ALC SUTOR

Release 0.1

P.E. Trevisanutto

Jun 30, 2024

CONTENTS:

CHAPTER

ONE

CLASS_NIST MODULE

class Nist_interpolation. Class for interpreting the Nist f1 and f2 files to calculate the ELF after 100 eV. Created on Fri Jul 21 11:32:35 2023 ALC Sutor Project

class class_nist.**Nist_interpolation**(*utilities*)

Bases: object

File_TDDFT = ''

Get_Core_elfi_Nist(*file*)

Method that interpolates the form factors f1 or f2 and plots the results

Parameters
file – name of the Nist file

Get_Core_f1_f2_Henke(*file*)

Get_Core_f1_f2_Nist(*file*)

Get_Core_f1_f2_Nist_interpolated(*file*)

Interpolating_Nist(*filename*)

Method that interpolates the form factors f1 or f2 and plots the results

Parameters
filename (*string*) – name of the Nist file

N_mix = 0

calc_ELF_compound_Nist()

Method for the ELF spectrum generation from the Nist f1 and f2 form factors

Returns
omega frequency (eV) ELF (float): ELF values

Return type
om (float)

n_k2elfi(*n*, *k*, *omg*, *name*)

reading_input()

Method to read the input file

transforming_f12__nk_henke_formula(*omega*, *rho*, *f1*, *f2*)

transforming_f1__n(*omega*, *rho*, *f1*, *N_i*, *atom*)

transforming_f2__k(*omega*, *rho*, *f2*, *N_i*, *atom*)

CHAPTER

TWO

CLASSUTILITY MODULE

Created on Fri Jul 21 11:32:35 2023 ALC Sutor Project Class utility

class with several plotting , maths and lmfit methods

class classutility.**utilities**(*Volume=5681.2801*)

Bases: `object`

Fano_Mau(*x*, *amplitude*, *center*, *sigma*, *q*, *alpha*)

Fano_function(*x*, *amplitude*, *center*, *sigma*, *q*)

method to Reconstruct Lmfit Fano function

Parameters

- **x** (*float*) – x-value
- **center** (*float*) – position
- **sigma** (*float*) – width
- **q** (*float*) – q asymmetric parameter

Returns

y (float)= Fano function y value

From_str2Integer(*str*)

method transforms string to integer :param str: string to transform :type str: string

Returns

rv (float)= integer value

From_str2float(*str*)

method transforms string to floats :param str: string to transform :type str: string

Returns

rv (float)= float value

Integration(*omega*, *imels*)

Method for ELF integration

Args:

filename (string): file

Returns:

Xeas (list) Yeas (list)

Mau_Lorentizian(*x*, *mu*, *a*, *gam*)

method to Reconstruct Asymmetric Lorentzian :param x: x-value :type x: float :param mu: Position :type mu: float :param a: Amplitude :type a: float :param gam: width :type gam: float

Returns

y (float)= Lorentzian y value

Mau_Lorentizian_F(*x*, *mu*, *a*, *gam*, *B*)

Mau_Lorentizian_h(*x*, *x0*, *a*, *gam*, *gam_r*)

method to Reconstruct Asymmetric Lorentzian :param x: x-value :type x: float :param a: Amplitude :type a: float :param x0: Position :type x0: float :param gam: width :type gam: float :param gam_r: asymmetric width :type gam_r: float

Returns
y (float)= Lorentzian y value

Polynomial(*x*, *coeff*)

method to Reconstruct Lmfit Polynomial function :param x: x-value :type x: float :param coeff: list of coefficients :type coeff: list

Returns
y (float)= Polynomial y value

Power_law(*x*, *par*)

method to Reconstruct Lmfit Power law function :param x: x-value :type x: float :param par: list of coefficients :type par: list

Returns
y (float)= Lorentzian y value

Power_law_h(*x*, *x_in*, *x_fin*, *Ampl*, *coeff*)

eV2cm(*omg*)

method transforming eV to cm-1

Parameters
omg (*float*) – frequency (eV)

Returns
rho_np (float)= frequency (cm-1)

get_Avogadro()

method get Avogadro value

get_hc()

method get Planck constant X Light speed value

get_henke()

method get constant for Henke calculations

index_of(*arrval*, *value*)

Method that Returns the index of an array *at or below* value.

Parameters

- **arrval** (*numpy array*)
- **value** – chosen value

is_number(*s*)

method to detect if the variable is a number :param omg: frequency (eV) :type omg: float

Returns
rho_np (float)= frequency (cm-1)

letsstart()

method let's start

lognormal_function(*x*, *A*, *mu*, *sigma*)

lorentzian(*x*, *a*, *x0*, *gam*)

method to Reconstruct Lmfit Lorentzian function :param x: x-value :type x: float :param a: Amplitude :type a: float :param x0: Position :type x0: float :param gam: width :type gam: float

Returns
y (float)= Lorentzian y value

multi_Fano(*x*, *par_fano*)

multi_lorentz(*x*, *par_q*, *par_pl*, *par_l*, *par_f*, *approx*)

method to sum the Lorentzians functions :param x: x-value :type x: float :param par_q: parameters for quadratic function :type par_q: list :param par_pl: parameters for polynomial function (not used) :type par_pl: list :param par_l: parameters for Lorentzians :type par_l: list :param par_f: parameters for several different functions (Fermi_lorentzian, Fano, LogNormal) :type par_f: list :param approx: approximation ("Fermi_lorentz", "Fano", "LN") :type approx: string

Returns
y (float)= reconstructed spectrum

multi_lorentz_h(*x*, *par_f*)

plot_data2(*Ysym_data*, *Yasym_data*, *X_data*, *save*)

General method for plotting three columns

Args:
Ysym_data (float): First ordinate Yasym_data (float): Second ordinate X_data (float): abscissa

save (boolean): true if file .dat saved

plotting_elf(*xData*, *yData*, *peaks*)

method that produces the ELF plot

Args:
xData (float): Energy (eV) yData (float): ELF

Returns:
plot

plotty(*x*, *y*, *name*, *color='blue'*, *scale='linear'*, *ylabel='IMFP ($\\AA$)'*, *save=True*)

General method for plotting

Args:
x (float): abscissa y (float): ordinate color (string): color of line scale (string): linear, semilog, loglog ylabel (string): label for ordinate save (boolean): true if file .dat saved

Returns:
plot and file .dat

powerlaw_fano(*x*, *par_pl*, *par_f*)

method to sum the Fano with lmfit powerlaw function :param x: x-value :type x: float :param par_pl: powerlaw parameters :type par_pl: list :param par_f: Fano parameters :type par_f: list

Returns
y (float)= quadratic y value

powerlaw_fermi(*x*, *par_pl*, *par_f*)

method to sum the fermi_lorentz with lmfit powerlaw function :param x: x-value :type x: float :param par_pl: powerlaw parameters :type par_pl: list :param par_f: Fermi_lorentzian parameters :type par_f: list

Returns
y (float)= quadratic y value

powerlaw_ln(*x*, *par_pl*, *par_f*)

method to sum the lognorm with lmfit powerlaw function :param x: x-value :type x: float :param par_pl: powerlaw parameters :type par_pl: list :param par_f: lognorm parameters :type par_f: list

Returns
y (float)= quadratic y value

powerlaw_lorentz(*x*, *par_pl*, *par_f*)

method to sum the lorentzian with lmfit powerlaw function :param x: x-value :type x: float :param par_pl: powerlaw parameters :type par_pl: list :param par_f: lorentzian parameters :type par_f: list

Returns
y (float)= quadratic y value

quadratic(*x*, *a*, *b*, *c*)

method to Reconstruct quadratic lmfit function :param x: x-value :type x: float :param a: x^2 coefficient :type a: float :param b: x coefficient :type b: float :param c: Constant :type c: float

Returns
y (float)= quadratic y value

reading_inputfile()

resultbestfitplot(*x*, *y*, *init*, *result*, *name*)

save_all(*x*, *y*, *name*)

method for saving files with name (two columns)

Args:
x (float): abscissa y (float): ordinate name (string): Name of saved file

save_all2(*x*, *y1*, *y2*, *name*)

method for saving files with name (three columns)

Args:
x (float): abscissa y1 (float): first ordinate y2 (float): second ordinate name (string): Name of saved file

save_pinel(*x*, *y*, *name*)

method for saving Cumulative Probabilities files with name (two columns)

Args:
x (float): abscissa y (float): ordinate name (string): Name of saved file

sutor()

method It is over

sutor_issue(*cissue*)

method to show an error

taking_data(*filename*)

Method for storing abscissa and ordinate in two lists

Args:
filename (string): file

Returns:
Xeas (list) Yeas (list)

taking_initialguess(*xData*, *yData*, *height*, *Guessing*)

Method for the determination of the initial guess for the lmfit (Lorentzian) parameter

Args:

xData (float): energy (eV) yData (float): ELF height (float): height sensitivity to individuate peaks Guessing (boolean): True lmfit find_peaks looks for parameter peaks, False it takes the parameter from file peaks.dat

Returns:

peas (integer): position in the ELF file rough_peak_positions (float) : guessed peak position Amp (float) : guessed peak heights gamm (float): guessed peak widths

CHAPTER

THREE

ELF_NIST_MAIN MODULE

CHAPTER

FOUR

CLASS_FIT_SPECTRUM_W_LORENTZIANS MODULE

Created on Tue May 2 17:03:01 2023 ALC_Sutor Project Class for fitting the ELF spectrum with lorentians and other lmfit funcitions Import lmfit model classes

Author: Paolo Emilio Trevisanutto (STFC-UKRI)

class class_fit_spectrum_w_lorentzians.**fitting_with_lorentzians**(*utilities*)

Bases: object

Created on Tue May 2 17:03:01 2023 ALC_Sutor Project Class for fitting the ELF spectrum with lorentians and other lmfit funcitions Import lmfit model classes

Author: Paolo Emilio Trevisanutto (STFC-UKRI)

add_peak(*prefix*, *center*, *amplitude*, *sigma*)

add_peak_ExponentialGaussianModel(*prefix*, *amplitude*, *center*, *sigma*, *gamma=0.1*)

add_peak_FanoModel(*prefix*, *center*, *amplitude*, *sigma*, *q=1*)

Method that creates parameters for the lmfit Fano Model modified

Parameters

- **prefix** (*string*) – Name of the Fano model
- **center** (*float*) – position function
- **amplitude** (*float*) – height for Fano function
- **sigma** (*float*) – width
- **q** – q for asymmetry of the Fano function

add_peak_FanoModel_Mau(*prefix*, *center*, *amplitude*, *sigma*, *q*, *al*)

Method that creates parameters for the lmfit Fano Model modified

Parameters

- **prefix** (*string*) – Name of the Fano model
- **center** (*float*) – position function
- **amplitude** (*float*) – height for Fano function
- **sigma** (*float*) – width
- **q** – q for asymmetry of the Fano function

add_peak_GaussianModel(*prefix*, *amplitude*, *center*, *sigma*)

Method that creates parameters for the lmfit gaussian Model

Parameters

- **prefix** (*string*) – Name of the lmfit model
- **center** (*float*) – position function
- **amplitude** (*float*) – height
- **sigma** – width

add_peak_GaussianlogModel(*prefix*, *center*, *amplitude*, *sigma*)

Method that creates parameters for the lmfit lognorm Model

Parameters

- **prefix** (*string*) – Name of the lmfit model
- **center** (*float*) – position function
- **amplitude** (*float*) – height
- **sigma** – width

add_peak_Mau(*prefix*, *center*, *amplitude*, *sigma*)

Method that creates parameters for a Lorentzian Model

Parameters

- **prefix** (*string*) – Name of the lmfit model
- **center** (*float*) – position function
- **amplitude** (*float*) – height
- **sigma** – width

add_peak_Mau_h_lor(*prefix*, *center*, *amplitude*, *sigma*, *sigma_r*)

Method that creates parameters for the lmfit Asymmetric Lorentzian Model

Parameters

- **prefix** (*string*) – Name of the lmfit model
- **center** (*float*) – position function
- **amplitude** (*float*) – height
- **sigma** (*float*) – width left from the center
- **sigma_r** – width right from the center

add_peak_PowerL_h(*prefix*, *xin*, *xfin*, *A*, *c=1*)

add_peak_fermi(*prefix*, *center*, *amplitude*, *sigma*, *B*)

add_peak_h_lor(*prefix*, *center*, *amplitude*, *sigma*, *sigma_r*)

CHAPTER

FIVE

CLASS_IMFP MODULE

Created on Tue May 30 11:07:59 2023

ALC_Sutor Project Class for calculations of Inelastic Mean Free Path and Comulative Probabilities for the IMFP at different initial kinetic energies.

Authors: P.E. Trevisanutto,

class class_IMFP.**class_imfp**(*EF*, *Eb*, *Tmax*, *cu*, *cutoff1*, *cutoff2*)

Bases: object

Plotting_analytical_vs_numerical(*filename*)

Method that plots the converged reconstructed ELF vs TD-DFT+Nist ELF

Parameters
filename (*string*) – imported ELF file

diimfp1(*Emin*, *Emax*, *T*, *par_q*, *par_p*, *par_l*, *par_f*, *par_pl1*, *par_f1*, *par_pl2*, *par_f2*, *Approximation*)

Method that integrates the differential cross section

Parameters

- **k** (*float*) – k momentum value
- **E** (*float*) – frequency to integrate (eV)
- **T** (*float*) – Initial electron kinetic energy (eV)
- **par_q** (*list*) – parameters for the quadratic lmfit function
- **par_p** (*list*) – parameters for the polynomial lmfit function
- **par_l** (*list*) – parameters for the lorentzian function
- **par_f** (*list*) – parameters for the Fermi-lorentzian function
- **par_pl1** (*list*) – parameters for the power law lmfit function in section 1
- **par_f1** (*list*) – parameters for the Fermi lorentizian lmfit function in section 1
- **par_pl2** (*list*) – parameters for the power law lmfit function in section 2
- **par_f2** (*list*) – parameters for the Fermi lorentizian lmfit function in section 2
- **Approximation** (*string*) – Approximation for the Nist tail

Returns
cross section for given value

Return type
y (float)

diimfp2(*Emin*, *Emax*, *T*, *par_pl*, *par_f*, *Approximation*)

diimfp3(*Emin*, *Emax*, *T*, *par_pl*, *par_f*, *Approximation*)

evolution_Fano(*q*, *par_f*)

Method to extend the lmfit Fano function to the finite k-momentum following the Drude-Lorentz dispersion law.

Parameters

- **q** (*float*) – k momentum value
- **par_f** (*list*) – list of parameters for the Fano peak

Returns
list of parameter extended with k momentum

Return type
parati_l (list)

evolution_LogNorm(*q*, *par_f*)

Method to extend the lmfit Log Norm function to the finite k-momentum following the Drude-Lorentz dispersion law.

Parameters

- **q** (*float*) – k momentum value
- **par_f** (*list*) – list of parameters for the LogNorm peak

Returns
list of parameter extended with k momentum

Return type
parati_l (list)

evolution_Lor(*q*, *par_f*)

Method to extend the Lorentzians created with lmfit model to the finite k-momentum following the Drude-Lorentz dispersion law.

Parameters

- **q** (*float*) – k momentum value
- **par_f** (*list*) – list of parameters for the Lorentzian peak

Returns
list of parameter extended with k momentum

Return type
parati_l (list)

evolution_Maur(*q*, *par_f*)

Method to extend the lmfit Lorentzians to the finite k-momentum following the Drude-Lorentz dispersion law.

Parameters

- **q** (*float*) – k momentum value
- **par_f** (*list*) – list of parameters for the Lorentzian peak

Returns
list of parameter extended with k momentum

Return type
parati_l (list)

evolution_Maur_F(*q*, *par_f*)

Method to extend the lmfit Lorentzians with Fermi distributionto the finite k-momentum following the Drude-Lorentz dispersion law.

Parameters

- **q** (*float*) – k momentum value
- **par_f** (*list*) – list of parameters for the Lorentzian peak

Returns
list of parameter extended with k momentum

Return type
parati_l (list)

`final_elf = []`

`generate_array`(*W_min*, *W_max*, *step1=5*, *step2=10*, *step3=100*, *step4=1000*, *step5=10000*)

Method to sample a list of energy value to be used to calculate the Cumulative Probabilities

Parameters

- **W_min** (*float*) – minimum energy (eV)
- **W_max** (*float*) – maximum energy (eV)
- **step1** (*float*) – First step value (eV)
- **step2** (*float*) – second step value (eV)
- **step3** (*float*) – third step value (eV)
- **step4** (*float*) – Fourth step value (eV)
- **step5** (*float*) – Fift step value (eV)

Returns
list of selected energies

Return type
list_w (list)

`get_Wmin`()

Method to retrieve the W energy minimum

Returns
energy minimum

Return type
__Wmin (float)

`get_digit`()

Method to retrieve the digit

Returns
Default digit

Return type
__digit (integer)

`inel_mean_free_path`(*par_q*, *par_p*, *par_l*, *par_f1*, *par_pl2*, *par_f2*, *par_pl3*, *par_f3*, *cApprox*)

Method that calculates the inelastic mean free path.

Parameters

- **par_q** (*list*) – parameters for the quadratic lmfit function
- **par_p** (*list*) – parameters for the polynomial lmfit function
- **par_l** (*list*) – parameters for the lorentzian function
- **par_f** (*list*) – parameters for the Fermi-lorentzian function
- **par_pl1** (*list*) – parameters for the power law lmfit function in section 1
- **par_f1** (*list*) – parameters for the Fermi lorentzian lmfit function in section 1

- **par_pl2** (*list*) – parameters for the power law lmfit function in section 2
- **par_f2** (*list*) – parameters for the Fermi lorentizian lmfit function in section 2
- **par_pl3** (*list*) – parameters for the power law lmfit function in section 3
- **par_f3** (*list*) – parameters for the Fermi lorentizian lmfit function in section 3
- **Approximation** (*string*) – Approximation for the Nist tail

integral_step(*W_in*, *Wmax*, *T*, *par_q*, *par_p*, *par_l*, *par_f1*, *par_pl2*, *par_f2*, *par_pl3*, *par_f3*, *cApprox*)

Method that integrates for a given initial T the Cumulative probabilities.

Args:

W_min (float): Minimum energy in the integration W_max (float): Maximum energy in the integration T (float): Initial electronic kinetic energy par_q (list):parameters for the quadratic lmfit function par_p (list): parameters for the polynomial lmfit function par_l (list): parameters for the lorentzian function par_f (list): parameters for the Fermi-lorentzian function par_pl1 (list): parameters for the power law lmfit function in section 1 par_f1 (list): parameters for the Fermi lorentizian lmfit function in section 1 par_pl2 (list): parameters for the power law lmfit function in section 2 par_f2 (list): parameters for the Fermi lorentizian lmfit function in section 2 par_pl3 (list): parameters for the power law lmfit function in section 3 par_f3 (list): parameters for the Fermi lorentizian lmfit function in section 3 Approximation (string): Approximation for the Nist tail

Results:

Integral_tot (float): Results of the integration

integrand1(*kappa*, *E*, *T*, *par_q*, *par_p*, *par_l*, *par_f*, *par_pl1*, *par_f1*, *par_pl2*, *par_f2*, *Approximation*)

