SEMPO.ParameterManager

Tensorize

parameterList	SEMPO base object

Turns the numpy arrays into torch tensors in parameterList

Numpify

parameterList	SEMPO base object

Turns the torch tensors into numpy arrays in parameterList

GenerateSEMParameterList

fctType	sem/gdl/mtse
tensorized	use np or torch
Hnr	optional non-resonant term
residues	optional initial residues (used with initial poles)
poles	optional initial poles
nPoles	number of non-initial poles (might become effective poles)
nPolesIm	number of non-initial purely imaginary poles (might become effective poles)
nPolesEff	number of effective poles
stableEffectivePoles	constraint the effective poles to the lower part of the complex frequency plane
poleOrigin	should a pole be added at the origin (used for fctType = sem)
w	optional frequency range over which the non-initial poles are drawn (if None, then randomly draw them)
minDist	min distance to imaginary axis below which initial poles are considered as imaginary

SEM: parameterList = [fctType, False, Hnr, pnC_R, pnC_I, rnC_R, rnC_I, pnI_I, rnI_I, pnE_R, pnE_I, rnE_R, rnE_I, r0, poleOrigin, stableEffectivePoles]

GDL: parameterList = [fctType, False, Hnr, G0, an, bn, s1n, Gn, wn, s2n, pnE_R, pnE_I, rnE_R, rnE_I, stableEffectivePoles]

MTSE: parameterList = [fctType, False, Hnr, mrn, trn, pnC_R, pnC_I, pnE_R, pnE_I, rnE_R, rnE_I, stableEffectivePoles]

GenerateSZFParameterList

tensorized	use np or torch
G0	optional scaling term
poles	optional initial poles
zeros	optional initial zeros
nPoles	number of non-initial poles (might become effective poles)
nPolesIm	number of non-initial purely imaginary poles (might become effective poles)
nPolesEff	number of effective poles
stableEffectivePoles	constraint the effective poles to the lower part of the complex frequency plane
poleOrigin	should a pole be added at the origin (used for fctType = sem)
nZeros	number of non-initial zeros (might become effective zeros)
nZerosIm	number of non-initial purely imaginary zeros (might become effective zeros)
nZerosEff	number of effective poles
reversibleZeros	constraint all the zeros to the lower part of the complex frequency plane
w	optional frequency range over which the non-initial poles and zeros are drawn (if None, then randomly draws them)
minDist	min distance to imaginary axis below which initial poles and zeros are considered as imaginary

SZF: parameterList = ["szf", False, H0, pnC_R, pnC_I, pnI_I, pnE_R, pnE_I, znC_R, znC_I, znI_I, znE_R, znE_I, poleOrigin, reversibleZeros, stableEffectivePoles]

ConvertToSEM

parameterList	SEMPO object to convert
wa	Arbitrary frequency wa used to compute Hnr from the SZF

Converts parameterList into the SEM format

ConvertToGDL

oldParameterList	SEMPO object to convert
wa	Arbitrary frequency wa used to compute Hnr from the SZF

Converts parameterList into the GDL format

ConvertToMTSE

oldParameterList	SEMPO object to convert
wa	Arbitrary frequency wa used to compute Hnr from the SZF

Converts parameterList into the MTSE format

ConvertToGDL

oldParameterList	SEMPO object to convert
wa	Arbitrary frequency wa used to compute Hnr from the SZF

Converts parameterList into the GDL format

GetPolesAndResidues

oldParameterList	SEMPO object from which to extract the poles and residues
wa	Arbitrary frequency wa used to compute Hnr from the SZF

Extracts just the poles and residues from the parameterList. This can be used with the SZF format by converting it to the SEM format.

GetPolesAndZeros

oldParameterList	SEMPO object from which to extract the poles and zeros

Extracts just the poles and zeros from **a parameterList in the SZF format.**

SEMPO.Cauchy

$$H(\omega) = g_0 \frac{\prod_\ell (\omega - z_\ell)}{\prod_\ell (\omega - p_\ell)}$$

GetResidueFromCauchy

g0	Scaling prefactor g0 obtained with the Cauchy method
poles	array of poles obtained with the Cauchy method
zeros	array of zeros obtained with the Cauchy method

Calculates the residues associated with each pole

rp: numpy array of residues

GetHnrFromCauchy

g0	Scaling prefactor g0 obtained with the Cauchy method
poles	array of poles obtained with the Cauchy method
zeros	array of zeros obtained with the Cauchy method
residues	array of residues associated with the poles
wa	Arbitrary frequency used to calculate HNR

Using the SZF and the residues, calculate the non-resonant term HNR

HNR: scalar non-resonant term

OriginalCauchyMethod

H0	Data to model, *i.e.* the numpy array of H(wi)
W0	The frequencies wi as a numpy array
nmbMaxPoles	Maximum number of poles used to approximate H0
Phys	Bool: should the poles & zeros be Hermitian-symmetric?
dWorigin	If Phys, min distance below which poles & zeros are imaginary
useLogPrec	Bool: use fixed threshold for the svd. If false, auto picks
logPrec	Threshold to use for the svd (if useLogPrec)
plotSingularValues	Bool: plot the singular values figure or not after the svd
figID	If plotSingularValues, corresponds to the ID of the figure

Runs the original Cauchy method on (H0, W0) to retrieve the parameters of the SZF

g0: scalar

p: numpy array

z: numpy array

CauchyMethod

H0	Data to model, *i.e.* the numpy array of H(wi)
W0	The frequencies wi as a numpy array
nmbMaxPoles	Maximum number of poles used to approximate H0
Phys	Bool: should the poles & zeros be Hermitian-symmetric?
dWorigin	If Phys, min distance below which poles & zeros are imaginary
useLogPrec	Bool: use fixed threshold for the svd. If false, auto picks
logPrec	Threshold to use for the svd (if useLogPrec)
plotSingularValues	Bool: plot the singular values figure or not after the svd
figID	If plotSingularValues, corresponds to the ID of the figure

Modified version of the Cauchy method with dynamical thresholding of the svd such that the error on the reconstruction is minimized. This is what we refer to as the **Accuracy-Driven Cauchy method (ADC method)**

g0: scalar
poles: numpy array
zeros: numpy array

CauchyMethodOptZP

H0	Data to model, *i.e.* the numpy array of H(wi)
W0	The frequencies wi as a numpy array
nmbMaxPoles	Maximum number of poles used to approximate H0
phys	Bool: should the poles & zeros be Hermitian-symmetric?
dWorigin	If Phys, min distance below which poles & zeros are imaginary
Stability	Bool: should a penalty be added to non-stable poles?
qStability	Weight given to the stability constraint (if Stability). Although available, it is not recommended!
useLogPrec	Bool: use fixed threshold for the svd. If false, auto picks
logPrec	Threshold to use for the svd (if useLogPrec)
diffZPMax	The maximum difference between the numbers of poles and zeros
plotSingularValues	Bool: plot the singular values figure or not after the svd
figID	If plotSingularValues, corresponds to the ID of the figure

Improved version of the ADC method where the difference between the numbers of poles and zeros is also swept through in order to lower the error on the reconstruction.

g0: scalar
poles: numpy array
zeros: numpy array

ChainCauchyMethodOptZP

H0	Data to model, *i.e.* the numpy array of H(wi)
W0	The frequencies wi as a numpy array
nmbWindows	Number of subdivisions of the spectral range
nmbMaxPoles	Maximum number of poles used to approximate H0
phys	Bool: should the poles & zeros be Hermitian-symmetric?
dWorigin	If Phys, min distance below which poles & zeros are imaginary
Stability	Bool: should a penalty be added to non-stable poles?
qStability	Weight given to the stability constraint (if Stability)
useLogPrec	Bool: use fixed threshold for the svd. If false, auto picks
logPrec	Threshold to use for the svd (if useLogPrec)
diffZPMax	The maximum difference between the numbers of poles and zeros
splitEvenly	If True, then the windows have the same number of points (evenly splits W0). Otherwise, the range is evenly split.
figID	If plotSingularValues, corresponds to the ID of the figure

Chain ADC method used in complex curves to get an approximation piece by piece, over sub-windows. The result is an approximation for each sub-window, not a complete approximation

g0n: list of scalars
pn: list of numpy arrays
zn: list of numpy arrays
W0n: list of numpy arrays (the sub-window frequencies)
H0n: list of numpy arrays (the sub-window data)

StackChainResult

g0n	g0n obtained with the Chain ADC method
pn	List of sub-window pole arrays
zn	List of sub-window zero arrays
distMax	Distance over which the poles and zeros can be counted as constants and put in the resulting G0

Merges the results of the Chain ADC method by stacking all the poles and zeros from pn and zn into numpy arrays. The poles and zeros too far from the origin in the complex frequency plane are considered as constants.

g0: scalar pre-factor term
p: numpy array of poles
z: numpy array of zeros

HardMergeChain

g0n	g0n obtained with the Chain ADC method
pn	List of sub-window pole arrays
zn	List of sub-window zero arrays
wn	List if sub-window frequency arrays

Merges the results of the Chain ADC method. For each sub-window in wn, there is an array of poles in pn, and an array of zeros in zn. We use them to obtain the residues associated with the poles. We filter out the poles that are outside of the sub-window frequency range. **The result is the set of parameters of the SEM**

hnr: scalar non-resonant term approximation
poles: numpy array of poles
residues: numpy array of residues

SEMPO.Autodiff

H_terms

w	Frequencies over which the terms are calculated
parameterList	SEMPO base object to use for the calculation of the terms

Uses the parameterList to compute the individual terms appearing in the SEM/GDL/MTSE/SZF expression.

fctType: string (sem/gdl/mtse/szf)
v: list of the individual terms

H

fctType	The type of parameterList (sem/gdl/mtse/szf) used to compute the individual terms
H_terms	The individual terms
Tensorized	If true, the terms are torch tensors. Otherwise numpy arrays

Calculates the full SEM or SZF expression using their individual terms (obtained *via* the H_terms function).

v: the numpy array or torch tensor of values

LossFct

Pr	The prediction, a torch tensor containing the approximation
Ta	The target, a torch tensor containing the values to approximate
alpha	4-upplet (a1, a2, a3, a4) giving the weight of each term in the loss function. ▪ a1 for the relative L_2 error ▪ a2 for the relative L_∞ error ▪ a3 for the error on the real part ▪ a4 for the error on the imaginary part
imaginaryPartPositive	Should the imaginary part of the prediction be positive? Useful mainly in the case of the permittivity.

Calculates the error between the prediction and the target using a custom loss function

l: the torch tensor containing the error

FitModel

W	The frequencies to use, over which the target is defined
Ta	The target to approximate
parameterList	SEMPO object used to generate the approximation
imaginaryPartPositive	Should the imaginary part of the prediction be positive? Useful mainly in the case of the permittivity.
alpha	4-upplet (a1, a2, a3, a4) giving the weight of each term in the loss function. ▪ a1 for the relative L_2 error ▪ a2 for the relative L_∞ error ▪ a3 for the error on the real part a4 for the error on the imaginary part
tensorizeInput	If true, assumes that the frequencies W and the target Ta are numpy arrays, and thus convert them before using them in the fitting process
nIter	Number of iterations
lr	Learning rate of the optimizer
scIter	Number of iterations after which the learning rate is multiplied by a scaling factor
scq	Scaling factor used to dynamically change the learning rate
vizIteration	Number of iterations after which the visualization is updated
visualizeLoss	If true, then a visdom object is used to visualize the loss function and the fitting process
loss_vizz	Visdom object to used (default to None)
logScalePoles	If True, a log-scale is used for the real part of the poles in the corresponding visualization window

Modifies the parameters of a SEMPO object by fitting a target over a set of frequencies. The fitting is performed by using auto-differentiation with a gradient-descent-like optimizer (AdamW).

parameterList: the updated SEMPO object

SEMPO.Model

G_ab

w	Frequencies
a	Coefficients of the polynomial in the numerator
B	Coefficients of the polynomial in the denominator

Calculates the SZF at a given set of frequencies, using the polynomial coefficients of the numerator and denominator.

v: numpy array of values

G_SZF

w	Frequencies
g0	Scaling prefactor
poles	Numpy array of poles
zeros	Numpy array of zeros

Calculates the SZF at a given set of frequencies, using the poles and zeros

v: numpy array of values

H_SEM

w	Frequencies
H_NR	Non-resonant term
P	Numpy array of poles
R	Numpy array of residues

Calculates the SEM at a given set of frequencies, using the poles and residues

v: numpy array of values

H_HermitianSEM

w	Frequencies
H_NR	Non-resonant term
P	Numpy array of poles
R	Numpy array of residues

Calculates the SEM at a given set of frequencies, using the poles and residues, by assuming a Hermitian-symmetric structure. Only the poles with a positive or null real part should be provided.

v: numpy array of values

MergePoles

poles	Numpy array of poles
residues	Numpy array of residues
W	Numpy array of frequencies. Only the boundaries are used
d0	Effective distance below which poles are merged
nPts	Number of frequencies used to calculate the effective distance

Merges poles by computing an effective distance between them. This effective distance corresponds to the L_2 error between the sum of their resonant terms, and the resonant term associated with the pole corresponding to their weighted average (the weights are the moduli of the residues).

