Analysis

Release 0.1

Chetan Sriram Madasu

Jun 21, 2022

CONTENTS:

CHAPTER

ONE

ANALYSIS

1.1 analysis package

1.1.1 Subpackages

analysis.Physics package

Subpackages

analysis.Physics.ElectricDipole package

Submodules

analysis.Physics.ElectricDipole.dipole module

analysis.Physics.ElectricDipole.polarizability module

class `analysis.Physics.ElectricDipole.polarizability.`**`ket`**(*n*, *L*, *S*, *J*, *I*, *F*, *mF*)

Bases: `object`

Definition of a state with appropriate quantum numbers as shown in the __init__.

`is_ket`()

`is_bra`()

class `analysis.Physics.ElectricDipole.polarizability.`**`bra`**(*n*, *L*, *S*, *J*, *I*, *F*, *mF*)

Bases: *`analysis.Physics.ElectricDipole.polarizability.ket`*

`is_ket`()

`is_bra`()

`analysis.Physics.ElectricDipole.polarizability.`**`alphaTensor`**(*F*, *K*, *eps*)

Full polarizability tensor eq. 5 in PRA reference

Parameters

- **`F`** – Total angular momentum of the required ground state
- **`K`** – coefficients in irreducible representation
- **`eps`** – polarization vector of the light (E_x, E_y, E_z)

`analysis.Physics.ElectricDipole.polarizability.`**`alpha`**(*F*, *mF*, *K*, *eps*)

Polarizability of each mF.

Parameters

- **`F`** – Total angular momentum of the required ground state.
- **`mF`** – magnetic quantum number.
- **`K`** – coefficients in irreducible representation
- **`eps`** – polarization vector of the light (E_x, E_y, E_z)

`analysis.Physics.ElectricDipole.polarizability.`**`tens`**(*j*, *M*, *ux*, *uy*, *uz*)

Helper function to convert polarization of the light into compound tensor components using equation 12 in epjd reference of cesium

`analysis.Physics.ElectricDipole.polarizability.`**`B_fs`**(*k_i*, *k_f*)

Branching ratio for fine structure.

Parameters

- **`k_i`** – excited state
- **`k_f`** – ground state

`analysis.Physics.ElectricDipole.polarizability.`**`B_hfs`**(*k_i*, *k_f*)

Branching ratio of hyperfine structure

Parameters

- **`k_i`** – excited state
- **`k_f`** – ground state

`analysis.Physics.ElectricDipole.polarizability.`**`K`**(*j*, *k_f*, *k_is*, *w_l*, *params*)

Coefficients in irreducible representation of polarizability as shown in equation 5 of PRA reference.

Parameters

- **`j`** – 0 for scalar, 1 for vector, 2 for tensor coefficient
- **`k_f`** – ground state
- **`k_is`** – list of all excited states
- **`w_l`** – wavelength
- **`params`** – spectroscopic Args of all excited states freq., z costant, decay rate

Submodules

analysis.Physics.OBS module

`analysis.Physics.OBS.`**`BStoDS`**(**args*)

Change of basis matrix from bare state basis {0, 1, 2, 3} to dressed basis {D1, D2, B1, B2}

`class` `analysis.Physics.OBS.`**`OBSolve`**(*t*, *decayMatrix*)

Bases: `object`

`Hamiltonian`(*H*, **HArgs*)

`Liovellian`(*t*)

`solve`(*initialCondition*, *max_step=inf*, *dense_output=False*)

solveFiniteT(*initialCondition*, *T*, *Na*)

analysis.Physics.allenDeviation module

class analysis.Physics.allenDeviation.**TSeries**(*dataPath*, *channel*, *f0*, *Amplitude=None*, *offset=None*)

Bases: object

fracFreq()

timeDiff()

allenVar(*m*)

allenDev(*m*)

oAllenVar(*m*)

oAllenDev(*m*)

analysis.Physics.basisMatrices module

class analysis.Physics.basisMatrices.**PauliBasis**

Bases: object

matrices()

decompose(*A*)

class analysis.Physics.basisMatrices.**GellMannBasis**

Bases: object

matrices()

structureConstants()

structureConstant(*i*, *j*, *k*)

decompose2(*A*)

decompose(*A*)

compose(*decomposition*)

analysis.Physics.createAndDestroy module

analysis.Physics.createAndDestroy.**a**(*N=5*)

Matrix representation of bosonic destruction operator.

