



PyMASK

G. Iadarola, G. Sterbini

Many thanks to:

R. De Maria, S. Fartoukh, F. Van Der Veken, S. Kostoglou



- **Introduction and motivation**
 - The cpymad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - Structure of the code
 - Configuration of the anticlockwise beam (b4)
 - Online resources (repository, documentation, examples)
 - First experience



Traditionally, a **“mask file”** is a **MAD-X script used to build a model of the LHC ring and then generate the corresponding input for tracking simulations** (sixtrack)

As described at the [Beam-beam and Luminosity Studies Meeting on 28 Feb 2020](#), to improve its usability and maintainability, the code has been recently **split in separate MAD-X scripts for the different functionalities** (modules)

- Tracked in a git repository (<https://github.com/lhcopt/lhcmask/>)
- Documented in a dedicated website (<http://cern.ch/lhcmask>)

Here we present a further evolution, which goes beyond the pure MAD-X environment, to handle more advanced simulation scenarios.



Recent study cases highlighted **limitations of existing mask files** that cannot be easily overcome using the the **conventional MAD-X scripting**:

- **Difficulty in handling sequences with opposite orientations** (b1 and b4) within the same MAD-X environment
- Difficulty in storing and handling **strings** (using knob names as parameters, manipulate file paths, implementing logics on sequence or element names)
- Impossibility to **allocate and manipulate arrays** (e.g. bunch-by-bunch intensities, locations of beam-beam encounters), etc.
- Difficulty to **profit from interactive development, debuggers, general purpose libraries** (FFT, advanced plotting, optimization, special functions etc.)

These issues were addressed by **integrating MAD-X with a more powerful programming language**

→ The natural choice is **python**, which has become a **standard for numerical computing and data analysis** at CERN and [in the world](#)

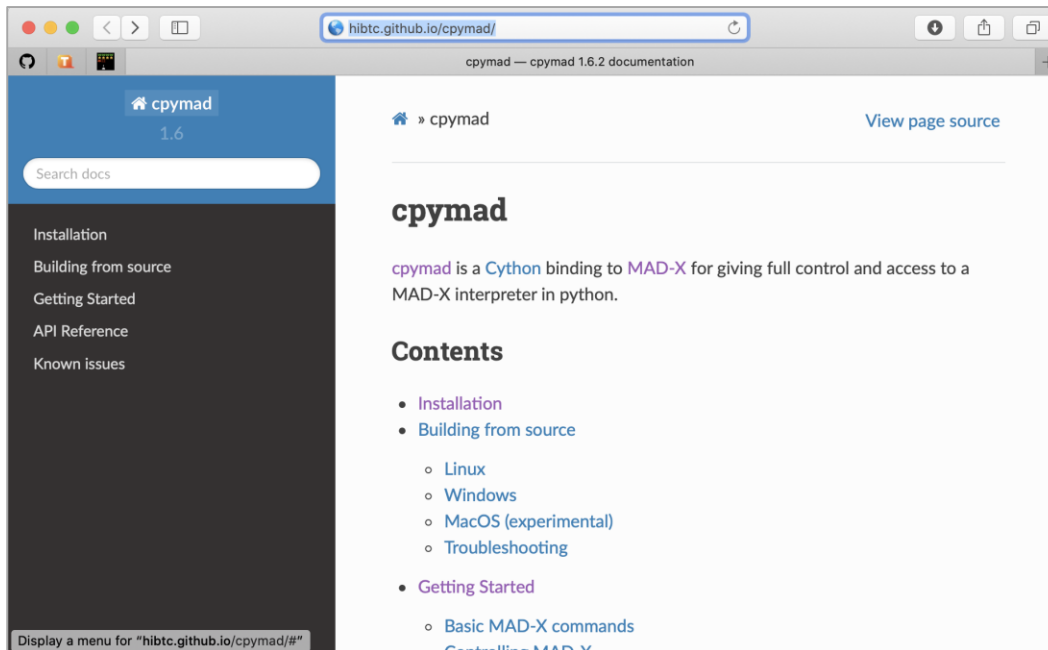


Interfacing MAD-X and python with cpymad

Efforts to **integrate MAD-X with python** date back to **2011** and led to the development of the **cpymad library** which provides **python bindings to MAD-X**.

By now the library is quite **mature**:

- Rather **robust** (python is never crashed by MAD-X)
- Reasonably **efficient** for the cases of interest
- Complete and **intuitive interface**
- Well **documented and supported** - presently developed at HIT (mainly) and at CERN



Some references:

[1] K. Fuchsberger, Y. Levinsen, “PyMad – Integration of MadX in python”, [WEPC119, IPAC11](#)

[2] R. Cee, T. Gläßle, et al. A MAD-X Model of the HIT Accelerator, [MOPME010, IPAC2014](#)

[3] <http://hibtc.github.io/cpymad/>



An **instance of MAD-X** is a **python object**:

```
from cpymad.madx import Madx
mad = Madx()
```

The usual MAD-X syntax can be used:

```
mad.input('a = 10;') # Variable assignment
mad.input('b := 2*a;') # Def. expression

# Call a mad-x script
mad.input('call, file="optics.madx;')

# Inject python variables into MAD-X command
qx = 62.31; qy = 60.32
mad.input(f'''
    match;
    global, q1={qx}, q2={qy};
    vary, name=kqtf, step=1.0E-7 ;
    vary, name=kqtd, step=1.0E-7 ;
    lmdif, calls=500, tolerance=1.0E-21;
    endmatch;
''')
```

MAD-X **variables, expressions, sequences and elements** are **accessible (read/write) in a pythonic way**:

```
mad.globals.on_sep8 = 1
mad.elements['mqml.6l11.b2'].kn1 = [0, 2e-3]
```

Pythonic access to **MAD-X tables** (no need to parse text files!)

```
mad.twiss()
tw = mad.table.twiss
plot(tw.s, tw.betx)
```

Multiple MAD-X instances can be handled **within the same python environment**:

```
# Create two instances
mad1 = Madx()
mad2 = Madx()
[...]
# They can interact through python
mad2.globals.a = 2 * mad1.globals.b
```

These simple features allow **accessing all functionalities of MAD-X in python** and to **combine them with other python packages** to obtain very powerful solutions (more in Guido's slides).



