LBNL-57173
HIFAN 1409

STATUS REPORT ON THE "MERGING" OF THE ELECTRON-CLOUD CODE POSINST WITH THE 3-D ACCELERATOR PIC CODE WARP*

J.-L. Vay#, M. A. Furman, A. W. Azevedo, Lawrence Berkeley National Laboratory, USA
R. H. Cohen, A. Friedman, D. P. Grote, Lawrence Livermore National Laboratory, USA
P. H. Stoltz, Tech-X Corporation, USA

Abstract

We have integrated the electron-cloud code POSINST [1] with WARP [2]–a 3-D parallel Particle-In-Cell accelerator code developed for Heavy Ion Inertial Fusion–so that the two can interoperate. Both codes are run in the same process, communicate through a Python interpreter (already used in WARP), and share certain key arrays (so far, particle positions and velocities). Currently, POSINST provides primary and secondary sources of electrons, beam bunch kicks, a particle mover, and diagnostics. WARP provides the field solvers and diagnostics. Secondary emission routines are provided by the Tech-X package CMEE.

INTRODUCTION

We have integrated the electron-cloud code POSINST with WARP–a 3-D parallel Particle-In-Cell accelerator code developed for Heavy Ion Inertial Fusion (HIF)–so that the two can interoperate. To the combined package, POSINST brings the models related to ECE studies, a mode of operation (thin slice fixed in the laboratory frame with time as the independent variable) adapted to studies of ECE in high energy accelerators, specialized preformatted input and output files, and related routines. WARP brings 3-D/R-Z time-dependent and X-Y z/s-dependent modes of operations with complicated geometries, MAD-like manipulation of accelerator beam lines, parallelism, advanced diagnostics and interactivity through the Python interpreter (already used in WARP) and a graphical user interface. Both codes are run in the same process and communicate through the Python interpreter and share certain key arrays (so far, particle positions and velocities). Currently, POSINST provides initial and secondary sources of electrons, beam bunch kicks, a particle mover, and diagnostics. WARP provides the field solvers and diagnostics. Secondary emission routines are provided by the Tech-X package CMEE, which is based, in turn, on the original secondary electron emission modules in POSINST [3].

DESCRIPTION OF POSINST AND WARP

We provide here an overall description of the codes POSINST and WARP, in their respective states preceding the combination. A summary is given in Table 1.

* Work supported by the US DOE under contract DE-AC03-76SF00098 at UC-LBNL and W-7405-ENG-48 at UC-LLNL
# JLVay@lbl.gov

POSINST

POSINST has been developed for e-cloud studies in High Energy accelerators or storage rings such as:

- APS: e^+ (e^-) short bunches (~1 cm), well-separated (~0.85-100 m, C~1.1 km), intense (N~5×10^{10}), high-energy (E~7 GeV, γ~14,000)
- PSR (see Figure 1): single long proton bunch (~60 m, C=90 m), intense (N~5×10^{13}), low-energy (E ~1.7GeV, γ =1.85)

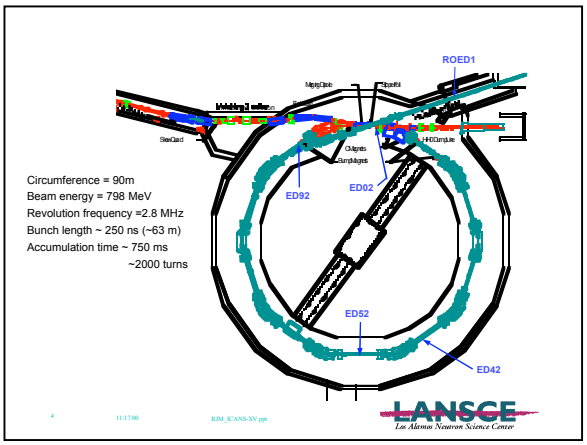

Figure 1: PSR Layout

The object of a POSINST simulation is a thin slice of electrons at a fixed location in the ring subject to their own self-fields and to the field created by particle bunches passing through it. The distribution of electrons is modeled as a collection of macro-particles while the effect of the charged-particle bunches is modeled as a chain of external kicks applied onto the macro-electrons. The possible sources of macro-electrons are (1) photoelectron emission, (2) secondary electron yield (SEY), (3) residual gas ionization, and (4) lost protons hitting the vacuum chamber walls. For each macro-electron, the self-field is computed by summing the contribution of fields from either all the other macro-electrons, or the nodes of a grid on which the electronic charge density has been deposited. The boundary condition is either open or a perfectly conducting pipe (surface charges included) with elliptical or rectangular geometry, and a possible antechamber. POSINST is made of about 9000 lines of code in FORTRAN77 with a few lines of FORTRAN90.

