Time integration for coupled circuit-EM systems

Michael Matthes    Sascha Baumanns

Mathematical Institute
University of Cologne

ICESTARS meeting in Hagenberg, Oct 9th 2009,
FH Oberösterreich, University of Applied Sciences,
Auditorium HS2
1 Part I - Public part
   Coupled system
   Space discretized system
   Solving the DAE
   Numerical Experiments

2 Part II - Internal part
   MECS - Python Version
   Example: Using the DAE solver
PART I

Public part
Modified Nodal Analysis (MNA)

\[ A_C \frac{dq_C(A_C^T e, t)}{dt} + A_R g(A_R^T e, t) + A_L j_L + A_V j_V + A_I i_s(t) = 0, \]
\[ \frac{d\phi(j_L, t)}{dt} - A_L^T e = 0, \]
\[ A_V^T e - v_s(t) = 0. \]

\[ A_X \]
\[ i_s(t) \text{ and } v_s(t) \]
\[ q_C, g \text{ and } \phi \]
\[ e(t) \]
\[ j_L(t) \]
\[ j_V(t) \]

| element-related (reduced) incidence matrices |
| current and voltage sources |
| physical relations given for conductance, charge and flux |
| node potentials for \( t \in \mathcal{I} \) |
| current through inductivities for \( t \in \mathcal{I} \) |
| current through voltage sources for \( t \in \mathcal{I} \) |
EM-Element

![Diagram showing a simulation domain with a conductive rod and contacts.]
EM-Model Equations

\[ -\nabla \cdot \sigma (\nabla V + \Pi) - \nabla \cdot \varepsilon \left( \nabla \frac{\partial V}{\partial t} + \frac{\partial \Pi}{\partial t} \right) = 0, \]

\[ \nabla \times \left( \frac{1}{\mu} \nabla \times A \right) - \frac{1}{\mu} \nabla \nabla \cdot A + \sigma \left( \nabla V + \frac{\partial A}{\partial t} \right) + \varepsilon \left( \nabla \frac{\partial V}{\partial t} + \frac{\partial \Pi}{\partial t} \right) = 0, \]

\[ \Pi - \frac{\partial A}{\partial t} = 0. \]

\( V \) | scalar potential
\( A \) | vector potential
\( \Pi \) | pseudo-canonical momentum = \( \frac{\partial A}{\partial t} \)

Remark: There is some kind of freedom where to plug in \( \Pi \).
EM-Model Boundary Conditions - I
EM-Model Boundary Conditions - II

contact boundary:

\[ \nu = f(x, t), \quad A \cdot \tau = 0, \quad \frac{\partial (A \cdot \nu)}{\partial \nu} = 0. \]

non-contact boundary:

\[ \left( \nabla \nu + \frac{\partial A}{\partial t} \right) \cdot \nu = 0, \quad A \cdot \tau = 0, \quad \frac{\partial (A \cdot \nu)}{\partial \nu} = 0. \]
Coupling Conditions

Coupling via boundary conditions and current equation.

Remark: EM-model is charge conserving.
Coupling Conditions: MNA to EM-Model

- $A_M$ describes the circuit topology of the device.
- $A_M^T e$ denotes the applied node potential.

Dirichlet boundary condition for $\mathcal{V}$:

$$\mathcal{V} = \mathcal{V}_{bi}(x) + \mathcal{V}_{ext}(x, A_M^T e)$$
Coupling Conditions: EM-Model to MNA

- $\Gamma_k$ $k$-th contact.

Current through the $k$-th contact of the device:

$$j_{s_k} = \int_{\Gamma_k} -\sigma (\nabla V + \Pi) - \varepsilon \frac{\partial}{\partial t} (\nabla V + \Pi) \cdot \nu ds$$
General Coupling

\( A_C \frac{dq_C(A_{Te}^T e, t)}{dt} + A_R g(A_{Re}^T e, t) + A_L j_L + A_N j_N + A_I i_s(t) + A_M j_M = 0 , \)

\( \frac{d\phi(j_L, t)}{dt} - A_L^T e = 0 , \)

\( A_V^T e - v_s(t) = 0 , \)

\( j_M - g(V, II, A_{Te}^T e) = 0 , \)

\( h(V, A, II, A_{Te}^T e) = 0 \)

with \( g \) being the function representing the current through the EM device and \( h \) being the EM-operator.
Space Discretized System

