LATTICE DESIGN CODES: LECTURE FOUR: LOOKING AT CODES

Hywel Owen, University of Manchester/Cockcroft Institute

The Cockcroft Institute
of Accelerator Science and Technology

Remember our matrix element?

$$\vec{x} = \begin{pmatrix} x \\ p_x \\ y \\ p_y \\ z \\ \delta \end{pmatrix} \qquad R = \begin{pmatrix} 1 & L & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{L}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Remember our matrix element R_{56} ?

This is a **ballistic** term, i.e. it advances or retards particles based on their velocity – faster particles are advanced, whilst slower ones are retarded.

Sometimes this component is present in a code, sometimes it is not. This is an example of a 'gotcha'.

Exercise: for a 1 GeV particle beam with 1% energy spread, and initial zero bunch length, what is the bunch lengthening per metre of drift space?

A bestiary of codes: 1

There are a large variety of codes available. Let's first look at the main places you can find out about them.

The CARE-HHH Code Repository:
http://care-hhh.web.cern.ch/care-hhh/simulation_codes_catalogue_and_repository.htm
Kind of the same as the ASTeC portal, but geared towards linear colliders.

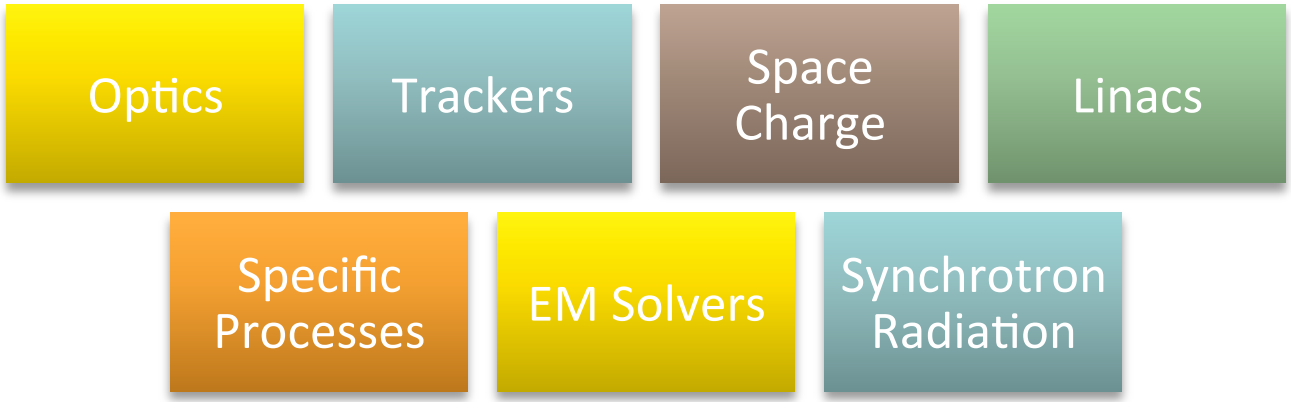
Los Alamos Code Group: http://laacg1.lanl.gov/laacg/software/software.phtml
Previously was the main page people went to for information about codes.

Andrei Semenov's Physics Code page: http://www.jlab.org/~semenov/rlinks/soft.html
A great web page giving links to LOTs of general scientific software – compilers, languages, tools, control systems etc.

John Jowett's pages: http://jowett.home.cern.ch/jowett/computing.html
A good example of an integrated suite of tools.

A bestiary of codes: 2

One way to characterise codes is as below. There are several available codes in each category of course!



Let's look at optics codes in more detail

An optics code does some or more of the following:

Describes the motion of a particle in the 6 dimensional phase space under the influence of external fields

Linear motion

- Definition of system
- Definition of geometry
- Fitting of linear optics functions etc.

Nonlinear motion

- Nonlinear perturbations
- Dynamic aperture

Other perturbations

- Error analysis
- Orbit analysis

Different points of view

The physicist who cares only about the methods/assumptions used.

The programmer who wants to implement the newest programming techniques.

The user (also a physicist/programmer) who doesn't care about methods and programming but likes a well documented, usable, cross-checked code to get the work done.

(from Winni Decking's nice talk on codes)

Beware of codes that are written **too** much as an exercise in programming rather than physics, or you may end up with 'abject-oriented programming', as Etienne Forest has called it.

