1. How to use

Below you will find an overview of all (important) EMerge functions. Click on a function or method name to jump to its description further on. The header contains a link for each category (**Geometry**, **Meshing**, etc.); click those to jump back to the overview pages.

1.1 What changed in EMerge 3.0

This sheet covers EMerge 3.0.4. Scripts written for 2.x need these changes (see the CHANGELOG for the full list):

- Solvers are classes: `em.EMSolver.PARDISO` → `model.set_solver(em.solvers.Pardiso)`. Hardware with `model.set_device("CPU" | "GPU" | "HYBRID")`, threads with `em.set_threads(n)`.
- `em.geo.PCBNew` → `em.geo.PCB`; `pcb.z(i)` arguments → layer indices (`trace_layer=`, `layer=`, `layer1=`/`layer2=`; `-1` is the top).
- `pcb.modal_port(...)` / `pcb.lumped_port(...)` take a port number and create the boundary condition; get it with `model.mw.bc.get_port(n)`.
- `commit_geometry()` takes no arguments; the default save format is `msgpack` (`store_system="joblib"` for lumped element functions).
- `run_sweep(parallel=True, n_workers=N)` → `run_sweep_distributed(n_workers=N)`; `multi_processing=True` → `worker_type="process"`.
- `farfield()` returns an `EHFieldFF`; `farfield_3d()` arrays are indexed `(theta, phi)`; `ScatteredField` uses spherical angles (`definition='EA'` for the old ones).
- Multi-mode ports with different β: `WavePort(..., number_of_modes=N)` instead of `ModalPort`; `WavePortIH` → `WavePort(..., modetype="TEM")`.
- `emerge.beta.*` → `emerge.read` / `emerge.write`; `adaptive_mesh_refinement()` and `run_adaptive_sweep()` have new arguments.

Good luck and happy simulating!

Robert Fennis

2. Global Functions

2.1 Global States and Definitions

Module / Class	Function / Method	Page
em	CENTER	8
	CORNER	8
	EMSolver	8
	GCS	8
	Material	8
	cleanup	8
	coax_rin	9
	coax_rout	8
	lib	8
	lumped_element_material	8
	uniaxial_tensor	8

2.2 Solvers and Threads

Module / Class	Function / Method	Page
Solver	set_nthreads	9
	set_options	9
em	get_threads	9
	set_threads	9
	solvers	9
em.solvers	available	9
em.solvers.CuDSS	set_precision	9
em.solvers.Sparta	set_options	9

2.3 Materials and Physical Attributes

Module / Class	Function / Method	Page
GeoObject	properties	9
em	FiniteThickness	9
	LumpedElementAttribute	10
	LumpedPortAttribute	10
	MetalCoating	9
	SurfaceRoughness	9
	VoidAttribute	10
	WavePortAttribute	10

2.4 Simulation Class

Module / Class	Function / Method	Page
em	Simulation	10
em.Simulation	activate	11
	adaptive_mesh_refinement	13
	all_boundaries	11
	all_geos	11
	cache_build	10
	cache_run	10
	check_version	10
	clean	13
	commit_geometry	11
	conducting_surfaces	13
	conducting_volumes	13
	export	12
	find_geo	12
	from_simulations	13
	generate_mesh	12
	get_solver	12
	load	11
	parameter_sweep	12
	ping	10
	quick_mesh	12
	reset	10
	save	11
	set_device	12
	set_loglevel	11
	set_periodic_cell	11
	set_physics	11
	set_scale_factor	13
	set_solver	12
	set_write_log	13
	view	11

3. GEOMETRY

3.1 Primitives

Module / Class	Function / Method	Page
em	Anchor	15
	Axis	13
	CoordinateSystem	14
	Line	15
	Plane	14
	cs	14
em.Axis	construct_cs	14
	cross	13
	dot	13
	neg	14
	pair	13
	tensor	14
em.CoordinateSystem	affine_from_global	15
	affine_to_global	15
	copy	14
	displace	14
	flip_z	14
	in_global_basis	15
	in_global_cs	15
	in_local_basis	15
	in_local_cs	15
	rotate	14
	swapxy	14
em.Line	cmid	15
	cpoint	15
	from_path	15
	from_points	15
	line_integral	16
	smooth	15
	subsample	15
em.Plane	cs	14
	flip	14
	grid	14
	span	14

3.2 3D Geometry

Module / Class	Function / Method	Page
em.geo	Box	16
	CoaxCylinder	16
	Cone	16
	Cylinder	16
	HalfSphere	16
	Horn	16
	STEPItems	17
	Sphere	16
	XYPolygon	17
	open_pml_region	16
	open_region	16
	pmlbox	17

3.3 2D Geometry

Module / Class	Function / Method	Page
em.geo	Disc	17
	Plate	17
	XYPlate	17

3.4 Curves

Module / Class	Function / Method	Page
em.geo	Curve	17
	Point	18
em.geo.Curve	helix_rh	17
	pipe	17

3.5 Operations

Module / Class	Function / Method	Page
em.geo	add	19
	bounding_box	20
	change_coordinate_system	19
	connect_faces	20
	duplicate	20
	embed	19
	expand_surface	20
	extrude	19
	intersect	19
	mirror	19
	remove/subtract	19
	rotate	19
	stick	20
	stretch	19
	thick_wall	20
	translate	19
	unite	20

3.6 XYPolygon

Module / Class	Function / Method	Page
XYPolygon	circle	18
	connect	19
	corrugated_line	18
	extend	18
	extrude	18
	fillet	18
	geo	18
	incs	18
	iterate	18
	parametric	18
	rect	18
	revolve	18
	shrink	19

3.7 GeoObject methods

Module / Class	Function / Method	Page
GeoObject	above	20
	background	20
	foreground	20
	give_name	20
	hide	20
	make_copy	20
	prio_set	20
	remove	20
	selection	21
	set_material	20
GeoSurface	assign_thickness	21
GeoVolume	all_faces	21
	boundary	21
	exterior_faces	21
	face	21
	face_from_anchor	21
	faces	21

3.8 Selections

Module / Class	Function / Method	Page
FaceSelection	normal / area	22
	rect_basis	22
	sample	22
Selection	around	22
	below	22
	centers / getCAs	22
	constrain	22
	exclude	22
	isolate	22
Selector	inlayer	21
	inplane	21
	near	21
em	select	21
em.Simulation	select	21

3.9 PCB Layouting

Module / Class	Function / Method	Page
em.geo	PCB	22
	PCBLayer	22
em.geo.PCB	add_hole	24
	add_poly	24
	add_vias	23
	compile_paths	24
	determine_bounds	23
	generate_air	23
	generate_all	24
	generate_pcb	23
	generate_silkscreen	24
	generate_vias	24
	load	23
	lumped_port	23
	lumped_port_pts	24
	modal_port	24
	new	23
	plane	23
	radial_stub	23
	set_board_outline	24
	set_bounds	23
	z	22
em.geo.PCB.calc	effective_er	25
	z0	24

3.10 StripPath

In practice you will call these methods after calling the pcb.new(...) function. Not on a manually created `StripPath` object. That won't work.

Module / Class	Function / Method	Page
em.geo.StripPath	bck	25
	curve	25
	cut	26
	end	25
	init	26
	jump	26
	lumped_element	25
	lumped_element_from_sp	26
	macro	27
	merge	26
	move	26
	pturn	25
	short	26
	skip	25
	split	25
	start	25
	store	25
	straight	25
	stub	26
	taper	25
	to	26
	turn	25
	via	26
	xs/ys	25

4. MESHING

Module / Class	Function / Method	Page
em.Mesher	set_algorithm	28
	set_boundary_size	27
	set_curved_boundary_meshing	27
	set_domain_size	27
	set_face_size	27
	set_global_max_size	27
	set_pec_domain	28
	set_pec_face	27
	set_periodic_cell	28
	set_size	27
	set_trace_refinement	27
	unset_constraints	27
em.Simulation	set_resolution	27

5. PHYSICS

5.1 Microwave Physics

Module / Class	Function / Method	Page
Microwave3D	conducting_surfaces	29
	eigenmode	29
	modal_analysis	28
	ports	29
	reset	28
	run_adaptive_sweep	29
	run_scattered	29
	run_sweep	28
	run_sweep_distributed	28
	set_basis_space	29
	set_frequency	28
	set_frequency_range	28
	set_resolution	28
	solve_cnosdd	29

5.2 Boundary Conditions

The following functions are called on:

python
```
1 pec_bc = my_model.mw.bc.PEC(...) # etc
```

Module / Class	Function / Method	Page
AbsorbingBoundary	optimize_for_maximum_angle	30
	set_zero_angles	30
FloquetPort	floquet_modes	32
	order_direction	33
	set_polarization	33
MWBoundaryConditionSet	AbsorbingBoundary	30
	CoaxPort	33
	FloquetPort	32
	LumpedElement	30
	LumpedPort	33
	ModalPort	31
	PEC	29
	PMC	29
	RectangularWaveguide	33
	ScatteredField	30
	SurfaceImpedance	31
	ThinConductor	31
	UserDefinedPort	33
	Void	31
	WavePort	31
	WavePortIH	31
	floquet_port	32
	get_port	33
	get_ports	33
ModalPort	align_modes	32
	reset	32
	set_integration_line	32
	set_terminals	32
PortBC	assume_homogeneous	32
	smatrix_index	32
RectangularWaveguide	align_modes	33
ScatteredField	set_excitations	30

6. Data

6.1 Generic

Module / Class	Function / Method	Page
em	MWData	33
	MWField	34
em.MWData	export_farfields	34
	merge_with	34
em.MWField	boundary	35
	cache	37
	combine_ports	34
	current_boundary	35
	cutplane	35
	cutplane_normal	35
	embed_external_component	37
	excite_port	34
	farfield	36
	farfield_2d	35
	farfield_3d	36
	farfield_3d_ports	36
	grid	35
	in_domain	36
	int_line	35
	int_lumped_element	35
	int_surf	34
	int_vol	34
	interpolate	34
	nearfield	36
	relative	34
	set_excitations	34
	trace_poynting_lines	36

6.2 S-parameters

Module / Class	Function / Method	Page
em	MWScalarNdim	37
em.MWScalarNdim	S	37
	Smat	37
	combine_ports	37
	dense_f	37
	embed_external_component	37
	export_touchstone	38
	model_S	37
	model_Smat	37
	renormalize	37
	save_smatrix	38

6.3 Datasets

Module / Class	Function / Method	Page
BaseDataset	axis	38
	filter	38
	find	38
	get_entry	38
	grid	38
	iter	38
	select	38
	slice_set	38
MWData.field/scalar	Datasets	38
data.sim	find	38
	iterate	38

6.4 EHField and EHFieldFF classes

The **EHField** class contains all the different possible electric and magnetic fields (E,H,D,B,P,S,Sm,J,Q) and the coordinates where they where computed. It also contains metadata on the structure of the underlaying coordinates (2D grid, 3D grid, unstructured grid, triangulated surface etc.).

All field attributes are: E, H, S, P, D, B, Ex, Ey, Ez, Hx, ... etc, Sm (time averaged Poynting vector), Js, Jv, normE, normB, normS, normSm etc.

The **EHFieldFF** class contains the following farfield properties: E, H, Ex, Ey, Ez, Hx, Hy, Hz, Etheta, Ephi, Htheta, Hphi, Erhcp, Elhcp, AR, gain, dir, RCS.

Some attributes allow for further sub attribute specifications as they return an object of type `FarFieldComponent` . This class itself has the attributes: x, y, z, theta, phi, lhcp, rhcp and norm. For example:

```python
farfield = field.farfield_3d()
farfield.Etheta #one way
farfield.E.theta #using FarFieldComponent
farfield.gain.theta
farfield.dir.lhcp
```

Technicall the EHfield and EHFieldFF classes are not part of EMerge but part of the shared `emsutil` library which is also used by Optycal.

Module / Class	Function / Method	Page
em	EHField	39
	EHFieldFF	39
	FarFieldComponent	39
em.EHField	int	39
	scalar	39
	vector	39
em.EHFieldFF	E	39
	H	39
	ang	39
	dir	39
	gain	39
	surfplot	39
	total_radiated_power_integral	39

7. Display

Module / Class	Function / Method	Page
EMergeDisplay	add_clip_volume	43
	add_cloud	43
	add_contour	43
	add_quiver	43
	add_scatter	42
	add_streamline	43
	add_surf	42
	add_text	44
	add_title	43
	add_trisurf	43
	add_vector_cloud	42
	animate	42
	cbar	41
	drawing_bw	42
	explode	41
	orbit	41
	save_vtk	42
	set_theme	41
	show	41
	view_x / view_y / view_z / view_iso	42
PVDisplay	add_anchors	40
	add_backgroundfields	41
	add_farfield3d	40
	add_field	42
	add_line	40
	add_object	40
	add_objects	40
	add_particle_lines	41
	add_portmode	40
	add_scatter	40
	add_solution_error	41
	populate	40

7.1 2D Plotting

Module / Class	Function / Method	Page
em.plot	plot	44
	plot_ff	44
	plot_ff_polar	45
	plot_sp	44
	plot_vswr	44
	smith	44

8. Import, Export and Macros

8.1 Import and Export

Module / Class	Function / Method	Page
FileBasedPCB	layer_from_file	45
	vias_from_file	45
em	TouchstoneData	45
emerge.optycal	farfield_model	46
	surface_model	46
emerge.read	FileBasedPCB	45
	GDSFile	45
	GDSFromXML	45
	ODBImport	45
	import_dxf	45
emerge.write	export_dxf	45
	pcb_to_dxf	46

8.2 Macros and Waveguide Assemblies

Module / Class	Function / Method	Page
em.macro	CoaxAssy	46
	SMAConnector	46
	design_yagi	47
	vivaldi_taper	46
em.wg	WGConfig	47
	WGETurnRound / WGHTurnRound / WGETurnMiter / WGHTurnMiter	47
	WGRect	47
em.wg component	place	47

9. GLOBAL

9.1 Global States and Definitions

FUNCTION

em.cleanup()

Clean-up all variables from the global state thus reducing the RAM requirements to the minimum possible.

RETURNS None

PROPERTY

em.GCS

The Global coordinate system object. See `CoordinateSystem`.

RETURNS em.CoordinateSystem

PROPERTY

em.CORNER

An Enum object that is used by CAD modelling to tell EMerge which point of a rectangle or box to align the position with. The corner corresponds to the most -X, -Y and -Z point (lower, left, front).

Example:

```python
box = em.geo.Box(1,1,1,(0, 2, 5), alignment=em.CORNER)
```

RETURNS em.Alignment

PROPERTY

em.CENTER

An Enum object that is used by CAD modelling to tell EMerge which point of a rectangle or box to align the position with.

Example:

```python
box = em.geo.Box(1,1,1,(0, 2, 5), alignment=em.CENTER)
```

RETURNS em.Alignment

PROPERTY

em.EMSolver

Removed in 3.0. Solvers are now selected by class from em.solvers. The enum is kept for a few versions so old scripts still autocomplete, but accessing a member raises an error that names its replacement, e.g. `em.EMSolver.PARDISO` → `em.solvers.Pardiso`.

RETURNS em.EMSolver

PROPERTY

em.lib

The built-in material library (from `emsutil`). Common entries: `em.lib.AIR`, `em.lib.PEC`, `em.lib.VACUUM`, metals `MET_COPPER`, `MET_GOLD`, `MET_ALUMINUM`, `MET_SILVER`, …, semiconductors `SEMI_SILICON`, `SEMI_GAAS`, `SEMI_AL2O3`, …, substrates `DIEL_FR4`, `DIEL_RO4350B`, `DIEL_RO4003C`, `DIEL_RO3003`, `DIEL_PTFE`, …, foams `FOAM_ROHACELL_31`, … and solder masks `SILK_LPI_GREEN`, …. `em.isola` and `em.rogers` hold the full laminate catalogues.

```python
sub = em.geo.Box(10*mm, 10*mm,
0.5*mm).set_material(em.lib.DIEL_RO4350B)
```

RETURNS module

FUNCTION

em.Material()

The Material class generalizes a material in the EMerge FEM environment.

A scalar `er`/`ur` is used as an isotropic tensor. Frequency or coordinate dependent properties are given with `em.FreqDependent(...)`, `em.CoordDependent(...)` or `em.FreqCoordDependent(...)`.

PARAMETERS
- **er**: float | complex | ndarray | MatProperty = 1.0 - Relative permittivity (scalar, 3x3 tensor or MatProperty)
- **ur**: float | complex | ndarray | MatProperty = 1.0 - Relative permeability
- **tand**: float | MatProperty = 0.0 - Dielectric loss tangent
- **cond**: float | MatProperty = 0.0 - Electric conductivity in S/m
- **density**: float | MatProperty = 1.0 - Mass density (heat conduction)
- **cond_thermal**: float | MatProperty = 1.0 - Thermal conductivity in W/(m·K)
- **specific_heat**: float | MatProperty = 1.0 - Specific heat in J/(kg·K)
- **color**: str = “#BEBEBE” - Display color (hex)
- **opacity**: float = 1.0 - Display opacity
- **name**: str | None = None - Material name

RETURNS Material

FUNCTION

em.uniaxial_tensor()

Generate a uniaxial tensor for a given axis and an along-axis value and in plane value (normal to the axis)

Returns $\varepsilon_\perp \cdot I + (\varepsilon_\parallel - \varepsilon_\perp)\cdot \hat{n}\hat{n}^T$. `Axis.tensor(ax_value, plane_value)` does the same for an Axis object.

PARAMETERS
- **axis**: ndarray | tuple | Axis - The the uniaxial axis
- **ax_value**: complex - The value along the axis
- **plane_value**: complex - The value normal to the axis

RETURNS ndarray

FUNCTION

em.lumped_element_material()

Generates a material that effectively turns some volume into a lumped element. The volume must be approximately that of an extruded surface like a box or cylinder. This extruded volume must be extruded in the current direction and the start and end surfaces must be a conductor. Then the volume in between can be assigned this material which will turn the volume into an effective lumped element with the provided capacitance, inductance, resistance or mix of all.

PARAMETERS
- **material_name**: str - The material name.
- **direction**: tuple[float, float, float] | Axis - The direction in which the lumped element is oriented. This is the current direction.
- **length**: float - The length of the lumped element component.
- **Area**: float - The cross sectional area of the lumped element component.
- **C**: float | None = None - The capacitance in F.
- **L**: float | None = None - The inductance in H.
- **R**: float | None = None - The resistance in Ω.
- **color**: str = “#999999” - The rendering color of the material
- **opacity**: float = 0.2 - Trendering opacity of the material.

RETURNS Material

FUNCTION

em.coax_rout()

Computes the outer radius given a dielectric constant, inner radius and characteristic impedance

PARAMETERS
- **rin**: float - The inner radius
- **eps_r**: float = 1 - The dielectric permittivity.
- **Z0**: float = 50 - The impedance.

RETURNS float

FUNCTION

em.coax_rin()

Computes the inner radius given a dielectric constant, outer radius and characteristic impedance

PARAMETERS
- **rout**: float - The outer radius
- **eps_r**: float = 1 - The dielectric permittivity.
- **Z0**: float = 50 - The impedance.

RETURNS float

9.2 Solvers and Threads

PROPERTY

em.solvers

All solvers of EMerge as classes. Select one with `model.set_solver(em.solvers.X)`. Solvers whose library is not installed raise an error with install instructions when selected. Each class has `.available` and `.requires`.

- Direct: `SuperLU`, `UMFPACK`, `Pardiso` (x86, MKL), `MUMPS`, `AASDS` (Apple Accelerate, installed by default on macOS), `CuDSS` (NVIDIA GPU, `pip install emerge[cudss]`), `CHOLMOD`, `RSLab`, `Sparta` (recommended: fastest CPU solver with Apple GPU and hybrid modes; picked automatically when installed)
- Iterative: `BiCGSTAB`, `CG` (heat conduction), `Corvid`
- Eigenvalue: `LAPACK`, `ARPACK`

```python
model.set_solver(em.solvers.Sparta).set_options(mode='CPU').s
print(em.solvers.available())
```

RETURNS module

FUNCTION

em.solvers.available()

The solvers of em.solvers that can be used on this system.

RETURNS list[type[Solver] | type[EigSolver]]

FUNCTION

em.set_threads()

Set the number of CPU threads EMerge uses, e.g. em.set_threads(6). The count applies to the compiled kernels (matrix assembly, interpolation, far fields, mesh processing) and is the default thread count of the solvers that support one. An nthreads argument of a run method, e.g. run_sweep(nthreads=4), still takes precedence for the solver. It can be called at any time. run_sweep_distributed() solves n_workers frequency points at the same time, each with this many solver threads.

PARAMETERS
- **nthreads**: int | None - The number of threads. None returns to the defaults: all logical cores for the kernels (or RAYON_NUM_THREADS) and each solver's own setting.

RETURNS None

FUNCTION

em.get_threads()

The number of CPU threads EMerge uses (set with em.set_threads, otherwise the kernels' default).

RETURNS int

FUNCTION

Solver.set_options()

Write generic simulation options to the solver object. Options may be ignored depending on the type of solver used.

Returns the solver so calls can be chained.

PARAMETERS
- **pivoting_threshold**: float | None = None - The pivoting threshold of direct solvers.

RETURNS Solver

FUNCTION

Solver.set_nthreads()

Install the preferred number of CPU threads for the next solves. Solvers without a thread count implementation ignore it.

`run_sweep(nthreads=...)` and the other run methods set this per solve.

PARAMETERS
- **nthreads**: int | None - The number of threads. None leaves the choice to the solver backend.

RETURNS Solver

FUNCTION

em.solvers.Sparta.set_options()

Set the SPARTA factorization mode: `'GPU'` (GPU only), `'CPU'` (CPU only) or `'HYBRID'` (combined GPU + CPU, Apple silicon). Follows `model.set_device()` by default.

PARAMETERS
- **mode**: "GPU" | "CPU" | "HYBRID"

RETURNS SolverSparta

FUNCTION

em.solvers.CuDSS.set_precision()

Sets the floating point precision of the cuDSS solve.

PARAMETERS
- **precision**: "auto" | "single" | "double" = "auto"

RETURNS None

9.3 Materials and Physical Attributes

PROPERTY

GeoObject.properties

Every geometry carries a set of physical attributes. Attributes are combined with `+` and added with `+=`. Ports, lumped elements, voids, surface impedance and thin-conductor boundary conditions are generated automatically from them.

```python
trace.properties += em.FiniteThickness(35e-6) +
em.SurfaceRoughness(1e-6)
box.properties += em.VoidAttribute()
```

RETURNS PhysicalAttributeSet

FUNCTION

em.FiniteThickness()

Assign a finite thickness to a (surface) object.

PARAMETERS
- **thickness**: float - The thickness in meters

RETURNS FiniteThickness

FUNCTION

em.SurfaceRoughness()

Assign a surface roughness to an object.

PARAMETERS
- **rmsval_m**: float - The RMS surface roughness in meters

RETURNS SurfaceRoughness

FUNCTION

em.MetalCoating()

Assign a metal coating to an object.

PARAMETERS
- **material**: Material - The coating material
- **thickness**: float - The coating thickness in meters

RETURNS MetalCoating

FUNCTION

em.**VoidAttribute**()

Marks a volume as a void: it is cut out of all other volumes and its boundary becomes PEC.

RETURNS VoidAttribute

FUNCTION

em.**WavePortAttribute**()

Mark a surface as a wave port.

PARAMETERS
- **port_number**: int - The port number
- **mode_type**: “TEM” | “TE” | “TM” = “TEM” - The mode type of the port.
- **assume_homogeneous**: bool = False - Treat the port face as homogeneous, see ModalPort.assume_homogeneous().
- **target_neff**: float | None = None - The effective index of the wanted mode, see ModalPort.
- **signal_point**: ndarray | None = None - A point (global coordinates) on the positive conductor of the port.

RETURNS WavePortAttribute

FUNCTION

em.**LumpedPortAttribute**()

Mark a surface as a lumped port.

PARAMETERS
- **port_number**: int - The port number
- **direction**: tuple[float, float, float] | Axis - The direction of the port excitation
- **width**: float - The port width
- **height**: float - The port height (along the direction)
- **impedance**: float = 50.0 - The port impedance. Defaults to 50.0.

RETURNS LumpedPortAttribute

FUNCTION

em.**LumpedElementAttribute**()

Mark a surface as a lumped element.

PARAMETERS
- **impedance_function**: Callable - The element impedance as a function of frequency
- **width**: float - The element width
- **height**: float - The element height

RETURNS LumpedElementAttribute

9.4 Simulation Class

FUNCTION

em.**Simulation**()

Creates an EMerge Simulation object. Used as the central point of contact for all simulation related activities.

PARAMETERS
- **modelname**: str - The name of the simulation model. This will be used for filenames and path names when saving data.
- **loglevel**: “TRACE” | “DEBUG” | “INFO” | “WARNING” | “ERROR” = “INFO” - The loglevel to use for loguru.
- **load_file**: bool = False - If the simulation model should be loaded from a file.
- **save_file**: bool = False - if the simulation file should be stored to a file.
- **write_log**: bool = False - If a file should be created that contains the entire log of the simulation.
- **path_suffix**: str = “.EMResults” - The suffix that will be added to the results directory. Defaults to “.EMResults”.
- **store_system**: “msgpack” | “joblib” = “msgpack” - The file format. `"msgpack"` (default since 3.0) is safe but cannot store Python functions such as lumped element impedance functions; use `"joblib"` for those. Never load `"joblib"` files from untrusted sources.
- **reset_settings**: bool = False - Reset the settings to their defaults. The settings are shared by every Simulation in the process, so changes made for an earlier Simulation otherwise carry over.

