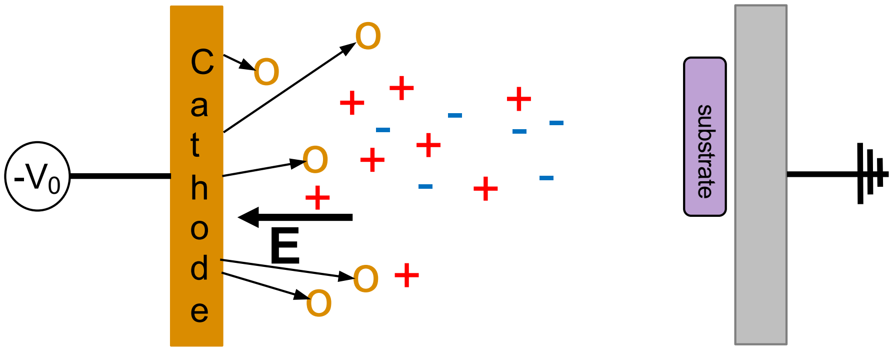
Text-based Setup of Electrostatic Simulations in VSim

TECH-X
SIMULATIONS EMPOWERING YOUR INNOVATIONS



Tom Jenkins

Tech-X Corporation

Tech-X Worldwide Simulation Summit
Boulder, Colorado
September 17, 2019

A brief introduction to me…

- Senior Research Scientist
- 9.5 years at Tech-X
- Ph.D. @ Princeton/PPPL (2007), in gyrokinetic PIC simulation
- Postdoc @ UW-Madison, working on RF/MHD coupling for electron cyclotron current drive in fusion plasmas
- Current research interests:
  - methods for speeding up particle-in-cell simulations (SLPIC)
  - modeling RF sheaths/impurity sputtering in fusion devices
  - kinetic theory – wave/particle interactions, etc.
  - PIC modeling of low-temperature plasmas
- Website, where this talk and other talks/papers/presentations are posted:

http://nucleus.txcorp.com/~tgjenkins

Standard electrostatics problem: Poisson

$$\frac{d^2\phi(x)}{dx^2} = -\frac{\rho(x)}{\epsilon_0} \quad ; \quad \phi(x=0) = \phi^{left}, \qquad \phi(x=L) = \phi^{right} \, ; x \in [0, L]$$

Numerical approach: discretize.

Define a grid: $\Delta x = \dfrac{L}{N} \quad ; \quad x_n = n\Delta x \;\; \forall \;\; n = 0,1,\dots,N$

Use finite-difference approximation to second derivative, at interior gridpoints:

$$-\epsilon_0 \left[\frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{\Delta x^2}\right] = \rho_j \qquad \forall \quad j = 1,2,\dots,N-1$$

Apply boundary conditions, at edge gridpoints:

$$\phi_0 = \phi^{left}$$
$$\phi_N = \phi^{right}$$

Solve the ensuing system of linear equations.

Solution error scales as $1/N^2$

$$\frac{d^2\phi(x)}{dx^2} = -\frac{\rho_0 \sin\left(\frac{\pi x}{L}\right)}{\epsilon_0} \quad ; \quad \phi(x=0) = \phi^{left}, \qquad \phi(x=L) = \phi^{right} \quad on\ [0,L]$$

has exact solution

$$\phi(x) = \phi^{left} + \left(\phi^{right} - \phi^{left}\right)\frac{x}{L} + \frac{\rho_0 L^2}{\epsilon_0 \pi^2}\sin\left(\frac{\pi x}{L}\right)$$

On the discrete grid, we have

$$\phi_j^{exact} = \phi^{left} + \left(\phi^{right} - \phi^{left}\right)\frac{j}{N} + \frac{\rho_0 L^2}{\epsilon_0 \pi^2}\sin\left(\frac{\pi j}{N}\right) \quad ; \quad \rho_j^{exact} = \rho_0 \sin\left(\frac{\pi j}{N}\right)$$

Putting these functions into the discretized Poisson equation yields

$$-\frac{\rho_0}{\epsilon_0}\sin\left(\frac{\pi j}{N}\right)\left\{\frac{2N^2}{\pi^2}\left[1 - \cos\left(\frac{\pi}{N}\right)\right]\right\} \approx -\frac{\rho_0}{\epsilon_0}\sin\left(\frac{\pi j}{N}\right)$$

$$\left\{\frac{2N^2}{\pi^2}\left[1 - \left(1 - \frac{\pi^2}{2N^2} + \frac{\pi^4}{24N^4} + \cdots\right)\right]\right\} \approx 1$$

What does this look like in VSim?

