To be presented at WCPEC8 in Milan Sep 2022

ADDING THE MLFM TO PVPMC/PVLIB

Steve Ransome[1] and Juergen Sutterlueti[2]
[1]Steve Ransome Consulting Ltd, #99 KT2 6AF, U.K. mailto:steve@steveransome.com
[2]Gantner Instruments GmbH, 6780 Schruns, Austria

ABSTRACT: PVPMC/PVLIB is a global collaborative project headed by Sandia to develop a verified open-source PV modelling library in python. The MLFM (Mechanistic Loss Factors Model) combines the Loss Factors Model LFM (orthogonal, normalised loss calculations from MPPT measurements, iv curves or IEC 61853-like matrices) with the Mechanistic Performance Model MPM (robust, orthogonal, normalised fitting coefficients to LFM losses). The MLFM is being added to PVLIB, this paper describes the new functionality as shown by these models and how to interpret the graphs and datafits. This paper also describes some of the procedures used to add code to PVLIB and aims to encourage others to submit their own PV modelling code following PVLIB's styles, nomenclature, naming conventions, formatting, guidelines, designs, tests etc.
Keywords: Energy Rating; Energy Performance; Modelling

1 INTRODUCTION

The MLFM equations and procedures have proven accurate and useful in PV module measurement characterization, identifying performance limits and loss mechanisms, deriving performance and temperature coefficients, analyzing degradation rates and causes and/or seasonal annealing, and comparing fit accuracies with other common models [1][2][3].

Their functionality is now being added to PVLIB which is the industry leading python open-source modelling library.

MLFM modelling algorithms can be used with existing PVLIB algorithms such as atmospheric data, TMY, solar position and angle of incidence, tracking, inverter modelling and the performance compared with other models in PVLIB (such as the 1-diode model, PVUSA and SAPM. [2])

The sequence, methods and procedures used to add code has been documented to help others contribute.

MLFM algorithms have been added as a Jupiter Notebook tutorial 'mlfm.ipynb' with a library file 'mlfm.py' to PVLIB The code is available as a downloadable python tutorial mlfm.ipynb in PVLIB[]. Three sample data files contain either indoor measurements (CFV IEC 61853) or outdoor (NREL and Gantner Instruments) with 4 or 6 datapoints.

A flow chart of the mlfm procedures added to PVLIB including graphs as from the tutorial is shown in figure 1.

Each box is identified by a letter [A] to [K] for later discussion in the paper.

