ECTSim MATLAB Toolbox is designed to model electrical tomography for both two-dimensional and three-dimensional applications. The example files, `example_2D.m` and `example_3D.m`, illustrate how to prepare a model within our software. They demonstrate simulating the distribution of electromagnetic fields, conducting complex impedance measurements, and performing image reconstructions using sensitivity matrices.

Example 2D

This example shows how to solve the forward and inverse problems for a 16-electrode sensor filled with materials of varying permittivity and conductivity.

Example 3D

This example demonstrates how to solve the forward and inverse problems in a three-dimensional numerical model. A 3D sensor with 32 electrodes, arranged in two rings of 16 electrodes each, is used.

Numerical model preparation

---

addElement

Adds an element with a given name to a numerical model ('Sensor' cell array). The element must exist in the 'Elements' cell array to be included in the final model and simulation. This function ensures that no element sharing grid points with any other element is added.

Usage: `[model] = addElement(model, 'el_name', permittivity, varargin);`

- *model* - Structure with a numerical model description.
- *el_name* - Name of the element.
- *permittivity* - Relative permittivity value (>=1).
- *varargin* - Optional parameters including conductivity and potential values applied if boundary conditions are set on this element; excitation potential if the potential is switched on the electrode.

combineElements

Performs an operation (addition, subtraction, or multiplication) on two regions.

Usage: `[Index] = combineElements(Index, Index2, operations);`

- *Index* - First region indices.
- *Index2* - Second region indices.
- *operations* - Defined as sum, difference, or product of the two regions.
- *returns* - Vector of indices resulting from the operation on two regions.

defineWorkspace

Initiates work on a model. Input arguments describe the size of the area in which the model will be constructed. Values for width and height should be provided in mm. This function creates global variables:

- *Elements* - A cell array that stores all created simple and complex elements.
- *Sensor* - A cell array that stores elements making up the final sensor model.
- *nargin* - 2 for 2D, 3 for 3D.

Usage: `[model] = defineWorkspace(width, height, depth);`

- *model* - Structure with a numerical model description.
- *width* - Workspace width.
- *height* - Workspace height.
- *depth* - Workspace depth (optional for 3D).

makeTomograph

Optionally prepares the entire body of a cylindrical sensor quickly.

Usage: `[model] = makeTomograph(model, radius, height, wallWidth, insulationWidth, numElectrodes, ringPositions, ringRadius, electrodeHeight, electrodeWidthDeg, electrodeThickness, const);`

- *model* - Structure with numerical model definition (post-*defineWorkspace* and *defineMesh*).
- *radius* - Inner radius of the cylindrical sensor.
- *height* - Height of the cylindrical sensor.
- *wallWidth* - Thickness of the cylinder wall.
- *insulationWidth* - Thickness of inner insulation.
- *numElectrodes* - Number of electrodes.
- *ringPositions* - Z-position of the center of electrode rings.
- *ringRadius* - Radius of electrode rings.
- *electrodeHeight* - Height of the electrodes.
- *electrodeWidthDeg* - Angular width of a single electrode.
- *electrodeThickness* - Thickness of electrodes.
- *const* - A structure containing lists of electrical permittivities and conductivities of the materials used in the sensor construction.

newComplexElement

Creates a new complex element by combining existing elements. The new element is added to the 'Elements' cell array in the numerical model.

Usage: `[model] = newComplexElement(model, name, formula);`

- *model* - Structure with model description.
- *name* - Name of the element to be created.
- *formula* - Operation on elements represented as 'el1+el2', 'el1-el2', or 'el1&el2'.

newSimpleElement

Creates a new solid geometry element and adds it to the 'Elements' cell array in the numerical model. This function checks whether the name has already been used and selects the appropriate function for calculating the geometry.

Usage: `[model] = newSimpleElement(model, shape, name, coord, angle, dimensions, bounding_angle1, bounding_angle2, ring_width);`

Example: `[model] = newSimpleElement(model, 'cylinder', 'test_element1', [0, 0, 0], [0, 45, 15], [50, 50, 50], 0, 270, 25);`

- *model* - Structure with a numerical model description.
- *shape* - Shape of the element, options include: 'ellipse', 'rectangle', 'ellipsoid', 'cuboid', and 'cylinder'.
- *name* - Name of the new element.
- *coord* - Coordinates of the center of the shape, matching the dimensionality of the model.
- *angle* - Rotation angle of the element.
- *dimensions* - Dimensions of the object along the x, y, and z axes (for 3D objects).
- *bounding_angle1* - Start angle for an ellipse segment; default is 0.
- *bounding_angle2* - End angle for an ellipse segment; default is 360.
- *ring_width* - Width of a created ring; default creates a solid interior.

prepareMaps

Prepares three essential maps for discretizing the mesh.

Usage: `[model] = prepareMaps(model);`

- *model* - Structure with a numerical model description.
- List of prepared maps:
  - *model.eps_map* - Map of relative permittivity (epsilon).
  - *model.sigma_map* - Map of conductivity (sigma).
  - *model.patternImage* - Map of elements in the model.

readFormula

Extracts names of elements and symbols of operations from an algebraic expression. This function is primarily used by `newComplexElement`.

Usage: `[components, operations] = readFormula(formula);`

- *formula* - Algebraic expression, e.g., from `newComplexElement`.
- *components* - Names of the elements in the formula.
- *operations* - Operational symbols used in the formula.

specifyFOV

Determines the indices of the area of interest inside the sensor. These indices are subsequently used by functions to present data and in `upscaleModel` during nonlinear reconstruction.

Usage: `[model] = specifyFOV(model, name);`

- *model* - Structure with a numerical model description.
- *name* - Name of the element to be saved as the Field of View (FOV).

Discretization mesh

---

defineMesh

Distributes points (pixels) across the workspace. It creates two variables—matrices X and Y—that store x and y coordinates, representing the grid. The inputs are numbers of points spread along the width and height of the workspace.

- `nargin = 3` -> 2D
- `nargin = 4` -> 3D

Usage: `[model] = defineMesh(model, varargin);`

`varargin` -> widthPoints, heightPoints, depthPoints

- *model* - structure with a numerical model description
- *widthPoints* - mesh width
- *heightPoints* - mesh height
- *depthPoints* - mesh depth

findIndexFwdp

Finds indices in a quadtree mesh.

Usage: `[index] = findIndexFwdp(model, element_name);`

- *model* - structure with a numerical model description
- *element_name* - the name of the element

findNeighbors

Finds pixels (four neighbors in 2D and six in 3D) near a pixel with a given index.

Usage: `[QT] = findNeighbors(QT, index);`

- *QT* - quadtree structure with a numerical model description
- *index* - the index of the pixel for which neighbors are searched

fineMesh

Modifies a pattern image used by the meshing algorithm. Generating a pattern in a specific geometrical element creates a finer mesh in that area. A chessboard pattern is generated in the selected element of the space.

Usage: `model = fineMesh(model, element, elementSize);`

- *model* - structure with a numerical model description
- *element* - the name of the element
- *elementSize* - determines the mesh size

meshing

The meshing function creates and manages a quadtree of an image.

Usage: `[model] = meshing(model, pixMin, pixMax);`

- *model* - structure with a numerical model description
- *pixMin* - minimum pixel size in a mesh
- *pixMax* - maximum pixel size in a mesh

qtComp

The qtComp function recreates a full matrix from a quadtree sparse with new chosen values.

Usage: `[fullMat] = qtComp(model, varargin);`

- *fullMat* - full matrix (uniform) with values
- *QT* - structure with a numerical model description necessary to create a quadtree
- *varargin* - nonuniform mesh to be represented in a uniform mesh

qtCut

Function for a single cut in a quadtree structure. Used as a recursive function by *qtDecom*.

Usage: `[qt] = qtCut(QT, A, S, param, pixMin, pixMax);`

- *qt* - quadtree mesh structure
- *A* - full matrix to cut
- *S* - temporary matrix that is part of A
- *param* - measure of non-uniformity
- *pixMin* - minimum pixel size in a mesh
- *pixMax* - maximum pixel size in a mesh

qtDecom

Function to create a quadtree structure. Used in *meshing*.

Usage: `qt = qtDecom(QT, A, param, pixMin, pixMax);`

- *QT* - structure with a numerical model description necessary to create a quadtree
- *A* - a map used for forming a uniform mesh
- *param* - measure of non-uniformity
- *pixMin* - minimum pixel size in a mesh
- *pixMax* - maximum pixel size in a mesh
- *qt* - quadtree structure with a numerical model description

recNeighSearch

Recurrent neighbor searching. Used by *findNeighbors*.

Usage: `[nList] = recNeighSearch(QT, index, pixSize, nList, plane);`

- *QT* - quadtree structure with a numerical model description
- *index* - the index of the pixel for which the neighbors are searched
- *pixSize* - size of an element
- *nList* - structure with the latest list of neighbors in plane
- *plane* - name of the plane to search: 1 - k, 2 - j, 3 - i

Forward problem functions

addNoise

Adds noise to measurements based on the specified signal-to-noise ratio.

Usage: `[model] = addNoise(model, SNRdb);`

- *model* - structure with a numerical model description.
- *SNRdb* - Signal to Noise Ratio (SNR) of the measurement.

boundaryVoltageVector

Identifies points with boundary conditions, finds points with unknown potential, sets the voltage on electrodes and screens (Dirichlet), and generates a linear vector (1D) of voltage distribution for a given electrode number. The matrix B is constructed using the column vectors of boundary voltage.

Important: The voltage can be gradually reduced at the tips of the electrodes to avoid singular points at sharp edges using a smoothing coefficient.

Usage: `[model] = boundaryVoltageVector(model, varargin);`

- *model* - structure with a numerical model description.
- *varargin* (if used):
  - *scoeff* - optional value in the range (0, 0.5] which controls the smoothness.

calculateComponents

Calculates values of measurement components, used by the `calculateMeasurement` function.

Usage: `[model] = calculateComponents(model);`

- *model* - structure with a numerical model description.
- Calculated values include:
  - *model.C* - Capacitance.
  - *model.G* - Conductivity.
  - *model.Y* - Admittance.

calculateElectricField

Calculates electric field vectors at the points of a nonuniform mesh stored in the qt structure. Electric field components [Ex, Ey] are stored in two matrices: qt.Ex and qt.Ey, where the number of rows equals the number of leaves, and the number of columns equals the number of excitations. Varargin is used only by the `calculatePotential` function.

Usage: `[model] = calculateElectricField(model, varargin);`

- *model* - structure with a numerical model description.
- *varargin* (if used by `calculatePotential`):
  - *mode* - 'mldivide' (default) or 'bicgstab'; 'bicgstab' is iterative and suggested for 3D simulations with an excessive number of mesh elements.
  - *tol* - tolerance value for the 'bicgstab' algorithm, default is 1e-3.

calculateMeasurement

Calculates values of measurements.

Usage: `[model] = calculateMeasurements(model);`

- *model* - structure with a numerical model description.
- Calculated values:
  - *model.K* - Complex capacitance.
  - Uses `calculateComponents` to obtain:
    - *model.C* - Capacitance.
    - *model.G* - Conductivity.
    - *model.Y* - Admittance.

calculatePotential

Calculates electric field potential at the points of a nonuniform mesh stored in the qt structure. If a 3D simulation in Matlab results in an "out of memory" error, an iterative calculation can be used by setting the mode to 'bicgstab'.

Usage: `[model] = calculatePotential(model, mode, tol);`

- *model* - structure with a numerical model description.
- *mode* - 'mldivide' (default) or 'bicgstab'; 'bicgstab' is iterative and is suggested for 3D simulations with an excessive number of mesh elements.
- *tol* - tolerance value for the 'bicgstab' algorithm, default is 1e-3.

calculateSensitivityMaps

Calculates sensitivity at the points of a nonuniform mesh stored in the qt structure using a scalar product with the conjugate. Sensitivity is stored in sparse vectors S(elem,pair), where the number of rows equals the number of leaves and the number of columns equals the number of electrode pairs. The calculated sensitivity depends on the size of the pixel and requires normalization before interpolation to a uniform mesh.

Usage: `[model] = calculateSensitivityMaps(model);`

- *model* - structure with a numerical model description.

electrodePairs

Calculates two indices of electrode numbers for electrode pairs (application electrode, sensing electrode). The position in the vectors corresponds to the order of measurement of mutual capacitance of electrodes.

Usage: `[application, receiving] = electrodePairs(elecNum, all);`

- *elecNum* - number of electrodes in the sensor.
- *all* - 0 for without, 1 with repeating measurements (1-2 and 2-1).

- *application* - vector with application electrode numbers.
- *receiving* - vector with sensing electrode numbers.

findBoundaryCondInd

Finds mesh points where:

- Boundary conditions are set (every point in the model that has a known value of electric potential).
- Potential is switched (e.g., excitation electrodes).
- Potential is unknown and should be calculated.

Usage: `[bcInd, elInd, calcInd] = findBoundaryCondInd(model);`

- *model* - structure with a numerical model description.
- Returns three sets of point indices.

findElement

Finds an element by name within a cell array and returns its index. If no element with the given name exists, the function returns 0.

Usage: `[n] = findElement(name, cellArray);`

- *name* - The name of the element.
- *cellArray* - Could be model.elements or model.sensor.
- *n* - The index of the element.

findIndex

Finds indices in a mesh based on the element name.

Usage: `index = findIndex(model, element_name);`

- *model* - Structure with a numerical model description.
- *element_name* - The name of the element.

getConductivityMap

Creates a vector of conductivity values for the elements in the workspace.

Usage: `[sigmaMap] = getConductivityMap(model);`

- *model* - Structure with a numerical model description.
- *sigmaMap* - Map of conductivity values normalized by (2\pi f \epsilon_0).

getElementMap

Generates a pixel map of elements in the model, where pixel values correspond to the unique number of each element.

Usage: `[epsilon_map, sigma_map] = getElementMap(model);`

- *model* - Structure with a numerical model description.
- *epsilon_map* - 2D matrix containing epsilon values for the model.
- *sigma_map* - 2D matrix containing sigma values for the model.

getPermittivityMap

Creates a vector of permittivity values for the elements in the workspace.

Usage: `[epsMap] = getPermittivityMap(model);`

- *model* - Structure with a numerical model description.
- *epsMap* - Map of relative permittivity values.

sensorElectrodePairs

Sets the number of electrode pairs and assigns excitation voltages to both excitation and receiving electrodes.

Usage: `[model] = sensorElectrodePairs(model);`

- *model* - Structure with a model description. This function requires some sensor fields to be set before running: sensor.measurements_all.

setElectrodes

Identifies and enumerates electrodes based on their excitation potential within the model.

Usage: `[model] = setElectrodes(model);`

- *model* - Structure with a numerical model description.

Inverse problem functions

---

clearFields

Function to remove specified fields from structures to reduce their memory usage.

Usage: `model = clearFields(model, nestedFieldsToRemove);`

- *model* - A cell array containing model names as strings (e.g., `{'model1', 'model2'}`).
- *nestedFieldsToRemove* - A cell array containing nested field names to be removed (e.g., `{'qt.vt', 'dd'}`).

defineMeshInvp

Distributes points (pixels) over the workspace and creates two matrices, X and Y, which store the x and y coordinates representing the grid. The input values are the number of points spread along the width and height of the workspace.

Usage: `[model] = defineMeshInvp(model, widthPoints, heightPoints, depthPoints);`

- *model* - Structure with a numerical model description.
- *widthPoints* - Mesh width.
- *heightPoints* - Mesh height.
- *depthPoints* - Mesh depth (optional, for 3D).

downscaleModel

Interpolates a model to a coarser mesh used in the inverse problem. This process involves decimation and sensitivity adjustment for the complex permittivity distribution within the model.

Usage: `[model] = downscaleModel(mesh, model);`

- *mesh* - Inverse problem mesh.
- *model* - Structure with a numerical model description.

findIndexInvp

Finds indices in a coarse mesh for the inverse problem.

Usage: `[Index] = findIndexInvp(modelInvp, element_name);`

- *modelInvp* - Structure with a numerical model description.
- *element_name* - Name of the element.

Landweber

Performs the Landweber iterative algorithm.

Usage: `[invp] = Landweber(invp, maxiter, alpha);`

- *invp* - Structure with an inverse model description.
- *maxiter* - Maximum number of iterations.
- *alpha* - Relaxation factor.

LBP

Performs the Linear Back-Projection algorithm.

Usage: `[invp] = LBP(invp);`

- *invp* - Structure with an inverse model description.

PINV

Performs the Moore–Penrose pseudoinverse operation.

Usage: `[invp] = PINV(invp);`

- *invp* - Structure with an inverse model description.
- *tol* - Damping parameter value (optional).

semiLM

Performs the semilinear Levenberg-Marquardt algorithm.

Usage: `[invp] = semiLM(invp, maxiter, alpha, lambda, maxUpdate);`

- *invp* - Structure with an inverse model description.
- *maxiter* - Maximum number of iterations.
- *alpha* - Relaxation factor.
- *lambda* - Damping parameter value.
- *maxUpdate* - Maximum number of sensitivity matrix updates.

upscaleModel

Interpolates the permittivity map to a finer qt mesh. This function requires the expansion of FOV data (extrapolating at FOV's edges).

Usage: `[model] = upscaleModel(eps_map, model);`

- *eps_map* - Permivitity map from an inverse problem mesh.
- *model* - Structure with a numerical model description.

withoutRepetition

Converts sensitivity matrices from 'all measurements' to 'without repetitions', reducing the number of rows in the sensitivity matrix by half.

Usage: `[model] = WithoutRepetition(model, modelList);`

- *model* - Vector with application electrode numbers.

- *modelList* - List of matrices to convert; example matrices include `{min, max, pha}`.

Visualization

---

drawElectricField

Draws a selected component or modulus of the electric field.

Usage: `[] = drawElectricField(model, mode, part, comp, electrode, method);`

- *model* - Numerical model structure.
- *mode* - Display mode ('px' for pixels, 'mm' for millimeters).
- *part* - Part of the field to display ('real', 'imag').
- *comp* - Component of the field to display ('Ex', 'Ey', 'Em' for modulus).
- *electrode* - 0 to map potential for every electrode, or a specific electrode number to map only that electrode.
- *method* (optional) - Method of presentation for 3D ('mpr' or 'slice').

drawInterpreter

Interprets parameters selected by the user, used by `drawMap` and `drawInvpMap`.

Usage: `[sets] = drawInterpreter(varargin);`

Varargin values can be in any order:

- *sets.mode* - 'mm' or 'px'.
- *sets.part* - 'real' or 'imag', applicable to potential, electric field, and sensitivity matrix.
- *sets.electrode* - Number of electrode or pair of electrodes.
- *sets.method* - 'mpr', 'surf', or 'slice' (only for 3D).
- *sets.ix* - Indices of mesh elements to present; 0 indicates the whole matrix will be presented.

drawInvpMap

Draws maps of selected parameters for the inverse problem.

Usage: `[] = drawInvpMap(model, parameter, varargin);`

- *model* - Structure with numerical model description.
- *parameter* - Parameter to be presented:
  - 'permittivity' - Permittivity distribution.
  - 'conductivity' - Conductivity distribution.

Varargin values are interpreted by `drawInterpreter` and can be provided in any order.

drawMap

Draws maps based on selected parameters.

Usage: `[] = drawMap(model, parameter, varargin);`

- *model* - Structure with numerical model description.
- *parameter* - Parameter to be presented:
  - 'V' - Electric potential distribution.
  - 'Em', 'Ex', 'Ey', 'Ez' - Electric field distribution components.
  - 'S' - Sensitivity matrix.
  - 'pattern' - List of elements in the model.
  - 'permittivity', 'epsilon' - Permittivity distribution.
  - 'conductivity', 'sigma' - Conductivity distribution.

Varargin values are interpreted by `drawInterpreter` and can be provided in any order.

drawPatternImage

Draws a pattern image representing the distribution of objects in the model.

Usage: `[] = drawPatternImage(model, mode, method);`

- *model* - Numerical model structure.
- *mode* - 'px' for pixels, 'mm' for millimeters.
- *method* (optional) - Presentation method for 3D ('surf', 'mpr', or 'slice').

drawPotential

Draws potential maps for selected excitations.

Usage: `[] = drawPotential(model, mode, part, electrode, method);`

- *model* - Numerical model structure.
- *mode* - 'px' for pixels, 'mm' for millimeters.
- *part* - 'real' or 'imag'.
- *electrode* - 0 for potential maps of every electrode, >0 for only the selected electrode.
- *method* (optional) - Presentation method for 3D ('mpr' or 'slice').

drawSensitivityMap

Draws sensitivity maps for specified electrodes.

Usage: `[] = drawSensitivityMap(model, mode, part, draw, method);`

- *model* - Numerical model structure.
- *mode* - 'px' for pixels, 'mm' for millimeters.
- *part* - 'real' or 'imag'.
- *draw* - Application electrode number or a pair of electrodes [e1, e2], e.g., [2, 13].
- *method* (optional) - Presentation method for 3D ('mpr' or 'slice').

MPR

Multiplanar reconstruction (MPR) is a method for 3D data presentation that displays three cross-sections (xy, xz, yz) in separate images. Users can change the displayed image by clicking in the desired location with the left mouse button, where crosshairs indicate the current position. Dragging the right mouse button adjusts the 'w' and 'c' parameters of windowing. This method can be used by `drawMap` and `drawInvpMap` for 3D data.

Usage: `[] = mpr(data, varargin);`

- *data* - 3D matrix with values to be presented.
- If *varargin*:
  - *mesh* - Structure with X, Y, and optionally Z meshgrid lists of pixel coordinates.

oneSliceView

This function presents a single slice from a dataset, primarily used for 2D data presentation. Dragging the right mouse button adjusts the 'w' and 'c' parameters of windowing. This method is used by `drawMap` and `drawInvpMap` for 2D data.

Usage: `[] = oneSliceView(data, varargin);`

- *data* - 2D matrix with values to be presented.
- If *varargin*:
  - *mesh* - Structure with X and Y meshgrid lists of pixel coordinates.

plotMeasurement

Plots mutual measurements of electrodes, with a maximum of 8 plots at a time.

Usage: `[] = plotMeasurements(mode, part, index, modelList, nameList);`

- *mode* - Display mode ('linear', 'log').
- *part* - Type of measurement ('C' for capacitance, 'G' for conductance).
- *index* - Range of displayed pairs of electrodes (e.g., [1:31]).
- *modelList* - List of models (e.g., `{modelMin, modelMax}`).
- *nameList* (optional) - Names for capacitance vectors used in legends (e.g., `{'min', 'max'}`); defaults to model1, model2, etc., if not provided.

plotNorm

Plots norms calculated during the reconstruction process.

Usage: `[] = plotNorm(parameter, model, name, part);`

- *parameter* - Norm to plot ('residue' or 'error').
- *model* - Structure with a numerical model description of the inverse problem.
- *name* - List of model names (e.g., `{'LBP', 'PINV'}`).
- *part* (optional) - Specifies whether to plot the real or imaginary part of the norm.

shadedSurfaceDisplay

Presents 3D data using Phong shading, primarily used for displaying permittivity and conductivity distributions by `drawInvpMap` and `drawMap`.

Usage: `[] = shadedSurfaceDisplay(patternImage, varargin);`

- *patternImage* - 3D matrix with parameter values to be presented.
- If *varargin*:
  - *mesh* - Structure with X, Y, and Z meshgrid lists of pixel coordinates.

shadedSurfaceDisplayPattern

A specific version of `shadedSurfaceDisplay` used for presenting lists of model elements, typically used by `drawPatternImage` with the 'surf' method.

Usage: `[] = shadedSurfaceDisplayPattern(patternImage, varargin);`

- *patternImage* - 3D matrix with numbers indicating the numbering of elements in the model.
- If *varargin*:
  - *mesh* - Structure with X, Y, and Z meshgrid lists of pixel coordinates.

sliceView

Presents a single image slice from 3D data (a slice across the Z-axis). Users can view different cross-sections by clicking and dragging the left mouse button from left to right or bottom to top. Dragging the right mouse button adjusts the 'w' and 'c' parameters of windowing. This method is used by `drawMap` and `drawInvpMap` for 3D data.

Usage: `[] = sliceView(data, varargin);`

- *data* - 3D matrix with values to be presented.
- If *varargin*:
  - *mesh* - Structure with X, Y, and Z meshgrid lists of pixel coordinates.

Contact information

This is documentation of the ECTsim toolbox.

Questions? Contact us at damian.wanta@pw.edu.pl or waldemar.smolik@pw.edu.pl

Visit our homepage: https://ectsim.ire.pw.edu.pl/

Data Acquisition and Processing Lab

Institute of Radioelectronics and Multimedia Technology

Warsaw University of Technology

Nowowiejska 15/19

00-665 Warsaw

Poland