RETURNS Simulation

FUNCTION

em.Simulation.**ping**()

Send a message to the NTFY app system (downloadable on Android and iPhone). These are public messages that will be send to people who subscribe to channels. Use a custom message and channel name to notify if a simulation is finished for example.

IMPORTANT: As said, these channels are public. Anyone can subscribe to channels you send messages to. Do not share any sensitive information in these channels. Use very hard to predict channel names to keep things as private as possible.

PARAMETERS
- **message**: str - The message
- **channel**: str - The channel

RETURNS None

FUNCTION

em.Simulation.**reset**()

Whipe internal simulation data.

PARAMETERS
- **mesh**: bool = False - If the mesh should be deleted
- **geometry**: bool = False - If the Geometry should be deleted and GMSH should be reset.
- **physics**: bool = False - If the physics and boundary conditions should be deleted.
- **data**: bool = False - If all simulation data should be deleted
- **all**: bool = False - If everything should be deleted.

RETURNS None

FUNCTION

em.Simulation.**cache_build**()

Checks if all the lines in the .py file above this line are the same. If they are this function returns false. Can be used to cache the entire build and simulation process of a simulations:

```python
if model.cache_build():
    box = em.geo.Box(...)
    model.commit_geometry()
    # etc
    # all simulation steps here
    data = model.mw.run_sweep()
    model.save()
else:
    data = model.data.mw
```

RETURNS bool

FUNCTION

em.Simulation.**cache_run**()

Checks if all the lines in the .py file above this line are the same. If they are this function returns false. Can be used to cache the simulation solve steps:

```python
if model.cache_run():
    data = model.mw.run_sweep()
    model.save()
else:
    data = model.data.mw
```

RETURNS bool

FUNCTION

em.Simulation.**check_version**()

Ensure the script targets an EMerge major.minor compatible with the current runtime. Only major.minor is enforced. Patch differences are ignored.

PARAMETERS
- **target_version**: str - The EMerge version this script was written for (e.g. “1.4.0” or “1.4”).
- **log**: bool = False - If True, also emit a single WARNING with the same message as the exception.

RETURNS None

RAISES VersionError

FUNCTION

em.Simulation.**set_physics**()

Tells EMerge which physics are used. Mostly important to deactivate Microwave physics and activate Heat Conduction for example when EMerge defines the default mesh size. Microwave is turned on by default. In these cases a heat-conduction simulation only would require a simualtion frequency because EMerge thinks it is going to do a microwave simulation.

PARAMETERS
- **microwave**: bool - If you are running microwave simulations
- **heatconduction**: bool - If you are running heatconduction simulations.

RETURNS None

FUNCTION

em.Simulation.**activate**()

Tells EMerge which solution number of parameter to activate as the current geometry/mesh. In case of parameter sweeps you can use this to pick a specific parameter permuation.

PARAMETERS
- **_indx**: int | None = None - A specific solution index
- ****variables** - A list of keyword variables. For example: `width=10*mm` for some parameter sweep iteration.

RETURNS None

FUNCTION

em.Simulation.**save**()

Saves the current Simulation object to a file.

PARAMETERS
- **_force_save**: bool = True - Will always save the file no matter what. If set to False it will only save a file if actually new simulation results are generated.
- **screenshot**: bool = False - Will cause a screenshot of the simulation model to also be saved. Has some annoying consequences to hijacking PyVista later in the script.

RETURNS None

FUNCTION

em.Simulation.**load**()

Orders EMerge to load all data.

RETURNS None

FUNCTION

em.Simulation.**set_loglevel**()

Set a new log level.

PARAMETERS
- loglevel = “TRACE” | “DEBUG” | “INFO” | “WARNING” | “ERROR”

RETURNS None

FUNCTION

em.Simulation.**view**()

Shows the 3D geometry as it is currently. By Default it tries to use EMerges PyVista based viewer. If there is no Mesh yet it will generate a very coarse one. If you have a geometry state that cannot be meshed, set `use_gmsh=True` to use GMSH’s build in GUI which can display the Geometry without requiring a mesh step.

PARAMETERS
- **selections**: list[Selection] | None - A list of selections to highlight-color
- **use_gmsh**: bool = False - If the GMSH geometry viewer should be used.
- **plot_mesh**: bool = False - If the mesh should be shown instead of the geometry volumes.
- **volume_mesh**: bool = True - If true then plot_mesh will plot the volume mesh, if False it will only plot the mesh on the surface of objects.
- **opacity**: float | None = None - An upper limit to the opacity of geometries
- **labels**: bool = False - If true, labels will be added to each geometry showing the geometry names.
- **bc**: bool = False - If True, EMerge will draw textures on surface with boundary condition name labels
- **bw**: bool = False - If True, the view will be black and white and with an optimized style for printing on black and white paper.
- **face_labels**: bool = False - If true EMerge will add labels to all the different face names in the geometry
- **screenshot**: str | None = None - If a filename is provided, EMerge will automatically generate a screenshot to a file with the provided filename
- **assigned_materials**: bool = False - If False (default) EMerge will draw all geometries as defined, overlapping with their respective materials. If True, Emerge will draw all disjoint volume elements with the actual material rendered as will be simulated. In case of overlapping geometries, this can be used to validate if the correct materials are assigned.
- **off_screen**: bool = False - If true, no display will be shown. Used in conjuction with .screenshot().

RETURNS None

FUNCTION

em.Simulation.**set_periodic_cell**()

Provide a PeriodicCell object such as `HexCell` and `RectCell` to automatically configure simulations with periodic boundaries.

PARAMETERS
- **cell**: PeriodicCell - The periodic cell object.

RETURNS None

FUNCTION

em.Simulation.**commit_geometry**()

Tell EMerge that you are done defining the geometries. This will initiate a boolean fragment and **has** to be called before generating a mesh or defining selections or boundary conditions. Takes no arguments since 3.0.

RETURNS None

FUNCTION

em.Simulation.**all_geos**()

Returns a list of all geometries

RETURNS list[GeoObject]

FUNCTION

em.Simulation.**all_boundaries**()

Returns a selection object of all boundaries

RETURNS FaceSelection

FUNCTION

em.Simulation.**find_geo**()

Search for a geometry by name assigned. Searches for an exact match first and then a partial match defined as any geometry name that contains the provided name. First result is returned.

PARAMETERS
- **name**: str

RETURNS GeoObject

FUNCTION

em.Simulation.**quick_mesh**()

Generates a quick mesh that does not require simulation frequency details.

RETURNS None

FUNCTION

em.Simulation.**generate_mesh**()

Tells EMerge to start generating the 3D mesh. Will not regenerate by default if a (non quick_mesh) mesh already exists. Also assignes the default boundary conditions when done.

PARAMETERS
- **regenerate**: bool = False - If the mesh should be regenerated.
- **remove_orphans**: bool = False - Remove orphan surfaces (no longer the default since 3.0).

RETURNS None

FUNCTION

em.Simulation.**parameter_sweep**()

Used as an iterator to auotmate parameter sweep simulations. Provide a list of keyword arguments with iterables that are to be sweeped over. With multiple variables, EMerge will **always** take the outer product (permutation of all veriable combinations). This iterator will yield a tuple of each variable that can be used in the simulation. **The variables are returned in alphabetical order** because Python dicts do not preserve order by default.

Example usage:

```python
sim = em.Simulation(...)
for width in sim.parameter_sweep(True, width=[1*mm, 2*mm, 3*mm]):
    box = em.geo.Box(width, 1, 2)
    # etc
    sim.mw.run_sweep()
data = sim.data.mw
field = data.field.find(freq=1e9, width=2*mm)
```

PARAMETERS
- **clear_mesh**: bool = True - If the mesh and geomometry should be reset in each iteration. Can be set to False in simulations where you might only change boundary condition settings, not the geometry itself.
- ****parameters** - A sequence of optional keyword arguments that are the variable name and the set of variables.

RETURNS Generator[tuple]

FUNCTION

em.Simulation.**export**()

Export the current Geometry or Mesh. Uses GMSHs built in save system. Supported extensions are:

3D Model:
.opt, .geo_unrolled, .brep, .xao ,.step and .iges

Mesh:
.msh, .inp, .key, .celum, .cgns, .diff, .unv, .ir3, .mes, .mesh, .mail, .m, .bdf, .off, .p3d, .stl, .wrl, .vtk, .dat, .ply2, .su2, .neu, .x3d

PARAMETERS
- **filename**: str - The filename plus extension for the file type.

RETURNS None

FUNCTION

em.Simulation.**set_solver**()

Use a specific solver. Pass a class from em.solvers. Returns the solver object with its own type, so it can be configured further.

```python
model.set_solver(em.solvers.Pardiso).set_nthreads(8)
```

PARAMETERS
- **solver**: type[Solver] | Solver - The solver class (or a solver object)

RETURNS Solver

RAISES RuntimeError if the solver is not installed

FUNCTION

em.Simulation.**get_solver**()

Return the solver object of a solver class, for example model.get_solver(em.solvers.Pardiso).

PARAMETERS
- **solver**: type[_S] - The solver class

RETURNS _S

RAISES RuntimeError

FUNCTION

em.Simulation.**set_device**()

Set the hardware the direct solvers use, for example model.set_device(“GPU”). - “CPU” (default): CPU solvers only. SPARTA, when installed, runs its CPU path, which is the fastest direct solver on Apple silicon. - “GPU”: SPARTA on the Apple GPU, otherwise cuDSS on an NVIDIA GPU (pip install emerge[cudss]). Without either, the CPU solvers are used and a warning is given. - “HYBRID”: SPARTA’s combined GPU + CPU factorization (Apple silicon only). Without it, “GPU” is used. Systems with fewer than 10,000 unknowns, eigenmode problems and run_sweep_distributed(worker_type=“thread”) keep using CPU solvers. A solver set with set_solver() is used as is, except that SPARTA’s mode follows the device.

PARAMETERS
- **device**: “CPU” | “GPU” | “HYBRID” - The device

RETURNS None

RAISES ValueError

FUNCTION

em.Simulation.**adaptive_mesh_refinement**()

Adaptive mesh refinement by local refinement of the existing mesh (AURA). Every pass the model is solved, the a-posteriori error is estimated and the mesh is refined locally (longest-edge bisection of the highest-error tetrahedra, followed by flips and smoothing) until it has (1+growth) times as many tetrahedra. All Mesh3D mappings are rebuilt without GMSH. New nodes on curved faces/curves are projected onto a one-time snapshot of the OCC geometry. On periodic models every split of an edge on a periodic face is mirrored on the paired face, so the periodic faces keep matching meshes.

Further tuning: `order`, `theta`, `growth_limits`, `max_depth`, `lepp_ratio`, `improve_sweeps`, `improve_layers`, `improve_initial`, `improve_global`, `strip_max_growth`, `strip_gated`, `project_boundaries`.

python

```
model.adaptive_mesh_refinement(max_steps=8, growth="auto", show_mesh=False)
data = model.mw.run_sweep()
```

PARAMETERS
- **max_steps**: int = 6 - The maximum number of solve passes.
- **min_refined_passes**: int = 1 - The number of subsequent passes that must meet the convergence criteria.
- **convergence**: float = 0.02 - The absolute convergence limit for any S-parameter |Sij^(k+1)-Sij^(k)|.
- **magnitude_convergence**: float = 2.0 - Convergence limit for only the magnitude |Sij^(k+1)|-|Sij^k|.
- **phase_convergence**: float = 180 - Absolute limit for the phase convergence in degrees.
- **max_tets**: int = 1000000 - The maximum number of tets before aborting.
- **minimum_steps**: int = 1 - The minimum number of refinement steps.
- **frequency**: float | list[float] | None = None - The convergence frequency/frequencies.
- **show_mesh**: bool = False - Show the mesh in between solves.
- **growth**: float | str | list[float] | Callable[[int, dict], float] = 0.3 - Relative increase of tets per pass: float, per-pass list, “auto” or f(step, info).
- **strip_ratio**: float = 0 - Split curve edges along narrow strips (thin traces) until they are at most this many strip widths long, so the strip is not meshed with slivers. 0 disables it.
- **nthreads**: int | None = None - The preferred number of CPU threads of the solver, for solvers that support it. None keeps the solver’s own setting.

RETURNS SimulationDataset

FUNCTION

em.Simulation.**set_scale_factor**()

Define the scaling factor at which modelling occurs relative to the actual size. The physical mesh will be descaled with 1/factor after meshing. For example, if you want to model some object 1000x larger than it really is, the scale factor becomes 1000 This can be used when modelling tiny structures like bond wires which may trip up GMSH and OpenCASCADE

Experimental.

PARAMETERS
- **scale_factor**: float - The modelling size divided by the physical size

RETURNS None

FUNCTION

em.Simulation.**conducting_volumes**()

Returns a selection of all conducting volumes

PARAMETERS
- **conductivity_level**: float = 100000.0 - The conductivity beyond which a domain is considered conducting.

RETURNS DomainSelection

FUNCTION

em.Simulation.**conducting_surfaces**()

Returns a face selection for all faces that are considered electrically conducting

PARAMETERS
- **conductivity_level**: float = 100000.0 - The conductivity beyond which a domain is considered conducting.

RETURNS FaceSelection

FUNCTION

em.Simulation.**set_write_log**()

Adds a file output for the logger.

RETURNS None

FUNCTION

em.Simulation.**clean**()

Cleans up the simulation object references by deleting all major components.

RETURNS None

FUNCTION

em.Simulation.**from_simulations**()

Construct a new simulation object but import the simulation date from multiple different other simulation objects.

PARAMETERS
- **name**: str - The Name of the new simulation object
- ***sims**: Simulation - A sequence of other simulation objects to merge.
- ****kwargs** - All other standard keyword arguments when constructing a `Simulation` object.

RETURNS None

10. GEOMETRY FUNCTIONS

10.1 Primitives

FUNCTION

em.**Axis**()

Creates an Axis object which is a normalized 3D vector with length 1. Globally available realizations are: em.XAX, em.YAX and em.ZAX

PARAMETERS
- **vector**: np.ndarray | Iterable[float, float, float]

RETURNS Axis

FUNCTION

em.Axis.**cross**()

Take the cross product with another axis.

PARAMETERS
- **other**: Axis

RETURNS Axis

FUNCTION

em.Axis.**dot**()

Take the dot product with another axis.

PARAMETERS
- **other**: Axis

RETURNS Axis

FUNCTION

em.Axis.**pair**()

Pair with another Axis object to create a Plane object

PARAMETERS
- **other**: Axis

RETURNS Plane

FUNCTION

em.Axis.**construct_cs**()

Creates a CoordinateSystem whose Z-axis is parallel to the Axis. The origin can be given as three coordinates, as one tuple or with `origin=`.

```python
cs = em.YAX.construct_cs(a/4, 0, L/2)
```

PARAMETERS
- **x**: float | tuple[float, float, float] | ndarray | None = None - The x-coordinate of the origin, or the whole origin as a tuple
- **y**: float | None = None - The y-coordinate
- **z**: float | None = None - The z-coordinate
- **origin**: tuple[float, float, float] | ndarray | None = None - The origin as one tuple (alternative)

RETURNS CoordinateSystem

FUNCTION

em.Axis.**tensor**()

Build a uniaxial material tensor $\varepsilon_\perp \cdot I + (\varepsilon_\parallel - \varepsilon_\perp)\cdot \hat{n}\hat{n}^T$ with this axis as the optical axis, e.g. `er=em.ZAX.tensor(3.4, 3.0)`.

PARAMETERS
- **axis_value**: complex - The value along the axis
- **plane_value**: complex - The value normal to the axis

RETURNS ndarray

PROPERTY

em.Axis.**neg**

The axis pointing in the opposite direction (`-axis` works too).

RETURNS Axis

FUNCTION

em.**Plane**()

Creates a 2D plane spanned by two Axis objects which need not be orthogonal. All permutations of XYPLANE, YZPLANE are defined in the global space (em.XYPLANE etc.).

PARAMETERS
- **uax**: Axis
- **vax**: Axis

RETURNS Plane

FUNCTION

em.Plane.**flip**()

Flips the plane axes

RETURNS Axis

FUNCTION

em.Plane.**cs**()

Create a CoordinateSystem object at origin (x0, y0, z0) such that the two axes span the XY plane

PARAMETERS
- **x0**: float = 0.0
- **y0**: float = 0.0
- **z0**: float = 0.0

RETURNS CoordinateSystem

FUNCTION

em.Plane.**grid**()

Creates the (N,M) shaped rectangular grid point set based on arguments that match `np.linspace`.

PARAMETERS
- **uax**: tuple[float, float, int] - The the start - end values and the number of points along the u-axis
- **vax**: tuple[float, float, int] - the start to end values and the number of points along the v axis.
- **origin**: tuple[float, float, float] - The origin in 3D coordinates
- **indexing**: “xy”, “ij”

RETURNS tuple[np.ndarray, np.ndarray, np.ndarray] - X,Y,Z grid arrays

FUNCTION

em.Plane.**span**()

Similar to `grid` but uses an equal number in of points along the u/v axes and starts in 0,0.

PARAMETERS
- **u**: float - The grid u-value range
- **v**: float - the grid v-value range
- **N**: int - The number of points along each axis
- **origin**: tuple[float, float, float] - the origin for the start of the grid.

RETURNS tuple[np.ndarray, np.ndarray, np.ndarray] - X,Y,Z grid arrays

FUNCTION

em.**CoordinateSystem**()

Creates a CoordinateSystem object from a set of axis objects and an origin.

PARAMETERS
- **xax**: em.Axis
- **yax**: em.Axis
- **zax**: em.Axis
- **origin**: np.ndarray

RETURNS CoordinateSystem

FUNCTION

em.**cs**()

Quickly build a CoordinateSystem from an axes string, e.g. `em.cs('yzx', origin=(0, 0, 1))` maps the local x, y, z axes onto the global y, z and x axes. Prefix an axis with `-` to flip it.

PARAMETERS
- **axes**: str = “xyz” - Three global axis letters for the local x, y and z axes
- **origin**: tuple[float, float, float] = (0.0, 0.0, 0.0) - The origin

RETURNS CoordinateSystem

FUNCTION

em.CoordinateSystem.**flip_z**()

Inverts the Z-axis

RETURNS CoordinateSystem

FUNCTION

em.CoordinateSystem.**displace**()

Displace the coordinate system origin

PARAMETERS
- **dx**: float = 0.0
- **dy**: float = 0.0
- **dz**: float = 0.0

RETURNS CoordinateSystem

FUNCTION

em.CoordinateSystem.**rotate**()

Rotate the coordinate system around an axis

PARAMETERS
- **axis**: tuple | em.Axis - The rotation axis
- **angle**: float - The rotation angle
- **degrees**: bool = True - If the angle is in degrees or not.
- **origin**: bool = False - If the origin itself should be rotated or just the basis vectors.

RETURNS CoordinateSystem

FUNCTION

em.CoordinateSystem.**copy**()

Creates a copy of this coordinate system.

RETURNS CoordinateSystem

FUNCTION

em.CoordinateSystem.**swapxy**()

Swaps the X and Y axes

RETURNS CoordinateSystem

FUNCTION

em.CoordinateSystem.**affine_from_global**()

Returns an affine transformation matrix which would transform coordinate 4-vectors from the global coordinate system into this one.

RETURNS np.ndarray

FUNCTION

em.CoordinateSystem.**affine_to_global**()

Returns an affine transformation matrix which would transform coordinate 4-vectors from this coordinate system into the global coordinate system.

RETURNS np.ndarray

FUNCTION

em.CoordinateSystem.**in_global_cs**()

Convert arrays of x,y and z coordinate values into the global coordinate system.

PARAMETERS
- **x**: float | np.ndarray
- **y**: float | np.ndarray
- **z**: float | np.ndarray

RETURNS tuple[np.ndarray, np.ndarray, n

FUNCTION

em.CoordinateSystem.**in_local_cs**()

Convert arrays of x,y and z coordinate values from the global coordinate system.

PARAMETERS
- **x**: float | np.ndarray
- **y**: float | np.ndarray
- **z**: float | np.ndarray

RETURNS tuple[np.ndarray, np.ndarray, n

FUNCTION

em.CoordinateSystem.**in_global_basis**()

Convert arrays of x,y and z vector components into the global coordinate system basis (does not use the origin).

PARAMETERS
- **x**: float | np.ndarray
- **y**: float | np.ndarray
- **z**: float | np.ndarray

RETURNS tuple[np.ndarray, np.ndarray, n

FUNCTION

em.CoordinateSystem.**in_local_basis**()

Convert arrays of x,y and z vector components from the global coordinate system (does not use the origin).

PARAMETERS
- **x**: float | np.ndarray
- **y**: float | np.ndarray
- **z**: float | np.ndarray

RETURNS tuple[np.ndarray, np.ndarray, n

FUNCTION

em.**Anchor**()

Creates and Anchor object. An anchor is a generalization of a coordinate system but embued with properties. They created for geometries on important positions. Anchors can be used to align certain parts of objects with other parts..

PARAMETERS
- **c0**: np.ndarray - The origin of the Anchor point
- **x**: np.ndarray - The x-axis of the anchor point
- **y**: np.ndarray - The y-axis of the anchor point
- **z**: np.ndarray - THe z-axis of the anchor point.

RETURNS tuple[np.ndarray, np.ndarray, n

FUNCTION

em.**Line**()

Creates a line object that can be used to perform path integrals. The Line is always represented as a segmented list of straight line segments.

PARAMETERS
- **xpts**: np.ndarray - A list of x-coordinates.
- **ypts**: np.ndarray - A list of y-coordinates.
- **zpts**: np.ndarray - A list of z-coordinates.

RETURNS Line

PROPERTY

em.Line.**cmid**

The midpoints of all line segments.

RETURNS tuple[np.ndarray, np.ndarray, np.ndarray]

PROPERTY

em.Line.**cpoint**

The coordinates of the line vertices.

RETURNS tuple[np.ndarray, np.ndarray, np.ndarray]

FUNCTION

em.Line.**from_points**()

A static method construction to create a straight line from a start to an end point.

As line integrals are performed as a discretized integral. Make sure to pick enough points to capture all the detail.

PARAMETERS
- **start**: np.ndarray - A (3,) numpy array for the start point
- **end**: np.ndarray - A (3,) numpy array for the end point
- **Npts**: int - The total number of points with which to segment the line.

RETURNS Line

FUNCTION

em.Line.**from_path**()

Creates a Line object from a series of points with approximate spacing ds.

PARAMETERS
- ***points**: tuple[float, float, float] | ndarray - The points the path runs through
- **ds**: float - The maximum step size between samples

RETURNS Line

FUNCTION

em.Line.**subsample**()

Splits the line segments in `n` equal parts.

PARAMETERS
- **n**: int - The number of subsegments

RETURNS Line

FUNCTION

em.Line.**smooth**()

Perform a laplacian smoothing to the line. While preseriving the arc length.

Each step replaces every interior point with the average of its two neighbors then rescales all segment lengths uniformly so the total arc length is unchanged. For closed loops(where the first and last point coincide), all points are treated as interior.

This function is created by Claude Code.

PARAMETERS
- **nsteps**: int = 1 - The number of smoothing steps.

RETURNS Line

FUNCTION

em.Line.**line_integral**()

Performs a line integral of some arbitrary vector field or scalar field. The function must respect the signatures:

- fn(x,y,z) → (3,N)
- fn(x,y,z) → (N,)

PARAMETERS
- **evalfunc**: Callable - The function to integrate

RETURNS `float | complex`

10.2 3D Geometry

FUNCTION

em.geo.**Box**()

Creates a box volume object. Specify the alignment of the box with the provided position. The options are CORNER (default) for the front-left-bottom node of the box or CENTER for the center of the box.

Faces: `.left`, `.right`, `.top`, `.bottom`, `.front`, `.back` and `.outside(*exclude)`.

PARAMETERS
- **width**: float - The x-size
- **depth**: float - The y-size
- **height**: float - The z-size
- **position**: tuple | Anchor = (0, 0, 0) - The position of the box.
- **alignment**: Alignment = Alignment.CORNER - Which point of the box is placed at the position. Defaults to Alignment.CORNER.
- **cs**: CoordinateSystem = GCS
- **name**: str | None = None

RETURNS `Box`

FUNCTION

em.geo.**Sphere**()

Creates a 3D sphere object with the center of the sphere at the provided position.

PARAMETERS
- **radius**: float
- **position**: tuple[float, float, float],

RETURNS `GeoVolume`

FUNCTION

em.geo.**HalfSphere**()

Creates a hemispherical geometry with the center of the sphere at the provided position. The sphere is cut in the plane defined by the direction argument. Faces: `.outside`, `.disc`.

PARAMETERS
- **radius**: float
- **position**: tuple[float, float, float] | Anchor = (0, 0, 0)
- **direction**: tuple | Axis = (1, 0, 0) - The normal axis of the cut plane
- **name**: str | None = None

RETURNS `HalfSphere`

FUNCTION

em.geo.**Cylinder**()

Generates a Cylinder object in 3D space. The cylinder will always be placed in the origin of the provided CoordinateSystem. The bottom cylinder plane is always placed in the XY-plane. The length of the cylinder is oriented along the Z-axis.

Faces: `.front`, `.back`, `.shell`.