In Fig. 2, the number of macroelectrons as a function of time illustrates the buildup of electrons in PSR after the passage of three bunches, as simulated by POSINST.

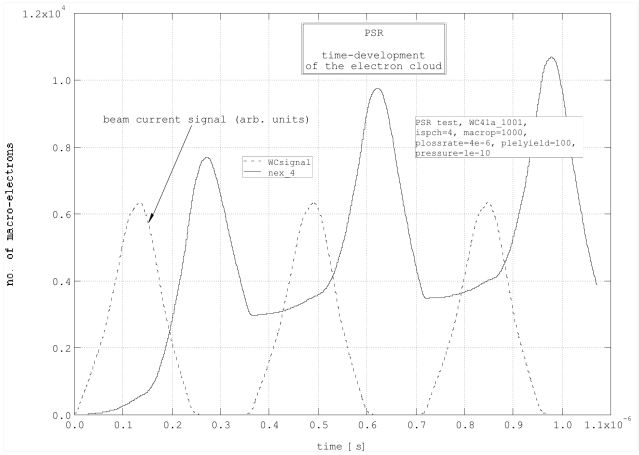

Figure 2: POSINST simulation of PSR

WARP

WARP is a multidimensional intense-beam simulation program being developed and used by the Heavy Ion Fusion Virtual National Laboratory [4], whose goal is to develop heavy-ion accelerators capable of igniting inertial-fusion targets for electric-power production. An artist's representation of a Heavy Ion Fusion power plant is rendered in Figure 3.

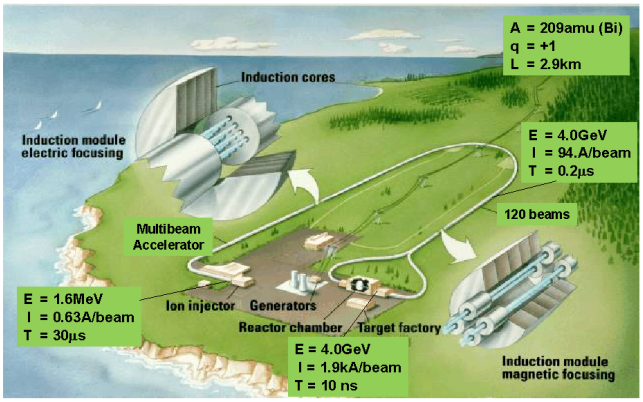

Figure 3: Artist's conception of a Heavy Ion Fusion power plant.

A set of parameters for accelerators currently in service or being considered as steps on the development path toward HIF are:

- HCX: 1 Pt^{+} beam, 180 mA, 1.8 MeV, 4 ms,
- IBX: 1 Pt^{+} beam, 500 mA, 1.7 MeV, 250 ns,
- driver: 120 Bi^{+} beams, 1 A-2 kA, 1.6 MeV-4 GeV, 30 ms-10 ns.

The HCX (High Current eXperiment) currently in operation at Lawrence Berkeley National Laboratory is the first transport experiment using a driver-scale heavy-ion beam. It is designed to address important science questions involving the optimum beam size and the preservation of good beam quality during transport. IBX (Integrated Beam eXperiment) is a possible future experiment to study the integration of beam injection, transport, and focusing, while "driver" is the full-power accelerator needed to ignite the DT capsules.

WARP is being designed and optimized for heavy ion fusion accelerator physics studies. It allows flexible and detailed multi-dimensional modeling of high current beams in a wide range of systems, including bent beam lines using a "warped" coordinate system (from which the code derives its name). At present it incorporates a 3-D description, an axisymmetric R-Z description, a transverse slice X-Y description, a simple envelope model used primarily to obtain a well-matched initial state, and envelope/fluid models used for scoping and design.

The discrete-particle models in WARP combine the particle-in-cell (PIC) technique commonly used for plasma modeling with a description of the "lattice" of accelerator elements. In 3-D and R-Z, WARP is a time-dependent plasma code - the particles are advanced in time and the self and applied fields are applied directly to update the particles' momenta. The calculation can follow the time-dependent evolution of beams, or can efficiently be used to study steady-state beam behavior in 3-D or 2-D R-Z by solving for the self-consistent field only infrequently or by using an iterative method. The transverse-slice model is s-dependent, and is effectively a steady-flow model. The beam can be initially generated from one of several general distributions or from first principles via space-charge-limited injection from an emitting surface. The self-consistent field is assumed electrostatic: Poisson's equation is solved on a Cartesian mesh that moves with the beam. In a bend, the solution is altered to include the curvature of the coordinates. Complex conductor geometry can be included in the field solution using a subgrid-scale, or "cut-cell", boundary algorithm to afford a realistic description of the geometry while minimizing the required grid resolution. Regions where the physics or the geometry require a small spatial scale can be resolved as finely as needed using Adaptive Mesh Refinement, implemented in WARP-R-Z [5]. A 3-D implementation is under development using the Chombo package [6]. In addition, a specialized refinement patch accommodates space-charge-limited injection with very fast rise time; near the emitting surface, the self-fields are calculated along independent one-dimensional lines normal to the surface, with increasing refinement towards the surface [5].

A general set of finite-length, possibly overlapping, accelerator elements can be specified, including quadrupoles, dipoles, accelerating gaps, and elements with arbitrary multipole content, using a MAD-like syntax. Individual elements can be defined and chained together into aggregate elements, which can be further combined. This is done using standard Python syntax (via Python objects), though it looks very similar to MAD syntax. A tool is provided which converts a MAD format file into a WARP-readable file. The fields of the elements can be specified at any of several levels of detail. At the simplest level, the applied fields are axially uniform

within hard-edged regions, and "residence corrections" are used in the particle mover so that the particles receive a correct impulse from each element independent of the number of times they "land" within the element on discrete time steps. At the next level, the fields are expressed as axially dependent multipole components. At the most detailed level, the fields are represented on three-dimensional grids. Electrostatic elements can be included from first principles via inclusion of the conductor geometry as a boundary condition in the solution of the self-fields. Another set of elements in the 3-D and slice models specifies the locations and curvatures of bends. These bends are not physical elements but are the appropriate coordinate transformations needed to follow the beam around the bends. WARP handles particle collisions with any object in the beam line, including the pipe wall, beam source components, diagnostics (Faraday cups, etc.).

WARP is written primarily in standard FORTRAN90 and is steered through a flexible and powerful user interface that uses the scripting language Python (a graphical user interface is also available). The code runs on all Unix-based systems, from a variety of workstations to vector and parallel supercomputers, including the 6000-processor IBM SP supercomputer at NERSC. Although WARP is primarily developed on Linux, it also runs on Windows and Macintosh OS X. Parallelization is implented using domain decomposition and the standard message passing interface (MPI) library. WARP consists of about 115,000 lines of code, distributed in 5,000 lines of descriptive files, 70,000 lines of FORTRAN90 and 40,000 lines of Python.

MOTIVATION FOR THE INTEGRATION

In high-energy physics, the electron cloud effect is a consequence of the strong coupling between the beam and its environment. Many ingredients are needed to properly describe the effect, such as bunch charge and spacing, photoelectric yield, beam energy, photon reflectivity, secondary emission yield, chamber size and geometry, particle loss rate, vacuum pressure, etc. This implies that a 3-D parallel particle code that includes all the elements cited above is ultimately needed, as is self-consistency in some cases.

In heavy ion fusion, there is a strong incentive to fill the pipe as completely as possible. This results in stronger interactions between the beam and the pipe, leading to a higher probability of electron cloud effects and gas desorption. An illustration is given in Figure 4 where the beam head is mismatched and particles hit the ESQ quadrupoles. Until now, the secondary electrons which would result from such an interaction of particles with structures were not modeled in WARP. Because there is the possibility of feedback between the beam bunch and the electron cloud, a self-consistent description is ultimately needed. The plan for achieving such a self-consistent model in WARP is given in Figure 5.

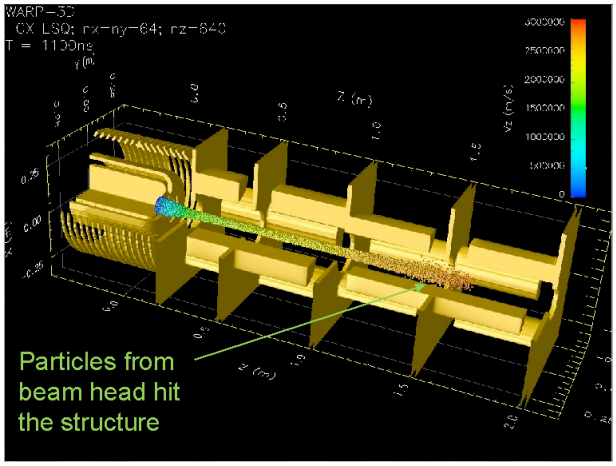

Figure 4: 3-D WARP simulation of the High Current Experiment (HCX) injector. Particles from the mismatched head hit the ESQ quadrupoles.

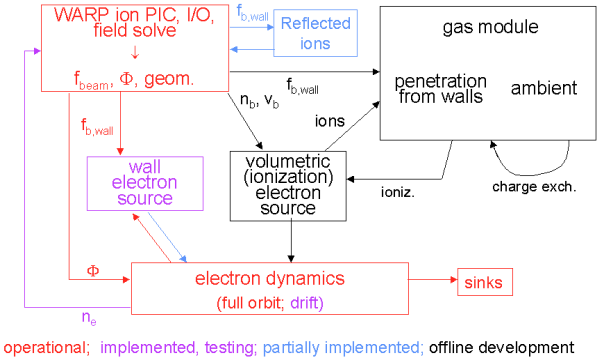

Figure 5: Plan for self-consistent modeling of electron cloud and ion beam with WARP.

In each case, it is clear from the description of POSINST and WARP given in the previous section that each code can provide functionalities required by the other.

THE INTEGRATION PROCESS

The following modifications were made to POSINST:

1. in collaboration with Tech-X, the SEY routines were extracted and packaged into the stand-alone library CMEE, distributed by Tech-X [7]. In the process, the calls to IMSL routines were replaced with calls to routines from open-source mathematical libraries (included in CMEE),
2. POSINST was modified for compatibility with the Python wrapper FORTHON [8]. In the process, FORTRAN77 common blocks were translated into descriptive files. Once parsed by FORTHON, the descriptive files are converted into FORTRAN90 modules and C routines which link the variables defined in the

modules (including FORTRAN90 derived types) to Python objects,

3. the main subroutines (controlling the execution at a high level) were translated to Python, giving full control of POSINST from Python,
4. the particle data structure was modified to match WARP's data structure.

The modifications 2-4 opened the way for very flexible communications between WARP and POSINST. It is worth underlining that this is not a merge; each code is a separate entity that stands on its own. In practice, two Python packages (one for WARP, one for POSINST) exist and can be invoked in Python separately or concurrently. For a run which requires capabilities from both packages, both WARP and POSINST variables are instantiated at the Python level. Since Python can handle separate name-spaces, WARP and POSINST variables coexist without conflict or confusion, even though some variables might have the same name in FORTRAN. At the Python level, the discrimination comes from a prefix added to the variable name. For example, the FORTRAN variable x (particle position along axis x in POSINST) is accessed through 'pos.x' in Python. The FORTRAN variable xp (particle position along axis x in WARP) is accessed in Python through 'top.xp' (WARP is made of several packages, including the package 'top'). Although separate name spaces are desirable for concurrent access to multiple packages, it is sometimes advantageous to have variables declared in two different packages sharing the same memory location. This is the case for the variables x (from POSINST) and xp (from WARP), given in the preceding example. Although declared in two separate packages, they describe the same quantity and should point to the same memory location. It turns out to be trivial at the Python level if the packages have been created using FORTHON, since the Python line 'pos.x=top.xp' suffices to ensure that both variables share the same memory location.

At present, POSINST provides the main input deck, the main control loop, the initial and secondary sources of electrons, the beam-bunch kicks, the particle mover, and diagnostics. WARP provides the field solvers and additional diagnostics. Secondary emission routines are provided by the Tech-X package CMEE.

Finally, a specialized page was added to the main notebook in WARP's graphical user interface. It allows the user to run POSINST interactively. A typical run using the graphical user interface follows the following steps:

1. select an input deck,
2. select the level of verbosity and whether to use the POSINST (default for now) or the WARP routines for field solution,
3. run,
4. plot data already in memory using WARP diagnostic GUI pages or commands entered at the Python prompt,
5. load the data saved in files into memory and plot.

Figures 6 and 7 illustrate the use of the GUI and plotting of data. The data saved into files can also be accessed later for post-processing using the GUI or other software.

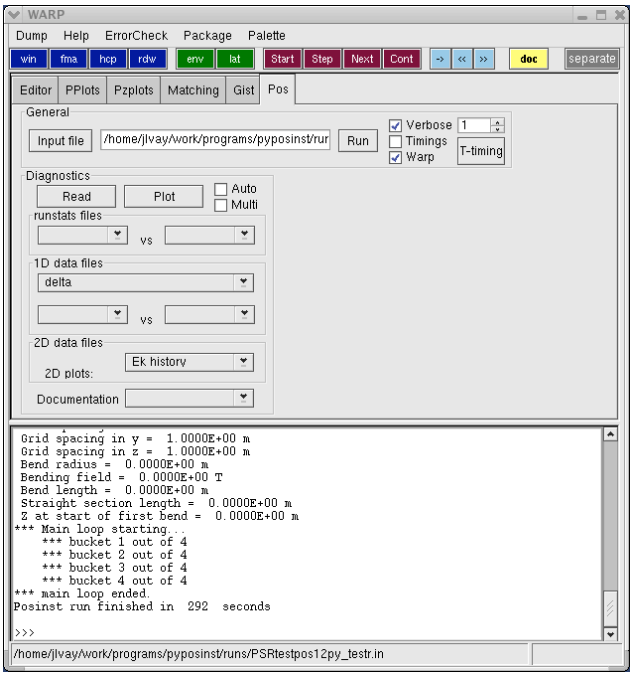

Figure 6: Example of POSINST run using WARP's graphical user interface. Electrons dynamics was simulated during the passage of four buckets in PSR.

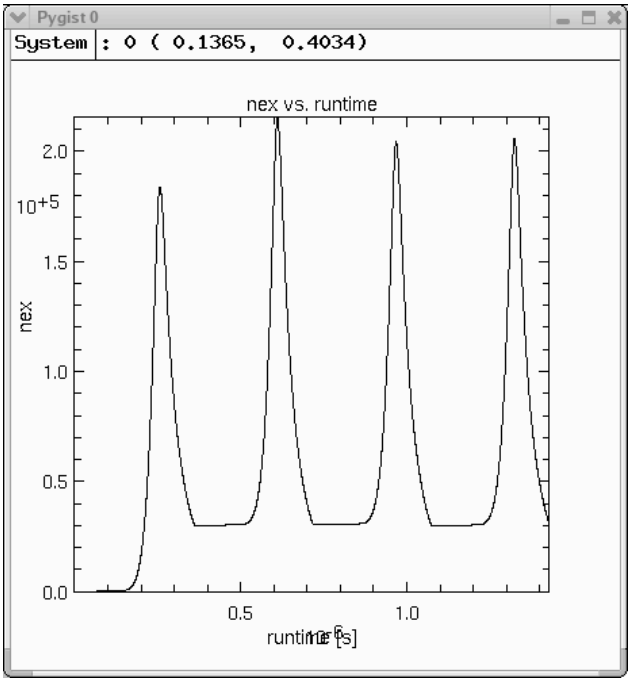

Figure 7: Electronic density as a function of time (PSR).

CONCLUSION

We have successfully modified the code POSINST so that it can cooperate with WARP through the scripting language Python. Although each code keeps its identity and can run "standalone," they can now run in symbiosis to solve problems that are beyond the reach of each code alone. Taking advantage of multiple name-spaces in Python and features of the FORTRAN90-to-Python wrapper FORTHON, efficiency was achieved by sharing the particle data memory locations between the two packages, without any added complexity. The combination provides the third dimension and parallelism to POSINST and secondary electrons physics to WARP, enabling the combined package to fulfill missions in High Energy Physics, Heavy Ion Inertial Fusion, and other fields.

REFERENCES

[1] Pivi MTF, Furman MA, ``Electron cloud development in the Proton Storage Ring and in the Spallation Neutron Source", Phys. Rev. STAB, vol.6, no.3, March 2003.

