J-PARC Trace3D Upgrades

Christopher K. Allen
Los Alamos National Laboratory

Trace3D "Project"

- Split the big FORTRAN file
  - Each SUBROUTINE has file
  - Project built using a Makefile (K. Furukawa)
  - Version controlled (K. Furukawa)
    - Repository jkksv01.j-parc.jp:/jk/master/t3d_cka
- Can reassemble back into the big FORTRAN file if desired

General Additions

- **Debugging and Data I/O**
  Added output subroutines for debugging and recording Twiss parameters along beamline
  - InputDump – stores input "deck" to disk (file "InputData.txt") to check that data is being properly input
  - TwissDump – stores Twiss parameters along beamline to disk file "TwissDump.txt". Twiss parameters are stored during simulation since trajectory data is not kept in Trace3D memory.
- Convenience Matrix Functions
  - Multiply, scalar multiply, commutator
  - Matrix exponent
  - Matrix logarithm (E. Forest)

General Additions (cont.)

- **Steering Magnets**

  Added steering magnets to the element library.

  - Note that only the centroid tracking is affected by this element, it has no effect on the second-order moment dynamics.
  - The type identifier is 19 (NT=19).
  - The parameters is $\Delta x'$, $\Delta y'$

General Additions (cont.)

- Arbitrarily Oriented Beam Ellipsoids

  To compute the electric self forces of the beam Trace3D performs a coordinate transform to "eigen-coordinates" of ellipsoid

  - In the current version this transform fails when the beam is skewed arbitrarily off the design axis
    - That is, skewed in a direction other that one of the two transverse directions.
  - This condition was fixed.

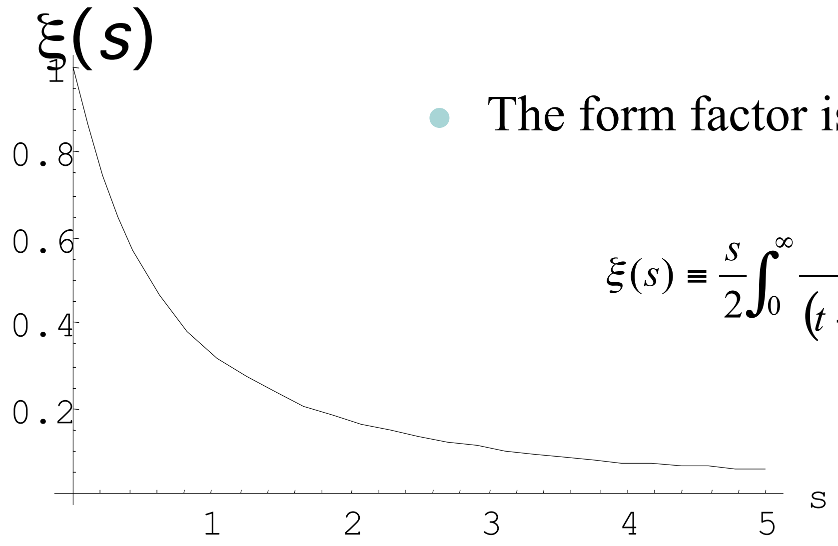
Choice of Elliptic Integral R_D or Form Factor ξ

- For the space charge calculations you may switch between the use of the form factor and direct (numerical) evaluation of the elliptic integral.
  - This feature is available using the input variable **iFlgUseRd** in the $DATA section of the Trace3D input file
  - iFlgUseRd = 0: use form factor ξ (default)
  - iFlgUseRd = 1: use elliptic integral *RD*

Form Factor vs. Elliptic Integral

- Original Trace3D uses a “form factor” ξ in the self field calculations.
  - The function $\xi(s)$ is part of an analytic approximation to an elliptic integral R_D, which is defined

$$R_D(x,y,z) \equiv \frac{3}{2}\int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)^{1/2}(t+z)^{3/2}}$$



- The form factor is defined

$$\xi(s) \equiv \frac{s}{2}\int_0^\infty \frac{dt}{(t+1)\left(t+s^2\right)^{3/2}} = \frac{1}{1-s^2}\begin{cases} 1-\dfrac{s}{\sqrt{1-s^2}}\cos^{-1}s & \text{for } s<1 \\ 1-\dfrac{s}{\sqrt{s^2-1}}\cosh^{-1}s & \text{for } s>1 \end{cases}$$

Form Factor vs. Elliptic Integral (cont.)

