fembril( removedElements );
bmlnElmnt* qq;
while(( qq = (bmlnElmnt*) fembril() )) {
    cout << qq->Type() << " " << qq->Name() << "\n";
}
cout << endl;
```

Partitioning the booster (cont..)

Step 3. Construct the maps.

```
// Prepare a Jet environment for use.  
Jet::BeginEnvironment( order );  
coord u(0.0), v(0.0), w(0.0),  
      U(0.0), V(0.0), W(0.0);  
Jet::EndEnvironment();  
  
// Array of Mappings to store the maps.  
Mapping higherMap[numberOfMarkers];  
  
// Construct the maps and store them.  
dbi.reset();  
for( i = 0; i < numberOfMarkers; i++ ) {  
    JetProton jp( energy );  
    q = dbi++;  
    while(( q != &spaceCharge )) {  
        q->propagate( jp );  
        if( 0 == (q = dbi++) ) {  
            cerr << "Whoops!" << endl;  
            exit(1);  
        }  
    }  
    higherMap[i] = jp.State();  
  
    // ... a little output along the way  
    cout << "\n\n --- i = " << i << endl;  
    higherMap[i].printCoeffs();  
}  
dbi.reset();  
  
return 0;  
}
```

Partitioning the booster: output

```
Command line: boosterMapDemo.4 5 4

Removed elements
drift 0
drift 000
drift 000
drift 0

--- i = 0

Begin JetVector::printCoeffs() .....
Dimension: 6, Weight = 4, Max accurate weight = 4

JetVector::printCoeffs(): Component 0

Count = 70, Weight = 4, Max accurate weight = 4
Reference point: 0, 0, 0, 0, 0, 0

Note: indices are ( x, y, c $\delta$ t, px/pref, py/pref,  $\delta$ p/pref ).

Index: ( 0, 0, 0, 0, 0, 1 ) Value: 4.06278296938
Index: ( 0, 0, 0, 1, 0, 0 ) Value: 16.4629748174
Index: ( 1, 0, 0, 0, 0, 0 ) Value: -0.531230267021
Index: ( 0, 0, 0, 0, 0, 2 ) Value: -14.7312777305
Index: ( 0, 0, 0, 0, 2, 0 ) Value: -0.350119587448
Index: ( 0, 0, 0, 1, 0, 1 ) Value: 204.936149996
Index: ( 0, 0, 0, 2, 0, 0 ) Value: -50.7652764878
Index: ( 0, 1, 0, 0, 1, 0 ) Value: -0.0467358127695
Index: ( 0, 2, 0, 0, 0, 0 ) Value: -0.0865038208791
Index: ( 1, 0, 0, 0, 0, 1 ) Value: 5.97984229811
Index: ( 1, 0, 0, 1, 0, 0 ) Value: -0.679071297676
Index: ( 2, 0, 0, 0, 0, 0 ) Value: 0.0030587082327
Index: ( 0, 0, 0, 0, 0, 3 ) Value: -125.095845785
Index: ( 0, 0, 0, 0, 2, 1 ) Value: -9.51253345232
Index: ( 0, 0, 0, 1, 0, 2 ) Value: -2005.28651108
Index: ( 0, 0, 0, 1, 2, 0 ) Value: -50.3279563533
Index: ( 0, 0, 0, 2, 0, 1 ) Value: 1156.51095645
Index: ( 0, 0, 0, 3, 0, 0 ) Value: -1101.34362221
Index: ( 0, 1, 0, 0, 1, 1 ) Value: -1.38244545263
Index: ( 0, 1, 0, 1, 1, 0 ) Value: 10.5934434098
Index: ( 0, 2, 0, 0, 0, 1 ) Value: 1.34766445401
Index: ( 0, 2, 0, 1, 0, 0 ) Value: -4.02162961877
Index: ( 1, 0, 0, 0, 0, 2 ) Value: 35.8558516817
Index: ... etc., etc., etc.
```

Partitioning the booster: demo 5 (Sectors).

Step 1: Construct a model of the booster.

```
#include <iostream.h>
#include <iomanip.h>

#include "beamline.h"

int main( int argc, char* argv[] )
{
    // Process command line
    if( argc != 3 ) {
        cout << "Usage: " << argv[0]
            << " <number of markers (int)> <order (int)>"
            << endl;
        exit(0);
    }
    else {
        cout << "Command line: ";
        for( int i = 0; i < argc; i++ ) {
            cout << argv[i] << " ";
        }
        cout << "\n" << endl;
    }

    int number0fSectors = atoi( argv[1] );
    int order          = atoi( argv[2] );

    // Create the booster model
    // ... Variables used by .cfg file in constructing the booster cell.
    const int ipt = 4;
    double npt   = ipt;
    double xpt   = 1.0/npt;

    #include "boosterv6.cfg"

    // beamline cell is created within boosterv6.cfg.
    // Construct a booster model from 24 cells.
    int i;
    beamline booster;
    for( i = 0; i < 24; i++ ) {
        booster.append( cell.Clone() );
    }
```

Partitioning the booster: (cont.)

Step 2: Partition the booster model.

```
// Insert equally spaced markers throughout the booster model.
const double momentum = PH_CNV_brho_to_p*( 296.5/10. );
const double energy   = sqrt( PH_NORM_mp*PH_NORM_mp + momentum*momentum );

double boosterLength = 0.0;

bmlnElmnt* q;
DeepBeamlineIterator dbi( booster );
while(( q = dbi++ )) {
    boosterLength += q->Length();
}
dbi.reset();

marker* spaceCharge[ numberOfSectors + 1 ];
double markerInterval = boosterLength / ((double) numberOfSectors);
double insertionPoint = markerInterval;
InsertionList insl( momentum );

spaceCharge[ 0 ] = new marker;
for( i = 1; i < numberOfSectors; i++ ) {
    spaceCharge[i] = new marker;
    insl.Append( new InsertionListElement( insertionPoint, spaceCharge[i] ) );
    insertionPoint += markerInterval;
}
spaceCharge[ numberOfSectors ] = new marker;

double s = 0.0;
slist removedElements;

booster.insert( spaceCharge[0] );
booster.InsertElementsFromList( s, insl, removedElements );
booster.append( spaceCharge[numberOfSectors] );

// Display the removed elements
cout << "Removed elements\n";
slist_iterator fembril( removedElements );
bmlnElmnt* qq;
while(( qq = (bmlnElmnt*) fembril() )) {
    cout << qq->Type() << " " << qq->Name() << "\n";
}
cout << endl;
```

Partitioning the booster (cont..)

Step 3: Construct the sectorized beamline.

```
// Prepare a Jet environment for use.  
Jet::BeginEnvironment( order );  
coord u(0.0), v(0.0), w(0.0),  
    U(0.0), V(0.0), W(0.0);  
Jet::EndEnvironment();  
  
// Sectorize between the partition markers.  
beamline splitBooster( "Sectorized booster" );  
for( i = 0; i < number0fSectors; i++ ) {  
    JetProton jp( energy );  
    splitBooster.append( booster.MakeSector( *(spaceCharge[i]),  
                                            *(spaceCharge[i+1]),  
                                            order, jp ) );  
    splitBooster.append( spaceCharge[i+1] );  
}  
  
return 0;  
}
```

Partitioning the booster: demo 6 (Read MAD file).

Step 1: Construct a model of the booster.