- **Introduction and motivation**
 - The cypmad interface to MAD-X
- **pymask**
 - General approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - Structure of the code
 - Configuration of the anticlockwise beam (b4)
 - Online resources (repository, documentation, examples)
 - First experience



The **main file is a python program** which **controls one or more instances of MAD-X** via cpymad:

- **Some parts of the old mask**, for which we are satisfied with the possibilities offered by the legacy MAD-X code, are **left untouched** and the MAD-X code is simply called
- **Other parts** for which we were limited by the shortcomings of MAD-X scripting, are **restructured and handled directly in python**

→ All capabilities of the legacy mask file are kept

→ More features are added

The **simulation** can be **defined by the following three files**:

- **pymask.py**: is the main executable. For standard simulation scenarios, it can be left untouched
- **config.py**: defines the simulation settings and parameters.
- **optics_specific_tools.py**: containing the small set of functions that need to be different when changing machine (LHC/HL-LHC) or optics



The configuration file (config .py) consists in a **python dictionary** with settings and configuration (structure is very flexible)

```
configuration = {  
[...]  
# Optics file  
'optics_file': ('/afs/cern.ch/eng/lhc/'  
+ 'optics/HLLHCv1.4/round'  
+ '/opt_round_150_1500_thin.madx'),  
  
# Enable checks  
'check_betas_at_ips' : True,  
'check_separations_at_ips' : True,  
'save_intermediate_twiss' : True,  
  
# Tolerance for check on flat machine  
'tol_co_flatness' : 1e-6,  
  
# Beam parameters  
'beam_norm_emit_x' : 2.5, # [um]  
'beam_norm_emit_y' : 2.5, # [um]  
'beam_sigt' : 0.076, # [m]  
'beam_sige' : 1.1e-4, # [-]  
'beam_npart' : 2.2e11, # [-]  
'beam_energy_tot' : 7000, # [GeV]  
  
[...]
```

pymask includes **checks on optics and knob self consistency** (not present in MAD-X mask)

```
# Tunes and chromaticities  
'qx0' : 62.31,  
'qy0' : 60.32,  
'chromaticity_x' : 5., # [-]  
'chromaticity_y' : 5., # [-]  
  
'oct_current' : -235, # [A]  
  
# Luminosity parameters  
'enable_lumi_control' : True,  
'sep_plane_ip2' : 'x',  
'sep_plane_ip8' : 'y',  
'lumi_ip8' : 2e33, # [Hz/cm2]  
'fullsep_in_sigmas_ip2' : 5,  
  
knob_naming = {...}  
  
knob_settings = {...}  
  
[...]  
}
```



- **Introduction and motivation**
 - The cypmad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - Structure of the code
 - Configuration of the anticlockwise beam (b4)
 - Online resources (repository, documentation, examples)
 - First experience



Handling different knob naming convention

The **proliferation of versions and “quasi-copies” of the old mask** (a nightmare to propagate bug fixes) was **largely caused by the need of adapting to different knob naming** conventions for different optics

Pymask file is instead **built to be independent on the specific knob naming** (exact same code is used for LHC Run 3 and HL-LHC optics):

- Closed **orbit** is configured by a **user-provided list of knobs** (the mask is agnostic w.r.t. naming)
- **Macros** (*crossing_save*, *crossing_disable*, *crossing_restore*) are **used to switch to flat orbit** and back, as already done in the most recent versions of the MAD-X mask

For LHC Run 3 optics

```
'knob_settings': {  
'on_x1'      : 150,          # [urad]  
'on_sep1'    : 0,           # [mm]  
'on_x2h'     : 0,           # [urad]  
'on_x2v'     : 200,         # [urad]  
'on_sep2h'   : 1.0,         # [mm]  
'on_sep2v'   : 0,           # [mm]  
'on_x5'      : 150,         # [urad]  
'on_sep5'    : 0,           # [mm]  
'on_x8h'     : -250,        # [urad]  
'on_x8v'     : 0,           # [urad]  
'on_sep8h'   : 0.,         # [mm]  
'on_sep8v'   : -1.0,        # [mm]  
[...]  
},
```

For HL-LHC

```
'knob_settings': {  
'on_x1'      : 250,          # [urad]  
'on_sep1'    : 0,           # [mm]  
'on_x2'      : -170,        # [urad]  
'on_sep2'    : 0.138,       # [mm]  
'on_x5'      : 250,         # [urad]  
'on_sep5'    : 0,           # [mm]  
'on_x8'      : -250,        # [urad]  
'on_sep8'    : -0.043,     # [mm]  
[...]  
},
```



Handling different knob naming convention

The **proliferation of versions and “quasi-copies” of the old mask** (a nightmare to propagate bug fixes) was **largely caused by the need of adapting to different knob naming** conventions for different optics

Pymask file is instead **built to be independent on the specific knob naming** (exact same code is used for LHC Run 3 and HL-LHC optics):

- Closed **orbit** is configured by a **user-provided list of knobs** (the mask is agnostic w.r.t. naming)
- **Macros** (*crossing_save*, *crossing_disable*, *crossing_restore*) are **used to switch to flat orbit** and back, as already done in the most recent versions of the MAD-X mask
- For knobs that need to be directly set by the mask, **naming convention is provided in input**