PARAMETERS
- **radius**: float - The radius of the Cylinder
- **height**: float - The height of the Cylinder
- **cs**: CoordinateSystem = GCS - The coordinate system.
- **Nsections**: int | None = None - The number of sections.
- **compensate_radius**: bool = False - Enlarge a polygonal (Nsections) cylinder so its cross-section area matches the circle
- **name**: str | None = None

RETURNS `Cylinder`

FUNCTION

em.geo.**CoaxCylinder**()

Creates a Coaxial 3D Cylinder with the core cutout of the geometry. The cylinder starts in the (0,0,0) of the local coordinate system and is aligned with the Z-axis. Pairs well with `em.coax_rin()` and `em.coax_rout()` to compute the radii.

PARAMETERS
- **rout**: float
- **rin**: float
- **height**: float
- **cs**: em.CoordinateSystem,
- **Nsections**: int | None, the number of polygonal sections instead of a curved perimeter.

RETURNS `GeoVolume`

FUNCTION

em.geo.**Cone**()

Creates a 3D cone

PARAMETERS
- **p0**: tuple[float, float, float] - the start coordinate
- **direction**: tuple[float, float, float] - the direction of the cone.
- **r1**: float - The bottom radius
- **r2**: float - the tip radius

RETURNS `GeoVolume`

FUNCTION

em.geo.**Horn**()

Creates a pyrimidal horn such that the horn aperture is in the local XY plane with the horn extruding in the positive Z-direction.

PARAMETERS
- **start_aperture**: tuple[float, float] - the starting aperture width, height
- **end_aperture**: tuple[float, float] - The ending aperture width, height
- **height**: the height/length of the horn
- **cs**: em.CoordinateSystem - The local coordinate system of the horn.

RETURNS `GeoVolume`

FUNCTION

em.geo.**open_region**()

Convenience function to create an airbox around the current geometry.

PARAMETERS
- **xmargin**: float | tuple[float, float] - either the margin on both sides (X-axis) or -X and +X respectively
- **ymargin**: float | tuple[float, float]
- **zmargin**: float | tuple[float, float]
- **material**: em.Material (default to AIR)

RETURNS `GeoVolume`

FUNCTION

em.geo.**open_pml_region**()

Convenience function to create an airbox around the current geometry plus PML regions. See `pmlbox`

PARAMETERS
- **xmargin**: float | tuple[float, float] - either the margin on both sides (X-axis) or -X and +X respectively
- **ymargin**: float | tuple[float, float]
- **zmargin**: float | tuple[float, float]
- **material**: em.Material (default to AIR)
- thickness, Nlayers, N_mesh_layers, exponent, deltamax, sides, top, bottom, ... - Passed on to `pmlbox`

RETURNS `GeoVolume`

FUNCTION

em.geo.**STEPItems**()

Open a .step file. The geometries are accessible through `.objects`, `.names` and `.dictionary`, or merged with `.as_volume()`, `.as_surface()`, `.as_edge()` and `.as_point()`. `.enclose(margin)` creates a box around the imported geometry.

PARAMETERS
- **name**: str - Name used in EMerge for the geometries
- **filename**: str - The filename of the .step file
- **unit**: float = 1.0 - The unit of the step file, e.g. 0.001 for mm
- **string_parser**: Callable = None - Optional function that sanitizes the object names (str → str)

RETURNS STEPItems

FUNCTION

em.geo.**pmlbox**()

Creates an air box plus the desired PML region boxes. The sides that should get a PML layer can be defined either by the sides argument or the top/bottom/.../right arguments.

PARAMETERS
- **width**: float
- **depth**: float
- **height**: float
- **position**: tuple[float, float, float]
- **alignment**: em.CENTER | em.CORNER
- **material**: em.MATERIAL (default AIR)
- **thickness**: float - the thickness of the PML layers
- **Nlayers**: int (default 1) - The number of PML box layers. Creates complex geometries but can improve performance.
- **N_mesh_layers**: Computes the local mesh size to realize N layers in the extrusion direction. This also influences the size in all directions which can explode the mesh. Increase the thickness to prevent a mesh explosion.
- **exponent**: float = 0.5 - The PML material scaling exponent
- **deltamax**: float = 2.0 - A reflection tuning parameters
- **sides**: str = ‘’. An list of which sides to add a PML layer to using the convention ([T]op, [B]ottom, [L]eft, [R]ight, [F]ront, b[A]ck)
- top, bottom, left, right, front back: bool = False - which face to add.

RETURNS list[GeoVolume]

FUNCTION

em.geo.**XYPolygon**()

Define a polygon in a non-embedded separate XY coordinate system. Can later be embedded in 3D space by using the .geo or .extrude arguments. See Section 3.6.

PARAMETERS
- **xs**: Iterable[float],
- **ys**: Iterable[float],
- **cs**: em.CoordinateSystem - Can optinally provide the coordinate system here already.
- **resolution**: float - The discretization resolution for curved boundaries.

RETURNS XYPolygon

FUNCTION

em.geo.**XYPlate**()

Creates a rectangular plate aligned along the X and Y axis of a local coordinate system.

PARAMETERS
- **width**: float
- **height**: float
- **position**: tuple[float, float, float]
- **alignment**: em.CENTER | em.CORNER

RETURNS GeoSurface

FUNCTION

em.geo.**Plate**()

Creates a parallelogram plate in an arbitrary 3D orientation from an origin and two spanning vectors u and v.

PARAMETERS
- **origin**: tuple[float, float, float] | Anchor - The corner point
- **u**: tuple[float, float, float] - The first spanning vector
- **v**: tuple[float, float, float] - The second spanning vector
- **name**: str | None = None - Optional name

RETURNS Plate

FUNCTION

em.geo.**Disc**()

Creates a 2D round or elliptical disc.

PARAMETERS
- **origin**: tuple[float, float, float] | Anchor - The center
- **radius**: float - The (major) radius
- **axis**: tuple[float, float, float] | Axis = (0.0, 0.0, 1.0) - The disc normal axis
- **radius_opt**: float | None = None - The minor ellipse radius (radius becomes the major one)
- **axis_opt**: tuple[float, float, float] | None = None - The direction of the major axis
- **name**: str | None = None - Optional name

RETURNS Disc

10.3 Curves

FUNCTION

em.geo.**Curve**()

Generate a Spline/Bspline or Bezier curve based on a series of points

PARAMETERS
- **xpts**: ndarray - x-coordinates of the control/through points
- **ypts**: ndarray - y-coordinates
- **zpts**: ndarray - z-coordinates
- **degree**: int = 3 - Spline degree
- **weights**: list[float] | None = None - Optional NURBS weights
- **knots**: list[float] | None = None - Optional knot vector
- **multiplicities**: list[float] | None = None - Optional knot multiplicities
- **ctype**: “Spline” | “BSpline” | “Bezier” = “Spline” - The curve type
- **name**: str | None = None - Optional name

RETURNS Curve

FUNCTION

em.geo.Curve.**helix_rh**()

Generates a Helical curve

`Curve.helix_lh(...)` creates the left-handed version with the same arguments.

PARAMETERS
- **pstart**: tuple[float, float, float] | Anchor - The start of the center of rotation (not the start of the curve)
- **pend**: tuple[float, float, float] | Anchor - The end of the center of rotation
- **r_start**: float - The (start) radius of the helix
- **pitch**: float - The pitch angle of the helix
- **r_end**: float | None = None - The ending radius. If default, the same is used as the start.
- **startfeed**: float = 0.0 - The length of an extra feed section added at the start of the helix. Defaults to 0.0 (none).

RETURNS Curve

FUNCTION

em.geo.Curve.**pipe**()

Extrudes a surface or XYPolygon object along the given curve

PARAMETERS
- **crossection**: GeoSurface | XYPolygon - The cross section definition to be used
- **max_mesh_size**: float | None = None - The maximum mesh size.
- **start_tangent**: Axis | tuple[float, float, float] | ndarray | None = None - The input polygon plane normal direction.
- **x_axis**: Axis | tuple[float, float, float] | ndarray | None = None - The reference X-axis to align the input polygon.
- **trihedron**: “DiscreteTrihedron” | “CorrectedFrenet” | “Fixed” | “Frenet” | “ConstantNormal” | “Darboux” | “GuideAC” | “GuidePlan” | “GuideACWithContact” | “GuidePlanWithContact” = “GuidePlan” - The GMSH/OCC trihedron mode used for the sweep.
- **name**: str = “PipedVolume” - The name of the object.

RETURNS GeoVolume

FUNCTION

em.geo.**Point**()

A single geometry point.

PARAMETERS
- **x**: float - x-coordinate
- **y**: float - y-coordinate
- **z**: float - z-coordinate

RETURNS Point

10.4 XYPolygon

FUNCTION

XYPolygon.**incs**()

Provide a coordinate system for the XYPolgyon object then return it.

PARAMETERS
- **cs**: em.CoordinateSystem

RETURNS XYPolygon

FUNCTION

XYPolygon.**extend**()

Extend the list of x and y coordinates with a new list of xy coordinates and return the object.

PARAMETERS
- **xpts**: list[float]
- **ypts**: list[float]

RETURNS XYPolygon

FUNCTION

XYPolygon.**iterate**()

Iterate over all the XY points as a tuple of coordinates.

RETURNS Generator[tuple[float, float]]

FUNCTION

XYPolygon.**fillet**()

Create a fillet (rounded corner) with a given radius for the points at the provided indices. Example:

```
xypolygon.fillet(1*mm, 0,1,5,6,10)
```

PARAMETERS
- **radius**: float
- **indices**: int, a sequence of indices.

RETURNS XYPolygon

FUNCTION

XYPolygon.**extrude**()

Extrude the XYPolygon a certain distance in the local coordinate system positive Z-direction.

PARAMETERS
- **length**: float - the Extrusion distance
- **cs**: em.CoordinateSystem | None - An optional specification of the coordinate system. Defaults to the previously defined or otherwise the global coordinate system.

RETURNS GeoPrism

FUNCTION

XYPolygon.**geo**()

Embed the current polygon in 3D space by creating a polygon GeoSurface

PARAMETERS
- **cs**: CoordinateSystem - An optional specification of the coordinate system if it wasn't already provided. Defaults to the global coordinate system GCS

RETURNS GeoSurface

FUNCTION

XYPolygon.**revolve**()

First embed the XYPolygon as a surface in 3D space and then revolve it around a revolution axis.

PARAMETERS
- **cs**: CoordinateSystem - The coordinate system to embed the polygon in
- **origin**: tuple[float, float, float] | Anchor - A point on the revolution axis (global coordinates)
- **axis**: tuple[float, float, float] | Axis - The revolution axis
- **angle**: float = 360.0 - The revolution angle in degrees
- **name**: str = "Revolution" - Optional name

RETURNS GeoPrism

FUNCTION

XYPolygon.**circle**()

Static method to automatically generate an XYPolygon object for a circle approximated by N sections. The number of sections can be controlled in three ways, either by providing a section edge size, tolerance form the curve or the number of sections.

PARAMETERS
- **radius**: float
- **dsmax**: the maximum section edge length - optional
- **tolerance**: the maximum error tolerance for the N-sectional polygon to deviate from the ideal curve.
- **Nsections**: The number of sections.

RETURNS XYPolygon

FUNCTION

XYPolygon.**rect**()

Static method to create a simple rectangle. `origin` may be an `(x, y)` tuple.

PARAMETERS
- **width**: float - Width
- **height**: float - Height
- **origin**: tuple[float, float] | Anchor - The origin
- **alignment**: Alignment = Alignment.CORNER - em.CORNER or em.CENTER

RETURNS XYPolygon

FUNCTION

XYPolygon.**corrugated_line**()

Add a corrugating line to the polygon path

PARAMETERS
- **direction**: tuple[float, float, float] | Axis - The direction of the line
- **depth**: float - The corrugation depth
- **total_length**: float - The total length
- **period**: float - The corrugation period
- **corr_axis**: tuple[float, float] | Axis | None = None - The axis along which the corrugations protrude
- **side**: "left" | "right" | None = None - The side of the line the corrugations are on
- **duty_cycle**: float = 0.5 - Fraction of the period that is raised
- **offset**: float = 0.0 - Offset of the corrugation pattern along the line

RETURNS XYPolygon

FUNCTION

XYPolygon.**parametric**()

Appends the polygon with a new curve section defined by a parametric function set

$$x(t), y(t) \forall t \in [\mathrm{tmin}, \mathrm{tmax}]$$

The detail of the appoximation is controlled by xmin and tolerance.

PARAMETERS
- **xfunc**: Callable - A parametric function that returns an x-coordinate as a function of float t.
- **yfunc**: Callable - Same as for xfunc
- **xmin**: float = 0.001 - The minimum segment size
- **tolerance**: float = 1e-5 - The amount the segmented line section can deviate from the ideal curve.
- **tmin**: float = 0.0 - The start value for the parametric line.
- **tmax**: float = 1.0 - The end value for the parametric line.

reverse: bool = False - If the curve should be traversed in reverse (tmax to tmin).

RETURNS XYPolygon

FUNCTION

XYPolygon.**connect**()

Connect two XYPolygons with a defined coordinate system creating a pipe between them

PARAMETERS
- **other**: XYPolygon - The XYPolygon object to connect this object to

RETURNS GeoVolume

FUNCTION

XYPolygon.**shrink**()

Shrinks the XY polygon by making each line segment move some distance inwards. This function does not safely take into account the posibility of segments disappearing.

PARAMETERS
- **distance**: float - The distance to move the polygon inwards.

RETURNS XYPolygon

10.5 Operations

FUNCTION

em.geo.**add**()

Perform a boolean union between two GeoObjects. Must be of the same dimension.

PARAMETERS
- **main**: GeoObject
- **other**: GeoObject
- **remove_object**: bool = True - If the main object should be removed after union.
- **remove_tool**: bool = True - If the tool object should be removed after union.

RETURNS GeoObject

FUNCTION

em.geo.**remove/subtract**()

Function goes by both names `remove` and `subtract`.

Perform a boolean difference between two GeoObjects. Subtracts the tool from the main object. Must be of the same dimension.

PARAMETERS
- **main**: GeoObject
- **other**: GeoObject
- **remove_object**: bool = True - If the main object should be removed after difference.
- **remove_tool**: bool = True - If the tool object should be removed after difference.

RETURNS GeoObject

FUNCTION

em.geo.**intersect**()

Perform a boolean intersection between two GeoObjects. Must be of the same dimension.

PARAMETERS
- **main**: GeoObject
- **other**: GeoObject
- **remove_object**: bool = True - If the main object should be removed after intersection.
- **remove_tool**: bool = True - If the tool object should be removed after intersection.

RETURNS GeoObject

FUNCTION

em.geo.**embed**()

Embeds a 2D surface into a 3D volume such that the surface will be meshed without itself being a separate object.

PARAMETERS
- **main**: GeoVolume
- **other**: GeoSurface

RETURNS None

FUNCTION

em.geo.**rotate**()

Rotates the input geometry.

PARAMETERS
- **main**: GeoObject
- **c0**: tuple[float, float, float] - the rotation axis origin
- **ax**: tuple[float, float float] - The rotation axis
- **angle**: float - The rotation angle in degrees or radians
- **make_copy**: bool = False - If rotated object should be of a copy of the original or the original.
- **degree**: bool = True - If the rotation angle is given in degrees.

RETURNS GeoObject

FUNCTION

em.geo.**translate**()

Translates the input geometry

PARAMETERS
- **main**: GeoObject
- **dx**: float = 0.0
- **dy**: float = 0.0
- **dz**: float = 0.0
- **make_copy**: bool = False - If rotated object should be of a copy of the original or the original.

RETURNS GeoObject

FUNCTION

em.geo.**mirror**()

Mirrors the input geometry

PARAMETERS
- **main**: GeoObject
- **origin**: tuple[float, float, float] = (0,0,0) the mirror plane origin.
- **direction**: tuple[float, float, float] = (0,0,1) the mirror plane normal axis.
- **make_copy**: bool = True - If rotated object should be of a copy of the original or the original.

RETURNS GeoObject

FUNCTION

em.geo.**change_coordinate_system**()

Transforms the input geometry from some coordinate system into another. The default starting and ending coordinate system is the global coordinate system.

PARAMETERS
- **main**: GeoObject
- **new_cs**: em.Coordinatesystem = GCS - The coordinate system that the object is transformed into.
- **old_cs**: em.CoordinateSystem = GCS - The coordinate system from which the object is transformed.

RETURNS GeoObject

FUNCTION

em.geo.**stretch**()

Stretches a geometry by some scalar ratio in the x, y and z axis relative to some origin

PARAMETERS
- **main**: GeoObject
- **fx**: float = 1.0
- **fy**: float = 1.0
- **fz**: float = 1.0
- **origin**: tuple[float, float, float] = (0,0,0)

RETURNS GeoObject

FUNCTION

em.geo.**extrude**()

Extrudes an input GeoSurface object along some axis specified by dx, dy and dz

PARAMETERS
- **main**: GeoSurface
- **dx**: float = 0.0
- **dy**: float = 0.0
- **dz**: float = 0.0

RETURNS GeoVolume

FUNCTION

em.geo.**unite**()

Unite a sequence of GeoObjects of the same dimension into one object (similar to recursive unions). Example:

```
out = em.geo.unite(obj1, obj2, ..., objn)
```

PARAMETERS
- ***objects**: GeoObject -

RETURNS GeoObject

FUNCTION

em.geo.**stick**()

Move some object such that the objects anchor point is placed at some other anchor point. See `Anchor`. Example usage:

```
em.geo.stick(box1, box1.anch.Y, box2.anch.x)
```

See the manual for more details.

PARAMETERS
- **obj**: GeoObject
- **p1**: em.Anchor
- **p2**: em.Anchor

RETURNS GeoObject

FUNCTION

em.geo.**duplicate**()

Create a copy of a geometry or selection.

PARAMETERS
- **entity**: GeoObject | Selection - The geometry or selection to copy

RETURNS GeoObject

FUNCTION

em.geo.**connect_faces**()

Connect two surface objects together

PARAMETERS
- **face1**: GeoSurface - The first surface to connect
- **face2**: GeoSurface - The second surface to connect
- **make_solid**: bool = True - Create a solid instead of only the side surfaces.
- **make_ruled**: bool = True - Create ruled (straight) side surfaces.
- **parametrization**: “ChordLength” | “Centripetal” | “IsoParametric” = “IsoParametric” - The OCC section parametrization.

RETURNS GeoVolume

FUNCTION

em.geo.**thick_wall**()

Add thickness to volumes and return a solid expanded geometry without internal holes.

PARAMETERS
- **thickness**: float - The wall thickness
- ***volumes** - The volumes to put a wall around

RETURNS GeoVolume

FUNCTION

em.geo.**expand_surface**()

EXPERIMENTAL: Expands an input surface. The surface must exist on a 2D plane.

PARAMETERS
- **surface**: GeoSurface - The input surface to expand
- **distance**: float - The exapansion distance

RETURNS GeoSurface

FUNCTION

em.geo.**bounding_box**()

Returns the dimensions of a bounding box around all simulation domain objects.

RETURNS tuple[float,...] - xmin, xmax, ymin, ymax, zmin, zmax

10.6 GeoObject methods

FUNCTION

GeoObject.**set_material**()

Assign a material. `GeoObject.material` is read-only since 3.0; use this method or `obj.properties += material`. Returns the object so it can be chained: `em.geo.Box(...).set_material(em.lib.DIEL_FR4)`.

PARAMETERS
- **material**: Material - The material

RETURNS T

FUNCTION

GeoObject.**give_name**()

Assign a name to this object

PARAMETERS
- **name**: str | None = None - The new name

RETURNS GeoObject

FUNCTION

GeoObject.**prio_set**()

Set the material priority level. Where geometries overlap, the material of the object with the highest priority is used. Related: `.prio_up()`, `.prio_down()`, `.above(other)`, `.below(other)`, `.background()` (lowest) and `.foreground()` (highest).

PARAMETERS
- **level**: int - The priority level

RETURNS T

FUNCTION

GeoObject.**above**()

Puts the priority of this object one higher than the other, then returns this object

PARAMETERS
- **other**: GeoObject - The object to place this one above

RETURNS T

FUNCTION

GeoObject.**background**()

Set the material selection priority to be on the background.

RETURNS T

FUNCTION

GeoObject.**foreground**()

Set the material selection priority to be on top.

RETURNS T

FUNCTION

GeoObject.**make_copy**()

Copies this object and returns a new object (also in GMSH)

RETURNS T

FUNCTION

GeoObject.**hide**()

Hides the object from views

`.unhide()` reverses it.

RETURNS T

FUNCTION

GeoObject.**remove**()

Delete the geometry from the simulation model

RETURNS None

PROPERTY

GeoObject.**selection**

The selection of all entities of the object: a `DomainSelection` for volumes, a `FaceSelection` for surfaces, etc.

RETURNS Selection

FUNCTION

GeoVolume.**face**()

Returns the FaceSelection for a given face name. The face name must be defined for the type of geometry.

PARAMETERS
- **name**: FaceNames = None - The name of the face to select.
- **tool**: GeoObject | None = None - Which object should be used as a source for the face selection.
- **no**: FaceNames = None - If everything BUT a face name should be selected, Equivalent to .boundary(exclude=name).

RETURNS FaceSelection

FUNCTION

GeoVolume.**faces**()

Returns the FaceSelection for a given face names. The face name must be defined for the type of geometry.

PARAMETERS
- ***names**: FaceNames - The names of the faces to select.
- **tool**: GeoObject | None = None - The tool object to use as source of the selection.

RETURNS FaceSelection

FUNCTION

GeoVolume.**boundary**()

Returns the complete set of boundary faces. If implemented, it is possible to exclude a set of faces based on their name or a list of tags.

PARAMETERS
- **exclude**: Iterable[FaceNames, ...] | str | None = None - (Iterable[str], str, None): A single string or list/tuple of strings.
- **tags**: list[int] | None = None - A list of face integers (if known)
- **tool**: GeoObject | None = None - The tool object to base the selection face names one.

RETURNS FaceSelection

FUNCTION

GeoVolume.**all_faces**()

Returns a list of all face selections of this object

PARAMETERS
- **tool**: GeoObject | None = None - Return the faces originating from this tool object instead.

RETURNS list[FaceSelection]

FUNCTION

GeoVolume.**face_from_anchor**()

Returns the boundary faces that lie in the plane of an anchor. The anchor origin is a point on the face and its Z-axis the face normal (either sign). Like .face(), all boundary faces of this object in that plane are selected.

PARAMETERS
- **anchor**: Anchor - The anchor to use as face pointer

RETURNS FaceSelection

FUNCTION

GeoVolume.**exterior_faces**()

Select the exterior faces of an object based on the face pointers of an original base object. For example

PARAMETERS
- **base_object**: GeoObject - The object to base the selection on.

RETURNS FaceSelection

FUNCTION

GeoSurface.**assign_thickness**()

Assign a virtual thickness to this surface object. The thickness will be used by the EM assembler to assign the ThinCondutor boundary condition.

PARAMETERS
- **thickness**: float - The thickness in meters

RETURNS GeoSurface

10.7 Selections

FUNCTION

em.**select**()

Generate a selection from a series of selections and/or objects that share the same dimension.

PARAMETERS
- ***items**: Selection | GeoObject - The selections and objects to combine

RETURNS Selection

PROPERTY

em.Simulation.**select**

The selector of a Simulation. Choose the entity type with `.node`, `.edge`, `.face` or `.domain` and then search with `.near(x, y, z)`, `.inplane(x, y, z, normal_axis)` or `.inlayer(x, y, z, vector)`.

```python
port = model.select.face.near(0, 0, 0)
top = model.select.face.inplane(0, 0, h, em.ZAX)
```

RETURNS Selector

FUNCTION

Selector.**near**()

Returns a selection of the releative dimeions by which of the instances is most proximate to a coordinate.

PARAMETERS
- **x**: float - The X-coordinate
- **y**: float - The Y-coordinate
- **z**: float = 0 - The Z-coordinate.

RETURNS Selection | PointSelection | EdgeSelection | FaceSelection | DomainSelection

FUNCTION

Selector.**inplane**()

Returns a FaceSelection for all faces that lie in a provided infinite plane specified by an origin plus a plane normal vector.

PARAMETERS
- **x**: float - The plane origin X-coordinate
- **y**: float - The plane origin Y-coordinate
- **z**: float - The plane origin Z-coordinate
- **normal_axis**: Axis | tuple[float, float, float] | None = None - The plane normal vector
- **plane**: Plane | None = None - A plane whose normal is used instead of normal_axis.
- **tolerance**: float = 1e-08 - An in plane tolerance (displacement and normal dot product).

RETURNS FaceSelection

FUNCTION

Selector.**inlayer**()

Returns a list of selections that are in the layer defined by the plane normal vector and the point (x,y,z)

PARAMETERS
- **x**: float - The X-coordinate from which to select
- **y**: float - The Y-coordinate from which to select
- **z**: float - The Z-coordinate from which to select
- **vector**: tuple[float, float, float] | ndarray | Axis - A vector with length in (meters) originating at the origin.

RETURNS Selection

FUNCTION

Selection.**constrain**()

Constrain this selection using a function xyz_function(x,y,z) -> bool. The function is evaluated at the minimum and maximum corners of the bounding box of each item in the selection.

PARAMETERS
- **xyz_function**: Callable = lambda x, y, z: True - A callable for (x,y,z) that returns True if the point satisfies the constraint.
- **complete**: bool = True - If True, both corners must satisfy the constraint (entirely inside). If False, one corner suffices (partially inside).