The approximations for R_D in terms of the form factor are then given as

$$R_D(Z^2, Y^2, X^2) = \frac{3}{XZ}\frac{1}{X+Y}\left[1 - \xi\left(\frac{Z}{\sqrt{XY}}\right)\right] + O(\varepsilon),$$

$$R_D(Z^2, X^2, Y^2) \approx \frac{3}{YZ}\frac{1}{X+Y}\left[1 - \xi\left(\frac{Z}{\sqrt{XY}}\right)\right] + O(\varepsilon),$$

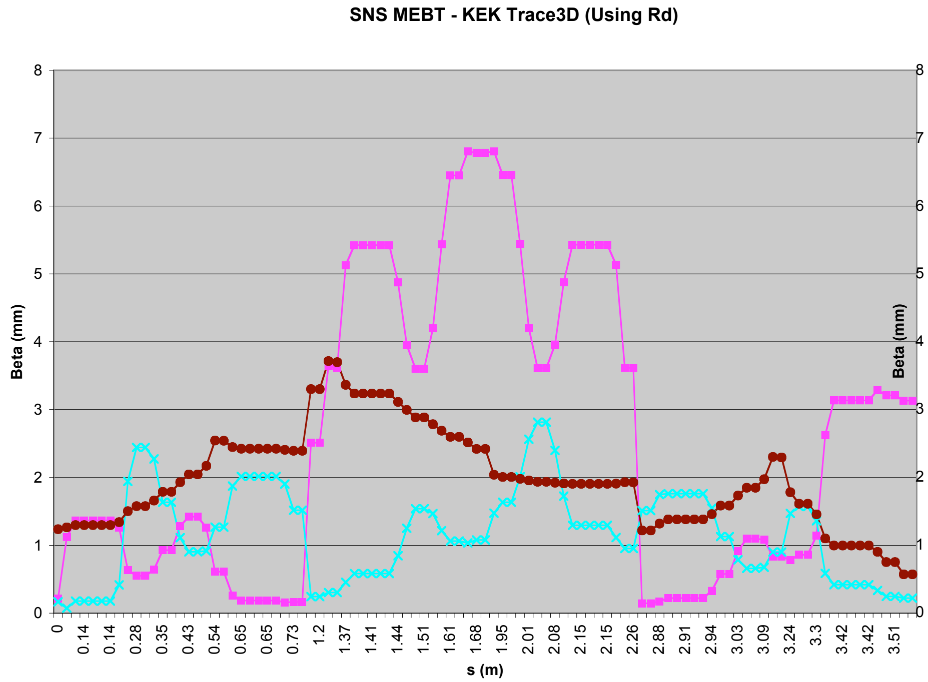
$$R_D(X^2, Y^2, Z^2) \approx \frac{3}{XYZ}\xi\left(\frac{Z}{\sqrt{XY}}\right) + O(\varepsilon^2),$$

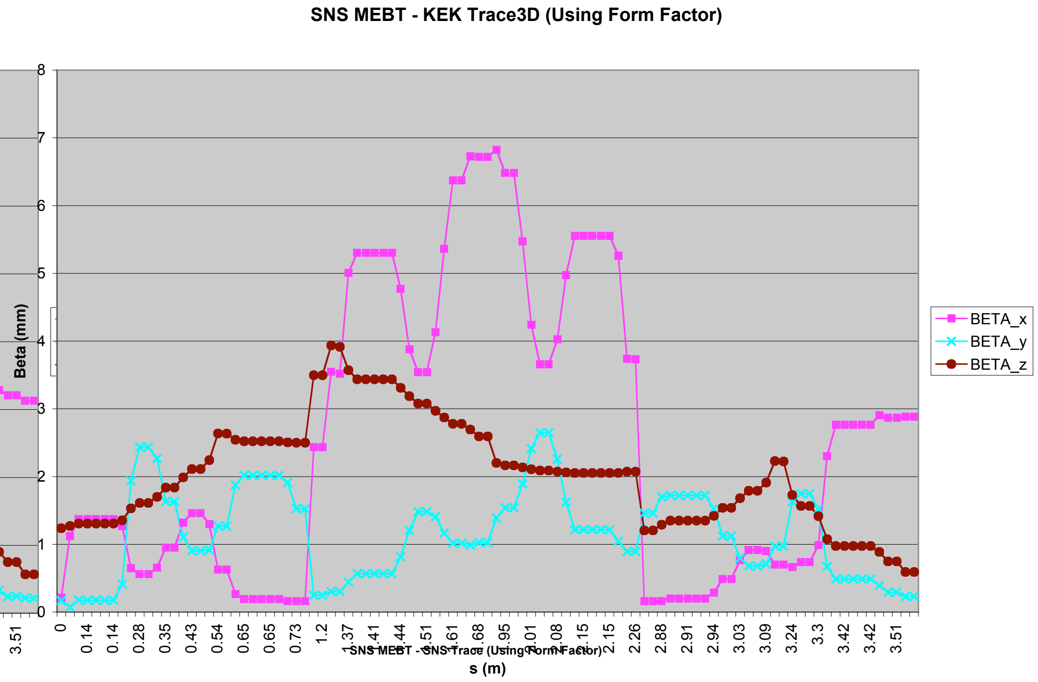
Where X, Y, and Z are the semi-axes of the beam ellipsoid and