For LHC Run 3 optics

```
'knob_names' : {
'qknob_1': {'lhcb1': 'dQx.b1_sq',
            'lhcb2': 'dQx.b2_sq'},
'qknob_2': {'lhcb1': 'dQy.b1_sq',
            'lhcb2': 'dQy.b2_sq'},
'chromknob_1': {'lhcb1': 'dQpx.b1_sq',
                'lhcb2': 'dQpx.b2_sq'},
'chromknob_2': {'lhcb1': 'dQpy.b1_sq',
                'lhcb2': 'dQpy.b2_sq'},
'cmrknob': {'lhcb1': 'CMRS.b1_sq', 'lhcb2': 'CMRS.b2_sq'},
'cmiknob': {'lhcb1': 'CMIS.b1_sq', 'lhcb2': 'CMIS.b2_sq'},
},
```

For HL-LHC

```
'knob_names' : {
'qknob_1': {'lhcb1': 'kqtf.b1',
            'lhcb2': 'kqtf.b2'},
'qknob_2': {'lhcb1': 'kqtd.b1',
            'lhcb2': 'kqtd.b2'},
'chromknob_1': {'lhcb1': 'ksf.b1',
                'lhcb2': 'ksf.b2'},
'chromknob_2': {'lhcb1': 'ksd.b1',
                'lhcb2': 'ksd.b2'},
'cmrknob': {'lhcb1': 'cmrskew', 'lhcb2': 'cmrskew'},
'cmiknob': {'lhcb1': 'cmiskew', 'lhcb2': 'cmiskew'},
},
```



- **Introduction and motivation**
 - The cpyrad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - **Configuration of the beam beam interactions**
 - Structure of the code
 - Configuration of the anticlockwise beam (b4)
 - Online resources (repository, documentation, examples)
 - First experience



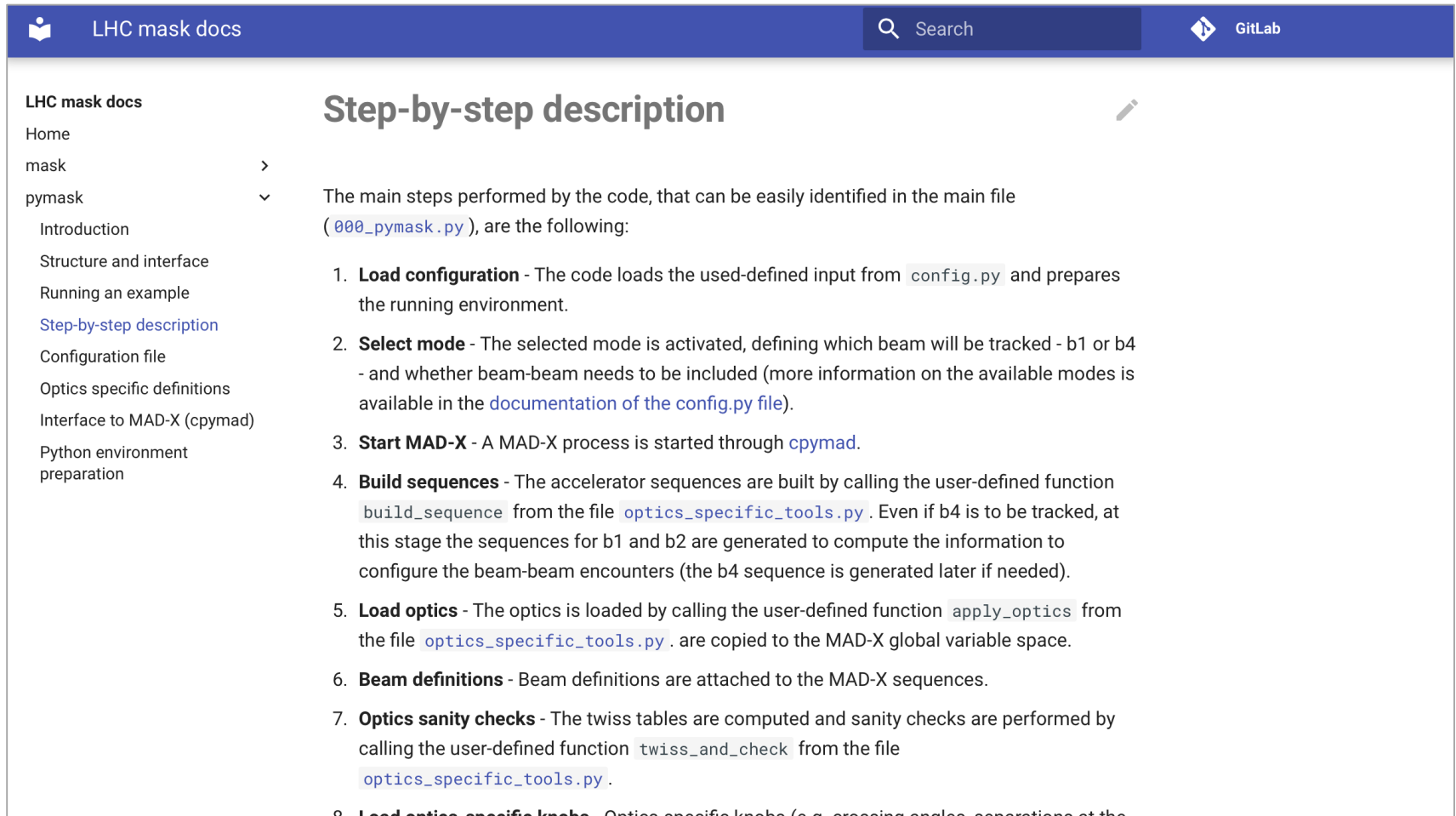
The MAD-X macros to configure the **beam-beam interactions in the legacy mask** have the **following limitations**:

- They are **not usable for the anticlockwise beam** (b4)
 - They are difficult to extend to a **general filling scheme**
 - They are difficult to extend to handle **bunch-by-bunch differences** in intensity or emittances
- The management of the **beam-beam interactions in pymask** is fully **restructured** and **handled directly in python** in order to allow these features
- The **procedure** to configure the beam-beam elements from the MAD-X model is **documented in [this note](#)** (containing all assumptions and sign conventions)



- **Introduction and motivation**
 - The cypmad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - **Structure of the code**
 - Configuration of the anticlockwise beam (b4)
 - Online resources (repository, documentation, examples)
 - First experience

A detailed **step-by-step description** of the code is available in the [documentation pages](#)
→ a symplified schematic is presented in the following



The screenshot shows a web page titled "LHC mask docs" with a search bar and a GitLab logo. The left sidebar contains a navigation menu with the following items: Home, mask, pymask, Introduction, Structure and interface, Running an example, Step-by-step description (highlighted), Configuration file, Optics specific definitions, Interface to MAD-X (cpymad), Python environment preparation. The main content area is titled "Step-by-step description" and contains the following text:

The main steps performed by the code, that can be easily identified in the main file (`000_pymask.py`), are the following:

1. **Load configuration** - The code loads the used-defined input from `config.py` and prepares the running environment.
2. **Select mode** - The selected mode is activated, defining which beam will be tracked - b1 or b4 - and whether beam-beam needs to be included (more information on the available modes is available in the [documentation of the config.py file](#)).
3. **Start MAD-X** - A MAD-X process is started through `cpymad`.
4. **Build sequences** - The accelerator sequences are built by calling the user-defined function `build_sequence` from the file `optics_specific_tools.py`. Even if b4 is to be tracked, at this stage the sequences for b1 and b2 are generated to compute the information to configure the beam-beam encounters (the b4 sequence is generated later if needed).
5. **Load optics** - The optics is loaded by calling the user-defined function `apply_optics` from the file `optics_specific_tools.py`. are copied to the MAD-X global variable space.
6. **Beam definitions** - Beam definitions are attached to the MAD-X sequences.
7. **Optics sanity checks** - The twiss tables are computed and sanity checks are performed by calling the user-defined function `twiss_and_check` from the file `optics_specific_tools.py`.
8. **Load optics-specific knobs** - Optics-specific knobs (e.g. crossing angles, separations at the



How does it work (b1)

MAD instance 1

Python

As already mentioned, the **“outside shell”** is a **python program** in which `cpymad` is used to launch an **instance of MAD-X controlled via python**



MAD instance 1

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Python

As already mentioned, the **“outside shell”** is a **python program** in which `cpymad` is used to launch an **instance of MAD-X controlled via python**

- The **MAD-X model of the collider** is built including the two **clockwise sequences** (B1 and B2)



MAD instance 1

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Python

As already mentioned, the **“outside shell”** is a **python program** in which `cpymad` is used to launch an **instance of MAD-X controlled via python**

- The **MAD-X model of the collider** is built including the two **clockwise sequences** (B1 and B2)
- **Twiss and survey** are performed for the two sequences



How does it work (b1)

MAD instance 1

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Python

Compute beam-beam tables
(locations, separations, sigmas)

BB table B1

BB table B2

We build **tables** with the **info on the beam-beam interactions**

using twiss and survey results:

- This stage is performed **entirely in python**
- The tables are handled as **pandas dataframes** (standard approach for the manipulation data)
- The tables **can be edited** to **customize** the simulation (e.g. with bbb intensities and emittances)

	charge_ppb	separation_x	separation_y	dpx	dpy	alpha	phi	Sigma_11	Sigma_13	Sigma_33
bb_lr.l1b1_25	2.200e+11	6.313e-02	-6.560e-14	-2.910e-04	3.523e-16	-1.211e-12	-1.455e-04	4.870e-06	7.944e-21	2.216e-06
bb_lr.l1b2_25	2.200e+11	-6.313e-02	6.560e-14	2.910e-04	-3.523e-16	-1.211e-12	1.455e-04	2.216e-06	-8.921e-20	4.870e-06
bb_lr.l1b2_01	2.200e+11	-1.870e-03	6.162e-15	-5.000e-04	1.769e-15	-3.538e-12	-2.500e-04	3.130e-08	-9.557e-22	3.130e-08
bb_ho.l1b2_05	2.000e+10	-1.356e-05	-3.030e-07	-5.000e-04	1.769e-15	-3.538e-12	-2.500e-04	6.071e-11	-8.259e-25	6.071e-11
bb_ho.c1b2_00	2.000e+10	0.000e+00	-4.536e-16	-5.000e-04	1.769e-15	-3.538e-12	-2.500e-04	5.026e-11	1.371e-24	5.026e-11
bb_ho.l1b2_05	2.000e+10	-1.356e-05	-3.030e-07	-5.000e-04	1.769e-15	-3.538e-12	-2.500e-04	6.071e-11	-8.259e-25	6.071e-11
bb_lr.r1b2_01	2.200e+11	1.870e-03	-7.069e-15	-5.000e-04	1.769e-15	-3.538e-12	-2.500e-04	3.130e-08	-7.466e-22	3.130e-08
bb_lr.r1b2_25	2.200e+11	6.313e-02	-7.271e-14	2.910e-04	-4.847e-16	-1.666e-12	1.455e-04	4.870e-06	-9.003e-20	2.216e-06



How does it work (b1)

Sequences: B1 and B2

MAD instance 1

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Python

Compute beam-beam tables
(locations, separations, sigmas)

BB table B1

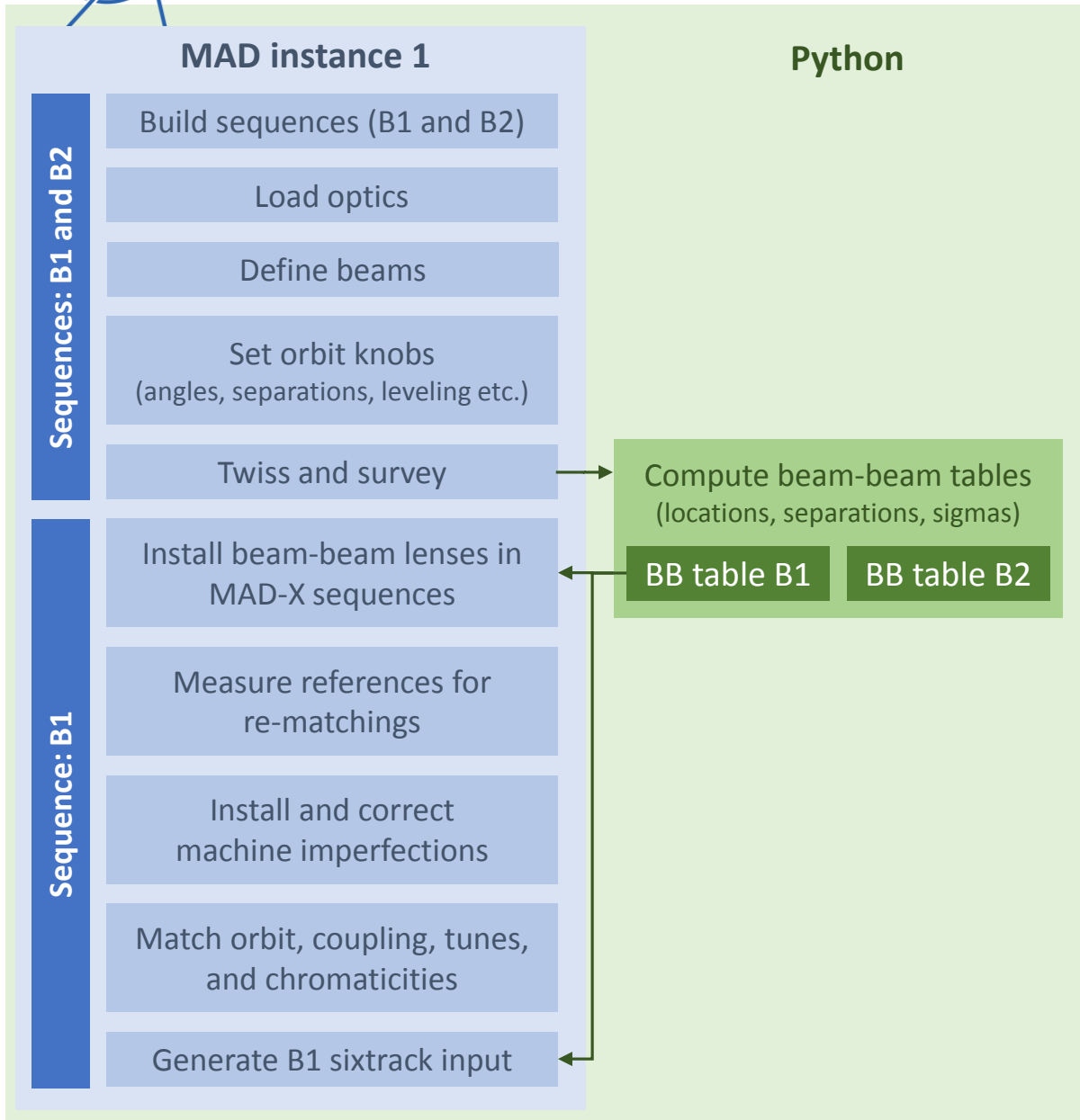
BB table B2

- **Definition and installation commands** of MAD-X beam-beam lenses are easily generated in python (using pandas)

elementName	elementDefinition	elementInstallation
bb_	elementName	elementDefinition
bb_	bb_lr.l1b2_25	install, element=bb_lr.l1b2_25, class=beambeam, at=93.5005723905724, from=ip1;
bb_	bb_lr.l1b2_01	install, element=bb_lr.l1b2_01, class=beambeam, at=-3.740022895622896, from=ip1;
bb_	bb_ho.l1b2_05	install, element=bb_ho.l1b2_05, class=beambeam, at=-0.06838770833711175, from=ip1;
bb_	bb_ho.c1b2_00	install, element=bb_ho.c1b2_00, class=beambeam, at=-0.0, from=ip1;
bb_	bb_ho.l1b2_05	install, element=bb_ho.l1b2_05, class=beambeam, at=-0.06838770833711175, from=ip1;
bb_	bb_lr.r1b2_01	install, element=bb_lr.r1b2_01, class=beambeam, at=-3.740022895622896, from=ip1;
bb_	bb_lr.r1b2_25	install, element=bb_lr.r1b2_25, class=beambeam, at=93.5005723905724, from=ip1;



How does it work (b1)



From this point we drop the B2 sequence and **work exclusively on b1** (beam to be tracked):

- Beam-beam **lenses are installed in the sequence** using commands in the tables
- **Reference** quantities like betas and orbit at the IPs are **saved** (with bb off)
- **Machine imperfections** are installed, and corrections are introduced
- **Orbit, coupling, tunes, chromaticities** are corrected
- The **Sixtrack input is generated** by MAD-X and the **beam-beam part in fort.3 generated in python** from the bb tables



- **Introduction and motivation**
 - The cypmad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - Structure of the code
 - **Configuration of the anticlockwise beam (b4)**
 - Online resources (repository, documentation, examples)
 - First experience