General structure of a code

1. Get the data/lattice into the code - the lattice parser
2. Calculate
   - Linear optics functions
   - Tracking
   - Construct Map
3. Analyze the result
   - Display optics function
   - Calculate DA, frequency map, nonlinear distortions

These are split up in a variety of ways in different codes, and there are 3 basic philosophies that you will see:

1. Monolithic codes that try to do 'everything' – i.e. single binary, built-in parser and parser 'language', built-in commands for plotting. **Example: MAD-8.**
2. Suite of codes, with multiple binaries, and helper files for things like plotting, data analysis. The suite is often joined together using a scripting language. **Example: Elegant.**
3. A set of libraries that provide functions to a programmer, who then writes (and compiles) a program to do a specific simulation. **Example: MERLIN.**

None of these 3 approaches is 'right' or 'wrong'. I prefer approach 2 myself.

East Enders

Over the years there have been lots of optics codes written:

AT, BETA, BMAD, COMFORT, COSY-INFINITY, DIMAD, ELEGANT, LEGO, LIAR, LUCRETIA, MAD, MARYLIE, MERLIN, ORBIT, PETROS, PLACET, PTC, RACETRACK, SAD, SIXTRACK, SYNCH, TEAPOT, TRACY, TRANSPORT, TURTLE, UAL

Just like in a soap opera, there are relationships between the characters that you may not be aware of. There is actually a family tree of codes, with a personality and a reason behind the writing of each.

Example: To sort out the mess of the MAD-8 code, the CLASSIC 'universal' library was proposed. It never really succeeded as a collaboration, but several codes came out of it (MERLIN, ELEGANT etc.)

Moral: Many people have proposed 'universal libraries' that 'do everything'. None of them have become universal.

See e.g. http://www.slac.stanford.edu/xorg/icap98/focused.htm

Single particle ray-tracing

Particle vector

$$\vec{X} = (x, x', y, y', z, \delta) \ , \ \delta = \frac{\Delta P}{P_0}$$

Transport through elements

using **R** matrix

$$\vec{X}_f = \mathbf{R}\vec{X}_i$$

Linear optics calculations

Concatenated by Matrix multiplication

Extended to higher order (e.g. in TRANSPORT)

$$x_{j,f} = \sum_k \mathrm{R}_{jk} x_{j,i} + \sum_{kl} \mathrm{T}_{jkl} x_{j,i} x_{l,i} + \sum_{klm} \mathrm{U}_{jklm} x_{j,i} x_{l,i} x_{m,i} + \cdots$$

NOT symplectic (see next slide)

This approach used for high accuracy and model/reality calibration in systems with small numbers of elements, e.g. for spectrometers.

AT, BETA, BMAD, COMFORT, COSY-INFINITY, DIMAD, ELEGANT, LEGO, LIAR, LUCRETIA, MAD, MARYLIE, MERLIN, ORBIT, PETROS, PLACET, PTC, RACETRACK, SAD, SIXTRACK, SYNCH, TEAPOT, TRACY, TRANSPORT, TURTLE, UAL

(from Winni Decking's nice talk on codes)

Symplecticity

Given a transfer Map $$\vec{z}^{\,f} = \mathbf{M}\vec{z}^{\,i}$$

And its Jacobian $$J_{ab}(z^i) = \frac{\partial z_a^f}{\partial z_b^i}$$

J is symplectic if $$\mathbf{J}^T\mathbf{S}\mathbf{J} = \mathbf{S}$$

- A Hamiltonian system is symplectic.
- Therefore a one-turn mapping which fulfills the symplectic conditions describes a Hamiltonian system.
- The one-turn map does not exactly match the real system, but does retain the symplectic condition.
- If violated, artificial damping/excitation of motion.
- These sorts of maps are important when looking a highly-periodic systems with slow damping, e.g. tracking of circular accelerators over many turns (particularly protons).
- This is a big area.

