Partial Element Equivalent Circuit (PEEC) Toolbox for MATLAB, integrating FastCap2 and FastHenry2 to Calculate Partial Elements, and Multisim or LTspice for circuit simulation

VJOSA SHATRI¹, RUZHDi SEFA¹, LAVDIM KURTAJ²
¹Department of Fundamental Engineering Subjects
²Department of Automation
Faculty of Electrical and Computer Engineering
University of Prishtina
Bregu i Diellit p.n., Prishtina
KOSOVA
vjosa.shatri@uni-pr.edu, ruzhdi.sefa@uni-pr.edu, lavdim.kurtaj@uni-pr.edu

Abstract: - Partial Element Equivalent Circuit (PEEC) method is one of numerical modeling methods for solving electromagnetic problems. Electromagnetic problem to be solved is expressed with electric circuit, and then solved by some standard circuit solver software. To make working on PEEC method more accessible, a PEEC toolbox for MATLAB is created. Toolbox contains functions for describing and composing geometry of problem with wire-based language, functions for meshing according to PEEC requirements for surface and volume cells, functions for plotting geometry and mesh data. Specific functions enable using FastCap2 and FastHenry2 to calculate partial elements and getting data back. With toolbox functions one can create Netlist for interfacing with circuit simulation software, Multisim from National Instruments or LTspice from Linear Technology. Toolbox may serve as powerful Front-End for FastCap2 and FastHenry2.

Key-Words: - PEEC Method, PEEC Toolbox, MATLAB, FastCap2, FastHenry2, Geometry Description.

1. Introduction
Numerical modeling is widely used for predicting performance of products before they are produced, in order to avoid prototyping-testing cycle, that can be time consuming and costly. Interest for Partial Element Equivalent Circuit (PEEC), as one of methods for solving electromagnetic problems, is ever increasing, since its development by Dr. Ruehli [1], [2]. Solving electromagnetic problems with electric circuits makes it attractive for wide range of electrical engineering community, making it possible to solve combined electromagnetic-circuit problems, and many other fields that used electric circuit method for solving problems.

When the problem is expressed with electric circuit, it may be analysed by standard simulation software of SPICE family that are widely available.

Road from problem to be solved, usually expressed in form of geometric drawing, up to its electric circuit model requires some steps [3], [4], and there is no freely available software to handle them. Steps to be taken are geometry description, meshing by specific requirements of PEEC method, calculation of circuit element parameters, and creation of circuit in form that is possible to handle by selected circuit solver. Number of circuit elements grows rapidly, even for simple problems, and some automation software is necessary for handling these steps.

MATLAB, as a powerful environment for solving problems and for graphical presentation [5], can extend its functionality by group of functions for specific problem, the toolbox.

Extending MATLAB toward PEEC method is started by using its programming and graphical capabilities for handling some parts of problem, and integrating other applications that proved successful in their filed. Geometry description and meshing, according to PEEC requirements, is done by MATLAB. For calculation of circuit parameters FastCap2 [6] and FastHenry2 [7] developed at MIT are used. Circuit is simulated by National Instruments Multisim [8] or Linear Technologies LTspice [9]. MATLAB will handle all data conversion for integrating these applications in a common environment. Same platform then can be used to extend its functionality and test new approaches for specific point of interest related to PEEC method.

Next section will provide insight to our approach for geometry description and language for wire-based geometry composition. Then PEEC compatible meshing for different modes will follow.
Integration of FastCap2 and FastHenry2 to calculate partial elements, and Multisim or LTspice for simulation are covered in Section 4 and Section 5.

2. Geometry Description

Current version of Toolbox is using text-based source files of defined syntax for geometry description. They can be created by any general purpose text editor, and specifically standard MATLAB script editor may be used.

System to be analyzed is supposed to be composed of a number of bodies in space. One Cartesian coordinate system is adopted as global, OXg Yg Zg.