Linear circuit with one EM-element (non-semiconductor material):

\[
\begin{bmatrix}
A_C C A_C^T & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & L & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_{SE} H A_{M}^T & 0 & 0 & 0 & A_{SV} & 0 & A_{SP} & (e)'
\end{bmatrix}
\begin{bmatrix}
jL \\
jV \\
0 \\
(jM) \\
V
\end{bmatrix}
\begin{bmatrix}
A_{GE} H A_{M}^T & 0 & 0 & 0 & A_{GV} & 0 & A_{GP} \\
A_{ME} H A_{M}^T & 0 & 0 & 0 & A_{MV} & A_{MA} & A_{MP} \\
0 & 0 & 0 & 0 & -I & 0
\end{bmatrix}
\]

Remark: \( A_{MP} \), \( B_{GV} \) and \( B_{MA} \) non-singular.
Remarks

- Linear DAE of the form $Ay' + By + q = 0$.
- Index: No result yet but we think that it depends on the underlying device geometry and circuit topology. This would be different to the results obtained from the coupling to the DD-model.
- For the index investigation a deep understanding of the underlying matrices $A_{XY}$ and $B_{XY}$ is needed.
Solving the DAE

The time discretization is done with one of the following DAE solvers:

- **DAEn - BDF solver**, can handle DAEs of the form
  \[ M(t)y' + f(y, t) = 0, \]
- **GLIMDA - GLM solver**, can handle DAEs of the form
  \[ Md(y, t)' + b(y, t) = 0, \]
- **own BDF solver**, can handle DAEs of the form
  \[ f(d(y, t)', y, t) = 0. \]

We implemented them all in Python.

Remark: The index of a DAE is very important for the efficiency and feasibility for the solver.
Numerical experiments - Oscillator

Oscillator vcoBi from D5.2:

**Goal:** Simulate the coupled inductors by the EM.

**MATLAB®:**
- Simulation done with equivalent circuits for MOSFETs
- Simulation done with drift-diffusion model for MOSFETs

**Python:**
- Equivalent circuit in progress.
- Drift-diffusion model not implemented so far.
- BSIM not coupled to the network equations so far (for reference purposes).
Numerical experiments - Simple circuits with EM - I

We tested different EM-elements in different circuit configurations. Index does not exceed 2 for simple grids (given by MAGWEL).
Numerical experiments - Simple circuits with EM - II

Simple circuit with sinusoidal voltage source and simple EM-element.

EM-element behaves like a resistor with a resistance of \(8.74 \times 10^{-4}\) Ohm.
PART II

Internal part

Background and Usage Cologne software
Part I - Public part

Coupled system
Space discretized system
Solving the DAE
Numerical Experiments

Part II - Internal part

MECS - Python Version
Example: Using the DAE solver
Why Python?

- easy to use, easy to program
- advanced numerical packages numpy and scipy
- easy to include modules written in other programming languages
Overview of MECS - I

MECS (Multiphysical Electric Circuit Simulator):

- developed and maintained by our research group.
- originally implemented in MATLAB®.
- using the drift-diffusion model for semiconductors.
- drift-diffusion model not available in the Python version so far.
Overview of MECS - II

MECS can generate DAEs of the type

- \( My' + b(y, t) = 0 \) (standard) or
- \( Md(y, t)' + b(y, t) = 0 \) (proper).

In the following we concentrate on the standard case. We need evaluations of

- \( M, b(y, t) \) and \( \frac{\partial b(y, t)}{\partial y} \) (standard case).
Overview of MECS - III

- **Input**: netlist and options.
- **MECS**:
  1. Initialization of the circuit elements and setup of the DAE.
  2. Simulation of the circuit by evaluation of the elements element-wise ($R, C, \ldots$). (Control goes to a DAE solver)

- **netlist**: Mainly spice3 with little extensions. So far not all spice3 elements available.

```plaintext
* Simple Circuit with EM-Element
V1 0 1 sin(0 1 1 0 0)
$EM1 1 0 prototype_inductor 0
```
MECS Elements - I

MECS Elements:

- resistors, capacitors, inductors, voltage and current sources.
- external elements (include e.g. MAGWEL EM-element).

Recall: DAE has the form $My' + b(y, t) = 0$ with $y = (e, j, x)$.

- $e$: node potentials.
- $j$: internal variables (currents through inductors and capacitors) + current through external elements.
- $x$: external variables (e.g. pseudo-canonical momentum, scalar and vector potential).
- $t$: time.
MECS Elements - II

Elements:

- Initialization:
  - Input: type, parameter and names of all elements of the type, massnode, DAE formulation.

- Evaluation:
  - Input: flag, y, timestep.
  - Output: Corresponding part of $M$, $b(y, t)$ or $\frac{\partial b(y, t)}{\partial y}$. 
MAGWEL EM-element in MECS

We can handle:

- insulating and conducting materials.
- mixture of both.
- two port elements.

Recall the EM equations

\[
\begin{align*}
\mathbf{j}_M - g(\mathbf{V}, \mathbf{\Pi}, A^T_M e) &= 0 \\
\mathbf{h}(\mathbf{V}, \mathbf{A}, \mathbf{\Pi}, A^T_M e) &= 0
\end{align*}
\]

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
x \\
A^T_M e
\end{bmatrix} = \begin{bmatrix}
0 \\
\mathbf{j}_M
\end{bmatrix}
\]

with \( x = (\mathbf{V}, \mathbf{A}, \mathbf{\Pi}) \).
EM-element in MECS: Communication MAGWEL/MECS

MAGWEL EM-element:

- **Initialization:**
  - Input: devicename, contact nodes, reference node
  - setup the coupling matrix $A_M$
  - call MAGWEL (once):
    - number of nodes and links
    - matrices $A, B, C, D$.
    - evaluate EM part of $M$ and $\frac{\partial b(y, t)}{\partial y}$ (once).

- **Evaluation:**
  - Input: flag, y.
  - Output: Corresponding part of $M$, $b(y, t)$ or $\frac{\partial b(y, t)}{\partial y}$. 
Communication MAGWEL/MECS

up to now:

- Communication via files. We need just the matrix dump. Possible because we have linear systems.

future:

- Direct communication via pointers. (MECS part easy)
Example: Using the DAEn solver

- Originally implemented in MATLAB® by Michael Hanke.
- BDF solver using a combination of error estimation and order selection strategies.
- handles DAEs of the form $M(t)y' + f(y, t) = 0$ with $M(t)$ possibly singular.
Using DAEn - Example: Ringmodulator

- The Ringmodulator is represented by a stiff system of 15 nonlinear ordinary differential equations.
- It depends on a factor $C_s$.
- For $C_s = 0$ we have a DAE of index 2 with 11 differential and 4 algebraic equations.
- From the testset on http://pitagora.dm.uniba.it/~testset/problems/ringmod.php
Using DAEn - Example: Ringmodulator

We have five relevant Python files:

- **csparse.py** - (main sparse matrix format)
- **ringmodulator.py** - (equations for the ringmodulator)
- **funcDAE_standard.py** - (interface for DAEs of the form $My' + f = 0$, provides numerical jacobians)
- **daen.py** - (BDF-solver DAEn)
- **test_daen_ringmodulator.py** - (script for running the solver)
Using DAEn - Example: Ringmodulator

ringmodulator.py:

class Ringmodulator(object):
    def __init__(self, y0=zeros(15), time_interval=(0, 1e-3), Cs=2e-12):
        self.C = 1.6*1e-8

    def mass(self, y, t):
        return Csparse(diag([self.C, self.C, self.Cs, self.Cs, self.Cs,
                              self.Cs, self.Cs, self.Cp, self.Lh, self.Lh, self.Ls2,
                              self.Ls3, self.Ls2, self.Ls3, self.Ls1, self.Ls1]))

    def f(self, y, t):
        return array([-(y[7] - 0.5*y[9] + 0.5*y[10] + y[13] - 1./self.R*y[0]) ,
                      ...])
Using DAEn - Example: Ringmodulator

test_daen_ringmodulator.py:

```python
ringmod = Ringmodulator(y0=zeros(15), timeinterval=(0, 1e-3), Cs=2e-12)
daes=FuncDAE_standard(ringmod.mass, ringmod.f, y0, timeinterval)
options={'abstol': 1e-4,
         'reltol': 1e-4,
         'estrat': 2,
         'ostrat': 'Std',
         'scaling': 'on',
         'report': 'none'}
daesolver=DAEn(dae, options)
daesolver.run()
tout, yout = daesolver.get_solution()
```
Using DAEn - Example: Ringmodulator

Comparison: low frequency input signal (component 1)
Using DAEn - Example: Ringmodulator

Comparison: resulting current (component 15)
Using the other solvers is done in a similar way. They just accept a different interface:

- GLIMDA - use FuncDAE_proper.py
- own BDF solver - use FuncDAE_fproper.py
The End

Thanks for your attention!