Kick codes for particle tracking

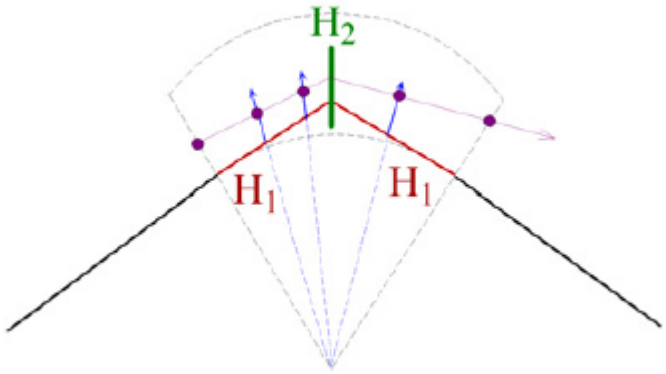
Elements described by thin lens kicks and drifts

Always symplectic

Long elements to be sliced -> slow

There are more sophisticated mathematical formalisms – see E.Forest for more details

AT, BETA, BMAD, COMFORT, COSY-INFINITY, DIMAD, ELEGANT, LEGO, LIAR, LUCRETIA, MAD, MARYLIE, MERLIN, ORBIT, PETROS, PLACET, PTC, RACETRACK, SAD, SIXTRACK, SYNCH, TEAPOT, TRACY, TRANSPORT, TURTLE, UAL



A drift-kick-drift in a bending magnet. Several kicks per element improve on the level of approximation with the real accelerator.

see e.g. the humorous review by E. Forest, “Geometric Acceleration for Particle Accelerators”, J. Phys. A: Math. Gen. **39 (2006) 5321–5377**

Accelerator libraries

1. Monolithic codes that try to do 'everything' – i.e. single binary, built-in parser and parser 'language', built-in commands for plotting. **Example: MAD-8.**
2. Suite of codes, with multiple binaries, and helper files for things like plotting, data analysis. The suite is often joined together using a scripting language. **Example: Elegant.**
3. A set of libraries that provide functions to a programmer, who then writes (and compiles) a program to do a specific simulation. **Example: MERLIN.**

Provide the user with a toolbox (library) rather than an existing program which does contain the needed elements and procedures

Realized in C++, F90, Pascal

AT, BETA, BMAD, COMFORT, COSY-INFINITY, DIMAD, ELEGANT, LEGO, LIAR, LUCRETIA, MAD, MARYLIE, MERLIN, ORBIT, PETROS, PLACET, PTC, RACETRACK, SAD, SIXTRACK, SYNCH, TEAPOT, TRACY, TRANSPORT, TURTLE, UAL

Can be tailored to specific problem, should be easy to maintain and extend

Accelerator libraries

1. Monolithic codes that try to do 'everything' – i.e. single binary, built-in parser and parser 'language', built-in commands for plotting. **Example: MAD-8.**
2. Suite of codes, with multiple binaries, and helper files for things like plotting, data analysis. The suite is often joined together using a scripting language. **Example: Elegant.**
3. A set of libraries that provide functions to a programmer, who then writes (and compiles) a program to do a specific simulation. **Example: MERLIN.**

Some codes are written directly into mathematical software such as Matlab. A good example of is **AT** (**A**ccelerator **T**oolbox).

Advantages: Easy writing of code, and doing plotting. In the case of AT, there is a good link with control system software via EPICs, which allows sophisticated machine control and machine/simulation comparisons.

Disadvantage: Because Matlab is an interpreted (non-compiled) language, it is quite slow.

Integrated codes

1. Monolithic codes that try to do ‘everything’ – i.e. single binary, built-in parser and parser ‘language’, built-in commands for plotting. **Example: MAD-8.**
2. Suite of codes, with multiple binaries, and helper files for things like plotting, data analysis. The suite is often joined together using a scripting language. **Example: Elegant.**
3. A set of libraries that provide functions to a programmer, who then writes (and compiles) a program to do a specific simulation. **Example: MERLIN.**

A more or less advanced process control is implemented in the code itself and allows complicated run logic

AT, BETA, BMAD, COMFORT, COSY-INFINITY, DIMAD, ELEGANT, LEGO, LIAR, LUCRETIA, MAD, MARYLIE, MERLIN, ORBIT, PETROS, PLACET, PTC, RACETRACK, SAD, SIXTRACK, SYNCH, TEAPOT, TRACY, TRANSPORT, TURTLE, UAL

Input formats and XSIF

A common format for input and output does not exist

XSIF, which really means 'XSIF-like' format, is used by quite a few codes

Much more information on a beam line element needed

- Errors, Aperture, Wakefields, ….
- Magnet errors
  - Systematic
  - Random
- Time dependent variation of magnet strength, magnet positions
- Correlation between errors important
  - Girder motion
  - Correlated systematic errors

Quite a few XML-based formats have been proposed/formulated, but they haven't caught on yet. SDDS (as used by Elegant and ASTRA) is a simpler alternative for data files that is a reasonable balance between flexibility and simplicity.

A typical XSIF file as used in Elegant

Elegant .lte file (XSIF)

```
……
!CELLH-DI-01:  CSBEND, L = 0.700, ANGLE = 0.23562, E1 = 0.11781, E2 = 0.11781

CELLH-Q-01:   QUAD, L = 0.25, K1 = -2.7017
CELLH-Q-02:   QUAD, L = 0.25, K1 =  5.0614

CELLH-DR-01: DRIF, L = 0.3
CELLH-DR-02: DRIF, L = 1.4
CELLH-DR-03: DRIF, L = 0.3
……
str:   LINE = (STR-DR-01,STR-Q-01, STR-DR-02,STR-Q-02, STR-DR-03,&
          STR-DI-01,STR-DR-04,STR-DI-02,STR-DR-05,STR-DI-03,&
          STR-DR-06,STR-DI-04,STR-M-01)
……
!Half an arc cell
cellh: LINE = (CELLH-M-01,CELLH-DI-01,CELLH-DR-01,CELLH-Q-01,CELLH-DR-02,&
CELLH-Q-02,CELLH-DR-03)
```

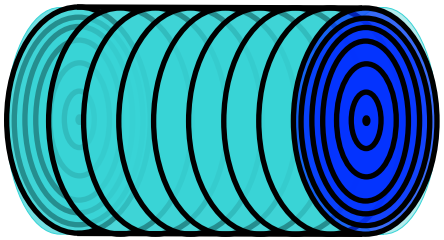
Elegant .ele file (command file)

```
&run_setup
   lattice      = hacl-021007.lte
   default_order = 1
   use_beamline  = str
   p_central_mev = 550
   magnets      = str.mag
   element_divisions = 5
&end

&twiss_output
                matched    = 0
                statistics = 1
                !From linac
   beta_x =  20.7, beta_y =  50.0, alpha_x =  0.53, alpha_y = -2.14
   filename        = str.twi
&end

&matrix_output
   full_matrix_only = 1
   SDDS_output     = str.matr
&end
```

These two pieces of information are combined into one file in MAD

Curvilinear vs. Cartesian codes

There is (very) broadly speaking a division between codes that assume a **geometry**, and those that simply integrate through a **Cartesian** coordinate system.



'Curvilinear' codes: (e.g. MAD, PTC, Elegant)

These are biased towards the following tasks:

- Optics calculations
- Particle tracking, particularly symplectic tracking
- Formal analysis of errors, e.g. orbit shifts from magnet misalignments

Important sub-group of 'curvilinear' codes are for space charge calculations:

- HOMDYN, ASTRA, PARMELA (the modules that assume cylindrical symmetry) – see e.g. C.Limborg et al., Code comparison for simulations of photo-injectors', *proceedings of PAC 2003*.

'Cartesian' codes: (e.g. GPT, GEANT/BDSIM, ray-tracing in OPERA)

These are biased towards the following tasks:

- Tracking of particle motion in exact fields
- Consideration of point-to-point effects
- Shower generation and tracking (e.g. FLUKA, GEANT-4)

Some 'gotchas'

We've already seen the ballistic term which is included in some codes and not in others.

Here is a list of some things in your code that might cause a problem.

Single particle problems

- Relativistic approximation – not correct if particle velocity is not c.
- Changing from electrons to positrons may not be handled correctly.
- Changing from electrons to hadrons may not be handled correctly.
- Small angle approximation may be being used, and not valid for your simulation.
- Edge focusing – presence, polarity.
- Energy – can your code vary the mean energy of the particle?

Multi-particle problems

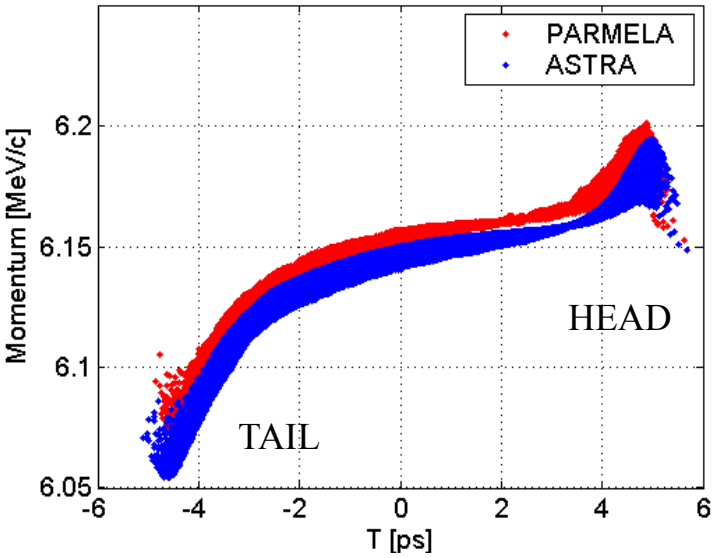
- Beam emittance ratio not valid for assumption of cylindrical symmetry (e.g. in ASTRA).
- Particle number/distribution incorrect for valid calculation of space charge, microbunching calculations.
- Binning – can cause all sorts of confusion. For example, comparison of peak current calculations in different simulations. Binning is especially important in numerical wakefield calculations.
- Emittance calculations – is it sigma, 90%, 95%? How is it being calculated? How is the FWHM value being derived? (is it just 2.35 x the Gaussian value?).
- Distribution shape and edges – have you considered whether your model distribution introduces unphysical artefacts. A good example of this is wakes induced by sharp edges in the generated bunch distribution; they aren't there in reality!

Polarity/reference problems

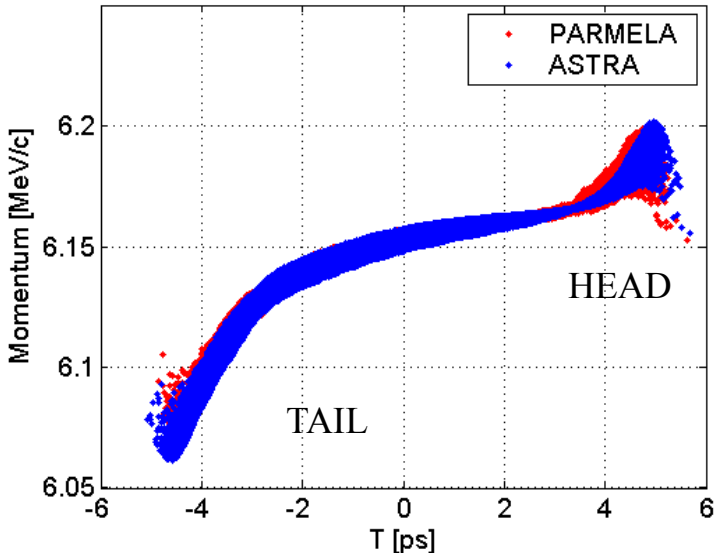
- Conversion from *t* to *v* for non-relativistic particles.
- Does *+t* mean particle is ahead or behind the reference particle? This caused confusion for over a year on the TTF project. The fix was to look at the actual distributions rather than the RMS values.
- How are things like phase and amplitude being translated from the input file to the actual calculation? For example, ASTRA/PARMELA/Elegant comparison use different ideas for the reference particles.