Parameters

- **n** – destroy boson at n in an N state Fock space
- **N** – dimensions of the Fock space, default 5

Returns a sparse matrix representation of a of shape (N, N) in Fock basis, $\{|n_1, n_2, n_3, ..., N\rangle\}$.

analysis.Physics.createAndDestroy.**a_dag**(*N=5*)

Matrix representation of bosonic creation operator.

Parameters

- **n** – create a boson at n in an N state Fock space
- **N** – dimensions of the Fock space, default 5

Returns a sparse matrix representation of a of shape (N, N) in Fock basis, $\{|n_1, n_2, n_3, ..., N\rangle\}$.

analysis.Physics.ehrenfest module

analysis.Physics.ehrenfest.**omegaConstant**(*t*)

analysis.Physics.ehrenfest.**omegaRamp**(*t*, *t_ramp*)

analysis.Physics.ehrenfest.**omegaGaussian**(*t*, *amp*, *Tc*, *sigma*)

analysis.Physics.ehrenfest.**omegaDoubleGaussian**(*t*, *amp*, *Tc1*, *Tc2*, *sigma*)

analysis.Physics.ehrenfest.**Map**(*fun*, *vec*, *args*, *dtype=<class 'float'>*)
Maps a function to a vector.

Parameters

- **fun** – function to be mapped
- **vec** – 1d-array, the vector to be mapped to the function
- **args** – tuple, secondary arguments to the function
- **dtype** – Default numpy.float, the data type of the elements of the array

Returns 1d-array with elements of given dtype mapped by the specified function.

class analysis.Physics.ehrenfest.**EhrenfestSU2**(*omega1*, *omega2*, *omega3*, *omega1_args*, *omega2_args*, *omega3_args*)
Bases: object

mixingAngles(*t*)

evolve(*t*, *y0*, *T*, *N=1000*, *p_x=0*, *p_y=0*, *d1=- 1*, *d2=1*, *d3=3*)

bareStatePop(*t*)

bareStatePop2(*t*, *result*)

spatialDistributions(*t*)

class analysis.Physics.ehrenfest.**Tripod**(*omega1*, *omega2*, *omega3*, *omega1_args*, *omega2_args*, *omega3_args*)
Bases: object

mixingAngles(*t*)

analysis.Physics.ehrenfest.**mixingAngles**(*omega_1*, *omega_2*, *omega_3*)

analysis.Physics.ehrenfest.**mixingAnglesSU2**(*omega_1*, *omega_2*, *omega_3*)

analysis.Physics.ehrenfest.**SU2_GaugeField**(*alpha*, *beta*)

analysis.Physics.ehrenfest.**SU2_ScalarTerm**(*alpha*, *beta*)

analysis.Physics.ehrenfest.**SU3_GaugeField**(*theta_l*, *phi_l*, *theta_r*, *phi_r*)

analysis.Physics.ehrenfest.**SU3_ScalarTerm**(*theta_l*, *phi_l*, *theta_r*, *phi_r*)

class analysis.Physics.ehrenfest.**EhrenfestSU3**(*Tripod_l*, *Tripod_r*)
Bases: object

evolve(*t*, *y0*, *T*, *N=1000*, *p_x=0*, *p_y=0*)

bareStatePop(*t*)

bareStatePop2(*t*, *result*)

analysis.Physics.run_ehren_mp module

analysis.Physics.spinInDickeBasis module

`analysis.Physics.spinInDickeBasis.`**`S_m`**(*s*)

Matrix representation of ladder lowering operator for spin, in general angular momentum.