RETURNS TSelection

FUNCTION

Selection.**below**()

Constrain this selection to items below some Z-level (-Z side).

Likewise `.above(z)`, `.left_of(x)`, `.right_of(x)`, `.front_of(y)` and `.behind(y)`.

PARAMETERS
- **z_level**: float - The Z-level
- **complete**: bool = True - If it should be entirely below.

RETURNS TSelection

FUNCTION

Selection.**around**()

Return all objects with bounding boxes within some distance of a point.

PARAMETERS
- **distance**: float - The maximum distance
- **coordinate**: tuple[float, float, float] = (0, 0, 0) - The coordinate.

RETURNS TSelection

FUNCTION

Selection.**exclude**()

Exclude points by evaluating a function(x,y,z)-> bool

PARAMETERS
- **xyz_excl_function**: Callable = lambda x, y, z: True - A callable for (x,y,z) that returns True if the point should be excluded.
- **axis**: Axis | None = None - Also exclude faces whose normal is (anti)parallel to this axis.

RETURNS Selection

FUNCTION

Selection.**isolate**()

Include points by evaluating a function(x,y,z)-> bool

PARAMETERS
- **xyz_excl_function**: Callable = lambda x, y, z: True - A callable for (x,y,z) that returns True if the point should be kept.
- **plane**: Plane | None = None - Currently unused.
- **axis**: Axis | None = None - Only keep faces whose normal points along this axis.

RETURNS Selection

PROPERTY

Selection.**centers / getCAs**

`.center` / `.centers` give the centre of the selection or of every tag; `.getCAs` gives `(centre, area)` per tag; `.bounding_box`, `.points` and `.tags` give the extent, points and GMSH tags.

RETURNS list

PROPERTY

FaceSelection.**normal / area**

The normal vector and the total area of a FaceSelection.

RETURNS ndarray / float

FUNCTION

FaceSelection.**sample**()

Sample the surface at the compiler defined parametric coordinate range. This function usually returns a square region that contains the surface.

PARAMETERS
- **Npts**: int - The number of samples per parametric direction.

RETURNS tuple[ndarray, ndarray, ndarray] | list[tuple[ndarray, ndarray, ndarray]]

FUNCTION

FaceSelection.**rect_basis**()

Returns a dictionary with keys: origin, axes, corners. The axes are the 3D basis vectors of the rectangle. The corners are the 4 corners of the rectangle.

RETURNS tuple[CoordinateSystem, tuple[float, float]]

10.8 PCB

FUNCTION

em.geo.**PCBLayer**()

Creates a PCBLayer class object specifying a dielectric layer of a PCB layer stack.

PARAMETERS
- **thickness**: float - The layer thickness
- **material**: Material - The dielectric layer material
- **name**: str | None = None - The layer name

RETURNS PCBLayer

FUNCTION

em.geo.**PCB**()

Creates a PCB layout object. Renamed from `PCBNew` in 3.0 (`PCBNew` is kept as an alias). Copper layers are addressed by **layer index**, counted from 0 at the bottom; negative indices count from the top (`-1` is the top layer).

```python
pcb = em.geo.PCB(0.508, unit=mm, material=em.lib.DIEL_RO4350B, layers=2)
w = pcb.calc.z0(50)            # 50 Ω trace width on the top layer
pcb.new(0, 0, w, (1, 0)).straight(20)
pcb.modal_port(pcb.load(...), 1, height=3)  # creates ModalPort 1
```

PARAMETERS
- **thickness**: float - The total PCB thickness
- **unit**: float = 0.001 - The units used for all dimensions. Defaults to 0.001 (mm).
- **cs**: CoordinateSystem | None = None - The coordinate system to place the PCB in (XY).
- **material**: Material = AIR - The dielectric material.
- **trace_material**: Material = PEC - The trace material.
- **layers**: int = 2 - The number of copper layers.
- **stack**: list[PCBLayer] = None - Optional list of PCBLayer classes for multilayer PCB with different dielectrics.
- **name**: str | None = None - The PCB object name.
- **trace_thickness**: float = 3.5e-05 - The conductor trace thickness if important.
- **zs**: ndarray | None = None - The z-coordinates of all copper layers; overrides thickness, layers and stack.
- **thick_traces**: bool = False - If traces should be given a thickness and modeled in 3D.

RETURNS PCB

FUNCTION

em.geo.PCB.**z**()

Returns the Z-height of the copper trace layer with the given index.

PARAMETERS
- **layer**: int - The layer index (negative counts from the top)

RETURNS float

FUNCTION

em.geo.PCB.**add_vias**()

Add vias at the specified coordinates. All arguments beyond the coordinates must be provided as keyword arguments. Example:

```
pcb.add_vias((0,0),(0,1),(2,0),...,radius=0.2, layer1=0, layer2=2)
```

PARAMETERS
- ***coordinates**: tuple[float, float] | Anchor - (x, y) tuples of the via positions
- **radius**: float - The via radius
- **layer1**: int = 0 - The start layer index
- **layer2**: int = −1 - The end layer index
- **segments**: int = 6 - The number of polygon segments of the via cross-section

RETURNS None

FUNCTION

em.geo.PCB.**load**()

Returns a RouteElement object for the given label name.

PARAMETERS
- **name**: str - The name of the x,y coordinate

RETURNS RouteElement

FUNCTION

em.geo.PCB.**determine_bounds**()

Establish the XY bounds of all the geometries in the PCB. Must be called after compile_paths.

PARAMETERS
- **leftmargin**: float = 0 - The left margin.
- **topmargin**: float = 0 - The top margin.
- **rightmargin**: float = 0 - The right margin.
- **bottommargin**: float = 0 - The bottom margin.

RETURNS None

FUNCTION

em.geo.PCB.**set_bounds**()

Hard set the boundaries of the PCB

PARAMETERS
- **xmin**: float - The minimum x-coordinate
- **ymin**: float - The minimum y-coordinate
- **xmax**: float - The maximum x-coordinate
- **ymax**: float - The maximum y-coordinate

RETURNS None

FUNCTION

em.geo.PCB.**plane**()

Creates a GeoSurface or GeoVolume plane object based on coordinate specifications relative to the PCB. If no dimensions and coordinates are provided, the entire bounds of the PCB will be used. Holes specified by vias will automatically be cut from these planes.

Can be used to generate a solid ground plane:

```
gnd = pcb.plane(0)
```

PARAMETERS
- **layer**: int - The layer index
- **width**: float | None = None - The width of the plate.
- **height**: float | None = None - The height of the plate.
- **origin**: tuple[float, float] | Anchor | None = None - The origin of the plate.
- **alignment**: Alignment = Alignment.CORNER - The alignment of the plate.
- **material**: Material | None = None - The optional plane material.
- **name**: str | None = “Plane” - The name of the object.

RETURNS GeoSurface | GeoVolume

FUNCTION

em.geo.PCB.**radial_stub**()

Add a radial stub geometry to the polygon list which will be generated.

PARAMETERS
- **pos**: tuple[float, float] | Anchor - The position of the stub origin
- **length**: float - The length of the stub
- **angle**: float - The angle of the stub in degrees
- **direction**: tuple[float, float] - The direction vector
- **Nsections**: int = 8 - Number of angle sections.
- **w0**: float = 0 - the start width.
- **layer**: int = −1 - the layer index.
- **material**: Material = None - the stub material.
- **name**: str = None - The geometry name.

RETURNS None

FUNCTION

em.geo.PCB.**generate_pcb**()

Generates either a list of GeoVolume objects or a single GeoVolume object for each dieletric layer.

PARAMETERS
- **split_z**: bool = True - If a PCB consisting of a thickness, material and n_layers should be split in sub domains.
- **merge**: bool = True - (bool, optional): If an output list of multiple volumes should be merged into a single object.

RETURNS list[GeoVolume] | GeoVolume

FUNCTION

em.geo.PCB.**generate_air**()

Creates an Air box exactly matching the PCB area above the PCB.

PARAMETERS
- **height**: float - The height of the air box
- **name**: str = “PCBAirbox” - The name of the object.
- **bottom**: bool = False - Place the air box below the PCB instead of on top.

RETURNS GeoVolume

FUNCTION

em.geo.PCB.**new**()

Start a new StripPath sequence for routing PCB traces. See the StripPath instructions in Section 3.10. Use `trace_layer` to route on another layer than the top.

PARAMETERS
- **x**: float | Anchor - The starting X-coordinate (local)
- **y**: float - The starting Y-coordinate (local)
- **width**: float - The (micro)-stripline width
- **direction**: tuple[float, float] - The direction.
- **trace_layer**: int = −1 - The trace layer.
- **name**: str | None = None - The name of the object.

RETURNS StripPath

FUNCTION

em.geo.PCB.**lumped_port**()

Create a lumped port surface at a stored route point **and** the LumpedPort boundary condition with the given port number. Retrieve the boundary condition with `model.mw.bc.get_port(n)`.

PARAMETERS
- **stripline**: StripLine | str - The stripline object or a reference to the stripline object
- **port_number**: int - The port number assigned to the lumped port
- **ground_layer**: int = 0 - The PCB layer the port is extruded down to (the ground).
- **impedance**: float = 50.0 - The port impedance in Ohm. Defaults to 50.0.
- **name**: str | None = “LumpedPort” - The name for the resultant object.

RETURNS GeoPolygon

FUNCTION

em.geo.PCB.**lumped_port_pts**()

Add the lumped port rectangle based on two coordinates that define a line segment which is extruded between two PCB layers. The LumpedPort boundary condition is created automatically.

PARAMETERS
- **p1**: tuple[float, float] - Point one of the line to be extruded (PCB units)
- **p2**: tuple[float, float] - Point two of the line to be extruded (PCB units)
- **layer**: int - The PCB layer the port starts at
- **port_number**: int - The port number assigned to the lumped port
- **ground_layer**: int = 0 - The PCB layer the port is extruded down to (the ground).
- **impedance**: float = 50.0 - The port impedance in Ohm. Defaults to 50.0.
- **name**: str | None = “LumpedPort” - The name for the resultant object.

RETURNS GeoPolygon

FUNCTION

em.geo.PCB.**modal_port**()

Create a modal port surface at a stored route point **and** the ModalPort boundary condition with the given port number (since 3.0 no separate `model.mw.bc.ModalPort(...)` call is needed). The trace is the positive terminal and the closest other conductor the negative one.

PARAMETERS
- **point**: StripLine | str - The location of the port.
- **port_number**: int - The port number
- **height**: float | tuple[float, float] - The height of the port above the PCB. A (dz, height) tuple additionally extends the port by dz below the PCB.
- **width_multiplier**: float = 5.0 - The width of the port in stripline widths. Defaults to 5.0.
- **width**: float | None = None - The absolute port width, overrides width_multiplier.
- **mode_type**: “TEM” | “TE” | “TM” | “any” = “any” - The mode type of the port.
- **name**: str | None = “ModalPort” - The name of the object.
- **assume_homogeneous**: bool = False - Treat the port face (substrate and air) as homogeneous: the modes are solved once and scaled to the other frequencies with a constant effective permittivity, see ModalPort.assume_homogeneous(). This ignores the dispersion of the line, which is small at a low frequency·substrate thickness.
- **target_neff**: float | None = None - The effective index β/k0 of the wanted mode, e.g. about sqrt(εr) for a microstrip mode. The first mode search looks around it and the mode closest to it is the port mode, see ModalPort. None searches around the mean of the largest and smallest εr of the port face.

RETURNS GeoSurface

FUNCTION

em.geo.PCB.**generate_vias**()

Generates all the queued vias as a list of GeoVolume objects or single GeoVolume object..

PARAMETERS
- merge = False - Whether to merge the result into a final object.
- **material**: Material | None = None - The optional material for the vias.

RETURNS list[Cylinder] | GeoVolume

FUNCTION

em.geo.PCB.**add_poly**()

Add an arbitrary polygon to the queue of objects to be created.

PARAMETERS
- **xs**: list[float] - A list of x-coordinates
- **ys**: list[float] - A list of y-coordinates
- **layer**: int = 0 - The layer index.
- **material**: Material = None - The material.
- **name**: str | None = None - The name of the object.

RETURNS None

FUNCTION

em.geo.PCB.**add_hole**()

Add a hole to be cut form all polygon objects.

PARAMETERS
- **xs**: list[float] - A list of x-coordinates
- **ys**: list[float] - A list of y-coordinates
- **layer**: int = 0 - The layer index.
- **name**: str | None = None - The name of the object.

RETURNS None

FUNCTION

em.geo.PCB.**compile_paths**()

Compile paths will turn all queued strip paths and polygons into a list of GeoPolygon objets or GeoVolume (if thick traces is turned on).

PARAMETERS
- **merge**: bool = False - Whether to merge the Polygons into a single.
- **fragment**: bool = False - Adds line segment points to improve mesh quality in combination with set_trace_refinement().

RETURNS list[GeoPolygon] | GeoSurface | GeoVolume

FUNCTION

em.geo.PCB.**generate_all**()

Generate all PCB components to finalize the model. Air is always returned as a list of Air elements. Either empty if no heights are provided, a sinle one or both (top first, then bottom)

PARAMETERS
- **merge**: bool = False - Merge the PCB dielectric volumes into a single object.
- **height_air_top**: float | None = None - The height of the air box above the PCB.
- **height_air_bottom**: float | None = None - The height of the air box below the PCB.

RETURNS tuple[GeoVolume, list[GeoVolume], list[GeoVolume]]

FUNCTION

em.geo.PCB.**set_board_outline**()

Define a custom (non-rectangular) board outline polygon.

PARAMETERS
- **xs**: list[float] - The outline x-coordinates, in order, open (not repeating the first point).
- **ys**: list[float] - The outline y-coordinates, in order, open (not repeating the first point).

RETURNS None

FUNCTION

em.geo.PCB.**generate_silkscreen**()

Generates a silkscreen over the entire PCB with the provided material and thickness. If a thickness is empty, no geometry is generated for that side of the PCB.

PARAMETERS
- **material**: Material - The material of the Silkscreen
- **thickenss_top**: float = 0.0 - The thickness of the Top layer. Defaults to 0.0.
- **thickness_bottom**: float = 0.0 - The thickness of the bottom layer. Defaults to 0.0.

RETURNS GeoVolume | tuple[GeoVolume, GeoVolume]

FUNCTION

em.geo.PCB.calc.**z0**()

Backward-compatible alias for microstrip inverse width solve.

PARAMETERS
- **Z0**: float - Target impedance in ohms.
- **layer**: int = −1 - Layer index in the calculator stackup.
- **ground_layer**: int = 0 - Layer index in the calculator stackup.
- **f0**: float = 1000000000.0 - Frequency in hertz.
- **er**: float | None = None - Relative permittivity; overrides stackup material when provided.
- **include_dispersion**: bool = True - Include the frequency-dispersion correction.

RETURNS float

FUNCTION

em.geo.PCB.calc.**effective_er**()

Effective dielectric constant between two layers.

PARAMETERS
- **layer**: int - Layer index in the calculator stackup.
- **ground_layer**: int - Layer index in the calculator stackup.
- **f0**: float - Frequency in hertz.
- **er**: float | None = None - Relative permittivity; overrides stackup material when provided.

RETURNS float

PROPERTY

em.geo.StripPath.**xs/ys**

A list of all x/y-coordinates.

RETURNS list[float]

PROPERTY

em.geo.StripPath.**start**

The first RouteElement object

RETURNS RouteElement

PROPERTY

em.geo.StripPath.**end**

The last RouteElement object

RETURNS RouteElement

PROPERTY

em.geo.StripPath.**bck**

Moves the last RouteElement one half stripline width packwards and return this object.

RETURNS StripPath

PROPERTY

em.geo.StripPath.**skip**

Prepares to remove half of the stripline width in distance from the next route element. Return this object.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**straight**()

Move the StripPath a fixed distance forward in the current route direction.

PARAMETERS
- **distance**: float - The length of the stripline
- **width**: float | None = None - The width of the stripline.
- **dx**: float = 0 - An x-direction offset.
- **dy**: float = 0 - A y-direction offset.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**taper**()

Move some distance forward and taper the line width to a new width

PARAMETERS
- **distance**: float - The length of the stripline
- **width**: float - The width of the stripline.
- **dx**: float = 0.0 - An x-direction offset.
- **dy**: float = 0.0 - A y-direction offset.
- **offset_r**: float = 0.0 - An offset in the direction right relative to foward.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**turn**()

Perform a turn at this location. The rotation point will be either the right or left corner of the current routing position. Not the heartline. Use `pturn` for the heartline.

PARAMETERS
- **angle**: float - The turning angle
- **width**: float | None = None - The stripline width.
- **corner_type**: “champher” | “square” | “miter” | None = None - The corner type.
- **dsratio**: float | None = None - (float, optional): The chamfer distance expressed as the proportional of the corner diagonal.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**pturn**()

Same as the turn but the rotation point is at the center of the trace. This will consume some distance of line from the last and next routing section. **can cause invalid traces when routed to closely to other turns**.

PARAMETERS
- **angle**: float - The turning angle
- **corner_type**: “champher” | “square” | “miter” | None = None - The corner type.
- **dsratio**: float | None = None - The chamfer distance expressed as the proportional of the corner diagonal.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**curve**()

Route a curved path with some circle radius. The in and outside line segments are approximated by straight line sections.

PARAMETERS
- **angle**: float - The turning angle
- **radius**: float - The bend radius
- **width**: float | None = None - The stripline width.
- **dang**: float = 10 - The angular step in degrees used to discretize the bend.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**store**()

Store the current position (RouteElement) in the PCB memory to be recalled for XY coordinates later using `load`. Also see `lumped_port` and `modal_port`

PARAMETERS
- **name**: str | int - Any hashable code or identifier.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**split**()

Store the current position and start a completely new trace at the current location. Can later be returned to using `merge`

PARAMETERS
- **direction**: tuple[float, float] | None = None - The departure direction of the new path.
- **angle**: float | None = None - Rotation of the departure direction in degrees.
- **width**: float | None = None - The width of the new path.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**lumped_element**()

Add a lumped_element surface polygon at the current position and move ahead to where the element stops.

PARAMETERS
- **impedance_function**: Callable - A function that computes the component impedance as a function of frequency.
- **size**: SIZE_NAMES | tuple[float, float] - The dimensions of the lumped element on PCB.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**stub**()

Add a signel rectangular stub in some direction allowing you to continue as you where going. Stubs are placed starting from the center of the stripline, not the side edge.

PARAMETERS
- **direction**: tuple[float, float] - The stub direction as a (dx, dy) tuple
- **width**: float - The stub width
- **length**: float - The stub length
- **mirror**: bool = False - Also add the same stub in the opposite direction.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**merge**()

Move the current routing position back to where `split` was called last

RETURNS StripPath

FUNCTION

em.geo.StripPath.**via**()

Add a via in the current position to a new layer.

PARAMETERS
- **new_layer**: int - The layer index the via goes to
- **radius**: float | None = None - The via radius.
- **proceed**: bool = True - Wether to continue with a new trace.
- **direction**: tuple[float, float] | None = None - The new direction.
- **width**: float | None = None - The new width.
- **extra**: float | None = None - How much extra stripline to add around the via.
- **segments**: int = 6 - The number of via polygon sections.
- **annular_ring_radius**: float | None = None - The radius of the annular ring cut out of the ground planes and polygons around the via.
- **hole_radius**: float | None = None - The via hole radius for the Ground layers and polies.
- **hole_skip_layers**: list[int] | None = None - A list of layer numbers where the via hole should not be created.
- **reverse**: float = 0.0 - A distance to place the via back in the direction the stripline is coming from. Defaults to 0.0.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**short**()

Add a short circuit via to some ground layer.

PARAMETERS
- **ground_layer**: int = 0 - The layer number to short to.
- **radius**: float | None = None - The via radius.
- **reverse**: float = 0 - A displacement distance in the reverse direction.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**jump**()

Start a new trace by jumping relative to the current position. The jump displacement can be provided in two ways:
- dx + dy
- gap + side + reverse (useful for coupled line sections)

PARAMETERS
- **dx**: float | None = None - The jumps dx distance.
- **dy**: float | None = None - The jumps dy distance.
- **width**: float | None = None - The new stripline width.
- **direction**: tuple[float, float] | None = None - The new stripline direction.
- **gap**: float | None = None - The gap between the current and next stripline.
- **side**: “left” | “right” | None = None - The lateral jump direction.
- **reverse**: float | None = None - How much to move back if a lateral jump is made.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**to**()

Does a straight forward auto-route to a destination location. The route tries to find a valid turn angle to point towards the intended “intercept point” and then arrives at the destination based on the arrival direction and distance.

PARAMETERS
- **dest**: tuple[float, float] | Anchor - The destination xy coordinate
- **arrival_dir**: tuple[float, float] | None = None - The arrival direction as dx/dy tuple.
- **arrival_margin**: float | None = None - The minimal travel distance in the arrival direction.
- **angle_step**: float = 90 - The angle step size to solve for.
- **corner_type**: “champher” | “square” | “miter” | None = None - The corner type.
- **dsratio**: float | None = None - The cut ratio for champher/miter corners. Defaults to 0.7.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**init**()

Initializes the StripPath object for routing.

PARAMETERS
- **x**: float | Anchor - The start x-coordinate (or an Anchor)
- **y**: float - The start y-coordinate
- **width**: float - The trace width
- **direction**: tuple[float, float] - The start direction as a (dx, dy) tuple
- **trace_layer**: int = –1 - The copper layer.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**move**()

Move the routing cursur to any new desired location

PARAMETERS
- **dx**: float | None = None - A horizontal displacement.
- **dy**: float | None = None - A vertical displacement.
- **x**: float | None = None - The new x-coordinate.
- **y**: float | None = None - The new y-coordinate.
- **direction**: tuple[float, float] | None = None - The new direction.
- **width**: float | None = None - The new width.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**cut**()

Starts a new path at the current end point with the same width and direction.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**lumped_element_from_sp**()

Adds a lumped element to the PCB whose impedance is read from a Touchstone (.sNp) file.

PARAMETERS
- **filename**: str - The Touchstone file with the S-parameters of the component.
- **size**: SIZE_NAMES | tuple[float, float] - The dimensions of the lumped element on PCB.
- **ground_layer**: int = 0 - The layer the shunt elements of a pi-network connect to.

RETURNS StripPath

FUNCTION

em.geo.StripPath.**macro**()

Parse an EMerge macro command string The start direction by default is the abslute current heading. If a specified heading is provided the macro language will assume that as the current heading and generate commands accordingly. The language is specified by a symbol plus a number. Symbols: - = X: Move X units forward - \> X: Turn to right and move X forward - < X: Turn to left and move X forward - v X: Turn to down and move X forward - ^ X: Turn to up and move X forward - T X,Y: Taper X forward to width Y - \\ X: Turn relative right 90 degrees and X forward - / X: Turn relative left 90 degrees and X forward

PARAMETERS
- **path**: str - The path command string
- **width**: float | None = None - The width to start width.
- **start_dir**: tuple[float, float] | None = None - The start direction to assume.

RETURNS StripPath

10.9 Meshing Functions

FUNCTION

em.Simulation.**set_resolution**()

Specifies the maximum mesh size as a fraction of th greatest simulation wavelength. A factor of 0.3 means that the mesh in any domain may not be larger than $0.3\frac{c_0}{f_{\max}}$. See also set_resolution#mw

PARAMETERS
- **resolution**: float - The max mesh size resolution.

RETURNS None

FUNCTION

em.Mesher.**set_curved_boundary_meshing**()

Specify a detail factor by which to approximate curved boundaries. A higher number means that curved boundaries are meshed with more elements.

PARAMETERS
- **factor**: float = 10.0 - The detail factor

RETURNS None

FUNCTION

em.Mesher.**unset_constraints**()

Remove all mesh size constraints from a geometric entity specified by a list of dimtag pairs.

PARAMETERS
- **dimtags**: list[tuple[float, float]] - A list of dimtag tuples.

RETURNS None

FUNCTION

em.Mesher.**set_boundary_size**()

Specify a mesh size on the boundary of a given geometric entity. The boundary of a geometry is always one dimension lower than the geometry itself. The growth-rate represents how much the mesh size is allowed to grow away from where its specified.
- 1.0 - No growth
- 2.0 - Each next element away from the last can be twice larger.

At some point a larger growth rate has very little effect.

PARAMETERS
- **boundary**: GeoObject | Selection | Iterable - A the geometries of which you want the boundary to be meshed finer.
- **size**: float - The desired mesh size on the boundary
- **growth_rate**: float = 3.0 - How fast the mesh size is allowed to grow away from the boundary
- **max_size**: float | None = None - The maximum mesh size away from the boundary. Defaults to the max mesh-size specified by the material + resolution.

RETURNS None

FUNCTION

em.Mesher.**set_trace_refinement**()

Apply a trace refinement mesh size specification for a set of 2D polygons in the same plane. The mesh size is determined based on how close parallel lines are with respect to each other. The trace refinement algorithm requires these line segments to overlap some minimum amount before they are counted as being parallel.