$$\varepsilon = (X-Y)/2$$

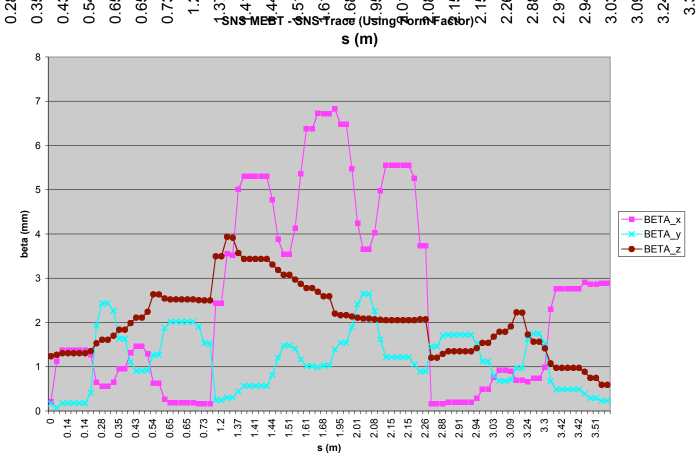
is related to the eccentricity of the transverse ellipse

Comparison of Trace3D elliptic integral and form factor simulations





SNS Version of Trace3D (form fac)



Adaptive Integration Stepping

Approach

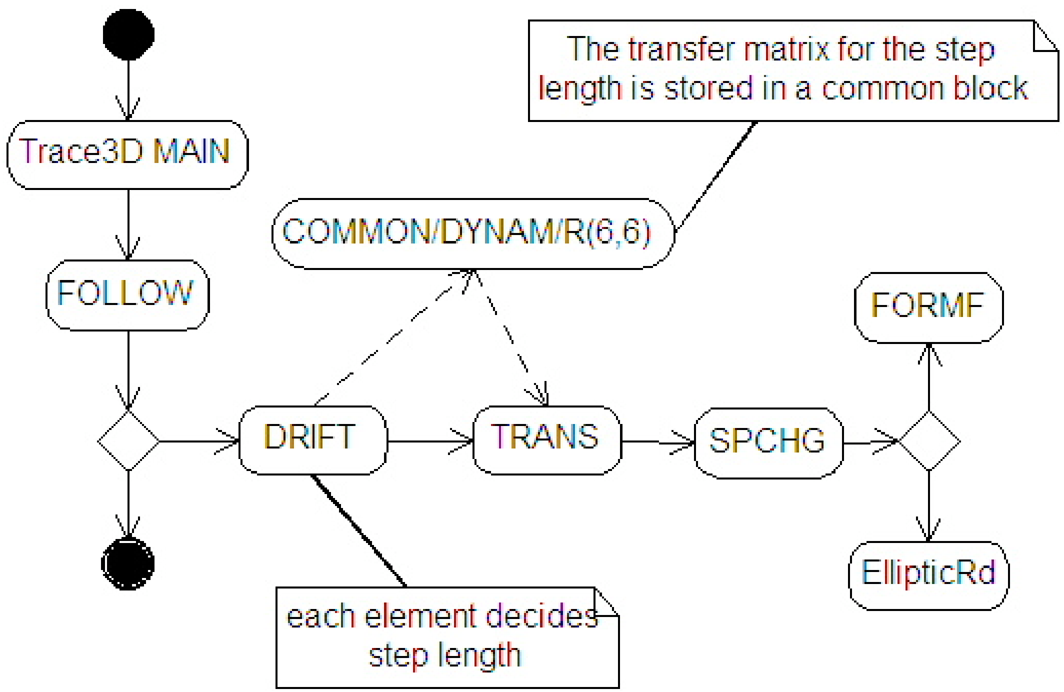
- Form a transfer matrix $\mathbf{\Phi}(s;s_0)$ that includes space effects to second order (2nd order accurate)