Method to create the cross section integrand

Parameters

- **k** (*float*) – k momentum value
- **E** (*float*) – frequency to integrate (eV)
- **T** (*float*) – Initial electron kinetic energy (eV)
- **par_q** (*list*) – parameters for the quadratic lmfit function
- **par_p** (*list*) – parameters for the polynomial lmfit function
- **par_l** (*list*) – parameters for the lorentzian function
- **par_f** (*list*) – parameters for the Fermi-lorentzian function
- **par_pl1** (*list*) – parameters for the power law lmfit function in section 1
- **par_f1** (*list*) – parameters for the Fermi lorentizian lmfit function in section 1
- **par_pl2** (*list*) – parameters for the power law lmfit function in section 2
- **par_f2** (*list*) – parameters for the Fermi lorentizian lmfit function in section 1
- **Approximation** (*string*) – Approximation for the Nist tail

Returns

cross section for given value

Return type

y (float)

integrand2(*kappa*, *E*, *T*, *par_pl*, *par_f*, *Approximation*)

integrand3(*kappa*, *E*, *T*, *par_pl*, *par_f*, *Approximation*)

om_max = 2000.0

om_min = 0.3

reading_background()

Method to read the background parameters (quadratic, polynomial or Power law)

Returns
number of the involved parameters

Return type
counter (float)

reading_first_sector()

Method to read the parameters in the first sector

Returns
number of the involved parameters

Return type
counter (float)

reading_parameters(*filename*)

Method to read the parameters for the lmft

Parameters
filename (*string*) – imported file

Returns
list of parameters for lmfit quadratic function self.__params_p (list): list of parameters for lmfit polynomial function self.__params_l(list): list of parameters for lmfit lorentz function self.__params_Nist1 (list):list of parameters for lmfit Lorentz (Fano, lognorm) function in the Nist region sector 1 self.__params_pl1 (list):list of parameters for lmfit power law function in the Nist region sector 1 self.__params_Nist2 (list):list of parameters for lmfit Lorentz (Fano, lognorm) function in the Nist region sector 2 elf.__params_pl2 (list): list of parameters for lmfit power law function in the Nist region sector 2 self.__params_Nist3 (list):list of parameters for lmfit Lorentz (Fano, lognorm) function in the Nist region sector 1 final_elf (array): Reconstructed ELF self.__cApprox (string): Approximation (Lorentz, Fano, LN) for Nist range of energy

Return type
self.__params_q (list)

reading_second_sector()

Method to read the parameters in the second sector

Returns
number of the involved parameters

Return type
counter (float)

reading_third_sector()

Method to read the parameters in the third sector

Returns
number of the involved parameters

Return type
counter (float)

set_approx(*appr*)

Method that sets the approximation

Parameters
approx (*String*) – Chosen Approximation

Returns
Approximation e.g. Lorentz, Fano, LN

Return type
__cApprox (string)

CHAPTER

SIX

CALCULATING_IMFP_MAIN MODULE

CHAPTER

SEVEN

FITTING_SPECTRUM_MAIN MODULE

CHAPTER

EIGHT

LMFIT MODULE

LMFIT: Non-Linear Least-Squares Minimization and Curve-Fitting for Python.

Lmfit provides a high-level interface to non-linear optimization and curve-fitting problems for Python. It builds on the Levenberg-Marquardt algorithm of *scipy.optimize.leastsq*, but also supports most of the other optimization methods present in *scipy.optimize*. It has a number of useful enhancements, including:

- Using Parameter objects instead of plain floats as variables. A Parameter has a value that can be varied in the fit, fixed, have upper and/or lower bounds. It can even have a value that is constrained by an algebraic expression of other Parameter values.
- Ease of changing fitting algorithms. Once a fitting model is set up, one can change the fitting algorithm without changing the objective function.
- Improved estimation of confidence intervals. While *scipy.optimize.leastsq* will automatically calculate uncertainties and correlations from the covariance matrix, lmfit also has functions to explicitly explore parameter space to determine confidence levels even for the most difficult cases.
- Improved curve-fitting with the Model class. This extends the capabilities of *scipy.optimize.curve_fit*, allowing you to turn a function that models your data into a Python class that helps you parametrize and fit data with that model.
- Many built-in models for common lineshapes are included and ready to use.

CHAPTER

NINE

TQDM MODULE

class tqdm.**TMonitor**(*tqdm_cls*, *sleep_interval*)

Bases: Thread

Monitoring thread for tqdm bars. Monitors if tqdm bars are taking too much time to display and readjusts miniters automatically if necessary.

Parameters

- **tqdm_cls** (*class*) – tqdm class to use (can be core tqdm or a submodule).
- **sleep_interval** (*float*) – Time to sleep between monitoring checks.

exit()

get_instances()

report()

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

exception tqdm.**TqdmDeprecationWarning**(*msg*, *fp_write=None*, **a*, ***k*)

Bases: *TqdmWarning*, DeprecationWarning

exception tqdm.**TqdmExperimentalWarning**(*msg*, *fp_write=None*, **a*, ***k*)

Bases: *TqdmWarning*, FutureWarning

beta feature, unstable API and behaviour

exception tqdm.**TqdmKeyError**

Bases: KeyError

exception tqdm.**TqdmMonitorWarning**(*msg*, *fp_write=None*, **a*, ***k*)

Bases: *TqdmWarning*, RuntimeWarning

tqdm monitor errors which do not affect external functionality

exception tqdm.**TqdmSynchronisationWarning**

Bases: RuntimeWarning

tqdm multi-thread/-process errors which may cause incorrect nesting but otherwise no adverse effects

exception tqdm.**TqdmTypeError**

Bases: TypeError

exception tqdm.**TqdmWarning**(*msg*, *fp_write=None*, **a*, ***k*)

Bases: Warning

base class for all tqdm warnings.

Used for non-external-code-breaking errors, such as garbled printing.

tqdm.**main**(*fp=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>*, *argv=None*)

9.1 Parameters (internal use only)

fp : file-like object for tqdm argv : list (default: sys.argv[1:])

tqdm.**tgrange**(**args*, ***kwargs*)

Shortcut for *tqdm.gui.tqdm(range(*args), **kwargs)*.

tqdm.**tnrange**(**args*, ***kwargs*)

Shortcut for *tqdm.notebook.tqdm(range(*args), **kwargs)*.

class tqdm.**tqdm**(*_, **__)

Bases: Comparable

Decorate an iterable object, returning an iterator which acts exactly like the original iterable, but prints a dynamically updating progressbar every time a value is requested.

Parameters

- **iterable** (*iterable, optional*) – Iterable to decorate with a progressbar. Leave blank to manually manage the updates.
- **desc** (*str, optional*) – Prefix for the progressbar.
- **total** (*int or float, optional*) – The number of expected iterations. If unspecified, len(iterable) is used if possible. If float("inf") or as a last resort, only basic progress statistics are displayed (no ETA, no progressbar). If *gui* is True and this parameter needs subsequent updating, specify an initial arbitrary large positive number, e.g. 9e9.
- **leave** (*bool, optional*) – If [default: True], keeps all traces of the progressbar upon termination of iteration. If *None*, will leave only if *position* is *0*.
- **file** (*io.TextIOWrapper* or *io.StringIO*, optional) – Specifies where to output the progress messages (default: sys.stderr). Uses *file.write(str)* and *file.flush()* methods. For encoding, see *write_bytes*.
- **ncols** (*int, optional*) – The width of the entire output message. If specified, dynamically resizes the progressbar to stay within this bound. If unspecified, attempts to use environment width. The fallback is a meter width of 10 and no limit for the counter and statistics. If 0, will not print any meter (only stats).
- **mininterval** (*float, optional*) – Minimum progress display update interval [default: 0.1] seconds.
- **maxinterval** (*float, optional*) – Maximum progress display update interval [default: 10] seconds. Automatically adjusts *miniters* to correspond to *mininterval* after long display update lag. Only works if *dynamic_miniters* or monitor thread is enabled.
- **miniters** (*int or float, optional*) – Minimum progress display update interval, in iterations. If 0 and *dynamic_miniters*, will automatically adjust to equal *mininterval* (more CPU efficient, good for tight loops). If > 0, will skip display of specified number of iterations. Tweak this and *mininterval* to get very efficient loops. If your progress is erratic with both fast and slow iterations (network, skipping items, etc) you should set miniters=1.

- **ascii** (*bool or str, optional*) – If unspecified or False, use unicode (smooth blocks) to fill the meter. The fallback is to use ASCII characters " 123456789#".
- **disable** (*bool, optional*) – Whether to disable the entire progressbar wrapper [default: False]. If set to None, disable on non-TTY.
- **unit** (*str, optional*) – String that will be used to define the unit of each iteration [default: it].
- **unit_scale** (*bool or int or float, optional*) – If 1 or True, the number of iterations will be reduced/scaled automatically and a metric prefix following the International System of Units standard will be added (kilo, mega, etc.) [default: False]. If any other non-zero number, will scale *total* and *n*.
- **dynamic_ncols** (*bool, optional*) – If set, constantly alters *ncols* and *nrows* to the environment (allowing for window resizes) [default: False].
- **smoothing** (*float, optional*) – Exponential moving average smoothing factor for speed estimates (ignored in GUI mode). Ranges from 0 (average speed) to 1 (current/instantaneous speed) [default: 0.3].
- **bar_format** (*str, optional*) – Specify a custom bar string formatting. May impact performance. [default: '{l_bar}{bar}{r_bar}'], where l_bar='{desc}: {percentage:3.0f}%|' and r_bar='| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, '

  '{rate_fmt}{postfix}]'

  Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt,
  percentage, elapsed, elapsed_s, ncols, nrows, desc, unit, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, postfix, unit_divisor, remaining, remaining_s, eta.

  Note that a trailing ": " is automatically removed after {desc} if the latter is empty.
- **initial** (*int or float, optional*) – The initial counter value. Useful when restarting a progress bar [default: 0]. If using float, consider specifying *{n:.3f}* or similar in *bar_format*, or specifying *unit_scale*.
- **position** (*int, optional*) – Specify the line offset to print this bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).
- **postfix** (dict or *, optional) – Specify additional stats to display at the end of the bar. Calls *set_postfix(**postfix)* if possible (dict).
- **unit_divisor** (*float, optional*) – [default: 1000], ignored unless *unit_scale* is True.
- **write_bytes** (*bool, optional*) – Whether to write bytes. If (default: False) will write unicode.
- **lock_args** (*tuple, optional*) – Passed to *refresh* for intermediate output (initialisation, iterating, and updating).
- **nrows** (*int, optional*) – The screen height. If specified, hides nested bars outside this bound. If unspecified, attempts to use environment height. The fallback is 20.
- **colour** (*str, optional*) – Bar colour (e.g. 'green', '#00ff00').
- **delay** (*float, optional*) – Don't display until [default: 0] seconds have elapsed.
- **gui** (*bool, optional*) – WARNING: internal parameter - do not use. Use tqdm.gui.tqdm(...) instead. If set, will attempt to use matplotlib animations for a graphical output [default: False].

Returns
out

Return type
decorated iterator.

`clear`(*nolock=False*)

Clear current bar display.

`close`()

Cleanup and (if leave=False) close the progressbar.

`display`(*msg=None*, *pos=None*)

Use *self.sp* to display *msg* in the specified *pos*.

Consider overloading this function when inheriting to use e.g.: *self.some_frontend(**self.format_dict)* instead of *self.sp*.

Parameters

- **msg** (str, optional. What to display (default: *repr(self)*).)
- **pos** (int, optional. Position to *moveto*) – (default: *abs(self.pos)*).

`classmethod external_write_mode`(*file=None*, *nolock=False*)

Disable tqdm within context and refresh tqdm when exits. Useful when writing to standard output stream

`property format_dict`

Public API for read-only member access.

`static format_interval`(*t*)

Formats a number of seconds as a clock time, [H:]MM:SS

Parameters
t (*int*) – Number of seconds.

Returns
out – [H:]MM:SS

Return type
str

`static format_meter`(*n*, *total*, *elapsed*, *ncols=None*, *prefix=''*, *ascii=False*, *unit='it'*, *unit_scale=False*, *rate=None*, *bar_format=None*, *postfix=None*, *unit_divisor=1000*, *initial=0*, *colour=None*, ***extra_kwargs*)

Return a string-based progress bar given some parameters

Parameters

- **n** (*int or float*) – Number of finished iterations.
- **total** (*int or float*) – The expected total number of iterations. If meaningless (None), only basic progress statistics are displayed (no ETA).
- **elapsed** (*float*) – Number of seconds passed since start.
- **ncols** (*int, optional*) – The width of the entire output message. If specified, dynamically resizes *{bar}* to stay within this bound [default: None]. If *0*, will not print any bar (only stats). The fallback is *{bar:10}*.
- **prefix** (*str, optional*) – Prefix message (included in total width) [default: ‘’]. Use as {desc} in bar_format string.
- **ascii** (*bool, optional or str, optional*) – If not set, use unicode (smooth blocks) to fill the meter [default: False]. The fallback is to use ASCII characters “ 123456789#”.
- **unit** (*str, optional*) – The iteration unit [default: ‘it’].

- **unit_scale** (*bool or int or float, optional*) – If 1 or True, the number of iterations will be printed with an appropriate SI metric prefix (k = 10^3, M = 10^6, etc.) [default: False]. If any other non-zero number, will scale *total* and *n*.
- **rate** (*float, optional*) – Manual override for iteration rate. If [default: None], uses n/elapsed.
- **bar_format** (*str, optional*) – Specify a custom bar string formatting. May impact performance. [default: '{l_bar}{bar}{r_bar}'], where l_bar='{desc}: {percentage:3.0f}%|' and r_bar='| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, '

  '{rate_fmt}{postfix}]'

  Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt,
  percentage, elapsed, elapsed_s, ncols, nrows, desc, unit, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, postfix, unit_divisor, remaining, remaining_s, eta.

  Note that a trailing ": " is automatically removed after {desc} if the latter is empty.
- **postfix** (*, optional) – Similar to *prefix*, but placed at the end (e.g. for additional stats). Note: postfix is usually a string (not a dict) for this method, and will if possible be set to postfix = ', ' + postfix. However other types are supported (#382).
- **unit_divisor** (*float, optional*) – [default: 1000], ignored unless *unit_scale* is True.
- **initial** (*int or float, optional*) – The initial counter value [default: 0].
- **colour** (*str, optional*) – Bar colour (e.g. 'green', '#00ff00').

Returns
out

Return type
Formatted meter and stats, ready to display.

`static format_num(n)`

Intelligent scientific notation (.3g).

Parameters
n (*int or float or Numeric*) – A Number.

Returns
out – Formatted number.

Return type
str

`static format_sizeof(num, suffix='', divisor=1000)`

Formats a number (greater than unity) with SI Order of Magnitude prefixes.

Parameters

- **num** (*float*) – Number (>= 1) to format.
- **suffix** (*str, optional*) – Post-postfix [default: ''].
- **divisor** (*float, optional*) – Divisor between prefixes [default: 1000].

Returns
out – Number with Order of Magnitude SI unit postfix.

Return type
str

classmethod get_lock()

Get the global lock. Construct it if it does not exist.

monitor = None

monitor_interval = 10

moveto(*n*)

classmethod pandas(***tqdm_kwargs*)

Registers the current *tqdm* class with
pandas.core. (frame.DataFrame | series.Series | groupby.(generic.)DataFrameGroupBy | groupby.(generic.)SeriesGroupBy).progress_apply

A new instance will be created every time *progress_apply* is called, and each instance will automatically *close()* upon completion.

Parameters
tqdm_kwargs (*arguments for the tqdm instance*)

Examples

```
>>> import pandas as pd
>>> import numpy as np
>>> from tqdm import tqdm
>>> from tqdm.gui import tqdm as tqdm_gui
>>>
>>> df = pd.DataFrame(np.random.randint(0, 100, (100000, 6)))
>>> tqdm.pandas(ncols=50)  # can use tqdm_gui, optional kwargs, etc
>>> # Now you can use `progress_apply` instead of `apply`
>>> df.groupby(0).progress_apply(lambda x: x**2)
```

References

<https://stackoverflow.com/questions/18603270/ progress-indicator-during-pandas-operations-python>

refresh(*nolock=False*, *lock_args=None*)

Force refresh the display of this bar.

Parameters

- **nolock** (*bool, optional*) – If *True*, does not lock. If [default: *False*]: calls *acquire()* on internal lock.
- **lock_args** (*tuple, optional*) – Passed to internal lock's *acquire()*. If specified, will only *display()* if *acquire()* returns *True*.

reset(*total=None*)

Resets to 0 iterations for repeated use.

Consider combining with *leave=True*.

Parameters
total (*int or float, optional. Total to use for the new bar.*)

set_description(*desc=None*, *refresh=True*)

Set/modify description of the progress bar.

Parameters

- **desc** (*str, optional*)

- **refresh** (*bool, optional*) – Forces refresh [default: True].

set_description_str(*desc=None*, *refresh=True*)

Set/modify description without ': ' appended.

classmethod set_lock(*lock*)

Set the global lock.

set_postfix(*ordered_dict=None*, *refresh=True*, ***kwargs*)

Set/modify postfix (additional stats) with automatic formatting based on datatype.

Parameters

- **ordered_dict** (*dict or OrderedDict, optional*)
- **refresh** (*bool, optional*) – Forces refresh [default: True].
- **kwargs** (*dict, optional*)

set_postfix_str(*s=''*, *refresh=True*)

Postfix without dictionary expansion, similar to prefix handling.

static status_printer(*file*)

Manage the printing and in-place updating of a line of characters. Note that if the string is longer than a line, then in-place updating may not work (it will print a new line at each refresh).

unpause()

Restart tqdm timer from last print time.

update(*n=1*)

Manually update the progress bar, useful for streams such as reading files. E.g.: >>> t = tqdm(total=filesize) # Initialise >>> for current_buffer in stream: ... ... t.update(len(current_buffer)) >>> t.close() The last line is highly recommended, but possibly not necessary if *t.update()* will be called in such a way that *filesize* will be exactly reached and printed.

Parameters

n (*int or float, optional*) – Increment to add to the internal counter of iterations [default: 1]. If using float, consider specifying *{n:.3f}* or similar in *bar_format*, or specifying *unit_scale*.

Returns

out – True if a *display()* was triggered.

Return type

bool or None

classmethod wrapattr(*stream*, *method*, *total=None*, *bytes=True*, ***tqdm_kwargs*)

stream : file-like object. method : str, "read" or "write". The result of *read()* and

the first argument of *write()* should have a *len()*.

```
>>> with tqdm.wrapattr(file_obj, "read", total=file_obj.size) as fobj:
...     while True:
...         chunk = fobj.read(chunk_size)
...         if not chunk:
...             break
```

classmethod write(*s*, *file=None*, *end='\n'*, *nolock=False*)

Print a message via tqdm (without overlap with bars).

class tqdm.**tqdm_gui**(*_, **__*)

Bases: *tqdm*

Experimental Matplotlib GUI version of tqdm!

clear(*_, **__)

Clear current bar display.

close()

Cleanup and (if leave=False) close the progressbar.

display(*_, **__)

Use *self.sp* to display *msg* in the specified *pos*.

Consider overloading this function when inheriting to use e.g.: *self.some_frontend(**self.format_dict)* instead of *self.sp*.

Parameters

- **msg** (str, optional. What to display (default: *repr(self)*).)
- **pos** (int, optional. Position to *moveto*) – (default: *abs(self.pos)*).

tqdm.**tqdm_notebook**(**args*, ***kwargs*)

See tqdm.notebook.tqdm for full documentation

tqdm.**tqdm_pandas**(*tclass*, ***tqdm_kwargs*)

Registers the given *tqdm* instance with *pandas.core.groupby.DataFrameGroupBy.progress_apply*.

tqdm.**trange**(**args*, ***kwargs*)

Shortcut for tqdm(range(*args), **kwargs).

CHAPTER

TEN

COLORAMA MODULE

CHAPTER

ELEVEN

NIST

11.1 conf module

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

l

t