TRANSOPTR Reference Manual

TRIUMF Beam Physics

December 18, 2024

Contents

Abstract

This is meant to be both an instruction manual on how to use `TRANSOPTR`, and a reference manual.

1 Quick start (i.e. you don't want to read the manual)

The manner in which TRANSOPTR works is perhaps not common. There are at least two files needed to run the code: a code file and a data file. In some cases there are also files to calculate either ambient or element fields.

The description of the transport channel is coded in FORTRAN77[1]. We name this file `sy.f`. Its first line is `SUBROUTINE tSYSTEM` and the remainder is chiefly a series of calls to subroutines that each transport through a beam optics element: drifts, dipoles, quadrupoles, solenoids, etc. A list of callable elements can be found in Section 5. Fitting is coded as calls to `FIT`; these calls simply add an error to the deviation vector whose weighted norm is to be minimized. Any parameter can be fitted, under any imaginable constraints.

The data file (`data.dat`) gives the beam characteristics and the parameters to be fit, and their allowable value ranges. These parameters are the users choice, they could be lengths, strengths, angles, etc.

1.1 Single dipole

Here is an example that transports through a single 90° bend with radius of 37.46 cm, gap of 6.35 cm and edge angles of 22.4°. To keep this first example simple, no parameters are passed from the data file.

```
SUBROUTINE tSYSTEM
angle=90.
rho=37.46
entr=22.4
gap=6.35
wq=1.
qK1=0.257
qK2=0.0
call edge(entr,rho ,angle,0.0,qK1,qK2,gap,0.0,wq)
call bend(rho ,angle,0.0,'dipole') !field index=0
call edge(entr,rho ,angle,0.0,qK1,qK2,gap,0.0,wq)
call print_transfer_matrix
RETURN
END
```

The traditional TRANSPORT dipole fringe field integral above is called `qK1`. (Beware: it's not named K1 as, without an `IMPLICIT` statement, variables whose first character is `I` through `N` are assumed to be integers. This is a FORTRAN convention.)

To run it, one must first compile the transport code and link it to the rest of TRANSOPTR code, which is mainly a set of transport element and fitting subroutines. The extra ingredient is the `data.dat` file as follows:

[1]See Appendix A.3 if you know nothing about this computer language.

```
 0.06 0 0 27930. 1 0.0 ! Energy[MeV]; mom.; brho; mass[MeV]; charge; beam current or bunch charge
 1 3 1. 0.5E-5 ! iprint (see below), IVOPT, initial RK step, error tolerance (see below)
0 0.0 1.0 0.0 ! for external ambient axial B field, see below.
.4 .0625  .4 .0625  0. .001 ! initial bunch dimensions: x,x',y,y',z(bunch half length),dp/p
 1.  1.  1.  1.  1.  1. 0. 1.! input units: 1 means x,y,z in cm, x',y',dp/p dimensionless (radians)
 0 ! number of normalized sigma matrix elements to be read, see below.
 0 ! number of parameters to be passed to sy.f.
 1.E-4 900 ! Convergence criterion for optimization; number of iterations allowed.
 01  0. 0.98 50 ! see below.
```

Running the code results in the output found here.

Here is the TRANSPORT file that makes the same calculation: singly charged particles of mass 30 amu and energy 60 keV through the same dipole. The calculated transfer matrix is here. GIOS[27] files (input, output) as well as COSY-∞[11] (input, output) are available for comparison. Note that K1=0.257 was chosen to match GIOS and COSY fringe field integrals.

So far, the TRANSOPTR code does nothing differently than any of these other three codes. Its advantages are clearer in a fitting example.

1.2 Single dipole fitted, and 4 to make a ring

So as a FIT example, let us take the previous example and ask to fit the magnet's edge angles and a drift length before and after, in order to create a 'point-to-parallel' condition both vertically and horizontally. This gives a symmetric cell a $\pi/2$ phase advance. Fits of desired beam condition or transfer matrix element are coded into sy.f by including the parameters to vary into a common block named BLOC1.

Further, we have expanded the calculation to do four bend cells in succession. This is done by simply putting the elements into a DO loop. Note the FIT statements are only called after the first cell.

The FIT statements contain the following parameters: first entry is 1 for a $\boldsymbol{\sigma}$-matrix fit, 2 for a transfer matrix fit. The second and third are the indices of the matrix element, fourth is the desired value, fifth is the weight, sixth is the exponent of the weight.

The data file for this case now contains the two parameters entr and d1. The lines containing the two parameters are structured as follows: $\langle Q_{\text{init}}, Q_{\text{low}}, Q_{\text{high}}, Q_{\text{vary}}\rangle$. Respectively, these are: initial value, lower limit, upper limit, and a flag which is zero if the parameter is fixed, 1 if it is to vary to achieve the fit.

1.3 Make the ring isochronous

As a further refinement to one-dipole-fitted, the field index of the 4 dipoles is fitted to make the ring isochronous. This is done with a CALL FIT(2,5,6,0.,1.,1) as the 56 transfer matrix element is the one-turn time increment for given energy increment.

The result is then used in a second calculation where the initial beam is adjusted according to the beta function found from the transfer matrix to make the beam exactly matched, and the second fit is to find the periodic dispersion. The `gnuplot` macro `mac` here has been set to output β functions, instead of the envelopes, to mimic synchrotron code output.

2 What it does / How it works

`TRANSOPTR` is very much a "nuts and bolts" code. It consists of a library of callable routines each representing an element type that can be applied to a beam of particles. The beam is set up from a data file of characteristics. A user-written "system" routine calls elements in turn, and there can be fit statements where the user specifies desired beam characteristics at that point. There is no constraint on the type of parameters that can be fit. The main routine handles optimization tasks and loops through the system routine as needed to converge to a fit. Standard fits are matching conditions of various kinds, but in principle the user can write their own specialized fits and elements or combination of elements.

`TRANSOPTR` is an envelope code, using only first order optics. Originally, it simply mimicked the existing code `TRANSPORT`, which included second order optics of dipoles. `TRANSOPTR` no longer does this, as there are far more complete codes that calculate to higher order. `COSY`-∞[11] in particular calculates to any order. Instead, aberration emittance growth effects of higher orders are calculated in `TRANSOPTR` on a per-element basis and so can be one of the minimization goals.

`TRANSOPTR` also has the capability to track a single trajectory, that of the reference particle. This is simultaneous to the envelope tracking so for example the user can plot the reference particle and "dress" it with the envelope, for visual effect. Only in cases of beam shifts in angle due to steerers, and misaligned focusing elements, will the reference particle have non-zero transverse coordinates. The time and energy coordinates are always non-zero and tracked; the have non-trivial behaviour in rf devices.

2.1 Envelopes

In simplest mode on execution, `TRANSOPTR` accumulates the system 6-by-6 transfer matrix $\mathbf{M}(s)$, and applies the initial beam matrix ($\boldsymbol{\sigma}_{\mathrm{i}}$) to it to find the beam at any point s ($\boldsymbol{\sigma}(s)$), according to

$$\boldsymbol{\sigma}(s) = \mathbf{M}\boldsymbol{\sigma}_{\mathrm{i}}\mathbf{M}^T \tag{1}$$

where needed, either for output plotting, or for comparing with the desired beam during fitting. See any article on this technique by Brown [12, 13, 14, 15, 16]. The difference is that `TRANSOPTR` is coded directly into `FORTRAN`, and this allowed far greater flexibility in fitting and constraints. A drawback of such codes is that only analytically integrable beam transport elements code be handled as they had known matrices. This limits element type and moreover requires them to be highly idealized as having piecewise constant, s-independent focal strengths.

These limitations were overcome by adding the possibility to integrate the equations of motion directly numerically. If we can define an *infinitesimal transfer matrix* $\mathbf{F}$ where $\mathbf{X'} = \mathbf{FX}$ and the transfer matrix of an infinitesimal length ds is $\mathbf{M} = \mathbf{I} + \mathbf{F}ds$, we find directly

$$\boldsymbol{\sigma}' = \mathbf{F}\boldsymbol{\sigma} + \boldsymbol{\sigma}\mathbf{F}^T. \tag{2}$$

This is the **envelope equation**. For the full 6D case, it represents 21 equations. (Because $\boldsymbol{\sigma}$ is symmetric.) The matrix $\mathbf{F}$ derives directly from the Hamiltonian and in principle this would not change the manner of calculation: transfer matrices would be found from numerical integration, rather than stored. However including space charge changes the manner of calculation since then $\mathbf{F}$ is recursively a function of $\boldsymbol{\sigma}$. This integration is thus a different mode of running where transfer matrix and beam matrix are calculated simultaneously together; 36+21 coupled ODEs.

2.2 Beam Center

We have added to the code the possibility to track also the first moments of the beam distribution: i.e. the deviation of the beam centroid w.r.t. the intended trajectory. A few routines such as `LINAC_DIPOLE` and `strayB` let you input field errors and track the effect on the trajectory of the beam centroid. To be more specific, the equations of motion of the beam first and second moments that `TRANSOPTR` integrates using standard fourth-order Runge-Kutta are:

$$\frac{\mathrm{d}\langle\mathbf{X}\rangle}{\mathrm{d}s} = \mathbf{F}_\perp + \mathbf{F}_\mathrm{e}\langle\mathbf{X}\rangle\,, \tag{3}$$

$$\frac{\mathrm{d}\boldsymbol{\sigma}}{\mathrm{d}s} = \mathbf{F}\boldsymbol{\sigma} + \boldsymbol{\sigma}\mathbf{F}^\mathrm{T}\,, \tag{4}$$

where

$$\mathbf{X} = (x, p_x, y, p_y, z, p_z)^\mathrm{T} \tag{5}$$

is the state vector of a single particle,

$$\langle\mathbf{X}\rangle = \frac{1}{N}\sum_{i=1}^{N}\mathbf{X} \tag{6}$$

is the vector of the beam's first moments (N is the total number of particles),

$$\boldsymbol{\sigma} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{X} - \langle\mathbf{X}\rangle)(\mathbf{X} - \langle\mathbf{X}\rangle)^\mathrm{T} \tag{7}$$

is the 6×6 covariance matrix of the beam, which contains the second moments of the beam distribution. $\mathbf{F} = \mathbf{F}_\mathrm{e} + \mathbf{F}_\mathrm{sc}$ is the infinitesimal transfer matrix [23] and $\mathbf{F}_\perp(s)$ is a vector of the on-axis force: the product between the symplectic matrix and the gradient of the Hamiltonian evaluated on axis $\mathbf{X} = \mathbf{0}$. So, in fact, `TRANSOPTR` tracks 36+21**+6** coupled ODEs.