Each body has its own local coordinate system, OXb Yb Zb. Position and orientation of body in system is defined by relative position of its local coordinate system with respect to global one, Tbi. Fig.1 shows system with two bodies with corresponding coordinate systems.

Positions are expressed in homogeneous coordinates as

\[
p = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}
\]  

while mapping from one coordinate system to the other is done by homogeneous transformation matrixes. Standard transformation matrixes used for rotation, translation, local scaling, and global scaling are given by [10]

\[
T_{x,y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]  

\[
T_{y,x} = \begin{bmatrix} \cos \phi & 0 & \sin \phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \phi & 0 & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]  

\[
T_{z,\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]  

\[
T_{(x,y,z)} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]  

\[
T_{scale_{,xy}} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]  

\[
T_{scale_{,global}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}
\]  

Bodies are constructed from pieces of wires (elements) with selected cross-section width and height, and of composable form. Width of wire is spread along x-axis, height along y-axis, and length goes along z-axis, Fig.2. While composing bodies, a number of predefined elements can be used. Beginning of first element of body may be redefined with respect to body coordinate system by ORigin instruction. Next elements will be sequentially chained by automatically assigning element coordinate system from ending of previous element. Three elements in chain to construct body2 are shown in Fig.1.

By default, origin of element coordinate system is at center of beginning cross-section face.
Figure 2. Element position, orientation and its parameters (width, height, and element homogeneous transform matrix).

Change of position/orientation caused by element is defined by element homogeneous transform matrix \( T_{biej} \). By ORigin instruction it is possible to change point-of-view for next element at any corner, center of edges, or XYposition of cross-section with respect to its default position.

Orientation of initial element progress can be set by any standard angle representation: Euler I, Euler II, Roll-Pitch-Yaw, Pan-Tilt, or axis-angle. Start of element in Fig.2 is translated by \((1,0,1)\) from body coordinate system and rotated by \(30^{\circ}\) around y-axis.

Element is defined in unit dimensions, and it is scaled to real dimensions by local scaling transform matrix \( T_{\text{scale}_{xyz}} \).

### 2.1 Geometry Description Instructions

Source file for geometry description is simple text file. Comments lines can be added and they begin with '% ' symbol. Empty lines and leading spaces or tabs are skipped. They can be used to make structure of file more readable. Space is used as separator between parameters of same line.

To make geometry description easier, variables with predefined values may be defined. Value may be changed by reassigning value to same variable, and it can be done at any line in input file. Standard MATLAB functions can be part of assignment expression. Syntax for \( \text{VL} \) instruction is (VariableLocal):

\[
\text{VL} \text{ Var1Name}=\text{Expression Var2Name}=\text{Expression}
\]

Defined variables can be used as parameters for other instructions.

Wire is basic body construction element, and defining wire dimensions with \( \text{WD} \) (Wire Dimensions) is part of standard initial declarations. \( \text{WD} \) syntax is:

\[
\text{WD} \text{ number } S \text{ width height } R \text{ diameter }
\]

number can be form 1 to 20, \( S \) is for square cross-section. \( R \) is for circular cross-section, but currently is not supported by other parts of software.

Standard structure for body definition starts with \( \text{BS} \) (BodyStart) instruction and ends with \( \text{BE} \) (BodyEnd) instruction, and optionally may have \( \text{OR} \) (ORigin) instruction:

\[
\text{BS Name ...} \\
\text{ OR ...} \\
\text{ SW ...} \\
* \\
* \\
* \\
\text{BE}
\]

Syntax for \( \text{BS} \) instruction is:

\[
\text{BS} \text{ BodyName Xb Yb Zb DX } [0] \\
\text{DY } [0] \\
\text{DZ } [0] \\
\text{PT } \text{pan [tilt]} \\
\text{RPY } \text{roll pitch yaw} \\
\text{E1 } \phi \theta \psi \\
\text{E2 } \phi \theta \psi
\]

where Xb, Yb and Zb define origin position for body coordinate system, with respect to global coordinate system. For orientation can be selected one of possible options, with \( \text{DZ 0} \) as default.

Instruction \( \text{BE} \) has no parameters.

\( \text{OR} \) instruction syntax is same as one for \( \text{BS} \) instruction with BodyName parameter missed.

To start composing body a specific wire, from predefined wires with \( \text{WD} \), may be selected by instruction \( \text{SW} \) (SelectWire) or the new one can be defined as part of same instruction according to following syntax:

\[
\text{SW} \text{ number } [\text{Referent node}] \\
\text{S } \text{width height } [\text{Referent node}] \\
\text{R } \text{diameter } [\text{Referent node}]
\]
where number is one of predefined wires, and two other forms for instruction serve for direct wire dimension definition. Each form may use optional parameter for Referent node, for selecting point-of-view in wire cross-section. By default it is C (Center) and next wire element will be positioned with its center to origin of current coordinate system. Other options are:

<table>
<thead>
<tr>
<th>Form</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU</td>
<td>Left</td>
<td>Up</td>
</tr>
<tr>
<td>RU</td>
<td>Right</td>
<td>Up</td>
</tr>
<tr>
<td>LD</td>
<td>Left</td>
<td>Down</td>
</tr>
<tr>
<td>RD</td>
<td>Right</td>
<td>Down</td>
</tr>
<tr>
<td>CU</td>
<td>Center</td>
<td>Up</td>
</tr>
<tr>
<td>CD</td>
<td>Center</td>
<td>Down</td>
</tr>
<tr>
<td>CL</td>
<td>Center</td>
<td>Left</td>
</tr>
<tr>
<td>CR</td>
<td>Center</td>
<td>Right</td>
</tr>
<tr>
<td>C</td>
<td>Center</td>
<td>(default value)</td>
</tr>
<tr>
<td>XY Xpos Ypos</td>
<td>- relative to center of wire</td>
<td></td>
</tr>
</tbody>
</table>

Left, Right, Up and Down are as in Fig.3.

When wire has been selected, bodies can be constructed with lines, corners and arcs. Planes are created with lines with appropriate width and length dimensions, and of desired height.

For creating lines, three instructions are available **LB** (LineBegin), **LE** (LineEnd), and **LR** (Line Relative). They behave the same and have the same syntax, but behave differently when meshing. LB will create capacitive cells at beginning cross-section face, while LE at ending face. All of them will create capacitive cells around wire in length direction. This approach has been selected to avoid geometry parsing for this version. Syntax for instruction **Lx** (where x is B, E, or R) is:

**Lx length**

\[ X_{rel} Y_{rel} Z_{rel} \]

parameter **length** specifies length of segment in direction of its own z-axis. Second form moves referent node to selected relative position with beginning and ending faces remaining parallel, and possibly not orthogonal to length. Also, it effectively decreases wire cross-section.

To change direction **TC** (TurnerCorner) instruction can be used. It produces sharp corners, 90° by default if second parameter is not used. Instruction may have one or two parameters:

**TC** dir1 [theta1] dir2 [theta2] ...

[TE]

dir can be any of directions **L**, **R**, **U**, **D**, **F** for Left, Right, Up, Down, and Forward. Optional parameter **theta** specifies angle between two faces, if not specified default value is 90°. If more than one set of parameters is used, branching will be created. Elements for branches follow same order as set on parameters of **TC**, and each branch must end with **TE** instruction. Ending coordinate system, at same time beginning of next segment, is ending of last branch. If only one parameter is used there is no need for using **TE** instruction.

Circular arcs are defined by **AR** (ArcRelative) instruction:

**AR dir radius theta**

Center of arc will be in **dir** side and orthogonal to ending vector of last element. If **dir** is L or R (for Left of Right) center will be on x-axis, else if **dir** is U or D (for Up and Down) center will be on y-axis. Two other parameters set **radius** and sector angle **theta**. Radius is measured from center to external surface of circular wire.

When radius is mode equal to wire width, corner with angle theta will be created. This corner will differ from corner created with **TC** in manner it connects external edges of wires. Whereas **TC** corner is four-face prism, **AR** corner will be three-face prism (if number of discrete segments per element is set to one) or multi-face prism (if number of discrete segments per element is set to more than one).

Input file in specified format is analyzed and structure with data is created with MATLAB function **File2GD** (File to Geometric Data)
function [GD, WD, CW] = File2GD(FileName)

All parameters for other steps, defined by VL instruction, like material data, are stored in GD structure. Other functions will use this structure, produce results, and store additional data to this structure. By function we may get two more structures: Wire Dimensions data created by WD instructions and Current Wire data.

2.2 Geometry Viewing

Important part of geometry description is to visualize system of bodies to make sure that description is proper. MATLAB function File2GD creates vertices and faces for each element of each body. Data is in format as requested by patch MATLAB function. Vertices are for given real dimensions but referenced to element coordinate system. Other MATLAB function plotGD will transform element data to its real position, by taking into account body and first element origin coordinate systems.

```
mv = (GD.B(i).Tb * ... GD.E(j).T * ... GD.E(j).Tcw * ... (GD.E(j).ed.v0(:,1:4))' )';
```

and plot them in the MATLAB figure window, from where viewing angle and size can be adjusted. mv line of code transforms vertices to its real position. First part, GD.B(i).Tb, is homogeneous transform matrix that relates i-th body coordinate system to global one. Second part, GD.E(j).T, gives position and orientation of j-th element referenced to body coordinate system. Third part, GD.E(j).Tcw, describes relation between point-of-view of cross-section of wire and its center, were the element prototype data are related to. Finally, the last part, (GD.E(j).ed.v0(:,1:4))', just takes element vertices in row array form and transforms them to column array form for compatibility with transform matrices. Product is transposed back for compatibility with drawing function. Face data has no need to be transformed and is only copied, because it is invariant to transformations. Plotted bodies are hollow; see body 1 in Fig.1, also surface is made partially transparent, in order to gain better view of geometry.

plotGD function enables us to select which coordinate functions to plot. We may select individually from following possibilities

```
1 - base KO
2 - body KO
4 - element KO
8 - first element KO of any body
16 - end element KO of any body
```

where KO stands for coordinate system. As prarameter (varargin(3)) to function we pass sum of preceding numbers we select.

Format for calling plotGD function is

```
function [ ] = plotGD(GD,varargin)
```

First and second parameter, varargin(1) and varargin(2), set length of coordinate axis and their thickness, as seen from Fig.1 to Fig.3.

Function plotGD uses functions drawMesh from [11] and modified version of drawAxis3d to draw axis to position and orientation given by transform matrix T, named drawAxis3dT, from [12].

3. Meshing

From its initial definition [1], the PEEC method defines two types of cells, surface cells for capacitive partitions and volume cells for inductive-resistive partitions. From geometry definition data, software must generate mesh data for both types of partitions.

With variable GD.meshGL (Global Local) we can choose between global mode, where elements of all bodies are meshed according to defined global parameters, and local mode, where for every element we can choose meshType and corresponding parameters, like number of segments. meshType (Global or Local) can be:

- 1D - for capacitive cells elements are partitioned in length direction only. Depending on four additional parameters (nw, nh, nhw, nhh) and frequency range of interest, volume meshing can be in one filament (nw=0, nh=0), row of filaments in width direction (nw>0, nh=0), column of filaments in height direction (nw=0, nh>0), and rows and columns of filaments (nw>0, nh>0). When one or both of nw and nh is zero, nhw and nhh can be sent transparently to part of software that calculates inductive-resistive parameters to account for skin-effect. In this case parameters are calculated for
specific frequency and are supposed to be used in narrow frequency range.

- **2D** - partitioning is done in length and width direction. Height direction partitioning for volume cells may still be controlled with \( nh \) and \( nhh \).

- **2DS** - this mode is practical for high frequencies and narrow band. In this case surface is partitioned at all directions for both cell types, but there is only one cell in depth.

- **3D** - partitioning is done at all directions for both cell types.

- **3Dxyz** - partitioning is done at all directions for both cell types and cell faces are parallel to global coordinate system axis \( x, y, \) and \( z \).

MATLAB function for meshing is \( \text{GD2GDmesh} \)

```matlab
function [ GDmesh ] = GD2GDmesh( GD )
GDmesh = GD;
end
```

**Figure 4.** Serpentine line meshed with 1D cells. Cells are scaled to highlight cell borders and to see current cells (green).

**Figure 5.** Zoomed part of serpentine line meshed with 3D cells and additional meshing for FastCap2. Only surface cells are plotted.

Parameters are same as for function \( \text{plotGD} \). Fig.4 shows serpentine line meshed with 1D cells and scaled to emphasize cell boundaries and volume cells.

One more additional meshing of surface cells may be necessary when partial capacitance calculation is done with FastCap2. This is controlled by three parameters: \( ncw, nch, \) and \( ncl \). They define number of divisions in corresponding direction.

For **3D** case plotting is done by calling MATLAB function \( \text{plotGDmesh3D} \) with same parameter meaning

```matlab
function [ ] = plotGDmesh3D(GDmesh,varargin)
end
```

part of zoomed serpentine line meshed with 3D cells with additional meshing for FastCap2 can be seen in Fig.5. For volume cells no faces need to be generated and they are not included in 3D mesh plot.

### 4. Partial Element Calculation

By calling function \( \text{GDmesh2FastCH} \)

```matlab
function [C_FC, L_FH, R_FH ] = GDmesh2FastCH( GDmesh )
end
```

with \( \text{GDmesh} \) structure as parameter that contains mesh data, files compatible with FastCap2 (generic format file for geometry of each body, *.qui, and list...
file for relating geometry files, *.lst) and FastHenry2 (*.inp file) will be created.

FastCap2 is called first with its window is started by MATLAB and progress of calculation can be observed, including errors, if any. After completing calculations Maxwell capacitance matrix is read to MATLAB. FastCap2 window will stay open and may be closed manually.

FastHenry2 is called next, again with its window open. When calculations are done, Inductance and Resistance matrices are read back to MATLAB. FastHenry2 window will also remain open.

Package for FastCap2 and FastHenry2 includes FastModel which can plot mesh data loaded from files, and serves as another check of data generation and conversion.

All mesh data and handles are available for implementing different partial element calculation methods.

5. SPICE Netlist Generation and Simulation
Three matrices (C_FC, L_FH, R_FH) resulted from FastCap2 and FastHenry2, together with GDmesh structure are called by function FCH2SPICE

```matlab
function [ ] = FCH2SPICE( td, C_FC, L_FH, R_FH, GDmesh, MS_LT )
```

to generate Netlist file (*.cir) that can be included in Multisim or LTspice. If parameter MS_LT='MS' then Netlist file is Multisim compatible. For 'LT' value it will make it LTspice compatible.

Parameter td will select between quasistatic (\(td=R\)) or retarded model (\(td=R'\)). When retarded model is selected, GDmesh data is used to calculate center-to-center distances and time-delays. Laplace controlled source is used for delay implementation.

6. Conclusion
PEEC toolbox for MATLAB with defined wire-based language enables easy construction of geometry for wide range of problems. Same Front-End may serve as powerful, higher level, interface for FastCap2 and FastHenry2, even for applications not related to PEEC. Different meshing modes allow flexibility between accuracy and size of problem. Visualisation functions will provide insight on defined geometry and mesh data. Partial elements calculated by FastCap2 and FastHenry2 are handled by toolbox functions and Netlist for interfacing to circuit simulators is created. Netlist may be selected to be compatible with Multisim from National Instruments, or LTspice from Linear Technology.

References: