

---

# **TOFU**

***Release 1.0.0***

**Chetan Sriram Madasu**

**Oct 01, 2022**



**CONTENTS:**

<b>1</b>	<b>TOFU</b>	<b>1</b>
1.1	TOFU package . . . . .	1
1.1.1	Subpackages . . . . .	1
1.1.2	Submodules . . . . .	3
1.1.3	TOFU.AndorSifReader module . . . . .	3
1.1.4	TOFU.Images module . . . . .	5
1.1.5	TOFU.fits module . . . . .	8
1.1.6	TOFU.jupyterTools module . . . . .	9
1.1.7	TOFU.numberOfAtoms module . . . . .	9
1.1.8	TOFU.picoMatTools module . . . . .	10
1.1.9	TOFU.sigma module . . . . .	10
1.1.10	TOFU.spectrum module . . . . .	11
1.1.11	Module contents . . . . .	12
<b>2</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>



## 1.1 TOFU package

### 1.1.1 Subpackages

TOFU.Physics package

Submodules

TOFU.Physics.allenDeviation module

```
class TOFU.Physics.allenDeviation.TSeries (dataPath, channel, f0, Amplitude=None, off-  
                                           set=None)  
    Bases: object  
    allenDev (m)  
    allenVar (m)  
    fracFreq ()  
    oAllenDev (m)  
    oAllenVar (m)  
    timeDiff ()
```

TOFU.Physics.basisMatrices module

```
class TOFU.Physics.basisMatrices.GellMannBasis  
    Bases: object  
    compose (decomposition)  
    decompose (A)  
    matrices ()  
    structureConstant (i, j, k)  
    structureConstants ()  
class TOFU.Physics.basisMatrices.PauliBasis  
    Bases: object  
    decompose (A)
```

**matrices()**

### TOFU.Physics.createAndDestroy module

TOFU.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,  $\{|n_1, n_2, n_3, \dots, N\rangle\}$ .

TOFU.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,  $\{|n_1, n_2, n_3, \dots, N\rangle\}$ .

### TOFU.Physics.spinInDickeBasis module

TOFU.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,  $\{|s, m_s\rangle\}$ .

TOFU.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,  $\{|s, m_s\rangle\}$ .

TOFU.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,  $\{|s, m_s\rangle\}$ .

TOFU.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,  $\{|s, m_s\rangle\}$ .

TOFU.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,  $\{|s, m_s\rangle\}$ .

TOFU.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  $S_x, S_y, S_z$

TOFU.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 of  $\mathbf{I} \otimes \mathbf{L} \otimes \mathbf{S}$  as sparse matrices) in tensor product space.

## Module contents

### 1.1.2 Submodules

### 1.1.3 TOFU.AndorSifReader module

**class** TOFU.AndorSifReader.**AndorSifFile**(*filename*)

Bases: object

This is main class to handle the reading of sif files. All the work is done in the constructor, the sif file is opened, data is read and the file is closed. This is adopted from <https://github.com/ardiloot/AndorSifReader> with modifications.

**Parameters** *filename* – str, Path to .sif file to read

**signal**

`_SifFrame` Instance of the `_SifFrame` helper class to store signal information.

**data**

ndarray, Convenience property linking to `signal.data`.

**props**

dict, Convenience property linking to `signal.props`.

**ANTSIF\_PROPS** = ['Type', 'Active', 'Version', 'Time', 'FormattedTime', 'FileName', 'Temp

**ATSIF\_AT\_32** = 1073741826

**ATSIF\_AT\_64** = 1073741828

**ATSIF\_AT\_8** = 1073741824

**ATSIF\_AT\_U32** = 1073741827

**ATSIF\_AT\_U64** = 1073741829

**ATSIF\_AT\_U8** = 1

**ATSIF\_Background** = 1073741826

**ATSIF\_CONV** = {1: <class 'numpy.uint8'>, 1073741824: <class 'numpy.int8'>, 1073741826

**ATSIF\_CalibX** = 1073741824

```
    ATSIF_CalibY = 1073741825
    ATSIF_CalibZ = 1073741826
    ATSIF_Double = 1073741831
    ATSIF_Float = 1073741830
    ATSIF_Live = 1073741827
    ATSIF_ReadAll = 1073741824
    ATSIF_ReadHeaderOnly = 1073741825
    ATSIF_Reference = 1073741825
    ATSIF_Signal = 1073741824
    ATSIF_Source = 1073741828
    ATSIF_String = 1073741832
    property data
    property props

class TOFU.AndorSifReader.SifError
    Bases: object

    This is a helper class to handle error codes produced by SIFReaderSDK.

    ATSIF_DATA_NOT_PRESENT = 'ATSIF_DATA_NOT_PRESENT'
    ATSIF_FILE_ACCESS_ERROR = 'ATSIF_FILE_ACCESS_ERROR'
    ATSIF_FILE_NOT_FOUND = 'ATSIF_FILE_NOT_FOUND'
    ATSIF_NO_SIF_LOADED = 'ATSIF_NO_SIF_LOADED'
    ATSIF_P1INVALID = 'ATSIF_P1INVALID'
    ATSIF_P2INVALID = 'ATSIF_P2INVALID'
    ATSIF_P3INVALID = 'ATSIF_P3INVALID'
    ATSIF_P4INVALID = 'ATSIF_P4INVALID'
    ATSIF_P5INVALID = 'ATSIF_P5INVALID'
    ATSIF_P6INVALID = 'ATSIF_P6INVALID'
    ATSIF_P7INVALID = 'ATSIF_P7INVALID'
    ATSIF_P8INVALID = 'ATSIF_P8INVALID'
    ATSIF_SIF_FORMAT_ERROR = 'ATSIF_SIF_FORMAT_ERROR'
    ATSIF_SUCCESS = 'ATSIF_SUCCESS'

    ERROR_STR = {22002: 'ATSIF_SUCCESS', 22003: 'ATSIF_SIF_FORMAT_ERROR', 22004: 'ATSIF_

    static FromCode(errorCode)
        Converts SIFReaderSDK error code to string.

        Parameters errorCode (int) – Error code number

        Returns Corresponding error string.

        Return type str
```