PARAMETERS
- **obj**: GeoSurface | FaceSelection | Iterable - The trace or set of traces to refine
- **qualtity_factor**: float = 3.0 - How much to refine. Greater means finer mesh. Defaults to 3.0.
- **growth_rate**: float = 3.0 - How quickly the mesh size is allowed to grow. Defaults to 3.0.
- **min_size**: float = 1e-05 - The minimum size of the mesh in meters.
- **max_size**: float | None = None - The maximum size to constrain to.
- **edge_size**: float | None = None - If given, an additional boundary mesh size constraint on the trace edges.
- **min_overlap**: float = 0.75 - The minimum overlap for two edges to be refined. Defaults to 0.75 = 75%

RETURNS None

FUNCTION

em.Mesher.**set_domain_size**()

Specify the maximum size in a domain

PARAMETERS
- **obj**: GeoObject | Selection
- **size**: float - The maximum size

RETURNS None

FUNCTION

em.Mesher.**set_face_size**()

Specify the maximum size on a surface

PARAMETERS
- **obj**: GeoSurface | Selection
- **size**: float - The maximum size

RETURNS None

FUNCTION

em.Mesher.**set_pec_face**()

Specifies that a face can be meshed as coarsely as is allowed. Can be used to save mesh elements on PEC faces where the E-field is 0 anyway.

PARAMETERS
- **obj**: GeoSurface | Selection
- **size**: float | None - An optional maximum mesh size.

RETURNS None

FUNCTION

em.Mesher.**set_size**()

Specify the maximum size in an object of any dimension.

PARAMETERS
- **obj**: GeoObject
- **size**: float - The maximum size

RETURNS None

FUNCTION

em.Mesher.**set_global_max_size**()

Sets an upper limit for the mesh element size everywhere in the model.

PARAMETERS
- **size**: float | None - Maximum element size in meters. None removes the limit.

RETURNS None

FUNCTION

em.Mesher.**set_pec_domain**()

Sets the size on PEC faces to be very coarse to save computational time.

PARAMETERS
- **obj**: GeoObject | Selection - The volumetric domain
- **size**: float = −1 - The maximum size

RETURNS None

FUNCTION

em.Mesher.**set_algorithm**()

Set the GMSH 3D meshing algorithm.

Uses 1 GMSH thread by default (was 16).

PARAMETERS
- **algorithm**: Algorithm3D - The meshing algorithm
- **nthreads**: int = 1 - The number of threads to use (HXT only if installed correctly).

RETURNS None

FUNCTION

em.Mesher.**set_periodic_cell**()

Sets the periodic cell information based on the PeriodicCell class object

PARAMETERS
- **cell**: PeriodicCell - The periodic cell
- **excluded_faces**: Selection | None = None - Faces to exclude from the periodic face pairs.

RETURNS None

10.10 Microwave Physics

FUNCTION

Microwave3D.**reset**()

Resets the boundary conditions and microwave physics state variables.

PARAMETERS
- **_reset_bc**: bool = True - If the boundary conditions should be reset.

RETURNS None

FUNCTION

Microwave3D.**set_frequency**()

Set a single frequency or a list of frequencies as simulation frequencies.

PARAMETERS
- **frequency**: float | Iterable[float] - The freqeuency or frequencies.

RETURNS None

FUNCTION

Microwave3D.**set_frequency_range**()

Specify the frequency range as a `np.linspace` like format.

PARAMETERS
- **fmin**: float - The minimum frequency
- **fmax**: float - The maximum frequency
- **Npoints**: int - The number of points

RETURNS None

FUNCTION

Microwave3D.**set_resolution**()

Specifies the maximum mesh size as a fraction of th greatest simulation wavelength. A factor of 0.3 means that the mesh in any domain may not be larger than $0.3\frac{c_0}{f_{\max}}$

PARAMETERS
- **resolution**: float - The resolution as a fraction of the wavelength.

RETURNS None

FUNCTION

Microwave3D.**modal_analysis**()

Perform a port eigenmode analysis to find the eigenmodes of a ModalPort object. Modal solutions are stored inside the port object.

PARAMETERS
- **port**: ModalPort - The port object to execute the analysis for.
- **nmodes**: int = 6 - The number of modes to solve for.
- **direct**: bool = True - Whether to use the direct solver (LAPACK) if True. Otherwise it uses the iterative ARPACK solver, which requires an estimate of the propagation constant and is faster for a large number of degrees of freedom.
- **TEM**: bool = False - Whether to estimate the propagation constant assuming its a TEM transmission line.
- **target_kz**: float | None = None - The expected propagation constant to find a mode for (direct = False).
- **target_neff**: float | None = None - The expected effective mode index defined as kz/k0 (1.0 = free space, <1 = TE/TM, >1 = slow waves).
- **freq**: float | None = None - The desired frequency at which the mode is solved. If None then it uses the lowest frequency of the provided range.
- **neff2_shifts**: list[float] | None = None - Search for the modes closest to these values of neff² = β²/k0² (negative for evanescent modes, np.inf for the largest εr·μr of the port face) instead of around target_kz. Evanescent modes are kept. Used by the mode tracking of WavePort.

RETURNS None

FUNCTION

Microwave3D.**run_sweep**()

Perform a frequency sweep simulation, one frequency point after the other (the solver itself is multi-threaded). Also used for a single frequency.

Changed in 3.0: `parallel`, `n_workers` and `multi_processing` are deprecated here; use `run_sweep_distributed` to solve several frequency points at once.

PARAMETERS
- **nthreads**: int | None = None - The number of CPU threads of the solver, for solvers that support it (see Solver.set_nthreads). None keeps the solver's own setting.
- **frequency_groups**: int = −1 - The number of frequency points in a solve group. −1 is all.
- **cache_harddisk**: bool = False - Cache the assembled matrices on disk instead of in memory.
- **harddisc_path**: str = "EMergeSparse" - The cached matrix path name.
- **automatic_modal_analysis**: bool = True - Automatically compute port modes.

RETURNS MWData

FUNCTION

Microwave3D.**run_sweep_distributed**()

Executes a frequency domain study, solving n_workers frequency points at the same time. Each worker solves one frequency point with nthreads CPU threads (for solvers that support it), so the sweep uses up to n_workers * nthreads threads in total. Every worker holds its own factorization, so the memory use grows with n_workers. This is faster than run_sweep() when one solve does not use all cores efficiently, e.g. for small models with many frequency points.

PARAMETERS
- **n_workers**: int = 2 - The number of frequency points solved at the same time.
- **nthreads**: int | None = None - The number of CPU threads of each worker's solver, for solvers that support it (see Solver.set_nthreads). None keeps the solver's own setting.
- **frequency_groups**: int = −1 - The number of frequency points in a solve group. −1 is all.
- **cache_harddisk**: bool = False - Cache the assembled matrices on disk instead of in memory.
- **harddisc_path**: str = "EMergeSparse" - The cached matrix path name.
- **worker_type**: "thread" | "process" = "thread" - Whether the workers are threads or separate processes (slower on most machines).
- **automatic_modal_analysis**: bool = True - Automatically compute port modes.

RETURNS MWData

FUNCTION

Microwave3D.**run_adaptive_sweep**()

Adaptive frequency sweep. Fits an AAA rational model to the S-parameters and adds frequency samples until the estimated error is below `tol`. Evaluate the dense response with `data.scalar.grid.model_S(...)` (see `model_S`).

PARAMETERS
- **n_initial**: int = 5 - The number of equidistant initial samples.
- **tol**: float = 0.005 - Absolute tolerance on the S-parameters.
- **max_samples**: int = 100 - Maximum total number of samples.
- **harddisc_path**: str = "EMergeSparse" - The cached matrix path name.
- **frequency_groups**: int = −1 - The number of frequency points in a solve group.
- **cache_harddisk**: bool = False - Cache the matrices on the hard disk.
- **multi_processing**: bool = False - Whether to use multiprocessing instead of multi-threaded (slower on most machines).
- **automatic_modal_analysis**: bool = True - Automatically compute port modes.
- **nthreads**: int | None = None - The preferred number of CPU threads of the solver, for solvers that support it (see Solver.set_nthreads). None keeps the solver's own setting.

RETURNS MWData

FUNCTION

Microwave3D.**run_scattered**()

Performs a scattered field simulation (requires a `ScatteredField` boundary condition), one frequency point after the other. Use `run_scattered_distributed(...)` (same arguments as `run_sweep_distributed`) to solve several points at once.

PARAMETERS
- **nthreads**: int | None = None - The number of CPU threads of the solver, for solvers that support it (see Solver.set_nthreads). None keeps the solver's own setting.
- **frequency_groups**: int = −1 - The number of frequency points in a solve group. −1 is all.
- **cache_harddisk**: bool = False - Cache the assembled matrices on disk instead of in memory.
- **harddisc_path**: str = "EMergeSparse" - The cached matrix path name.
- **automatic_modal_analysis**: bool = True - Automatically compute port modes.

RETURNS MWData

FUNCTION

Microwave3D.**eigenmode**()

Perform an eigenmode simulation on the simulation domain.

PARAMETERS
- **search_frequency**: float - The frequency around which you would like to search
- **nmodes**: int = 6 - The number of modes.
- **k0_limit**: float = 1 - The lowest k0 value before which a mode is considered part of the null space.
- **direct**: bool = False - Use a direct solver for the shift-invert step.
- **mode**: "LM" | "LR" | "SR" | "LI" | "SI" = "LM" - Which eigenvalues to find (ARPACK convention).

RETURNS MWData

FUNCTION

Microwave3D.**ports**()

A list of all port boundary conditions.

RETURNS list[PortBC]

FUNCTION

Microwave3D.**conducting_surfaces**()

Returns a list of all surfaces that are in some way considered conducing by the boundary conditions.

RETURNS FaceSelection

FUNCTION

Microwave3D.**set_basis_space**()

Define the finite element basis order or element space.

PARAMETERS
- **element_order**: 1 | 2 = 2 - The order of basis functions
- **complete**: bool = False - If complete over mixed set of basis function is to be used.
- **elementspace**: ElementSpace | None = None - Manually specify the ElementSpace.
- **dof_set**: DoFSet | None = None - A user defined DoF set.

RETURNS None

FUNCTION

Microwave3D.**solve_cnosdd**()

Conformal, Non-overlapping Schwarz Domain Decomposition solve

Experimental domain-decomposition solver.

PARAMETERS
- **frequency**: float - The simulation frequency.
- **domains**: list[GeoVolume] - The domain partition.
- **ports**: list[int | float] | None = None - Which driven ports' smat_index values to excite and solve for. Each requested port costs one full DD/GMRES solve (subdomain factorizations are reused across ports, but the Krylov solve itself is not).
- **solver**: SolverName = "gmres" - The Krylov solver of the interface problem.

RETURNS MWData

10.11 Microwave Boundary-Conditions

The following are available under your simulation object:

```python
sim = em.Simulation(...)
sim.mw.bc.PEC(...) # For examle
```

FUNCTION

MWBoundaryConditionSet.**PEC**()

Add a perfect-electric-conductor boundary condition.

PARAMETERS
- **face**: FaceSelection | GeoSurface

RETURNS BoundaryCondition

FUNCTION

MWBoundaryConditionSet.**PMC**()

Add a perfect-magnetic-conductor boundary condition.

PARAMETERS
- **face**: FaceSelection | GeoSurface

RETURNS BoundaryCondition

FUNCTION

MWBoundaryConditionSet.**AbsorbingBoundary**()

Adds an absorbing boundary condition(ABC). This is also called the Radiation boundary condition in many solvers. It can be the first order ABC or second order (ABC). The second order can absorb radiation better from larger angles of incidence. (See manual). The ABC type for order changes at what angle of incidence range the second order correction is optiomized. See the graph below.

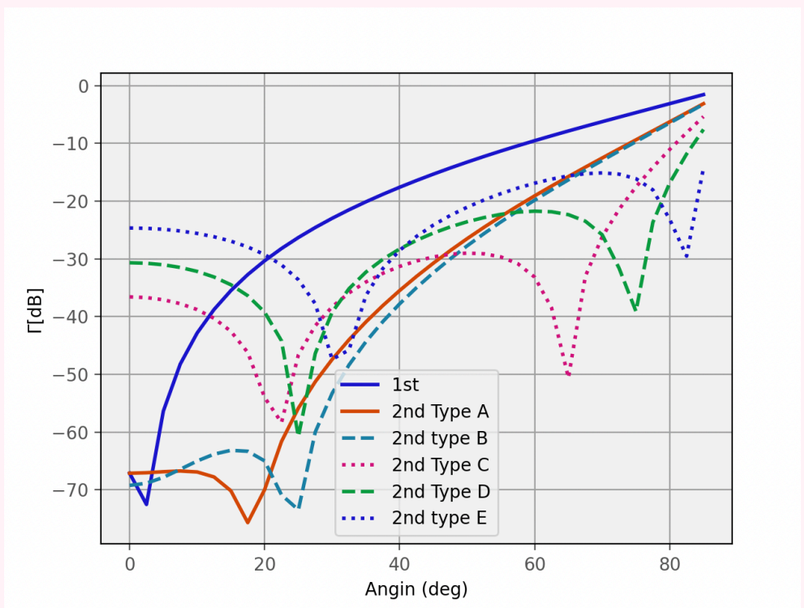


FIGURE 1 The reflection coefficient of the different ABC types.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The absorbing boundary face(s)
- **order**: int = 2 - The order of the boundary condition (1 or 2).
- **origin**: tuple | None = None - The radiation origin.
- **abctype**: “A” | “B” | “C” | “D” | “E” = “B” - The second order ABC coefficient set. A is the classic second order Taylor expansion, B to E trade normal incidence accuracy for a wider angular range.

RETURNS AbsorbingBoundary

FUNCTION

AbsorbingBoundary.**optimize_for_maximum_angle**()

Sets the second-order ABC coefficients (c1, c2) that minimize the worst-case reflection coefficient across the incidence angles [0, angle_deg]. the worst-case reflection coefficient across the range [0, theta_max_deg].

PARAMETERS
- **angle_deg**: float - Maximum incidence angle in degrees (e.g. 60.0).

RETURNS None

FUNCTION

AbsorbingBoundary.**set_zero_angles**()

Define two angles for which the absorbing boundary condition has a zero reflection

PARAMETERS
- **ang1_deg**: float - The first zero reflection incidence angle in degrees
- **ang2_deg**: float - The second zero reflection incidence angle in degrees

RETURNS None

FUNCTION

MWBoundaryConditionSet.**ScatteredField**()

Adds a ScatteredField boundary condition. This boudnary condition should be applied to a surface enclosing the entire simulation domain. The ScatteredField boudnary condition is used exclusively for electromagnetic scattering simulations such as radar-cross-section computations.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The boundary face selection
- **power_density**: float = 1.0 / (2 * Z0) - The power density of the incident plane waves in W/m².
- **definition**: “EA” | “SPH” = “SPH” - Angle convention: `"SPH"` standard spherical angles (default since 3.0) or `"EA"` elevation/azimuth (old behaviour)
- **material_correction**: bool = False - Scale the absorbing constant with the adjacent permittivity
- **cs**: CoordinateSystem | None = None - The coordinate system of the excitation angles.

RETURNS ScatteredField

FUNCTION

ScatteredField.**set_excitations**()

Define the set of plane wave excitations for the ScatteredField boundary condition. The excitations use an elevation/azimuth coordinate system definition:

- $\theta(\mathrm{deg})$ - elevation
- $\varphi(\mathrm{deg})$ - azimuth
- $\psi(\mathrm{deg})$ - polarization

Then:

- $\theta = 0, \varphi = 0, \psi = 0$: $\boldsymbol{k} = +X$, $\boldsymbol{E} = E_z\hat{z}$
- $\theta = 0, \varphi = 90, \psi = 0$: $\boldsymbol{k} = +Y$, $\boldsymbol{E} = E_z\hat{z}$
- $\theta = 90, \varphi = 0, \psi = 0$: $\boldsymbol{k} = -Z$, $\boldsymbol{E} = E_x\hat{x}$
- $\theta = 0, \varphi = 0, \psi = 90$: $\boldsymbol{k} = +X$, $\boldsymbol{E} = E_y\hat{y}$

Please check the manual for more details.

PARAMETERS
- **thetas**: np.ndarray | float = 0.0 - One or more theta angles
- **phis**: np.ndarray | float = 0.0 - One or more phi angles
- **polarizations**: np.ndarray | float = 0.0 - On or more psi (polarization) angles.

RETURNS None

FUNCTION

MWBoundaryConditionSet.**LumpedElement**()

Adds a lumped element boundary condition to a surface. This boundary condition only holds true results if the surface you assign it to is isometric to a rectangle (so rectangles, boundaries of cylinders etc.)

PARAMETERS
- **face**: FaceSelection | GeoSurface - The element surface
- **impedance_function**: Callable | None = None - The element impedance Z(f) as a function of frequency in Hz. Taken from the face’s LumpedElementAttribute if None.
- **width**: float | None = None - The element width (meters).
- **height**: float | None = None - The element height (meters) along which the potential is imposed.

RETURNS LumpedElement

FUNCTION

MWBoundaryConditionSet.**SurfaceImpedance**()

Adds a SurfaceImpedance boundary condition. The surface impedance boundary condition must be used to model the boundaries of conductors. It accurately models the skin effect. **Do not use the SurfaceImpedance boundary condition to model 2D geometries that represent traces with finite thickness such as (micro-)striplines. Use the `ThinConductor` boundary condition instead**.

This boundary condition can be used to model for example the losses on the walls of a waveguide or the boundary surface of a coaxial core conductor.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The face to apply this condition to.
- **material**: Material | None = None - The matrial to assign.
- **surface_conductivity**: float | None = None - The specific bulk conductivity to use.
- **surface_roughness**: float = 0 - The surface roughness.
- **thickness**: float | None = None - The layer thickness.
- **sr_model**: “Hammerstad-Jensen” = “Hammerstad-Jensen” - The surface roughness model.
- **impedance_function**: Callable | None = None - A user defined surface impedance function as function of frequency.
- **surface_conductance**: float | None = None - Deprecated alias of surface_conductivity.

RETURNS SurfaceImpedance

FUNCTION

MWBoundaryConditionSet.**ThinConductor**()

Adds a ThinConductor boundary condition. The thin impedance boundary condition must be used to model thin conducting object that you wish to model as a 2D surface instead. ThinConductor boundaries must not border on a bulk conductor. It must be assigned to flat structures like PCB traces etc.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The face to apply this condition to.
- **material**: Material | None = None - The matrial to assign.
- **thickness**: float | None = None - The layer thickness.
- **surface_conductance**: float | None = None - The specific bulk conductivity to use.
- **surface_roughness**: float = 0 - The surface roughness.
- **sr_model**: “Hammerstad-Jensen” = “Hammerstad-Jensen” - The surface roughness model.
- **impedance_function**: Callable | None = None - A user defined surface impedance function as function of frequency.

RETURNS ThinConductor

FUNCTION

MWBoundaryConditionSet.**Void**()

Marks a volume as a void: it is removed from the simulation domain and its boundary becomes PEC. Equivalent to adding `em.VoidAttribute()` to the object’s properties.

PARAMETERS
- **assignment**: GeoVolume | DomainSelection - The void volume

RETURNS Void

FUNCTION

MWBoundaryConditionSet.**ModalPort**()

Adds a ModalPort object to the simulation. A modal port is also called a “Wave Port” or “Numeric Port” in other solvers. All modes of a ModalPort are absorbed with one propagation constant, which only works for degenerate modes (e.g. the two TE11 polarizations); use `WavePort` for modes with different β. Since 3.0.4 a ModalPort also works below the cutoff of its mode. Modal ports require a modal field solution to first be computed. This is done automatically when you call `.run_sweep()` or can be done manually with the `.modal_analysis()` function of the `Microwave3D` class.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port mode face
- **port_number**: int - The port number as an integer
- **cs**: CoordinateSystem | None = None - The local coordinate system of the port face.
- **power**: float = 1 - The radiated power.
- **modetype**: “TE” | “TM” | “TEM” | “any” | None = None - Force the port mode type (None selects automatically). “TE” or “TM” only accepts modes of that type. “TEM” treats every mode as (quasi-)TEM, so a quasi-TEM mode that is classified as TE or TM (settings.qtem_limit) still gets a characteristic impedance and its sign from the voltage.
- **number_of_modes**: int = 1 - The number of port modes to compute.
- **impedance_definition**: “PV” | “PI” | “VI” = “PV” - How the characteristic impedance is computed: from power and voltage, power and current, or voltage and current.
- **assume_homogeneous**: bool = False - Treat the port face as homogeneous without checking its materials, see .assume_homogeneous().
- **target_neff**: float | None = None - The effective mode index β/k0 of the wanted mode. The first modal analysis searches around it and the modes are ordered by their distance to it, so mode 1 is the mode closest to target_neff. At the next frequencies the mode is followed from the previous one. None searches around the mean of the largest and smallest εr·μr of the port face and orders the modes by β.

RETURNS ModalPort

FUNCTION

MWBoundaryConditionSet.**WavePort**()

A wave port that absorbs and excites each of its modes with the mode’s own propagation constant.

The modes are selected at the highest frequency and followed down the sweep, so an S-matrix index (`1.1`, `1.2`, …) is the same physical mode at every frequency. Modes below cutoff stay in the S-matrix as evanescent modes (β = −jα). A port face with one lossless, non-dispersive material is solved once and scaled analytically.

```python
p1 = model.mw.bc.WavePort(wg.front, 1, number_of_modes=3)
S = data.scalar.grid.S(1.2, 2.2)  # mode 2 of port 1 → mode 2 of port 2
```

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port mode face
- **port_number**: int - The port number as an integer
- **cs**: CoordinateSystem | None = None - The local coordinate system of the port face.
- **power**: float = 1 - The radiated power.
- **number_of_modes**: int | None = 1 - The number of port modes. None includes every mode that propagates at the highest frequency.
- **impedance_definition**: “PV” | “PI” | “VI” = “PV” - How the characteristic impedance is computed: from power and voltage, power and current, or voltage and current.
- **modetype**: “TE” | “TM” | “TEM” | “any” | None = None - Only accept port modes of this type. None or “any” accepts all modes.
- **assume_homogeneous**: bool = False - Treat the port face as homogeneous without checking its materials, see .assume_homogeneous().

RETURNS WavePort

PROPERTY

MWBoundaryConditionSet.**WavePortIH**

Deprecated in 3.0. Still works, with a warning, as `WavePort(..., modetype="TEM")`.

RETURNS WavePort

FUNCTION

ModalPort.**set_integration_line**()

Define the integration line for the voltage between two points in 3D space.

PARAMETERS
- **c1**: tuple[float, float, float] - The start of the integration line
- **c2**: tuple[float, float, float] - The end of the integration line
- **N**: int = 51 - The number of points along the integration line.

RETURNS None

FUNCTION

ModalPort.**align_modes**()

Define a sequence of Axis objects or vectors as tuples that are used in order to align/orient the modal solutions. This can be used with square or circular waveguides for example to make sure that the two fundamental modes are ordered consistently and point in the same direction at each frequence.

This will cause the order to be kept consistent and it will reverse the polarization so that the field at phase=0 is always pointing in the same direction.

Example: You have a square waveguide in the XZ plane and you want to make sure that mode 1 points in the +X direction and mode 2 in the +Z direction then you can do:

```python
mp_object.align_modes(em.XAX, em.ZAX)
# or
mp_object.align_modes((1,0,0), (0,0,1))
```

PARAMETERS
- ***axes**: tuple | np.ndarray | Axis - A sequence of vectors/axes to align the modes with.

RETURNS None

FUNCTION

ModalPort.**set_terminals**()

Label geometries as terminals in case you have a modal port that consists of 3 or more separate conductors such as differential microstriplines. The two terminals will be connected straight with a voltage integration line. Also used for two-conductor ports to fix the sign of the mode; with only one terminal the other is the closest other conductor.

PARAMETERS
- **positive**: Selection | GeoObject | None = None - The postive terminal.
- **negative**: Selection | GeoObject | None = None - The negative terminal.
- **ground**: Selection | GeoObject | None = None - The ground terminal (currently unused).

RETURNS None

FUNCTION

PortBC.**assume_homogeneous**()

Treat the port face as one homogeneous material, without checking the materials on it. The modes of a homogeneous port are solved once and scaled analytically to the other frequencies: $\beta^2 = \beta_{ref}^2 + \varepsilon_r \cdot \mu_r (k_0^2 - k_{0,ref}^2)$ for TE and TM modes (with the mean $\varepsilon_r \cdot \mu_r$ of the face), $\beta \propto k_0$ for TEM modes, which keeps the effective permittivity of a quasi-TEM line. The mode shapes are taken frequency independent, so dispersion of an inhomogeneous line (microstrip at a high frequency·thickness) is ignored.

PARAMETERS
- **active**: bool = True - Whether to assume a homogeneous port face.

RETURNS PortBC

FUNCTION

PortBC.**smatrix_index**()

The S-matrix index of port mode mode_nr, to read the results with (data.scalar.grid.S(i, j)). A port with one mode has its port number as index. With more modes the index is port.mode: 1.1, 1.2, ... and from 10 modes on the mode number has two digits: 1.01, 1.02, ..., 1.12.

PARAMETERS
- **mode_nr**: int = 1 - The port mode number (1-based).

RETURNS int | float

FUNCTION

ModalPort.**reset**()

Clear all computed modes. The terminal assignment (.set_terminals) is kept.

