pyXDSM

unknown

Jan 05, 2022

USER GUIDE

1 Introduction 1

2 How to use it 3

Index 9

CHAPTER

ONE

INTRODUCTION

`pyXDSM` is a python library to generate XDSM diagrams in high-quality pdf format.

1.1 What is XDSM?

The eXtended Design Structure Matrix (XDSM) is a graphical language for describing the movement of data and the execution sequence for a multidisciplinary optimization problem. You can read the paper by Lambe and Martins for all the details.

CHAPTER

TWO

HOW TO USE IT

The following pages provide detailed info on how to use the python library:

2.1 Installation

This package is available on *pyPI* and can be installed using:

```
pip install pyxdsm
```

Alternatively, clone this repo or download the zip and unzip it, then:

```
cd pyxdsm
pip install .
```

2.2 Examples

Here is a simple example. There are some other more advanced things you can do as well. Check out the examples folder for more complex scripts.

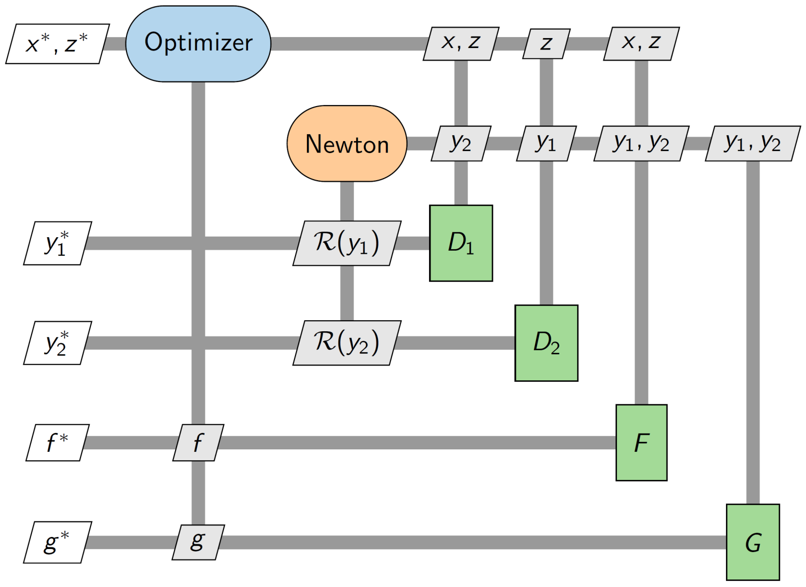
```
from pyxdsm.XDSM import XDSM, OPT, SOLVER, FUNC, LEFT

# Change `use_sfmath` to False to use computer modern
x = XDSM(use_sfmath=True)

x.add_system("opt", OPT, r"\text{Optimizer}")
x.add_system("solver", SOLVER, r"\text{Newton}")
x.add_system("D1", FUNC, "D_1")
x.add_system("D2", FUNC, "D_2")
x.add_system("F", FUNC, "F")
x.add_system("G", FUNC, "G")

x.connect("opt", "D1", "x, z")
x.connect("opt", "D2", "z")
x.connect("opt", "F", "x, z")
x.connect("solver", "D1", "y_2")
x.connect("solver", "D2", "y_1")
x.connect("D1", "solver", r"\mathcal{R}(y_1)")
x.connect("solver", "F", "y_1, y_2")
```

(continues on next page)

(continued from previous page)

