matpower-pip

Make MATPOWER installable from `pypi`. This package make `MATPOWER` copy (currently Version `7.1`) as `python package`. Use this package with `mypower` (the recommended way) or `oct2py` to run `MATPOWER` using octave client. `matlab.engine` is also supported. For the latest docs, read README on GitHub.

This project also listed on related links on matpower official website. Please visit that site to find other useful resources.

Installation

`matpower`

For downloading `MATPOWER` only (maybe you will run it using `matlab.engine` or any other method, or simply want an easy `MATPOWER` downloader):

```
pip install matpower
```

`oct2py` (Windows)

For callable `matpower` via `oct2py` (require octave on environment system `PATH`):

1. Download octave.
2. Install octave, write down the destination path.
3. Open Environment Variable. You can access it by pressing `Windows-Key`, type `edit the system environment variables`, and press `Enter` to search.
4. Add new Environment Variable to execute `octave-cli`. The path is likely to be `C:\Program Files\octave-5.2.0-w64\mingw64\bin\octave-cli.exe`.

   Variable name: `OCTAVE_EXECUTABLE`

   Variable value: `location:\\of\\octave\\bin\\octave-cli.exe`
5. Restart computer to make `os.environ` recognize the new path.
6. Install `matpower` that include `oct2py`.

   ```
   pip install matpower[octave]
   ```

Extra (require `oct2py` or `matlab.engine`)

If `oct2py` or `matlab.engine` is installed, `matpower.start_instance` can be used to run `octave` or `MATLAB` with `MATPOWER` path added. Default engine is `octave`. You also can use `mypower` for added functionality as shown in `mypower tutorial`.

```python
from matpower import start_instance

m = start_instance()
m.runpf()
```

```python
from matpower import start_instance

m = start_instance()
mpc = m.eval('case9', verbose=False)
mpc = m.runpf(mpc)
```

```python
from matpower import path_matpower

print(path_matpower) # matpower installation location
```

Since `mpc = m.runopf()` will make `mpc` contain unsupported `<object opf_model>`, we can avoid it by request maximum number of outputs using `nout='max_nout'` in octave.

```python
from matpower import start_instance

m = start_instance()

mpc = m.loadcase('case9')
mpopt = m.mpoption('verbose', 2)
[baseMVA, bus, gen, gencost, branch, f, success, et] = m.runopf(mpc, mpopt, nout='max_nout')
```

Alternatively, it would be better to not parse back value that will not be use on python using `oct2py .eval` method. Use `;` to avoid octave print output on running the command.

```python
# import start_instance to start matpower instance
from matpower import start_instance

# start instance
m = start_instance()

# use octave native to run some commands
m.eval("mpopt = mpoption('verbose', 2);")
m.eval("mpc = loadcase('case9');")
m.eval("r1 = runopf(mpc, mpopt);") # we avoid parse `r1` that containts unsupported `<objec

# fech data to python (.eval is used because .pull is not working in acessing field)
r1_mpc = {}
r1_mpc['baseMVA'] = m.eval('r1.baseMVA;')
r1_mpc['version'] = m.eval('r1.version;')
r1_mpc['bus'] = m.eval('r1.bus;')
r1_mpc['gen'] = m.eval('r1.gen;')
r1_mpc['branch'] = m.eval('r1.branch;')
```

```
r1_mpc['gencost'] = m.eval('r1.gencost;')

# modify variable if necessary
[GEN_BUS, PG, QG, QMAX, QMIN, VG, MBASE, GEN_STATUS, PMAX, PMIN, MU_PMAX,
 MU_PMIN, MU_QMAX, MU_QMIN, PC1, PC2, QC1MIN, QC1MAX, QC2MIN, QC2MAX,
 RAMP_AGC, RAMP_10, RAMP_30, RAMP_Q, APF] = m.idx_gen(nout='max_nout')
gen_index = 2 # index of generator to be changed
gen_index_ = int(gen_index - 1) # -1 due to python indexing start from 0
PMAX_ = int(PMAX -1) # -1 due to python indexing start from 0
r1_mpc['gen'][gen_index_,PMAX_] = 110 # in this example, we modify PMAX to be 110

[PQ, PV, REF, NONE, BUS_I, BUS_TYPE, PD, QD, GS, BS,
 BUS_AREA, VM, VA, BASE_KV, ZONE, VMAX, VMIN, LAM_P,
 LAM_Q, MU_VMAX, MU_VMIN] = m.idx_bus(nout='max_nout')
bus_index = 7 # index of bus to be changed
bus_index_ = int(bus_index - 1) # -1 due to python indexing start from 0
PD_ = int(PD-1) # -1 due to python indexing start from 0
r1_mpc['bus'][bus_index_,int(PD-1)] = 80 # in this example, we modify PD to be 150

# push back value to octave client
m.push('mpc', r1_mpc) # push r1_mpc in python to mpc in octave

# test if we can retrive pushed value
mpc = m.pull('mpc')

# test if our pushed variable can be used
m.eval("r1 = runopf(mpc, mpopt);")
```

Also support using `matlab.engine`.

```
from matpower import start_instance

m = start_instance(engine='matlab') # specify using `matlab.engine` instead of `oct2py`
mpc = m.runpf('case5', nargout=0)
```

You can explore other example on notebooks.

Known engine issue

Octave

1. `m.runopf()` will make `mpc` contain unsupported `<object opf_model>`. See: https://github.com/MATPOWER/matpower/issues/134#issuecomment-1007798733

   Impacted case:

   ```
   r1 = m.runopf(mpc)
   ```

Solution:

```
m.push('mpc', mpc)
m.eval("r1 = runopf(mpc, mpopt);")

r1_mpc = {}
r1_mpc['baseMVA'] = m.eval('r1.baseMVA;')
r1_mpc['version'] = m.eval('r1.version;')
r1_mpc['bus'] = m.eval('r1.bus;')
r1_mpc['gen'] = m.eval('r1.gen;')
r1_mpc['branch'] = m.eval('r1.branch;')
r1_mpc['gencost'] = m.eval('r1.gencost;')
```

Versioning

This package maintain `MATPOWER` version with added version mark, i.e. `MATPOWER 7.1` become `7.1.0.x.x.x` where `.x.x.x` come from `matpower-pip` versioning. The `matpower-pip` versioning is not released on `pypi` since `matpower-pip` is restricted for development only (and development should use git instead).

TODO

1. `conda` and `docker` installation that include octave-cli installation.

Authors

- **Muhammad Yasirroni** - yasirroni

Cite

We do request that publications derived from the use of `matpower-pip` explicitly acknowledge that fact by including all related MATPOWER publication and the following citation:

> M. Yasirroni, Sarjiya, “matpower-pip: Make MATPOWER installable from pypi”, GitHub, 2021. [Online]. Available: https://github.com/yasirroni/matpower-pip.

If a journal publication from the author to appear soon should be cited instead.

Acknowledgement

This repository was supported by the Faculty of Engineering, Universitas Gadjah Mada under the supervision of Mr. Sarjiya. If you use this package, we are very glad if you cite any relevant publication under Mr. Sarjiya’s name that can be found on the semantic scholar or IEEE for the meantime, since publication related to this repository is ongoing. This work is also partly motivated after I found out that `oct2py` supports running `octave` client from `python`, but the

only implementation for running `MATPOWER` that I know, that is oct2pypower, requires `docker` and is not newbie-friendly. Nevertheless, I would like to say thank you to all people who contributed to `oct2py`, `oct2pypower`, and more importantly `MATPOWER`.