MAD instance 1

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Sequence: B1

Install beam-beam lenses in
MAD-X sequences

Measure references for
re-matchings

Install and correct
machine imperfections

Match orbit, coupling, tunes,
and chromaticities

Generate B1 sixtrack input

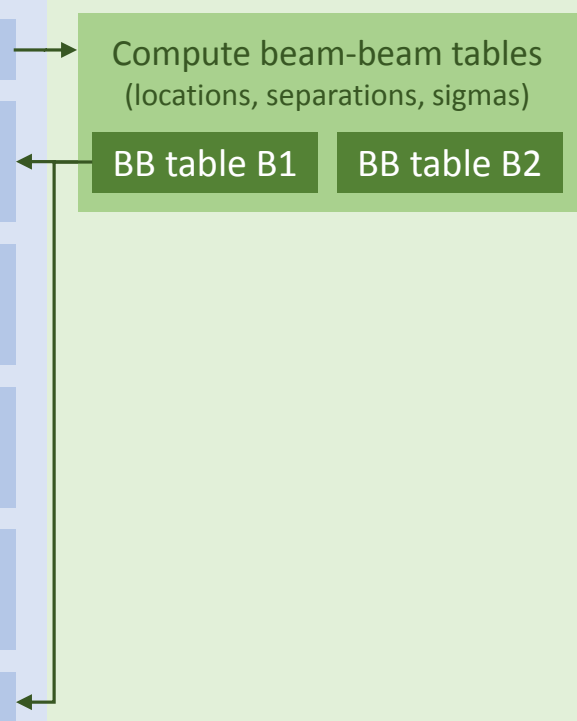
Python

The anticlockwise beam (B4) is handled by opening a **second instance of MAD-X within the same python environment.**

Compute beam-beam tables
(locations, separations, sigmas)

BB table B1

BB table B2





How does it work (b4)

MAD instance 1

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Sequence: B1

Install beam-beam lenses in
MAD-X sequences

Measure references for
re-matchings

Install and correct
machine imperfections

Match orbit, coupling, tunes,
and chromaticities

Generate B1 sixtrack input

Python

The anticlockwise beam (B4) is handled by opening a **second instance of MAD-X within the same python environment.**

Compute beam-beam tables
(locations, separations, sigmas)

BB table B1

BB table B2

MAD instance 2

Build sequence (B4)

Sequence: B4



How does it work (b4)

MAD instance 1

Python

MAD instance 2

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Sequence: B1

Install beam-beam lenses in
MAD-X sequences

Measure references for
re-matchings

Install and correct
machine imperfections

Match orbit, coupling, tunes,
and chromaticities

Generate B1 sixtrack input

All **MAD-X the variables** are **transferred** from the first to the second instance via python

Transfer
beam, knobs, variables

→ this **automatically configures optics and orbit** of B4 based on the configuration of B2 (including angles, separations etc.)

Build sequence (B4)

Optics, beam and orbit knobs
from twin MAD-X instance

Sequence: B4



MAD instance 1

Sequences: B1 and B2

Build sequences (B1 and B2)

Load optics

Define beams

Set orbit knobs
(angles, separations, leveling etc.)

Twiss and survey

Sequence: B1

Install beam-beam lenses in
MAD-X sequences

Measure references for
re-matchings

Install and correct
machine imperfections

Match orbit, coupling, tunes,
and chromaticities

Generate B1 sixtrack input

Python

Transfer
beam, knobs, variables

Compute beam-beam tables
(locations, separations, sigmas)

BB table B1

BB table B2

MAD instance 2

Build sequence (B4)

Optics, beam and orbit knobs
from twin MAD-X instance

Sequence: B4

The **beam-beam tables** for the **anticlockwise beams (b3 b4)** can be **generated in python** from those of the clockwise beams
→ it's a simple **reference frame transformation**



Coordinate transformations

Positions	Momenta
$x^{ACW} = -x^{CW}$	$p_x^{ACW} = +p_x^{CW}$
$y^{ACW} = +y^{CW}$	$p_y^{ACW} = -p_y^{CW}$
$s^{ACW} = -s^{CW}$	

Σ -matrix

$\Sigma_{11}^{ACW} = +\Sigma_{11}^{CW}$	$\Sigma_{23}^{ACW} = +\Sigma_{23}^{CW}$
$\Sigma_{12}^{ACW} = -\Sigma_{12}^{CW}$	$\Sigma_{24}^{ACW} = -\Sigma_{24}^{CW}$
$\Sigma_{13}^{ACW} = -\Sigma_{13}^{CW}$	$\Sigma_{33}^{ACW} = +\Sigma_{33}^{CW}$
$\Sigma_{14}^{ACW} = +\Sigma_{14}^{CW}$	$\Sigma_{34}^{ACW} = -\Sigma_{34}^{CW}$
$\Sigma_{22}^{ACW} = +\Sigma_{22}^{CW}$	$\Sigma_{44}^{ACW} = +\Sigma_{44}^{CW}$

Sequences: B1 and B2

Sequence: B1

Sequence: B4

Twiss and survey

Compute beam-beam tables
(locations, separations, sigmas)

Install beam-beam lenses in
MAD-X sequences

BB table B1 BB table B2

Measure references for
re-matchings

Reference frame
transformations

Install and correct
machine imperfections

BB table B3 BB table B4

Match orbit, coupling, tunes,
and chromaticities

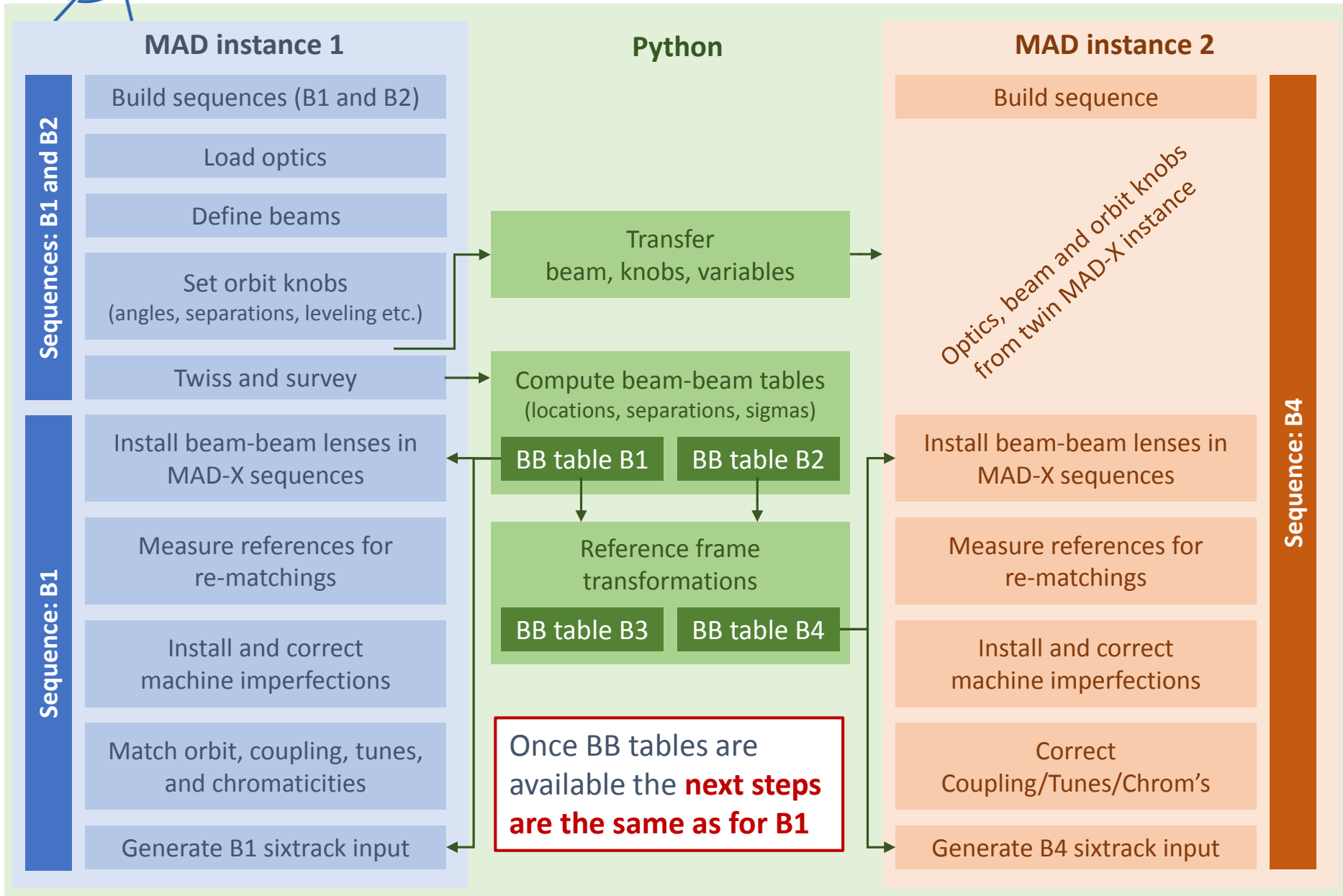
The **beam-beam tables** for the **anticlockwise beams (b3 and b4)** can be **generated in python** from those of the clockwise beams