RETURNS None

FUNCTION

MWBoundaryConditionSet.**FloquetPort**()

Creates a FloquetPort boundary condition. If you used `RectCell` or `HexCell` to define the periodicity of your simulation domain use `floquet_port` instead.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port face
- **port_number**: int - The port number
- **cs**: CoordinateSystem | None = None - Not used, the modes are defined in the global XY plane.
- **polarization_deg**: float = 0.0 - For number_of_modes=1: the polarization of the single port mode, 0 = TE (S), 90 = TM (P). Defaults to 0.0.
- **power**: float = 1.0 - The port power. Defaults to 1.0.
- **er**: float = 1.0 - The relative permittivity of the medium at the port. Defaults to 1.0.
- **number_of_modes**: int | None = 1 - The number of port modes in the S-matrix. 1: one mode of the (0, 0) harmonic with the given polarization (both polarizations are still absorbed). 2: TE00 and TM00. More: also the grating modes (m, n) ≠ (0, 0), ordered by their cutoff at the highest frequency. None: every mode that propagates at the highest frequency.

RETURNS FloquetPort

FUNCTION

MWBoundaryConditionSet.**floquet_port**()

Adds a Floquet port on a periodic cell (`RectCell`, `HexCell`, ...). With `number_of_modes=1` (default) one polarization-mixed mode; `2` gives TE00 and TM00 as `n.1` and `n.2`; more modes add the higher diffraction orders as TE/TM pairs; `None` takes every order that propagates at the highest frequency.

PARAMETERS
- **poly**: GeoSurface - The port face
- **port_number**: int - The port number
- **number_of_modes**: int | None = 1 - The number of Floquet modes
- **polarization_deg**: float = 0.0 - The polarization angle of the single-mode excitation in degrees
- **er**: float = 1.0 - The relative permittivity at the port

RETURNS FloquetPort

FUNCTION

FloquetPort.**floquet_modes**()

The port modes: per mode its S-matrix index, Floquet order (m, n) and polarization (“TE” or “TM”). The modes are selected at the highest frequency when the simulation runs, so call this after run_sweep(). A port with number_of_modes=1 has one mode of order (0, 0) whose polarization is “mixed” unless the polarization angle is 0 (TE) or 90 (TM) degrees.

RETURNS list[tuple[int | float, int, int, str]]

FUNCTION

FloquetPort.**order_direction**()

The direction of Floquet order (m, n) at freq, away from the cell through this port.

PARAMETERS
- **freq**: float - The frequency in Hz
- **m**: int - The order along the first lattice vector
- **n**: int - The order along the second lattice vector

RETURNS `tuple[float, float] | None`

FUNCTION

FloquetPort.**set_polarization**()

Set the polarization angle of a single mode port from 0deg = S (TE) to 90deg = P (TM).

PARAMETERS
- **polarization_deg**: float - The polarization angle in degrees

RETURNS None

FUNCTION

MWBoundaryConditionSet.**RectangularWaveguide**()

Creates a RectangularWaveguide boundary condition. The RectangularWaveguide boundary condition must be applied to surfaces that are rectangular.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port boundary face selection
- **port_number**: int - The port number
- **mode**: tuple[int, int] = (1, 0) - The mode number (m, n).
- **mode_type**: “TE” | “TM” = “TE” - The mode family.
- **er**: float = 1.0 - The Dielectric constant. Defaults to 1.0.
- **cs**: CoordinateSystem | None = None - The local coordinate system.
- **dims**: tuple[float, float] | None = None - The port face.
- **power**: float = 1 - The port power.

RETURNS RectangularWaveguide

FUNCTION

RectangularWaveguide.**align_modes**()

Define vector using a tuple or numpy array or Axis objects that will be used used to align the modes in the right direction. For the fundamental TE10 mode, the phase=0 polarization may point either up or down which an add arbitrary phase shifts. By setting this vector you can make the port phase constistent.

Example: You have a rectangular waveguide in the XZ plane and you want to make sure that mode points up in the +Z direction use:

```python
mp_object.align_modes(em.ZAX, )
# or
mp_object.align_modes((0,0,1))
```

PARAMETERS
- **axis**: tuple | np.ndarray | Axis - The alignment vector

RETURNS None

FUNCTION

MWBoundaryConditionSet.**CoaxPort**()

Creates a CoaxPort boundary condition. The coax port only supports the fundamental TEM coax mode. It is usually less accurate than that of a ModalPort because the curved boundaries of a coax port are rarely so ideal that the analytical solution is sufficiently accurate. Differences can be small. This will save a lot of simulation time in some cases.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port boundary face selection
- **port_number**: int - The port number
- **rad_in_out**: tuple[float, float] - The inner and outer conductor radius (meters)
- **cs**: CoordinateSystem - The local coordinate system with its origin at the coax center and Z-axis along the coax.
- **er**: float = 1.0 - The dielectric constant of the coax filling. Defaults to 1.0.
- **power**: float = 1 - The port power.

RETURNS CoaxPort

FUNCTION

MWBoundaryConditionSet.**UserDefinedPort**()

Create A UserDefinedPort object with a manual E-field function definition.

Since version 2.7, this port may have issues with TEM like non plane wave ports because EMerge requires the magnetic field definition as well for the characteristic impedance calculation. It currently does not finite differences for example to compute the curl.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port boundary face selection
- **port_number**: int - The port number
- **Ex**: Callable | None = None - The Ex(k0,x,y,z) field
- **Ey**: Callable | None = None - The Ey(k0,x,y,z) field
- **Ez**: Callable | None = None - The Ez(k0,x,y,z) field
- **kz**: Callable | None = None - The out of plane propagation constant kz(k0)
- **power**: float = 1.0 - The port output power
- **modetype**: “TEM” | “TE” | “TM” = “TEM” - The mode type of the port.
- **cs**: CoordinateSystem | None = None - The local coordinate system of the port face.

RETURNS UserDefinedPort

FUNCTION

MWBoundaryConditionSet.**LumpedPort**()

Create A LumpedPort object in the simulation domain. Lumped ports are ought to be used on small surfaces that are isometric to a rectangle (so rectangles and the boundary of a cylinder etc.). It works best if the port is small relative to the wavelength. The port mode will be a uniform E-field. The lumped port will become a sheet resistor that has a total resistance equal to the port impedance.

You must provide EMerge with information of the port dimensions and the direction of the E-field. If you pass the geometry defined by the `PCB` class’s `lumped_port` function, these properties will be passed as metadata automatically.

PARAMETERS
- **face**: FaceSelection | GeoSurface - The port surface
- **port_number**: int - The port number
- **width**: float | None = None - The port width (meters).
- **height**: float | None = None - The port height (meters).
- **direction**: Axis | tuple[float, float, float] | None = None - The port direction as an Axis object (em.Axis(..) or em.ZAX)
- **power**: float = 1 - The port output power.
- **Z0**: float = 50 - The port impedance.

RETURNS LumpedPort

FUNCTION

MWBoundaryConditionSet.**get_port**()

Return the port boundary condition object for the given port number.

PARAMETERS
- **port_number**: int - The desired port number

RETURNS PortBC | None

FUNCTION

MWBoundaryConditionSet.**get_ports**()

Returns a tuple of all port boundary conditions

RETURNS `tuple[PortBC]`

11. Data

FUNCTION

em.**MWData**()

A generic dataclass container for all microwave related data. Has two main properties:
- `.scalar` - All scalar data like S-parameters and port impedances.
- `.field` - Stores all field solutions

PARAMETERS Should not be created manually by users.

RETURNS None

FUNCTION

em.MWData.**merge_with**()

Merge all solutions of this dataset with other MWData objects.

PARAMETERS
- ***others**: MWData - Other dataset objects as a sequence of arguments.

RETURNS MWData

FUNCTION

em.MWData.**export_farfields**()

Exports all farfield data to a file.

PARAMETERS
- **filename**: str - The filename to export to
- **face**: FaceSelection | GeoSurface - The integration surface for the farfield calculation.
- **thetas**: ndarray - An optional array of theta angles
- **phis**: ndarray - An optional array of phi angles
- **origin**: tuple[float, float, float] | None = None - An optional array for a radiation origin used to determine the normal vectors of farfield boundaries if inside vs. outside is not well defined.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes where Ex is E-symmetry in x=0.
- **precision**: int = 4 - The number of decimals for the output file.
- **frequencies**: list[float] | None = None - The frequencies to pick for the output.
- **parameters

RETURNS None

FUNCTION

em.**MWField**()

A class containing a set of EM field solutions for each excited port for a single frequency. The solution is stored as the solution vector to the original FEM problem: $Ax = b$. To compute the E-field at specific coordinates you must use the `interpolate` function to compute the field at those coordinates.

PARAMETERS Should not be created manually by users.

RETURNS None

PROPERTY

em.MWField.**relative**

A method chaining property that flips the fields to the relative field instead of the total field. Usable with fields generated by the scattered field simulation. Example:

```python
data = sim.mw.run_scattered()
field = data.field.find(freq=1e9).relative.some_function()
```

RETURNS MWField

FUNCTION

em.MWField.**excite_port**()

Set the port excitation of a single port number with a specific complex amplitude.

PARAMETERS
- **number**: int | float = The port (port mode) index of the port(mode) to be excited.
- **excitation**: complex = 1 + 0j - The complex port excitation.

RETURNS None

FUNCTION

em.MWField.**set_excitations**()

Specify the port excitation of each port in the dataset as a sequence of complex amplitude arguments. Example usage:

```python
field.set_excitations(0,1.0,0) # three port simulation
```

PARAMETERS
- ***excitations**: complex - A sequence of arguments for each port(mode) excitation in order.

RETURNS None

FUNCTION

em.MWField.**combine_ports**()

Combines two ports to function as a differential port where the two port indices i and j are replaced respectively by:
- $i \to i + j$ differential mode
- $j \to i + j$ common mode

PARAMETERS
- **p1**: int - The first port number
- **p2**: int - The second port number

RETURNS MWField

FUNCTION

em.MWField.**interpolate**()

Compute the E-field and H-field at the requested coordinates and return an MWField data container object. See `EHField`

The data can be passed as a numpy array of any dimensionality. Data will be returned in the same dimension.

PARAMETERS
- **xs**: ndarray - The X-coordinates
- **ys**: ndarray - The Y-coordinates
- **zs**: ndarray - The Z-coordinates
- **usenan**: bool = True - Return NaN for points outside the mesh (otherwise zero).
- **mapping**: ndarray | None = None - A precomputed point to tetrahedron mapping.

RETURNS EHField

FUNCTION

em.MWField.**int_surf**()

Computes a surface integration along a surface object or face selection.

To specify which field to integrate, the user must provide a function that takes a single argument (EHField object) and returns a specific field component. Example

```python
field.int_surf(selection, lambda eh: eh.Sx)
```

Currently normal vectors are not available quantities which means that flux integrations of curved boundaries is difficult.

PARAMETERS
- **surface**: FaceSelection | GeoSurface - The surface to integrate
- **argument**: Callable - A lambda function that takes an EHField object and returns a field component to integrate.
- **gqo**: int = 4 - The Gauss Quadrature integration order.

RETURNS complex | float

FUNCTION

em.MWField.**int_vol**()

Computes a volume inside a given volume.

To specify which field to integrate, the user must provide a function that takes a single argument (EHField object) and returns a specific field component. Example

```python
field.int_vol(selection, lambda eh: eh.Qv)
```

PARAMETERS
- **domain**: DomainSelection | GeoVolume - The domain to integrate over.
- **argument**: Callable - A lambda function that takes an EHField object and returns a field component to integrate.
- **gqo**: int = 4 - The Gauss Quadrature integration order.

RETURNS complex | float

FUNCTION

em.MWField.**int_line**()

Computes a line integral along a `Line` object.

Generally less practical than using the Line objects own line integral function. See `line_integral`.

To specify which field to integrate, the user must provide a function that takes a single argument (EHField object) and returns a specific field component. Example

python

```python
field.int_line(lineobj, lambda eh: eh.Hx)
```

PARAMETERS
- **line**: Line - The line object to integrate
- **argument**: Callable - A lambda function that takes an EHField object and returns a field component to integrate.

RETURNS `complex | float`

FUNCTION

em.MWField.**int_lumped_element**()

Performs an averaged line integration of either the E, H or S field along the lumped element to compute for example the lumped element voltage (E). Currently it always takes the integral dot dl, so integrating H does not return the current (not really a useful quantity).

PARAMETERS
- **lumped_element**: LumpedElement - The lumped element boundary condition object. See `LumpedElement`.
- **axis**: Axis | tuple[float, float, float] - An integration direction axis. This is not available in the LumpedElement itself.
- **quantity**: “E”, “H”, “S” = “E” - Only “E” is really meaningful.

RETURNS `complex | float`

FUNCTION

em.MWField.**boundary**()

Interpolates the field solution on the boundary specified by a surface. The E-field will be computed on the boundary triangle vertices.

The triangulation and data structure is passed to the EHField object allowing it to be used with the `add_field` function of the display class.

PARAMETERS
- **selection**: FaceSelection | GeoSurface - The boundary to compute the E-field on.

RETURNS `EHField`

FUNCTION

em.MWField.**current_boundary**()

Interpolates the field solution on the boundary specified by a surface. The E-field will be computed on the boundary triangle vertices. This function also computes the current density on the surface which is otherwise not incuded when calling `boundary`.

The triangulation and data structure is passed to the EHField object allowing it to be used with the `add_field` function of the display class.

PARAMETERS
- **selection**: FaceSelection - The boundary surface
- **eps**: float = 1e-06 - The offset distance along the face normal at which H is sampled on both sides.
- **n_samples**: int = 3 - The number of offsets (between 0.75 and 1.25 eps) averaged per side.

RETURNS `EHField`

FUNCTION

em.MWField.**cutplane**()

Computes the coordinates for a cut-plane through the simulation domain based on a specified grid step size. Then returns an EHField object (see `EHField`) with the field data and meta data.

The cutplane is either an XY plane, YZ plane or XZ plane.

PARAMETERS
- **ds**: float - The grid step size
- **x**: float | None = None - If provided an YZ-plane is generated at the given x-coordinate.
- **y**: float | None = None - If provided an XZ-plane is generated at the given y-coordinate.
- **z**: float | None = None - If provided an XY-plane is generated at the given z-coordinate.

RETURNS `EHField`

FUNCTION

em.MWField.**cutplane_normal**()

Computes the coordinates for a cut-plane through the simulation domain based on a specified grid step size. Then returns an EHField object (see `EHField`) with the field data and meta data.

This cutplane is defined in any arbitrary orientation.

PARAMETERS
- **point**: tuple[float, float, float] - The plane normal axis origin.
- **normal**: tuple[float, float, float] - The plane normal vector.
- **npoints**: int = 300 - THe number of points along eath plane vector.
- **usenan**: bool = True - If points outside the simulation domain should be NaN instead of 0.0.

RETURNS `EHField`

FUNCTION

em.MWField.**grid**()

Evaluates a 3D grid enclosing the entire simulation domain (or a restriction) and returns and EHField object with the required metadata.

You can either specify a grid step size or a total number of points in the 3D grid.

PARAMETERS
- **ds**: float | None = None - The grid step size in meters
- **N**: int = 10_000, The number of total grid points. Not exactly respeted.
- **usenan**: bool = True - If points outside the simulation domain should contain NaN instead of 0s.
- **x_range**: tuple[float, float] | None = None - An x-range restriction. Otherwise it uses the geometry bounds.
- **y_range**: tuple[float, float] | None = None - An z-range restriction. Otherwise it uses the geometry bounds.
- **z_range**: tuple[float, float] | None = None - An z-range restriction. Otherwise it uses the geometry bounds.

RETURNS `EHField`

FUNCTION

em.MWField.**farfield_2d**()

Computes the farfield radiation pattern in a single polar cutplane using the Stratton-Chu integrals and returns the data in an EHFieldFF object (see `EHFieldFF`).

PARAMETERS
- **ref_direction**: tuple[float, float, float] | Axis - The direction for angle=0
- **plane_normal**: tuple[float, float, float] | Axis - The rotation axis of the angular cutplane
- **faces**: FaceSelection | GeoSurface - The faces to integrate over
- **ang_range**: tuple[float, float] = (-180, 180) - The angular rage limits.
- **Npoints**: int = 201 - The number of angular points.
- **origin**: tuple[float, float, float] | None = None - The farfield origin.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes where Ex is E-symmetry in x=0.

RETURNS `EHFieldFF`

FUNCTION

em.MWField.**farfield_3d**()

Computes the farfield radiation pattern in a spherical 2D surface (θ, φ) using the Stratton-Chu integrals and returns the data in an EHFieldFF object (see `EHFieldFF`). Arrays are indexed `(theta, phi)` since 3.0.

PARAMETERS
- **faces**: FaceSelection | GeoSurface - The integration faces
- **thetas**: ndarray | None = None - The 1D array of theta values.
- **phis**: ndarray | None = None - A 1D array of phi values.
- **origin**: tuple[float, float, float] | None = None - The boundary normal alignment origin.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes where Ex is E-symmetry in x=0.

RETURNS `EHFieldFF`

FUNCTION

em.MWField.**farfield**()

Computes the farfield radiation pattern on the requested pherical coordinates using the Stratton-Chu integrals and returns the data in an EHFieldFF object (see `EHFieldFF`). Returns an `EHFieldFF` since 3.0 (was an `(E, H, Ptot)` tuple): use `.E`, `.H`.

PARAMETERS
- **theta**: ndarray - The theta angles (any shape)
- **phi**: ndarray - The phi angles (same shape as theta)
- **faces**: FaceSelection | GeoSurface - The integration faces
- **origin**: tuple[float, float, float] | None = None - The boundary normal alignment origin.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes where Ex is E-symmetry in x=0.

RETURNS `EHFieldFF`

FUNCTION

em.MWField.**farfield_3d_ports**()

Compute the 3D farfield of every port excited on its own, in one pass. Gives the same result as calling excite_port(p) and farfield_3d for every port, but builds the integration surface and the phase factors only once. The excitation of this field is left unchanged.

PARAMETERS
- **faces**: FaceSelection | GeoSurface - The integration faces
- **thetas**: ndarray | None = None - The 1D array of theta values.
- **phis**: ndarray | None = None - A 1D array of phi values.
- **origin**: tuple[float, float, float] | None = None - The boundary normal alignment origin.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes where Ex is E-symmetry in x=0.
- **ports**: list[int | float] | None = None - The port numbers.

RETURNS `dict[int | float, EHFieldFF]`

FUNCTION

em.MWField.**nearfield**()

Compute the E and H-field at arbitrary points in space from the fields on a closed surface (Stratton-Chu). This is the near-field counterpart of farfield(): the fields on the faces are radiated to the provided coordinates through the free space Green’s function, so the points do not have to be inside the mesh. The result is only valid for points outside the closed integration surface (in free space); points inside it ideally give zero field. A point exactly on a surface sample point is singular.

PARAMETERS
- **x**: ndarray | float - The X-coordinates
- **y**: ndarray | float - The Y-coordinates
- **z**: ndarray | float - The Z-coordinates
- **faces**: FaceSelection | GeoSurface - The integration faces
- **origin**: tuple[float, float, float] | None = None - The boundary normal alignment origin.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes where Ex is E-symmetry in x=0.
- **structure**: “none” | “grid1d” | “grid2d” | “grid3d” | “trisurf” | “unstructured” | DataStructure = “none” - The data structure of the coordinates, stored in the EHField for plotting (“grid1d”, “grid2d”, “grid3d”, “trisurf” or “unstructured”).

RETURNS `EHField`

FUNCTION

em.MWField.**trace_poynting_lines**()

Trace particle/field-line trajectories through the time-averaged Poynting vector field using an adaptive-step embedded RK4(5) (Dormand-Prince) integrator.

Plot the result with `add_particle_lines`.

PARAMETERS
- seed_points = None - Explicit start coordinates, either shape (3,N) or a tuple (xs, ys, zs) of 1D arrays. Takes priority over everything else.
- seed_surface = None - FaceSelection to seed particles from.
- seed_domain = None - DomainSelection to seed particles from.
- **density**: str = “medium” - ‘sparse’ (just nodes), ‘medium’ (+ edge centers), or ‘dense’ (+ triangle centers / tet centroids – everything). Only used for *unweighted* seeding (weight_by is None), with seed_surface or seed_domain.
- **weight_by**: “E” | “H” | “EH” | None = None - If set to ‘E’, ‘H’, or ‘EH’, seed points are instead importance-sampled from Gauss-quadrature points across the surface/domain, weighted by |E|, |H|, or |E|*|H| (times the local area/volume element) – i.e. more seeds land on triangles/tets with higher local field energy, with sub-triangle/sub-tet placement precision. Requires seed_surface or seed_domain, and an explicit n_particles.
- **quad_order**: int = 4 - Gauss quadrature order used to generate the candidate points when weight_by is set (higher = more candidate points per triangle/tet = finer sub-element resolution, at the cost of more field evaluations).
- **n_particles**: int | None = None - Cap on the number of seed points. Default None means “use every candidate point at the chosen density” for unweighted seeding – only when a limit is given does it randomly subsample down to that many points. Required (must not be None) when weight_by is set, since importance sampling needs an explicit target count. For the fallback random-bounding-box seeding (no seed source given at all), None falls back to 200 points.
- **max_steps**: int = 5000 - Hard cap on integrator iterations (safety stop).
- **ds_init**: float | None = None - The initial arc-length step size.
- **ds_min**: float | None = None - The minimum arc-length step size.
- **ds_max**: float | None = None - The maximum arc-length step size.
- **rtol**: float = 0.0001 - Relative local error tolerance for step control.
- **atol**: float = 1e-09 - Absolute local error tolerance for step control.
- **normalize**: bool = True - If True (recommended), integrate the unit tangent dr/ds = S/|S| (stable arc-length parameterization). If False, integrates the raw field as a literal velocity field (dr/dt = S).
- **direction**: int = 1 - +1 to trace along the Poynting vector, −1 to trace against it.
- **stagnation_eps**: float = 1e-12 - |S| below this is treated as a stagnation point (terminate).
- **safety**: float = 0.9 - Safety factor (<1) applied to the adaptive step-size update. Defaults to 0.9.
- **verbose**: bool = False - Print periodic progress.
- **dx**: float = 0.0 - Constant x-offset applied to all generated seed points. Defaults to 0.0.
- **dy**: float = 0.0 - Constant y-offset applied to all generated seed points. Defaults to 0.0.
- **dz**: float = 0.0 - Constant z-offset applied to all generated seed points. Defaults to 0.0.

RETURNS `list[ndarray]`

FUNCTION

em.MWField.**in_domain**()

Create an unstructured dataset inside some domain. Can only be used with vector plots

PARAMETERS
- **selection**: DomainSelection - The domain(s) in which to sample the field.
- **nodes**: bool = True - If the nodes should be included.
- **edges**: bool = False - If edge centers should be included.
- **tris**: bool = False - If triangle centers should be included.
- **tets**: bool = False - If tet centroids should be included.

RETURNS `EHField`

FUNCTION

em.MWField.**embed_external_component**()

Embed an external N-port network (Touchstone file) onto some of this field's ports, e.g. a matching network on an antenna feed. Embedded ports are kept but marked inactive, so port numbers never change and it can be called repeatedly. Exciting an embedded port raises `KeyError`.

PARAMETERS
- **touchstone_file**: str - The filename of the touchstone file to import
- **port_indices**: list[int | float] - The port numbers (smat_index) of this field to connect the touchstone file to. Must all currently be active (not already embedded by a previous call).

RETURNS None

PROPERTY

em.MWField.**cache**

Method-chaining property that reuses interpolation data for repeated evaluations at the same points, e.g. `field.cache.cutplane(...)` inside a loop over excitations.

RETURNS MWField

11.1 S-parameters (MWScalarNdim)

PROPERTY

em.**MWScalarNdim**

The N-dimensional scalar dataset returned by `data.scalar.grid`. All quantities are arrays over the swept variables and the frequency (`.freq`). Port properties: `.Z0`, `.beta`, … Use `.S(i, j)` for one S-parameter or `.Smat` for the full matrix.

python

```python
data = model.mw.run_sweep()
g = data.scalar.grid
em.plot.plot_sp(g.freq, [g.S(1, 1), g.S(2, 1)],
  labels=['S11', 'S21'])
g.export_touchstone('filter.s2p')
```

RETURNS MWScalarNdim

FUNCTION

em.MWScalarNdim.**S**()

Return the S-parameter S_{ij} as an array over all swept dimensions. Use `1.2` style indices for mode 2 of port 1.

PARAMETERS
- **i**: int | float - The receiving port (mode) index
- **j**: int | float - The exciting port (mode) index

RETURNS ndarray

PROPERTY

em.MWScalarNdim.**Smat**

The full S-matrix with the port dimensions last: shape (..., F, N, N). Embedded external ports are `NaN`.

RETURNS ndarray

FUNCTION

em.MWScalarNdim.**renormalize**()

Renormalize the S-parameters to a new reference impedance.

PARAMETERS
- **Z0ref**: ndarray | float | complex - The new reference impedance (per port or a single value)

RETURNS MWScalarNdim

FUNCTION

em.MWScalarNdim.**combine_ports**()

Combine ports p1 and p2 into a differential and common mode port respectively. The p1 index becomes the differential mode port The p2 index becomes the common mode port

PARAMETERS
- **p1**: int - The first port number (becomes the differential mode port)
- **p2**: int - The second port number (becomes the common mode port)
- **Z0renorm**: ndarray | float | complex | None = None - If given, first renormalize all ports to this reference impedance (and store it as the new Z0).

RETURNS MWScalarNdim

FUNCTION

em.MWScalarNdim.**embed_external_component**()

Embed an external N-port network (Touchstone file) onto some ports of this dataset, in place. The Touchstone data is renormalized to the Z0 of each port it connects to, and parameter sweeps are supported. Embedded ports stay in `Sp` / `Smat` as `NaN` and keep their numbers.