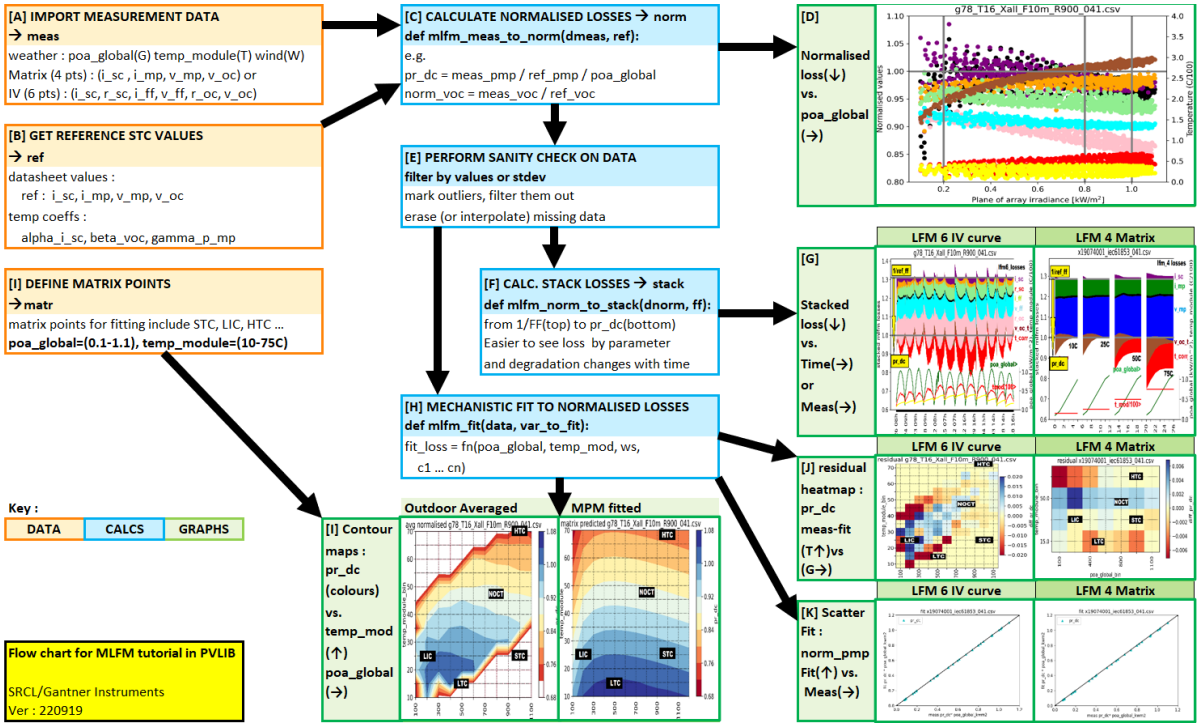


Figure 1: MLFM Flow chart of methodology and graphs being added to PVLIB

2 FURTHER DETAILS

The Loss Factors Model derives meaningful, orthogonal, and normalised loss factors from the shape of IV curves compared with reference STC values as

illustrated in figure 2.

These coefficients show the magnitude of losses attributable to each of the factors i_sc to v_oc.

The number of normalised loss factors depends on whether measurements of iv parameters include r_sc and r_oc slopes (1/dI/dV at i_sc or v_oc respectively).

The product of the loss coefficients equation (1) multiplies the datasheet module ref_p_mp point ① to measured meas_pmp point ⑥ using the equation for pr_dc6 (with r_sc and r_oc as in a full iv curve) or pr_dc4 (without r_sc and r_oc e.g. as with an IEC 61853 matrix).

(Spectral and reflectivity vs. aoi are not included here).

Normalised LFM loss equation:

$$pr\_dc_n = \frac{1}{ref\_ff} \times \prod_{i=1}^{n} norm\_lfm_{(i)} \quad (1)$$

Where:

$norm\_lfm_{(1..6)}$ = ['i_sc','r_sc','i_ff','v_ff','r_oc','v_oc']

or

$norm\_lfm_{(1..4)}$ = ['i_sc', 'i_mp', 'v_mp', 'v_oc']

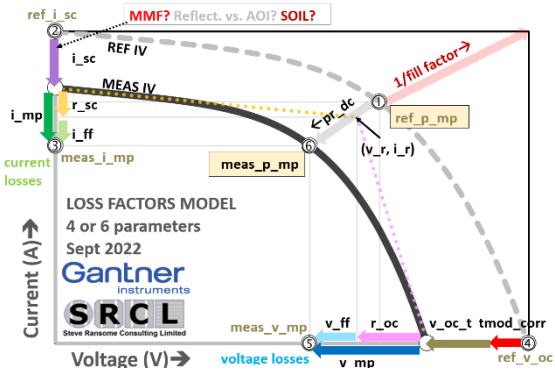


Figure 2: Loss Factors model using 4 or 6 normalised losses (plus optional temperature correction on v_oc) to calculate to ⑥meas_p_mp from ①ref_p_mp.

The names and standard LFM colours of the losses are listed in table I.

Table I: LFM_6 and LFM_4 parameters and colours (with optional temperature correction on v_oc).

LFM_6 e.g. IV curves (with r_sc and r_oc) # Param	Colour	LFM_4 e.g. IEC 61853 matrix (no r_sc, no r_oc) # Param	Colour
Ref Lossless =① * 1/ref_ff		Ref Lossless =① * 1/ref_ff	
i_sc	Purple	i_sc	Purple
r_oc	Orange	i_mp	Green
i_ff	Lime	v_mp	Blue
v_ff	Cyan		
r_oc	Pink		
v_oc(_t)	Brown	v_oc(_t)	Brown
(t_corr)	Red	(t_corr)	Red
⑥ Meas = Pr_dc		⑥ Meas = Pr_dc	

3 MLFM PROCEDURES AND GRAPHS

Each box and graph from figure 1 is identified by letter [A] to [K] and is described with some python code below.

Sample python code from pvlib is included but its whitespace (tabbing) may have been modified for clarity.

Note: The code within pvlib is correctly formatted.

2.1 [A] 'Import measurement data → meas'

The MLFM analysis process imports measured weather and iv point or slope data "-1/(di/dv)" from csv files into a dataframe 'dmeas' with the following essential variable names which follow the pvlib naming convention where possible.

Optional parameters such as 'temp_air' can be included but aren't analysed as standard.

```
# Python code [A]

dmeas :
# essential measured values
#                [unit]  # comment
# weather
    poa_global   [W/m^2]
    temp_module  [C]
    wind_speed   [ms^-1] # 0 if indoor
# iv points
    i_sc         [A]
    v_oc         [V]
    i_mp         [A]
    v_mp         [V]
    p_mp         [W]    # p_mp = i_mp * v_mp
# optional
    r_sc         [Ohm] #-1/IVslope@Isc
    r_oc         [Ohm] #-1/IVslope@Voc
```

Three sample data sources (Gantner 6 parameter and NREL 4 parameter outdoor plus an indoor CFV Matrix 4 parameter) are included but more can easily be added in the required meas csv format above.

2.2 [B] 'Get reference stc values → ref'

STC Reference data 'ref' (from datasheets) is imported for the module chosen in (A) again following pvlib naming conventions.

```
# Python code [B]

ref :
# Essential Reference values at STC.
#                [unit]  # comment
# electrical
    i_sc         [A]
    i_mp         [A]
    v_mp         [V]
    v_oc         [V]
    p_mp         [W]     # = i_mp * v_mp
# temperature coefficients
    gamma_p_mp   [1/C]   # pmp temp. coeff.
    beta_v_oc    [1/C]   # voc temp. coeff.
    alpha_i_sc   [1/C]   # isc temp. coeff.
```

2.3 [C] 'Calculate multiplicative normalised losses → norm'

Normalised multiplicative losses are calculated from meas and ref using equation (1) for either LFM_6 or LFM_4 parameter measurements.

Voltage values are normalised as 'norm=meas/ref', Current values are normalised as 'norm= meas/ref/irradiance_suns'.

Further equations are used to derive normalised losses from the r_sc and r_oc using i_r and v_r which is the intercept of the r_sc and r_oc lines in figure 2.

```
# Python code [C]

def mlfm_meas_to_norm(dmeas, ref):
```

```
# Convert measured power, current and voltage to
# normalized values e.g.

dnorm['i_sc'] = dmeas['i_sc'] \
    / (dmeas['poa_global'] / G_STC) / ref['i_sc']

dnorm['v_oc'] = dmeas['v_oc'] / ref['v_oc']

# where
# i_r = ((i_sc*r_sc - v_oc)/(r_sc - r_oc))
# v_r = ((r_sc*(v_oc - i_sc*r_oc)/(r_sc - r_oc)))

if lfm_6: # full iv curve with r_sc and r_oc
    # calculate normalised resistances r_sc, r_oc

    dnorm['r_sc'] = i_r / dmeas['i_sc']
    dnorm['r_oc'] = v_r / dmeas['v_oc']

    # calculate remaining fill factor losses
    # partitioned to i_ff, v_ff

    dnorm['i_ff'] = dmeas['i_mp'] / i_r
    dnorm['v_ff'] = dmeas['v_mp'] / v_r

if lfm_4: # matrix no r_sc or r_oc
    dnorm['i_mp'] = dmeas['i_mp'] / dmeas['i_sc']
    dnorm['v_mp'] = dmeas['v_mp'] / dmeas['v_oc']
…
return dnorm
```

2.4 **[D] 'Graph of normalised lfm : loss(↓) vs. poa_global(→)'**

Figure 3 illustrates the 6 different normalised loss parameters for a cSi module at Gantner Instruments' OTF in Tempe, AZ vs. poa_global.

Loss parameters i_ff, v_ff, r_sc and r_oc are all quite narrow and smooth indicating good module behaviour and well measured. The LFM analysis can often identify temperature coefficients of terms such as r_sc and r_oc [JAPAN Tues]

i_sc is a little more widely scattered (as it has uncorrected soiling and spectral losses, particularly at lower light levels where aoi/reflectivity differ from clear skies to diffuse).

Because v_oc is quite temperature sensitive it is usually better plotted temperature corrected by beta_v_oc.

```
# Python code [D]

dnorm['v_oc_temp_corr'] = dnorm['v_oc'] \
    * (1 - ref['beta_v_oc'] \
    * (dmeas['temp_module'] - T_STC))
```

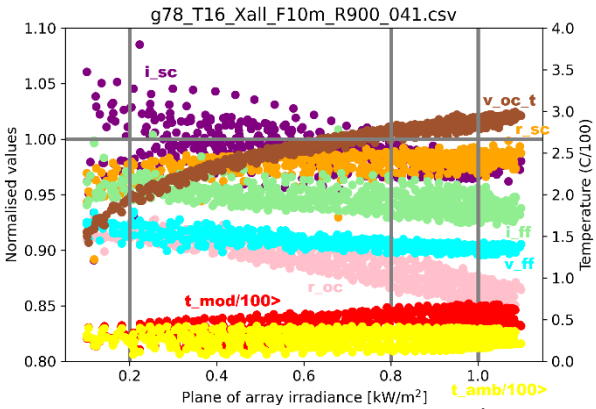


Figure 3: Normalised LFM_6 losses(↓) vs. Irradiance(→) typical c-Si module at Tempe, AZ.

Pr_dc is the product of all these loss parameters as in equation (1). Therefore if any of the coefficients change with irradiance or temperature it will affect the overall pr_dc performance and can be quantified as in figure 3 .

- Low light pr_dc drop(↙) is caused by r_sc (r_shunt) and v_oc (which depends on ln(poa_global)).
- High light pr_dc drop(↘) is caused by r_oc (r_series).
- Coefficients that are almost 'flat' with irradiance = i_ff, v_ff.

2.5 **[E] 'Perform sanity checks on data'**

Normalised data should be checked for missing values, errors, and outliers.

Both normalised limits (e.g. '0.7 < pr_dc < 1.3') and standard deviations (e.g. < 3σ) can be used for outdoor data, whereas normalised indoor matrix measurements would likely be remeasured if there were any outliers.

The limits needed will depend on the accuracy and variability of the measurements.

```
# Python code [E]

# remove values outside limits e.g. <0.5 or >1.5

norm = norm[((norm['pr_dc'] > 0.5) &
             (norm['pr_dc'] < 1.5))]

# remove all mlfm values outside x~3 stdevs
if qty_mlfm_vars == 6:
    stdevs = 3
    # remove data > x stdevs
    for lfm in ('i_sc','r_sc','i_ff',
                'v_ff','r_oc','v_oc'):
        norm = norm[
            ((norm[lfm] - norm[lfm].mean()) /
                 norm[lfm].std()).abs() < stdevs
        ]
```

2.6 **[F] 'Calculate stacked losses → stack (subtractive)'**

Plots of multiplicative LFM losses (1) as shown in figure 3 overlap, sometimes it's difficult to discern relative losses and changes.

It can be useful to mathematically transform them to subtractive losses which can be represented more easily on a stack plot as in figs 4 and 5 using equation (2).

```
# Python code [F]

def mlfm_norm_to_stack(dnorm, fill_factor):
'''Converts normalised values to stacked
    subtractive normalized losses.
    Normalized values can reveal losses via
    scatter plots vs. irradiance or temperature.
    Stacked subtractive losses can show relative
    loss proportions.
    Stacked losses partition the difference
    between the normalized power and the power
    that corresponds to the reference fill
    factor. '''
…
return dstack
```

Stack loss equation

$$pr\_dc_n = \frac{1}{ff} - \sum_{i=1}^{n} stack\_lfm_{(i)} \qquad (2)$$

Where:

$stack\_lfm_{(1..6)}$ =
`['i_sc','r_sc','i_ff','v_ff','r_oc','v_oc']`

or

$stack\_lfm_{(1..4)}$ =
`['i_sc', 'i_mp', 'v_mp', 'v_oc']`

2.7 **[G] 'Plot stacked subtractive lfm : loss(↓)**

vs. time(outdoor) or measurement(indoor)(→)'

Stacked losses for 6 and 4 parameters are indicated by their colours from top to bottom as listed in Table I. The height of each loss is proportional to its magnitude. Loss colours are from Table I.

Losses fall from from 1/ref_ff down to pr_dc. Irradiance (green) and temp_mod/100 (red) are shown both on the right y axis.

Figure 4 (left) plots one clear day a month (Jan – Dec for a year) for a Gantner c-Si module in Tempe AZ. There are six independent losses plus temperature corrections in a stacked format from a lossless limit 1/ref_ff (top), subtracting each loss value in turn until it reaches pr_dc (bottom).

```
# Python code [G]
```

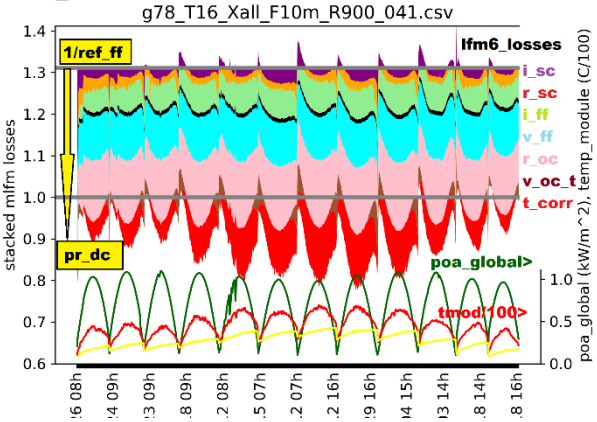
```
# Stacked loss values(y) vs. date and time
#(or matrix measurement)(x)

fig_stack = plot_mlfm_stack(
    dmeas=meas, dnorm=norm, dstack=stack,
        # dataframes measurements
    xaxis_labels=12,
        # show #x_labels or 0=show all
    is_i_sc_self_ref=False,
        # is i_sc self referenced?
    Is_v_oc_temp_module_corr=True
        # is v_oc temp corrected?
)
```

If IV curves are not available to find r_sc and r_oc (e.g. just indoor matrices IEC 61853) then similar stacked loss plots with just 4 parameters can be generated.

Figure 4(right) plots losses for LFM_4(i_sc, i_mp, v_mp, v_oc_t, temperature correction) for a cSi module IEC 61853 matrix measured by CFV.

LFM_6 – Outdoor c-Si Gantner Instruments



LFM_4 – IEC 61853 Matrix c-Si CFV

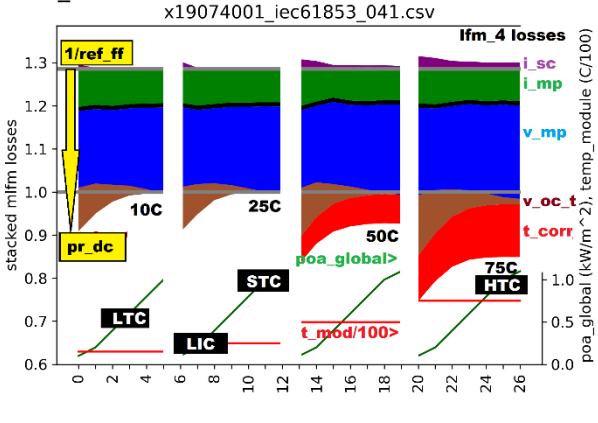


Figure 4: Stacked loss LFM_6 outdoor (left) and LFM_4 iec 61853 (right) plots for cSi modules

.

Explanations for figure 4 left:

- 1 clear day per month Jan to Dec Tempe, AZ
- Lowest pr_dc at middle of days due to highest temperature and roc losses (~r_series).
- Lowest pr_dc summer months due to temperature.
- This module has much higher r_oc (~r_series) loss than r_sc (~ r_shunt).
- Isc loss is non-zero, cause is soiling and spectral?

Explanations for figure 4 right:

- IEC 61853 matrix measurements indoor
- i_sc loss (purple) ~0 (measured at AM1.5, aoi=0 and soil = 0 so no corrections are needed).
- i_mp loss is almost constant, less than v_mp.
- v_mp increases with temp. and irradiance (→ ~r_series).
- v_oc loss worst at low light (↙left of each stack).
- temp_module losses worst at high temp. (↘right=75C).

2.8 **[H] 'Mechanistic fit to normalised losses'**

A mechanistic performance model MPM [ref] is used to fit any of the LFM parameter behaviours with irradiance and module temperature (3).

This algorithm minimizes rmse using python's optimize.curve_fit.

$$\boldsymbol{mpm} = c_1 + c_2 * (temp\_module - 25) + c_3 * log_{10}(poa\_global) + c_4 * poa\_global + c_5 * ws + c_6 / poa\_global \qquad (3)$$

```
# Python code [H]

# setup initial values and initial boundary
# conditions

# initial   c1    c2    c3    c4    c5   c6<0
p_0       = (1.0, 0.01, 0.01, 0.01, 0.01, -0.01)
# boundaries
```

```
bounds = ([ -2,   -2,   -2,   -2,   -2,    -2],
          [  2,    2,    2,    2,    2,     0])
```

The mpm gives meaningful, normalised and orthogonal coefficients unlike other models.

For indoor measurements c_5 (wind_speed) is 0, most modules can be fitted with a c_6 =0 but if not, it must be <0 as the boundary conditions constrict it.

Some typical example coefficients are given in table II

Table II : Typical MPM coefficients

Coeff	Value	Unit	Comment
C_1	+106.8%		Overall quality
C_2	-0.45%	/K	Temp. coeff. gamma
C_3	+0.48%		Low light Voc. Rsh
C_4	-7.03%	/(kW/m^2)	High light Rs
C_5	-0.063%	/(ms^{-1})	Windspeed
C_6	-1.54%	*(kW/m^2)	Extra Low light

2.9 **[I] 'Plot contour maps: pr_dc(colours) vs. temp_mod(↑) and poa_global(→)'**

Figure 5 left shows the measured (left) vs. mpm fitted (right) pr_dc per irradiance bin (100W/m^2→) and temp_module bin(5C↑) from a Gantner Instruments cSi module in Tempe, AZ.Figure 5 right gives the IEC 61853 Matrix by CFV.