Parameters **s** – spin quantum number.

Returns a sparse matrix representation of S^- of shape (2s+1, 2s+1) in Dicke basis, $\{|s, m_s\rangle\}$.

`analysis.Physics.spinInDickeBasis.`**`S_p`**(*s*)

Matrix representation of ladder raising operator for spin, in general angular momentum.

Parameters **s** – spin quantum number.

Returns a sparse matrix representation of S^+ of shape (2s+1, 2s+1) in Dicke basis, $\{|s, m_s\rangle\}$.

`analysis.Physics.spinInDickeBasis.`**`S_x`**(*s*)

Matrix representation of x component of the operator for spin, in general angular momentum.

Parameters **s** – spin quantum number.

Returns a sparse matrix representation of S_x of shape (2s+1, 2s+1) in Dicke basis, $\{|s, m_s\rangle\}$.

`analysis.Physics.spinInDickeBasis.`**`S_y`**(*s*)

Matrix representation of y component of the operator for spin, in general angular momentum.

Parameters **s** – spin quantum number.

Returns a sparse matrix representation of S_y of shape (2s+1, 2s+1) in Dicke basis, $\{|s, m_s\rangle\}$.

`analysis.Physics.spinInDickeBasis.`**`S_z`**(*s*)

Matrix representation of z component of the operator for spin, in general angular momentum.

Parameters **s** – spin quantum number.

Returns a sparse matrix representation of S_z of shape (2s+1, 2s+1) in Dicke basis, $\{|s, m_s\rangle\}$.

`analysis.Physics.spinInDickeBasis.`**`Spin`**(*s*)

Matrix representation of quantum mechanical spin, in general angular momentum, s.

Parameters **s** – int or half int, the spin quantum number.

Returns a tuple of sparse matrices corresponding to S_x, S_y, S_z

`analysis.Physics.spinInDickeBasis.`**`SpinAngularMomenta`**(*I*, *L*, *S*)

Returns angular momenta operators of a state given I, L, S in the tensor product basis.

Parameters

- **I** – nuclear spin quantum number of the atomic state
- **L** – orbital angular momentum quantum number
- **S** – spin quantum number of the state

Returns a tuple of angular momenta, $((I_x, I_y, I_z), (L_x, L_y, L_z), (S_x, S_y, S_z))$ (each a tuple of components as sparse matrices) in tensor product space.

1.1.2 Submodules

1.1.3 analysis.Images module

class `analysis.Images.`**`rcParams`**

Bases: `object`

A class that is designed to read and update resource parameters of the package. These parameters are usually the hardware parameters that are used while converting certain digital values to real units. Currently these are imaging parameters like magnification, pixel size etc. The idea behind this is that once these parameters are set, these are used by many function and classes throughout the package without having to explicitly pass them. Open the file rcParams.json using the notepad to see the current parameters used throughout the library.

`params`

dict, current parameters used by the package

`update`(*key*, *value*)

A method to update the rcParams.

Parameters

- **`key`** – string, key of the parameter. Careful about the spelling of the key. If its wrong, then you will add another parameter with the wrong spelling instead of updating the desired parameter.
- **`value`** – value of the parameter to update.

class `analysis.Images.`**`ShadowImage`**(*filePath*)

Bases: `object`

The main class to extract relevant information from the shadow imaging sequences that are obtained in the experiment. The multiple image sequence consists of the shadow or the absorption image, image of the incident probe and the image of the background for as many runs of the experiment.

Parameters **`filePath`** – str, Path to the multiple-image file

`filePath`

str, Path to the multiple-image file

`ext`

str, extension of the image file passed.