Example: ASTRA vs. PARMELA in the LCLS injector

ASTRA GUN FIELD = 120 MV/m



ASTRA GUN FIELD = 120.15 MV/m

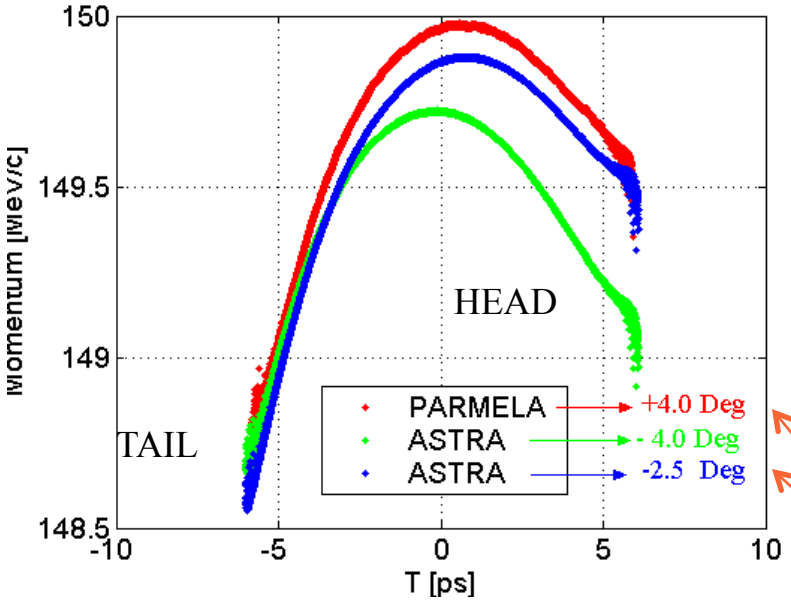


The Cockcroft Institute
of Accelerator Science and Technology

Example: ASTRA vs. PARMELA in the LCLS injector

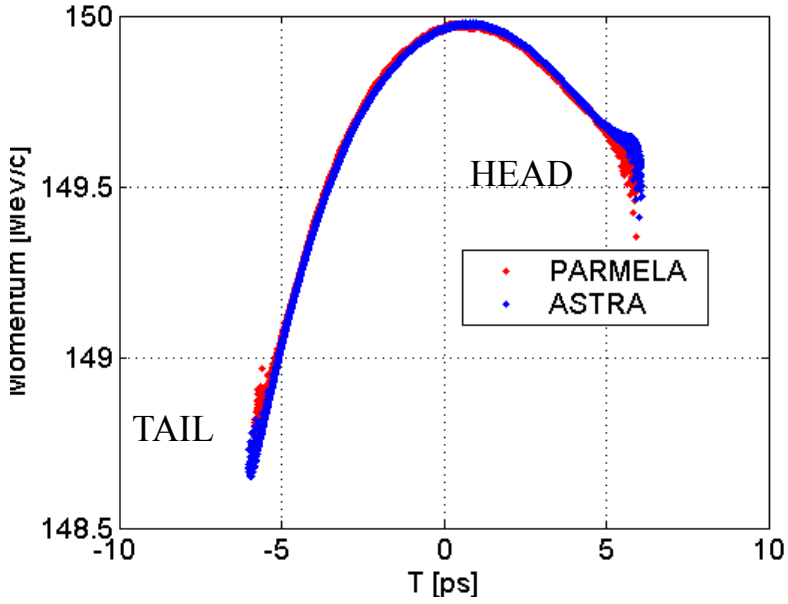
Long. Phase Space for 3 phases of L01

MAX FIELD L02 = 40.66 MV/m



Phase ASTRA L01 at -2.5 Deg

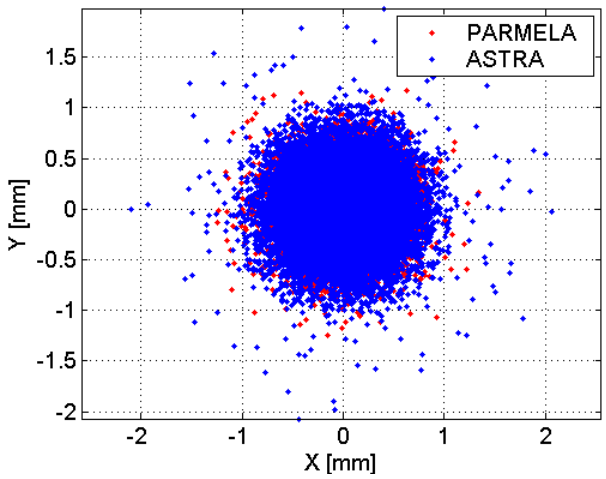
MAX FIELD L02 = 40.68 MV/m

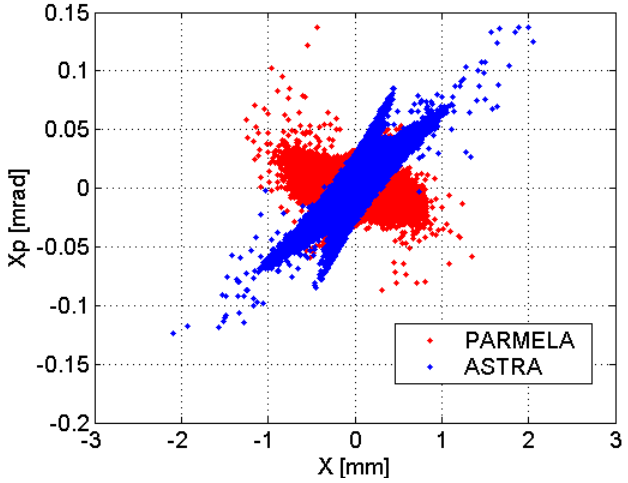


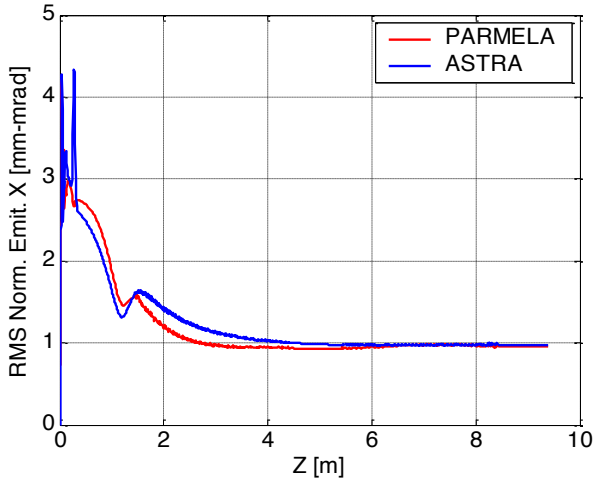
Note the reversal of the polarity needed here

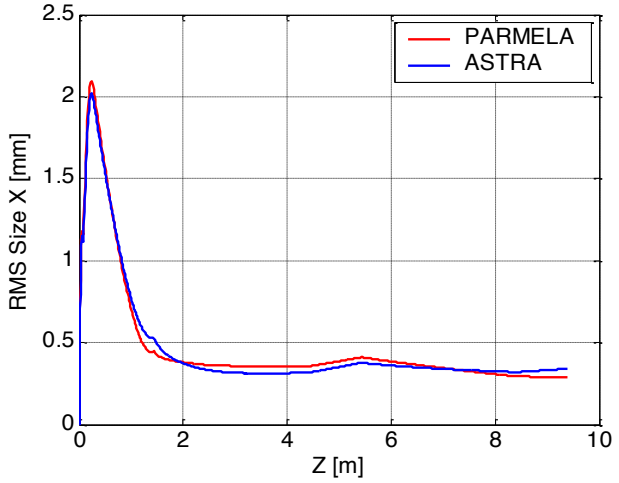
The Cockcroft Institute of Accelerator Science and Technology

Example: ASTRA vs. PARMELA in the LCLS injector









Agreement of transverse focusing is rather poor. Just looking at the Twiss functions derived from these data will be misleading.

The Cockcroft Institute
of Accelerator Science and Technology

Summary

There are many codes available for different purposes, and with different philosophies. Most were written with a particular job or range of validity in mind, and most of those have been used outside their range of validity. Even the authors won't know all the bugs.

It is therefore worth considering comparisons and benchmarking of your simulations, to keep a reality check.

Remember that most 'integrated' lattice codes, e.g Elegant, only handle certain phenomena in a simplistic way, such as:

- RF focusing (this is often different in different codes)
- Wakefields
- Coherent radiation and associated aperture effects
- Magnet fields, including edge focusing
- Intrabeam effects such as scattering

and may completely ignore other important processes, such as:

- Space charge
- Beam loss and showers
- RF effects such as beam loading

Dedicated 'process' codes must be used to check these simplistic calculations for sensitive regions of an accelerator.