2.3 Acceleration

Two additional ODEs are also integrated. These are time and energy of the reference particle. In point of fact, the coordinates tracked are ct and $\gamma - 1$. These are only non-trivial in case of acceleration. The time coordinate can be used to determine and optimize phases of RF devices relative to each other.

`TRANSOPTR`, when integrating, uses only canonical coordinates, e.g. p_x is tracked, not x'. Any elements that change P_0 (e.g. einzel lens, rf gap, linac) will use canonical momentum scaled by

$P_0|_{\mathrm{start}}$, the momentum at the start of the element internally up to the end of integration, and then multiplied by $\frac{P_0|_{\mathrm{start}}}{P_0|_{\mathrm{end}}}$ to convert back to scaling by the final $P_0|_{\mathrm{end}}$ at the end of the element. Solenoids and other axial magnetic fields will use the canonical transverse momenta that include the vector potential and so for example, the second and fourth coordinates output will differ from x' and y' at any s where the axial field is nonzero. Symplecticity is never violated; it in fact is used as a handy check on integration accuracy and drives the adaptive stepsize control.

2.4 Optimization

In either case (stored transfer matrices mode or integration mode), as the code executes, it accumulates a vector we name $\vec{\chi}$. Its components are the scaled differences of calculated characteristics and the desired ones. The desired characteristics may be beam-related (e.g. envelope size, waist condition, etc.) or transfer matrix (e.g. point-to-point condition, magnification, etc.), or even a combination of the two (e.g. mass resolution). Special cases, where the desired value is always zero, are matching calculations, and aberration estimations; in both those cases, the calculated parameter is fractional emittance growth $\Delta\epsilon/\epsilon$.

Differences between desired and found values are scaled by user-supplied weights, and at the end of the transport calculation the norm of $\vec{\chi}$ calculated. If there are free parameters in the parameter list, then the optimizer starts varying those parameters in attempt to minimize it. The technique is downhill simplex, augmented by simulated annealing algorithm for more difficult many-parameter cases. Optimization ceases when change in $|\vec{\chi}|$ for small chnages in parameter values is below some tolerance. Generally, this minimal value is not zero. In fact, even in simplest cases that include aberrations, exact match with zero aberration is not physically possible; `TRANSOPTR` finds the best compromise between mismatch and aberration.

2.5 Physical Units

The units used for electric and magnetic quantities is a bit of a "mish-mash". We apologize for this; it is a result of the code's intrinsic extensibility, namely, as the transport system is simply a subroutine, the user is free to define their own elements as attached subroutines, and these can eventually become part of the main code. Some of this is alleviated by defining elements by their geometry rather than their fields. E.g. bends are simply given by radius and angle, and the magnetic field is an output rather than an input parameter. Still, electric quads' strengths are in kV, while accelerator fields are in MV. Masses and energies in MeV. Magnetic fields generally in Tesla.

The units of the particle coordinates are at the discretion of the user, specified in `data.dat`, line 5 (3.2.5). But internally, the length units are cm, and momentum units are dimensionless, being normalized by the reference momentum, so that e.g. transverse momenta are angles in radians. Acceleration elements, where the reference momentum changes, though, do not use this definition as it is not canonical. Simply, through the element, the momentum is normalized by the momentum at injection, then renormalized by the final momentum at the exit.

3 In-Depth Example

The goal of this exercise is to model the beam extracted from the ISAC Offline Ion Source (OLIS), through the OLIS mass separator consisting of a dipole magnet and its entrance and exit slits, aiming to produce a transverse twiss parameter match criterion. The system file is shown below. There are 8 quadrupoles in this example, IOS:Q1 to Q8, however we're only going to be changing Q1 through Q6. The remaining two are a fixed periodic cell. Besides the 4 conditions needed for the match, the call to `SLIT` compares the beam envelope to the slit size and calculates a beam loss, in x and y. Lastly, each quadrupole has an independent calculation of emittance growth estimation due to third order aberration. There are 12 deviations in all and this vector is multiplied by a vector of weights and added in quadrature. This single parameter (`CHI`) is minimized in the six-dimensional space of the six tunable quadrupoles. The minimization is a Nelder-Mead algorigthm which samples parameter space to find the minimum without needing partial derivatives of `CHI` with respect to the six parameters.

3.1 `sy.f`

```
      SUBROUTINE tSYSTEM
      COMMON /BLOC1/q1,q2,q3,q4,q5,q6,q7,q8
c All the parameters passed from data.dat are in this COMMON.
      COMMON/SCPARM/QSC,ISC,CMPS !common block to fetch CMPS.
      cmps=2. !this is # of cm per plotting step. If omitted, plots will be "kinky".
      call DRIFT(                      3.8100,".")
c Drift length, name
      call DRIFT(                      8.8900,".")
      call DRIFT(                     19.0500,".")
      call EEdge(  25.4,  36.0,   1.0,   0.2122,   3.8100,   1.0)
c Electric bend edge; radius, angle, index, fringe field integral,
c                                                electrode gap, aberration weight.
      call EBEND(  25.4,  36.0,   1.0,"IOS:B1A")
c Electric bend; radius, angle, index, name.
c Index means ratio of bend to non-bend curvature radius
c                                             (1 means spherical, 0 is cylindrical).
      call EEdge(  25.4,  36.0,   1.0,   0.2122,   3.8100,   1.0)
      call DRIFT(                     21.0820,".")
      call DRIFT(                      2.7940,".")
      call deflectx(   5.5880,   9.0,0,0)
c Parallel plate deflector; length, angle, 0, 0 (deprecated entries)
      call DRIFT(                      2.7940,".")
      call DRIFT(                      8.1280,".")
      call fringeq(0.108,     0.004,    0.042,0.0)
c The four quadrupole fringe field integrals as defined by Wollnik.
c These apply to all downstream quads until reset by another call.
      call EQUAD(  Q1,2.5400,   5.2680,   1.0,"IOS:Q1")
c Electrostatic quadrupole; strength, aperture radius, length, aberr. weight, name.
```

```
      call DRIFT(                      4.9682,".")
      call EQUAD(  Q2,2.5400,  10.2464,   1.0,"IOS:Q2")
      call DRIFT(                      2.4282,".")
      call EQUAD(  Q3,2.5400,   5.2680,   1.0,"IOS:Q3")
      call DRIFT(                      4.8590,".")
      call SLIT(0.05,.5,1.,'MCOL3A')
c Slit will cut beam size; x 1/2 aperture, y 1/2 aperture, weight, name.
      call DRIFT(                     51.9615,".")
      call Edge(  0.0, 30.0, 60.0, 0.0,   0.3170,   2.0, 5.08,0.0,1.0)
c Magnetic bend edge; edge angle, bend radius, bend angle, 0,
c K1 fringe field integral, K2 fringe field integral, gap, index, weight for aberr.
      call BEND(     30.0, 60.0,  0.0,"IOS:MB")
c Magnetic bend; radius, angle, field index, name.
      call Edge(  0.0, 30.0, 60.0, 0.0,   0.3170,   2.0, 5.08,0.0,1.0)
      call DRIFT(                     51.9615,".")
      call vective(10)
c For sigma matrix printout.
      call SLIT(0.05,.5,1.,'MCOL3B')
      call DRIFT(                      4.8590,".")
      call EQUAD(  Q4,2.5400,   5.2680,   1.0,"IOS:Q4")
      call DRIFT(                      3.6982,".")
      call EQUAD(  Q5,2.5400,  10.2464,   1.0,"IOS:Q5")
      call DRIFT(                      3.6982,".")
      call EQUAD(  Q6,2.5400,   5.2680,   1.0,"IOS:Q6")
      call DRIFT(                     27.8714,"         IOS:FC6")
      call DRIFT(                     12.6175,".")
      call fringeq(0.087,0.005,0.033,0.0)
      call EQUAD(  Q7,2.5400,   6.1620,   1.0,"IOS:Q7")
      call DRIFT(                     61.8998,".")
      call EQUAD(  Q8,2.5400,   6.1620,   1.0,"IOS:Q8")
      call DRIFT(                     12.6175,"        IOS:RPM8")
      call twissmatch( 1,   2.2040,  84.8,   1.0, 1)
c Twiss match fit; 1 for x and 3 for y, then alpha, beta, weight, exponent.
      call twissmatch( 3,  -2.2040,  84.8,   1.0, 1)
      RETURN
      END
```

Further elaboration of the elements in `sy.f` can be found in [8].

3.2 `data.dat`

, the input datafile, contains all simulation starting parameters and element setpoints that `TRANSOPTR` will use for its computation. **Note: line numbers on the left (and dots) added here for reference:**

```
 1......... 0.06 0. 0. 18629.9 1. 0.0
 2.......... 1 3 1. 1.0E-4
 3.......... 0 -0.
 4.......... .054 .0108  .054 .0108   2.667 0.000375
 5.......... 1.  1.  1.  1.  1.  1. 0. 1.
 6.......... 0
 7.......... 8
 8..........  3.2887  -10. 10. 1 ! IOS:Q1 Q1 kV
 9.......... -2.5937  -10. 10. 1 ! IOS:Q2 Q2 kV
10..........  2.4190  -10. 10. 1 ! IOS:Q3 Q3 kV
11.......... -2.444   -10. 10. 1 ! IOS:Q4 Q4 kV
12.......... -0.      -10. 10. 1 ! IOS:Q5 Q5 kV
13..........  3.9090  -10. 10. 1 ! IOS:Q6 Q6 kV
14.......... -2.0526  -10. 10. 0 ! IOS:Q7 Q7 kV
15..........  2.0526  -10. 10. 0 ! IOS:Q8 Q8 kV
16.......... 1.E-5 900
17.......... 1 0. 0.95 20
```

Commenting in the file is permitted: everything to the right of the ! character is ignored by `TRANSOPTR`. A line-by-line summary of the parameters is presented. All inputs are assumed to be real numbers (float), unless specified as an integer (int). Floats and integers should not be mixed.

3.2.1 data.dat line 1:

`[energy], [momentum], [b*rho], [mass], [chargestate], [current or bunch-charge]`

The first three entries on this line are energy ($(\gamma-1)mc^2$ in MeV), momentum (MeV/c), and $B\rho$ in Tesla-metres. Only one of these three is to be specified; the others should be zero to avoid confusion. Fourth entry is mass in MeV/c^2. Then particle charge number, and lastly either the current (Amp) or bunch charge (Coulomb) depending upon whether `IVOPT` is 4 or 5 resp. We note that in this example, items 2 and 3 are 0 with the input energy specified. Regarding the final parameter, if `TRANSOPTR` is run with `IVOPT`=5 (referred to as 'mode-5', see next line), the user specifies bunch charge in Coulombs, mode 4 expects a current in Amps; modes < 4 do not treat space charge so would ignore this parameter.

3.2.2 data.dat line 2:

`[IPRINT], [IVOPT], [initial Runge-Kutta stepsize], [RK error tolerance (relative)]`

The integer `IPRINT`, when negative, produces the consolidated output beam envelope file: `fort.envelope`. This places all possible outputs in one file: all 36 transfer matrix elements, the $\boldsymbol{\sigma}$-matrix as 6 envelopes and 15 correlations, the energy, the axial field, the emittances where appropriate, the element local focal strength. Each line is at a value of the independent variable s, and steps in s are controlled by the parameter `CMPS` described below.

If `IPRINT` > 0, the original `TRANSOPTR` output file structure is featured, with many outputs in various `fort.nn` files (`nn` is an integer), to be described below.

The absolute value of `IPRINT` controls the output in `fort.8`. See section 4.2.4.

The **second** entry `IVOPT` is an integer that determines the calculation 'mode'.

1. is deprecated.
2. has `TRANSOPTR` calculate the coordinates of a single particle, using stored transfer matrices.
3. calculates full $\boldsymbol{\sigma}$-matrix formalism, using analytic transfer matrices for the optics elements. This restricts the elements to those that are integrable in closed form, i.e. hard edged dipoles, quadrupoles and solenoids. This mode reproduces precisely the `TRANSPORT` first order optics.
4. Runge-Kutta integrates the 21 equations of motion of the $\boldsymbol{\sigma}$-matrix. This mode is used for elements that have no closed-form expressions for their transfer matrices, and also adds the linear space charge, but only for continuous beams.
5. is the same as 4 but for bunched beam space charge. The model used is that of an ellipsoidally-symmetric bunch in free space. It requires numerical evaluation of the Carlson elliptic integrals[2].

There is no way at the moment (2022) to track the single particle as in mode 2, through elements that have no stored transfer matrices. This is related to the way the code has developed, with at first the integration mode being used only for cases with space charge. Note that one can 'fake' a single particle by setting emittance to zero, but its six phase space coordinates will then be their absolute values.

The remaining two parameters are only relevant for modes 4 and 5, as they pertain to the Runge-Kutta integration process.

The stepsize entered as the **third** entry on this line is the initial **integration** step (**not** the print step), in cm. During integration, the integration step size is adaptively controlled by comparing relative errors in symplecticity with the specified **fourth** entry. To avoid calculations driving the step size to zero and thereby causing an infinite loop, there is a minimum step size that is somewhat arbitrarily set to 0.001 times the initial step size. It is safer therefore to set this initial step size too large rather than too small. If set to zero, it is automatically re-set to 1 cm. It should be set to something like the length of the typical focusing element, but only a few steps are wasted if set ten times too large.

Generally, this **fourth** entry should be somewhere between machine precision and square root of machine precision. (10^{-7} to 3×10^{-4} for single, though `TRANSOPTR` can be compiled in double precision. See section 4.1.6 on this point.) More details on `TRANSOPTR`'s RK engines can be found in TRI-BN-18-09 [9].

If there are warnings as *****WARNING:Minimum Step Size is reached at ...**, the user should try to diagnose by varying the initial step size, or going to the double precision compiler option.

3.2.3 data.dat line 3:

```
[int], [s_0], [s units], [B_s units], [x-Cart.], [y-Cart.], [z-Cart.]
```

This line has two to seven entries and serves three functions: it sets the initial value s along the reference Frenet-Serret axis, it allows to specify an ambient axial field along this axis, and it allows to specify a location in absolute Cartesian space for the start of the beamline. These last three are optional; they only apply to the `fort.3` output. (See section 4.2.6.) In addition to being able to specify the starting coordinates, it is possible to specify the initial direction of the beam, using `uxyz`(5.5.2).

The use of this line to input an ambient axial field is deprecated. It is retained to ensure backward compatibility, but we recommend to input stray magnetic field using the routine strayB, see section 5.2.2.

The **first** parameter is an integer specifying the number of values (s, B_s) in `fort.2` to be read. If zero, there is no axial field. Ambient axial field can clearly only be used in modes 4 and 5. **Parameter 2** is the starting value of the simulation's longitudinal co-ordinate s.

The **two last entries** allow for unit conversion of the pairs read in from `fort.2`. The **third** parameter is usually 1.; it converts the the first of every pair to the units used for longitudinal coordinates, see line 5 below. The **fourth** converts the second of every pair to kiloGauss.

The s-values read in are in the same coordinate system as the second parameter. Example: if parameter 2 is zero, and the `fort.2` values cover the range -100 to $+100$, the calculation will start at zero, i.e. in the centre of the data. In this case there will be a warning that the particles originate inside a magnetic field.

An example is injection and deceleration for soft landing ions into a solenoidal field. It is available here.

3.2.4 data.dat line 4:

```
[x], [P_x/P_0], [y], [P_y/P_0], [z], [P_z/P_0]
```

These are the six bunch dimensions: the squares of the diagonal elements of the $\boldsymbol{\sigma}$-matrix, or in the case of mode 2, the actual particle coordinates. Momenta are normalized by the particles momentum P_0 and so they are angles for x and y, $\Delta p/p$ for z.

Without space charge, one is free to use any desired convention: rms size, twice rms, or actual ellipsoid dimensions in a water-bag model case. Then the output of the calculation will also be defined in this manner.

But the **space charge** effect is only correct if the dimensions are **twice rms** for the continuous beam case (mode 4, in which coordinate 5 is ignored), and $\sqrt{5}$ **times rms** for the bunched case mode 5. See the Appendix or TRI-BN-21-04 [2] for further details.

Coordinates 5 and 6, z and P_z/P_0 are defined consistent with most other transport codes, often stated as “bunch length” and “relative momentum spread”. These are in fact not rigorous: bunch length requires simultaneous measurement of the positions of the front and rear edges of a bunch, something that we know since Einstein to be dependent on reference frame.

In fact, as the independent variable is not time but distance along the reference trajectory, coordinate 5 must be **time**, not length. To make it consistent with what most people understand as bunch length, we scale it by the speed of the reference particle. Thus coordinate 5 is,

$$z = \beta c \Delta t. \tag{8}$$

Canonically, the sixth coordinate is energy if the fifth is time, and we scale it also with the reference particle's speed:

$$P_z = \frac{\Delta E}{\beta c}. \tag{9}$$

Coordinate 6 is then $\frac{\Delta E}{\beta c P_0} = \frac{\Delta E}{\beta^2 \gamma m c^2}$.

In magnetic systems, there is no scalar potential, and we have $E = \gamma m c^2$, $\mathrm{d}E = \beta c\, \mathrm{d}P$, so the sixth coordinate is the usual $\frac{\Delta P}{P}$. But `TRANSOPTR` is not restricted to magnetic systems: there are electrostatic elements and RF devices where the scalar potential is not zero. A fuller explanation plus the difference between our definitions and those in other transport codes, is given in reference [25].

It should be noted that the initial beam need not be specified on this line. An alternative method is a call to subroutine `CIC` as the first element in the `sy.f` routine. It has the advantage that the initial beam parameters can be fitted, thus allowing e.g. to find matched envelopes for periodic systems such as synchrotrons.

3.2.5 data.dat line 5:

```
[x unit], [P_x unit], [y unit], [P_y unit], [z unit], [P_z unit], [flag], [s unit]
```

The first six entries are the units the user intends to use for the six dependent coordinates. Default dimensions for x, y, z are cm, meaning if the first third and fifth entries are 1.0, their units are cm. **Important**: this also applies to other transverse dimensions such as aperture size. Dimensions for $(P_x, P_y, P_z)/P_0$ are radians if entries 2,4,6 are 1.0. Entry 7 is normally zero, entry 8 applies to the independent variable and so to all lengths along the reference trajectory.

It of course makes little sense to have different units in the two transverse planes, so this is not a possibility. However, it is quite possible to use say mm for all transverse dimensions, and metres for dimensions, such as quad, drift lengths and bend radius that pertain to the distance along the reference trajectory.

The line contains the number of user's units per default unit. For instance, if one wishes to use inches for the x-dimension, the first entry in line 5 would read 0.3937 (=1/2.54). If one wishes the length unit along the beamline to be meters, entry 8 must be 0.01. As example, if one wishes to use cm for bunch dimensions, mrad for transverse momenta, percent for $\Delta p/p$, and meters for length, this line is

```
1. 1000. 1. 1000. 1. 100. 0. 0.01
```

(Easter egg: If entry 7 is 1.0, then the output on unit 8 (`fort.8`) for the `|IPRINT|`=1 case are in internal rather than user units. This allows easy conversion of the input `sy.f` file to cm. See section 4.2.4.)

3.2.6 data.dat line 6:

```
[(int)]
```

This is the number N_{corr} of non-zero correlation parameters to appear in the subsequent lines.

3.2.7 data.dat lines to $6 + N_{\text{corr}}$:

```
 i j  r_ij
```

(repeated N_{corr} times)

These are the indices and normalized off-diagonal elements of the initial $\boldsymbol{\sigma}$-matrix; also known as correlation coefficients $r_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}}$. These can be on separate lines or all together on one line; separators are spaces.

3.2.8 data.dat line $7 + N_{\text{corr}}$:

```
[(int)]
```

This line tells `TRANSOPTR` how many parameters to read in on the next number of lines; one parameter per line. The parameters are defined entirely at the user's discretion; there could be none, in which case there can be no fitting, or many. In the example file shown here, all 8 of the strengths of the 8 quadrupoles in `sy.f` can be varied. Note that parameters from top to bottom in `data.dt` are passed to the `COMMON` block variables from left to right. For instance, in `sy.f`, the block reads as:

```
      COMMON /BLOC1/q1,q2,q3,q4,q5,q6,q7,q8
```

with Q1 to 8 being stored in `data.dat` lines 8 through 15.

3.2.9 data.dat tuneable element lines:

```
[setpoint], [min value], [max value], [boolean: optimize?]
```

Each tunable element in the file needs a user specified setpoint (initial value). The minimum and maximum values for the setpoint are needed if `TRANSOPTR` is called to perform an optimization. The optimization call, which will vary the element setpoint to achieve a user-specified fit condition in `sy.f`, is enabled with the final boolean entry on the line, 1 for variable and 0 for fixed.

Parameters in the common `BLOC1` are not restricted to only control system variables such as quadrupole voltage; they can also be physical parameters such as drift lengths, element lengths, apertures, etc. Here is an example where 4 quadrupoles vary in strength and in **position**, a total of 8 parameters, to achieve a match from periodic transport into an RFQ. Note how the total length of this section is kept constant while the quad-quad drifts are all varied.

Neither is it necessary to restrict parameters to explicitly physical ones. For example a useful trick during beamline design is to allow element lengths to vary while also letting their **integrated**

strengths (product of length and strength) vary. This makes element strength to be an implicit parameter. The optimization works far more efficiently because individual focal element strengths and lengths are too sensitive and almost indistinguishable as fitting parameters. Changing the length without changing the integrated strength is far more productive in finding an optimumal match while minimizing aberrations.

3.2.10 data.dat second last and last lines:

`[CLIM], [MAXIT]`

`[METHOD],[TEMPTR],[DEC],[IITER]`

These last two lines supply parameters for the optimization routine, determining also whether simulated annealing is used. `CLIM` is the convergence criterion. If the relative change or the size (whichever is smaller) of the minimization parameter `CHI` is less than this number it is declared converged. Usually `CLIM` can be set to 10^{-4} in mode 4 or 5, but can be down to 10^{-7} in mode 3; it is the square root of the precision when doing a Runge-Kutta step. It would be 10^{-8} for double precision (`optr -d`).

`MAXIT` is the maximum number of iterations. The optimizer will stop if it is exceeded. Usually set to some number between 100 and 1000. For testing, it can be useful to set MAXIT to 1, causing `TRANSOPTR` to only run once.

`METHOD` is a flag, mostly used for purposes known only to the developer. The exception is that if it is 10, the simulated annealing parameters `DEC` and `IITER` are set automatically.

`TEMPTR` is the starting 'temperature' for the annealing procedure. If set to zero, simulated annealing is not used at all. If it is 1., all free parameters are allowed to explore the full parameter space set by the allowed range of the parameter in that parameter's input line. If already near optimum, temperature can be set to 0.1, reducing the range. If the optimum setting is unknown, setting the temperature to 10 will maximize chances of finding the minima, as the full parameter space is randomly sampled for many more iterations.

`DEC` and `IITER` set the cooling scheme for the 'temperature': For a given temperature, the parameters are allowed to fluctuate randomly and at each fluctuation, CHI is evaluated by running through `sy.f`. The number of evaluations of CHI at one temperature is `IITER`, after which the temperature is multiplied by `DEC` and another iteration started. Through many many calculations with many different numbers of parameters from 3 to 30, I discovered a rule. If there are lots of free parameters, I think it's clear that you need lots of evaluations or else the parameter space is not properly sampled. It turns out that `IITER` should be exponential in number of parameters p and I use `IITER` $= 2^p$. So for 5 parameters, `IITER` $= 32$. Further, for more parameters, you also need slower cooling. I use `DEC` $= 1. - 1./$`IITER`. This means for 5 parameters, `DEC` $= 1 - 1/32 = 0.96875$. If you set `METHOD` $= 10$, both `DEC` and `IITER` are set to these values above, and the last two entries in the last line of data.dat are ignored.

This would mean computing time is $\propto 2^{2p}$ and so is unworkable for $p > 6$ or so. The user is advised to rather subdivide the optimization.

4 Output

TRANSOPTR has no graphics capability; it only outputs lists of values as functions of the independent variable s. These include all transfer matrix elements and $\boldsymbol{\sigma}$-matrix elements, and element names and strengths, and orbit coordinates. The code is typically run from a script that also calls a graphics program such as gnuplot to plot the desired beam and element characteristics.

Text output occurs only at the initial parameter settings, and, if optimization has taken place, again at the final settings. There is no output while optimizing as the print statements in the main code are inhibited, the main.f control program temporarily setting IPRINT to zero.

The user can print output while running, simply by placing write statements in sy.f The first entry of the second line in data.dat is named IPRINT; it controls output. As well, it can be accessed as the first location in the common block /PRINT/, so the user can use it to print any parameter to an output file. During optimization, this is set temporarily to zero, so the user can test for it being nonzero and print the value only after fitting has ceased. For example, let PARM1 be a parameter of any kind in sy.f. Including a statement

```
write(123,*)PARM1
```

at any point will print its value at that point of the transport line every time sy.f is called. These will appear in a file named fort.123. This might not be what the user wants, as there may thousands of calls to sy.f as the code is looping trying to converge to a FIT. If the user instead has in their sy.f:

```
...
COMMON/PRINT/IPRINT
...
IF(IPRINT.NE.0)write(6,*)' Here is parameter 1',PARM1
```

then there will only be one value of PARM in the default output of unit 6, among the other lines of standard output.

4.1 Screen or unit 6 output

The terminal screen by default outputs some details related to the calculation: warnings, error messages, will tell you possibly why the calculation failed.

4.1.1 Beam parameters

Firstly, the beam and parameter properties from data.dat are echoed. Then the requested output resulting from executing sy.f are printed sequentially.

4.1.2 Requested outputs

The user's write statements, which include the basic technique of using IPRINT as outlined above, but there also are canned print statements at the user's disposal:

- `call print_transfer_matrix` gives complete transfer matrix at that point, properly formatted and converted to the user's units. In the special case when `|IPRINT|` is 3, this matrix is output in `COSY-`∞ format, see section 4.2.4.
- `call print_lattice` prints transfer matrix as above, but also prints Twiss parameters and phase advance of the transport, provided the transport represents one period of a transversely decoupled lattice. These are lattice Twiss parameters, not the beam Twiss parameters.
- `call vective(n)` If $n = 0$ nothing happens; this is a feature used internally. If $n > 0$, the $\boldsymbol{\sigma}$-matrix is printed, in a style where the first column is the 6 envelope sizes, and the remaining columns are a triangular matrix giving the 15 correlation parameters. If $n = 10$, the raw $\boldsymbol{\sigma}$-matrix is also printed; this is mainly for debugging.

4.1.3 Energy change

During execution, the beam basic parameters (energy, momentum, rigidity ($B\rho$), emittances) are re-echoed at every change in beam energy.

4.1.4 Aberrations

Fractional emittance growth increments are given as each element is traversed. If the weight in the argument string for the element call for aberration calculation is zero, it is not printed. The fractional emittance change in both planes, multiplied by the weight is one of the components of the minimization vector.

4.1.5 Output vector

Next, at the end of a printable (not-during-optimizing) pass through the `sy.f` there come the outputs that have been stored during execution. This is firstly the main summary of beam characteristics at every s-step (set by `CMPS`, see sec. 5.4.2). If there are more than 100 such steps, there is only the statement:

```
***** Too Many Elements: Output Vectors on unit 16
```

The format of this output (either directly on screen or in `fort.16`) is the six envelope sizes, the value of s, the element number, the element name. Fit parameters and results are given at the right margin. This is an old style dating back to `TRANSPORT`. Here is a snippet:

```
 ----------Output Vector----------

    x          theta         y          phi           l         delta      System       Transport
  ( cm )       ( rad)      ( cm )      ( rad)       ( cm )      ( rad)    Length( cm )  Element #

 0.40500     0.88000     0.40500     0.88000     0.14000E-01  1.0000       0.0000          0 ____
 0.39198E-01 0.35786E-01 0.39198E-01 0.35786E-01  1.0215     0.20318E-03  6.0000          1 Entr
 0.15342     0.26958E-01 0.15342     0.26958E-01  1.2801     0.11633E-03  12.000          2 Exit
 0.27049     0.26968E-01 0.27049     0.26968E-01  1.2802     0.12108E-03  16.388          3 .
```

4.1.6 RK results (mode 4 and 5)

In integration mode, various outputs related to the Runge-Kutta integrator are printed: time of flight, number steps, stepsize, symplectic error. The latter can be used to tune the integration tolerance: the symplectic error in units of the integration tolerance must not be allowed to be larger than 1. If it is, it means the integration adaptive stepsize is fighting against the error caused by machine precision, and the tolerance must be increased. Ideally, this ratio should be in the range of ~ 0.1 to 0.3. Typically, this requires RK error tolerance to be $\sim 0.3 \times 10^{-4}$.

4.1.7 Optimization results

First run: Next are the optimization parameters for initiating the optimization, then the optimizer figure of merit to be minimized (`sqrt(chi)`, the norm of the minimization vector) and the variable parameters are printed at each iteration. This enables the user to monitor in case the optimization goes awry. (Usually these are printed too quickly though.)

Final run: If there was successful minimization, the number of iterations used is printed as well as the individual components of the final minimization vector. Finally, all the parameters in `BLOC1` are echoed with their final optimized values. If `sqrt(chi)` did not decrease, there is no printout as it would simply duplicate the initial run.

4.2 Other output files

If `IPRINT` is negative, the three output files `fort.envelope, fort.label, fort.xml` are produced in place of `fort.1` (legacy envelope-plotting), `fort.81` (Twiss parameters in x), `fort.83` (Twiss parameters in y), `fort.91-fort.96` (the six rows of the transfer matrix), `fort.12` (label-plotting).

4.2.1 `fort.envelope`

This is the main output, it contains many columns of data. The columns have proper headings, so a graphics app such as `gnuplot` will be able to pick the correct data column as long as its header is given exactly. Here are the column headers given serially:

```
s      x-envelope x'-envelope  y-envelope y'-envelope  z-envelope z'-envelope
 kx            ky           Bs              E            ct
  x-emittance y-emittance z-emittance cartesian-x cartesian-y cartesian-z
   r12        r13        r23        r14        r24
    r34        r15        r25        r35        r45
     r16        r26        r36        r46        r56
      m11        m12        m13        m14        m15        m16
       m21        m22        m23        m24        m25        m26
        m31        m32        m33        m34        m35        m36
         m41        m42        m43        m44        m45        m46
          m51        m52        m53        m54        m55        m56
     m61       m62       m63       m64       m65       m66
      beta        FS-x       FS-x'       FS-y       FS-y'
```

The full header has two lines: the column names and its units.

- First column is s, the independent variable.
- The next 6 columns are the beam sizes (envelopes), i.e., the square roots of the diagonal elements $\sqrt{\sigma_{ii}}$ of the $\boldsymbol{\sigma}$-matrix. (See the separate section below for what is meant by 'beam size'.)
- Local focal powers in x and y.
- the ambient axial magnetic field.
- Two first-order beam coordinates, namely the energy and time as ct. These are non-trivial only if there is acceleration.
- The 3 spatial coordinates of the Frenet-Serret coordinates system. These are useful to the beamline layout designers.
- 15 correlation parameters; the off-diagonal elements of the $\boldsymbol{\sigma}$-matrix, but written in the usual normalized form $r_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}}$.
- The three emittances are given if the beam is sufficiently decoupled.
- The 36 transfer matrix elements.
- The β.
- The four coordinates of the reference particle with respect to the reference trajectory, labelled for Frenet and Serret. These coordinates are non-zero when beamline elements are misaligned or have unintended transverse fields.

4.2.2 `fort.label`

For plotting purposes: to add the name tags of the various beamline elements on the envelope plot.

4.2.3 `fort.xml`

Used to help build the XML ACC database for the beamline sections where all we had was a `TRANSOPTR` description of the optics layout.

4.2.4 `fort.8`

Particlular absolute values of `|IPRINT|` will auto-generate input files for various other transport codes, facilitating comparison with other transport codes; these will appear in `fort.8`:

1. `TRANSOPTR sy.f` bare system file, with fixed rather than variable parameters as arguments for the calls to elements. The elements are in the units chosen by the user, unless entry 7 of line 5 is non-zero. If this entry is 1, the outputs will be in the default units (cm, rad).
2. `GIOS` input file.
3. `COSY` input file. As well, in this case, the transfer matrices output by the `print_transfer_matrix` call will be printed in the `COSY` format: transposed matrix, in units of meters and radians, with `COSY`'s definition of coordinates 5 and 6.
4. `TRANSPORT` input file.

4.2.5 `fort.1`

Envelope dimensions, focal strengths, energy, axial field, for plotting. This file has been superseded by `fort.envelope` and only appears if `IPRINT` > 0.

4.2.6 `fort.3`

Layout file: Gives absolute coordinates of the reference orbit. This can be post-processed to plot the reference trajectory in 3D space. It is also useful for comparing the optics model with as-built drawings. The initial coordinates and direction in space can be set by a call to the routine `uxyz`(5.5.2).

4.2.7 `fort.14,15`

See section 5.3.1 below.

4.2.8 `fort.16`

This is an "overflow": if there are too many print steps, the envelope output vector appears here, as explained in section 4.1.5.

4.2.9 `fort.22`

Each element's vital characteristics are listed sequentially here. This is to facilitate hardware specification, and check that fields are realistic. For example, since benders are specified by only radius and angle, their fields are printed here.

4.2.10 `fort.81, fort.83, fort.85`

These apply to decoupled motion. Columns are element#, label, s, Twiss alpha, beta, emittance, dispersion, dispersion derivative, tune. 81 is for x, 83 is for y.

4.2.11 `fort.9i`

Contains the i$^{\text{th}}$ row of the transfer matrix, at every print step. This file has been superseded by `fort.envelope` and only appears if `IPRINT` > 0.

5 Callable routines and other statements used in `sy.f`

5.1 Closed-Form-Integrated Optics Elements

In the following listing the convention is that all the variable names ending in `u` are in units specified by the user. For example, a gap is in the transverse length unit, a length is in the user's longitudinal unit (entry 8, line 5: 3.2.5). All `wt` are weights for calculating aberration. Their effect is to add a fractional emittance growth to the vector $\vec{\chi}$. If set to zero, it is turned off. `qlab` is in every case simply the element's label. Argument strings ending in `0,0` are a holdover from a previous way of printing matrices. Just leave them at zero.

These will work in any mode, as their transfer matrices are known and hard-coded. For extended elements in space-charge modes (4 and 5), though, the $\boldsymbol{\sigma}$-matrix as numerically integrated using the coded infinitesimal transfer matrix.

5.1.1 X direction Magnetic Bend

`call BEND(Ru,TT,EN,qlab)` : Magnetic bender about y-axis; radius, angle, field index, label. Note bend can be in either direction by changing sign of both angle and radius.

5.1.2 X direction Magnetic Bend Edge, with Aberrations

`call edge(Uu,Ru,Theta,dum,AK1,AK2,Gapu,Enn,wt)` : Bender edge; edge angle, bender radius, bender angle, `dum` is no longer used, `AK1,AK2` are fringe field integrals K_1, K_2 as defined by Brown[16], full gap size in transverse units, field index, aberration weight.

5.1.3 Y direction Magnetic Bend

`call YBEND(Ru,TT,EN,qlab)` : See above; bend is in y direction, i.e. about x-axis.

5.1.4 Y direction Magnetic Bend Edge

`call yEDGE(Uu,Ru,Theta,Rabtu,AK1,AK2,Gapu,Enn,wt)` : See above.

5.1.5 X direction Electric Bend

`call EBEND(Ru,TT,cee,qlab)`

5.1.6 X direction Electric bend Edge

`call Eedge(Ru,Theta,cee,AK1,Gapu,wt)`

5.1.7 Y direction Electric Bend

`call YEBEND(Ru,TT,cee,qlab)`

5.1.8 Y direction Electric bend Edge

`call yEedge(Ru,Theta,cee,AK1,Gapu,wt)`

5.1.9 Parallel-plate electric deflector

`call deflectx(elu,theta,0,0)` : Like an electric bender, but this is a zero length device; it does not advance s. The length `elu` is a parameter to judge its focal strength. There is also a `defy`.

5.1.10 Drift

`call DRIFT(ELu,qlab)` : The simplest element.

5.1.11 Electric Quadrupole

`call EQUAD(VX,AAu,ELu,wt,qlab)` : First two arguments are the electrode voltage in kV, and aperture radius in transverse units. Then effective length in longitudinal units, aberration weight and label. In addition to the aberration calculation giving a fractional emittance increase that is minimized whenever `wt` is greater than zero, this routine can also fit the beam size to be smaller than the quad aperture. This fit is activated by specifying a **negative** aperture.

5.1.12 Wien Filter

`call WF(Ru,ELu,0,0)` : Wien filter; crossed electric and magnetic fields; if it had no magnetic field, it would bend beam in x-direction, with radius `Ru`. There is also a `WFY` for the y-direction.

5.1.13 Generalized Wien Filter

`call ExB(Ru,RBu,TT,cee,0,0)` : Electric and magnetic fields bend beam in opposite directions but do not cancel. `Ru` is the net bending radius and `RBu` is the bending radius for the magnetic field acting on its own.[5]

5.1.14 Electrostatic Mirror

`call MIrror(Zu,0,0)` : Bends beam by 90°. `Zu` is its height.

5.1.15 Magnetic Quadrupole

`call MQUAD(BXT,AAu,ELu,wt,qlab)` : `BXT` is poletip field in Tesla. Then as with the electric quad, aperture radius in transverse units, effective length, aberration weight and label. In addition to the aberration calculation giving a fractional emittance increase that is minimized whenever `wt` is greater than zero, this routine can also fit the beam size to be smaller than the quad aperture. This fit is activated by specifying a **negative** aperture.

5.1.16 Solenoid

`call solenoidn(BZT,ELu,AAu,wt,qlab)` : This is a simple solenoid with constant axial field `BXT` (Tesla). The aperture is used to calculate aberration.

5.1.17 Beam Rotation

`call RoTate(THETA,qlab)` : Rotation about the beam axis (degrees). This can be used to uncouple a rotated beam, or preceeding a bend to bend the beam in other than x and y directions.

5.1.18 RF Gap

`call RFGAP(V,phi,freq,asym)` : Gap of zero length. Voltage `V` is in megavolts, `phi` in degrees. Should only be used in mode 5.

5.1.19 RF Buncher Gap

`call RF(VCPH,QLOHu,0,0)` : Routine for rf gap, but without acceleration. It is meant only for the longitudinal focusing that comes from the slope of the waveform. Fist input is $V \cos(\phi_s)$ in MV i.e. voltage per unit phase and second input is either (machine length)/(harmonic number)=$\beta\lambda$ or (if < 0) it is frequency (Hz), and $\beta\lambda$ is calculated. In the case of mode 4 (continuous space charge mode), calling this routine will change the mode, `IVOPT`, to 5 (3D space charge). In that case, the input bunch length is ignored, and a new one calculated based on the frequency. Bunching will of course be sawtooth. However, departure from linearity can partly be taken into account be having an appropriate initial energy spread.

5.1.20 Smooth Focusing section

`call smoofc(betxu,betyu,ELu,0,0)` : First two entries are beta-functions.

5.1.21 Arbitrary Transfer Matrix

`call UdMAT(AA,dzed,0,0)` : User must give a full 6-by-6 transfer matrix. The independent variable is incremented by `dzed`. In some cases, the element with this transfer matrix also modifies the reference trajectory. In that case, execute `call UXYZMAT(AA,dzed,angX,angY,qlab)` where `angX,angY` are the angular changes of the trajectory direction (radians, these must be small).

5.1.22 Thin Lens

`call Thinlens(flu,0,0)` : A symmetric thin lens. `flu` is focal length. There is also a `Thinlens2(wflxu,xflyu,0,0)` for different focal lengths in x and y but in this case the arguments are reciprocal focal lengths.

5.2 Non-Closed-Form-Integrated Optics Elements

Though the intent of the ODE solver was to add space charge forces only, this opened the way to allow calculating optics through any element whose matrix is not calculable in closed form. The following additional elements do not function in modes 2 and 3.

5.2.1 Ambient Axial Magnetic Field

This is not a separate element but acts to augment the fields of other elements. As mentioned in section 3.2.3 the use of this functionality is deprecated (but retained for backward compatibility). We suggest that you use instaed the general ambient magnetic field element `strayB`, see section 5.2.2.

An axially-symmetric magnetic field is specified in `fort.2` which will contain the on-axis field values as a function of independent variable. The values are spline-interpolated, and radial fields are implicitly included because the momenta are transformed to canonical ones. This can be applied as a background field, through drifts with additional electric fields supplied for example by electrostatic quadrupoles or einzel lenses as well. This feature is especially useful for beam entering on-axis into an experiment's solenoid or into a cyclotron axial injection system.[4]

5.2.2 General Ambient Magnetic Field

This is a zero-length element which acts to augment the fields of other **downstream** elements. It take as input either one or all three components of the on-axis magnetic field $B_x(s)$, $B_y(s)$, $B_s(s)$. To expand the field off axis, we use the lowest-possible-order series that satisfies Maxwell's equations (in their homogeneous form) along the axis. This corresponds to the lowest-order field harmonics: solenoidal in the longitudinal and dipolar in the transverse directions. The `strayB` routine is called as follows:

```
call strayB(IUNIT, IB, bscale, zshift_u, izxy)
```

A total of `IB` lines are read from the input file `fort.IUNIT`. The number of columns in the input file, and what the data in each column represent, is determined by the value of the integer `izxy`.

- if `izxy` = 0, the data file must have 4 column containing s Bz Bx By (in this order: **Bz first!**)
- if `izxy` = 1, the data file must have 2 column containing s Bz
- if `izxy` = 2, the data file must have 2 column containing s Bx
- if `izxy` = 3, the data file must have 2 column containing s By

In all 4 cases the first column is given in the user-defined `[s unit]`, see section 3.2.5. `bscale` is a factor that scales the input field into what you want it to be in **Tesla**. Note that you can use `bscale` as a fitable parameter, see section 5.3.1. The reference point `s` = 0 for the values in the first column of your `fort.IUNIT` file is the position at which the `strayB` routine in called in your `sy.f` input file. The parameter `zshift_u` allows you to shift this reference point by a value given in the user-defined `[s unit]`, see section 3.2.5.

The routine `strayB` is actually a wrapper around the more general routine `strayB_absoluteZ` that you can call like this:

`strayB_absoluteZ(IUNIT, IB, bscale, zscale, zshift, izxy)`

The extra parameter `zscale` allows you to scale the coordinate in the first column of `fort.IUNIT`: this is usefull in the case the unit used in this file is not the user-defined `[s unit]`. Also, unlinke with `strayB`, the the values in the first column of `fort.IUNIT` are to be given w.r.t. the **absolute** `s` coordinate of your simulation (value which is accessible as the second variable of the common block `COMMON/ZED/ZINIT,Z`).

5.2.3 Axial Magnetic Field Element

`call AXBZ(IUNIT,IPO,BSCALE,ELu,qlab)` : This is similar to ambient axial magnetic field but is a separate element rather than a background field. The field is given in `fort.IUNIT`, `IPO` is the number of pairs of z, B to be read, `BSCALE` converts these to kG. `ELu` gives the length scale as the pairs cover this distance. This can be used to model a soft-edged solenoid whose field has been measured.

5.2.4 Arbitrary Axial Electric Field

`call AXEZ(IUNIT,IPO,VSCALE,ELu,nsubdr,0,0)` : This is same as `AXBZ` above[6]. The field is given in `fort.IUNIT`, `IPO` is the number of pairs of z, V to be read, `VSCALE` converts these to kV. `ELu` gives the length scale as the pairs cover this distance. `nsubdr` is not used. This can be used for general acceleration columns and einzel lenses.

Axial electric fields can also be combined and separately controlled. This is the case for example inside an EBIS. In that case, `call AXEZ_MULTI(IUNIT,IPO,VSCALE,NINP,ELu,ISPL,qlab)`. `NINP` is the number of columns of potentials, `VSCALE` is a vector of scaling potentials, whose length is `NINP`. `ISPL` is the spline flag for the end conditions.

5.2.5 Soft-edge Acceleration Column

`call ACCEL_KOST(VSMV,AAu,ELu,nsubdr,qlab)` : Linear acceleration column with cubic entry and exit potentials.[22] `VSMV` is in megavolts, `Elu` is not the physical length but the physical length plus 4 times the aperture radius `AA`.

5.2.6 Soft Solenoid

`call CMR(CUR,AAu,wt,qlab)` : This is a solenoid modelled as a current loop, as in COSY-∞[11]. It has zero length so integration will backtrack the required distance before using the field, and then again backtrack to the centre when done.

5.2.7 Einzel Lens

`call EINZEL(VkV,AAu,ELu,gapu,nsubdr,0,0)` : Similar to the COSY-∞ routine CEA[11]. `VkV` is in kV, ELu is length, being inner electrode plus 2 times the gap plus 4 times the aperture radius.

5.2.8 Spiral Inflector

`call INF(Eradu,Bradu,TILT,0,0)` : This is the commonly used device to steer the beam from the cyclotron axis onto the median plane. It is a complex device, coupling all 3 directions.[4] Electric radius `Erad` also known as its height. This must be inside an axial magnetic field; `Brad` is the beam's rigidity divided by magnetic field. If `Brad` is zero, the field is given by the ambient field as read in from `fort.2`.

5.2.9 Linear Accelerator

`call LINAC(IUNIT,IPO,VSCALE,ELu,freq,phasedeg,nsubdr)` : On-axis electric field is specified similar to the axial DC field above. However, not knowing a priori either the phase or the energy at any location, it necessitates simultaneously finding those first moments.[7]

5.2.10 Short Soft-Edge Quadrupole

`call MSQUAD(STRK,AAQu,wt,qlab)` : Short quadrupoles are very poorly described by the standard constant (and discontinuous) strength function. This routine uses a bell-like strength function that describes short-limit (length $\sim$ aperture) quadrupoles.[1]

5.2.11 Permanent Magnet Axial Lens

`call PMS(BZ0,AAQu,RAQu,QLENu,wt,qlab)` : Sometimes referred to as a "permanent magnet solenoid", but this is incorrect as there is no beam rotation. It consists of many azimuthal sectors, all with their magnetization oriented radially. It thus creates an axial field that changes sign at the lens' centre. The arguments are: field in Tesla, aperture radius, outer radius, length, aberration weight, label.

5.2.12 RFQ

`call RFQn(IUNIT,IPO,VMV,ELu,freq,phasedeg,nsubdr,qlab)` : A total of `IPO` values of the vane-shape parameters s[`elu` units], A_{01}, A_{10}, k[inverse `elu` units] are read from `fort.IUNIT`. `VMV` is vane voltage in MV. Others are obvious; `nsubdr` is not used. In contrast to other RFQ codes, the integration is not done cell-by-cell, but step size is governed by an error tolerance. More details on input specification can be found in TRI-BN-22-07[26].

5.3 Calls for fitting

5.3.1 `FIT`

The basic fit call is

```
call FIT(N,I,J,VALEL,WEIGHT,IVAL)
```

There are only two possible values for `N`: 1 means fitting a $\boldsymbol{\sigma}$-matrix element, and 2 means fitting a transfer matrix element. `I` and `J` are the matrix index, `VALEL` is the desired value. For `N=1` the value of the fit target `VALEL` is not directly the matrix element, but rather if it is a diagonal element, it is the size directly, namely, the square root of the matrix element. For off-diagonal elements, it is not directly σ_{ij}, but the normalized value: $r_{ij} = \sigma_{ij}/\sqrt{\sigma_{ii}\sigma_{jj}}$. `WEIGHT` is the factor by which the difference between desired and found values are scaled. If it is zero, the fit is cancelled. `IVAL` is the exponent. Ordinarily, `IVAL=1` and the weighted difference is added as a component to the vector $\vec{\chi}$. Since the minimization is applied to the norm of this vector, the effect of this fit as compared to other fits comes in as the square of the weighted difference. If `IVAL=2`, it comes in as the fourth power, and so on. An exception is when `IVAL=0`. In that case, the fit condition is that the found matrix element is **less** than `VALEL`, rather than equal. This is useful for example for fitting the beam size to be smaller than a given aperture size. It is built into the quadrupole routines and can be switched on without a separate fit call, by simply giving a quadrupole a negative aperture.

Lastly, it is also possible to fit the coordinates of the reference particle. For example, this allows a user to find steerer settings to place the beam back on axis. For this case, `I=13`, and `J` is either 1,2,3, or 4, respectively for fitting x, x', y, y'.

There is a variant

```
call FITN(N,I,J,VALEL,WEIGHT,IVAL,qlab)
```

that allows one to name the location of the fit.

Another is

```
call fitsz(SZX,SZY,WX,WY)
```

which simply calls two fits, one for each transverse direction to fit the beam size. The `WX,WY` are the weights. It has the added feature that the desired size and fitted size are listed along with the independent variable s in files `fort.14`, `fort.15`, for easy plotting on a graph with the envelopes. This is useful when beam sizes are measured by for example scanning wires, and it is desired to fit the initial beam.

In the following routines, `N,I,J,WEIGHT,IVAL` have the same meaning as above.

One special consideration for fitting matrix elements is that often the matrix to be fit is **not** the matrix from the start of the transport line, but from another point. An example is where one wishes to have point-to-point focus from some location (A) to another location (B). In that case, the `FIT` call is placed at B, and another call:

```
call resetrm
```

is placed at A. But this resets all matrix elements to the identity matrix, so any subsequent matrix fit call will only apply to the matrix accumulated from A onward.

5.3.2 MCOMP

```
call MCOMP(N,I,J,IE,JE,RATIJ,WEIGHT,IVAL)
```

This is to fit two matrix elements (`I,J` and `IE,JE`) to be in a ratio `RATIJ`.

5.3.3 WAIST

```
call WAIST(Ixory,WEIGHT,IVAL)
```

Fits Twiss α_x or α_y to zero depending on whether `Ixory` is 1 or 3.

5.3.4 TWISSMATCH

```
call twissmatchu(Ixory,alpha,beta,WEIGHT,IVAL)
```

Fits α_x, β_x or α_y, β_y to `alpha,beta` depending on whether `Ixory` is 1 or 3. β is in the units of length scale (entry 8, line 5: 3.2.5). You can also `call twissmatch(Ixory,alpha,beta,WEIGHT,IVAL)`, but there, `beta` must be in units of cm.

5.3.5 RESOLUTION

```
call resolution(Ixory,dres,WEIGHT,IVAL)
```

`dres` is the desired resolution. The fit is made by combining the betatron component of beam size and the dispersion. `IVAL` has the same meaning as above except that negative values are allowed. For the exponent the absolute value is taken; a negative value means `dres` is interpreted as the logarithm of resolution.

5.3.6 `FITARB`

```
call FITARB(dsrd,curnt,WEIGHT,IVAL)
```

Fits an arbitrary parameter of the user's own choosing. `curnt` is the found parameter, and `dsrd` is its desired value.

5.4 Common blocks

The user is free to access many other variables in the calculation. This is done by including the relevant common block. Besides the `/PRINT/` common, mentioned in section 4, a few others should be mentioned.

5.4.1 `BLOC1`

```
COMMON /BLOC1/<All the parameters passed from data.dat are here.>
```

as stated, this contains all the parameters that the user wishes to pass from the data file `data.dat` to `sy.f`. Besides optimizable ones such as quad strengths, or positions through drift lengths, this list can contain any flags for if statements; the parameters need not be related to optimization at all. The internal arrays are dimensioned to allow up to 100 parameters.

5.4.2 `CMPS`

```
COMMON/SCPARM/QSC,ISC,CMPS
```

allows the user to set `CMPS`, the number of cm (not user units) per print step. This makes plots smoother, less kinky. It subdivides every element. As well, the space charge parameter `QSC` can be scaled for example to take into account neutralization effects that change depending upon local electric fields.

5.4.3 `MOM`

```
COMMON/MOM/P,BRHO,pMASS,ENERGK,GSQ,ENERGKi,charge,current
```

contains beam dynamic parameters: momentum, rigidity, kinetic energy ($(\gamma - 1)mc^2$), γ^2, initial kinetic energy, particle charge, current or bunch charge for mode 5.

5.4.4 `SS`

```
COMMON/SS/SX(13,6)
```

contains both the beam and the transfer matrix. In general it is recommended to use the `FIND` routine rather than this raw matrix.

5.4.5 RKKIND

```
common/RKKIND/IRK
```

allows the user to choose among 4 different Runge-Kutta engines[9].

5.5 Other Calls

5.5.1 FRINGEQ

```
call fringeq(0.35,0.00,0.24,-0.40)        ! typical mag quad
call fringeq(0.087,0.005,0.033,-0.234)        ! typical electric quad
```

sets the quadrupole fringe field integrals (I_1, I_2, I_3, I_4) as defined by Wollnik[27]. These are corrections to the first order effects of quadrupoles and are imposed at entrance and exit. (Except for I_4 which corrects the aberration.) The corrections are often negligible when quad length is large compared with its aperture, so for such cases, no call is needed. A new call must be made for every quadrupole type. For a series of quads all with same fringe fields, this needs to be called only once at the start.

5.5.2 UXYZ

```
call uxyz(0.,angle,iaxis,dum1,dum2,dum3)
```

can be used to set the initial direction of the reference trajectory. It is used here to rotate by `angle` radians about the `iaxis` axis, 1,2,3 for resp. x, y, z. (Normally it is a utility routine used internally to update the absolute space coordinates of the reference trajectory, where the first argument is the length of the element, and outputting the last 3 entries as the new coordinates.) This routine affects only the output on `fort.3`. (See section 4.2.6.)

5.5.3 ADDSCATTER

```
call addscatter(SCAT,SCAT,SCATL,E)
```

causes the beam to scatter in angle by `SCAT` in x' and y', add `SCATL` to $\Delta p/p$, and resets the energy to `E`. This breaks the continuity of the transfer matrix so that it restarts from identity matrix in mode 3 case, but not in integration mode (4 or 5).

5.5.4 SLIT

```
call SLIT(xhsize,yhsize,wt,qlab)
```

The call compares the beam envelope to the slit size and calculates a beam loss, in x and y. The slit dimensions `xhsize, yhsize` are the **half**-sizes of the slit. The weight `wt` for fitting the cost of the slit is the negative logarithm of the fraction of transmitted beam, based on an assumption of gaussian profiles. x and y contributions to loss are added together. See beam note[10]. `SLIT` is for all modes 2,3,4,5 (`SLIT4` is deprecated). This routine does not handle dispersed beams or beams with any other coupling; the coupling in the beam matrix is simply zeroed.

5.5.5 CIC, CIC3

```
call cic(ax,bxu,exu,ay,byu,eyu)
call cic3(ax,bxu,exu,ay,byu,eyu,az,bzu,ezu)
```

sets the initial beam in such a way that these can be optimized. `cic` is for continuous beams or where the longitudinal dimension is not of interest, `cic3` is for bunches. Inputs are $(\alpha, \beta, \epsilon)$ for each dimension, all in user units: $\beta_{x,y}$ in length (s, or unit parameter 8) units, $\epsilon_{x,y}$ in units of $x \times$ units of x'. This is used for example for matching a periodic section, or for fitting initial beam from experimental measurements. An example of such use is in the following:

```
call cic(ax,bxu,exu,ay,byu,eyu)
call EQuad(Q111,1.00,4.+dx,0.,'Q111')
call DR(5.0600-dx/2., 0,0)
call DRift(1.95000,'W061')
CALL FIT(1,1,1,sV, 1., ifv)
CALL FIT(1,3,3,sH, 1., ifh)
```

Here x and y beam sizes have been measured at `W061` for many different values of the quadrupole strength `Q111`. This segment is run in a `DO` loop for the different values of `Q111` and the sizes `sV,SH`, while the optimizer varies the initial beam conditions `(ax,bxu,exu,ay,byu,eyu)`, which have for this purpose been placed in common block `BLOC1`. In this way the inital beam is fitted.

5.5.6 RESETUP, REGSETUP, REASETUP,RESETRM

```
call RESETUP
```

will reset the calculation to start again, while still accumulating the error vector $\vec{\chi}$. This is called implicitly by `CIC`.

There are variants: if you want to restart the independent variable s and absolute coordinates over, then call `REGSETUP`. When plotting, this will put all passes on top of each other. `fort.3` output is still sensible as well. If you have only a modification to the beam, such as scattering, where the angles are augmented, or slits, when the envelope widths are, then call `REASETUP`. This is used in `ADDSCATTER`.

Finally, as mentioned in section 5.3.1,

```
call RESETRM
```

to restart the transfer matrix accumulation. All subsequent prints or fits of transfer matrices will then be only the matrix from that point onward.

5.5.7 DISP_MAT

```
call disp_mat(etaxu,etapxu,etayu,etapyu,iv,im)
```

will artificially add dispersion to the beam. This can be used to remove the dispersion component of the beam size, investigate it, and then re-constitute it. Arguments are x and y dispersion plus derivatives. This is done symplectically so the bunch length is also correctly affected. Ignore the last two arguments.

5.5.8 FIDUCIAL

called with no arguments, it simply sets a print position; it is a zero-length drift.

5.5.9 FIND

```
call FIND(N,I,J,A)
```

allows user to find and presumably use matrix element `IJ`. `N=1` means a $\boldsymbol{\sigma}$-matrix element, and 2 means a transfer matrix element. `A` is the element value. Note that for off-diagonal elements of the $\boldsymbol{\sigma}$-matrix the normalized form is returned $r_{ij} = \sigma_{ij}/\sqrt{\sigma_{ii}\sigma_{jj}}$. For diagonal elements of the $\boldsymbol{\sigma}$-matrix the root of the element, $\sqrt{\sigma_{ii}}$ is returned.

5.5.10 GENSHIFT

```
call GENSHIFT(dx,dpx,dy,dpy)
```

allows user to shift the beam from the axis of an element. There are two basic uses: Shifting in x, y before and after a beam element models misalignments, and shifting in p_x, p_y models the effect of a short local steerer. This has an effect in mode 2 (where only the reference particle is tracked), or in mode 4 and 5, where the reference particle coordinates are tracked along with the σ-matrix. The mode 2 case only operates for stored matrices and not in integration mode. For elements whose characteristics are functions of s, mode 4 or 5 is required.

A Appendices

A.1 History

TRANSOPTR is a fully 6-dimensional beam envelope code. It integrates the equations of motion of the second moments ($\boldsymbol{\sigma}$-matrix) of the particle beam. For a bunched beam, there are 21 possible second moments. This follows from the “sigma-matrix” formalism first developed in the 60s: the elements of the $\boldsymbol{\sigma}$-matrix are essentially the beam’s second moments. The early codes developed to calculate particle beams and design beamlines were TRANSPORT GIOS, TRACE, and many more, of which TRANSPORT became the standard. It was however not a beam envelope integrator; instead it had stored the known analytic transfer matrices of standard elements such as quadrupoles, dipoles, solenoids.

This is how TRANSOPTR also originally functioned also. TRANSOPTR however, was written by Heighway and Hutcheon[20] to broaden the fitting capabilities of TRANSPORT. The limitation to analytically integrable elements remained however, so that in either code, the only way to include space charge was to subdivide all elements into sufficiently small increments and interleave with artificial thin lenses, whose strengths depended upon the beam size. Kapchinsky and Vladimirsky had shown in 1959 [21] that uniformly populated ellipses in 2D subspaces can be transported self-consistently because the space charge forces are entirely linear in this limit. However, it was known that uniform charge density in the 4 phase spaces of a DC beam was unrealistic and moreover not even possible for the 6 subspaces of a bunched beam [19]. So it was felt at the time that serious treatment of space charge effects would only be possible with multiparticle simulations, and not reliable if treated in the statistical approach of the $\boldsymbol{\sigma}$-matrix formalism.

This changed in 1970 when Sacherer demonstrated that: *“In fact, the K-V equations are not restricted to uniformly charged beams, but are equally valid for any charge distribution with elliptical symmetry, provided the beam boundary and emittance are defined by rms (root-meansquare) values... This is also true in practice for three-dimensional bunched beams with ellipsoidal symmetry, and allows the formulation of envelope equations that include the effect of space charge on bunch length and energy spread”*[23, 24]. These envelope equations are the 21 first order equations that govern the evolution of the $\boldsymbol{\sigma}$-matrix, as shown below. In 1983, deJong and Heighway incorporated these equations into TRANSOPTR converting it from a transfer matrix code to an integration code [18].

Completely aside from the capability of space charge envelope calculations, this allowed also the calculation of any element whose fields are known but whose matrices cannot be found analytically. These include: axial magnetic fields such as soft edge solenoids, einzel lenses and electrostatic accelerators in general, soft-edge quadrupoles, and many others. In principle, any transport element whose Hamiltonian is known, can easily be coded into TRANSOPTR. Linear accelerators have been added: cylindrically symmetric in 2015[3] and RFQ in 2019[25].

A.2 What is ‘Beam Size’?

TRANSOPTR calculates the “sigma-matrix”, either by matrix multiplication or by directly RK integrating the equations of motion.

1. The `S1` in `COMMON/SIGS/S1(6,6),RX(6,6)` (for the first type) and `SX` in `COMMON/SS/SX(13,6)` (for the integration case) are the "$\boldsymbol{\sigma}$ matrix", often denoted as σ_{ij}. (`SX` also contains 7 other rows; the first 6 are the $\boldsymbol{\sigma}$-matrix, the next 6 are the transfer matrix, and the 13th is the six coordinates of the reference particle w.r.t. the reference trajectory.)
2. The correlation parameters `RX` are normalized $\boldsymbol{\sigma}$-matrix elements: `RX(I,J)`$= \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}}$, so do not scale. For example, they are independent of emittance. These are only used during input and output and not used internally.
3. Absent of space charge and aberrations and slits, the scale of the $\boldsymbol{\sigma}$-matrix can be anything. That is because it is a linear optics code. If you make a calculation where neither of these effects are relevant, it is your choice: you can use rms values, twice rms values; you can even use an artificial scale parameter such that the σ_{11} value is the Courant-Snyder (Twiss) function β_x directly in metres.
4. The "envelope" is plotted from the outputs in `fort.1` or `fort.envelope` and these are the square roots of the diagonal elements of the $\boldsymbol{\sigma}$-matrix. So again absent of space charge and aberrations, this envelope reflects how the beam data has been input. If you have a model of some binomial distribution and you want the envelope to be the 100% emittance, then your inputs should reflect that, and they will be scaled by some appropriate factor w.r.t. rms size.
5. However, if you want the linear space charge effect to be correct for your distribution, and/or also the aberration estimations to give actual emittance fractional growth, you no longer have a choice on scale, as these effects are built into `TRANSOPTR` for only one assumption of the definition of the $\boldsymbol{\sigma}$-matrix. This assumption is that for (a) continuous beams (mode 3,4) the beam rms size ($= \sqrt{\int \mathrm{d}y' \int \mathrm{d}y \int \mathrm{d}x' \int \mathrm{d}x x^2 f(x, x', y, y')}$, f is the normalized distribution function) is one half of the square root of the appropriate diagonal $\boldsymbol{\sigma}$-matrix element and (b) for bunched beam (mode 5) rms size ($= \sqrt{\int \mathrm{d}z' \int \mathrm{d}z \int \mathrm{d}y' \int \mathrm{d}y \int \mathrm{d}x' \int \mathrm{d}x x^2 f(x, x', y, y', z, z')}$) is $1/\sqrt{5}$ of the square root of the diagonal $\boldsymbol{\sigma}$-matrix element.
6. The origin of the factor of 2 and $\sqrt{5}$ is as follows. The space charge calculation is in fact exact for uniformly filled ellipsoids. In the continuous case, the edge of this uniform beam is exactly twice the rms size, and in the bunched case, it is exactly $\sqrt{5}$ times the rms size. Remember that in calculating rms size, all dimensions are projected. Thus, a bunch, being smaller transversely at the ends than in the middle, has a larger ratio of envelope size to rms size than a continuous beam has. The space charge effect is exact for uniformly filled ellipses or ellipsoids in real space, but as Sacherer[23] pointed out, are still correct to a good approximation for rms sizes when distributions are not uniform.

A.3 `FORTRAN`

For a primer on `FORTRAN`, I warmly recommend a small note by David A. Clarke[17]. However, you might get by with the following minimal set of notes.

- As the language was written when input was via cards, there are limits on number of characters per line, and character placement matters. Think of fixed width characters, each in its own column numbered 1 through 72:

  - Yes, use only 72 columns. The 73rd column and beyond are ignored.
  - The first 6 columns are special; most code is in columns 7 to 72.
  - Column 1 containing a `c` (or a `C` as upper/lower case makes no difference) means that the rest of the line is a comment; it's not code.
  - Column 6 containing anything but a zero is a continuation code, meaning it is a continuation of the previous line of code.
  - Columns 1 to 5 are used as a **numerical** integer label, for branching, looping, format statements.
  - You can also put a comment on a line containing code. Everything after a `!` is comment and not code.

- Case doesn't matter. E.g. `goto` and `GOTO` and `GoTo` are identical code.
- Variables' names must be no more than 8 characters, start with a letter, but the remainder can be any of the 26 alphabet letters, 10 digits, and underscore; nothing else.
- **Important:** If no special care is taken, variables whose names start with letters `I` through `N` (upper or lower case) will be assumed to be integers. Real numbers will be truncated. Specail care means there is no `IMPLICIT` statement in the routine to re-define this convention.
- Special characters are used for operations. `+,-,/,*` are obvious, but the operation for exponential is a double asterisk `**`.
- Write statements are in the form `write(n,m)VAR1` where `n` is a 'unit number' and `m` is the label of a format statement and `VAR1` is an example variable name. Unit number 6 is the screen (monitor), so if you use it, it will appear in the screen output. All other units, will print the variable to a file named `fort.n`.
- For input, `read` statements work the same way, but unit 5 is your keyboard. E.g. `read(4,*)VAR2` will open a file named `fort.4` and read a line and first real number it comes across will be placed in the real variable memory location named `VAR2`.
- Large or small real numbers that need exponents are both input and output in a weird format: whatever follows the `E` or `D` is interpreted as a power of 10. Thus e.g. `5.60e-4` is 5.60×10^{-4}.
- A `SUBROUTINE` is called with a `call` statement that includes its argument list. The variables in a subroutine are transferred in this way, and need not have the same name in the subroutine as in the calling routine. The variables in the routine are not known to the other routines **unless** there is a `COMMON` statement that shares them. `COMMON` block names (the name between slash (`/`) symbols) are, however, universal.

References

[1] R Baartman. Quadrupole shapes. *Phys.Rev.ST Accel.Beams*, 15:074002, 2012.

[2] R Baartman. Bunched beam space charge in transoptr. Technical Report TRI-BN-21-04, TRIUMF, 2021.

[3] RA Baartman. Fast Envelope Tracking for Space Charge Dominated Injectors. In *Proceedings, 28th International Linear Accelerator Conference (LINAC16): East Lansing, Michigan, September 25-30, 2016*, page FR1A01, 2017.

[4] RA Baartman and W Kleeven. A Canonical Treatment of the Spiral Inflector for Cyclotrons. *Particle Accelerators*, 41:41–53, 1993.

[5] Richard Baartman. The TRIUMF Deflector, and Generalized Wien Filter. Technical Report TRI-DN-07-06, TRIUMF, 2007.

[6] Richard Baartman. Bunch dynamics through accelerator column. Technical Report TRI-BN-10-01, TRIUMF, 2010.

[7] Richard Baartman. Linac Envelope Optics. Technical Report TRI-BN-15-03, TRIUMF, 2015.

[8] Richard Baartman. `TRANSOPTR`: Changes since 1984. Technical Report TRI-BN-16-06, TRIUMF, 2016.

[9] Richard Baartman. Runge-Kutta Customization for TRANSOPTR. Technical Report TRI-BN-18-09, TRIUMF, 2018.

[10] Richard Baartman. SLIT routine for TRANSOPTR. Technical Report TRI-BN-19-21, TRIUMF, 2019.

[11] Martin Berz. COSY INFINITY Version 8.1 — User's Guide and Reference Manual. Department of Physics and Astronomy MSUHEP-20704, Department of Physics and Astronomy, Michigan State University, 2002.

[12] Karl L. Brown. A first and second order matrix theory for the design of beam transport systems and charged particle spectrometers. *Adv. Part. Phys., also internal report SLAC-75*, 1:71–134, 1968.

[13] Karl L Brown, David C Carey, F Christoph Iselin, and F Rothacker. *TRANSPORT: a computer program for designing charged-particle beam-transport systems.* CERN Yellow Reports 80-04,73-16. CERN, Geneva, 1980. Also publ. as SLAC-91 and FERMILAB NAL-91.

[14] KL Brown. A first-and second-order matrix theory for the design of beam transport systems and charged particle spectrometers, SLAC 75, 1971.

[15] KL Brown, R Belbeoch, and P Bounin. First- and second-order magnetic optics matrix equations for the midplane of uniform-field wedge magnets. *Rev. Sci. Instr.*, 35, 4 1964.

[16] KL Brown, DC Carey, FC Iselin, and F Rothacker. *TRANSPORT: a computer program for designing charged-particle beam transport systems; 2nd ed.* CERN Yellow Reports: Monographs. CERN, Geneva, 1973. Also publ. as SLAC and FERMILAB.

[17] David C. Clarke. A FORTRAN Primer. 2011.

[18] MS De Jong and EA Heighway. A first order space charge option for TRANSOPTR. *IEEE Trans. Nucl. Sci. (Proc. PAC'83)*, 30(4):2666–2668, 1983.

[19] RL Gluckstern. Notes on beam dynamics in linear accelerators. Technical report, Los Alamos Scientific Lab., 1980.

[20] EA Heighway and RM Hutcheon. `TRANSOPTR`–a second order beam transport design code with optimization and constraints. *Nuclear Instruments and Methods in Physics Research*, 187(1):89–95, 1981.

[21] I.M. Kapchinskij and V.V. Vladimirskij. Limitations Of Proton Beam Current In A Strong Focusing Linear Accelerator Associated With The Beam Space Charge. In *Proceedings, 2nd International Conference on High-Energy Accelerators and Instrumentation, HEACC 1959: CERN, Geneva, Switzerland, September 14-19, 1959*, pages 274–287, 1959.

[22] Corrie Kost and Werner Joho. `SPEAM`: A Computer Program for Space Charge Beam Envelopes. Technical Report TRI-DN-73-11, TRIUMF, 1973.

[23] FJ Sacherer. RMS envelope equations with space charge. Technical report, CERN DL/70-12, 1970.

[24] Frank J Sacherer. Rms envelope equations with space charge. *IEEE Transactions on Nuclear Science*, 18(3):1105–1107, 1971.

[25] O Shelbaya, R Baartman, and O Kester. Fast radio frequency quadrupole envelope computation for model based beam tuning. *Physical Review Accelerators and Beams*, 22(11):114602, 2019.

[26] Olivier Shelbaya. ISAC-RFQ Parameter Documentation . Technical Report TRI-BN-22-07, TRIUMF, 2022.

[27] H Wollnik, J Brezina, and M Berz. Gios-beamtrace–a program package to determine optical properties of intense ion beams. *Nucl. Instrum. Meth.*, 258:408–411, 1987.