p: numpy array of poles
r: numpy array of residues

RemovePoles

hnr	Non-resonant term HNR
poles	Numpy array of poles
residues	Numpy array of residues
W	Numpy array of frequencies. Only the boundaries are used
d0	Threshold below which the contribution of a resonant term is considered as negligeable
s0	Threshold below which the variations brought by a resonant term are considered as negligeable
nPts	Number of frequencies used to calculate the resonant terms

Removes the poles associated with resonant terms which have a negligeable contribution to the full SEM expression. The non-resonant term is also updated.

hnr: updated non-resonant term
p: numpy array of poles
r: numpy array of residues

StabilizePoles

p	Numpy array of poles
r	Numpy array of residues
q0	Distance to the real frequency axis where the poles are put

Turns unstable poles on the real frequency axis into conjugate pairs of poles really close to that axis. Should be used for **the amplitude or the phase**

pC: numpy array of stabilized poles
rC: numpy array of associated residues

SEMPO.DataManager

ReadDataFile

dataFilePath	Path to the file containing the data
columnNameIndex	Optional: which columns contains the indices
FreqColName	Name of the column with the frequencies/wavelength. Default to the first one
DataColNameR	Name of the column with the real part of the data. Default to the second one
DataColNameI	Name of the column with the imaginary part of the data. Default to the third one
sep	Separator used in the file. Default to ","
Decimal	Character used for decimal values. Default to "."
useLambda	If True, assumes that the wavelengths are used in the file, not the frequencies
FreqConversion	Multiplication factor for the frequencies. Default to 1e-15 (optical frequencies)
w1	Minimum frequency allowed (filters below)
w2	Maximum frequency allowed (filters above)
nPts	Number of frequencies used (draws a uniform sample)

Reads a data file and extract the arrays of frequencies and values

W: numpy array of frequencies

Hw: numpy array of values

SEMPO.Results

GetAxCoordinate

ax	Matplotlib axes object

Returns the coordinate of a Matplotlib axes object

bbox: numpy array of coordinates

AmplitudeMap

Hw	Numpy 2D array of data
W	Numpy 2D array of frequencies (complex frequency plane)
ax	Optional: axes where to plot the amplitude map
figID	Matplotlib figure to use
fs	Figure size
adjustFigure	If True, then modifies the figure in order to fit the colorbar

Plots the log-amplitude in the complex frequency plane

fig: matplotlib figure
ax: figure axes

AmplitudeCurve

Hw	Numpy array of data
W	Numpy array of frequencies
ax	Optional: axes where to plot the amplitude map
figID	Matplotlib figure to use
fs	Figure size

Plots the amplitude at the selected frequencies

fig: matplotlib figure
ax: figure axes

PhaseMap

Hw	Numpy 2D array of data
W	Numpy 2D array of frequencies (complex frequency plane)
ax	Optional: axes where to plot the amplitude map
figID	Matplotlib figure to use
fs	Figure size
adjustFigure	If True, then modifies the figure in order to fit the colorbar

Plots the phase in the complex frequency plane

fig: matplotlib figure

ax: figure axes

PhaseCurve

Hw	Numpy array of data
W	Numpy array of frequencies
ax	Optional: axes where to plot the amplitude map
figID	Matplotlib figure to use
fs	Figure size

Plots the phase at the selected frequencies

fig: matplotlib figure

ax: figure axes

BodeDiagram

Hw	Numpy array of data
W	Numpy array of frequencies
figID	Matplotlib figure to use
fs	Figure size

Generates a figure with the amplitude and the phase at the selected frequencies

fig: matplotlib figure

BodeMap

Hw	Numpy 2D array of data
W	Numpy 2D array of frequencies (complex frequency plane)
figID	Matplotlib figure to use
fs	Figure size

Generates a figure with the log-amplitude and the phase maps in the complex plane

fig: matplotlib figure

AmplitudeFig

Hx	Numpy 2D array of data
Wx	Numpy 2D array of frequencies (complex frequency plane)
Hw	Numpy array of data
W	Numpy array of frequencies
figID	Matplotlib figure to use
fs	Figure size

Plots the amplitude at real frequencies, and the log-amplitude in the complex frequency plane

fig: matplotlib figure

PhaseFig

Hx	Numpy 2D array of data
Wx	Numpy 2D array of frequencies (complex frequency plane)
Hw	Numpy array of data
W	Numpy array of frequencies
figID	Matplotlib figure to use
fs	Figure size

Plots the phase at real frequencies and in the complex frequency plane

fig: matplotlib figure