```
#include <iostream.h>
#include <iomanip.h>
#include "PhysicsConstants.h"
#include "beamline.h"
#include "bmlfactory.h"

// Nonsense needed for bmlfactory.
madparser* mp = 0;

main( int argc, char* argv[] )
{
    // Process command line
    if( argc != 5 ) {
        cout << "Usage: " << argv[0]
            << " <MAD file> <acc name> <kinetic energy [GeV/c]> <number of markers>" 
            << endl;
        exit(0);
    }
    else {
        cout << "Command line: ";
        for( int i = 0; i < argc; i++ ) {
            cout << argv[i] << " ";
        }
        cout << "\n" << endl;
    }

    char* madFile      = argv[1];
    char* accName      = argv[2];
    double kineticEnergy = atof( argv[3] );
    int numberOfWorkers = atoi( argv[4] );
    int i;

    double energy     = kineticEnergy + PH_NORM_mp;
    double momentum = sqrt( energy*energy - PH_NORM_mp*PH_NORM_mp );
    cout << "momentum = " << momentum << " [GeV/c]" << endl;

    // Create the booster model
    double brho = ( fabs( momentum ) )/PH_CNV_brho_to_p;
    bmlfactory bf( madFile, brho );
    beamline* boosterPtr = bf.create_beamline( accName );
```

Partitioning the booster: (cont.)

Step 2: Partition the booster model.

```
// Insert equally spaced markers throughout
// the booster model.
double boosterLength = 0.0;

bmlnElmnt* q;
DeepBeamlineIterator dbi( boosterPtr );
while(( q = dbi++ )) {
    boosterLength += q->Length();
}
dbi.reset();

marker spaceCharge( "Space Charge Marker" );

double markerInterval = boosterLength / ((double) numberOfMarkers);
double insertionPoint = markerInterval;
InsertionList insl( momentum );
for( i = 0; i < numberOfMarkers-1; i++ ) {
    insl.Append( new InsertionListElement( insertionPoint, &spaceCharge ) );
    insertionPoint += markerInterval;
}

double s_0 = 0.0;
slist removedElements;
boosterPtr->InsertElementsFromList( s_0, insl, removedElements );
boosterPtr->append( spaceCharge );

// Display the removed elements
cout << "Removed elements\n";
slist_iterator fembril( removedElements );
bmlnElmnt* qq;
while(( qq = (bmlnElmnt*) fembril() )) {
    cout << qq->Type() << " " << qq->Name() << "\n";
}
cout << endl;
```

Partitioning the booster (cont..)

Step 3. Construct the maps.

```
// Prepare a Jet environment for use.  
Jet::BeginEnvironment( 1 );  
coord u(0.0), v(0.0), w(0.0),  
      U(0.0), V(0.0), W(0.0);  
Jet__environment* pje = Jet::EndEnvironment();  
  
// Array of matrices to store the maps.  
// If higher order maps were needed, this would  
// be an array of instances of Mapping.  
MatrixD linearMap[numberOfMarkers];  
  
// Store particle indices, for later use.  
int x = Particle::_x();  
int y = Particle::_y();  
int xp = Particle::_xp();  
int yp = Particle::_yp();  
  
// Partition the booster.  
dbi.reset();  
for( i = 0; i < numberOfMarkers; i++ ) {  
    JetProton jp( energy );  
    q = dbi++;  
    while(( q != &spaceCharge )) {  
        q->propagate( jp );  
        if( 0 == (q = dbi++) ) {  
            cerr << "Whoops!" << endl;  
            exit(1);  
        }  
    }  
    linearMap[i] = jp.State().Jacobian();
```

Partitioning the booster (cont...)

```
// ... a little output along the way
cout << "\n\n --- i = " << i << endl;
cout << linearMap[i];

cout << "      Horizontal:           Vertical:" << "\n"
<< setprecision(8) << setw(12) << linearMap[i](x,x)
<< setprecision(8) << setw(12) << linearMap[i](x,xp)
<< " "
<< setprecision(8) << setw(12) << linearMap[i](y,y)
<< setprecision(8) << setw(12) << linearMap[i](y,yp)
<< "\n"
<< setprecision(8) << setw(12) << linearMap[i](xp,x)
<< setprecision(8) << setw(12) << linearMap[i](xp,xp)
<< " "
<< setprecision(8) << setw(12) << linearMap[i](yp,y)
<< setprecision(8) << setw(12) << linearMap[i](yp,yp)
<< endl;
}
dbi.reset();

return 0;
}
```