Generate B1 sixtrack input

→ it's a simple **reference frame transformation**

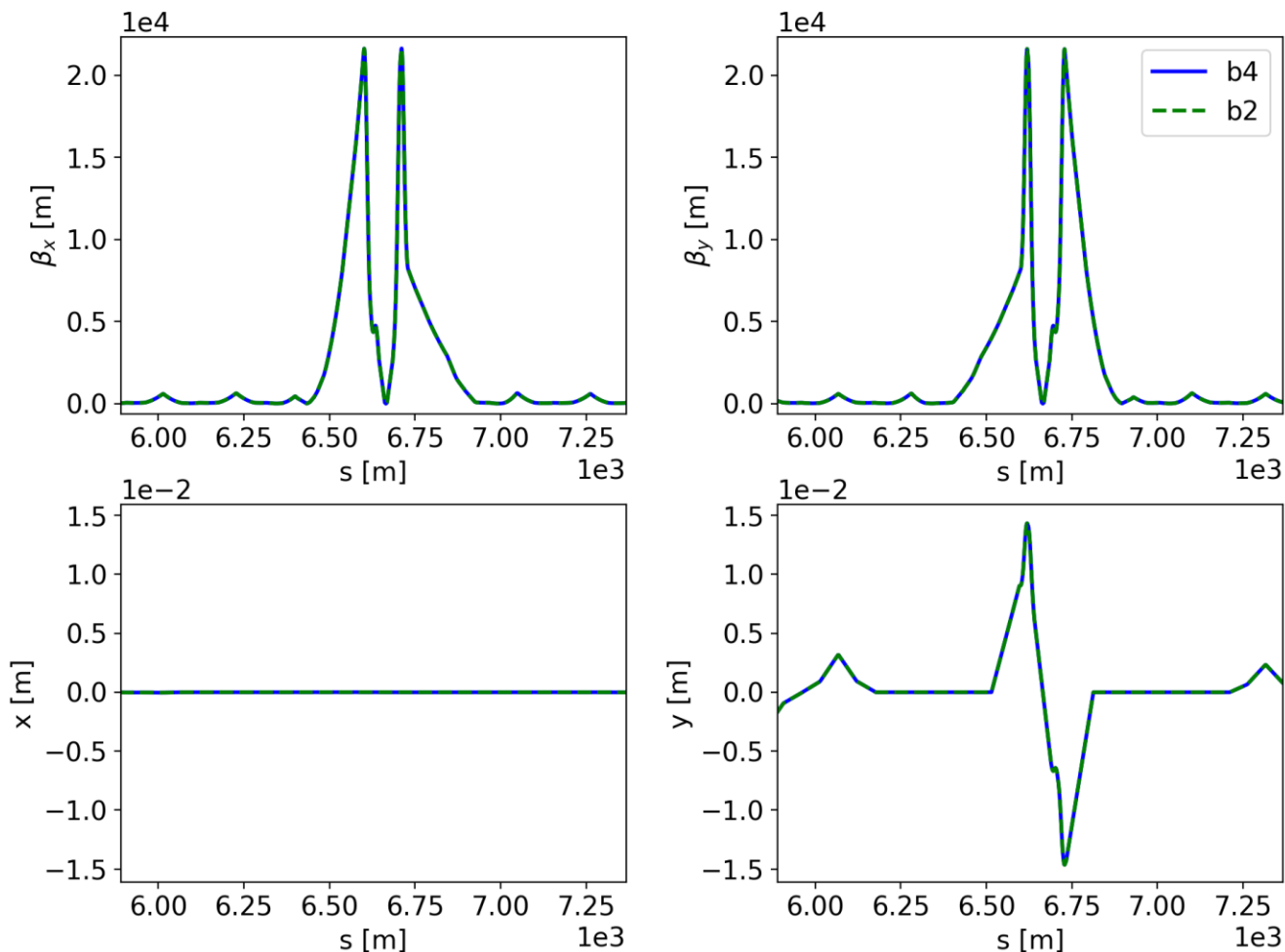


How does it work (b4)



The approach was **extensively tested**. In particular:

- **Linear optics is found to be identical** for b2 and b4 (once appropriate transformations are applied)
- In the absence of bb, the **sixtrack input** obtained with pymask is **identical (to the bit) to the one obtained by the legacy MAD-X mask on b4 alone**





- **Introduction and motivation**
 - The cypmad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - Structure of the code
 - Configuration of the anticlockwise beam (b4)
 - **Online resources (repository, documentation, examples)**
 - First experience



The **development** of pymask is **managed on github**:

- Code hosted at <https://github.com/lhcopt/lhcmask>
- Github **releases** used to deploy new features and fixes
 - New version number associated with each release
- **Github issues** used to **track bugs, enhancements, and future work to do**

The screenshot displays the GitHub interface for the repository `lhcmask`. On the left, the file explorer shows the repository structure, including folders like `examples`, `pymask`, and `python_examples`, and files like `.gitattributes`, `.gitignore`, `README.md`, and `module_01_orbit`. The main content area is split into two panels. The top panel shows the 'LHC Mask Version 0.3.1' release page, indicating it is the 'Latest release' and 'Verified'. The bottom panel shows the 'Issues' page, which lists 13 open issues and 6 closed issues. The issues are filtered by 'is:issue is:open'. The visible issues include:

- #34: Extracting the knobs of a sequence (opened 8 days ago by sterbini)
- #33: Coupling correction/measurement (enhancement) (opened 8 days ago by sterbini)
- #26: Possibility to select columns in twiss parquet (enhancement) (opened on 18 Sep by giadarol)
- #24: Generated pysixtrack line with errors is not correct (bug) (opened on 11 Sep by giadarol)
- #23: call files from within files (example Efcomp_MQXFbody.madx) (opened on 17 Jul by JoschD)



Documentation is available at <http://cern.ch/lhcmask/pymask>

It includes:

- **“Getting started” guide** (how to run at example)
- Description of the **interface**
- Step-by-step **description of the algorithm**
- A **guide to setup the python environment** (not required on lxplus and lxbatch)

Working exaples for **LHC Run 3** and **HL-LHC** are available in the repository ([here](#))

The screenshot shows the 'LHC mask docs' website. The header includes a search bar and a GitLab logo. The main content area is titled 'Documentation for pymask v0.3'. On the left, there is a navigation menu with the following items: Home, mask, pymask (expanded), Introduction, Structure and interface, Running an example, Step-by-step description, Configuration file, Optics specific definitions, Interface to MAD-X (cpymad), and Python environment preparation. The main text describes the development of pymask as an evolution of the traditional MAD-X approach, highlighting its flexibility and the fact that the code is hosted on GitHub at <https://github.com/lhcopt/lhcmask>. It also provides a complete example found [here](#). The text concludes by stating that pymask offers all the features of the legacy MAD-X mask file and provides the following capabilities that are not easily available with the pure MAD-X approach:

- Create code that is independent on knob naming conventions. This, for example, allows the usage of the same code for LHC and HL-LHC;
- Configuration of anticlockwise beam (b4) with beam-beam lenses by using multiple MAD-X instances within the same python environment;
- Manipulation of arrays and strings as well as interfacing to other python packages (general purpose and custom made). This allows, for example, importing measured bunch-by-bunch intensities, or realistic filling patterns.



- **Introduction and motivation**
 - The cypmad interface to MAD-X
- **pymask**
 - The approach
 - Handling different knob naming conventions
 - Configuration of the beam beam interactions
 - Structure of the code
 - Configuration of the anticlockwise beam (b4)
 - Online resources (repository, documentation, examples)
 - **First experience**



pymask has been (is being) used in **pilot applied studies**:

- [HL-LHC DA studies for B1 and B2](#) (Sofia)
 - pymask used within sixdesk
 - Several sanity checks passed
- [Effect of BB on beta-beating for LHC Run 3 scenarios](#) (Guido)
- [Quench heater failure scenarios simulations](#) (TE-MPE, Cedric, Oscar)
- [HL-LHC studies on crab-bump non-closure](#) (Sofia)

Overall **experience was rather smooth**:

- **Users could leverage their knowledge of python** to make advanced customization of their simulations
- The python interactive interpreter and debugger allow for **quick and powerful checks!**

Many thanks to the present users for the very useful feedback!

The tool is ready for more use cases, **so feel free to give it a try...**

(of course we are here to help)



Thanks for your attention!