- Choose error tolerance ε in the solution ($\sim 10^{-5}$ to 10^{-7})

- Use $\mathbf{\Phi}(s;s_0)$ to propagate $\boldsymbol{\sigma}$ in steps h whose length is determined adaptively to maintain ε

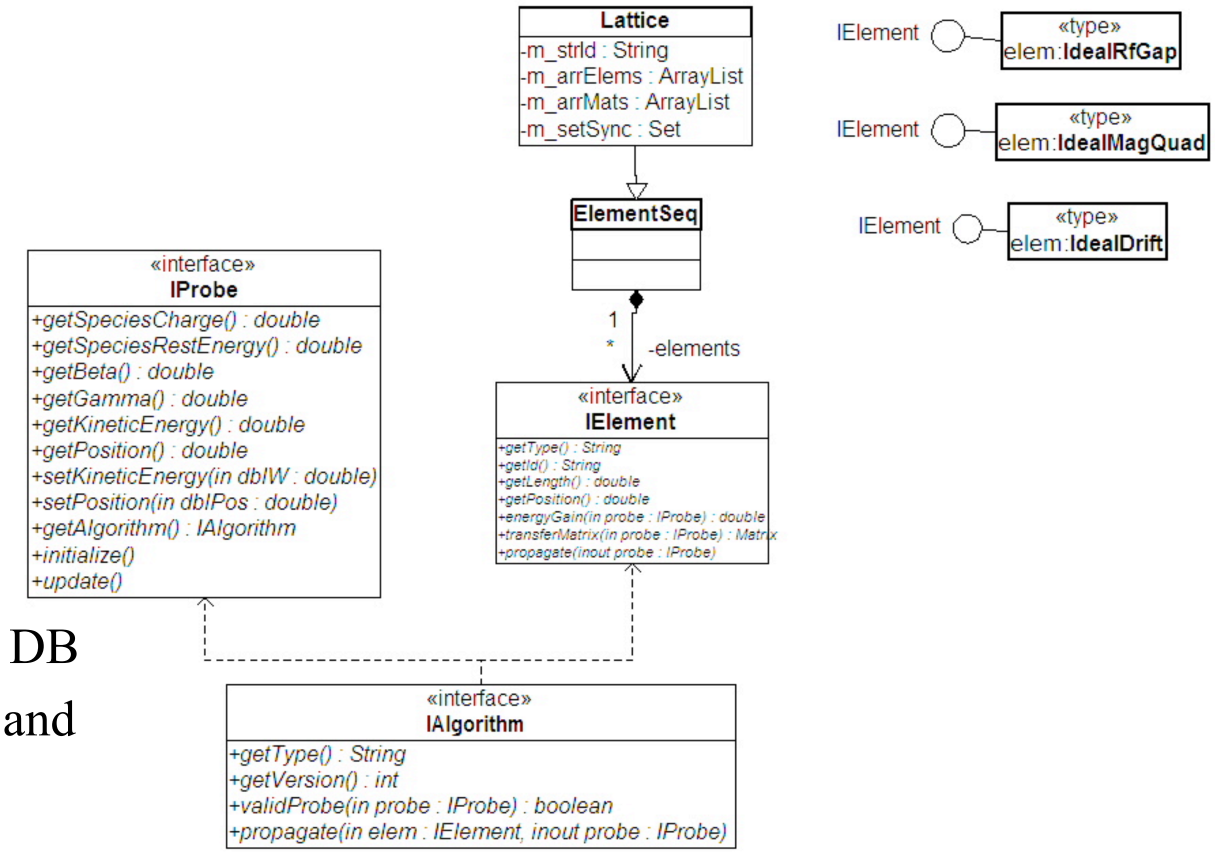
Adaptive Stepping and Trace3D

- Due to Trace3D “architecture” implementing adaptive stepping **may** require a major rewrite
  - Brittle – dangerous
- Implementation possible if it can be done in SUBROUTINE TRANS
  - Compute log(Φ) = Δ*s***A**
  - Compute exp(*h***A**)
  - May be too CPU intensive

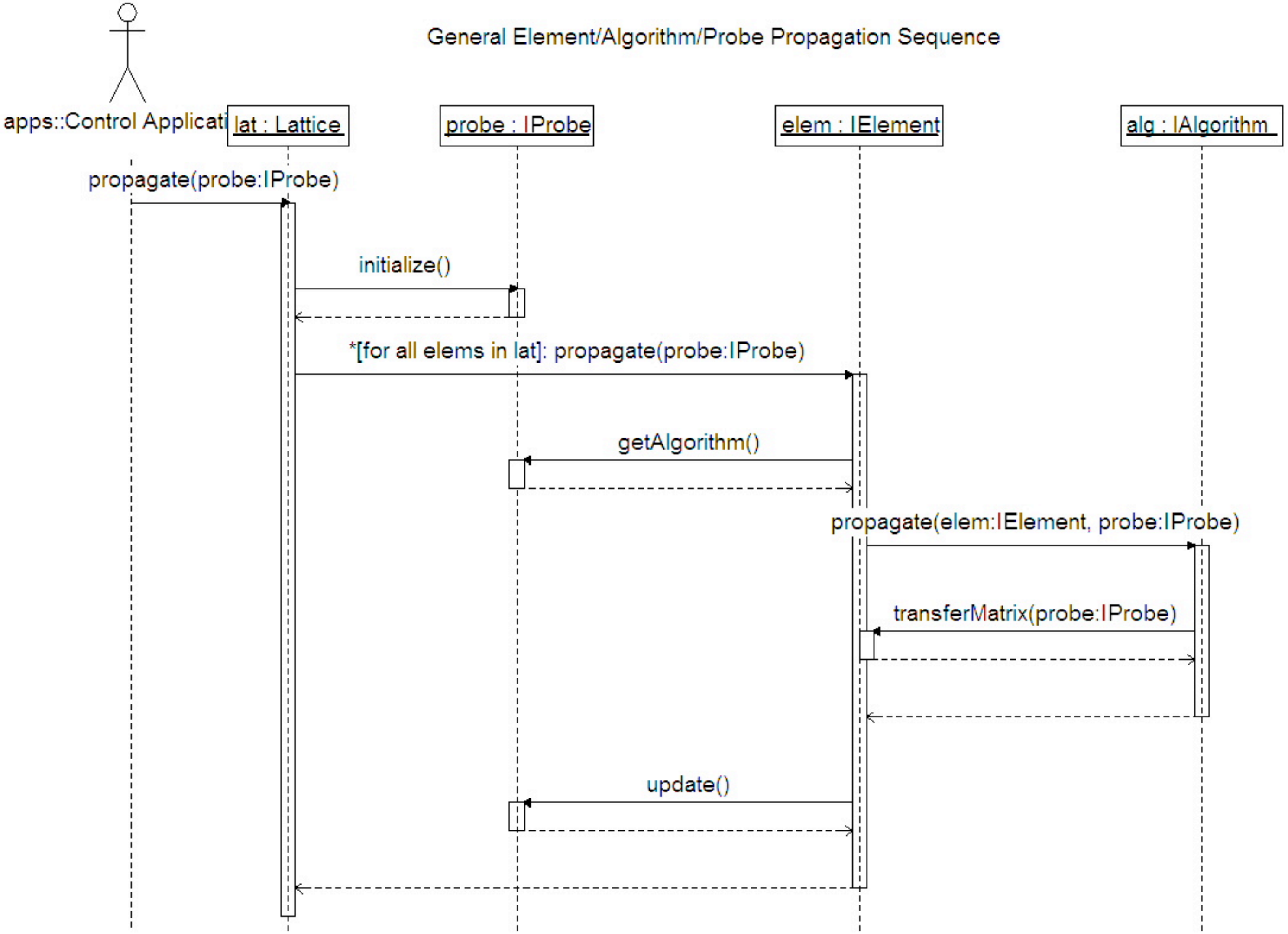


XAL Architecture

- XAL Architecture is modern
  - Not coupled
  - Easier to upgrade
  - Easier to maintain
- XAL Appl. Devel
  - Hard part is setting up DB
  - API is object oriented and documented (easy)
  - New features are (relatively) easily installed