`im`

PIL.TiffImagePlugin.TiffImageFile or AndorSifFile._SifFrames

`n`

int, total number of data = number of images/3

`frames`

ndarray, all the frames in the image file

`tags`

dict, the named tag dictionary of the TiffImage or properties of sif file

`transmission`

ndarray, All the transmission images after subtracting the background

`incidence`

ndarray, All the images of incident probe after subtracting the background

`OD`

ndarray, optical depth all the runs of the experiment calculated from the transmission and incidence

`ODaveraged`
ndarray, OD averaged with averaging like a superloop

`averagedOD`
ndarray, OD of the averaged signal with averaging on superloop

`averagedOD2`
ndarray, OD of the averaged signal with averaging on loop

`images`()
Returns the images present in the ShadowImage object as an ndarray.

`opticalDepth`(*xSpan*, *ySpan*)
Returns the optical depth calculated from the ShadowImage as an ndarray.

Parameters

- **`xSpan`** – list, containing the range of pixels of the image in x direction.
- **`ySpan`** – list, containing the range of pixels of the image in y direction.

`opticalDepthAveraged`(*nAveraging*)
Calculates the optical depth from every triad of the images and the take average with nAveraging as the Superloop in the experiment.

Parameters **`nAveraging`** – int, number of averaging to be used.

Returns an ndarray of length equal to Serie in the experiment.

`averagedSignalOD`(*nAveraging*, *truncate=()*)
Calculates the average signal with nAveraging being the Superloop in the experiment and finds the optical depth after the averaging.

Parameters

- **`nAveraging`** – int, number of averaging to be used.
- **`truncate`** – tuple or list, the superloops to be ignored while averaging. Default is ().

Returns an ndarray of length equal to Serie in the experiment.

`averagedSignalOD2`(*nAveraging*, *truncate=()*)
Calculates the average signal with nAveraging being the loops in the experiment and finds the optical depth after the averaging.

Parameters

- **`nAveraging`** – int, number of averaging to be used.
- **`truncate`** – tuple or list, the superloops to be ignored while averaging. Default is ().

Returns an ndarray of length equal to Serie in the experiment.

`plotAveragedSignalOD`(*nAveraging*, *ROI*, *truncate=()*)
Calculates and plots the average signal with nAveraging being the Superloop in the experiment and finds the optical depth after the averaging.

Parameters

- **`nAveraging`** – int, number of averaging to be used.
- **`ROI`** – list, ROI in the images to be plotted.
- **`truncate`** – tuple or list, the superloops to be ignored while averaging. Default is ().

Returns None.

comment (*comment*)
Adds comment to the tif image under the ImageDescription tag and replaces it silently with the new image, if the user has the permissions.

Parameters **comment** – str, comment to be added.

description ()

Returns the ImageDescription tag of the tiff image file.

redProbeIntensity (*ROI*)
Estimates red probe intensity and power from the reference image using the losses in imaging system and quantum effeiciency of the camera for red light. rcParams used in calculating the intensity are `quantum efficiency`, `pixelSize`, `binning` and `magnification`.

Parameters **ROI** – list, ROI in which to estimate the intensity.

Returns intensity ($\mu W/cm^2$) and power (μW)

blueProbeIntensity (*ROI*)
Estimates blue probe intensity and power from the reference image using the losses in imaging system and quantum effeiciency of the camera for red light. rcParams used in calculating the intensity are `quantum efficiency`, `pixelSize`, `binning` and `magnification`.

Parameters **ROI** – list, ROI in which to estimate the intensity.

Returns intensity ($\mu W/cm^2$) and power (μW)

class analysis.Images.**FluorescenceImage** (*filePath*)
Bases: `object`

The main class to extract relevant information from the fluorescence image sequences that are obtained in the experiment. The multiple image sequence consists of fluorescence signal image and a background image acquired immediately after the signal from the atomic cloud for as many runs of the experiment.

Parameters **filePath** – str, Path to the multiple-image file

filePath
str, Path to the multiple-image file

im
PIL.TiffImagePlugin.TiffImageFile or AndorSifFile._SifFrames

n
int, total number of data = number of images/3

frames
ndarray, all the frames in the image file

tags
dict, the named tag dictionary of the TiffImage or properties of sif file

images ()
Returns the images present in the multiple image file as an ndarray.

fluorescence (*xSpan*, *ySpan*)
Returns the fluorence calculated from the FluorencenceImage as an ndarray.

Parameters

- **xSpan** – a list containing the range of pixels of the image in x direction.
- **ySpan** – a list containing the range of pixels of the image in y direction.

`averagedSignal`(*nAveraging*, *truncate=()*)
Calculates the average signal with nAveraging being the Superloop in the experiment and finds the fluorescence after the averaging.

Parameters

- **`nAveraging`** –

  type int, the repetitions of the experiment

- **`truncate`** –

  type tuple or list, the super loops to be ignored.

Returns an ndarray of length equal to Serie in the experiment.

`plotAveragedSignal`(*nAveraging*, *ROI*, *truncate=()*)
Calculates and plots the average signal with nAveraging being the Superloop in the experiment.

Returns None.

1.1.4 analysis.fits module

`analysis.fits.`**`gaussian`**(*x*, *amplitude*, *xo*, *sigma*, *offset*)

`analysis.fits.`**`gaussianFit`**(*x*, *array*, *p0=[]*, *bounds=[(), ()]*, *plot=True*)
Fits the given array to an 1D-gaussian.

Parameters

- **`x`** – 1darray, the argument values of the gaussian
- **`array`** – 1darray, the data to fit to the gaussian
- **`p0`** – ndarray, initial guess for the fit params in the form of [amplitude, xo, sigma, offset]. Default is None.
- **`bounds`** – tuple of lower bound and upper bound for the fit. Default is None.

Returns optimized parameters in the same order as p0 pCov: covarience parameters of the fit. Read scipy.optimize.curve_fit for details.

Return type pOpt

`analysis.fits.`**`gaussian2D`**(*X*, *amplitude*, *xo*, *yo*, *sigma_x*, *sigma_y*, *theta*, *offset*)

`analysis.fits.`**`gaussian2DFit`**(*image*, *p0=None*, *bounds=[(), ()]*, *plot=True*, *title=''*)
Fits an image with a 2D gaussian.

Parameters

- **`image`** – numpy ndarray
- **`p0`** – ndarray, initial guess for the fit params in the form of [amplitude, xo, yo, sigma_x, sigma_y, theta, offset]. Default None (fits for OD images).
- **`bounds`** – tuple of lower bound and upper bound for the fit. Default None (fits for OD images)
- **`plot`** – bool to show the plot of the fit. Default True.

Returns optimized parameters in the same order as p0 pCov: covarience parameters of the fit. Read scipy.optimize.curve_fit for details.

Return type pOpt

`analysis.fits.`**`threeGaussian2D`**(*X*, **args*)

`analysis.fits.`**`threeGaussian2DFit`**(*image*, *p0*, *bounds*, *TOF*, *plot=True*, *cropSize=6*, *logNorm=False*)

`analysis.fits.`**`multipleGaussian2D`**(*X*, **args*)

`analysis.fits.`**`multipleGaussian2DFit`**(*image*, *p0*, *bounds*, *TOF*, *plot=True*, *cropSize=6*, *tolerence=0.3*, *logNorm=False*)

`analysis.fits.`**`lorentzian`**(*x*, *amplitude*, *xo*, *gamma*, *offset*)

`analysis.fits.`**`lorentzianFit`**(*x*, *array*, *p0=[]*, *bounds=[(), ()]*, *plot=False*)

Fits the given array to a Lorentzian.

Parameters

- **`array`** – 1darray, the data to fit to the lorentzian
- **`p0`** – ndarray, initial guess for the fit params in the form of [amplitude, xo, gamma, offset]. Default is None.
- **`bounds`** – tuple of lower bound and upper bound for the fit. Default is None.

Returns optimized parameters in the same order as p0 pCov: covarience parameters of the fit. Read scipy.optimize.curve_fit for details.

Return type pOpt

1.1.5 analysis.numberOfAtoms module

`analysis.numberOfAtoms.`**`numAtomsBlue`**(*image*, *delta*, *imaging_params*, *s=0*, *plot=True*)

Calculates number of atoms from blue shadow imaging.

Parameters

- **`image`** – a numpy.ndarray, OD from the experiment
- **`delta`** – a float, detuning of the probe, 2*(AOMFreq-69) MHz
- **`imaging_params`** – a dictionary with keys as follows ex: {‘binning’:2, ‘magnification’: 2.2, ‘pixelSize’: 16*micro, ‘saturation’: 1 }
- **`s`** (*`optional`*) – a float, saturation parameter of the probe. Default is 0.
- **`plot`** (*`optional`*) – a bool, flag to plot the gaussian fits if True. Default is True.

Returns a tuple, (number of atoms from 2D gaussian fit, number of atoms from pixel sum, number density from gaussian fit, sigma_x, sigma_y, amplitude, x0, y0)

`analysis.numberOfAtoms.`**`numAtomsRed`**(*image*, *delta*, *imaging_params*, *s=0*, *plot=True*, *p0=None*, *bounds=[(), ()]*)

Calculates number of atoms from red shadow imaging.

Parameters

- **`image`** – a numpy.ndarray, OD from the experiment
- **`delta`** – float, detuning of the probe in kHz
- **`imaging_params`** – a dictionary with keys as follows ex: {‘binning’:2, ‘magnification’: 2.2, ‘pixelSize’: 16*micro }
- **`s`** (*`optional`*) – a float, saturation parameter of the probe. Default is 0.
- **`plot`** (*`optional`*) – a bool, a flag to plot the gaussian fits if True. Default is True.

Returns a tuple, (number of atoms from 2D gaussian fit, number of atoms from pixel sum, number density from gaussian fit, sigma_x, sigma_y, amplitude, x0, y0)

1.1.6 analysis.picoMatTools module

`analysis.picoMatTools.`**`picoMatRead`**(*filePath*, *channels=['A']*)

`analysis.picoMatTools.`**`PSD`**(*path*, *avg=30*, *channels=['A']*)

`analysis.picoMatTools.`**`RIN`**(*path*, *channel='A'*)

1.1.7 analysis.rigol module

`class` `analysis.rigol.`**`FuncGen`**(*connectionString*)

Bases: `object`

`arbBurst`(*waveform*, *frequency*, *channel=1*)

`switch`(*channel*, *status*)

1.1.8 analysis.sigma module

`analysis.sigma.`**`sigmaBlue`**(*delta*, *A*, *s=0*)

Calculates the scattering cross-section of Sr for $^1S_0 \rightarrow ^1P_1$ taking into account the isotope shift w.r.t. A=88 isotope and hyperfine levels present in the excited state.

Parameters

- **`delta`** – float, detuning of the probe w.r.t the reference in MHz.
- **`s`** – float, saturation parameter of the probe, I/I_s.
- **`A`** – int, mass number of the isotope.

Returns A float, Scattering cross-section.

Comment: All frequencies are in MHz.

`analysis.sigma.`**`sigmaRed`**(*delta*, *s=0*)

Calculates the scattering cross-section of Sr for $^1S_0 \rightarrow ^3P_1$ taking saturation into account.

Parameters

- **`delta`** – float, detuning of the probe w.r.t the reference in kHz.
- **`s`** – float, saturation parameter of the probe, I/I_s.

Returns A float, Scattering cross-section.

Comment: All frequencies are in kHz.

1.1.9 analysis.spectrum module

`analysis.spectrum.`**`spectroscopy`**(*ODimages*, *f*, *d=4*, *loss=False*, *plot=True*, *fileNum=''*, *savefig=False*)

Adds the OD of the pixels around the centre and uses the sum to plot the spectrum of the scan corresponding to the given frequencies. This is then fit to a lorentzian to find the center and linewidth.

Parameters

- **`ODimages`** – ODimages extracted from ShadowImaging sequences
- **`f`** – array of frequencies for which the scan is done
- **`d`** – int, to specify size of the image area to consider around the centre of OD
- **`plot`** – bool, default is True to specify if the data has to be plotted
- **`fileNum`** – string, file number (plus any additional description) of the image file for which the analysis is done.
- **`savefig`** – bool, default is False. Change it to true if you want to save the spectrum as .png

Returns a tuple, (amp, centre, gamma, offset) of the lorentzian fit

`analysis.spectrum.`**`bv`**(*f*, *f0*, *b0*, *T*, *s*)

Function representing convolution of the lorentzian line shape of the red transition and gaussian maxwell distribution. This is taken from 5.13 from Chang chi's thesis and added the effect of saturation parameter.

Parameters

- **`f`** – numpy.array, frequency vector
- **`f0`** – float, resonance frequency or centre of the spectrum
- **`b0`** – float, optical depth at resonance at zero temperature
- **`T`** – float, temperature in micro K
- **`s`** – float, saturation parameter of the probe, I/I_s.

Returns optical depth for frequencies f in the shape f.

`analysis.spectrum.`**`bvFit`**(*f*, *array*, *p0=None*, *bounds=None*)

Function to fit spectroscopy data to real line shape of the transition.

Parameters

- **`f`** – numpy.array, frequency vector
- **`array`** – float, optical depth at scan frequencies f
- **`p0`** – initial guess for the fit as [f_0, b_0, T (in μK), s]
- **`bounds`** – bounds for the fit as ([lower bounds], [upper bounds])

Returns (pOpt, pCov)

Return type a tuple with optimized parameters and covariance ex

`analysis.spectrum.`**`spectroscopyFaddeva`**(*ODimages*, *f*, *imaging_params*, *plot=True*, *fileNum=''*, *savefig=False*)

Fits od images to a gaussian and uses its amplitude to plot the spectrum of the scan corresponding to the given frequencies. This is fit to $b_v(\delta)$ from Chang Chi's thesis to extract temperature in addition to center.

Parameters

- **`ODimages`** – ODimages extracted from ShadowImaging sequences

- **f** – array of frequencies for which the scan is done
- **imaging_params** – a dictionary with keys as follows ex: {‘binning’:2, ‘magnification’: 2.2, ‘pixelSize’: 16*micro, ‘saturation’: 1 }
- **plot** – bool, default is True to specify if the data has to be plotted
- **fileNum** – string, file number (plus any additional description) of the image file for which the analysis is done.
- **savefig** – bool, default is False. Change it to true if you want to save the spectrum as .png

Returns a tuple, (centre, b_0, T, s) of fit to $b_v(\delta)$

1.1.10 analysis.tripodTools module

`analysis.tripodTools.`**`detunings`**(*p*, *theta*)

`analysis.tripodTools.`**`detunings2`**(*p_x*, *p_y*)

`analysis.tripodTools.`**`BStoDS`**(**args*)

Change of basis matrix from bare state basis {0, 1, 2, 3} to dressed basis {D1, D2, B1, B2}

`analysis.tripodTools.`**`Ds`**(*p1*, *p2*, *p3*, *p4*, *p5*)

Given the bare state populations of five +ve m_F ground states, calculates the state in bark state basis.

$$|\psi\rangle = d_l e^{i\alpha}|D_l\rangle + d_0|D_0\rangle + d_r e^{i\beta}|D_r\rangle$$

Parameters

- **p1** – population of $|m_F = 1/2\rangle$
- **p2** – population of $|m_F = 3/2\rangle$
- **p3** – population of $|m_F = 5/2\rangle$
- **p4** – population of $|m_F = 7/2\rangle$
- **p5** – population of $|m_F = 9/2\rangle$

Returns a tuple of d_l, d_0, d_r, α, β and state array $|\psi\rangle$

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX