Alamode: A LAyered MOdel Development Environment for simulation of impurity diffusion in semiconductors

D.W. Yergeau and R.W. Dutton
Stanford University

August 20, 1997

Contents

Chapter 1

Introduction

Alamode (A LAyered Model Development Environment) is a program which solves systems of partial differential equations (PDEs) that are described using a dial-an-operator paradigm. Its purpose is to allow quick prototyping of PDE-based models for thermal diffusion in semiconductors, and also provides a framework for evaluating new numerical techniques to discretize and solve these equations. The emphasis in the user interface is to provide for complete specification of the equations and boundary conditions comprising the model, while eventually providing additional layers of access to the discretization and numerical techniques used to solve the equations.

Unlike SUPREM, which is designed to be a *process simulator*, *Alamode* is only a *PDE solver*. This difference is reflected strongly in the user interface. Whereas SUPREM has reasonable defaults for models and parameters, *Alamode* requires the complete specification of all terms, parameters, and boundary/interface constraints for a model as applicable to the specific device structure being simulated. While this may initially seem excessive, this enables more flexibility in model description than SUPREM allows and ensures that there are no hidden features of any model.

Everything described in this manual is subject to change, and backward compatibility to prior versions will be attempted, but is not guaranteed. Currently available features will probably not be deleted, but they may be significantly changed. User feedback is important to help mold future changes. Send suggestions to `alamode@gloworm.Stanford.EDU`.

This manual corresponds to release 97.06.18 of *Alamode*. The most recent version of the documentation is available on the WWW from http://WWW-TCAD.Stanford.EDU/tcad/programs/alamode.ht

The first widely distributed release was 96.11.07.

Between 96.08.19 and 97.08.19:

- PMM support.

- KPM support.
- Fixes to the `profile` function and command.

Between 96.11.07 and 97.06.18:

- Added `profile` function and command.
- Added `box` function.
- Added `erfc` function.
- Added `sqrt` function.
- Added `fermi_nnp` function.
- Added `min` function.
- Added `max` function.
- Added `arcsinh` function.
- TCL variables should now be accepted in all slots that used to require fields or functions.
- Nodal weighting for triangle elements is based on the intersection of the voronoi cell with the element instead of a row-sum lumped mass matrix. This seems to be a better estimate of the area for nodal lumping and preserves symmetry in meshes derived from a tessellation of rectangles.
- Implemented a primitive estimate for the initial time step (`-estimateFirstTimeStep`).
- Added "field -linkVariableAtPoint" which links a TCL variable to a field value nearest a given point. This is useful for monitoring concentrations at a specific point.
- Replaced the quasiFermiEField operator with quasiNeutralEField. In addition to being incorrect from a model perspective, the quasiFermiEField usually had slow nonlinear convergence. Another alternative for electric field enhancements is to add a Poisson equation to your model to solve for ψ and use the diffusion operator with the diffusivity concentration product in the diffusivity slot (i.e. $\nabla \cdot DC \nabla \psi$).
- Changed the default diffusion operator to a more efficient geometric diffusion operator. This may produce slightly different results than older releases, but they are just as accurate. The old diffusion operator is still available as "cdiffusion".
- Timestepping error control, update norm convergence, and threshold detection/reporting criteria and are now specified for each field instead of for each model.

- Restructured the code to support nodal assembly and eliminate duplicated evaluation during assembly. The result has been speedups ranging from 2 to 20.
- Error-driven timestep estimation has been added to the generalized trapezoid family of integration algorithms. This has not been well-tested.

Chapter 2

Model development in *Alamode*

A model in *Alamode* is a set of coupled systems of PDEs describing transport of impurities and defects in semiconductor materials and interfaces. An annotated mesh data structure is used to represent the physical structure (wafer state) of a device, including materials, interfaces, and impurity and defect profiles. A *dial-an-operator* paradigm is used to build up systems of equations from a library of operators and functions. Systems of equations are attached to materials and interfaces to construct a model. The abstractions used to represent models are:

The **domain** is the collection of regions and fields of interest to represent the physical state of a device.

A **region** is a subset of the domain in which the same system of equations will be applied. Regions are usually materials, material interfaces, and artificial boundaries, but they may also be used to isolate subset of a material where the equations differ from those applied elsewhere in the same material (e.g. a damaged or non-crystalline region of a normally crystalline material).

A **constant variable** is a named scalar value that is not spatially varying.

A **field variable** is named scalar value that is interpolated on the mesh in a region. Fields are often discontinuous at material interfaces (e.g. impurity concentration with a density change), and, as a result, the naming convention adopted for fields is *VariableInMaterial*. This naming convention has been the source of a lot of confusion, mainly because interior regions are also named with the material name and because there is not a material "object" in *Alamode*. While it may seem more natural to just refer to fields by the impurity/variable name (e.g. Boron), that convention would not provide a way to distinguish between field values on either side of material interface without an additional key, and the additional key would likely be another source of confusion. This convention of *VariableInMaterial* provides an unambiguous way to distinguish the values on either side of a material interface and it simplified the implementation by allowing fields to be consistently treated as simple interpolated scalar quantities.

Functions are one of the equation building blocks. Functions provide a way to combine constant variables, field variables, and previously defined functions into more complex expressions. Examples of functions include sum and product, as well as more complicated expression such as that used to calculate a normalized quasi-Fermi carrier concentration.

Operators are terms in an equation. Operators may reference constant variables, field variables, and previously defined functions as needed.

An **equation** is a list of operators (sum of terms) used to solve for a single field. Presently, equations can be used for elliptic or time-dependent partial differential equations as well as algebraic interface constraint equations.

A **systemInRegion** is a list of equations describing the coupling among fields within a single region.

A **model** is an aggregation of system/region pairs describing globally coupled system of equations in the domain.

The following sections discuss how TCL (Tool Command Language) is used both for model description and as a command language for *Alamode*.

2.1 Commands Overview

The shell used by *Alamode* is an extension of a standard TCL shell. This document makes no attempt to teach TCL. See *Tcl and the Tk Tookit* by John K. Ousterhout [3] for a definitive reference on TCL.

Recommended reading in *Tcl and the Tk Tookit*:

- Chapter 3: Tcl Language Syntax. This will help to avoid mistakes using quotes, brackets, etc.
- Section 6.1-2: Lists.
- Section 15.5,7: Tk widgets. Although *Alamode* doesn't use any Tk widgets, the philosophy for these widgets is similar to that of the model definition commands.
- Section 28.3: Action-oriented vs. object-oriented. This provides more information on why models are implemented as they are.

Both ordered and unordered lists are used extensively in *Alamode* model descriptions: unordered lists of operators are used to create an equation and ordered lists are used to pass parameters to operators and functions. Although TCL provides two ways to create lists (i.e. via the list command or using quoting), the list command should always be used. The list command always does the right thing, even if the values of variables used to build up the list contain spaces. The following example demonstrates the differences.

```
# The variable "a" contains a space.
% set a "foo bar"
foo bar
% set b "baz"
baz
# Quoting does not preserve the space.
% set c "$a $b"
foo bar baz
% set d [list $a $b]
{foo bar} baz
# The result is a list with three items instead of the expected 2 item.
% llength $c
3
% llength $d
2
```

The *Alamode* shell is not an extension of the SUPREM shell. One difference between the SUPREM command language and *Alamode*'s is that parameters use `-tag value`, which is common in TCL extensions, as opposed to SUPREM's GENII-style `tag = value`. Another key difference is that many commands return usable values. These values are used to build up a data structure describing a model. Some of the commands also create new commands, which can then be used as any other *Alamode* command.

Beyond the standard TCL shell commands, *Alamode* provides commands for input/output of mesh via file formats described in Appendix 5 and Appendix 6, query of regions and fields, model definition, and model solution. These commands are discussed in detail in the following sections.

2.2 Input and output commands

Presently, the only supported mesh I/O commands are `structure` and `sstructure`.

2.2.1 `structure`

Usage: `structure [ -infile fileName ] | [ -outfile fileName ]`

The structure command initializes *Alamode*'s mesh and field representation from a 1D structure file or saves the current field values to a 1D structure file. See Appendix 5 for format of this structure file.

The 1D structure file allows for multiple material regions with the same set of fields supported in each material, even if it doesn't make sense to have such a field in the material (e.g. silicon vacancy in a oxide material). The storage format considers all fields at interfaces to be discontinuous. The names for materials and fields are specified as strings in the file.

2.2.2 `sstructure`

Usage: `sstructure [ -infile fileName ] | [ -outfile fileName ]`

The sstructure command initializes *Alamode*'s mesh and field representation from a Stanford SUPREM structure file or saves the current field values in a Stanford SUPREM structure file.

This command does not support reading or writing structure files from commercial versions of SUPREM.

`sstructure [ -infile fileName ] | [ -outfile fileName ]`

Caveats:

- Currently only 2D SUPREM structure files are supported by the `sstructure` command. As was the case for SUPREM releases before 9130, a two line quasi-1D SUPREM mesh should be used for 1D simulations.

This SUPREM structure file format is documented in Appendix 6.

2.3 Query commands

2.3.1 `regions`

Prints out the regions in the current mesh. This is the definitive way of determining the region names.

Interior regions are typically named using the material. The 1D structure file format picks up the material name from the file. SUPREM materials are translated using Table 2.1.

Interface regions are typically named with the two material names separated by a slash ("/"). For example, an interface between silicon and silicon dioxide in a SUPREM structure is named

Material number	*Alamode* name
0	Gas
1	Oxide
2	Nitride
3	Silicon
4	PolySilicon
5	OxiNitride
6	Aluminum
7	Photoresist
8	GaAs

Table 2.1: *Alamode* names for SUPREM materials

"Silicon/Oxide". The order of the materials is unimportant. "Oxide/Silicon" will match a "Silicon/Oxide" region and vice versa.

Artificial interfaces are specially named. For SUPREM structures, these are named by concatenating "Exposed", "Backside", or "Reflecting" with the material name. For the 1D structure file format, the left boundary (smallest x coordinate) is named by concatenating "Exposed" with the material name; the right boundary is named by concatenating "Backside" with the material name.

The above naming conventions are what is implemented by the present structure file I/O commands, not requirements for *Alamode*. Regions are not required to be partitioned by material. The only requirement is that they must be partitioned appropriately for the implementation of models which consist of a homogeneous system of equations within each region.

2.3.2 `region regionName`

Prints out the geometric mesh in a region in a format useful to those familiar with the internal data structures of *Alamode*.

2.3.3 `fields`

Prints out the fields in the current mesh. This is the definitive way of determining the field names.

The structure file interfaces name fields by concatenating the impurity/variable name with the material. This naming convention provides and unambiguous way of selecting the appropriate values of an impurity/variable on either side of a material interface.

The 1D structure file format picks up the names of impurities/variables from the file. SUPREM impurities/variables are translated using Table 2.2.

Impurity	*Alamode* name	Impurity	*Alamode* name
0	Vacancy	25	Gallium
1	Interstitial	26	DeviceCKT
2	Arsenic	27	Impurity27
3	Phosphorus	28	Impurity28
4	Antimony	29	Impurity29
5	Boron	30	Impurity30
6	Electrons	31	Beryllium
7	Holes	32	ElectricallyActiveBeryllium
8	Xvelocity	33	Magnesium
9	Yvelocity	34	ElectricallyActiveMagnesium
10	Dry02	35	Selenium
11	Wet02	36	ElectricallyActiveSelenium
12	Trap	37	Silicon
13	Gold	38	ElectricallyActiveSilicon
14	Potential	39	Tin
15	Xstress	40	ElectricallyActiveTin
16	Ystress	41	Germanium
17	MixedCompOfStress	42	ElectricallyActiveGermanium
18	Cesium	43	Zinc
19	ChangeInInterfaceArea	44	ElectricallyActiveZinc
20	ElectricallyActiveArsenic	45	Carbon
21	ElectricallyActivePhosphorus	46	ElectricallyActiveCarbon
22	ElectricallyActiveAntimony	47	Generic
23	ElectricallyActiveBoron	48	ElectricallyActiveGeneric
24	PolysiliconGrainSize	49	DummyImpurity

Table 2.2: *Alamode* names for SUPREM impurities/variables

2.3.4 `field`

This command can be used to initialize a field's values, get/set minimum and maximum values, link a TCL variable to a node nearest a specified point, set threshold and timestep error parameters for a field, and print out the field mesh.

`-setValue`

Usage: `field fieldName [-inRegion regionName] -setValue value`

This form initializes the values of a field. The value can be a numeric value, a field, or a function. For example,

```
field BoronInGas -inRegion ExposedSilicon -setValue 1.0e22
```

could be used to set values at gas nodes for a gaseous predeposition, and

```
set BssValue 1.5e20
field ActiveBoronInSilicon -inRegion Silicon -setValue \
  [function ssactive -parameters [list BoronInSilicon BssValue]]
```

could be used to update the active boron field after an anneal.

The 97.11.07 version had an undocumented limitation for field initialization using functions. If the destination field was also used as a source field in a function initialization, the results were undefined. That limitation has been removed.

`-getMinValue`

Usage: `field fieldName [-inRegion regionName] -getMinValue`

The form returns the minimum value at any node for the field.

`-getMaxValue`

Usage: `field fieldName [-inRegion regionName] -getMaxValue`

The form returns the maximum value at any node for the field.

`-minValue`

Usage: `field fieldName [-inRegion regionName] -minValue value`

The form sets the minimum value at any node for the field. This only affect the current values, and is not applied after updated during any solves (see `-minimumAfterUpdate` below for the command that affects minimum values during a solve).

`-maxValue`

Usage: `field fieldName [-inRegion regionName] -maxValue value`

The form sets the maximum value at any node for the field. This only affect the current values, and is not applied after updated during any solves.

`-linkVariableAtPoint`

Usage: `field fieldName -linkVariableAtPoint variable coordinate`

This form links a TCL variable to the node nearest the specified coordinate. The coordinate is given as a list of floating point numbers in *Alamode*mesh units (i.e. centimeters), and unspecified axes default to 0.0.

For example,

```
field BoronInSilicon -linkVariableAtPoint exposedBoron [list 0.1e-4]
puts "Value of boron at exposed surface is $exposedBoron"
```

links the TCL variable "exposedBoron" to the value closest to $(x = 0.1\mu\text{m}, y = 0\mu\text{m}, z = 0\mu\text{m})$.

```
field BoronInSilicon -linkVariableAtPoint monitor [list 0.0e-4 0.2-e4]
```

links the TCL variable "monitor" to the values closest to $(x = 0\mu\text{m}, y = 0.2\mu\text{m}, z = 0\mu\text{m})$.

If the mesh changes (e.g. another structure is read in), subsequent access to the field value via the TCL variable will produce undefined results, and may even lead to memory access violations (i.e. Unix segmentation violations).

`-thresholdValue/-warnThreshold/-minimumAfterUpdate`

Usage: `field fieldName [threshold parameter] value`

This form set the threshold parameters for the field. These parameters are used during the nonlinear solve to check and adjust nodal values that may cause nonconvergence in the nonlinear solve. The parameters that can be set are `-thresholdValue`, `-warnThreshold`, and `-minimumAfterUpdate`.

If the model requests `-checkThresholds` (default), a single warning message is printed if any nodes fall below the warn threshold after a nonlinear update.

If the model requests `-adjustThresholds` (default), any nodes that violate the threshold value condition are set to be either the minimum after update (default) or the old concentration (if `-minConcIsOldConc` is specified).

If the model requests `-printThresholdViolation`, all nodes that violate the warning threshold value are printed.

The default values are 0 for `-thresholdValue`, 0 for `-warnThreshold`, and 10^{-10} for `-minimumAfterUpdate`. These are reasonable for most concentration fields.

If thresholds are adjusted after the last nonlinear update (i.e. convergence is detected), dose conservation may be violated, so the default is to `-failIfHadThresholdViolation`. Note that although node that fall below the threshold value may be adjusted, only nodes that fall below the warn threshold value will be considered to be a violation. The split of values provides for robust nonlinear solution (i.e. recombination can't become generation because of an incorrect sign) without unnecessary repeating of time steps due to numerical noise leading to insignificant negative values after an update.

-relTSTol/-absTSTol

Usage: `field fieldName [time error control parameter] value`

This form set the timestepping error control parameters for the field. The parameters that can be set are `-relTSTol` (ϵ_R in 4.35) and `-absTSTol` (ϵ_A in 4.35).

The default values are 0.01 for `-relTSTol` and 10^{-5} for `-absTSTol`.

-relNLUpdateTol/-absNLUpdateTol

Usage: `field fieldName [nonlinear update error parameter] value`

This form set the nonlinear update error control parameters for the field. The parameters that can be set are `-relNLUpdateTol` and `-absNLUpdateTol` in equation 4.42.

The default values are 10^{-8} for `-relNLUpdateTol` and 1 for `-absNLUpdateTol`.

-offsetVariable

Usage: `field fieldName -offsetVariable variable`

This form associates a variable containing an offset value with the field. This is used for setting up nonzero offset fields, which can improve the numerical robustness of the timestepping algorithm.

`-print`

Usage: `field fieldName -print`

This form prints out the field mesh for a field in a format useful to those familiar with the internal data structures of *Alamode*.

2.4 Model definition commands

The model definition commands build up a complete model as a hierarchy of systems, equations, operators, and functions based on the available regions and fields. Each of the model definition commands creates a new TCL commmand, so you will often see TCL code similar to

```
set myModel [model ...

$myModel -modelCmd
```

In this fragment, the TCL variable `myModel` is used to store the name of the command created by the `model` command. The syntax `$myModel` substitutes the string stored by the TCL variable `myModel` (i.e. the name of the new command) before interpreting the second line and running the new command.

The commands for the components of a model definition are described below in a bottom-up fashion.

2.4.1 `function`

A function is an object that produces a scalar value from an expression composed of fields, other functions, and variables.

`function functionName [ -parameters parameterList ]`

The available functions and their parameters are listed in section 3.1.

Parameter lists are simply ordered lists of strings describing one of a field, a function, or a TCL variable name in the appropriate slot for that particular function. The string for a function is the value returned by a previous `function` command.

2.4.2 operator

```
operator operatorName [ -parameters parameterList ]
          [ -sign +/-1 ]
          [ -extraIntegrationOrder n ]
          [ -useNodalQuadrature ]
          [ -useInteriorQuadrature ]
```

The available operators and their parameters are listed in section 3.2.

The operator command creates a new command and returns the name of this command. The returned command name is used as a reference to the instance of the function and is used to build up the data structure describing a complete model.

Parameter lists are simply ordered lists of strings describing one of a field, a function, or a TCL variable name in the appropriate slot for that particular operator. The string for a function is the value returned by a previous `function` command.

The default sign is +1. Be very careful with the sign on the recombination operators. They are actually generation operators with the default sign.

`-extraIntegrationOrder` can be used to reduce or increase the order of the numerical integration used to evaluate the operator. Each operator estimates the optimal order needed based on the order of interpolation of the fields and functions it was given. Some functions, however, provide nonlinear mappings which do not have a finite polynomial series expansion. In these cases, the interpolation order estimate provided by the function may not be adequate for accurate integration.

`-useNodalQuadrature` and `-useInteriorQuadrature` allow the user to override the default quadrature used by operators. Reaction operators typically use nodal quadrature which amounts to pointwise evaluation at the nodes of field elements. Most other operators use interior quadrature, which uses an integration scheme appropriate to exactly integrate the fields or functions it is operating on. It is not recommended to change the defaults.

2.4.3 equation

An equation combines a list of operators into a scalar partial differential equation. Equations are used both for operators applied to the interior of a structure as well as for the aggregation of operators applied to a boundary or interface to enforce a boundary condition.

```
equation -solvesFor fieldName
          -operators operatorList
          [ -parabolic ]            [ -name equationName ]
```

The name of the equation is for debugging purposes. Any string may be used.

The fieldName being solved for specifies to which field nonlinear updates are applied. For parabolic systems of differential equations, this should be the field in the time derivative term.

Parabolic indicates that the PDE is of the form

$$\frac{\partial C}{\partial t} = \text{sum of operators}$$

where C is the fieldName being solved for. Instantaneous algebraic flux constraints, such as equations composed of a `segregation` operator are not parabolic.

If `-parabolic` is not specified, the equation is an elliptic PDE or a constraint equation of the form

$$0 = \text{sum of operators}$$

and the nonlinear update is applied to the fieldName being solved for.

Future directions:

- The equation name will eventually be optional.
- Stabilized formulations may require that further distinction be made among the operators. Galerkin/Least Squares (GLS) and upwinding requires distinguishing between advective and diffusive operators.

2.4.4 `systemInRegion`

A system in a region couples the equations implementing homogeneous physics in a region.

```
systemInRegion [ -name systemName ] -region regionName -equations equationList
```

The name of the system is used for debugging purposes. Any string may be used.

2.4.5 `model`

A model is the aggregation of all of the systems and fixed (dirichlet) boundary conditions that implements a multi-region system of PDEs. A model provides the coupling of the multi-region physics into a single system of nonlinear equations. A system provides the coupling within a single region.

```
model -systems systemList
          [ -fixedBC dirichletList ]
```

```
[ -continuityConstraints constraintsList ]
[ -atStartOfEachTimeStep script ]
[ -atEndOfEachTimeStep script ]
[ -afterConverged script ]
[ -preSolve script]
[ -postSolve script ]
```

Each item in a dirichletList is list containing a field name and a region name. The following code segment will provide a dirichlet boundary condition for vacancies and interstitials at a silicon/oxide interface.

```
$model -fixedBC [list \
    [list VacancyInSilicon {Oxide/Silicon}] \
    [list InterstitialInSilicon {Oxide/Silicon}] ]
```

Each item in a constraintsList is a list containing two field names and an interface region. This makes is possible to enforce continuity of a field across an interface, even if the mesh represents the fields as discontinuous across an interface as the present structure files do. The following code segment enforces continuity in Potential across the silicon/oxide interface.

```
$model -continuityConstraints [list \
    [list PotentialInSilicon PotentialInOxide {Oxide/Silicon}] ]
```

`-atStartOfEachTimeStep/-atEndOfEachTimeStep/-afterConverged` allow the user to specify a TCL script that gets executed during each time step or after the nonlinear solve has converged. The `-atStartOfEachTimeStep` script is executed before nonlinear solve is attempted with a candidate Δt_n (TIME_STEP). `-atEndOfEachTimeStep` is executed at the end of a time step only if the nonlinear solves were successful. `-afterConverged` is executed before returning from the nonlinear solver if it succeeded.

The script can checkpoint solutions, modify TCL variables (including TIME_STEP), and update fields. The TCL variables `SOLVE_TIME` and `TIME_STEP` are available (they must tagged as global within a TCL `proc`). `SOLVE_TIME` is the start time of the current time step. `TIME_STEP` is Δt_n for this time step.

The user may also call the model command with `-getUnknown equationNumber` or `-getUnknowns` to get the value of a single unknown or of all of the unknowns. Since these return values a strings via the TCL interpreter, they are inefficient. They are primarily used for testing the numerical algorithms with very small examples where the mapping of unknowns to equation numbers is known.

`-preSolve` and `-postSolve` allow the user to specify a TCL script that gets executed before or after the the solution command. These are intended to be used for validity checking (e.g. to ensure that necessary fields are actually part of the domain), activation, etc.

2.5 Simulation commands

Once a complete model is created, the user can simulate the model to update the state of the structure. For diffusion simulation, a `-transientSolve` is usually performed for a certain time interval. A `-staticSolve` is also available for solving elliptic systems of PDEs.

The default timestepping algorithm for `-transientSolve` is TR/BDF2. The timestepping algorithm can be changed using `-timeIntegrator algorithm`. The algorithms that are currently available are `TRBDF2` (second-order accurate, A-stable, L-stable, not monotonic), `backwardEuler` (generalized trapezoid with $\alpha = 1$, first-order accurate, A-stable, L-stable, monotonic?), `implicitEuler` (generalized trapezoid with $\alpha = 0$, first-order accurate, conditionally A-stable), and `trapezoidalRule` (generalized trapezoid with $\alpha = \frac{1}{2}$, second-order accurate, A-stable, not L-stable, not monotonic).

2.5.1 Universal timestepping parameters

A transient solve integrates the spatially discretized PDEs from $t = 0$ to $t = T_{\text{stop}}$. Any of the parameters below can be modified by a `-atStartOfEachTimeStep` or a `-atEndOfEachTimeStep` script.

`-stopTime tmax`

sets T_{stop}. Note that there are no units assumed for the time time scale; T_{stop} should be appropriate for the units on the parameters and the assumed mesh units

External units in the the 1D file format are assumed to be $10^{-6}m$ and are mapped into internal units of cm.

`-estimateFirstTimeStep/-noEstimateFirstTimeStep`

Try to estimate a first time step instead of using the user-provided one or the global default. The estimate is primitive, but it should be enough to guarantee nonlinear convergence in the first step. It is not guaranteed to be sufficiently small to avoid repeating time intervals due to temporal error estimates.

If the estimate is determined to be unreasonable, the default initial step is used. The estimation is not turned on in the present version.

`-deltaT initial`

sets the initial time step. There is no automatic estimation of the first time step. The default is 0.01.

`-deltaTMult m`

Specifies a multiplier for subsequent time steps ($\Delta t_{n+1} = m\Delta t_n$) if timestep control is not used. The default is 1, giving uniform timesteps.

`-useTimeErrorControl/-noUseTimeErrorControl`

Specifies whether or not timestep selection is based on an estimate of error in the solution. The default is on.

`-minDeltaT dtmin`

sets the smallest time step to be accepted. If timestep estimation or nonlinear failures produce a Δt_n smaller than this, the simulation gives up. The default is 10^{-30}.

`-maxDeltaT dtmax`

sets a ceiling on the largest time step that can be taken. Although error driven timestep control usually provides a reasonable next time step, this can be used as an addition cap if timesteps above a certain threshold cause convergence or accuracy problems. The default is 10^{30}.

`-printEvery n`

specifies how often to print out the complete solution vector. The default is -1 (i.e. do not print). This is a debugging command left over from early implementations, but it is sometimes still useful.

2.5.2 TR/BDF2 timestepping parameters

Details on the TR/BDF2 algorithm can be found in section 4.2.2 and [1].

`-gamma g`

specifies the γ parameter in the TR/BDF2 ($0 < \gamma < 1$). The default value is $2 - \sqrt{2} \approx 0.59$. This value minimizes the constant in the local truncation error expression.

`-trbdfErrorMethod method`

specifies the which approach is used to estimate the local truncation error and which norm is used. The available methods are `dd/pointwise` (divided differences for LTE estimate with a pointwise norm to measure the error), `dd/global` (divided differences for the LTE estimate with a global norm to measure the error), and `milne`, which uses a Milne's device to estimate the LTE and a pointwise norm to measure the error. The Milne approach uses a second TR step over the second half of the interval (and, hence an expensive third nonlinear solve). Although that approach has had a good deal of success with quasi-equilibrium diffusion models, it seem to be prone to oscillations from the lack of A-stability in the TR-TR step when used with fully kinetic models.

The default is `dd/pointwise`.

2.5.3 Generalized trapezoid timestepping parameters

Details on the generalized trapezoid family can be found in section 4.2.1 and [2]. The present implementation does not contain a timestep estimator for this family of algorithms.

2.5.4 Nonlinear solution parameters

A variant of Newton's method is used to solve the nonlinear system of equations. The algorithm is outlined in section 4.3. The parameters discussed below provide access to convergence criteria, non-convergence detection, update strategies, and detection/correction of non-physical behavior in the solution.

`-maxNLSteps n`

Specifies the maximum number of nonlinear steps to take in trying to meet the convergence criteria. The default is 20. Values less than or equal to zero are equivalent to infinity (never give up). Reasonable nonlinear systems should converge in 3-10 nonlinear steps.

A failure of the nonlinear solve does not terminate the simulation. It exits the nonlinear solver and returns an error condition to the calling level. If the calling level was a `-staticSolve`, the error will be caught by the TCL interpreter. If the calling level was a timestepping algorithm, the timestepping algorithm will attempt to resolve with a smaller time step unless `-minDeltaT` is reached.

`-extraNLSteps n`

Specifies the number of extra nonlinear solves and updates to apply after convergence is detected. The default is 0.

Newton algorithms have a second-order convergence rate when sufficiently close to the solution. Taking a few extra nonlinear steps can often produce a solution to the discretized nonlinear system that is accurate to machine precision. This is not to say that there would be no discretization error. Discretization error is related to the timestep, element size, spatial interpolation, and the operators in the equations.

`-useResidualNorm/-noResidualNorm`

Turns on/off the use of residual norm (equations 4.40 and 4.41 as a way of determining convergence. The default is on.

`-absNLTol tolerance`

Specifies ϵ_A in 4.41. The default is 1.0.

`-relNLTol tolerance`

Specifies ϵ_R in 4.40. The default is 10^{-8}.

`-useUpdateNorm/-noUpdateNorm`

Turns on/off the use of the update norm (eq. 4.42) as a way of determining convergence. The default is on. This should be turned off if it believed that the update norm is not a reasonable convergence measure.

Turning this on is useful if the behavior of the residual indicates numerical truncation error (i.e. if the residual reduces rapidly, but then bottoms out at a level insufficient to satisfy the residual convergence tests). This behavior can show up with very small time steps (TR/BDF2 and the concentration form of the generalized trapezoid family subtract nearly equal vectors that are proportional to $1/\Delta t_n$) and nearly equilibrium reactive-diffusive systems.

`-useConvergeHeuristics/-noConvergeHeuristics`

Turns on/off several heuristics that are used to determine if nonlinear solution will probably not converge in a reasonable number of steps. These heuristics are a sufficient reduction condition (i.e. after 10 steps the residual reduces by at least 0.5% on each subsequent step) and a cap on the growth in the residual (i.e. the residual does not grow more than a factor of 10^{10} between any two steps).

Regardless of whether or not these tests are turned on, the residual is checked for obvious non-convergence indicators (i.e. IEEE NaN and infinity) on hardware that supports it.

`-checkThreshold/-noCheckThreshold`

Turns on/off (default on) checking for values falling below the field warn threshold value after a nonlinear update. A summary is printed if any are found.

In *Alamode* the unknowns typically represent a concentration, and it is not physical to have negative concentrations. The nonlinear solution may compute negative concentrations due to overshoot. It is usually a good idea to adjust these values to avoid converging to an incorrect fixed point in the nonlinear solver.

`-adjustThreshold/-noAdjustThreshold`

Turns on/off (default on) the adjustment of values falling below the field threshold value after a nonlinear update.

`-printThresholdViolations/-noPrintThresholdViolations`

If turned on, the unknowns that violated the threshold are printed along with the former and adjusted values. Turing this on can generate a lot of output. It is off by default.

`-minConcIsOldConc`

Specifies to use the previous value at the node rather than the constant given by `-minConc`.

`-failIfHadThresholdViolation/-noFailIfHadThresholdViolation`

If this is turned on and if there are threshold violations on the last nonlinear iteration when the convergence criteria was met, the nonlinear solver will report a failure to the calling level.

The default is that threshold violations on the last nonlinear step are a reason for failure.

2.5.5 Linear solver parameters

Alamode uses the Simplified Linear Equation Solver (SLES) library form the Portable Extensible Toolkit for Scientific Computation (PETSc) as its linear equation solver. This provides access to a wide range of preconditioned iterative algorithms for solving linear systems of equations.

Most linear systems can be solved with `-slesPreconditioning ilu -slesAccelerator bcgs`, but some may require a direct factorization.

`-slesPreconditioning method`

Specifies the preconditioning scheme to be used by the SLES linear solver. The default is "ilu" with 0 fill. Other preconditions methods include: lu, jacobi, ssor, icc, iccjp, bdd, osm, and none.

For a direct solve, use "lu" with "none" for the accelerator.

`-slesAccelerator method`

Specifies the acceleration scheme to be used for the SLES linear solver. The default is "bcgs". Other accelerators include: none, richardson, chebychev, cg, gmres, cgsplit, tfqmr, lsqr, and cr.

`-slesOrdering orderType`

Specifies the sparse storage format. It is not recommended that this be changed. Available orderings for the matrix storage are: rcm, qmd, nd, 1wd, and sdo.

`-slesFill fillVal`

Specifies how much fill should be generated for ILU. The default is 10.

Recent optimizations have provided more room for tradeoffs in performance between the quality of the preconditioner and the expense of computing the preconditioner. The total time for a iterative linear solve is the time to compute/apply the preconditioner plus the number of iterations needed for accelerator times the time per iteration. The default fill provides more space to compute a good preconditioner at the expense of more memory usage and more time to compute/apply the preconditioner. It can be reduced, but the number of accelerator iterations may begin to climb and the overall time for a linear solve may increase.

The SLES linear solver will print iteration counts, total flops, and memory usage (memory usage is somewhat misleading because it does not include all the space used by linear solver). The best performance is typically achieved if the accelerator iterations is kept small (less than 3) instead of allowing it to creep into the 15-50 range.

It is highly recommended to tune the fill parameter to the model and dimensionality of the mesh if a large number of runs will be made using the model with similar parameters and similar initial conditions. There can easily be a factor of 2 improvement with an optimal fill.

`-slesAccuracy accuracy`

Specifies the accuracy desired for a linear solution. The default is 10^{-5}.

2.5.6 Spatial error estimation and adaptation parameters

`-computeSpatialError method`

Specifies which method to use for estimating the spatial error. This by itself does not imply that the mesh will be adapted. Use `-maxSpatialAdaptiveSteps` for that. `method` can be

- `erm` for implicit element residual estimator
- `explicit` for explicit residual estimator
- `postpro` for postprocessing estimator

`-maxSpatialAdaptiveSteps maxValue`

Specifies that spatial adaptation is to be done. `maxValue` is the maximum permissible times the mesh will be adapted. (This is OK for static problems. The behavior is undefined for transient problems, as of now.) `maxValue` has to be specified. There is no default value.

Remember that spatial adaptation can be done only if you are using the `-interface` command and not the `-astructure`, `-sstructure` series of commands. Moreover the mesh interface should have the capabilities of adapting the mesh.

`-relativeSpatialTolerance value`

Specifies the tolerance for the relative spatial error. Currently this one value is used for all the fields. The default is 0.1

`-maxSpatialRefinementRatio value`

`-maxSpatialUnrefinementRatio value`

`-elementToleranceRatio value`

Chapter 3

Function and Operator Reference

In the following sections, *variable* refers to a predefined TCL variable holding a value that evaluates to a floating point number, *region* refers to the name of a region, *field* refers to the name of a field, and *function* refers to the name of a command created by the function command.

3.1 Functions

3.1.1 `const`

This implements a constant function. It is present because functions are required in certain places where variables cannot be used.

Parameters: v

v is a variable.

3.1.2 `xCoord`

This function returns the x coordinate of the integration point in an element. Note that the coordinate returned is in *Alamode* mesh units (centimeters), not microns.

3.1.3 `yCoord`

This function returns the y coordinate of the integration point in an element. Note that the coordinate returned is in *Alamode* mesh units (centimeters), not microns.

3.1.4 zCoord

This function returns the z coordinate of the integration point in an element. Note that the coordinate returned is in *Alamode* mesh units (centimeters), not microns.

3.1.5 profile

This function interpolates a 1D profile contained in the specified file onto the mesh. It should not be used as part of a model.

Parameters: d f i

d is an arbitrary function returning values in *Alamode* mesh units.

f is the name of the file containing the profile. The file contains a two column list of distances and values. The distances in the file are assumed to be in microns, and they are mapped into *Alamode* mesh units. A minimum of two distance/value pairs must be present in the file. Comment lines are indicated by a # in the first column.

i is the interpolation used. Is is either `log` or `linear`.

The following TCL proc is provided as part of *Alamode*.

```
proc profile {fld dir file type} {
    set coord $dir
    append coord Coord
    set dist [function $coord -parameters {}]
    set func [function profile -parameters [list $dist $file $type]]
    field $fld -setValue $func
}
```

Thus, the following command reads in a Boron profile from `boron.profile` along the x coordinate axis.

```
profile BoronInSilicon x boron.profile linear
```

3.1.6 box

This function returns 0 or 1, depending if the coordinate lies within the box provided. It can be used to selectively apply a function to a rectangular region. It should not be used in a model.

Parameters: ll ur

ll is the lower left coordinate of the box

ur is the upper right coordinate of the box

The coordinates are given as a list of numbers, indicating the x, y (optional), and z (optional) coordinates of the lower left or upper right. The coordinates are assumed to be given in microns. Points that lie on the box are considered to be within the box, but the user should be aware that because of rounding errors, this command may not always produce the expected results for points that lie exactly on the box.

Example:

```
field PotentialInSilicon -setValue [function box -parameters \
    [list "1.0 0.0 0.0" "5.0 10.0 10.0"]]
```

3.1.7 `scaledField/scaling`

This function scales a field by a constant value.

Parameters: C s

C is a field or function

s is a variable

3.1.8 `abs`

This function takes the absolute value of a field.

Parameters: C

C is a field or function

3.1.9 `min`

This function returns the minimum of two fields or functions.

Parameters: $C1$ $C2$

$C1$ is a field, function, or variable.

$C2$ is a field, function, or variable.

3.1.10 `max`

This function returns the maximum of two fields or functions.

Parameters: $C1$ $C2$

$C1$ is a field, function, or variable.

$C2$ is a field, function, or variable.

3.1.11 `exp`

This is the exponential function (e^C)

Parameters: C

C is a field or function

3.1.12 `log`

This is the natural log function.

Parameters: C

C is a field or function

3.1.13 `arcsinh`

This is the inverse hyperbolic sine function. It is not intended to be used in a model, but it can be used for initialization (e.g. to approximate potential from a net carrier profile).

Parameters: C

C is a field or function

3.1.14 `sqrt`

This is the square root function.

Parameters: C

C is a field or function

3.1.15 `erfc`

This is the complimentary error function:

$$\mathrm{erfc}(x) = 1 - \frac{2}{\pi} \int_0^x e^{-t^2} dt$$

If used with a coordinate based function, be aware that *Alamode*'s internal mesh units are centimeters. This function makes no coordinate scaling adjustments to its argument.

This function should not be used in a model.

Parameters: C

C is a field or function

3.1.16 `sum`

This function provides a scaled general sum of any permutation of two fields and functions.

Parameters: s_1 C_1 s_2 C_2

s_1 is a variable

C_1 is a field or a function

s_2 is a variable

C_2 is a field or a function

3.1.17 `difference`

This function provides a scaled difference of fields and functions $(s_1 C_1 - s_2 C_2)$.

Parameters: s_1 C_1 s_2 C_2

s_1 is a variable

C_1 is a field or a function

s_2 is a variable

C_2 is a field or a function

3.1.18 `times`

This function provides a scaled product of any permutation of two fields and functions.

Parameters: C_1 C_2 s

C_1 is a field or a function

C_2 is a field or a function

s is a variable

3.1.19 `div`

This function provides a scaled quotient of any permutation of two fields and functions (sC_1/C_2).

Parameters: C_1 C_2 s

C_1 is a field or a function

C_2 is a field or a function

s is a variable

3.1.20 `power`

This function provides for exponentiation of a fields or functions (C^x).

Parameters: C x

C is a field or a function

x is a variable

3.1.21 `carrierConc`

This function produces the normalized quasi-fermi carrier concentration given by

$$\frac{n}{n_i} = \frac{C_N}{2n_i} + \sqrt{\left(\frac{C_N}{2n_i}\right)^2 + 1}$$

where C_N is the net active doping. Note that charge of the dopant is not specified, so this function can be used for either n/n_i or p/n_i.

Parameters: C_N n_i

C_N is a field, function, or variable

n_i is a field, function, or variable

3.1.22 `efieldfactor`

This function provides a scaling factor to enhance the diffusivity to account for the electric field without adding a separate electric field flux term. This should only be used for models with n or p dopants, but not both as it produces an effective flux in the incorrect direction if used for a net active with the incorrect sign. The enhancement factor is

$$h = 1 + \frac{\frac{C_N}{2n_i}}{\sqrt{\left(\frac{C_N}{2n_i}\right)^2 + 1}}$$

Parameters: C_N n_i

C_N is a field or function

n_i is a variable

3.1.23 `fermi`

This function produces a fermi-level dependent diffusion coefficient given by

$$D_A = D_{A^0} + D_{A^-}\left(\frac{n}{n_i}\right) + D_{A^=}\left(\frac{n}{n_i}\right)^2$$

Parameters: $\frac{n}{n_i}$ D_{A^0} D_{A^-} $D_{A^=}$

$\frac{n}{n_i}$ is a field or function

D_{A^0} is a variable

D_{A^-} is a variable

$D_{A^=}$ is a variable

3.1.24 `fermi_nnp`

This function produces a fermi-level dependent diffusion coefficient given by

$$D_A = D_{A^0} + D_{A^-}\left(\frac{n}{n_i}\right) + D_{A^=}\left(\frac{n}{n_i}\right)^2 + D_{A^+}\left(\frac{n}{n_i}\right)^{-1}$$

Parameters: $\frac{n}{n_i}$ D_{A^0} D_{A^-} D_{A^+} $D_{A^=}$

$\frac{n}{n_i}$ is a field or function

D_{A^0} is a variable

D_{A^-} is a variable

D_{A^+} is a variable

$D_{A^=}$ is a variable

3.1.25 `ssactive`

This function provides a smooth ceiling function appropriate for capping a profile at a solid solubility level.

Parameters: C C_{max}

C is a field or a function

C_{max} is a variable

3.1.26 `constInjectionRate`

This function provides a constant rate injection from a finite source. It is used in conjunction with `constInjectionLoss`.

Parameters: C r Δt_n

C is the field containing the remaining source quantity

r is a variable containing the maximum constant rate of injection

Δt_n should be the TIME_STEP variable

The actual rate injected depends on rate, the quantity remaining, and the time step. If the quantity remaining is non-zero then the rate of injection is the minimum of the maximum constant rate of injection and the rate to inject the remaining source quantity over the time step.

The following fragment demonstrates the use of these functions.

```
set one 1.0

# Icluster is the finite source field
# rate is the constant rate of injection
set IRate [function constInjectionRate \
        -parameters [list $Icluster rate TIME_STEP] ]

# IReduction and NextIcluster are used to compute and
# update the quantity decreased at a constant rate
set IReduction [function constInjectionLoss \
        -parameters [list $Icluster rate TIME_STEP] ]
set NextIcluster [function difference \
        -parameters [list one $Icluster one $IReduction] ]

# Call this -atEndOfEachTimeStepScript in the model to deplete
# the finite source.
proc updateIcluster {} {
   global Icluster
   global NextIcluster
   global SOLVE_TIME

   field $Icluster -setValue $NextIcluster
}
```

3.1.27 `constInjectionLoss`

This function provides a way to subtract off the quantity depleted from a finite source during a time step. It is used in conjunction with `constInjectionRate`. See `constInjectionRate` for an example.

Parameters: C r Δt_n

C is the field containing the remaining source quantity

r is a variable containing the rate of injection

Δt_n should be the TIME_STEP variable

The actual rate depleted depends on rate, the quantity remaining, and the time step. If the quantity remaining is non-zero then the rate of injection is the minimum of the maximum constant rate of injection and the rate to inject the remaining source quantity over the time step.

3.1.28 `cstar2`

This implements a function of the form

$$C^* = C_{\text{star}} \frac{\text{neu} + \text{neg}\left(\frac{n}{n_i}\right) + \text{dneg}\left(\frac{n}{n_i}\right)^2 + \text{pos}\left(\frac{p}{p_i}\right) + \text{dpos}\left(\frac{p}{p_i}\right)^2}{\text{neu} + \text{neg} + \text{dneg} + \text{pos} + \text{dpos}}$$

which is appropriate for computing an equilibrium interstitial or vacancy population.

Parameters: $\frac{n}{n_i}$ $\frac{p}{p_i}$ C_{star} neu neg dneg pos dpos

$\frac{n}{n_i}$ is a field or function

$\frac{p}{p_i}$ is a field or function

C_{star} is a variable

neu is a variable

neg is a variable

dneg is a variable

pos is a variable

dpos is a variable

3.1.29 `cstar3`

This implements a function of the form

$$C^* = C_{\text{star}} \frac{\text{neu} + \text{neg}\left(\frac{n}{n_i}\right) + \text{dneg}\left(\frac{n}{n_i}\right)^2 + \text{tneg}\left(\frac{n}{n_i}\right)^3 + \text{pos}\left(\frac{p}{p_i}\right) + \text{dpos}\left(\frac{p}{p_i}\right)^2 + \text{tpos}\left(\frac{p}{p_i}\right)^3}{\text{neu} + \text{neg} + \text{dneg} + \text{tneg} + \text{pos} + \text{dpos} + \text{tpos}}$$

which is appropriate for computing an intrinsic interstitial or vacancy population.

Parameters: $\frac{n}{n_i}$ $\frac{p}{p_i}$ C_{star} neu neg dneg tneg pos dpos tpos

$\frac{n}{n_i}$ is a field or function

$\frac{p}{p_i}$ is a field or function

C_{star} is a variable

neu is a variable

neg is a variable

dneg is a variable

tneg is a variable

pos is a variable

dpos is a variable

tpos is a variable

3.2 Operators

Interior quadrature is used for the `cdiffusion` operator. Nodal quadrature is used for the rest of the operators. These defaults can be overridden by `-useNodalQuadrature` or `-useInteriorQuadrature`. It is not recommended to do so, however.

3.2.1 `diffusion`

This implements a discretized diffusion operator

$$\nabla\cdot D\nabla C$$

The discretization decouples obtuse elements at the element level and uses nodal values for D and C.

Parameters: D C

D is either a field, function, or variable

C is a field or function

3.2.2 `gdiffusion`

This implements a discretized diffusion operator

$$\nabla\cdot D\nabla C$$

The discretization decouples obtuse elements at the assembled diffusion matrix level and uses nodal values for D and C. This technique only conserves dose if the diffusivity is constant over a region.

Parameters: D C

D is a variable

C is a field or function

3.2.3 `cdiffusion`

This implements a discretized diffusion operator

$$\nabla \cdot D \nabla C$$

The discretization is a consistent finite element discretization. It uses interior quadrature and evaluates D and C at the quadrature points within an element. This operator must be used for spatial adaptation.

Parameters: D C

D is either a field, function, or variable

C is a field or function

3.2.4 `simplerecombination`

This implements a discretized simple recombination operator

$$Kr\,(C - C^*)$$

Parameters: C k C^*

C is a field or function

k is a field, function, or variable

C^* is a field, function, or variable

3.2.5 `bimole`

This implements a discretized bimolecular recombination operator

$$Kr\,(IV - I^* V^*)$$

Parameters: $I\ V\ Kr\ I^*\ V^*$

I is a field, function, or variable

V is a field, function, or variable

Kr is a field, function, or variable

I^* is a field, function, or variable

V^* is a field, function, or variable

3.2.6 `k1fg_k2h`

This operator implements a first order chemical reaction of the form $F + G \Longleftrightarrow H$.

$$k_1 f g - k_2 h$$

Parameters: $f\ g\ h\ k_1\ k_2$

f is the field, function, or variable

g is the field, function, or variable

h is the field, function, or variable

k_1 is a variable

k_2 is a variable

3.2.7 `segregation`

This operator implements a segregation transport flux across an interface.

$$-\vec{n} \cdot D\nabla C = Tr\,(C_1 - C_2/m)$$

Parameters: $C_1\ C_2\ Tr\ m$

C_1 is a field, function, or variable

C_2 is a field, function, or variable

Tr is a variable

m is a variable

Note that both sides of an interface should have the same segregation flux, but with opposite signs. It is recommended that the same form be used for both, but with different `-solvesFor` and `-sign`.

A useful rule for determining the sign of each operator is that the sign should be chosen to reduce the `-solvesFor` field if the `-solvesFor` field increases.

3.2.8 `surfacerecombination`

This operator implements a recombination flux of the form

$$-\vec{n} \cdot D\nabla C = k\left(C - C^*\right)$$

Parameters: C k C^*

C is a field, function, or variable

k is a field, function, or variable

C^* is a field, function, or variable

A useful rule for determining the sign of each operator is that the sign should be chosen to reduce the `-solvesFor` field if the `-solvesFor` field increases.

3.2.9 `quasiFermiEField`

This operator is no longer available. It was replaced by quasiNeutralEField, but there are some differences. This documentation is still present to show the differences

This operator implements an electric field flux of the form

$$\nabla \cdot DC\nabla \ln\left(\frac{n}{n_i}\right)$$

Parameters: C D $\frac{n}{n_i}$

C is a field, function, or variable for the mobile concentration

D is a field, function, or variable for the diffusivity

$\frac{n}{n_i}$ is a field, function, or variable approximating the normalized carrier concentration

3.2.10 `quasiNeutralEField`

This operator implements an electric field flux of the form

$$\nabla \cdot DC \nabla \psi$$

where ψ is calculated using a quasi-neutral assumption

$$\psi = \sinh^{-1} (\mathrm{netActive}/2n_i)$$

Parameters: C D sNA

C is a field, function, or variable for the mobile concentration

D is a field, function, or variable for the diffusivity

sNA is the scaled net active dopant concentration $(\mathrm{netActive}/2n_i)$.

3.2.11 `rhsFunction/functionTerm`

This operator lets a function be a term in the equation. It uses nodal quadrature by default.

Parameters: f

f is a field or a function

Chapter 4

Discretizations and Numerical Algorithms

Alamode uses a semi-discrete finite element approach to discretize the parabolic partial differential equations. The equations are discretized in space using a finite element approach, resulting in a system of nonlinear ordinary differential equations. The system is integrated using implicit techniques appropriate for stiff ODEs. The nonlinear systems of equations arising from the integration of the ODEs are solved using Newton techniques.

4.1 Spatial Discretization

In a semi-discrete finite element approach, space is discretized using a finite element method, resulting in a system of ODEs. This technique is illustrated using the following multi-dimensional diffusion initial boundary-value problem (IBVP). The diffusion coefficient is assumed to be a function of the impurity concentration.

The concentration of the impurity is an unknown function of space and time ($C = C(\vec{x}, t)$) on a domain ($\Omega\times]0, T[$). The strong form (S) of the PDE model is

$$\begin{aligned} C_{,t} &= \nabla\cdot(D(C)\nabla C) \\ C(\vec{x}) &= g(\vec{x}, t) \text{ on } \Gamma_g \\ \vec{n}\cdot D(C)\nabla C &= h(\vec{x}, t, C) \text{ on } \Gamma_h \\ C(\vec{x}, t=0) &= C_0(\vec{x}) \end{aligned} \tag{4.1}$$

Note that it is not necessary to have a Dirichlet boundary ($\Gamma_g = \emptyset$) for well-posedness. Also, for notational simplicity, $D(C)$ will usually be given as D, even though it depends on C. For the present, we assume that the domain and boundary conditions are fixed (i.e. Ω does not change over time nor do the the boundaries where Γ_g and Γ_h are applied).

This PDE model will be transformed to an equivalent semi-discrete weak form of the problem by multiplying by a spatially varying weighting function and integrating over the domain (Ω). Two classes of functions will be introduced. The first represents possible solutions (trial functions) for the concentration, and the strong form requires that this function space must be restricted by $C(\vec{x}) = g(\vec{x})$ on Γ_g. The second represents allowable functions for the weighting space (test functions), and it is restricted by the homogeneous counterpart of the trial space ($w(\vec{x}) = 0$ on Γ_g). Continuity constraints in the weak form will add additional restrictions to these function spaces. The transformation from strong to weak form proceeds as follows:

$$\begin{aligned}\int_\Omega wC_{,t}d\Omega = \int_\Omega w\nabla\cdot(D\nabla C)d\Omega &= \int_\Omega w\left(DC_{,i}\right)_{,i} d\Omega \\ &= -\int_\Omega w_{,i}DC_{,i}d\Omega + \int_\Omega \left(wDC_{,i}\right)_{,i} d\Omega \\ &= -\int_\Omega w_{,i}DC_{,i}d\Omega + \int_\Gamma n_i wDC_{,i}d\Gamma\end{aligned} \tag{4.2}$$

Note the use of Einstein notation ($C_{,i}$) for taking derivatives with respect to an arbitrary number of spatial dimensions.

The trial and test function spaces are constrained as a result of this weak form. Because of the first order derivatives in the weak form, both the test and trial spaces must be first order continuous (i.e. C^0) over the domain. Thus, the test and trial spaces are

$$\mathcal{S} = \left\{C(\vec{x},t) \in C^0(\Omega\times]0,T[) \mid C(\vec{x},t) = g(\vec{x},t) \text{ on } \Gamma_g\right\} \tag{4.3}$$

$$\mathcal{V} = \left\{w(\vec{x}) \in C^0(\Omega) \mid w(\vec{x}) = 0 \text{ on } \Gamma_g\right\} \tag{4.4}$$

The weak form (W) of the problem follows directly. Find $C(\vec{x},t) \in \mathcal{S}$ such that, for all $w(\vec{x}) \in \mathcal{V}$,

$$\int_\Omega wC_{,t}d\Omega = -\int_\Omega w_{,i}DC_{,i}d\Omega + \int_{\Gamma_h} wh(\vec{x},t,C)\,d\Gamma \tag{4.5}$$

A mesh is introduced consisting of a set of elements (ϵ) and a set of nodes (η). The mesh is a simple tessellation of the domain (i.e. $\bar{\Omega} = \cup_{e\in\epsilon}\Omega^e$ and $\bar{\Omega}^a \cap \bar{\Omega}^b = \emptyset$ for $\{a, b \in \epsilon$ and $a \neq b\}$). The trial and test function spaces are restricted to interpolations on this mesh (i.e. $C(\vec{x},t) \in \mathcal{S}^h \subset \mathcal{S}$ and $w(\vec{x}) \in \mathcal{V}^h \subset \mathcal{V}$). Note that introducing this interpolation may violate $\mathcal{S}^h \subset \mathcal{S}$, as the interpolation is not exact, but $\mathcal{S}^h$ is assumed to match $\mathcal{S}$ as closely as possible along the prescribed boundary to minimize the impact of this violation. The Galerkin FEM approximation (G) of the weak form follows. Find $C^h(\vec{x},t) \in \mathcal{S}^h$ such that, for all $w^h(\vec{x}) \in \mathcal{V}^h$,

$$\int_\Omega w^h C^h_{,t}d\Omega = -\int_\Omega w^h_{,i}D\left(C^h\right)C^h_{,i}d\Omega + \int_{\Gamma_h} w^h h\left(\vec{x},t,C^h\right)d\Gamma \tag{4.6}$$

Choose a basis for the set of interpolation functions on the mesh such that the value of a function in this basis is 1 at the node it is associated with ($N_A(\vec{x}_A) = 1, A \in \eta$) and 0 at all other nodes ($N_A(\vec{x}_B) = 0$ for $A \neq B$). Because the Galerkin approximation must hold for all $w^h(\vec{x}) \in \mathcal{V}^h$, let w^h consist of all linear combinations of functions in $\mathcal{V}^h$. That is,

$$w^h(\vec{x}) = \sum_A w_A N_A(\vec{x}) \tag{4.7}$$

where the w_A's are unconstrained except for homogeneity constraint on $\mathcal{V}^h$ which requires that $w_A = 0$ for nodes on the dirichlet boundary ($A \in \eta_g$). The trial solutions will be drawn from the same basis, leading to

$$C^h(\vec{x}, t) = \sum_A c_A(t) N_A(\vec{x}) \tag{4.8}$$

where $c_A(t) = g(\vec{x}_A, t)$ for nodes on the dirichlet boundary. Finally, note that this set of basis functions is only spatially varying. This leads to the following form for $C^h_{,t}$.

$$C^h_{,t}(\vec{x}, t) = \sum_A \dot{c}_A(t) N_A(\vec{x}) \tag{4.9}$$

Replacing the appropriate functions in (G) will produce a nonlinear system of ordinary differential equations.

$$\int_\Omega \sum_A w_A N_A \sum_B \dot{c}_B N_B d\Omega = -\int_\Omega \sum_A w_A N_{A,i} D\left(C^h\right) \sum_B c_B N_{B,i} d\Omega + \int_{\Gamma_h} \sum_A w_A N_A h\left(\vec{x}, t, C^h\right) d\Gamma \tag{4.10}$$

Linearity of integration allows the exchange of the integration operator with the summation operator, giving

$$\sum_A w_A \int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega = -\sum_A w_A \int_\Omega N_{A,i} D\left(C^h\right) \sum_B c_B N_{B,i} d\Omega + \sum_A w_A \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma \tag{4.11}$$

or

$$\sum_A w_A \underbrace{\left(\int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega - R_A\right)}_{G_A} = 0 \tag{4.12}$$

where

$$R_A = -\int_\Omega N_{A,i} D\left(C^h\right) \sum_B c_B N_{B,i} d\Omega + \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma \tag{4.13}$$

Because the w_A's are unconstrained (for $A \in \eta - \eta_g$), each corresponding G_A must be 0.

$$G_A = \int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega + \int_\Omega N_{A,i} D\left(C^h\right) \sum_B c_B N_{B,i} d\Omega - \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma = 0 \tag{4.14}$$

for $A \in \eta - \eta_g$.

At this point, the use of Einstein notation becomes cumbersome and confusing so we return to the gradient notation. This equation is

$$G_A = \int_\Omega N_A \sum_B \dot{c}_B N_B d\Omega + \int_\Omega \nabla N_A \cdot D\left(C^h\right) \nabla C^h d\Omega - \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma = 0 \tag{4.15}$$

or, with $\mathbf{c} = \{c_B\}$ and $\mathbf{R} = \{R_A\}$, 4.15 can be rewritten as

$$\begin{aligned} \mathbf{M}\dot{\mathbf{c}} &= -\int_{\Omega} \nabla N_A \cdot D\left(C^h\right) \nabla C^h d\Omega + \int_{\Gamma_h} N_A h\left(\vec{x}, t, C^h\right) d\Gamma \\ &= \mathbf{R}\left(\mathbf{c}\right) \end{aligned} \tag{4.16}$$

This is a system of nonlinear ordinary differential equations which can be solved using ODE techniques.

4.2 Time integration algorithms

The transient solution of 4.16 will be considered next. The initial condition ($\mathbf{c}\,(t=0) = \mathbf{c}_0$) can be computed from the given $C_0\,(\vec{x}, t=0)$. As is the case with most ODE integration techniques, time is discretized on intervals given by of $\Delta t_n = t_{n+1} - t_n$.

4.2.1 Generalized Trapezoid Family

A generalized trapezoid family of methods [2] is one approach available for time integration. This method consists of the following equations.

$$\mathbf{M}\mathbf{v}_{n+1} = \mathbf{R}\left(\mathbf{c}_{n+1}\right) \tag{4.17}$$

$$\mathbf{c}_{n+1} = \mathbf{c}_n + \Delta t_n \mathbf{v}_{n+\alpha} \tag{4.18}$$

$$\mathbf{v}_{n+\alpha} = (1-\alpha)\mathbf{v}_n + \alpha \mathbf{v}_{n+1} \tag{4.19}$$

where $\mathbf{c}_n$ and $\mathbf{v}_n$ are approximations to $\mathbf{c}\,(t_n)$ and $\dot{\mathbf{c}}\,(t_n)$, respectively. The parameter α is taken to be in the interval $[0,1]$. Equations 4.17-4.19 can be solved either for concentrations or rates.

Concentration form

In the concentration implementation, we define a predictor for $\mathbf{c}_{n+1}$ as

$$\tilde{\mathbf{c}}_{n+1} = \mathbf{c}_n + (1-\alpha)\Delta t_n \mathbf{v}_n \tag{4.20}$$

and combine this with 4.18 and 4.19 to get

$$\mathbf{c}_{n+1} = \tilde{\mathbf{c}}_{n+1} + \alpha \Delta t_n \mathbf{v}_{n+1} \tag{4.21}$$

or

$$\mathbf{v}_{n+1} = \frac{\mathbf{c}_{n+1} - \tilde{\mathbf{c}}_{n+1}}{\alpha \Delta t_n} \tag{4.22}$$

This equation is substituted into 4.17 resulting in the following algebraic system of nonlinear equations.

$$\frac{1}{\alpha \Delta t_n} \mathbf{M} \left(\mathbf{c}_{n+1} - \tilde{\mathbf{c}}_{n+1} \right) = \mathbf{R} \left(\mathbf{c}_{n+1} \right) \tag{4.23}$$

Thus, the nonlinear system becomes

$$\mathbf{R}' \left(\mathbf{c}_{n+1} \right) = \mathbf{R} \left(\mathbf{c}_{n+1} \right) - \frac{1}{\alpha \Delta t_n} \mathbf{M} \mathbf{c}_{n+1} + \frac{1}{\alpha \Delta t_n} \mathbf{M} \tilde{\mathbf{c}}_{\mathbf{n+1}} = \mathbf{0} \tag{4.24}$$

The rates can be computed by rewriting 4.19,

$$\mathbf{v}_{n+1} = \frac{\mathbf{c}_{n+1} - \tilde{\mathbf{c}}_{n+1}}{\alpha \Delta t_n} \tag{4.25}$$

Rate form

Again, we define a predictor for $\mathbf{c}_{n+1}$ as

$$\tilde{\mathbf{c}}_{n+1} = \mathbf{c}_n + (1 - \alpha) \Delta t_n \mathbf{v}_n \tag{4.26}$$

If this is combined with 4.18 and 4.19, we obtain

$$\mathbf{c}_{n+1} = \tilde{\mathbf{c}}_{n+1} + \alpha \Delta t_n \mathbf{v}_{n+1} \tag{4.27}$$

If this is substituted into 4.17, we obtain an equation that will solve for the rates.

$$\mathbf{R}' \left(\mathbf{v}_{n+t} \right) = \mathbf{R} \left(\tilde{\mathbf{c}}_{n+1} + \alpha \Delta t_n \mathbf{v}_{n+1} \right) - \mathbf{M} \mathbf{v}_{n+1} = \mathbf{0} \tag{4.28}$$

Remarks on Generalized Trapezoid

The generalized trapezoid algorithm provides access to three specific timestepping methods depending on the value of α: forward Euler ($\alpha = 0$), trapezoid rule ($\alpha = \frac{1}{2}$), and backward Euler ($\alpha = 1$).

- Any method with $\alpha < \frac{1}{2}$ is conditionally A-stable; time-steps are constrained by the maximum eigenvalue of the discretized system. Arbitrarily large time-steps will produce an amplification matrix with entries larger than 1 in magnitude. This can lead to incorrect exponential growth in the solution.

- The trapezoid rule ($\alpha = \frac{1}{2}$) is second order accurate in time. All other values of α produce first order methods.

- Backward Euler ($\alpha = 1$) is the only method that is both A-stable and L-stable. With $\alpha = 1$, the entries in the amplification matrix are bounded by $[0, 1]$ (L-stable). Other values of α ($\alpha \geq \frac{1}{2}$) can produce amplification matrices with values near -1 leading to solutions with oscillatory behavior.

4.2.2 TR/BDF2

Although backward Euler has properties appropriate for robustly integrating the stiff systems of ODEs arising from semiconductor diffusion models, it is only first-order accurate. The TR/BDF2 method, discussed in [1], provides a second order accurate A-stable and L-stable single step method comprised of a trapezoid step over part of the time-step followed by a second order backward difference step over the remainder of the time step.

Let $0 < \gamma \leq 1$ and $\mathbf{c}_{n+\gamma}$ be the computed solution at $t_n + \gamma\Delta t_n$. The TR substep goes from t_n to $t_n + \gamma\Delta t_n$ via

$$\mathbf{R}'\left(\mathbf{c}_{n+\gamma}\right) = \mathbf{R}\left(\mathbf{c}_{n+\gamma}\right) - \frac{2}{\gamma\Delta t_n}\mathbf{M}\mathbf{c}_{n+\gamma} + \frac{2}{\gamma\Delta t_n}\mathbf{M}\mathbf{c}_n + \mathbf{R}\left(\mathbf{c}_n\right) = \mathbf{0} \tag{4.29}$$

The BDF2 substep goes from $t_{n+\gamma}$ to $t_{n+1} = t_n + \Delta t_n$ via

$$\mathbf{R}'\left(\mathbf{c}_{n+1}\right) = \mathbf{R}\left(\mathbf{c}_{n+1}\right) - \frac{(2-\gamma)}{(1-\gamma)\,\Delta t_n}\mathbf{M}\mathbf{c}_{n+1} + \frac{1}{\gamma\,(1-\gamma)\,\Delta t_n}\mathbf{M}\mathbf{c}_{n+\gamma} - \frac{(1-\gamma)^2}{\gamma\,(1-\gamma)\,\Delta t_n}\mathbf{M}\mathbf{c}_n = \mathbf{0} \tag{4.30}$$

The constant multiplying the truncation error is minimized if $\gamma = 2-\sqrt{2}$. This value of gamma also produces identical jacobians for both the TR and BDF2 steps.

The local truncation error (LTE) can be used to estimate a reasonable next time step. One approach is to use the divided difference formula

$$\tau_{n+1} = 2C\Delta t_n \left|\gamma^{-1}\mathbf{R}\left(\mathbf{c}_n\right) - \gamma^{-1}\left(1-\gamma\right)^{-1}\mathbf{R}\left(\mathbf{c}_{n+\gamma}\right) + \left(1-\gamma\right)^{-1}\mathbf{R}\left(\mathbf{c}_{n+1}\right)\right| \tag{4.31}$$

is used to estimate the LTE. Note that C is the constant multiplying the highest order truncation term in the method

$$C = \frac{-3\gamma^2 + 4\gamma - 2}{12(2-\gamma)} \tag{4.32}$$

The LTE is compared to an allowable error tolerance to produce a RMS measure of the normalized pointwise error. Let

$$\mathbf{q}_{n+1} = \mathbf{M}\mathbf{c}_{n+1} \tag{4.33}$$

and

$$e_{n+1,i} = \epsilon_R|q_{n+1,i}| + \epsilon_A \tag{4.34}$$

Then, the RMS measure of the normalized pointwise error is given by

$$r^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\tau_{n+1,i}}{e_{n+1,i}}\right)^2 \tag{4.35}$$

A candidate next time step is chosen as

$$\Delta t_{n+1}{}^{*} = \Delta t_n r^{-1/3} \tag{4.36}$$

If r is reasonably small (say less than 2), the error in the timestep is assumed to be acceptable and the next time step is chosen to be $\min\left(\Delta t_{n+1}{}^{*}, 2\Delta t_n\right)$. The 2 times cap on increases is to avoid stepsize oscillations. If r is too large, the current timestep is repeated with $\Delta t_n = 0.7\Delta t_{n+1}{}^{*}$.

A second timestep estimator is based on a LTE estimate computed using a Milne's device. A second TR step is taken from $t = t_n + \gamma \Delta t_n$ to t_{n+1}. The solutions of the BDF2 step and the second TR step are compared and used to estimate the truncation error. See [4] for more details.

4.3 Nonlinear Solution Algorithms

A variant of Newton's method is used to solve the resulting system of nonlinear equations. Newton's method solves a system of nonlinear equations, $\mathbf{R}'(\mathbf{c}) = \mathbf{0}$, by successively applying updates in the direction of

$$\Delta \mathbf{c}^{k+1} = -\left(\mathbf{T}\left(\mathbf{c}^k\right)\right)^{-1} \mathbf{R}'\left(\mathbf{c}^k\right) \tag{4.37}$$

This involves calculation of a residual, which can be calculated directly from any of 4.24, 4.28, 4.29, and 4.30, and an approximation to the tangent ($\mathbf{T}$). The best approximation to the tangent is the Frechet derivative,

$$T_{AB} = \frac{\partial R'_A}{\partial c_B} \tag{4.38}$$

For the nonlinear systems given by 4.24, 4.29, and 4.30, the tangent matrix is simply the sum of tangent of the static residual, $\mathbf{R}(\mathbf{c})$, and a scaled mass matrix.

$$T_{AB} = \frac{\partial R_A}{\partial c_B} + sM_{AB} \tag{4.39}$$

Convergence of Newton methods can be determined using a variety of approaches. One method measures the magnitude of residual. The nonlinear method has converged if

$$\frac{\left|\left|\mathbf{R}\left(\mathbf{c}^{k+1}\right)\right|\right|}{\left|\left|\mathbf{R}\left(\mathbf{c}^{0}\right)\right|\right|} < \epsilon_R \tag{4.40}$$

or

$$\left|\left|\mathbf{R}\left(\mathbf{c}^{k+1}\right)\right|\right| < \epsilon_A \tag{4.41}$$

Another convergence criteria measures the magnitude of the relative change in the full update. The nonlinear method has converged if

$$\left(\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\Delta \mathbf{c}_{,i}^{k+1}}{\epsilon_R \mathbf{c}_{,i}^{k+1} + \epsilon_A}\right)^2\right)^{\frac{1}{2}} < 1 \tag{4.42}$$

4.4 Overcoming Numerical Difficulties

The defaults for the numerical techniques in *Alamode* were chosen to provide a reasonably robust and accurate solution to the limited set of problems that we have tried. They are not optimal for every arbitrary system of equations that can be composed out of the available operators. The following sections discuss the characteristics of some of the numerical difficulties and provides hints about which parameters should be adjusted to help to overcome them.

4.4.1 Divergence or failure to converge in the nonlinear solution

Potential causes of divergence in the nonlinear solution include:

- taking too large of a time step. The initial guess is outside of the region of convergence for the Newton method.
- an incorrect system of equations.

Suggestions:

- Try taking a smaller initial time step or use `-estimateInitialTimeStep`.
- Turn on `-useConvergeHeuristics`.
- Carefully validate that the model is implemented correctly and that the systems of equations are well-posed. Too few boundary conditions (i.e. an underconstrained system of equations) can also lead to wandering and divergence.
- Try turning on `-applyDamping`. The damped Newton will likely converge very slowly, and this should be used only as a last resort.

4.4.2 Residuals "bottom out"

The bottoming out may be due to:

- normal truncation error in finite precision arithmetic. If this is the case, the update norm (equation 4.42) will usually provide a reasonable convergence condition, but the tolerances used to compute that norm may have to be relaxed for some fields (section 2.3.4).
- an incorrect system of equations. Too many boundary conditions may lead to an overconstrained system of nonlinear equations in which you can only hope to minimize the residual, not reduce it to zero.

- if the bottoming out occurs along with threshold adjustments, this may be an indication of sign flips inherent in the timestepping algorithm. Approaches to overcoming this problem are discussed in section 4.5.

4.5 Threshold violations

The threshold violation warnings are meant to point out potential problems with the solution. They may not actually indicate a serious problem with the solution. *Alamode* checks for violations by default because it is likely that all of the unknowns represent concentrations, and it is not physical to have negative concentrations. Thresholds are adjusted after a nonlinear update because negatives can cause nonlinear convergence problems.

Threshold violation warnings that are printed can usually be ignored if they do not occur on the last nonlinear update. These are just overshoot in the nonlinear update, and they do not indicate a problem with the converged solution.

Threshold violation warnings/adjustments that occur on the last nonlinear update may be an indication of a problem with the converged solution. Also, if the violations are large and they were adjusted, conservation of dose is no longer guaranteed. For this reason, `-failIfHadThresholdViolation` is on by default.

Suggestions:

- Check the magnitude, field, and location of the violations (turn on `-printThresholdViolations`). This can generate a lot of output because every nodal violation is printed.
- Change the warn threshold for offending fields (`-warnThreshold`). If the solution of any field can go negative, this should be set to a large negative value along with `-thresholdValue` for that field. Concentration fields that dissolve entirely (e.g. clusters) should have a small negative value for the warn threshold to avoid numerical noise being falsely interpreted as a failure criteria.

Threshold violations that occur with bottoming out of residual and/or update norms in the Newton solve may be due to taking too large of a timestep in either the TR or BDF2 step. Neither substep is monotonic, and a consistent application of the algorithm to an ODE system may lead to oscillating signs. The cause of these oscillations can be seen by applying TR/BDF2 to the test ODE

$$\frac{\partial C}{\partial t} = -\lambda C \tag{4.43}$$

where $\lambda > 0$. Let $z = \Delta t \lambda$. The TR step is

$$C_{n+\gamma} = K_{\mathrm{TR}}(\gamma, z) C_n \tag{4.44}$$

where $K_{\mathrm{TR}}(\gamma, z) = (2 - \gamma z)/(2 + \gamma z)$ and $-1 < K_{\mathrm{TR}}(\gamma, z) < 1$. Thus, the TR sign change comes from taking a large timestep compared to the eigenvalues of the system before the modes associated with the large eigenvalues have decayed enough. The BDF2 step is

$$C_{n+1} = K_{\mathrm{BDF2}}(\gamma, z)(C_{n+\gamma} - (1-\gamma)^2 C_n) \tag{4.45}$$

where $K_{\mathrm{BDF2}}(\gamma, z) > 0$. If the concentration has been reduced by significant fraction over the TR step ($(1-\gamma)^2 C_n > C_{n+\gamma}$), the sign on C will change.

These sign changes are a result of the consistent application of the timestepping algorithm. Adjust negative values in the nonlinear solve will not lead to convergence, because the converged values for the consistent timestep equations should be negative. Although this situation will fix itself through nonlinear failures and repeating integration over the time step interval with a smaller time step, the failures are not a productive use of computing resources.

Suggestions for dealing with threshold violations arising from the sign change conditions in TR/BDF2 include:

- Let the field's concentration go significantly negative (e.g. -10^{10}). For fields representing cluster concentrations, this often does not cause problems for the nonlinear solutions (the `field` commands `-warnThreshold` and `-thresholdValue` are used to set threshold detection and adjustment parameters).
- Tighten the timestepping tolerance associated with field controlling the time step (`-relTSTol`).
- Add `-avoidBDF2DecreaseViolations` to the model to detect the reduction condition after the TR substep and estimate a new timestep that should not cause nonlinear failure during BDF2. The compute time spent on the TR substep is wasted, but there were usually fewer nonlinear iterations spent computing that than would be taken before giving up during the BDF2 substep. This approach is justified based on analysis of the method for the test equation and is not guaranteed to avoid all failures in BDF2.
- Use a non-zero offset field to hide large relative reductions the equivalent zero-offset field. The `-offsetVariable` (2.3.4) is provided to ensure that the equivalent zero-offset field's values will in computing nonlinear convergence norms and temporal local truncation error estimates that are used to control the timestep. For example, a cluster equation given by

$$\frac{\partial C}{\partial t} = (K_f I - K_r)\, C \tag{4.46}$$

  could be rewritten in terms of an offset field ($O = C + o$, where o is the offset which would typically be 10^5 to 10^{10}) as

$$\frac{\partial O}{\partial t} = (K_f I - K_r)\, C = (K_f I - K_r)\,(O - o) \tag{4.47}$$

The offset value should not be too large as that will lead to problems related to floating point truncation error in the nonlinear update and temporal LTE norm calculations based on the zero-offset field.