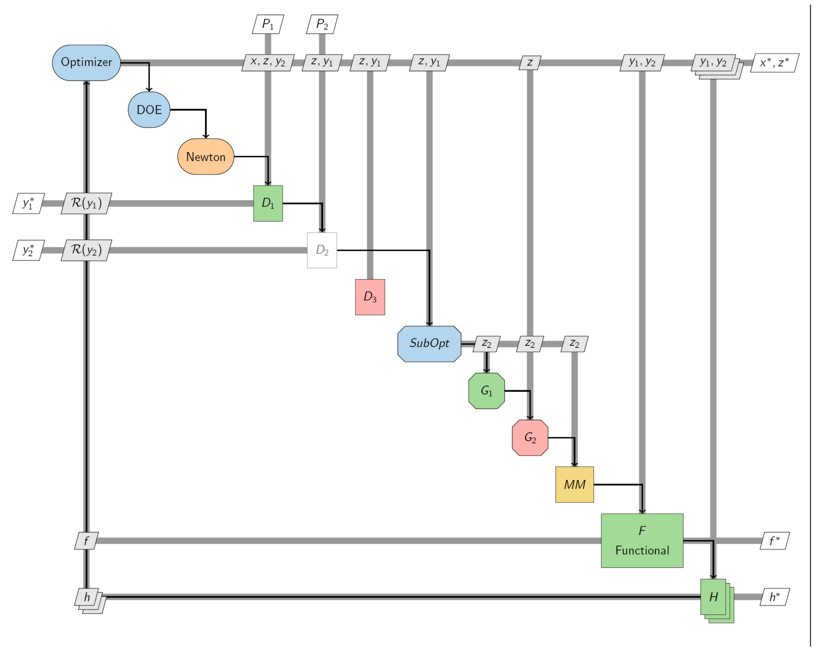
```
x.connect("D2", "solver", r"\mathcal{R}(y_2)")
x.connect("solver", "G", "y_1, y_2")

x.connect("F", "opt", "f")
x.connect("G", "opt", "g")

x.add_output("opt", "x^*, z^*", side=LEFT)
x.add_output("D1", "y_1^*", side=LEFT)
x.add_output("D2", "y_2^*", side=LEFT)
x.add_output("F", "f^*", side=LEFT)
x.add_output("G", "g^*", side=LEFT)
x.write("mdf")
```

This will output `mdf.tex`, a standalone tex document that (by default) is also compiled to `mdf.pdf`, shown below:



2.2.1 More complicated example

Here is an example that uses a whole bunch of the more advanced features in `pyXDSM`.



It is mostly just a reference for all the customizations you can do. The code for this diagram is provided here

2.3 pyXDSM API

class pyxdsm.XDSM.**XDSM**(*use_sfmath=True*)

add_input(*name*, *label*, *label_width=None*, *style='DataIO'*, *stack=False*)
Add an input, which will appear in the top row of the diagram.

Parameters

name [str] The unique name given to this component

label [str or list/tuple of strings] The label to appear on the diagram. There are two options for this: - a single string - a list or tuple of strings, which is used for line breaking In either case, they should probably be enclosed in ext{ } declarations to make sure the font is upright.

label_width [int or None] If not None, AND if `label` is given as either a tuple or list, then this parameter controls how many items in the tuple/list will be displayed per line. If None, the label will be printed one item per line if given as a tuple or list, otherwise the string will be printed on a single line.

style [str] The style given to this component. Can be one of ['DataInter', 'DataIO']

stack [bool] If true, the system will be displayed as several stacked rectangles, indicating the component is executed in parallel.

add_output(*name*, *label*, *label_width=None*, *style='DataIO'*, *stack=False*, *side='left'*)
Add an output, which will appear in the left or right-most column of the diagram.

Parameters

name [str] The unique name given to this component

label [str or list/tuple of strings] The label to appear on the diagram. There are two options for this: - a single string - a list or tuple of strings, which is used for line breaking In either case, they should probably be enclosed in ext{ } declarations to make sure the font is upright.

label_width [int or None] If not None, AND if `label` is given as either a tuple or list, then this parameter controls how many items in the tuple/list will be displayed per line. If None, the label will be printed one item per line if given as a tuple or list, otherwise the string will be printed on a single line.

style [str] The style given to this component. Can be one of `['DataInter', 'DataIO']`

stack [bool] If true, the system will be displayed as several stacked rectangles, indicating the component is executed in parallel.

side [str] Must be one of `['left', 'right']`. This parameter controls whether the output is placed on the left-most column or the right-most column of the diagram.

`add_process`(*systems*, *arrow=True*)
Add a process line between a list of systems, to indicate process flow.

Parameters

systems [list] The names of the components, in the order in which they should be connected. For a complete cycle, repeat the first component as the last component.

arrow [bool] If true, arrows will be added to the process lines to indicate the direction of the process flow.

`add_system`(*node_name*, *style*, *label*, *stack=False*, *faded=False*, *label_width=None*, *spec_name=None*)
Add a "system" block, which will be placed on the diagonal of the XDSM diagram.

Parameters

node_name [str] The unique name given to this component

style [str] The type of the component

label [str or list/tuple of strings] The label to appear on the diagram. There are two options for this: - a single string - a list or tuple of strings, which is used for line breaking In either case, they should probably be enclosed in ext{ } declarations to make sure the font is upright.

stack [bool] If true, the system will be displayed as several stacked rectangles, indicating the component is executed in parallel.

faded [bool] If true, the component will be faded, in order to highlight some other system.

label_width [int or None] If not None, AND if `label` is given as either a tuple or list, then this parameter controls how many items in the tuple/list will be displayed per line. If None, the label will be printed one item per line if given as a tuple or list, otherwise the string will be printed on a single line.

spec_name [str] The spec name used for the spec file.

`connect`(*src*, *target*, *label*, *label_width=None*, *style='DataInter'*, *stack=False*, *faded=False*)
Connects two components with a data line, and adds a label to indicate the data being transferred.

Parameters

src [str] The name of the source component.

target [str] The name of the target component.

label [str or list/tuple of strings] The label to appear on the diagram. There are two options for this: - a single string - a list or tuple of strings, which is used for line breaking In either case, they should probably be enclosed in ext{ } declarations to make sure the font is upright.

label_width [int or None] If not None, AND if `label` is given as either a tuple or list, then this parameter controls how many items in the tuple/list will be displayed per line. If None, the label will be printed one item per line if given as a tuple or list, otherwise the string will be printed on a single line.

style [str] The style given to this component. Can be one of `['DataInter', 'DataIO']`

stack [bool] If true, the system will be displayed as several stacked rectangles, indicating the component is executed in parallel.

faded [bool] If true, the component will be faded, in order to highlight some other system.

`write`(*file_name*, *build=True*, *cleanup=True*, *quiet=False*, *outdir='.'*)

Write output files for the XDSM diagram. This produces the following:

- **{file_name}.tikz** A file containing the TikZ definition of the XDSM diagram.
- **{file_name}.tex** A standalone document wrapped around an include of the TikZ file which can be compiled to a pdf.
- **{file_name}.pdf** An optional compiled version of the standalone tex file.

Parameters

file_name [str] The prefix to be used for the output files

build [bool] Flag that determines whether the standalone PDF of the XDSM will be compiled. Default is True.

cleanup [bool] Flag that determines if pdflatex build files will be deleted after build is complete

quiet [bool] Set to True to suppress output from pdflatex.

outdir [str] Path to an existing directory in which to place output files. If a relative path is given, it is interpreted relative to the current working directory.

`write_sys_specs`(*folder_name*)

Write I/O spec json files for systems to specified folder

An I/O spec of a system is the collection of all variables going into and out of it. That includes any variables being passed between systems, as well as all inputs and outputs. This information is useful for comparing implementations (such as components and groups in OpenMDAO) to the XDSM diagrams.

The json spec files can be used to write testing utilities that compare the inputs/outputs of an implementation to the XDSM, and thus allow you to verify that your codes match the XDSM diagram precisely. This technique is especially useful when large engineering teams are collaborating on model development. It allows them to use the XDSM as a shared contract between team members so everyone can be sure that their codes will sync up.

Parameters

folder_name: str name of the folder, which will be created if it doesn't exist, to put spec files into

2.4 TikZ and LaTeX

You need to install these libraries for pyXDSM to work. See the install guide for your platform.

2.4.1 Embedding the diagram directly in LaTeX

In addition, the file, `mdf.tikz`, can be embedded in another *.tex* file using the `\input` command:

```
\begin{figure}
\caption{Example of an MDF XDSM.}
\centering
\input{mdf.tikz}
\label{fig:xdsm}
\end{figure}
```

The following is required to be in the preamble of the document:

```
\usepackage{geometry}
\usepackage{amsfonts}
\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{tikz}

\usetikzlibrary{arrows,chains,positioning,scopes,shapes.geometric,shapes.misc,shadows}
```

INDEX

A

add_input() (*pyxdsm.XDSM.XDSM method*), 5
add_output() (*pyxdsm.XDSM.XDSM method*), 5
add_process() (*pyxdsm.XDSM.XDSM method*), 6
add_system() (*pyxdsm.XDSM.XDSM method*), 6

C

connect() (*pyxdsm.XDSM.XDSM method*), 6

W

write() (*pyxdsm.XDSM.XDSM method*), 7
write_sys_specs() (*pyxdsm.XDSM.XDSM method*), 7

X

XDSM (*class in pyxdsm.XDSM*), 5