pymad8 Documentation

Release 2.0.0

Royal Holloway

Mar 19, 2023

CONTENTS

pymad8 is a Python package to aid in the preparation, running and validation of BDSIM models.

CHAPTER

ONE

LICENCE & DISCLAIMER

This software is provided “AS IS” and any express or limit warranties, including, but not limited to, implied warranties of merchantability, of satisfactory quality, and fitness for a particular purpose or use are disclaimed. In no event shall Royal Holloway, University of London be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this software, even if advised of the possibility of such damage.

CHAPTER

TWO

AUTHORSHIP

The following people have contributed to pymad8:

- Stewart Boogert
- Andrey Abramov
- Laurie Nevay
- Will Parker
- William Shields
- Jochem Snuverink
- Stuart Walker
- Marin Deniaud

CHAPTER

THREE

INSTALLATION

3.1 Packages requirements

- numpy
- matplotlib
- pylab
- pandas
- fortranformat

pymad8 is developed for Python 3 but should be Python 2 compatible.

3.2 Installation

To install pymad8, simply run `make install` from the root pymad8 directory.:

```
cd /my/path/to/repositories/
git clone http://bitbucket.org/jairhul/pymad8
cd pymad8
make install
```

Alternatively, run `make develop` from the same directory to ensure that any local changes are picked up.

CHAPTER

FOUR

MAD8 FILE LOADING & MANIPULATION

MAD8 outputs Twiss information in their own file format. pymad8 includes a class called Output for the purpose of loading and manipulating this data.

The MAD8 format is described in the manual available from the mad8 website. The format is roughly described as a text file. The MAD8 files contain first a few lines that indicate when the file was generated and other general informations. After this, each segment of 5 lines typically represents the values of the lattice for a particular element with each new segment containing the values at a subsequent element in the lattice.

4.1 Output Class Features

- Loading different types of MAD8 files.
- Get a particular column.
- Get a particular row.
- Get elements of a particular type.
- Get a numerical index from the name of the element.
- Find the curvilinear S coordinate of an element by name.
- Find the name of the nearest element at a given S coordinate.
- Plot an optics diagram.
- Calculate a beam size given the Twiss parameters, dispersion and emittance.
- Make a slice of the initial lattice

4.2 Loading

MAD8 files can be of different types. Twiss files are the main ones but we can also load Rmat files, Chrom files, Envelope files or Survey files

A file may be loading by constructing an Output instance from a file name :

```
>>> import pymad8
>>> t = pymad8.Output("myTwissFile")
>>> r = pymad8.Output("myRmatFile", "rmat")
>>> c = pymad8.Output("myChromFile", "chrom")
>>> e = pymad8.Output("myEnvelopeFile", "envel")
>>> s = pymad8.Output("mySurveyFile", "survey")
```

Note: The import will be assumed from now on in examples.

4.3 Querying

The Output class can be used to query the data in various ways.

4.3.1 Basic Information

- All data is stored in the **data** object inside the class
- The number of elements is stored in **nrec**.
- The file name is stored in **filename**.
- The file type is stored in **filetype**.
- A dict of each element type and corresponding properties is stored in **keys_dict**
- The names of columns common to all file types is stored in **colnames_common**.

The **data** object is a pandas dataframe that can be displayed as follow :

```
>>> t = pymad8.Output("myTwissFile")
>>> t.data
```

4.3.2 Indexing and Slicing

The information stored in the dataframe is accessible using regular pandas syntax :

```
t.data.iloc[3]                      # 4th element in sequence (Pandas.Series returned)
t.data.iloc[3: 10]                  # 4th to 11th elements (Pandas.Dataframe returned)
t.data.iloc[[3, 5, 10]]             # 4th, 6th and 11th elements (Dataframe)
t.data['S']                         # column named exactly S (Series)
t.data[['S', 'L']]                  # columns named exactly S and L (Dataframe)
t.data['S'][3]                      # value of the 4th element in the column S
t.data[t.data['NAME'] == 'INITIAL'] # Row of the element with this exact name␣
↪(Dataframe)
```

But you can also find information about elements usind built-in functions.

To get index for example

```
t.getIndexByNames('INITIAL')              # can have a list of names as input
t.getIndexByTypes('QUAD')                 # can have a list of types as input
t.getIndexByValues(key='S', minValue=200) # indices of elements with S value above 200
t.getIndexByNearestS(150)                 # index of element with S value closest to␣
↪150
```

The results are returned in the form of one value or a list of values, depending on the input given. In the case of the getIndex function, we get either an integer or a list of integers

Note: Similar functions are avaliable to find names and types of lattice elements

4.3.3 Rows and Columns

A row of data is an entry for a particular element. The Output class is conceptually a list of elements. Each element is represented by a row in the pandas dataframe that has a key for each column. The list of acceptable keys (i.e. names of columns) can be found in the member named 'colums' :

```
t.data.columns #prints out list of column names
```

A specific row or set of rows can be accessed using similar functions as those previously shown :

```
t.getRowsByIndex(3)                       # can have a list of indices as input
t.getRowsByNames('INITIAL')               # can have a list of names as input
t.getRowsByTypes('QUAD')                  # can have a list of types as input
t.getRowsByValues(key='S', minValue=200) # rows of elements with S value above 200
t.getRowByNearestS(150)                   # row of element with S value closest to 150
```

The results are return either in the form of a dataframe or serie (which is equivalent to a dataframe with only one row), depending on the input given.

A specific column or set of columns can be accessed using its keys (i.e. its names) :

```
t.getColumnsByKeys(['S','L'])
```

4.4 Beam Sizes

For convenience the beam size is calculated from the Beta amplitude functions, the emittance, dispersion and enegy spread using *calcBeamSize()*. The emittance is defined by 'EmitX' and 'EmitY' and the energy spread by 'Esprd'. Those three parameters aren't provided by MAD8 and must be manualy given to the function :

```
EmitX = 3e-11
EmitY = 3e-11
Esprd = 1e-6
t.calcBeamSize(EmitX, EmitY, Esprd)
```

In this function, the beam sizes are calculated according to :

$$\sigma_x = \sqrt{\beta_x \epsilon_x + D(S)^2 \frac{\sigma_E^2}{E_0^2}}$$

$$\sigma_y = \sqrt{\beta_y \epsilon_y + D(S)^2 \frac{\sigma_E^2}{E_0^2}}$$

CHAPTER

FIVE

PLOTTING

The *pymad8.Plot* module provides various plotting utilities.

5.1 Plotting Features

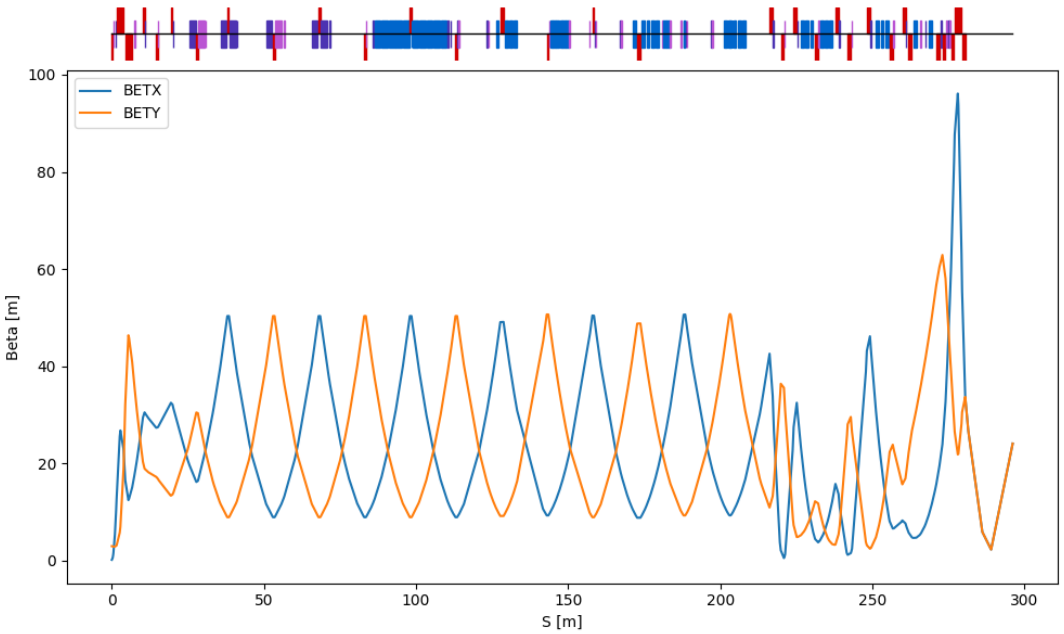
- Make default optics plots.
- Add a machine lattice to any pre-existing plot.

5.2 Optics Plots

A simple optics plot may be made with the following syntax :

```
p = pymad8.Plot.Optics("mytwissfile")
p.Beta()
```

This creates a plot of the Beta amplitude functions against curvilinear S position. A colour diagram representing the machine is also produced above the graph as shown below :



Other than beta, other optics plots can be made using *Alpha()*, *Mu()*, *Disp()* or *Sigma()*. These functions are provided as a quick utility and not the ultimate plotting script.

5.3 Machine lattice

The user can make their own plot and then append a machine diagram at the end if they wish :

```
f = matplotlib.pyplot.figure()
# user plotting commands here
pymad8.Plot.AddMachineLatticeToFigure(f, "mytwissfile")
```

gcf() is a matplotlib.pyplot function to get a reference to the current matplotlib figure and can be used as the first argument :

```
pymad8.Plot.AddMachineLatticeToFigure(gcf(), "mytwissfile")
```

Note: It becomes difficult to adjust the axes and layout of the graph after adding the machine description. It is therefore strongly recommended to do this last.

5.4 Colour Coding

Each magnet is colour coded an positioned depending on its type and strength.

Type	Shape	Colour	Vertical Position
drift	N/A	Not shown	N/A
sbend	Rectangle	Blue	Central always
rbend	Rectangle	Blue	Central always
hkicker	Rectangle	Purple	Central always
vkicker	Rectangle	Pink	Central always
quadrupole	Rectangle	Red	Top half for K1L > 0; Bottom half for K1L < 0
sextupole	Hexagon	Yellow	Central always
octupole	Hexagon	Green	Central always
multiple	Hexagon	Light grey	Central always
rcollimator	Rectangle	Black	Central always
ecollimator	Rectangle	Black	Central always
any other	Rectangle / Line	Light Grey	Central always

Note: In all cases if the element is a magnet and the appropriate strength is zero, it is shown as a grey line.

CHAPTER

SIX

CONVERTING MODELS

pymad8 provdies converters to allow BDSIM models to prepared from optical descriptions in MAD8.

For conversion of MAD8 to BDSIM GMAD format, please see the pybdsim documentation http://www.pp.rhul.ac.uk/bdsim/pybdsim/convert.html#mad8-twiss-2-gmad.

6.1 Mad8 output required

Listed here are the MAD8 lines required to generate various output files.

To make the Twiss and Rmat outputs :

```
use, LINE
twiss, beta0=LINE.B0, save, couple, tape=TWISS_LINE rtape=RMAT_LINE
```

To make the Chrom output :

```
use, LINE
twiss, beta0=LINE.B0, save, chrom, tape=CHROM_LINE
```

To make the Envelope output :

```
use, LINE
envel, sigma0=LINE.SIGMA0, save, tape=ENVEL_LINE
```

To make the Survey output :

```
use, LINE
survey, tape=SURVEY_LINE
```

CHAPTER

SEVEN

SIMULATIONS

The *pymad8.Sim* module provides various simulation utilities

7.1 Simulation features

- Particle tracking using Mad8 rmat files

7.2 Tracking

There is two tracking related classes in this module : *Track_Collection* and *Tracking*. These classes features are the following :

- Set the different particles we want to track
- Write the track collection as Mad8 or Bdsim inputs
- Set the different sampler at which we want to observe the particles
- Run the tracking using Rmatrices generated by Mad8
- Plot various results such as trajectory, phase space ...
- Load tracking data generated by BDSIM and compare with pymad8 trajectory

7.2.1 Building tracking classes

A collection of tracks can by setup by constructing a Track_Collection instance for a given reference energy in GeV :

```
track_collection = pymad8.Sim.Track_Collection(E_0)
```

At first there is no tracks in the collection. We can add each individual track by giving the starting position, angle, and energy difference with respect to the reference :

```
track_collection.AddTrack(x, xp, y, yp, z, DE)
```

The Tracking instance is generated from Mad8 twiss and rmat files :

```
tracking = pymad8.Sim.Tracking('twiss_tape', 'rmat_tape')
```

Then we can add the samplers at which we want to observe the particles. Multiples samplers can be added at once and we can either find them by index, name or type:

```
tracking.AddSampler(10)                        # Add element with index 10 as sampler
tracking.AddSampler('INITIAL', select='name') # Add element named 'INITIAL' as sampler
tracking.AddSampler('QUAD', select='type')    # Add all quadrupoles as samplers
```

7.2.2 Write track collection

In order to run other tracking codes with the same initial particles, we can write the track collection into text files that can be input for Mad8 or for Bdsim :

```
track_collection.WriteMad8Track(mad8outputfile)
track_collection.WriteBdsimTrack(bdsimoutputfile)
```

7.2.3 Run tracking

Now that we have both Track_Collection and Tracking instances we can run the pymad8 tracking :

```
tracking.RunPymad8Tracking(track_collection)
```

In case we have a circulare maching and we want the tracking data after a certain number of turn one can do :

```
tracking.RunPymad8Tracking(track_collection, turns=nb_turns)
```

The calculated data is then shaped as a pandas DataFrame and stored in **pymad8_df**

7.2.4 Plotting

After runing the tracking function. One can use the plotting function avaliable to show the trajectory, phase space, correlation and profile histograms :

```
tracking.PlotTrajectory(5, 'X')                    # Trajectory of the 5th particle␣
↪in X
tracking.PlotPhaseSpace(50, 'X')                   # Phase space in X at 50m
tracking.PlotCorrelation(50, 'X', ref_sampler='IP') # Correlation in X at 50m w.r.t␣
↪sampler named 'IP'
tracking.PlotHist(50, 'X')                         # Beam profile in X at 50m
```

7.2.5 Loading external tracking data

One can use the Tracking instance to load a BDSIM root file and store the tracking data in a similar way as **pymad8_df** :

```
tracking.LoadBdsimTrack(rootfile)
```

The data is then stored in **bdsim_df** and can then be use to plot trajectory comparison between pymad8 and BDSIM.

Note: Make sure to use the same initial particles using the *WriteBdsimTrack* function

CHAPTER

EIGHT

SUPPORT

All support issues can be submitted to our issue tracker

8.1 Feature Request

Feature requests or proposals can be submitted to the issue tracker - select the issue type as proposal or enhancement..

Please have a look at the existing list of proposals before submitting a new one.

CHAPTER

NINE

MODULE CONTENTS

This documentation is automatically generated by scanning all the source code. Parts may be incomplete.

pymad8 - python tools for working with MAD8 output and input.

Authors:

- Stewart Boogert
- Laurie Nevay
- Andrey Abramov
- William Shields
- Jochem Snuverink
- Stuart Walker
- Marin Deniaud

Dependencies: (*package - minimum version required*)

- numpy - 1.7.1
- matplotlib - 1.3.0
- pylab - 1.3.0 (dependancy of matplotlib)
- pandas - 1.4.3
- fortranformat - 1.2.0

Modules: (*script name - usage*)

- Input - Tidy Mad8 input
- Output - Load Mad8 files into dataframes
- Plot - Draw machine lattice and quick optics plots
- Sim - Perform simulations on a machine, like particle tracking
- Track - Old particle tracking code
- Visualisation - Old survey plotting code

9.1 pymad8.Input module

pymad8.Input.**decodeFileLine**(*input*)

Decode line for each element type
input : string of a mad8 line

pymad8.Input.**removeComments**(*input*)

Remove comment lines
input : list of file lines

pymad8.Input.**removeContinuationSymbols**(*input*)

Remove continuation symbols from input
input : list of file lines

pymad8.Input.**tidy**(*input*)

Tidy input, remove EOL, remove empty lines
input : list of file lines

9.2 pymad8.Output module

class pymad8.Output.**Output**(*filename*, *filetype='twiss'*)

Bases: object

Class to load different Mad8 output files in a Pandas DataFrame
>>> twiss = pymad8.Output('/twiss.tape')
>>> rmat = pymad8.Output('/rmat.tape','rmat')
>>> chrom = pymad8.Output('/chrom.tape','chrom')
>>> envel = pymad8.Output('/envel.tape','envel')
>>> survey = pymad8.Output('/survey.tape','survey')
>>> struct = pymad8.Output('/struct.tape','struct')
>>> track = pymad8.Output('/track.tape','track')
>>> optics = pymad8.Output('/optics.tape','optics')

By default the filetype is twiss

Clear()

Empties all data structures in this instance

calcBeamSize(*EmitX*, *EmitY*, *Esprd*, *BunchLen=0*)

Calculate the beam sizes and beam divergences in both planes for all elements Then the four columns are added at the end of the DataFrame Works only if a twiss file was loaded previously

getAperture(*index*, *defaultAperSize=0.1*)

Get aperture of an element using corresponding index

If none provided or is 0 : set to a default aperture of 0.1m

getColumnsByKeys(*keylist*)

Return Columns that ar in keylist

getElementByIndex(*indexlist*, *keylist*)

Return value of elements given their indices and for chosen columns

getElementByNames(*namelist*, *keylist*)

Return value of elements given their names and for chosen columns

getIndexByNames(*namelist*)

Return Index or Index list of elements that are in namelist

getIndexByNearestS(*s*)

Return Index of the closest element in the beamline

getIndexByTypes(*typelist*)

Return Index or Index list of elements that are in typelist

getIndexByValues(***args*)

Return Index or Index list of elements that have certain values for a chosen column
Same arguments as getRowsByValues

getNameByNearestS(*s*)

Return Name of the closest element in the beamline

getNamesByIndex(*indexlist*)

Return Name or Name list of elements that are in indexlist

getNamesByTypes(*typelist*)

Return Name or Name list of elements that are in typelist

getNamesByValues(***args*)

Return Name or Name list of elements that have certain values for a chosen column
Same arguments as getRowsByValues

getRowByNearestS(*s*)

Return Rows of the closest element in the beamline

getRowsByFunction(*f*)

Return a sub-datafarme using a boolean function

getRowsByIndex(*indexlist*)

Return Rows of elements that are in indexlist

getRowsByNames(*namelist*)

Return Rows of elements that are in namelist

getRowsByTypes(*typelist*)

Return Rows of elements that are in typelist

getRowsByValues(*key=None*, *minValue=-inf*, *maxValue=inf*, *equalValues=None*)

Return Rows of elements that have certain values for a chosen column

getTypeByNearestS(*s*)

Return Type of the closest element in the beamline

getTypesByIndex(*indexlist*)
Return Type or Type list of elements that are in indexlist

getTypesByNames(*namelist*)
Return Type or Type list of elements that are in namelist

getTypesByValues(***args*)

Return Type or Type list of elements that have certain values for a chosen column
Same arguments as getRowsByValues

plotXY(*Xkey*, *Ykey*, *color=None*, *label=None*)
Quick plot of one colums of our dataframe w.r.t. another

sMax()
Return maximal S value

sMin()
Return minimal S value

subline(*start*, *end*)
Select only a portion of the inital lattice

9.3 pymad8.Plot module

pymad8.Plot.**AddMachineLatticeToFigure**(*figure*, *mad8opt*, *tightLayout=True*)

Add a diagram above the current graph in the figure that represents the accelerator based on a mad8 twiss file.

Note you can use matplotlib's gcf() 'get current figure' as an argument.

>>> pymad8.Plot.AddMachineLatticeToFigure(gcf(), '/mad8_twiss_tape')

class pymad8.Plot.**Optics**(*mad8file*, *beamParams=None*)
Bases: object

Class to load pymad8 DataFrames and make optics plots
>>> plot_data = pymad8.Plot.Optics('/mad8_twiss_tape')
>>> plot_data.Betas()

Plot avaliable are plotBetas, plotAlphas, plotMus, plotDisp and plotSigmas

Alpha()
Plot the Alpha functions for both planes and both Mad

Beta()
Plot the Beta functions for both planes and both Mad

Disp()
Plot the Dispertion functions for both planes and both Mad

Mu()
Plot the Mu functions for both planes and both Mad

Sigma()

Plot the beam size and beam divergence functions for both planes and both Mad

9.4 pymad8.Sim module

pymad8.Sim.**CheckUnits**(*coord*)

class pymad8.Sim.**Track_Collection**(*E_0*)

Bases: object

Class to create and store different initial particles for future tracking
Requires only the reference enegy in GeV
>>> track_collection = pymad8.Sim.Track_Collection(14)

AddTrack(*x*, *xp*, *y*, *yp*, *z*, *DE*)

Add a new track to the collection with given position, angle and energy difference from the reference one

GenerateNtracks(*nb*, *paramdict*)

Add multiple track for which all parameters follow a gaussian disstribution

WriteBdsimTrack(*outputfile*)

Write the track collection in a file that can be read by BDSIM

WriteMad8Track(*outputfile*)

Write the track collection in a file that can be read by Mad8

class pymad8.Sim.**Tracking**(*twiss*, *rmat*)

Bases: object

Class to run tracking for a given track collection using Mad8 Rmat files
Requires the Mad8 twiss and rmat files
>>> tracking = pymad8.Sim.Tracking('twiss_tape', 'rmat_tape')

Include a set of plotting function for tracjectory, phase space, ...
One can also load tracking data generated by BDSIM in order to make comparison plots

AddAllElementsAsSamplers()

Register all lattice elements as samplers for the tracking

AddSamplers(*value*, *select='index'*)

Add one or multiple samplers which will be used when runing the tracking

By default taking the indices of the elements we want as a samplers but one can change the select parameter to find samplers with names or types

GenerateSamplers(*nb*)

Add multiple samplers spread evenly along the lattice

LoadBdsimTrack(*inputfilename*)

Load Bdsim root file and generate orbit files for each particle in the track collection
Then store the data in a structure similar to the pymad8 generated one

LoadMad8Track(*inputfilename*)

Load the Mad8 track files and store it in a structure similar to the pymad8 generated data
/!/ NOT WORKING /!/

PlotCorrelation(*index*, *coord*, *ref_index*, *ref_coord*, *linFit=False*, *noPlots=False*)

Correlation plot for a given coordinate at the closest sampler from given S
By default the reference sampler is LUXE IP

PlotHist(*S*, *coord*)

Plot histogram for a given coordinate at the closest sampler from given S

PlotPhaseSpace(*S*, *coord*, *linFit=False*)

Phase space plot for a given coordinate at the closest sampler from given S

PlotTrajectory(*particle*, *coord*, *bdsimCompare=False*, *relativePlots=False*)

Trajectory plot for a given coordinate and for a given particle number

RunPymad8Tracking(*track_collection*, *turns=1*)

Run tracking for all particles using the rmat from Mad8
>>> track.RunPymad8Tracking(track_collection)

reduceDFbyIndex(*index*)

pymad8.Sim.**setSamplersAndTrack**(*twissfile*, *rmatfile*, *nb_sampl*)

pymad8.Sim.**setTrackCollection**(*nb_part*, *energy*, *paramdict*)

pymad8.Sim.**testTrack**(*twissfile*, *rmatfile*)

9.5 pymad8.Track module

pymad8.Track.**MakeObserveFile**(*elementlist*, *filename*)

pymad8.Track.**MakeTableArchiveFile**(*elementlist*, *filename*)

pymad8.Track.**MakeTableMapFile**(*observeElements*, *observeIndex*, *filename*)

pymad8.Track.**MakeTrackCallingFile**(*fileNameStub*)

pymad8.Track.**MakeTrackFiles**(*savelineFileName*, *line*, *outputFileNameStub*)

9.6 pymad8.Visualisation module

pymad8.Visualisation.**MakeCombinedSurveyPlot**(*name*, *QUAD=True*, *RBEN=True*, *SBEN=True*, *MONI=True*, *MARK=True*)

Takes a list of Survey filenames, plots them all on the same 2D plot. For branching machines or segmented models. Elements selectable via booleans, default to true

class pymad8.Visualisation.**OneDim**(*survey*, *debug*)

Bases: object

plot(*colour=True*)

class pymad8.Visualisation.**TwoDim**(*survey*, *debug=False*, *annotate=False*, *fancy=False*)

Bases: object

plot(*event=None*)

plotUpdate(*event*)

pymad8.Visualisation.**testOneDim**()

pymad8.Visualisation.**testTwoDim**()

pymad8.Visualisation.**transformedPoly**(*xy*, *xyc*, *theta*)

pymad8.Visualisation.**transformedRect**(*xyc*, *dx*, *dy*, *theta*)

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

O

P

R

S

T

W