- Use an alternative timestep algorithm. Backward Euler (`-timeIntegrator backwardEuler`) is a first-order method that does not appear to have a structural sign change. Since it is a first-order method, it will likely require many more time steps to achieve the same accuracy as TR/BDF2.

Chapter 5

Simple 1D Structure File Format

Alamode assumes that coordinates in the file are in microns and scales them internally to be in centimeters. This allows the same units for diffusivities to as for SUPREM.

```
################################################################################
#
# Proposal for a simple 1D file format for use with Alamode
#
# Martin Giles    Intel Corporation   June 1996
# Revision 0.1
#
#
# All lines beginning with a "#" character are comments and will be ignored.
#
# Comment lines and empty lines (spaces and tabs only) can appear anywhere
# in the file.
#
# The maximum allowable line length is 511 characters.
#
# Each line in the file consists of a key character followed by arguments.
# Entries must appear in the order shown in this example file.
# Whitespace can be any combination of spaces and tabs.  String-like arguments
# are always enclosed in double quotes and may contain whitespace.
#
################################################################################
#
# The first entry in the file describes the file format version.  This will
# allow us to recognize old files after the format changes.
#
```

```
# v versionnumber
#
v 1
#
##############################################################################
#
# The next entry allows us to store descriptive strings for this file.  These
# may be ignored or, hopefully, used in output logs and the results files.
#
# s argstring
#
s "Example 1D input file format"
s "Martin Giles     6/27/96"
s "There is no limit to how many of these there can be, but I expect that"
s "most programs will ignore all but the first one or two"
#
##############################################################################
#
# The next entries in the file list the field names.  Names must eventually
# match names used in the Alamode equation script.  It would be best to use
# standard names for the variables but it is not mandatory.  Standard names do
# not contain any whitespace.  We will add to the standard list as necessary.
# Capitalization in the names is significant.
#
# Initial standard field name list for diffusion problems:
#
#    Antimony       ActiveAntimony     SBIClusters
#    Arsenic        ActiveArsenic      ASIClusters
#    Boron          ActiveBoron        BIClusters
#    Indium         ActiveIndium       INIClusters
#    Phosphorus     ActivePhosphorus   PIClusters
#    Interstitials  311Defects
#    Vacancies
#    Defect
#    Traps
#    NetDoping
#    Potential
#
# f fieldname
#
f "Boron"
f "ActiveBoron"
```

```
f "Arsenic"
f "ActiveArsenic"
#
##############################################################################
#
# The input is divided into regions of a single material.  As with fields,
# some conventional material names are defined to help keep scripts
# consistent.  Regions must be listed in order of increasing x, with
# the lowest x value assumed to be the exposed wafer surface.
#
# Initial standard material name list for diffusion problems:
#
#    Silicon
#    SiliconDioxide
#    SiliconNitride
#    SiliconOxyNitride
#    PolySilicon
#    Aluminum
#    Titanium
#    TitaniumSilicide
#    TitaniumNitride
#
# r regionname
#
r "PolySilicon"
#
##############################################################################
#
# For each region the x locations of the nodes in that region must be listed
# in depth order.  For each node, solution values for all fields must be
# provided.
#
# x xlocation field0val field1val field2val ...
#
x 0.0 1.0e18 1.0e18 1.0e21 2.0e20
x 0.1 1.0e18 1.0e18 9.0e20 1.8e20
x 0.2 1.0e18 1.0e18 6.0e20 1.2e20
x 0.3 1.0e18 1.0e18 3.0e20 8.0e19
#
##############################################################################
#
# Regions are listed in sequence, and must start at the same x location
```

```
# that the previous region ended at.  Touching regions will usually
# be of different materials, but this is not required.
#
r "Silicon"
x 0.3 7.0e17 7.0e17 2.0e20 7.0e19
x 0.5 7.0e17 7.0e17 1.0e20 6.0e19
x 1.0 7.0e17 7.0e17 5.0e19 5.0e19

r "SiliconDioxide"
x 1.0 0.0 0.0 0.0 0.0
x 1.5 0.0 0.0 0.0 0.0
x 2.0 0.0 0.0 0.0 0.0

#
#############################################################################
#
# If there is other simulator-specific information that needs to be stored,
# then private data commands can be used.  These are ignored by any
# simulator that does not recognize the source.  Private commands
# can appear anywhere in the input file.
#
# p simulatoridstring private information ......
#
p "Intel Suprem4 2.11" a 2 37.7
p "Intel Suprem4 2.11" a 3 1035 111
p "Intel Suprem4 2.11" other junk here
#
#############################################################################
#
# There is no special end-of-file marker.
#
#############################################################################
```

Chapter 6

SUPREM Structure File Format

This documentation was provided by Dr. Goodwin Chin. It has not been checked for accuracy. It is know to be out of date with respect to the field and materials in SUPREM-IV.GS.

```
All lines begin with a one character code.  After this character
is parsed, subsequent characters are parsed based on the
character code.  The currently used single letter codes are as
follows:

v - Version number
(e.g. v SUPREM-IV 9130)

D - dimensionality (only for versions 9130 or greater)
valid values are 1 and 2
(e.g. D 2)

c - coordinate (coordinates start counting from 1)
the format of this line is as follows:
c coordinate_number x.value [y.value] [z.value] [spacing]
note that the units are microns

y.value and z.value are only valid for D=2, D=3,
respectively

the spacing value was parsed in versions A.9030 and older
but never used

(e.g. c 1 1.0 0.5 0)
```

```
In the SUPREM-IV coordinate system, into the silicon is
positive (y-axis).  The x-axis values are
numbered from left to right.

e - edge (edges start counting from 1)
while not used by SUPREM-IV (it generates it's own edge
tables from scratch) it was included as output
for versions BEFORE 9130.

the format of this line is as follows:
e edge_number coordinate.number coordinate.number boundary_code

the boundary_code value was never used

(e.g. e 1 2 3 0)

r - region (regions start counting from 1)
multiple regions of the same material can exist in SUPREM-IV
so be careful

the format of this line is as folows:
r region_number material_number

material_number is a special integer code used in SUPREM-IV.
While this can be changed easily by modification of
include/material.h, the material codes have remained fairly
stable through the releases with only possible additions
in newer versions for new materials.  Here is the mapping
from releases (A.8912, A.9030, and 9130).


Material Integer Code
--------------------
Gas 0
Silicon Dioxide 1
Nitride 2
Si 3
Polysilicon 4
Oxinitride 5
Al 6
Photoresist 7
```

b - boundary (boundary)
this code traces the boundary for each region. after each region specification (r), the next lines comprise the boundary of the region, until the presence of the next non-b command. The boundary is traced in a clockwise fashion. As with the e code, the b code has been elimiated and is only included for versions BEFORE 9130.

the format of this line is as follows:
```
b edge_number
```

BEWARE. there is nothing that states that the edge are directed properly. Edge cannot be directed properly since for structures such as the interface for the gate oxide in a MOS device traversal of the silicon and oxide polygons would require that the edges point in different directions.

t - elements (triangles) (triangles start counting from 1)
this one is complicated so be careful. SUPREM-IV has a lot of internal smarts and rebuilds neighbor information from the nodal description and does not rely on the neighbor information dumped out. The neighbor information is usually correct though.

the format of this line is as follows:
```
t triangle_number region_number \  (no new line)
coordinate_number coordinate_number coordinate_number \
neighbor_triangle_number neighbor_triangle_number \
neighbor_triangle_number father_triangle \
first_offspring_triangle
```

region_number is the same as those used in the r line.
coordinate_number are the same as those used in the c lines.

the concept of neighbors is somewhat more difficult to explain. given a triangle with vertices a, b, c, edge a is the edge OPPOSITE the vertex (i.e. edge a containes vertices b and c). see below.

a

```
     x x
    x   x
            edge c x     x  edge b
                  x       x
                 x         x
                bxxxxxxxxxxxc

  edge  a
```

now since edges aren't used in SUPREM-IV, the number of the
neighbor trangle containg the edge in question is given
in leau of the edge number. these triangle numbers
correspond to the ones now used in the t-list.
of course the problem is what do we do when the
triangle in question does not exist (e.g. at the
surface and sides of a wafer). In this case a
dummy code is given describing the boundary condition
used. The codes currently supported by SURPEM-IV
are

-1024 : reflecting boundary (von Neumann)
-1022 : exposed surface (Dirichlet)
-1023 : backside (version 9130 only)

the concept of father and offspring triangles are only
supported in version 9130. In most cases the
value is -1 to denote the absence of parents
and children. the intent is to provide a hook
for refinement and unrefinement of triangles as
done in PISCES.

s - solution information
the purpose of this line is to define the solution varialbes
available from the simulation. the format of the
line is as follows:

s number_of_solutions solution_1 solution_2 .... solution_number_of_solutions
the integer codes for solutions can be found in include/impurity.h.
These have slightly changed over time.

Impurity Integer Code Comments

```
---------------------------
Vacancy 0
Interstitial 1
Arsenic 2
Phosphorus 3
Antimony 4
Boron 5
Electrons 6
Holes 7
X component of velocity 8 used for oxidation
Y component of velocity 9 used for oxidation
dry O2 10
wet O2 11
trap 12
Gold 13
Potential 14
x component of stress 15
y component of stress 16
mixed component of stress 17
Cesium 18
change in interface area 19
Electrically active Arsenic 20 version >= relA.9030
Electrically active Phosphorus 21 version >= relA.9030
Electrically active Antimony 22 version >= relA.9030
Electrically active Boron 23 version >= relA.9030
Polysilicon Grain Size  24 version >= rel.9130
Gallium 25 version >= rel.9130


n - node information
remember that in SUPREM-IV you can have multiple nodes
for each point in space, one for each material.
the format is as follows:

n point_number-1 material_code value_1 value_2 ... value_number_of_solutions

It is IMPORTANT that you remember the "off by 1" for the point
number.  This was just an efficiency hack since C arrays start
counting from 0 and by this time, the point arrays have been
created.  One might argue that the same thing should have been
done for the rest of the arrays.  But it's a historical (HYSTERICAL)
problem that keeps SUPREM-IV developers employed.
```

The material codes have been already shown under the r command.
Note that regions were not used here so that for the
case where you have two regions of the same material
intersections (e.g. the oxides in the TCAD Framework
demo), only one set of solutions values are stored
for the oxide, not 2.

M - model information
this line can be pretty much ignored but I will give the
syntax anyways

the syntax is:
M substrate_orientation last_temperature

codes for substrate_orientation are as follows (include/material.h)

Orientation Code

<100>0
<110>1
<111>2

last_temperature is in the units of Kelvin

Bibliography

[1] R.E. Bank et al. Transient simulation of silicon devices and circuits. *IEEE Transactions on Electron Devices*, 32(10):1992–2007, Oct. 1985.

[2] Thomas J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[3] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[4] H.R. Yeager and R.W Dutton. An approach to solving multiparticle diffusion exhibiting nonlinear stiff coupling. *IEEE Transactions on Electron Devices*, 32(10):1964–1976, Oct. 1985.