Both show close MPM fits as there are close similarities against temp_module(C↑) and irradiance(W/m^2→).

Some of the test condition points are shown as the Matrix measurement does not have linear irradiance and temperature values.

```
# Python code [I]
# Measured vs. MPM Fitted vs. poa_global(→)
# and temp_module(↑)
contour_plot = plot_contourf(
    df=matr2,
    y_axis='temp_module',
    x_axis='poa_global',
    z_axis=mlfm_sel,
    title='matrix predicted ' + mlfm_meas_file,
    vmin=0.7,
    vmax=1.05,
    levels=9
```

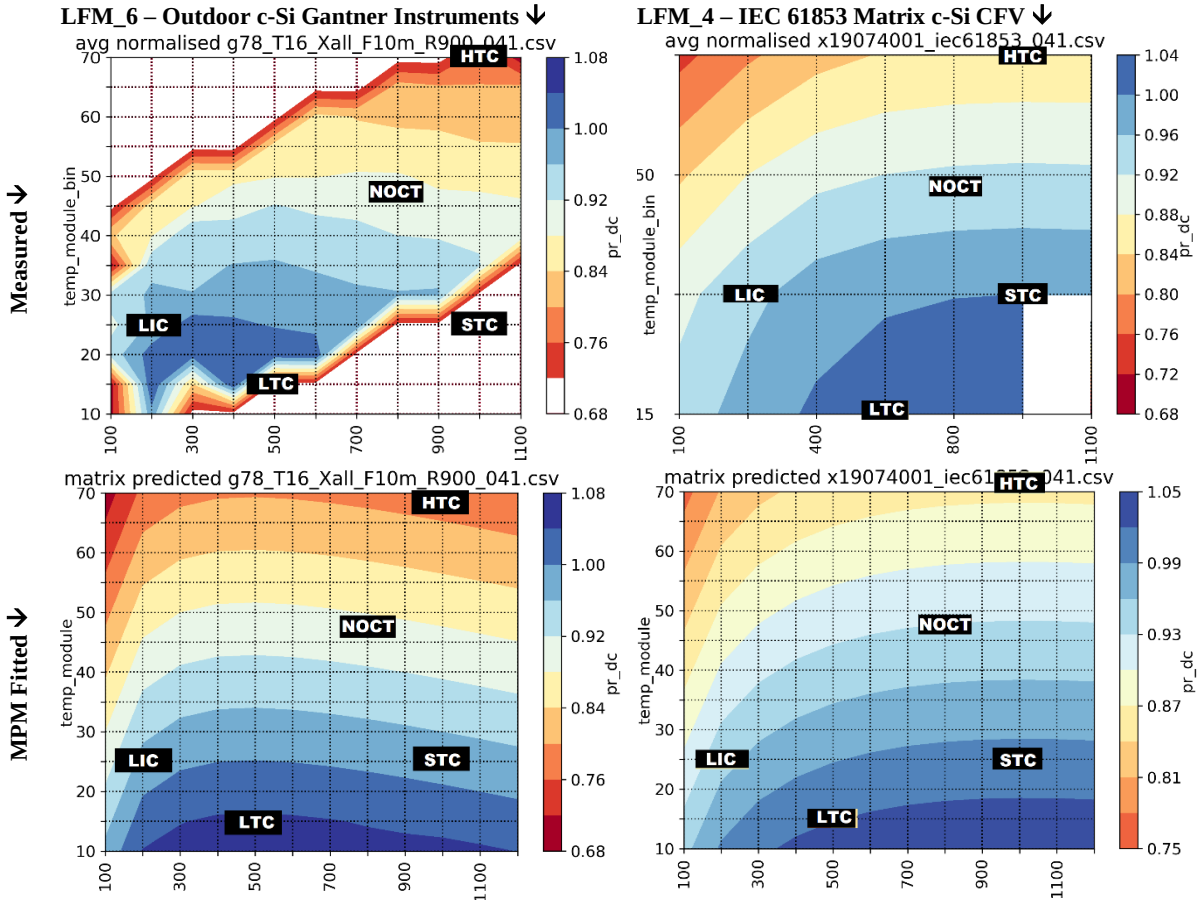


Figure 5: Contour plots of averaged pr_dc measured (top) vs. MPM fitted (bottom) for LFM6 (left) and LFM4 (right). Note: HTC should be at 75C (off the top of most of graphs)

2.10 **[J] 'Plot residual fit heatmap: pr_dc 'meas – fit' vs. temp_mod(↑), irradiance (→)**'

Figure 6 plots residuals '(fit – meas pr_dc)' against irradiance (W/m^2→) and module temperature (C↑) bins for LFM_6 (left) and LFM_4 (right).

