**Data Loading and Fitting Process**

The full process is demonstrated using Teflon as an example.

1. Create an instance of the Sproc class. When creating the object, you can provide the sample thickness and cutoff wavelength. These values can also be updated later by accessing the object properties L and lambda\_c.

Create Sproc instance named tef:

tef = Sproc(1.92e-3,0.04572);

Change sample thickness and/or cutoff wavelength:

tef.L = 2.7e-3;

tef.lambda\_c = 0.041;

1. Load a raw data file to the object. If the file is not in the working directory (folder in which you are working in Matlab), you must specify the path to the file (i.e ‘path/filename’). If there is a header in the file, add the name-value pair ‘HeaderLines’, n to the function call.

No header:

tef.load(data/'X\_band\_teflon\_notape.txt')

With header:

tef.load(data/'X\_band\_teflon\_notape.txt',’HeaderLines’,8)

* 1. Once the file is loaded, you can plot the S parameters.

tef.plot\_Sij

1. Perform NRW or NNI extraction. You can specify the branch using the name-value pair ‘BaseBranch’, p (the function defaults to the principal branch, i.e. p=0). The function will automatically look for discontinuities in the permittivity and increment the branch at the frequencies at which those discontinuities appear. You can disable this by using the name-value pair ‘IncrementBranch’, false.

Perform NRW w/branch 0:

tef.extract('nrw')

Perform NNI w/branch 0:

tef.extract('nni')

Perform NRW w/branch -1:

tef.extract('nrw',’BaseBranch,-1)

Perform NRW w/branch -1 and branch increment disabled:

tef.extract('nrw',’BaseBranch,-1,’IncrementBranch’,false)

* 1. Once the extraction is complete, you can plot the extracted permittivity and permeability.

tef.plot\_PP

1. Fit a Laurent model to the data (for magnetic materials). This uses a slightly different model formulation than the one described by Baker-Jarvis et al., but provides a very similar line shape that still obeys the Kramer-Kronig relations. The model approximates permittivity and permeability with truncated Laurent series:

where is the relative permittivity, is the relative permeability, is the imaginary number, and is the angular frequency. The A, B, and D terms are constants that must be determined by fitting the model to the data. The are poles, which may be real or complex, but must lie in the left half plane (the real part must be negative). The and terms represent 1st-order poles, while the and terms represent 2nd-order poles. Complex B terms must occur in complex conjugate pairs. The (for k>0) are residues indicating the contribution of each pole; if is complex, the corresponding must also be complex, and if is real, the corresponding must also be real.

* 1. Perform initial fit of 1st-order poles to permittivity and permeability using 1-10 poles and display results. A rule of thumb for selecting the appropriate number of poles is to use at least two poles per peak in the function that you want to fit. Another general guideline is to use as few poles as possible to obtain an acceptable starting fit. The r2 values which are reported in the table output can be helpful for determining the appropriate number of poles, but are significantly influenced by noise in the data; noisier data will result in lower r2 scores for appropriate fits. Thus, the best way to choose the appropriate number of poles is to review the table output and then plot several initial fits and compare.

tef.lm\_initialfit

* 1. Plot initial fits to aid in selection of appropriate number of 1st-order poles.

Plot initial fit of mu with one 1st-order pole:

tef.lm\_plot\_initfit('mu',1)

Plot initial fit of epsilon with 2 1st-order poles:

tef.lm\_plot\_initfit('eps',2)

* 1. Get initial parameter estimates using chosen number of 1st- and 2nd-order poles. The number of 2nd-order poles has much less influence on the final fit than the number of 1st-order poles. The number of 2nd-order poles defaults to 1, but can be set using the name-value pair ‘NPoles2’, n. Generally, one 2nd-order pole works well, but a second 2nd-order pole may improve fit quality in some cases. Increasing the number of 2nd-order poles beyond 2 is generally not beneficial.

Get initial parameters using one 1st-order pole for mu, one 1st-order pole for epsilon, and one 2nd-order pole for each:

tef.lm\_estimate\_x0('MuNPoles1',1,'EpsNPoles1',1)

Get initial parameters using one 1st-order pole for mu, three 1st-order poles for epsilon, and two 2nd-order poles for each:

tef.lm\_estimate\_x0('MuNPoles1',1,'EpsNPoles1',3,’NPoles2’,2)

* 1. Optimize the model by performing a least-squares fit to the measured S parameters. This uses the initial parameter estimates obtained in the previous step as a starting point. The initial parameter estimates are very important to obtaining a good solution. Generally, multiple local minima of the objective function exist, and the choice of appropriate starting parameters is crucial to obtaining the desired (global) minimum. By default, the optimization algorithm will launch the optimization from 10 different initial parameter sets to improve the probability of converging to the desired solution. This can be turned off by passing the name-value pair ‘MultiStart’, false. The number of different initial points from which the optimization is started can be adjusted with the name-value pair ‘MultiStartPoints’, n. By default, the parameters for the first-order poles are left identical for different starting points (i.e., only the 2nd-order poles are varied). The first-order poles can be varied by setting the name-value pair ‘ShiftFOPoles’, true; this may be beneficial for fitting large numbers of 1st-order poles, i.e. 5 or more total 1st-order poles. The maximum number of function evaluations may be adjusted by passing the name-value pair ‘MaxFEval’, n; it may be necessary to increase this threshold to allow optimizations to converge. A final optimization option is the number of complex conjugate pairs in the 2nd-order poles, which may be set by passing ‘NCCP2’, n. This defaults to 0, which works well in most cases, but may be adjusted as a last resort if acceptable fits are not obtained. Also note that due to the branch ambiguity, there are multiple sets of permittivity and permeability that yield the same S parameters. In order to converge to the desired branch, it is important to obtain initial parameter estimates based on the desired branch (i.e., perform the NRW or NNI extraction with the desired branch and then perform the initial fit to the mu and epsilon extracted for that branch).

Optimize with default options:

tef.lm\_lsqfit

Optimize with MultiStart turned off (only start from the initial parameters obtained in step c):

tef.lm\_lsqfit(‘MultiStart’,false)

Optimize with 20 MultiStart points with varying 1st-order poles, allowing up to 5000 function evaluations per run:

tef.lm\_lsqfit(‘MultiStartPoints’,20,’ShiftFOPoles’,true,’MaxFEval’,5000)

Optimize with 1 complex conjugate pair in 2nd-order poles:

tef.lm\_lsqfit(‘NCCP2’,1)

* 1. Once the optimization is complete, you can plot the optimized fit of the S parameters, and the optimized mu and epsilon.

Plot S parameter fit:

tef.lm\_plot\_Sfit

Plot optimized mu and epsilon vs. extracted values:

tef.lm\_plot\_PPfit

* 1. The model results can finally be saved to a file by running lm\_output and passing the name-value pair ‘SaveFile’, ‘filename’. If you want to save in a different folder than the working directory, you must specify the path to the file. If you do not specify SaveFile, the function simply outputs the model results to a variable.

Output results to variable out:

out = tef.lm\_output;

Save to file in fit\_results folder:

tef.lm\_output('SaveFile','fit\_results/lmOut\_X\_band\_teflon\_notape.txt');

1. Fit a dielectric model to the data (dielectric materials, assumes that permeability is one). This uses the model described by Baker-Jarvis et al.:

Here, the C, z, and p terms are constants that must be optimized. The are zeros, which may be real or complex. The are poles, which must be real and negative. C is allowed to be complex. Note that in many cases, the Laurent model may yield a better fit of data from dielectric materials. Since the dielectric model forces mu to unity, it may produce odd results for epsilon if the S parameter data is not highly accurate. Before using the dielectric model, perform the NRW extraction of your data and check if the extracted permeability is close to one (i.e. real part is one, imaginary part is zero) across the entire frequency range measured. If it is not, the dielectric model will likely give inaccurate results.

* 1. Get initial parameter estimates using a chosen number of poles to fit epsilon. By default, the algorithm uses two poles; this may be changed by passing the name-value pair ‘NPoles’, n. The specified number of poles will be uniformly (linearly) distributed across the frequency range. Alternatively, you may specify pole frequencies directly using the name-value pair ‘PoleFreq’, [frequencies].

Get initial parameters for 3 uniformly distributed poles:

tef.de\_estimate\_x0(‘NPoles',3)

Get initial parameters for 2 poles at 5 GHz and 10 GHz:

tef.de\_estimate\_x0('PoleFreq',[5e9 1e10])

* 1. Optimize the model by performing a least-squares fit to the S-parameter data. Many of the same considerations apply to this optimization as to the Laurent model optimization. The optimization options MultiStart, MultiStartPoints, and MaxFEval are also available for the dielectric optimization and have the same meanings and default values.

Optimize with default options:

tef.de\_lsqfit

* 1. Once the optimization is complete, you can plot the optimized fit of the S parameters, and the optimized mu and epsilon (note that mu is always one for the dielectric model).

tef.de\_plot\_Sfit

tef.de\_plot\_PPfit

* 1. Finally, the model results can saved to a file or stored in a variable using de\_output. This function operates in exactly the same way as lm\_output. If the name-value pair ‘SaveFile’, filename is passed, the result is saved to file; otherwise it is stored in a variable.

tef.de\_output('SaveFile','fit\_results/deOut\_X\_band\_teflon\_notape.txt');

Summarized Documentation

Full documentation is available in the code. Open the .m file for the specific function or class you are interested in. There should be a commented block of text describing the function’s use and input arguments directly below the first line of code.

**Classes**

* Sproc: class for S parameter processing. Use to load and store data, extract via NRW/NNI, and fit Laurent model and/or dielectric model
  + Properties (each property can be accessed via obj.propertyname)
    - L : sample thickness
    - lambda\_c: cutoff wavelength
    - Sdata: measured S parameters
    - PPdata: extracted permittivity and permeability
    - extract\_method: method used for extraction (NRW or NNI)
    - mu\_np1: number of 1st-order poles used to fit mu (Laurent model only)
    - eps\_np1: number of 1st-order poles used to fit epsilon (Laurent model only)
    - np2: number of 2nd-order poles used to fit mu and epsilon (Laurent model only)
    - wg\_x0: initial parameter vector for waveguide (L1, L2, lambda\_c)
    - mu\_x0: initial parameter vector for mu (Laurent model only)
    - eps\_x0: initial parameter vector for epsilon
    - lm\_initfits: table of initial 1st-order pole fits for mu and epsilon with metrics (Laurent model only)
    - lmfit: Laurent model fit result
    - lm\_internal: internal variables from Laurent model fit, stored for troubleshooting
    - defit: dielectric model fit result
    - de\_internal: internal variables from dielectric model fit, stored for troubleshooting
  + Methods:
    - load(filename,<options>): load raw data file
      * filename: full name (with extension) of data file to load. If the file is not in the working directory, filename must specify the path to the file
      * <options>
        + HeaderLines: number of header lines to skip in file. Defaults to 0
    - extract(meth,<options>): perform NRW or NNI extraction to get permittivity and permeability
      * meth: extraction method. Must be ‘nrw’ or ‘nni’
      * <options>
        + BaseBranch: which branch to use for calculation. Defaults to 0
        + IncrementBranch: if true, check for discontinuities indicative of branch changes and increment branch from BaseBranch accordingly
    - plot\_PP: plot extracted mu and epsilon vs. frequency
    - plot\_Sij: plot all measured S parameters vs. frequency
    - lm\_initialfit(<options>): Perform initial fit of Laurent model to mu and epsilon with 1st-order poles
      * <options>
        + MaxNumPoles: maximum number of poles to fit. Default 10
        + CrossValidate: if true, perform cross-validation for each number of poles and return CV metrics. Default true
        + TrainSize: fraction of data to use for training in CV. Defaults to 0.1
        + EvalSplits: number of CV splits to evaluate. Default 5
        + NPoles: number of poles to fit. If specified, MaxNumPoles will be ignored and fit(s) will be performed only with NumPoles. Not set by default
    - lm\_plot\_initfit(field,NPoles): plot initial fit of Laurent model to mu or epsilon
      * field: which field to plot. Must be ‘mu’ or ‘eps’
      * NPoles: number for poles to plot fit for
    - lm\_estimate\_x0(<options>): get initial parameter vectors for mu and epsilon fits
      * <options>
        + NPoles1: number of 1st-order poles to use for both mu and eps. Default 2
        + NPoles2: number of 2nd-order poles to use for both mu and eps. Default 1
        + EstOrd2: if true, estimate initial parameters for 2nd-order poles. If false, initialize 2nd-order poles parameters at zero. Default true
        + MuNPoles1: number of 1st-order poles to use for mu. If specified, overrides NPoles1
        + EpsNPoles1: number of 1st-order poles to use for eps. If specified, overrides NPoles1
    - lm\_lsqfit(<options>): perform least-squares fit of Laurent model to measured S parameters
      * <options>
        + MaxFEval: maximum number of function evaluations to allow the optimizer to perform. Defaults to 100\*NParams, where NParams = 7 + 4\*(MuNPoles1 + EpsNPoles1) + 8\*NPoles2
        + MultiStart: if true, start the optimization from multiple random initial points. Default true
        + MultiStartPoints: number of points from which to initialize the
        + optimization. Default 10
        + ShiftFOPoles: if true, shift the parameters for first-order poles when generating MultiStart points. If false, leave the first-order pole parameters the same for all MultiStart points (only shift 2nd-order poles). Default false
        + NCCP2: number of complex conjugate pairs in second-order poles. 0 (real 2nd-order poles only) tends to work well, but in some cases complex 2nd-order poles may be beneficial. Use 0 first, and then try increasing to 1 if 0 doesn’t provide good results. Default 0
    - lm\_plot\_Sfit: plot Laurent model fit of S parameters
    - lm\_plot\_PPfit: plot Laurent model fit of permittivity and permeability
    - lm\_output(<options>): output full Laurent model results, including measured S parameters, extracted mu and epsilon, and Laurent model fits of S parameters, mu, and epsilon
      * <options>
      * SaveFile: filename to which to save results. if specified, save output to filename; otherwise, output results to variable. If the file is not in the working directory, filename must specify the path to the file
    - de\_estimate\_x0(<options>): estimate initial parameters for dielectric model by fitting poles to extracted epsilon
      * <options>
        + NPoles: Number of poles to be distributed uniformly across frequency range. Defaults to 2
        + PoleFreq: pole frequencies. If specified, NPoles is ignored, and one pole is initialized at each frequency specified.
    - de\_lsqfit(<options>): perform least-squares fit of dielectric model to measured S parameters
      * <options>
        + MaxFEval: maximum number of function evaluations to allow the optimizer to perform. Defaults to 100\*NParams, where NParams = 5 + 3\*NPoles
        + MultiStart: if true, start the optimization from multiple random initial points. Default true
        + MultiStartPoints: number of points from which to initialize the
        + optimization. Default 10
    - de\_plot\_Sfit: plot dielectric model fit of S parameters
    - de\_plot\_PPfit: plot dielectric model fit of permittivity and permeability
    - de\_output(<options>): output full dielectric model results, including measured S parameters, extracted mu and epsilon, and Laurent model fits of S parameters, mu, and epsilon
      * <options>
      * SaveFile: filename to which to save results. if specified, save output to filename; otherwise, output results to variable. If the file is not in the working directory, filename must specify the path to the file

**Functions**

Data Loading

* loadconvert\_raw: load raw file and convert the Sij magnitude and phase to complex values
  + Input: raw data file
  + Output: table with frequency, S11, S21, S12, and S22
  + Depends on:
    - load\_raw: load raw files without conversion

NRW Extraction

* nrw\_extract: perform point-by-point calculation on Sij data (loaded by loadconvert\_raw) to obtain permittivity and permeability
  + Inputs:
    - table returned by loadconvert\_raw
    - sample thickness, L
    - cutoff wavelength, lambda\_c
  + Output: table with frequency, mu, and epsilon
  + Depends on:
    - nrw\_ks: calculate wavenumber
    - nrw\_lambda: calculate big lambda
    - nrw\_ref\_coef: calculate reflection coefficient
    - nrw\_trans\_coef: calculate transmission coefficient
    - nrw\_Z: calculate Z term

Reverse S-Parameter calculation

* rev\_transform: perform point-by-point calculation on mu and epsilon to obtain S11, S21, S12, and S22
  + Inputs:
    - table with frequency, mu, and epsilon
    - Cutoff wavelength, lambda\_c
    - Sample thickness, L
    - Sample offsets, L1 and L2
  + Output: table with frequency, S11, S21, S12, and S22
  + Depends on:
    - rev\_gamma: calculate material propagation constant gamma
    - rev\_gamma\_0: calculate air propagation constant gamma\_0
    - rev\_ref\_coef: calculate reflection coefficient
    - rev\_Ri: calculate rotational term
    - rev\_trans\_coef: calculate transmission coefficient

Laurent Model Fit

* lm\_estimate\_x0: estimate the initial fit parameters by fitting separate Laurent models to mu and epsilon
  + Inputs:
    - table with frequency, mu, and epsilon
    - Number of first-order poles to fit for mu and epsilon
    - Number of second-order poles to fit
  + Outputs:
    - initial guess parameters for mu model
    - initial guess parameters for epsilon model
  + Depends on:
    - lm\_initialfit: fit 1st-order poles for mu and epsilon
    - lm\_initialfit2: fit 2nd-order poles for mu and epsilon
    - lm\_extract\_rfx: extract parameter vector from rationalfit object
    - lm\_expand\_x: convert real parameter vector x to separate vectors of complex Laurent model terms
    - lm\_flatten\_terms: convert vectors of complex Laurent model terms to a real parameter vector. Required for optimization functions
    - score\_ratfit: calculate R2 score of rationalfit
    - lm\_eval: evaluate Laurent model
    - lm\_ord2ratfit: fit 2nd-order poles for single function
* lm\_lsqfit: perform least-squares optimization of Laurent model fit to S parameters
  + Inputs:
    - table with frequency and measured S11, S21, S12, and S22
    - sample holder thickness, L\_tot
    - initial guess parameters for mu model
    - initial guess parameters for epsilon model
    - initial guesses for L1, L2, and lambda\_c
    - number of first-order poles for mu and epsilon
    - number of second-order poles
    - optimization options
  + Outputs:
    - lmfit object with optimized parameters
    - struct with internal variables for troubleshooting
  + Depends on:
    - lm\_prep\_x0: prepares initial guess vector for optimization
    - lm\_unpair\_cc: unpairs complex conjugates in Laurent model terms prior optimization
    - lm\_pair\_cc: re-pairs complex conjugates in a single term vector
    - lm\_repair\_x: re-pairs complex conjugates in full parameter vector
    - lm\_xbounds: determines bounds for each parameter in parameter vector
    - rev\_transform
    - lm\_lsqfun: objective function for lsqcurvefit; calculates Sij values given parameter vector
    - lm\_mspoints: generates random starting points for MultiStart optimization
* lm\_objfunfit: same as lm\_lsqfit, but uses a different optimization algorithm (doesn’t work as well as lm\_lsqfit, don’t use)
* lm\_plot\_Sfit: plot Laurent model fit of Sij values versus measured values
  + Inputs:
    - lmfit object
    - table with frequency and measured S11, S21, S12, and S22
  + Outputs: none
  + Depends on:
    - lm\_rev\_eval: evaluate Laurent models for mu and epsilon, then perform reverse transform to calculate Sij
* lm\_plot\_PPfit: plot Laurent model fit of mu and epsilon versus NRW-calculated values
  + Inputs:
    - lmfit object
    - table with frequency and NRW-calculated mu and epsilon
  + Outputs: none
  + Depends on:
    - lm\_eval: evaluate Laurent model

Dielectric Model Fit

* de\_estimate\_x0: estimate initial parameters for dielectric model by fitting fixed-frequency poles to epsilon
  + Inputs:
    - Table with frequency and extracted epsilon
    - Number of poles to fit (optional)
    - Pole frequencies (optional)
  + Output: initial parameter vector
  + Depends on:
    - de\_initialfit: perform initial fit of dielectric model to epsilon
* de\_lsqfit: perform least-squares optimization of dielectric model fit to S parameters
  + Inputs:
    - table with frequency and measured S11, S21, S12, and S22
    - sample holder thickness, L\_tot
    - initial guess parameters for epsilon model
    - initial guesses for L1, L2, and lambda\_c
    - optimization options
  + Outputs:
    - defit object with optimized parameters
    - struct with internal variables for troubleshooting
  + Depends on:
    - de\_xbounds: get bounds for parameter vector
    - de\_eval: evaluate dielectric model for epsilon for given parameter vector and frequencies
    - de\_rev\_eval: evaluate dielectric model for epsilon, then perform reverse transform to calculate Sij
    - de\_lsqfun: objective function for least-squares optimization
    - lm\_mspoints: generates random starting points for MultiStart optimization
* de\_plot\_Sfit: plot dielectric model fit of Sij values versus measured values
  + Inputs:
    - defit object
    - table with frequency and measured S11, S21, S12, and S22
  + Outputs: none
  + Depends on:
    - de\_rev\_eval: evaluate dielectric model for epsilon, then perform reverse transform to calculate Sij
* de\_plot\_PPfit: plot dielectric model fit of mu and epsilon versus NRW-calculated values
  + Inputs:
    - lmfit object
    - table with frequency and NRW-calculated mu and epsilon
  + Outputs: none
  + Depends on:
    - de\_eval: evaluate dielectric model for epsilon for given parameter vector and frequencies