[2] D. P. Grote, A. Friedman, I. Haber, "Methods used in WARP3d, a Three-Dimensional PIC/Accelerator Code", Proc. of the 1996 Comp. Accel. Physics Conf., AIP Conference Proceedings 391, p. 51 (1996).

[3] M. A. Furman and M. T. F. Pivi, "Probabilistic Model for the Simulation of Secondary Electron Emission", PRSTAB/v5/i12/ e124404 (2003).

[4] http://hif.lbl.gov

[5] Vay JL., Colella P., Kwan JW., McCorquodale P., Serafini DB., Friedman A., Grote DP., Westenskow G., Adam JC., Heron A., Haber I., "Application of adaptive mesh refinement to particle-in-cell simulations of plasmas and beams", *Phys. of Plasmas*, **11**(5), 2928-2934, 2004.

[6] P. McCorquodale, P. Colella, D. P. Grote, J.-L. Vay, "A Node-Centered Local Refinement Algorithm for Poisson's Equation in Complex Geometries", *J. Comput. Phys.*, in press

[7] http://www.txcorp.com/technologies/CMEE

[8] For more information, contact D. P. Grote at DPGrote@lbl.gov.

Table 1. Summary of POSINST and WARP functionalities

Functionality	POSINST	WARP
Particles	• Various with pre-defined attributes, or generic particle (given q, m); x, y, β_x, β_y, β_z (β=v/c)	• q, m, weight; x, y, z, u_x, u_y, u_z (u=γv) + user- (or pre-) defined attributes
Self-fields	• electrostatics on regular cartesian grid - 2-D XY, 1-D R • perfect-conductor B.C.s (surface charges included) - elliptical or rectangular vacuum chamber geometry, with a possible antechamber	• electrostatic in moving window on regular cartesian grids - 3-D XYZ warped coordinates, 2-D XY, 2-D R-Z, 1-D R, 1-D Z • AMR available for R-Z, in development for 3D with CHOMBO • images from embedded conductors through "cut-cell" method
External Fields	• beam bunches - multi-bunch passages - bunch divided longitudinally into N_k kicks (2D, purely transverse) • certain magnetic elements	• lattice elements - sharp-edged, multipole expansion, data or first-principles on 3-D grid - MAD-like description of lattice (MAD-to-WARP translator available) ▪ pre-defined elements: dipoles, accelerating gaps, quadrupoles, sextupoles, … ▪ user-defined elements using box, sphere, cylinder, cone, torus, … primitives
Independent variable	• t: evolution of thin slice at fixed station	• s: progression of thin slice along propagation axis in steady flow • t: evolution of thick (3-D window) or thin slice at fixed or moving station; 3 modes: - pure time-dependent ($\Delta t_{fieldsolve}=\Delta t_{particles}$) - quasi time-dependent ($\Delta t_{fieldsolve}>\Delta t_{particles}$) - "steady-state" (converges to steady-flow solution iteratively)
Particle generation	• photoelectron emission (very simplified input model) • secondary electron yield (SEY) (detailed model included) • residual gas ionization • lost protons hitting the vacuum chamber walls	• beam loading - predefined KV, semi-gaussian, user-defined function, … - list from previous run or data (can be time-dependent) • beam injection - Child-Langmuir, Gauss pill-box, fixed current - from flat or curved surface with special mesh refinement patch for fast rise-time
Diagnostics	• histories of electron density, energy, % loss, energy loss, angle loss, ionization, SEY, … • visualization with third party software	• fields, particles, lattice, moments, histories, user-defined, dumps, • 2-D line, contours, scatter plots, 3-D surface through Gist/OpenDX
Cross-platform	• ~9000 lines of FORTRAN77 (a few in F90) • uses IMSL	• uses D.P. Grote F90+Python wrapper FORTHON - ~5000lines of descriptive files, ~70000 lines of F90, ~40000 lines of Python
Parallel	• N/A	• MPI, 1-D decomposition in Z (different for particles and fields)
Interpreter	• N/A	• Python - provides interactivity, extensibility, steering - access to a huge collection of freely available third party libraries and softwares
GUI	• N/A	• interactive plotting, Python-smart editor, step-by-step running • user-expandable
Others	• N/A	• envelope/fluid equation solvers