Apart from some extreme outdoor weather conditions i.e. the top and bottom of the weather distribution most weather bins have the 'meas – fit' within ±-1% (light cyan to light orange).

Note the fit to the indoor measurements is even better at mostly ± 0.4%. The 50C lines is all overestimated a little, it's possible that this chart has just identified a slight measurement inaccuracy at 50C as they are all ~+0.3% rather than -0.1%. This will be investigated further.

```
# Python code [J]

# Residual MLFM fit heatmap(colours)
# vs. poa_global(x), temp_module(y)
heatmap_plot = plot_heatmap(
    dnorm=norm,
    dmeas=meas,
    fit=mlfm_sel,
    y_axis='temp_module_bin',
    x_axis='poa_global_bin',
    z_axis='diff_' + mlfm_sel,
    title='residual ' + mlfm_meas_file
)
```

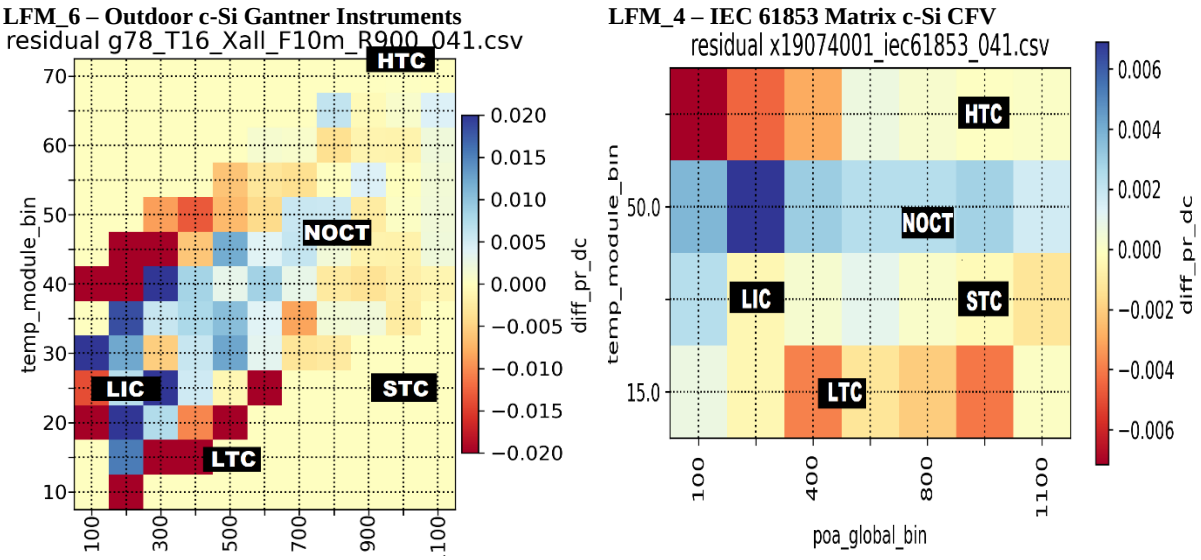


Figure 6: Residual error heatmaps pr_dc 'meas – fit' for LFM6 (left) and LFM4 (right)

2.11 **[K] 'Scatter plot: norm pmp fit(↑) vs. meas(→)'**

Figure 7 gives the mpm fitted vs. measured normalised p_mp (= meas_p_mp / ref_p_mp) with a 1:1 line) indicating a very good fit for both LFM_6 and LFM_4.

```
# Python code [K]

# plot fit vs. measured,
# include a 1:1 line
fit_plot = plot_fit(
    dmeas=meas,
    dnorm=norm,
    fit=mlfm_sel,
    title='fit ' + mlfm_meas_file
)
```