**static ProcessErrorCode** (*errorCode*)

Processes error codes, raises RuntimeError if the error code indicate problems. In case of success, does nothing.

**Parameters** **errorCode** (*int*) – Error code number

### 1.1.4 TOFU.Images module

**class** TOFU.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

**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.

**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.

**images** ()

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

**plotAveragedSignal** (*nAveraging*, *ROI*, *truncate=()*)

Calculates and plots the average signal with nAveraging being the Superloop in the experiment.

**Returns** None.

**class** TOFU.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

**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.

#### **blueProbeIntensity** (*ROI*)

Estimates blue probe intensity and power from the reference image using the losses in imaging system and quantum efficiency 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 ( $\mu W$ )

#### **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.

#### **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.

#### **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.

#### **redProbeIntensity** (*ROI*)

Estimates red probe intensity and power from the reference image using the losses in imaging system and quantum efficiency 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 ( $\mu W$ )

**class** TOFU.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.

## 1.1.5 TOFU.fits module

TOFU.fits.**gaussian** (*x*, *amplitude*, *xo*, *sigma\_x*, *offset*)

TOFU.fits.**gaussian2D** (*X*, *amplitude*, *xo*, *yo*, *sigma\_x*, *sigma\_y*, *theta*, *offset*)

TOFU.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: covariance parameters of the fit. Read `scipy.optimize.curve_fit` for details.

**Return type** pOpt

TOFU.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: covariance parameters of the fit. Read `scipy.optimize.curve_fit` for details.

**Return type** pOpt

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

`TOFU.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: covariance parameters of the fit. Read `scipy.optimize.curve_fit` for details.

**Return type** pOpt

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

`TOFU.fits.multipleGaussian2DFit(image, p0, bounds, TOF, plot=True, cropSize=6, tolerance=0.3, logNorm=False)`

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

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

## 1.1.6 TOFU.jupyterTools module

`TOFU.jupyterTools.disp(idx, symObj)`

`TOFU.jupyterTools.update_progress(progress, clear=True, comment="")`

## 1.1.7 TOFU.numberOfAtoms module

`TOFU.numberOfAtoms.numAtomsBlue(image, delta, s=0, plot=True, p0=None, bounds=[(), ()])`

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)

`TOFU.numberOfAtoms.numAtomsRed(image, delta, 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)

TOFU.numberOfAtoms.**temperature** (*sizes, timeStamps, plot=True*)

Calculates temperature of the cold gas given size of the cloud at various times of flight.

**Parameters**

- **sizes** – 2D array, sizes of the cloud extracted from gaussian2D fit in pixel units. Eg: ([[x1, y1], [x2, y2], [x3, y3]])
- **timeStamps** – array, corresponding times of flight in s.
- **plot** – bool, to plot  $v^2$  vs  $TOF^2$ . Default: True.

**Returns** a tuple, (temperature in K along x, temperature in K along y)

### 1.1.8 TOFU.picoMatTools module

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

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

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

### 1.1.9 TOFU.sigma module

TOFU.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.

TOFU.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.

### 1.1.10 TOFU.spectrum module

`TOFU.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.

`TOFU.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  $\mu K$ ), s]
- **bounds** – bounds for the fit as ([lower bounds], [upper bounds])

**Returns** (pOpt, pCov)

**Return type** a tuple with optimized parameters and covariance ex

`TOFU.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

`TOFU.spectrum.spectroscopyFaddeva(ODimages, f, imaging_params, plot=True, atom_loss=False, 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** – ODimagess 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  $b_v(\delta)$

**1.1.11 Module contents**



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### t

TOFU, [12](#)  
TOFU.AndorSifReader, [3](#)  
TOFU.fits, [8](#)  
TOFU.Images, [5](#)  
TOFU.jupyterTools, [9](#)  
TOFU.numberOfAtoms, [9](#)  
TOFU.Physics, [3](#)  
TOFU.Physics.allenDeviation, [1](#)  
TOFU.Physics.basisMatrices, [1](#)  
TOFU.Physics.createAndDestroy, [2](#)  
TOFU.Physics.spinInDickeBasis, [2](#)  
TOFU.picoMatTools, [10](#)  
TOFU.sigma, [10](#)  
TOFU.spectrum, [11](#)