PARAMETERS
- **touchstone_file**: str - The filename of the touchstone file to import
- **port_indices**: list[int] - The port numbers of this dataset to connect the touchstone file to.

RETURNS MWScalarNdim

FUNCTION

em.MWScalarNdim.**dense_f**()

Specify a frequency subsample point density or provide a list of denser frequency points.

PARAMETERS
- **N**: int | None = None - The number of frequency points
- **frequencies**: list[float] | ndarray | None = None - A list of frequency points.

RETURNS ndarray

FUNCTION

em.MWScalarNdim.**model_S**()

Returns an S-parameter model object at a dense frequency range. This method fits a rational model through the datasets frequency points to determine a model for the linear system. If no frequency array is provided the .dense_f(NF) method should have been called. Two methods are available: 'vf' (Vector Fitting) and 'aaa' (AAA rational approximation of all ports jointly). 'auto' uses 'aaa' for data from run_adaptive_sweep, which is the model the adaptive sweep was converged on, and 'vf' otherwise.

PARAMETERS
- **i**: int - The first S-parameter index
- **j**: int - The second S-parameter index
- **freq**: ndarray | None = None - The frequency sample points.
- **Npoles**: int | "auto" = "auto" - VF only: The number of poles to use (approx 2x divice order).
- **inc_real**: bool = True - VF only: Wether to allow for a real-pole.
- **maxpoles**: int = 30 - VF only: the maximum number of poles.
- **minpoles**: int = 1 - VF only: the minimum number of poles.
- **method**: "auto" | "aaa" | "vf" = "auto" - The fitting method.
- **tol**: float | None = None - AAA only: The fit tolerance.

RETURNS ndarray

FUNCTION

em.MWScalarNdim.**model_Smat**()

Generates a full S-parameter matrix on the provided frequency points using a rational model.

PARAMETERS
- **frequencies**: ndarray | None = None - The sample frequencies
- **Npoles**: int = 10 - VF only: The number of poles to fit.
- **inc_real**: bool = True - VF only: Wether allow for a real pole.
- **method**: "auto" | "aaa" | "vf" = "auto" - The fitting method.
- **tol**: float | None = None - AAA only: The fit tolerance.

RETURNS ndarray

FUNCTION

em.MWScalarNdim.**export_touchstone**()

Export the S-parameter data to a touchstone file

PARAMETERS
- **filename**: str - The File name
- **Z0ref**: float | None = None - The reference impedance to normalize to.
- **format**: “RI” | “MA” | “DB” = “RI” - The dataformat used in the touchstone file.
- **custom_comments**: list[str] | None = None - list[str], optional. List of custom comment strings to add to the touchstone file header. Each string will be prefixed with “! “ automatically.
- **funit**: “Hz” | “KHz” | “MHz” | “GHz” = “GHz” - The frequency unit written to the file.
- **dense_freq**: ndarray | None = None - An optional dense interpolation frequency range

RETURNS None

FUNCTION

em.MWScalarNdim.**save_smatrix**()

Save an S-parameter matrix to a touchstone file.

PARAMETERS
- **filename**: str - The filename
- **Smatrix**: ndarray - The S-parameter matrix with shape (Nfreq, Nport, Nport)
- **frequencies**: ndarray - The frequencies with size (Nfreq,)
- **Z0ref**: float | None = None - An optional reference impedance to normalize to.
- **format**: “RI” | “MA” | “DB” = “RI” - The S-parameter format.
- **custom_comments**: list[str] | None = None - list[str], optional. List of custom comment strings to add to the touchstone file header. Each string will be prefixed with “! “ automatically.
- **funit**: “Hz” | “KHz” | “MHz” | “GHz” = “GHz” - The frequency unit written to the file.
- **Z0_override**: ndarray | None = None - Reference impedance array matching Smatrix’s port axis, for when it’s a subset of the full self.Z0 (e.g. active ports only). Defaults to self.Z0.

RETURNS None

11.2 Datasets

PROPERTY

MWData.field/scalar.**Datasets**

`data.field` and `data.scalar` are datasets of all solutions of a sweep. Select entries by their variables (`freq` and any parameter-sweep variable) or get everything on an N-dimensional grid with `.grid`.

```python
field = data.field.find(freq=10e9, width=2*mm)
for sc in data.scalar.iter(): ...
S11 = data.scalar.grid.S(1, 1)   # shape (n_width, n_freq)
```

RETURNS BaseDataset

FUNCTION

BaseDataset.**find**()

Returns the physics dataset that is closest to the constraint given by the variables.

PARAMETERS
- **variables

RETURNS T

FUNCTION

BaseDataset.**select**()

Returns the first physics dataset that satisfies the variable assignemnt.

PARAMETERS
- **variables

RETURNS T

FUNCTION

BaseDataset.**filter**()

Returns a list of all physics datasets that are valid for the given variable assignment

PARAMETERS
- **variables

RETURNS list[T]

FUNCTION

BaseDataset.**axis**()

Returns a sorted list of all variables for the given name

PARAMETERS
- **name**: str - The name of the variable axis

RETURNS ndarray

FUNCTION

BaseDataset.**iter**()

Iterate over all entries.

RETURNS Generator[T, None, None]

FUNCTION

BaseDataset.**slice_set**()

Reduces the DataSet object to a slice with the given range

PARAMETERS
- **i1**: int - Start index
- **i2**: int | None = None - final index.
- **sort_by**: str | None = None - Sort the slice by this attribute of the entries.

RETURNS BaseDataset[T, M]

FUNCTION

BaseDataset.**get_entry**()

Returns the physics dataset for the given index

PARAMETERS
- **index**: int - The index of the solution

RETURNS T

PROPERTY

BaseDataset.**grid**

Combine all scalar solutions into one N-dimensional `MWScalarNdim` object with one axis per swept variable.

RETURNS MWScalarNdim

FUNCTION

data.sim.**iterate**()

Iterate over all simulation states of a parameter sweep. Yields `DataEntry` objects (`.vars`, `.data`, `.reports`) since 3.0 (was `(vars, dict)` tuples).

RETURNS Generator[DataEntry, None, None]

FUNCTION

data.sim.**find**()

Returns the DataEntry closest to the provided parametric sweep setting.

PARAMETERS
- **vars

RETURNS DataEntry

11.3 EHField class

The EHField class contains all the different possible electric and magnetic fields (E,H,D,B,P,S,Sm,J,Q) and the coordinates where they where computed. It also contains metadata on the structure of the underlaying coordinates (2D grid, 3D grid, unstructured grid, triangulated surface etc.).

Technicall the EHfield and EHFieldFF classes are not part of EMerge but part of the shared `emsutil` library which is also used by Optycal.

All field attributes are: E, H, S, P, D, B, Ex, Ey, Ez, Hx, ... etc, Sm (time averaged Poynting vector), Js, Jv, normE, normB, normS, normSm etc.

FUNCTION

em.**EHField**()

A data container class containing the complete set of different fields at specified coordinates. Also contains metadata regarding the underlaying structure of the coordinates.

PARAMETERS Should not be created manually
RETURNS EHField

FUNCTION

em.EHField.**vector**()

Returns a FieldPlotData object to be used with the `add_field` function when creating a vector plot.

PARAMETERS
- **field**: “E” | “H” | “B” | “D” | “P” | “S” | “Sm” | “Jv” - The field to plot
- **metric**: “real” | “imag” | “complex” - Which metric of the complex field vlaue to plot.

RETURNS FieldPlotData

FUNCTION

em.EHField.**scalar**()

Returns a FieldPlotData object to be used with the `add_field` function when creating a scalar plot.

PARAMETERS
- **field**: “Ex” | “Ey” | “Hx” | “normE” etc. - The field to plot
- **metric**: “real” | “imag” | “complex” | “abs” - Which metric of the complex field vlaue to plot.

RETURNS FieldPlotData

FUNCTION

em.EHField.**int**()

PARAMETERS
- **field**: str | Callable
- **metric**: “abs” | “real” | “imag” | “complex” | “” = “”

RETURNS float | complex

The EHFieldFF class contains the following farfield properties: E, H, Ex, Ey, Ez, Hx, Hy, Hz, Etheta, Ephi, Htheta, Hphi, Erhcp, Elhcp, AR, gain, dir, RCS.

Some attributes allow for further sub attribute specifications as they return an object of type `FarFieldComponent` . This class itself has the attributes: x, y, z, theta, phi, lhcp, rhcp and norm.

FUNCTION

em.**FarFieldComponent**()

A data container class for farfield quantities.

PARAMETERS Should not be created by users.
RETURNS FarFieldComponent

FUNCTION

em.**EHFieldFF**()

A data container class containing the complete set of different farfield components at specified coordinates. Also contains metadata regarding the underlaying structure of the coordinates.

PARAMETERS Should not be created by users.
RETURNS EHFieldFF

PROPERTY

em.EHFieldFF.**ang**

The local 1D polar coordinate angle array. In arbitrary polar coordinate slices, each angle is represented by a pair of θ and φ coordinates. This ang axis always contains the angles relative to the local polar coordinate plane as defined by `farfield_2d`,

RETURNS np.ndarray

FUNCTION

em.EHFieldFF.**total_radiated_power_integral**()

Performs a power integral assuming that the farfield angles are a complete closed 2D set of points along the surface.

PARAMETERS
- **r**: float = 1.0 - The numerical radius to compute the farfiel dat. Does not impact the solution.
- **use**: “EH” - Which field definition to use. Defaults to “EH” only option.

RETURNS FieldPlotData

PROPERTY

em.EHFieldFF.**E**

Returns the E-farfield component. Can be used as following:

```python
field.farfield_3d().E.lhcp
```

RETURNS FarFieldComponent

PROPERTY

em.EHFieldFF.**H**

Returns the H-farfield component. Can be used as following:

```python
field.farfield_3d().H.norm
```

RETURNS FarFieldComponent

PROPERTY

em.EHFieldFF.**gain**

Returns the isotropic gain farfield component.

```python
field.farfield_3d().gain.theta
```

RETURNS FarFieldComponent

PROPERTY

em.EHFieldFF.**dir**

Returns the directivity farfield component.

```python
field.farfield_3d().dir.theta
```

RETURNS FarFieldComponent

FUNCTION

em.EHFieldFF.**surfplot**()

Returns the plot data (FieldPlotData) to pass to the the `add_field` function to plot farfield plots in 3D.

PARAMETERS
- **polarization**: str - A string that retrevies the farfield component to be plotted. You can use this as if you are selecting attributes. Example: `E` or `gain.lhcp` etc.
- **quantity**: “abs” | “real” | “imag” | “angle” - Which farfield component quantity to plot.
- **dB**: bool = False - Converts the quanity to decibels.
- **dBfloor**: float = −30.0 - The dB level to correspond to a 0-radius.
- **rmax**: float | None - The maximum radius to use for the greatest farfield quantity.
- **offset**: tuple[float, float, float] = (0,0,0) - The 3D origin for the farfield plot.

RETURNS FieldPlotData

12. Display

The following functions can be called on the PVDisplay class which is the PyVista interface class available under:

```python
my_model.display.
```

FUNCTION

PVDisplay.**add_anchors**()

Add a series of achor points to the plot window. See `Anchor`

PARAMETERS **anchors**: list[Anchor] - A list of anchor objects size: float = 1.0 - A sizing scaler to control how big the anchors are plotted

RETURNS None

FUNCTION

PVDisplay.**add_object**()

Adds an EMerge GeoObject or Selection of any dimension to the plot window.

PARAMETERS
- **obj**: GeoObject | Selection - The object or selection to add
- **mesh**: bool = False - If the mesh of the object should be added instead of a 3D render of the geometry
- **volume_mesh**: bool = True - If true, the volume mesh is plotted, otherwise only the surface meshes.
- **label**: bool = False - If a label is to be added with the object/ selectio name
- **label_text**: str | None = None - What text to add in case a label is added
- **texture**: str | None = None - Specify a texture name. Used when automatically plotting boundary conditions.
- **opacity**: float | None = None - The opacity of the object/ selection. Will take the lowest of this value and the objects own opacity
- **draw_line**: bool = True - If true, thicker lines outlining the geometry are plotted
- **pbr**: bool = True - If physics based rendering in PyVista should be allowed.
- **smooth_shading**: bool = False - If smooth shading should be used during physics based rendering. This renders spherical/ curved geometries better but the corners for boxes will look rounded.
- **selectable_as**: str | None = None - Makes the object selectable under a specific name.

RETURNS None

FUNCTION

PVDisplay.**add_objects**()

Add multiple objects with a certain opacity as a list of objects. Also accepts all `add_object` arguments.

PARAMETERS
- ***objects** - The objects to add
- **opacity**: float | None = None - The objects plot opacity.
- **minimize_opacity**: bool = True - Minimize the provided opacity with the material opacity.
- ****kwargs** - Passed on to add_object

RETURNS None

FUNCTION

PVDisplay.**populate**()

Add all geometries in the current model to the plot.

PARAMETERS
- **opacity**: float | None = None - The max opacity to use for all geometries. Defaults to 1.0.
- **minimize_opacity**: bool = True - If the minimum of the objects natural opacity and true opacity should be chosen.
- **skip_air**: bool = False - (bool, optional): Do not include air objects in the visualization (skips materials whose name lowercase equals “air”)
- **kwargs

RETURNS None

FUNCTION

PVDisplay.**add_scatter**()

Add scatter plot points to the 3D plot window.

PARAMETERS
- **xs**: np.ndarray - A numpy array of x-coordinates.
- **ys**: np.ndarray - A numpy array of y-coordinates.
- **zs**: np.ndarray - A numpy array of z-coordinates.

RETURNS None

FUNCTION

PVDisplay.**add_line**()

Add an EMerge Line object to the plot. See `Line` for more details.

PARAMETERS
- **line**: Line - The line object
- **width**: float = 3.0 - The line width
- **color**: str = “EMERGE-RED” - The line color

RETURNS None

FUNCTION

PVDisplay.**add_portmode**()

Add the port mode of a Port boundary condition object (any of the port boundary conditions) to the 3D plot.

PARAMETERS
- **port**: PortBC - The Port Boundary Condition
- **dv**: tuple[float, float, float] = (0,0,0) - A slight offset to give the port mode plot elements. Sometimes useful if the port mode is z-fighting with a nother plot.
- **XYZ**: tuple[np.ndarray,...] - A set coordinates where to evaluate the port mode quiver plot. By default it will pick triangle centers.
- **field**: “E” | “H” | “gradE” | “Ez” - Which port mode field componet to plot.
- **k0**: float | None = None - At which simulation phase constant to plot the mode. quasi-TEM modes can change shape.
- **cmap**: str | None = None - A colormap to use for the quiver plot
- **mode_number**: int | None = None - A selection of which mode number to plot. Default to the first mode
- **color**: str | None = None - A color string or hex code to use for the arrows.
- **Npoints**: Depricated

RETURNS None

FUNCTION

PVDisplay.**add_farfield3d**()

Add the results of a 3D farfield calculation to the plot window as a 3D farfield pattern.

The Farfield data objects can be generated as following:

```python
solution = model.mw.run_sweep()
field = solution.field.find(freq=...)
ff3d = field.farfield_3d(ff_boundary)
```

Choices for the farfield component are: Ex, Ey, Ez, Hx, Hy, Hz, Sx, Sy, Sz, Etheta, Ephi, Htheta, Hphi, Stheta, Sphi, Elhcp, Erhcp, AR, gain.norm. gain.Ex, gain.Ey, gain.Ez, gain.Etheta, gain.Ephi, gain.Erhcp, gain.Elhcp, normE, normH, normS.

PARAMETERS
- **farfield_obj**: EHFieldFF - The farfield data objects.
- **component**: “Ex”, “Elhcp”, “Hx”, “Htheta”, “gain.norm” = “normE” - The field component to plot.
- **quantity**: “abs” | “real” | “imag” | “angle” - Which quantity of the quantity to plot. Defaults to “abs”
- **dB**: bool = False - If the field should be converted to dB.
- **dBfloor**: float = −30. Which dB level to use for the 3D polar plot R=0.
- **rmax**: float | None = None - Which maximum radius in meters to use for the highest farfield plot magnitude.
- **offset**: tuple[float, float, float] - Where to put the farfield plot. Defaults to the origin.
- **opacity**: float | None = None - The opacity of the farfield plot.

RETURNS None

FUNCTION

PVDisplay.**add_particle_lines**()

Adds particle trajectories as smooth tubes with gradient coloring and directional arrows.

Use with the output of `trace_poynting_lines`.

PARAMETERS
- **lines**: list[ndarray] | ndarray - The trajectories, e.g. from MWField.trace_poynting_lines
- **tube_radius**: float = 0.0001 - The tube radius. Defaults to 0.0001.
- **cmap**: str = “coolwarm” - The colormap.
- **add_arrows**: bool = True - Add direction arrows along the lines.
- **num_arrows**: int = 8 - The number of arrows per line.
- **arrow_scale**: float = 0.003 - The arrow size. Defaults to 0.003.
- **show_scalar_bar**: bool = False - Show the scalar bar.

RETURNS None

FUNCTION

PVDisplay.**add_solution_error**()

Displays a per-tetrahedron error indicator.

PARAMETERS
- **error_value**: ndarray - Error values, one per tetrahedron. If tet_ids is None, this must have length self._mesh.n_tets and is assumed to be given in the same order as self._mesh.tets. If tet_ids is provided, error_value must have the same length as tet_ids, and only those tetrahedra are used.
- **tet_ids**: ndarray | None = None - The tetrahedron indices to plot (e.g. the tets with the largest error).
- **mode**: “volume” | “points” | “surface” = “volume” - How to render. “volume” does true volumetric rendering with transparency, so interior tets are actually visible through the exterior – this is almost always what you want for an error plot, since a plain opaque solid (“surface”) only ever shows you the outer boundary tets. “points” instead draws one sphere per tet at its centroid, sized AND colored by error, which avoids any transparency/occlusion ambiguity at the cost of losing the continuous solid shape.
- **cmap**: str | None = None - Colormap name.
- **log_scale**: bool = True - If True, colors on a log scale – error estimates commonly span many orders of magnitude and a linear scale usually washes out everything except the single worst tet.
- **opacity**: float | str = “linear” - For mode=“volume”, this is pyvista’s opacity TRANSFER FUNCTION (e.g. “linear”, “sigmoid”, or an array), not a single alpha value – it controls how strongly each error magnitude is allowed to occlude what’s behind it. For “points”/“surface” this is a plain 0-1 opacity.
- **show_scalar_bar**: bool = True - Whether to show the color legend.
- **clim**: tuple[float, float] | None = None - Manual color limits.
- **scalar_name**: str = “Error” - Label used for the scalar bar.
- **point_size_range**: tuple[float, float] | None = None - Only used for mode=“points”. Min/max sphere RADIUS in your model’s own length units (not fixed on-screen pixels) – the lowest-error tet gets the smaller sphere, the highest gets the larger.

RETURNS None

FUNCTION

PVDisplay.**add_backgroundfields**()

Draw the propagation direction and the E and H polarization of background (plane wave) fields.

PARAMETERS
- **fields**: list[BackgroundField] - The background fields
- **radius**: float - The distance from the origin at which the arrows are drawn

RETURNS None

FUNCTION

EMergeDisplay.**cbar**()

Method chaining function to specify the color bar details for the plot that will be added next. Example usage:

python
```
model.display.cbar(...).add_field()
```

Multiple plot with the same color bar name will share the same color bar. Final settings apply.

PARAMETERS
- **name**: str - The name for the color bar.
- **n_labels**: int = 5 - How many value labels to add
- **interactive**: bool = False -
- **clim**: tuple[float, float] - The color bar limits lowest and highest value.

RETURNS EMergeDisplay

FUNCTION

EMergeDisplay.**set_theme**()

Apply a color theme as EMergeTheme object.

PARAMETERS **theme**: EMerge**Theme** - The EMergeTheme object.

RETURNS None

FUNCTION

EMergeDisplay.**show**()

Tells the display to show generating a new window. May not get priority from the operating system.

PARAMETERS
- **screenshot**: str | None = None - If given, saves a screenshot to this file instead of opening a window.
- **off_screen**: bool = False - Render without opening a window.
- **zoom**: float = 1.0 - Camera zoom factor applied to the default view; values above 1 zoom in. Also applies to screenshots and orbit recordings. Defaults to 1.0.

RETURNS None

FUNCTION

EMergeDisplay.**orbit**()

Makes the camera orbit around the geometry when the display is shown.

PARAMETERS
- **n_frames**: int = 180 - Frames per full 360° revolution.
- **fps**: int = 30 - Frames per second.
- **axis**: “x” | “y” | “z” | tuple[float, float, float] = “z” - The world axis to orbit around.
- **record**: str | None = None - If given, renders off-screen and saves the orbit to this file (.gif, or a movie format such as .mp4) instead of opening a window.
- **loops**: int = 1 - Number of revolutions to record.
- **window_size**: tuple[int, int] = (1920, 1088) - Recording resolution in pixels (width, height), rounded to multiples of 16.
- **quality**: int = 10 - Movie encoding quality from 0 to 10 (ignored for GIF).

RETURNS EMergeDisplay

FUNCTION

EMergeDisplay.**explode**()

Explodes everything plotted inside the with-block along the direction of ds.

PARAMETERS
- **ds**: tuple[float, float, float] | ndarray = (0, 0, 0.004) - The displacement between two consecutive layers. Defaults to (0, 0, 0.004).
- **tol**: float | None = None - Items whose center lies within tol of the previous item along ds share a layer. Defaults to 0.1% of the spread of all items.

RETURNS Iterator[EMergeDisplay]

FUNCTION

EMergeDisplay.**view_x / view_y / view_z / view_iso**()

Set the camera to look along the x, y or z axis, or to an isometric view. Chain before `.show()`.

RETURNS None

FUNCTION

EMergeDisplay.**add_vector_cloud**()

Adds a 3D volumetric cloud of a vector field where the colour shows the direction axis and the opacity the magnitude.

PARAMETERS
- **X**: ndarray - A 3D Grid of X-values
- **Y**: ndarray - A 3D Grid of Y-values
- **Z**: ndarray - A 3D Grid of Z-values
- **vx**: ndarray - The X-component of the vector field (may be complex)
- **vy**: ndarray - The Y-component of the vector field (may be complex)
- **vz**: ndarray - The Z-component of the vector field (may be complex)
- **color_mapping**: tuple[str, str, str] = (“r”, “g”, “b”) - The colour for the x, y and z axis, from r, g, b (red, green, blue) and c, m, y (cyan, magenta, yellow), e.g. (“c”, “m”, “y”). The three colours must be independent: none can be a mix of the other two, so (“r”, “g”, “y”) is not allowed.
- **vivid**: bool = True - Make every direction equally bright instead of dimming in-between directions.
- **color_gamma**: float = 0.5 - Balance between axis and in-between colours. 1 maps the direction components directly, which makes axis colours dominate; lower values give in-between colours more room. 0.8 spreads the hue almost evenly over the angle. Defaults to 0.8.
- **scale**: “lin” | “log” = “lin” - Map |F| to opacity linearly or in dB.
- **clim**: tuple[float, float] | None = None - The |F| range (min, max) mapped from transparent to opaque.
- **clim_crop_factor**: float = 1.0 - Multiplier on the automatic upper limit; below 1 makes weaker fields more visible. Defaults to 1.0.
- **log_range_db**: float = 40.0 - Dynamic range shown with scale=‘log’.
- **opacity**: ndarray | None = None - A 256 point opacity curve over the magnitude.

RETURNS None

FUNCTION

EMergeDisplay.**animate**()

A method chaining function to tell the plot window that the next plot function should be animated (if possible).

Make sure that the field component you plot is “complex”. Otherwise all waves will be standing waves.

Example usage:

```python
model.display.animate().add_field()
```

PARAMETERS
- **Nsteps**: int = 35 - The number of steps to animate for a complete 360 degree phase cycle.
- **fps**: int = 25 - The number of frames per second for the animation.

RETURNS EMergeDisplay

FUNCTION

EMergeDisplay.**drawing_bw**()

Sets the drawing mode to black and white.

RETURNS EMergeDisplay

FUNCTION

EMergeDisplay.**save_vtk**()

Saves the current plot window elements to a VTK file.

PARAMETERS **base_path**: str - The path to store the file to.

RETURNS None

FUNCTION

EMergeDisplay.**add_scatter**()

Add scatter plot points to the 3D plot window.

PARAMETERS
- **xs**: np.ndarray - A numpy array of x-coordinates.
- **ys**: np.ndarray - A numpy array of y-coordinates.
- **zs**: np.ndarray - A numpy array of z-coordinates.

RETURNS None

FUNCTION

PVDisplay.**add_field**()

Add an generid FieldPlotData object to the plot window. These are created from datasets and contain all metadata to tell the display class how to display it.

Example of FieldPlotData creation:

```python
solution = sim.mw.run_sweep()
fieldata = solution.field.find(freq=1e9).grid(N=100_000)
```

PARAMETERS
- **field**: FieldPlotData - The field data object
- **scale**: “lin” | “log” | “symlog” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm and symlog applies a symmetric logarithm for more detail in wave animations.
- **cmap**: str | None - The colormap name to use
- **clim**: tuple[float, float] - The color bar limits to use.
- **opacity**: float | None = None - The opacity of the field.
- **voltype**: “cloud” | “contour” | “clip” - Which plot style to use for volumetric data (3D data arrays). “cloud” is a cloud plot. “contour” is a 3D contour (NLevels sets the number of contour isosurfaces), “clip” adds a cloud like plot with a movable clip plane.
- **clim_crop_factor**: float = 1.0 - A scaling factor for the color bar limits if defaults are chosen. 0.5 means that the color bar will reach to half the maximum detected value.
- **symmetrize**: bool = False - If true, the colorbar will always range from -vmax to +vmax where vmax is max(abs(values)). Will also automatically select a symmetric colormap if no value is provided.
- **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_surf**()

Add a 2D surface plot to the plot window.

PARAMETERS
- **x**: np.ndarray - a (N,M) array of X-coordinates.
- **y**: np.ndarray - a (N,M) array of Y-coordinates.
- **z**: np.ndarray - a (N,M) array of z-coordinates
- **field**: np.ndarray - a (N,M) field of scalar values for each coordinate.
- **scale**: “lin” | “log” | “symlog” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm and symlog applies a symmetric logarithm for more detail in wave animations.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **clim_crop_factor**: float = 1.0 - A scaling factor for the color bar limits if defaults are chosen. 0.5 means that the color bar will reach to half the maximum detected value.
  - **symmetrize**: bool = False - If true, the colorbar will always range from -vmax to +vmax where vmax is max(abs(values)). Will also automatically select a symmetric colormap if no value is provided.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_trisurf**()

Adds an unstructured trianglulated surface plot.

PARAMETERS
- **x**: np.ndarray - a (N,) array of X-coordinates.
- **y**: np.ndarray - a (N,) array of Y-coordinates.
- **z**: np.ndarray - a (N,) array of z-coordinates
- **field**: np.ndarray - a (N,) field of scalar values for each coordinate.
- **tris**: np.ndarray - a (3,M) array for the triangulation of the point set.
- **scale**: “lin” | “log” | “symlog” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm and symlog applies a symmetric logarithm for more detail in wave animations.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **clim_crop_factor**: float = 1.0 - A scaling factor for the color bar limits if defaults are chosen. 0.5 means that the color bar will reach to half the maximum detected value.
  - **symmetrize**: bool = False - If true, the colorbar will always range from -vmax to +vmax where vmax is max(abs(values)). Will also automatically select a symmetric colormap if no value is provided.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_clip_volume**()

Adds a 3D clip volume plot using 3D strutured data.

PARAMETERS
- **X**: np.ndarray - a (M,N,K) array of X-coordinates.
- **X**: np.ndarray - a (M,N,K) array of Y-coordinates.
- **X**: np.ndarray - a (M,N,K) array of z-coordinates
- **V**: np.ndarray - a (M,N,K) field of scalar values for each coordinate.
- **scale**: “lin” | “log” | “symlog” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm and symlog applies a symmetric logarithm for more detail in wave animations.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **clim_crop_factor**: float = 1.0 - A scaling factor for the color bar limits if defaults are chosen. 0.5 means that the color bar will reach to half the maximum detected value.
  - **symmetrize**: bool = False - If true, the colorbar will always range from -vmax to +vmax where vmax is max(abs(values)). Will also automatically select a symmetric colormap if no value is provided.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_contour**()

Adds a 3D contour volume plot using 3D strutured data.

PARAMETERS
- **X**: np.ndarray - a (M,N,K) array of X-coordinates.
- **X**: np.ndarray - a (M,N,K) array of Y-coordinates.
- **X**: np.ndarray - a (M,N,K) array of z-coordinates
- **V**: np.ndarray - a (M,N,K) field of scalar values for each coordinate.
- **Nlevels**: int = 5 - The number of contour plot isosurface levels.
- **scale**: “lin” | “log” | “symlog” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm and symlog applies a symmetric logarithm for more detail in wave animations.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **clim_crop_factor**: float = 1.0 - A scaling factor for the color bar limits if defaults are chosen. 0.5 means that the color bar will reach to half the maximum detected value.
  - **symmetrize**: bool = False - If true, the colorbar will always range from -vmax to +vmax where vmax is max(abs(values)). Will also automatically select a symmetric colormap if no value is provided.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_cloud**()

Adds a 3D cloud plot using 3D strutured data.

PARAMETERS
- **X**: np.ndarray - a (M,N,K) array of X-coordinates.
- **X**: np.ndarray - a (M,N,K) array of Y-coordinates.
- **X**: np.ndarray - a (M,N,K) array of z-coordinates
- **V**: np.ndarray - a (M,N,K) field of scalar values for each coordinate.
- **scale**: “lin” | “log” | “symlog” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm and symlog applies a symmetric logarithm for more detail in wave animations.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **clim_crop_factor**: float = 1.0 - A scaling factor for the color bar limits if defaults are chosen. 0.5 means that the color bar will reach to half the maximum detected value.
  - **symmetrize**: bool = False - If true, the colorbar will always range from -vmax to +vmax where vmax is max(abs(values)). Will also automatically select a symmetric colormap if no value is provided.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_quiver**()

Adds a quiver vector arrow plot to the plot window.

PARAMETERS
- **x**: np.ndarray - a (N,) array of X-coordinates.
- **y**: np.ndarray - a (N,) array of Y-coordinates.
- **z**: np.ndarray - a (N,) array of z-coordinates.
- **dx**: np.ndarray - a (N,) array of X-vector components.
- **dy**: np.ndarray - a (N,) array of Y-vector components.
- **dz**: np.ndarray - a (N,) array of z-vector components.
- **scale**: float = 1.0 - An extra size scaling to apply over the vector lengths.
- **scalemode**: “lin” | “log” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_streamline**()

Adds a streamline plot to the plot window using vector field like data.

PARAMETERS
- **x**: np.ndarray - a (N,) array of X-coordinates.
- **y**: np.ndarray - a (N,) array of Y-coordinates.
- **z**: np.ndarray - a (N,) array of z-coordinates.
- **dx**: np.ndarray - a (N,) array of X-vector components.
- **dy**: np.ndarray - a (N,) array of Y-vector components.
- **dz**: np.ndarray - a (N,) array of z-vector components.=
- **source_points**: np.ndarray | None = None - A (3,M) array of source point coordinates where to start streamlines from.
- **scale**: float = 1.0 - An extra size scaling to apply over the vector lengths.
- **scalemode**: “lin” | “log” = “lin” - A resaling to apply. Lin is default (no scaling) log applies a logarithm.
- **cmap**: str | None - The colormap name to use.
- **clim**: tuple[float, float] - The color bar limits to use.
  - **opacity**: float | None = None - The opacity of the field.
  - **_fieldname**: str - The name to use for the plotted field. Defaults to a name selected by the field data.

RETURNS None

FUNCTION

EMergeDisplay.**add_title**()

Adds a tile text to the plot

PARAMETERS
- **title**: str - The title string
- **color**: str - The color to use for the text.

RETURNS None

FUNCTION

EMergeDisplay.**add_text**()

Adds text to the plot window

PARAMETERS
- **text**: str - The texst string
- **color**: str - The color to use for the text.
- **position**: “lower_left” | “lower_right” | “upper_left” | “upper_right” | “lower_edge” | “upper_edge” | “right_edge” | “left_edge” - The position of the text.
- **abs_position**: tuple[float, float] | None - The absolute position in the viewport.

RETURNS None

12.1 2D Plotting

`em.plot` re-exports the plotting functions of `emsutil`.

FUNCTION

em.plot.**plot_sp**()

Plot S-parameters in dB and phase

PARAMETERS
- **f**: ndarray | list[ndarray] - Frequency vector or list of frequencies
- **S**: list[ndarray] | ndarray - S-parameters to plot (list or single array)
- dblim = [−40, 5] - Decibel y-axis limit.
- xunit = “GHz” - Frequency unit.
- **levelindicator**: int | float | None = None - Level at which annotation arrows will be added.
- noise_floor = −150 - Artificial random noise floor level.
- **fill_areas**: list[tuple] | None = None - Regions to fill (fmin, fmax).
- **spec_area**: list[tuple[float, ...]] | None = None - Optional specification boxes to draw, as (fmin, fmax, ymin, ymax) tuples
- unwrap_phase = False - If or not to unwrap the phase data.
- **logx**: bool = False - Whether to use logarithmic frequency axes.
- **labels**: list[str] | None = None - A lists of labels to use.
- **linestyles**: list[str] | None = None - The linestyle to use (list or single string).
- **colorcycle**: list[int] | None = None - A list of colors to use.
- **filename**: str | None = None - The filename (will automatically save).
- **show_plot**: bool = True - If or not to show the resulting plot.
- **figdata**: tuple | None = None

RETURNS tuple[plt.Figure, plt.Axes, plt.Axes]

FUNCTION

em.plot.**smith**()

Plot S-parameter traces on a Smith chart with optional per-trace styling and sparse frequency annotations (e.g., labeled by frequency).

PARAMETERS
- **S**: ndarray | Sequence[ndarray] - One or more 1D complex arrays of reflection coefficients (Γ) to plot (each shaped like (N,)).
- **f**: Optional[ndarray | Sequence[ndarray]] = None - Frequency vector(s) aligned with `s` for sparse on-curve labels; provide a single array for all traces or one array per trace.
- **colors**: Optional[Union[str, Sequence[Optional[str]]]] = None - Color for all traces or a sequence of per-trace colors.
- **markers**: Optional[Union[str, Sequence[str]]] = None - Marker style for all traces or per-trace markers.
- **labels**: Optional[Union[str, Sequence[str]]] = None - Legend label for all traces or a sequence of per-trace labels. If omitted, no legend is shown.
- **title**: Optional[str] = None - Axes title.
- **linewidth**: Optional[Union[float, Sequence[Optional[float]]]] = None - Line width for all traces or per-trace widths.
- **n_flabels**: int = 8 - Approximate number of frequency labels to place per trace (set 0 to disable, even if `f` is provided).
- **funit**: str = “GHz” - Frequency unit used to scale/format labels. One of {‘Hz’,‘kHz’,‘MHz’,‘GHz’,‘THz’} (case-insensitive).

RETURNS None

FUNCTION

em.plot.**plot**()

Plot one or more y-series against a common x-axis, with extensive formatting options.

PARAMETERS
- **x**: ndarray
- **y**: Union[ndarray, Sequence[ndarray]]
- **grid**: bool = True
- **labels**: Optional[List[str]] = None
- **xlabel**: str = “x”
- **ylabel**: str = “y”
- **linestyles**: Union[str, List[str]] = “-”
- **linewidth**: float = 2.0
- **markers**: Optional[Union[str, List[Optional[str]]]] = None
- **logx**: bool = False
- **logy**: bool = False
- **transformation**: Optional[Callable[[ndarray], ndarray]] = None
- **xlim**: Optional[Tuple[float, float]] = None
- **ylim**: Optional[Tuple[float, float]] = None
- **title**: Optional[str] = None

RETURNS None

FUNCTION

em.plot.**plot_vswr**()

Plot S-parameters in VSWR

PARAMETERS
- **f**: ndarray | list[ndarray] - Frequency vector or list of frequencies
- **S**: list[ndarray] | ndarray - S-parameters to plot (list or single array)
- swrlim = [1, 5] - VSWR y-axis limit.
- xunit = “GHz” - Frequency unit.
- **levelindicator**: int | float | None = None - Level at which annotation arrows will be added.
- **fill_areas**: list[tuple] | None = None - Regions to fill (fmin, fmax).
- **spec_area**: list[tuple[float, ...]] | None = None - Optional specification boxes to draw, as (fmin, fmax, ymin, ymax) tuples
- **labels**: list[str] | None = None - A lists of labels to use.
- **linestyles**: list[str] | None = None - The linestyle to use (list or single string).
- **colorcycle**: list[int] | None = None - A list of colors to use.
- **filename**: str | None = None - The filename (will automatically save).
- **show_plot**: bool = True - If or not to show the resulting plot.
- **figdata**: tuple | None = None

RETURNS tuple[plt.Figure, plt.Axes, plt.Axes]

FUNCTION

em.plot.**plot_ff**()

Far-field rectangular plot of E-field magnitude vs angle.

PARAMETERS
- **theta**: ndarray | list[ndarray]
- **E**: Union[ndarray, Sequence[ndarray]]
- **grid**: bool = True
- **dB**: bool = False
- **labels**: Optional[List[str]] = None
- **xlabel**: str = “Theta (rad)”
- **ylabel**: str = “”
- **linestyles**: Union[str, List[str]] = “-”
- **linewidth**: float = 2.0
- **markers**: Optional[Union[str, List[Optional[str]]]] = None
- **xlim**: Optional[Tuple[float, float]] = None
- **ylim**: Optional[Tuple[float, float]] = None
- **title**: Optional[str] = None

RETURNS None

FUNCTION

em.plot.**plot_ff_polar**()

Far-field polar plot of E-field magnitude vs angle.

PARAMETERS
- **theta**: ndarray
- **E**: Union[ndarray, Sequence[ndarray]]
- **dB**: bool = False
- **dBfloor**: float = −30
- **labels**: Optional[List[str]] = None
- **linestyles**: Union[str, List[str]] = “-”
- **linewidth**: float = 2.0
- **markers**: Optional[Union[str, List[Optional[str]]]] = None
- **zero_location**: str = “N”
- **clockwise**: bool = False
- **rlabel_angle**: float = 45
- **rlim**: tuple[float, float] | None = None
- **title**: Optional[str] = None

RETURNS None

13. Import and Export

FUNCTION

emerge.read.**FileBasedPCB**()

A `PCB` that builds its layers from Gerber/Excellon files. Requires `pip install emerge[cad]`. Moved from `emerge.beta.gerber` in 3.0.

PARAMETERS
- (same arguments as `PCB`)

RETURNS FileBasedPCB

RAISES ImportError if the cad extra is missing

FUNCTION

FileBasedPCB.**layer_from_file**()

Create a layer from a Gerber file.

PARAMETERS
- **layer**: int - The layer number.
- **filename**: str - The path to the Gerber file.
- **res_mm**: float = 0.01 - The geometric resolution in mm (arc chord tolerance and minimum edge length during simplification). Defaults to 0.01.
- **curved**: bool = False - Build the layer from exact lines and circular arcs instead of polygons. Round pads, track ends and arcs are then modelled exactly and no simplification is needed. n_circ_segments, min_incl_angle, min_remove_angle and simplify are ignored (except for the few shapes that fall back to polygons).

RETURNS GeoSurface | GeoVolume

FUNCTION

FileBasedPCB.**vias_from_file**()

Add vias from an Excellon file.

PARAMETERS
- **filename**: str - The path to the Excellon file.
- **layer1**: int = 0 - The bottom layer index.
- **layer2**: int = −1 - The top layer index.
- **Nsections**: int = 6 - The number of sections for the vias. Defaults to 6. z1, z2: Deprecated and ignored, use layer1 and layer2.
- **z1**: float | None = None
- **z2**: float | None = None

RETURNS None

FUNCTION

emerge.read.**import_dxf**()

Imports a DXF file as a `PCB` object. Moved from `emerge.beta.dxf` in 3.0.

PARAMETERS
- **filename**: str - The DXF Filename
- **material**: Material - The material to use for the PCB
- **thickness**: float | None = None - The thickness to use for the PCB.
- **unit**: float | None = None - The pcb unit.
- **cs**: CoordinateSystem | None = GCS - The coordinate system.
- **trace_material**: Material = PEC - The material for traces.
- **split**: bool = True - If a loop splitting algorithm is to be used.

RETURNS PCB

FUNCTION

emerge.read.**GDSFile**()

Imports a GDSII file as EMerge geometry.

PARAMETERS
- **filename**: str | Path - Path to the .gds file.
- **layers**: dict[LayerKey, GDSLayer] | None = None - Maps a GDS layer number (any datatype) or a (layer, datatype) tuple to a GDSLayer. Layers that are not mapped are skipped. With None, the file is only parsed so you can inspect it (see .found_layers and .summary()).
- **cell**: str | None = None - Cell to import.
- **cs**: CoordinateSystem = GCS - Coordinate system the layout is placed in. Layers extrude along its z-axis.
- **scale**: float = 1.0 - Extra scale factor applied to all coordinates (GDS units are already converted to meters).

RETURNS GDSFile

FUNCTION

emerge.read.**GDSFromXML**()

Imports a GDSII layout using a gds2palace stack-up XML file for all layer and material data.

PARAMETERS
- **xml**: str | Path - The stack-up XML file.
- **gds**: str | Path - The GDSII file.
- **via_merge**: float | None = None - Merge vias of every via-type layer whose centers are closer than this distance (physical meters) into solid blocks/walls. None keeps every via.
- **variables**: dict[str, float | str] | None = None - Overrides for the XML’s <Variables>.
- **size_scale**: float = 1.0 - Modelling scale factor. Must match Simulation.set_scale_factor(...).
- **cell**: str | None = None - Cell to import.
- **cs**: CoordinateSystem = GCS - Coordinate system the layout is placed in.

RETURNS GDSFromXML

FUNCTION

emerge.read.**ODBImport**()

Load an ODB++ PCB design and turn it into `emerge` CAD geometry.

PARAMETERS
- **filename**: str - path to the ODB++ product model directory.
- **pcb_material**: em.Material - `emerge` material for the dielectric slabs.
- **stack_thickness**: list[float] | None = None - per-structural-layer thickness override, forwarded to `PCBView(thickness_stack=...)` – see that class for the exact top-to-bottom ordering.
- **reverse_stack**: bool = False - Z-flip the whole board (see `PCBView(reverse=...)`).
- **copper_placement**: “bottom” | “center” | “top” = “center” - where a copper layer’s thickness sits relative to its neighboring dielectrics – see `PCBView`’s own docs.

RETURNS ODBImport

FUNCTION

em.**TouchstoneData**()

A touchstone file importer for S-parameter data.“

PARAMETERS
- **filename**: str - The filename of the touchstone file.
- **n_ports**: int = None - The number of ports.
- **ignore_extension**: bool = False - Ignore the file extension.
- **interp_type**: str = “cubic” - The interpolation type.

RETURNS TouchstoneData

FUNCTION

emerge.write.**export_dxf**()

Exports all 2D Geometries on a single or multiple z-heights to a DXF file.

PARAMETERS
- **simulation**: Simulation - Your simulation object.
- **filename**: str - The filename to save the DXF as.
- **z_height**: float | list[float] - The z-height to export the 2D geometries at.
- **selection**: list[Selection | GeoObject] = None

RETURNS None

FUNCTION

emerge.write.**pcb_to_dxf**()

Exports all PCB traces to a DXF file.

PARAMETERS
- **pcb**: PCB
- **filename**: str - The filename to save the DXF as.

RETURNS None

FUNCTION

emerge.optycal.**surface_model**()

Generate a dataset that Optycal can interpret to form a surface model for Near and Farfield calculations

PARAMETERS
- **mwdata**: MWData - The simulation data set.
- **farfield_boundary**: FaceSelection | GeoSurface | None = None - The surface whose fields form the model.

RETURNS dict

FUNCTION

emerge.optycal.**farfield_model**()

Generate a dataset that Optycal can interpret to form a farfield model for Near and Farfield calculations

PARAMETERS
- **mwdata**: MWData - The simulation data set.
- **farfield_boundary**: FaceSelection | GeoSurface | None = None - The surface used for the near to far field transformation.
- **dangle_deg**: float = 1.0 - The angular resolution of the far field grid in degrees. Defaults to 1.0.
- **syms**: list[“Ex” | “Ey” | “Ez” | “Hx” | “Hy” | “Hz”] | None = None - E and H-plane symmetry planes of the simulation domain, where Ex is E-symmetry in x=0.

RETURNS dict

14. Macros and Waveguide Assemblies

FUNCTION

em.macro.**CoaxAssy**()

A coaxial line along the z-axis of cs, from z = 0 to z = length. Each end of the line is one of: ‘none’: the pin, dielectric and shield end flush. ‘pin’: the pin sticks out pin_length beyond the end, e.g. as a probe. ‘port’: a wave port on the end face of the dielectric. Behind it is a void of void_length, closed by the shield, which is extended by void_length + shield_thickness.

Attributes: `.pin`, `.diel`, `.shield` and, for port ends, `.start_port` / `.end_port`.

python

```python
coax = em.macro.CoaxAssy(0.5*mm, 10*mm, em.lib.DIEL_PTFE, em.ZAX.construct_cs(0, 0, 0),
                         Z0=50, start='port', start_port_nr=1, end='pin', pin_length=3*mm)
```

PARAMETERS
- **rin**: float - The radius of the inner conductor (pin)
- **length**: float - The length of the dielectric
- **dielectric**: em.Material - The material between the conductors
- **cs**: em.CoordinateSystem - The coordinate system; the line starts in its origin, along its z-axis
- **rout**: float | None = None - The inner radius of the shield. Give rout or Z0.
- **Z0**: float | None = None - The characteristic impedance, from which rout is computed.
- **conductor**: em.Material = em.lib.PEC - The material of the pin and shield. Defaults to em.lib.PEC.
- **shield_thickness**: float = 0.001 - The wall thickness of the shield. Defaults to 0.001.
- **Nsections**: int | None = None - Build the cylinders as prisms with this many sides.
- **start**: “none” | “pin” | “port” = “none” - End type at z = 0: `'none'`, `'pin'` or `'port'`
- **end**: “none” | “pin” | “port” = “none” - End type at z = length
- **pin_length**: float | None = None - How far the pin sticks out at a ‘pin’ end.
- **void_length**: float = 0.0005 - The length of the void behind a ‘port’ end. Defaults to 0.0005.
- **start_port_nr**: int | None = None - The port number of a ‘port’ start.
- **end_port_nr**: int | None = None - The port number of a ‘port’ end.

RETURNS CoaxAssy

FUNCTION

em.macro.**SMAConnector**()

A model of a PCB edge SMA connector. Call `.build()` to create the geometry.

PARAMETERS
- **thpcb**: float - The PCB thickness
- **cs**: em.CoordinateSystem - The coordinate system of the connector
- **Nsections**: int = 12 - Polygon sections of the round parts

RETURNS SMAConnector

FUNCTION

em.macro.**vivaldi_taper**()

PARAMETERS
- **length**: float
- **gap**: float
- **opening**: float
- **curve_coefficient**: float = 200.0
- **mirrored**: bool = True
- **dilation**: float = 0.0
- **cs**: CoordinateSystem = GCS

RETURNS GeoSurface

FUNCTION

em.macro.**design_yagi**()

Compute Yagi-Uda element lengths and spacings for a given frequency.

Variants: `design_yagi_dl6wu(frequency_hz, num_elements, ...)` (DL6WU long-boom design) and `design_yagi_tapered(frequency_hz, num_elements, ...)`. The returned `YagiDesign` has a `.summary()`.

PARAMETERS
- **frequency_hz**: float
- **num_elements**: int | None = None
- **boom_length_wl**: float | None = None
- **element_diameter_m**: float | None = None
- **boom_material**: str = "dielectric"
- **driven_element_type**: str = "dipole"

RETURNS `YagiDesign`

FUNCTION

em.wg.**WGConfig**()

The configuration shared by the components of a waveguide assembly: the inner dimensions (or a `WGStandard` such as `em.wg.WGStandard.WR_90`) and optional wall geometry.

PARAMETERS
- **dims**: tuple[float, float] | WGStandard - (a, b) in mm, or a WGStandard
- **model_walls**: bool = False - Also build the metal walls
- **wall_thickenss_mm**: float = 1.0 - Wall thickness in mm
- **material**: em.Material = em.lib.PEC - Wall material
- **coating**: em.Material | None = None - Optional coating material
- **coating_thickenss_um**: float = 1.0 - Coating thickness in µm

RETURNS `WGConfig`

FUNCTION

em.wg.**WGRect**()

A straight rectangular waveguide section. Like all components it has `front` and `back` anchors; `.place(other.back)` connects it to another component.

PARAMETERS
- **length**: float - The length
- **config**: WGConfig - The WGConfig

RETURNS `WGRect`

FUNCTION

em.wg.**WGETurnRound / WGHTurnRound / WGETurnMiter / WGHTurnMiter**()

E-plane and H-plane waveguide bends with rounded or mitered corners at any angle from −90 to 90 degrees.

```python
cfg = em.wg.WGConfig(em.wg.WGStandard.WR_90)
s1 = em.wg.WGRect(20*mm, cfg)
b1 = em.wg.WGHTurnRound(cfg,
angle_deg=90).front.place(s1.back)
```

PARAMETERS
- **config**: WGConfig - The waveguide configuration.
- **ratio**: float = 1.2 - The outer corner radius divided by w. A miter cuts the corner between the same points as the fillet. Defaults to 1.2.
- **angle_deg**: float = 90 - The bend angle in degrees, between −90 and 90 and not 0. A positive angle bends towards +Y (E-plane) or +X (H-plane).
- **inside_radius**: float = 0.0 - The inner corner radius in meters, 0 for a sharp corner.
- **lead**: float | None = None - The length of a straight section in meters added to both legs before the bend. Must be > 0 for ratio >= 1.

RETURNS `WGTurnGeneric`

FUNCTION

em.wg component.**place**()

Move a waveguide component so that its active anchor (`.front` or `.back`) lands on the anchor of another component.

PARAMETERS
- **anchor**: Anchor | component - The anchor (or component with an active anchor) to connect to
- **flipx**: bool = False - Mirror the target anchor in x
- **flipy**: bool = False - Mirror the target anchor in y
- **flipz**: bool = False - Mirror the target anchor in z

RETURNS `component`

RAISES `ValueError if no active anchor is set`