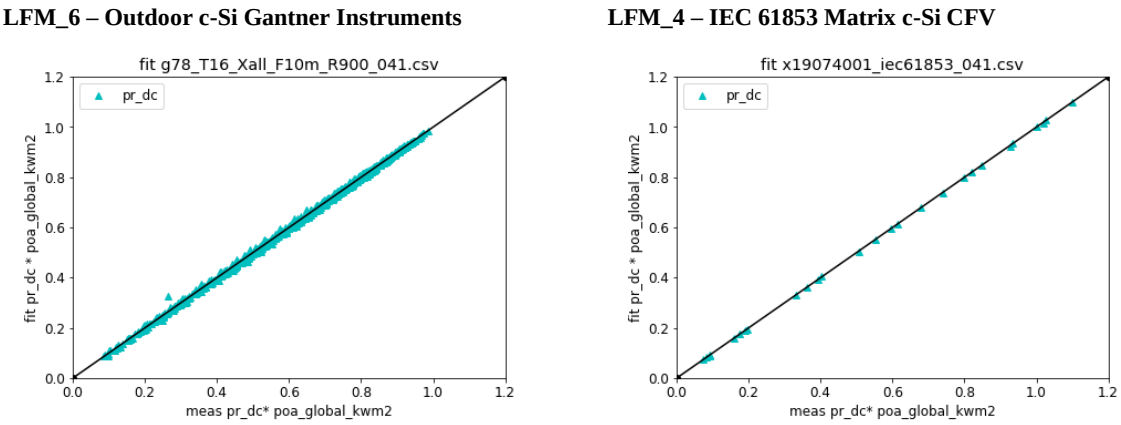


Figure 7: Scatter plot showing normalised p_mp fitted (↑) vs. measured (→)

It is hoped that the mlfm can be put into pvlib soon. Please see the pull [4] request and help if you can.

Table III summarises links and information on helping adding code to PVLIB.

4. CONTRIBUTING

Table III: How to contribute to PVLIB

INFORMATION :

• Stack Overflow :	http://stackoverflow.com/questions/tagged/pvlib
• Google groups :	https://groups.google.com/forum/#!forum/pvlib-python
• GitHub issues :	https://github.com/pvlib/pvlib-python/issues

• Pull requests : https://github.com/pvlib/pvlib-python/pulls
• Jupyter Notebook tutorials : https://github.com/pvlib/pvlib-python/tree/master/docs/tutorials
• Follow PEP8 : https://www.python.org/dev/peps/pep-0008/. (Max line length 79 chars)
• Add your project : https://github.com/pvlib/pvlib-python/wiki/Projects-and-publications-that-use-pvlib-python.
• Variables and Symbols : https://pvlib-python.readthedocs.io/en/stable/user_guide/variables_style_rules.html#variables-style-rules
• Documentation style : https://pvlib-python.readthedocs.io/en/stable/contributing.html#documentation

Coding :
• PVSystem and Location classes provide convenience wrappers around the core PVLIB functions.
• Remove logging calls and print statements.
• Include documentation and Comprehensive unit tests
• Functions must return the desired output for all inputs see https://github.com/pvlib/pvlib-python/issues/394

PVPMC : https://pvpmc.sandia.gov/
Document Library : https://pvpmc.sandia.gov/resources-and-events/documents/
Code of conduct : https://github.com/pvlib/pvlib-python/blob/master/CODE_OF_CONDUCT.md
Cite this paper: https://doi.org/10.21105/joss.00884

5. Acknowledgements

https://doi.org/10.21105/joss.00884

Gantner Instruments https://www.gantner-instruments.com/ and NREL for outdoor data,

CFV for indoor data
https://pvpmc.sandia.gov/download/7701/

6. CONCLUSIONS

The MLFM is being added to PVLIB python

The robust modelling of the LFM and MPM have been described

These are proving useful in industrial projects from single module OTFs to large power plants

Information has been given on how other coders can contribute their projects

7. REFERENCES

[1] "Modelling of PV modules and systems" VIRTUAL PVCOST Training School 2021 Romania http://www.steveransome.com/pubs/2021_07_PVCOST_Romania_Ransome_210706t11tobepresented.pdf
[2] “Benchmarking PV performance models with high quality IEC 61853 Matrix measurements (Bilinear interpolation, SAPM, PVGIS, MLFM and 1-diode)” PVSC 49 Jun 2022 Philadelphia, USA http://www.steveransome.com/pubs/2206_PVSC49_philadelphia_4_presented.pdf
[3] “Improving IEC 61853 Energy Yield Modelling with LFM and MPM models” 2022 PVPMC Salt Lake City USA https://pvpmc.sandia.gov/download/8494/
[4] MLFM pull request https://github.com/pvlib/pvlib-python/pull/1354
[5] "Quantifying and analysing the variability of PV module resistances Rsc and Roc to understand and optimise kWh/kWp modelling" Asian PVSEC-27 2017 Shiga Japan. http://www.steveransome.com/PUBS/1711_7TuPo.225_27PVEC_Ransome_3.pdf
[6] William F. Holmgren, Clifford W. Hansen, and Mark A. Mikofski. “pvlib python: a python package for modeling solar energy systems.” Journal of Open Source Software, 3(29), 884, (2018).