Let's set up a basic simulation and run it for one step:

Parameters
VLEFT = 0
VRIGHT = 1
LX = 1
NX = 10
RHOZERO = 20

Basic Settings
number of steps = 1
steps between dumps = 1
dimensionality = 1
field solver = electrostatic

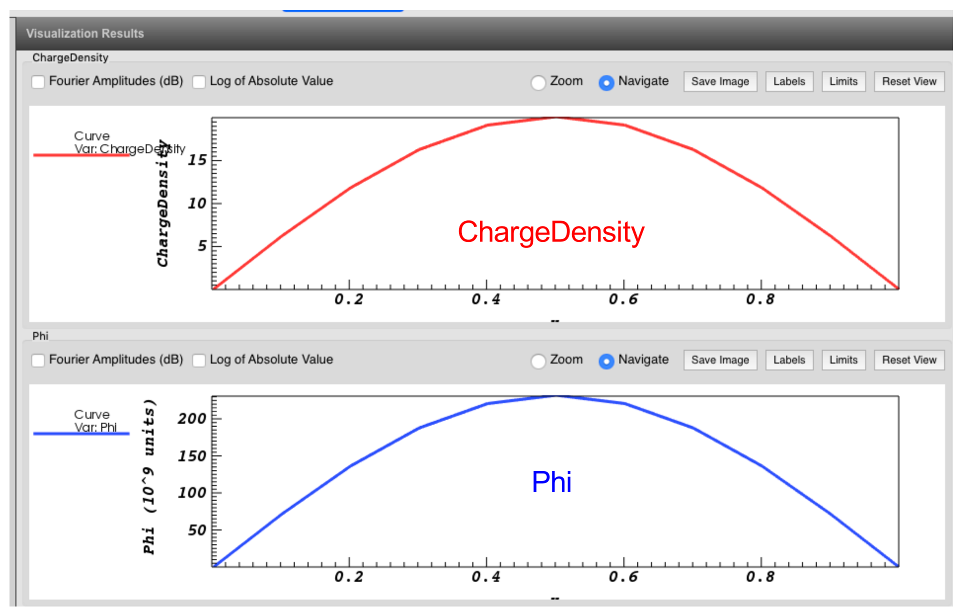
SpaceTimeFunctions
RHOxt=RHOZERO*sin(PI*x/LX)

Grids
xMin = 0
xMax = LX
xCells = NX

Field Dynamics: Fields
Background Charge Density RHO=RHOxt

Field Dynamics: FieldBoundaryConditions
Dirichlet on lower x: VLEFT
Dirichlet on upper x: VRIGHT

Field Dynamics: PoissonSolver
preconditioner = no preconditioner
solver = SuperLU



Looking at vsim.in – input blocks

FRONTMATTER

```
<Grid globalGrid>
 …
</Grid>

<Decomp decomp>
 …
</Decomp>

<MultiField NAME_OF_MULTIFIELD>

  <Field NAME_OF_FIELD>
   …
  </Field>

  <FieldUpdater NAME_OF_FIELDUPDATER>
   …
  </FieldUpdater>

  <InitialUpdateStep NAME_OF_INITIALUPDATESTEP>
   …
  </InitialUpdateStep>

  <UpdateStep NAME_OF_UPDATESTEP>
   …
  </UpdateStep>

  updateStepOrder = [NAME_OF_UPDATESTEP_1 NAME_OF_UPDATESTEP2 ...]
</MultiField>
```

Key VSim concept 0: block structures

Or very generally,

```
<OBJECT objectName>
    …
  object features
    …
</OBJECT>
```

Looking at vsim.in – overall structure

Key VSim concept 1: the MultiField block

```
FRONTMATTER

<Grid globalGrid>
  …
</Grid>

<Decomp decomp>
  …
</Decomp>

<MultiField NAME_OF_MULTIFIELD>

  <Field NAME_OF_FIELD>
   …
  </Field>

  <FieldUpdater NAME_OF_FIELDUPDATER>
   …
  </FieldUpdater>

  <InitialUpdateStep NAME_OF_INITIALUPDATESTEP>
   …
  </InitialUpdateStep>

  <UpdateStep NAME_OF_UPDATESTEP>
   …
  </UpdateStep>

  updateStepOrder = [NAME_OF_UPDATESTEP_1 NAME_OF_UPDATESTEP2 ...]
</MultiField>
```

- Field: functions: electric field, electrostatic potential, charge density, …
- FieldUpdater: define mathematical operations on existing fields, e.g. taking a gradient of a scalar field
- InitialUpdateStep: Set initial conditions – done only once at simulation outset
- UpdateStep: Call the previously defined FieldUpdaters

Looking at vsim.in - frontmatter

```
nsteps = 1
dumpPeriodicity = 1
dt = 1.0
dimension = 1
floattype = double
verbosity = 127
copyHistoryAtEachDump = 0
useGridBndryRestore = False
constructUniverse = False
```

number of steps in simulation (nsteps)

write data every 1 timestep (dumpPeriodicity)

timestep (dt)

1D simulation (dimension)

```
<Grid globalGrid>
verbosity = 127
numCells = [10 11 12]
lengths = [1.0 1.0 1.0]
startPositions = [0.0 -0.5 0.0]
maxCellXings = 1
</Grid>
```

3D grid: default y, z values

$\Delta x = 1/10$; $\Delta y = 1/11$; $\Delta z = 1/12$
(extra dimensions not used in computation; still present in several parts of input file though)

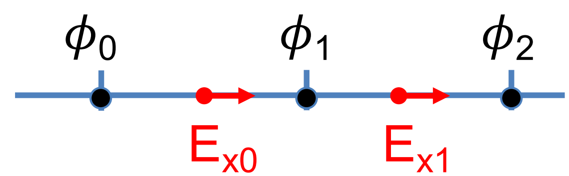
Looking at vsim.in – Field blocks

```
<Field E>
  numComponents = 3
  offset = edge
  kind = regular
  overlap = [1 1]
  labels = [E_x   E_y   E_z]
</Field>
```

- numComponents = 3: vector field
- offset = edge: lives on edges between grid points
- labels: names of field components in output file

```
<Field Phi>
  numComponents = 1
  offset = none
  kind = regular
  overlap = [1 2]
  labels = [Phi]
</Field>
```

- numComponents = 1: scalar field
- offset = none: lives on grid points
- kind, overlap: messaging instructions (for computing in parallel): ordinary field update




```
<Field ChargeDensity>
  numComponents = 1
  offset = none
  kind = depField
  overlap = [1 2]
  labels = [ChargeDensity]
</Field>
```

- numComponents = 1: scalar field
- offset = none: lives on grid points
- kind, overlap: messaging instructions (for computing in parallel): include data from guard cells

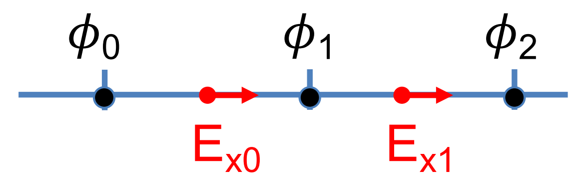
Looking at vsim.in – FieldUpdater blocks

built-in operation that computes the gradient of a scalar

```
<FieldUpdater gradPhi>
  kind = gradVecUpdater
  factor = -1.0
  lowerBounds = [0  0  0]          (inclusive)
  upperBounds = [10   11   12]     (exclusive)
  readFields = [Phi]
  writeFields = [E]
</FieldUpdater>
```

in other words,

$$\vec{E} = -\vec{\nabla}\phi.$$



names of previously defined Field blocks:
scalar input, vector output for this FieldUpdater kind.

```
<FieldUpdater RHO>
  kind = STFuncUpdater
  operation = add
  lowerBounds = [0  0  0]          (inclusive)
  upperBounds = [11  12  13]       (exclusive)
  writeFields = [ChargeDensity]
  component = 0
  cellsToUpdateAboveDomain = [False  False  False]
  <STFunc f>
    kind = expression
    expression = (20.0*sin(3.141592653589793*x/1.0))
  </STFunc>
</FieldUpdater>
```

operation: adds (subtracts, multiplies, etc.) the specified STFunc to the specified writeField

component: scalar

expression: $= \rho_0 \sin\left(\frac{\pi x}{L}\right)$

Looking at vsim.in – InitialUpdateStep blocks

These updates are performed only once, at the simulation outset.

```
<InitialUpdateStep RHOInitStep>
  alsoAfterRestore = True
  updaters = [RHO]
  messageFields = []
</InitialUpdateStep>
```

- alsoAfterRestore: Also do this step when restarting a simulation
- updaters = [RHO]: Previously defined field updater, defines rho field

```
<InitialUpdateStep esSolveInitStep>
  alsoAfterRestore = True
  updaters = [esSolve]
  messageFields = [Phi]
</InitialUpdateStep>
```

- updaters = [esSolve]: Previously defined field updater, solves Poisson equation for phi field

```
<InitialUpdateStep gradPhiInitStep>
  alsoAfterRestore = True
  updaters = [gradPhi]
  messageFields = [E]
</InitialUpdateStep>
```

- updaters = [gradPhi]: Previously defined field updater, computes E from phi.

Looking at vsim.in – UpdateStep blocks

These updates are performed at every timestep in the simulation.

```
<UpdateStep RHOStep>
  toDtFrac = 1.0
  updaters = [RHO]
  messageFields = []
</UpdateStep>
```

Advance to next full timestep (toDtFrac = 1.0)

Previously defined field updater, defines rho field (just as in InitialUpdateStep call) (updaters = [RHO])

```
<UpdateStep esSolveStep>
  toDtFrac = 1.0
  updaters = [esSolve]
  messageFields = [Phi]
</UpdateStep>
```

Previously defined field updater, solves Poisson equation for phi field (just as in InitialUpdateStep call) (updaters = [esSolve])

```
<UpdateStep gradPhiStep>
  toDtFrac = 1.0
  updaters = [gradPhi]
  messageFields = [E]
</UpdateStep>
…
updateStepOrder = [RHOStep esSolveStep gradPhiStep]
```

Previously defined field updater, computes E from phi (just as in InitialUpdateStep call). (updaters = [gradPhi])

UpdateSteps can appear in the input file in any order you like, the updateStepOrder determines which ones will be called when.

Regroup and Review

So far, we have:

-built an .sdf file in VSim that solves the 1D Poisson equation

-found the .in file that VSim built from our initial .sdf file

-looked at the general block structure of that .in file

-looked at some typical Field, FieldUpdater, InitialUpdateStep, and UpdateStep blocks that live in the larger MultiField block

Now, we'll do a bit of a deeper dive into how VSim solves the Poisson equation, and learn a bit more about how data is organized 'under the hood' in VSim.

Electrostatic solves, without VSim

VSim solves the Poisson equation

$$\frac{d^2\phi(x)}{dx^2} = -\frac{\rho(x)}{\epsilon_0} \quad ; \quad \phi(x=0) = \phi^{left}, \qquad \phi(x=L) = \phi^{right}\,; x \in [0, L]$$

with Fields and FieldUpdaters and UpdateSteps.

Let's first build a discretized version of this problem "by hand", to see what kinds of things we might expect VSim to be doing:

Grid:

$$\Delta x = \frac{L}{N} \quad ; \quad x_n = n\Delta x \quad \forall \; n = 0,1,\dots,N$$

Discrete Poisson equation:

$$-\epsilon_0 \left[\frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{\Delta x^2}\right] = \rho_j \qquad \forall \quad j = 1,2,\dots,N-1$$

Boundary conditions:

$$\phi_0 = \phi^{left}$$
$$\phi_N = \phi^{right}$$

Constructing the matrix – interior points

$$-\epsilon_0\left[\frac{\phi_{j+1}-2\phi_j+\phi_{j-1}}{\Delta x^2}\right]=\rho_j \qquad \forall \quad j=1,2,\ldots,N-1$$

becomes

$$\left(-\frac{\epsilon_0}{\Delta x^2}\right)\begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}\begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{j-1} \\ \phi_j \\ \phi_{j+1} \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \\ \phi_N \end{bmatrix}=\begin{bmatrix} \rho_0 \\ \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{j-1} \\ \rho_j \\ \rho_{j+1} \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \\ \rho_N \end{bmatrix}$$

Invalid for first/last rows of matrix. Instead, use boundary conditions there.

Constructing the matrix – boundary conditions

$$\phi_0 = \phi^{left}$$
$$\phi_N = \phi^{right}$$

becomes

$$\left(\frac{-\epsilon_0}{\Delta x^2}\right)\begin{bmatrix} -\gamma\Delta x^2/\epsilon_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu\Delta x^2/\epsilon_0 \end{bmatrix}\begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{j-1} \\ \phi_j \\ \phi_{j+1} \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} \gamma\phi^{left} \\ \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{j-1} \\ \rho_j \\ \rho_{j+1} \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \\ \mu\phi^{right} \end{bmatrix}$$

necessitating a change in the right-hand side vector.

Rescaling factors γ, μ may be used to adjust matrix condition number.

Canonical form: Ax = b.

linearSolveUpdater – solving the Poisson equation

Now let's look at how this is done in the vsim.in file.

One of VSim's built-in FieldUpdater blocks is the linearSolveUpdater, which solves equations of the form Ax = b.

Looking at vsim.in – linearSolveUpdater

```
<FieldUpdater esSolve>
kind = linearSolveUpdater
lowerBounds = [0]  (inclusive)
upperBounds = [11] (exclusive)
readFields = [ChargeDensity]
readComponents = [0]
writeFields = [Phi]
writeComponents = [0]
writeEquationToFile = 0

<MatrixFiller interiorFiller>
kind = stFuncStencilFiller
verbosity = 127
minDim = 1
lowerBounds = [1 1 1]      (inclusive)
upperBounds = [10 11 12]   (exclusive)
component = 0

<STFunc coeff>
kind = expression
expression = -8.854187817591624e-12
</STFunc>
```

1D linear solve

Input: scalar ρ

Output: scalar ϕ

Can use this to look at the matrix (we will do this in a moment)

3D matrix template (even though we only need 1D)

$= -\epsilon_0$

MatrixFiller blocks do just what they sound like – filling rows in the matrix.

linearSolveUpdater - StencilElements

Inside the MatrixFiller block, we have various StencilElements:

```
<STFuncStencilElement phi_dxp>
value = -100.0
minDim = 1
cellOffset = [0 0 0]
functionOffset = [0.5 0. 0.]
rowFieldIndex = 0
columnFieldIndex = 0
</STFuncStencilElement>
```

No offset

```
<STFuncStencilElement phi_npx>
value = 100.0
minDim = 1
cellOffset = [1 0 0]
functionOffset = [0.5 0. 0.]
rowFieldIndex = 0
columnFieldIndex = 0
</STFuncStencilElement>
```

+1 cell

$-1/\Delta x^2$

$1/\Delta x^2$

functionOffset is irrelevant for node-centered fields

```
<STFuncStencilElement phi_dxm>
value = -100.0
minDim = 1
cellOffset = [0 0 0]
functionOffset = [-0.5 0. 0.]
rowFieldIndex = 0
columnFieldIndex = 0
</STFuncStencilElement>
```

No offset

```
<STFuncStencilElement phi_nmx>
value = 100.0
minDim = 1
cellOffset = [-1 0 0]
functionOffset = [-0.5 0. 0.]
rowFieldIndex = 0
columnFieldIndex = 0
</STFuncStencilElement>
```

-1 cell

A generic interior row in the 1D Poisson matrix is

$$coeff \cdot [\cdots \quad 0 \quad phi_{nmx} \quad (phi_{dxm}+phi_{dxp}) \quad phi_{npx} \quad 0 \quad \cdots]$$

linearSolveUpdater – boundary conditions

LHS (matrix)

```
<MatrixFiller RIGHTBCFiller>
kind = stencilFiller
verbosity = 127
minDim = 1
lowerBounds = [10 0 0]
upperBounds = [11 12 13]
component = 0

<StencilElement ident>
value = 1.7708375635183248e-09
minDim = 0
cellOffset = [0 0 0]
rowFieldIndex = 0
columnFieldIndex = 0
</StencilElement>
</MatrixFiller>
```

Only rightmost cell

$= 2\epsilon_0/\Delta x^2$ (this is the μ factor from the earlier slide, on the LHS)

RHS (vector)

```
<VectorWriter RIGHTBCWriter>
kind = stFuncVectorWriter
verbosity = 127
minDim = 1
lowerBounds = [10 0 0]
upperBounds = [11 12 13]
component = 0

<STFunc function>
kind = expression
expression = 1.0
</STFunc>

scaling = 1.7708375635183248e-09
</VectorWriter>
```

Only rightmost cell

VRIGHT (chosen boundary condition)

$= 2\epsilon_0/\Delta x^2$ (again, the μ factor from the earlier slide, on the RHS)

linearSolveUpdater – the linearSolver block

```
<LinearSolver linearSolver>
kind = directSolver
solverType = superLU
verbosity = 127
</LinearSolver>
```

Solve Ax = b by computing A^{-1} directly. Simplest VSim solver option (by the length-of-input-file metric, at least), but not useful if your problem is too large.

All other VSim solver types are iterative:

- generalized minimum residual
- conjugate gradient
- biconjugate gradient
- etc.

Iterative solvers can be sped up by appropriate multigrid preconditioners (for which many options are available in VSim).

Looking at the matrix

- Edit the vsim.in file so that writeEquationToFile = 1.
- If you hit the ”Save” button, VSim Composer will
  - re-read the vsim.sdf file, and
  - generate a new .in file from the information it finds there.
- This will overwrite the change you just made.

- Therefore: if you want to do text-based problem setup, you'll need to do something like the following:
  - Generate the initial .in file from the sdf file with the “Save” button
  - Open a terminal window
  - Go to the directory where the .in file lives
  - Rename the .in file to something different, e.g. vsimTextBased.in
  - Edit this new .in file in the way you want to
  - Run VSim from the terminal window, pointing to the new .in file:

```
YOUR/PATH/TO/VSim-10.0/VSimComposer.app/Contents/Resources/engine/bin/vorpalser -dt 1.0 -d 1 -n 1 -i vsimTextBased.in
```

Assuming Ax=b, A is in esSolveMatrix.mtx

```
%%MatrixMarket matrix coordinate real general
11 11 29
1 1 1.7708375635183248e-09
2 1 -8.8541900000000002e-10
2 2 1.7708380000000000e-09
2 3 -8.8541900000000002e-10
3 2 -8.8541900000000002e-10
3 3 1.7708380000000000e-09
3 4 -8.8541900000000002e-10
4 3 -8.8541900000000002e-10
4 4 1.7708380000000000e-09
4 5 -8.8541900000000002e-10
5 4 -8.8541900000000002e-10
5 5 1.7708380000000000e-09
5 6 -8.8541900000000002e-10
6 5 -8.8541900000000002e-10
6 6 1.7708380000000000e-09
6 7 -8.8541900000000002e-10
7 6 -8.8541900000000002e-10
7 7 1.7708380000000000e-09
7 8 -8.8541900000000002e-10
8 7 -8.8541900000000002e-10
8 8 1.7708380000000000e-09
8 9 -8.8541900000000002e-10
9 8 -8.8541900000000002e-10
9 9 1.7708380000000000e-09
9 10 -8.8541900000000002e-10
10 9 -8.8541900000000002e-10
10 10 1.7708380000000000e-09
10 11 -8.8541900000000002e-10
11 11 1.7708375635183248e-09
```

```
%%MatrixMarket matrix coordinate real general
11 11 29
1 1 2*eps0/dx^2
2 1 -eps0/dx^2
2 2 2*eps0/dx^2
2 3 -eps0/dx^2
3 2 -eps0/dx^2
3 3 2*eps0/dx^2
3 4 -eps0/dx^2
4 3 -eps0/dx^2
4 4 2*eps0/dx^2
4 5 -eps0/dx^2
5 4 -eps0/dx^2
5 5 2*eps0/dx^2
5 6 -eps0/dx^2
6 5 -eps0/dx^2
6 6 2*eps0/dx^2
6 7 -eps0/dx^2
7 6 -eps0/dx^2
7 7 2*eps0/dx^2
7 8 -eps0/dx^2
8 7 -eps0/dx^2
8 8 2*eps0/dx^2
8 9 -eps0/dx^2
9 8 -eps0/dx^2
9 9 2*eps0/dx^2
9 10 -eps0/dx^2
10 9 -eps0/dx^2
10 10 2*eps0/dx^2
10 11 -eps0/dx^2
11 11 2*eps0/dx^2
```

$$\left(\frac{-\epsilon_0}{\Delta x^2}\right)\begin{bmatrix} -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix}$$

Assuming Ax=b, x and b are esSolve vectors

esSolveWriteVector.mtx (b)

```
%%MatrixMarket matrix array real general
11 1
0.0000000000000000e+00
6.1803398874989481e+00
1.1755705045849464e+01
1.6180339887498949e+01
1.9021130325903069e+01
2.0000000000000000e+01
1.9021130325903069e+01
1.6180339887498949e+01
1.1755705045849465e+01
6.1803398874989499e+00
1.7708375635183248e-09
```

First entry: $= \frac{2\epsilon_0}{\Delta x^2} \cdot \phi^{left}$

Entries 2–10: $= 20 \sin\left(\frac{\pi x_j}{L}\right) = \rho_j$

Last entry: $= \frac{2\epsilon_0}{\Delta x^2} \cdot \phi^{right}$

esSolveReadVector.mtx (x)

```
%%MatrixMarket matrix array real general
11 1
0.0000000000000000e+00
7.1308064483412903e+10
1.3563599878269949e+11
1.8668693648958243e+11
2.1946365612852356e+11
2.3075774401243900e+11
2.1946365612872348e+11
1.8668693648998233e+11
1.3563599878329944e+11
7.1308064484212875e+10
1.0000000000000000e+00
```

First entry: $= \phi^{left}$

Entries 2–10: $= \phi_j$

Last entry: $= \phi^{right}$

Regroup and Review

So far, we have:

-solved the discrete 1D Poisson equation 'by hand' and looked at the matrix and the vectors involved in that process

-looked at how VSim builds this matrix and these vectors with a FieldUpdater (of kind linearSolveUpdater), using MatrixFiller and StencilElement and LinearSolver blocks

-seen how to run VSim from the command line to point at a modified .in file

-seen how to examine the matrix and vectors VSim builds.

But:

-most interesting problems are not 1D

-most interesting problems involve particles, complicated geometries, and/or complicated boundary conditions

Let's add some interesting features to our input file, and see how things change.

Moving to 2D

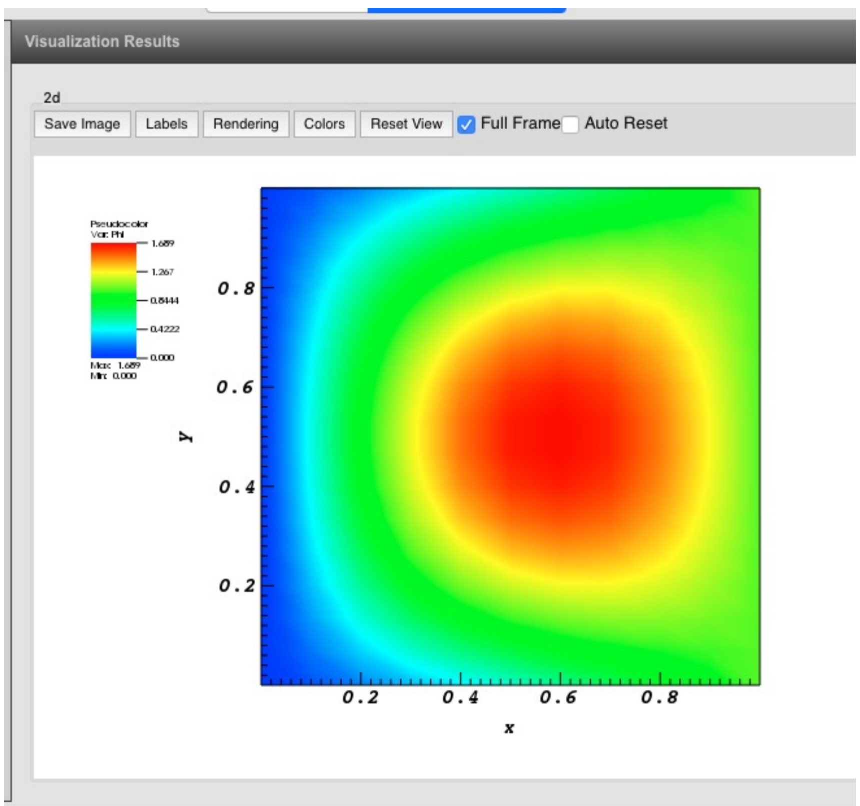
Let's copy the simulation we had before into a new simulation, and add:

Parameters

LY = 1

NY = 15

RHOZERO = 2.0e-10



SpaceTimeFunctions

RHOxt=RHOZERO*sin(PI*x/LX)*sin(PI*y/LY)

LINEARPHIxt=VLEFT+(VRIGHT-VLEFT)*x/LX

FieldBoundaryConditions

TOPBC, Dirichlet, LINEARPHIxt, upper y

BOTTOMBC, Dirichlet, LINEARPHIxt, lower y

Basic Settings

dimensionality = 2

Grid

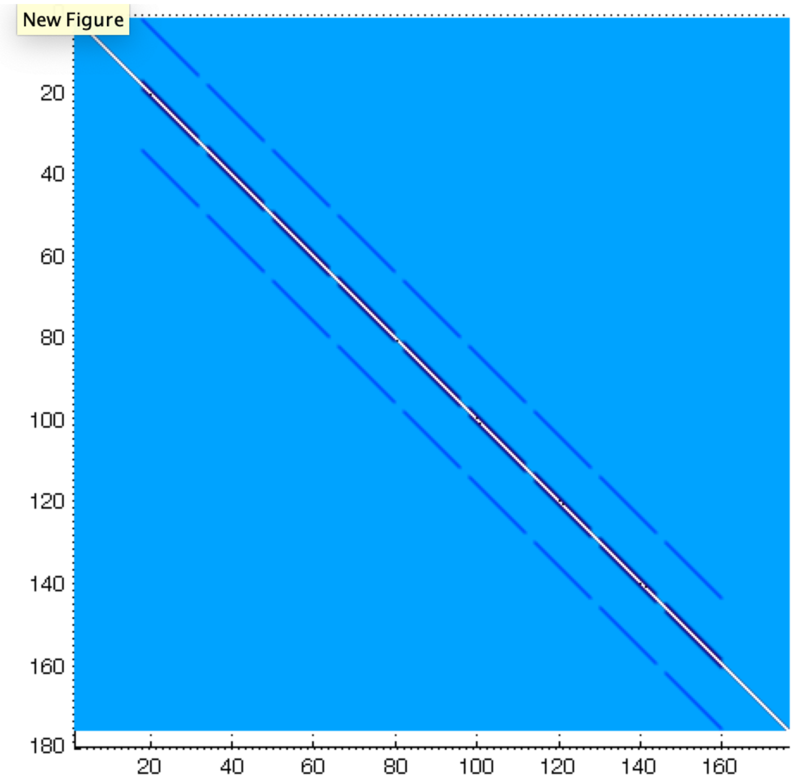
yMin=0

yMax=LY

yCells=NY

Matrix is larger, no longer tridiagonal

Now 176 x 176 [176 = 11*16 = (NX+1)*(NY+1)] and band-structured



ρ and ϕ arrays are now representing 2D quantities in a vector, e.g.

$$\begin{bmatrix} \rho_{1,1} \\ \vdots \\ \rho_{1,N} \\ \rho_{2,1} \\ \vdots \\ \rho_{2,N} \\ \vdots \\ \rho_{M,N} \end{bmatrix}$$

The same approach generalizes to 3D also; we will have large sparse matrices.

In general this 2D input file looks pretty similar to the 1D version.

Additional StencilElements relevant in 2D/3D

Typical stencil elements:

Δy^2 → value; Only if ≥ 2D → minDim; +1 cell in y → cellOffset

```
<STFuncStencilElement phi_npy>
value = 225.0
minDim = 2
cellOffset = [0 1 0]
functionOffset = [0. 0.5 0.]
rowFieldIndex = 0
columnFieldIndex = 0
</STFuncStencilElement>

...
```

Δz^2 → value; Only if ≥ 3D → minDim; -1 cell in z → cellOffset

```
<STFuncStencilElement phi_nmz>
value = 144.0
minDim = 3
cellOffset = [0 0 -1]
functionOffset = [0. 0. -0.5]
rowFieldIndex = 0
columnFieldIndex = 0
</STFuncStencilElement>
```

In 2D, general matrix row is

$$coeff \cdot [\cdots \quad 0 \quad phi_{nmy} \quad \cdots \quad phi_{nmx} \quad (phi_{dxm}+phi_{dxp}+phi_{dym}+phi_{dyp}) \quad phi_{npx} \quad \cdots \quad phi_{npy} \quad 0 \quad \cdots]$$

Adding GridBoundary geometric features

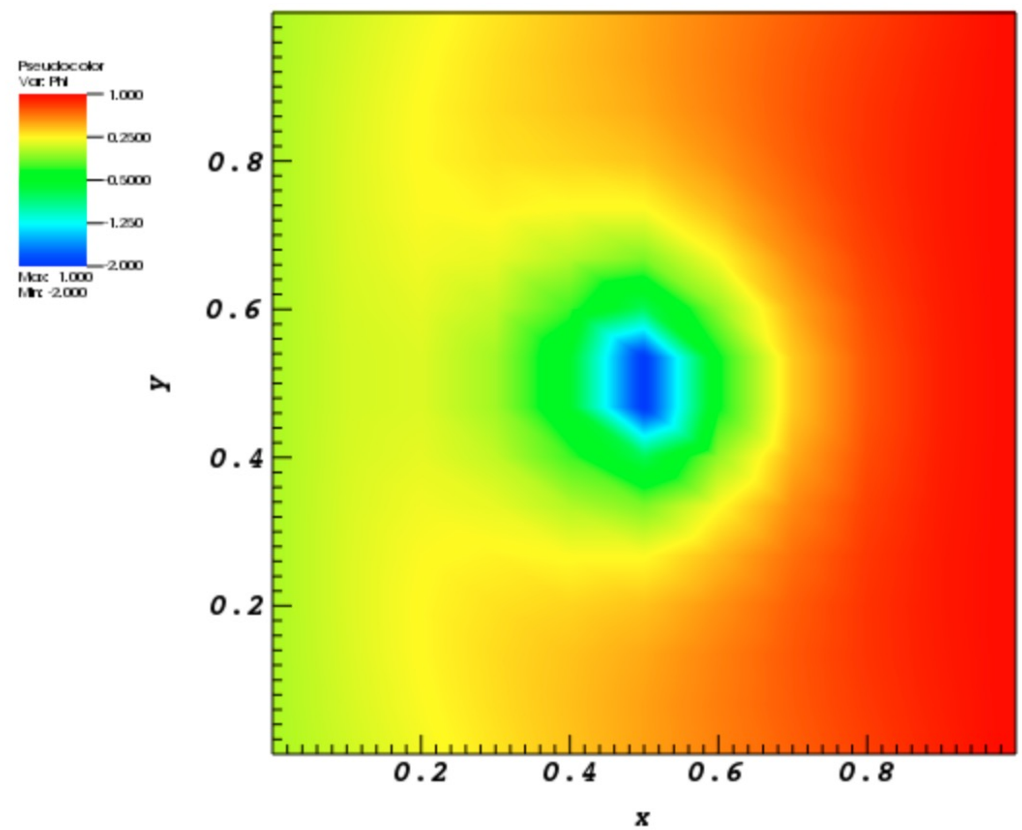
Let's modify our simulation some more, to add geometric features:

Geometries

Add Primitive: cylinder
material = PEC
length = 0.5
radius = 0.1
x position = 0.5
y position = 0.5
z position = -0.25
axis direction x = 0.0
axis direction y = 0.0
axis direction z = 1.0

FieldBoundaryConditions

Dirichlet, on cylinder, -2.0 V



New: EmMaterial and GridBoundary blocks

```
<EmMaterial PEC>
kind = conductor
resistance = 0.0
</EmMaterial>

<GridBoundary cylinder0>
kind = gridRgnBndry
calculateVolume = 1
dmFrac = 0.5
polyfilename = cylinder0.stl
flipInterior = True
scale = [1.0 1.0 1.0]
printGridData = False
mappedPolysfile = cylinder0_mapped.stl
</GridBoundary>
```

See documentation…

New: GridBoundary MatrixFillers

```
<MatrixFiller CYLINDERFiller>
kind = nodeStencilFiller
gridBoundary = cylinder0
rowInteriorosity = [cutByBoundary outsideBoundary]
colInteriorosity = [cutByBoundary outsideBoundary]
component = 0
minDim = 1
lowerBounds = [1 1 1]
upperBounds = [10 15 12]

<StencilElement ident>
value = 5.7552220814345554e-09
minDim = 1
cellOffset = [0 0 0]
rowFieldIndex = 0
columnFieldIndex = 0
</StencilElement>

</MatrixFiller>
```

```
<VectorWriter CYLINDERWriter>
kind = stFuncNodeVectorWriter
gridBoundary = cylinder0
minDim = 1
lowerBounds = [1 1 1]
upperBounds = [10 15 12]
component = 0
interiorosity = [cutByBoundary outsideBoundary]

<STFunc function>
kind = expression
expression = -2.0
</STFunc>

scaling = 5.7552220814345554e-09
</VectorWriter>
```

See documentation…

We could presumably go and look at the matrix again, and see how these operations changed it, and get a sense for what VSim is doing behind-the-scenes.

Adding particles

- Instead of doing this through the visual setup, let’s just open an example and test our developing .in-file-reading skills.

- File > New From Example > VSim for Plasma Discharges > Capacitively Coupled Plasma > Turner Case 2

- I’ll show a quick movie of this discharge so that you have a sense for what we’ll be looking at: available here: http://nucleus.txcorp.com/~tgjenkins/movies/ShortCCPmovie.mov

- Neutral gas is contained between two parallel plates; one plate is grounded and the other biased with RF. The motion of free electrons creates plasma between the plates, and the formation of plasma sheaths is observed. The long-time steady state of the discharge is a balance between collisional ionization (source) and wall losses (sink).

Looking at the Turner .in file

- Some familiar things: Fields, FieldUpdaters, UpdateSteps, MultiFields, etc.
- Some new things: Species, Fluid, History, collisional physics, etc.