XAL Architecture – Sequence Diagram



Summary

- Adding adaptive stepping to Trace3D may not be worth the effort
  - Ability to implement it in TRANS
  - Compute matrix logarithms and exponentials
- Add space charge to SAD?
  - Unknown effort – inexperienced with SAD
- Add J-PARC features to XAL?
  - Is XAL a player?
  - Very familiar with XAL

4. Space Charge Algorithm (cont.)

Assume that **A** and **B** are constant

- Then full transfer matrix $\mathbf{\Phi}(s;s_0) = e^{(s-s_0)(\mathbf{A}+\mathbf{B})}$

ϒ For practical reasons, we are usually *given* $\mathbf{\Phi_A}$ and $\mathbf{\Phi_B}$

- Beam optics provides $\mathbf{\Phi_A}(s)$
- In ellipsoid coordinates $\mathbf{\Phi_B}(s)$ has simply form because $\mathbf{B}^2 = \mathbf{0}$

$$\mathbf{\Phi_B}(s) = e^{s\mathbf{B}} = \mathbf{I} + s\mathbf{B}$$

Define $\mathbf{\Phi}_{ave}(s) =$ _[$\mathbf{\Phi_A}(s)\mathbf{\Phi_B}(s) + \mathbf{\Phi_B}(s)\mathbf{\Phi_A}(s)$]

- By Taylor expanding $\mathbf{\Phi}(s) = e^{s(\mathbf{A}+\mathbf{B})}$ we find

$$\mathbf{\Phi}(s) = \mathbf{\Phi}_{ave}(s) + O(s^3)$$

That is, $\mathbf{\Phi}_{ave}(s)$ is a second-order accurate approximation of $\mathbf{\Phi}(s)$

4. Space Charge Algorithm (cont.)

- We now have a stepping procedure which is second-order accurate in step length h.

- Now consider the effects of "step doubling"
  - Let $\boldsymbol{\tau}_1(s+2h)$ denote the result of taking one step of length $2h$
  - Let $\boldsymbol{\tau}_2(s+2h)$ denote the result of taking twos steps of length h

$$\boldsymbol{\Delta}(h) \equiv \boldsymbol{\tau}_1(s+2h) - \boldsymbol{\tau}_2(s+2h) = 6\mathbf{c}h^3$$

where the constant $\mathbf{c} = d\boldsymbol{\tau}(s')/ds$ for some $s' \in [s,s+2h]$

- Consider ratio of $|\boldsymbol{\Delta}|$ for steps of differing lengths h_0 and h_1

$$h_1 = h_0 \, [|\boldsymbol{\Delta}_1| / |\boldsymbol{\Delta}_0|]^{1/3}$$

4. Space Charge Algorithm (cont.)

We use the formula $h_1 = h_0\,[|\mathbf{\Delta}_1|/\,|\mathbf{\Delta}_0|]^{1/3}$ as the basis for adaptive step sizing

Given $|\mathbf{\Delta}_1| = \varepsilon$, a prescribed solution residual error we can tolerate

For each iteration k

- Let $|\mathbf{\Delta}_0| = |\,\boldsymbol{\tau}_1(s_k+2h) - \boldsymbol{\tau}_2(s_k+2h)\,|$, the residual error
- Let $h_0 = h_k$ be the step size at iteration k
- Let $h_1 = h_{k+1}$ be the step size at iteration $k+1$

$$h_{k+1} = h_k\,[\varepsilon/|\,\boldsymbol{\tau}_1(s_k+2h) - \boldsymbol{\tau}_2(s_k+2h)\,|]^{1/3}$$

where if $h_{k+1} < h_k$, we must re-compute the k^{th} step using the new steps size h_{k+1} to maintain the same solution accuracy