fuefit Documentation

Release 0.0.6

Kostis Anagnostopoulos

December 05, 2014

Contents

Release 0.0.6

Documentation https://fuefit.readthedocs.org/

Source https://github.com/ankostis/fuefit

PyPI repo https://pypi.python.org/pypi/fuefit

Keywords automotive, car, cars, consumption, engine, engine-map, fitting, fuel, vehicle, vehicles

Fuefit is a python package that calculates fitted fuel-maps from measured engine data-points based on coefficients with physical meaning.

CHAPTER 1

Introduction

1.1 Overview

The *Fuefit* calculator was developed to apply a statistical fit on measured engine fuel consumption data (engine map). This allows the reduction of the information necessary to describe an engine fuel map from several hundred points to seven statistically calculated parameters, with limited loss of information.

More specifically this software works like that:

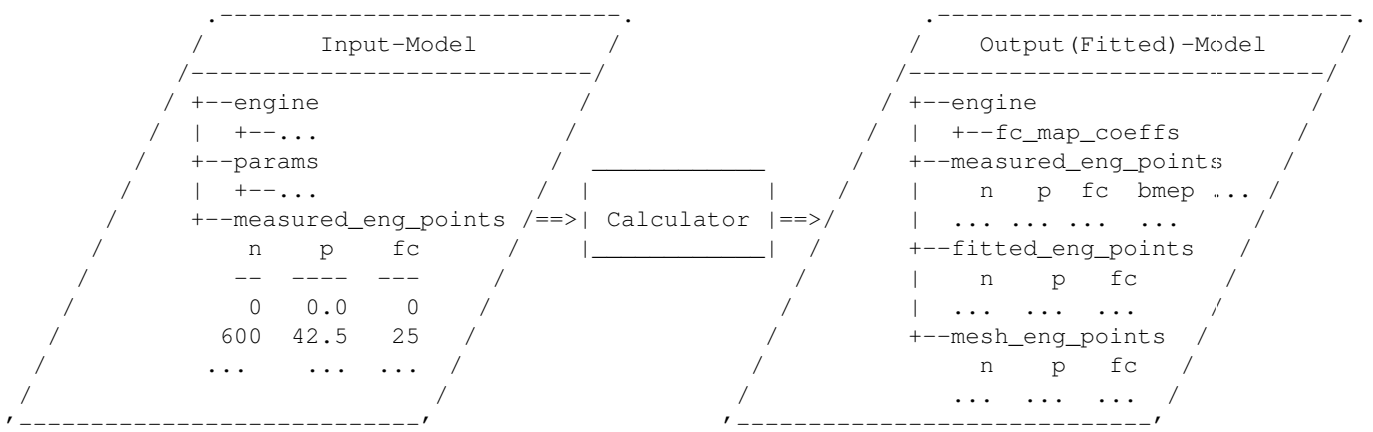
1. **Accepts engine data as input**, constituting of triplets of RPM, Power and Fuel-Consumption or equivalent quantities eg mean piston speed (CM), brake mean effective pressure (BMEP) or Torque, fuel mean effective pressure (PMF).
2. **Fits the provided input** to the following formula [1] [2] [3]:

$$\mathbf{BMEP} = (a + b \times \mathbf{CM} + c \times \mathbf{CM^2}) \times \mathbf{PMF} + (a2 + b2 \times \mathbf{CM}) \times \mathbf{PMF^2} + loss0 + loss2 \times \mathbf{CM^2}$$

3. **Recalculates and (optionally) plots engine-maps** based on the coefficients that describe the fit:

$$a, b, c, a2, b2, loss0, loss2$$

An "execution" or a "run" of a calculation along with the most important pieces of data are depicted in the following diagram:



[1] Bastiaan Zuurendonk, Maarten Steinbuch(2005): "Advanced Fuel Consumption and Emission Modeling using Willans line scaling techniques for engines", *Technische Universiteit Eindhoven*, 2005, Department Mechanical Engineering, Dynamics and Control Technology Group, http://alexandria.tue.nl/repository/books/612441.pdf

[2] Yuan Zou, Dong-ge Li, and Xiao-song Hu (2012): "Optimal Sizing and Control Strategy Design for Heavy Hybrid Electric Truck", *Mathematical Problems in Engineering* Volume 2012, Article ID 404073, 15 pages doi:10.1155/2012/404073

[3] Xi Wei (2004): "Modeling and control of a hybrid electric drivetrain for optimum fuel economy, performance and driveability", Dissertation Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

Apart from various engine-characteristics under `/engine` the table-columns such as `capacity` and `p_rated`, the table under `/measured_eng_points` must contain *at least* one column from each of the following categories (column-headers are case-insensitive):

1. Engine-speed:

```
N         [1/min]
N_norm    [-]         : where N_norm = (N - N_idle) / (N_rated-N_idle)
CM        [m/sec]
```

2. Load-Power-capability:

```
P         [kW]
P_norm    [-]         : where P_norm = P/P_MAX
T         [Nm]
BMEP      [bar]
```

3. Fuel-consumption:

```
FC        [g/h]
FC_norm   [g/KWh]     : where FC_norm = FC[g/h] / P_MAX [kW]
PMF       [bar]
```

The *Input & fitted data-model* described above are trees of strings and numbers, assembled with:

- sequences,
- dictionaries,
- `pandas.DataFrame`,
- `pandas.Series`.

1.2 Quick-start

The program runs on **Python-3.3+** and requires *numpy/scipy*, *pandas* and *win32* libraries along with their native backends to be installed.

On *Windows*/*OS X*, it is recommended to use one of the following "scientific" python-distributions, as they already include the native libraries and can install without administrative priviledges:

- WinPython (*Windows* only),
- Anaconda,
- Canopy,

Assuming you have a working python-environment, open a *command-shell* (in *Windows* use **cmd.exe** BUT ensure **python.exe** is in its `PATH`) and try the following *console-commands*:

Install

```
$ pip install fuefit
$ fuefit --winmenus                          ## Adds StartMenu-items, Windows only.
```

See: *Install*

Cmd-line

```
$ fuefit --version
0.0.6

$ fuefit --help
...

## Change-directory into the 'fuefit/test/' folder in the  *sources*.
```

```
$ fuefit -I FuelFit_real.csv header+=0 \
    -I ./FuelFit.xlsx sheetname+=0 header@=None names:='["p","n","fc"]' \
    -I ./engine.csv file_frmt=SERIES model_path=/engine header@=None \
    -m /engine/fuel=petrol \
    -m /params/plot_maps@=True \
    -O full_results_model.json \
    -O fit_coeffs.csv model_path=/engine/fc_map_coeffs   index?=false \
    -O t1.csv model_path=/measured_eng_points   index?=false \
    -O t2.csv model_path=/mesh_eng_points       index?=false \
```

See: *Cmd-line usage*

Excel

```
$ fuefit --excelrun                                          ## Windows & OS X only
```

See: *Excel usage*

Python-code

```
>>> import pandas as pd
>>> from fuefit import datamodel, processor, test

>>> inp_model = datamodel.base_model()
>>> inp_model.update({...})                                  ## See "Python Usage" bel
>>> inp_model['engine_points'] = pd.read_csv('measured.csv')    ## Pandas can read Excel,
>>> datamodel.set_jsonpointer(inp_model, '/params/plot_maps', True)

>>> datamodel.validade_model(inp_model, additional_properties=False)

>>> out_model = processor.run(inp_model)

>>> print(datamodel.resolve_jsonpointer(out_model, '/engine/fc_map_coeffs'))
a            164.110667
b           7051.867419
c          63015.519469
a2             0.121139
b2          -493.301306
loss0      -1637.894603
loss2   -1047463.140758
dtype: float64
```

See: *Python usage*

Tip: The commands beginning with `$`, above, imply a *Unix* like operating system with a *POSIX* shell (*Linux*, *OS X*). Although the commands are simple and easy to translate in its *Windows* counterparts, it would be worthwile to install Cygwin to get the same environment on *Windows*. If you choose to do that, include also the following packages in the *Cygwin*'s installation wizard:

```
* git, git-completion
* make, zip, unzip, bzip2
* openssh, curl, wget
```

But do not install/rely on cygwin's outdated python environment.

CM Mean Piston Speed, a measure for the engines operating speed [m/sec]

BMEP Brake Mean Effective Pressure, a valuable measure of an engine's capacity to do work that is independent of engine displacement) [bar]

PMF *Available Mean Effective Pressure*, the maximum mean effective pressure calculated based on the energy content of the fuel [bar]

JSON-schema The JSON schema is an IETF draft that provides a *contract* for what JSON-data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. You can learn more about it from this excellent guide, and experiment with this on-line validator.

JSON-pointer JSON Pointer(**RFC 6901**) defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document. It aims to serve the same purpose as *XPath* from the XML world, but it is much simpler.

CHAPTER 2

Install

Fuefit-0.0.6 runs on **Python-3.3+**, and it is distributed on Wheels.

Note: This project depends on the *numpy/scipy*, *pandas* and *win32* python-packages that themselfs require the use of *C* and *Fortran* compilers to build from sources. To avoid this hussle, you can choose instead a self-wrapped python distribution like *Anaconda/minoconda*, *Winpython*, or *Canopy*.

Tip:

- Under *Windows* you can try the self-wrapped WinPython distribution, a higly active project, that can even compile native libraries using an installations of *Visual Studio*, if available (required for instance when upgrading `numpy/scipy`, `pandas` or `matplotlib` with **pip**).

  Just remember to **Register your WinPython installation** after installation and **add your installation into** `PATH` (see *Frequently Asked Questions*):

  - To register it, go to *Start menu* → *All Programs* → *WinPython* → *WinPython ControlPanel*, and then *Options* → *Register Distribution* .
  - For the path, add or modify the registry string-key `[HKEY_CURRENT_USEREnvironment] "PATH"`.

- An alternative scientific python-environment is the Anaconda cross-platform distribution (*Windows*, *Linux* and *OS X*), or its lighter-weight alternative, miniconda.

  On this environment you will need to install this project's dependencies manually using a combination of **conda** and **pip** commands. See `requirements/miniconda.txt`, and peek at the example script commands in `.travis.yaml`.

- Check for alternative installation instructions on the various python environments and platforms at the pandas site.

See *Install* for more details

Before installing it, make sure that there are no older versions left over. So run this console-command (using **cmd.exe** in windows) until you cannot find any project installed:

```
$ pip uninstall fuefit                                    ## Use 'pip3' if both python-2 & 3 ar
```

You can install the project directly from the *PyPi* repo the "standard" way, by typing the **pip** in the console:

```
$ pip install fuefit
```

- If you want to install a *pre-release* version (the version-string is not plain numbers, but ends with `alpha`, `beta.2` or something else), use additionally `--pre`.
- If you want to upgrade an existing installation along with all its dependencies, add also `--upgrade` (or `-U` equivalently), but then the build might take some considerable time to finish. Also there is the possibility

the upgraded libraries might break existing programs(!) so use it with caution, or from within a *virtualenv* (isolated Python environment).

- To install an older version issue the console-command:

```
$ pip install fuefit=1.1.1                    ## Use '--pre' if version-string has a build-su
```

- To install it for different Python environments, repeat the procedure using the appropriate **python.exe** interpreter for each environment.
-

Tip: To debug installation problems, you can export a non-empty `DISTUTILS_DEBUG` and *distutils* will print detailed information about what it is doing and/or print the whole command line when an external program (like a C compiler) fails.

After a successful installation, it is important that you check which version is visible in your `PATH`, so type this console-command:

```
$ fuefit --version
0.0.6
```

2.1 Installing from sources (for advanced users familiar with *git*)

If you download the sources you have more options for installation. There are various methods to get hold of them:

- Download and extract a release-snapshot from github.
- Download and extract a `sdist` *source* distribution from *PyPi* repo.
- Clone the *git-repository* at *github*. Assuming you have a working installation of git you can fetch and install the latest version of the project with the following series of commands:

```
$ git clone "https://github.com/ankostis/fuefit.git" fuefit.git
$ cd fuefit.git
$ python setup.py install                                  ## Use 'python3' if both python-2 &
```

When working with sources, you need to have installed all libraries that the project depends on. Particularly for the latest *WinPython* environments (*Windows* / *OS X*) you can install the necessary dependencies with:

```
$ pip install -r requirements/execution.txt .
```

The previous command installs a "snapshot" of the project as it is found in the sources. If you wish to link the project's sources with your python environment, install the project in development mode:

```
$ python setup.py develop
```

Note: This last command installs any missing dependencies inside the project-folder.

2.2 Anaconda install

The installation to *Anaconda* (ie *OS X*) works without any differences from the `pip` procedure described so far.

To install it on *miniconda* environment, you need to install first the project's *native* dependencies (numpy/scipy), so you need to download the sources (see above). Then open a *bash-shell* inside them and type the following commands:

```
$ coda install `cat requirements/miniconda.txt`
$ pip install lmfit             ## Workaround lmfit-py#149
$ python setup.py install
```

```
$ fuefit --version
0.0.6
```

CHAPTER 3

Usage

3.1 Excel usage

Attention: Excel-integration requires Python 3 and *Windows* or *OS X*!

In *Windows* and *OS X* you may utilize the xlwings library to use Excel files for providing input and output to the program.

To create the necessary template-files in your current-directory, type this console-command:

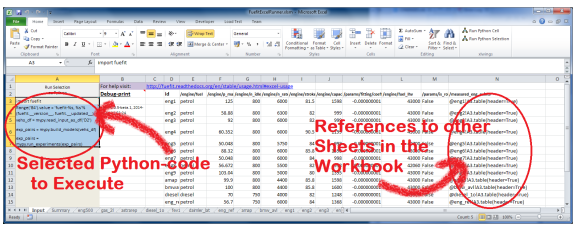
```
$ fuefit --excel
```

Type `fuefit --excel file_path` if you want to specify a different destination path.

In *windows*/*OS X* you can type `fuefit --excelrun` and the files will be created in your home-directory and the Excel will immediately open them.

What the above commands do is to create 2 files:

`FuefitExcelRunner#.xlsm` The python-enabled excel-file where input and output data are written, as seen in the screenshot below:



After opening it the first tie, enable the macros on the workbook, select the python-code at the left and click the *Run Selection as Pyhon* button; one sheet per vehicle should be created.

The excel-file contains additionally appropriate *VBA* modules allowing you to invoke *Python code* present in *selected cells* with a click of a button, and python-functions declared in the python-script, below, using the `mypy` namespace.

To add more input-columns, you need to set as column *Headers* the *json-pointers* path of the desired model item (see *Python usage* below,).

`FuefitExcelRunner#.py` Python functions used by the above xls-file for running a batch of experiments.

The particular functions included reads multiple vehicles from the input table with various vehicle characteristics and/or experiment coefficients, and then it adds a new worksheet containing the cycle-run of each vehicle . Of course you can edit it to further fit your needs.

Note: You may reverse the procedure described above and run the python-script instead:

```
$ python FuefitExcelRunner.py
```

The script will open the excel-file, run the experiments and add the new sheets, but in case any errors occur, this time you can debug them, if you had executed the script through LiClipse, or *IPython*!

Some general notes regarding the python-code from excel-cells:

- An elaborate syntax to reference excel *cells*, *rows*, *columns* or *tables* from python code, and to read them as `pandas.DataFrame` is utilized by the Excel . Read its syntax at `resolve_excel_ref()`.
- On each invocation, the predefined VBA module `pandalon` executes a dynamically generated python-script file in the same folder where the excel-file resides, which, among others, imports the "sister" python-script file. You can read & modify the sister python-script to import libraries such as 'numpy' and 'pandas', or pre-define utility python functions.
- The name of the sister python-script is automatically calculated from the name of the Excel-file, and it must be valid as a python module-name. Therefore: * Do not use non-alphanumeric characters such as spaces(' `), dashes(-) and dots(.`') on the Excel-file. * If you rename the excel-file, rename also the python-file, or add this python `import <old_py_file> as mypy`'
- On errors, a log-file is written in the same folder where the excel-file resides, for as long as **the message-box is visible, and it is deleted automatically after you click 'ok'!**
- Read http://docs.xlwings.org/quickstart.html

3.2 Cmd-line usage

Example command:

```
fuefit -v\
  -I fuefit/test/FuelFit.xlsx sheetname+=0 header@=None names:='["p","rpm","fc"]' \
  -I fuefit/test/engine.csv file_frmt=SERIES model_path=/engine header@=None \
  -m /engine/fuel=petrol \
  -O ~t2.csv model_path=/fitted_eng_points    index?=false \
  -O ~t2.csv model_path=/mesh_eng_points      index?=false \
  -O ~t.csv model_path= -m /params/plot_maps@=True
```

3.3 Python usage

The most powerful way to interact with the project is through a python REPL (Read-Eval-Print Loop). So fire-up a **python** or **ipython** shell and first try to import the project just to check its version:

```
>>> import fuefit

>>> fuefit.__version__                  ## Check version once more.
'0.0.6'

>>> fuefit.__file__                     ## To check where it was installed.
/usr/local/lib/site-package/fuefit-...
```

If the version was as expected, take the **base-model** and extend it with your engine-data (strings and numbers):

```
>>> from fuefit import datamodel, processor

>>> inp_model = datamodel.base_model()
>>> inp_model.update({
...     "engine": {
...         "fuel":     "diesel",
...         "p_max":    95,
```

```
...         "n_idle":   850,
...         "n_rated":  6500,
...         "stroke":   94.2,
...         "capacity": 2000,
...         "bore":     None,       ##You do not have to include these,
...         "cylinders": None,      ##  they are just for displaying some more engine properties.
...     }
... })

>>> import pandas as pd
>>> df = pd.read_excel('fuefit/test/FuelFit.xlsx', 0, header=None, names=["n","p","fc"])
>>> inp_model['measured_eng_points'] = df
```

For information on the accepted model-data, check both its *JSON-schema* at `model_schema()`, and the `base_model()`:

Next you have to *validate* it against its *JSON-schema*:

```
>>> datamodel.validate_model(inp_model, additional_properties=False)
```

If validation is successful, you may then feed this model-tree to the `fuefit.processor`, to get back the results:

```
>>> out_model = processor.run(inp_model)

>>> print(datamodel.resolve_jsonpointer(out_model, '/engine/fc_map_coeffs'))
a            164.110667
b           7051.867419
c          63015.519469
a2             0.121139
b2          -493.301306
loss0      -1637.894603
loss2   -1047463.140758
dtype: float64

>>> print(out_model['fitted_eng_points'].shape)
(262, 11)
```

Hint: You can always check the sample code at the Test-cases and in the cmdline tool `fuefit.__main__`.

3.3.1 Fitting Parameterization

The 'lmfit' fitting library can be parameterized by setting/modifying various input-model properties under `/params/fitting/`.

In particular under `/params/fitting/coeffs/` you can set a dictionary of *coefficient-name* –> `lmfit.parameters.Parameter` such as `min/max/value`, as defined by the *lmfit* library (check the default props under `fuefit.datamodel.base_model()` and the example columns in the *ExcelRunner*).

See also:

http://lmfit.github.io/lmfit-py/parameters.html#Parameters

CM Mean Piston Speed, a measure for the engines operating speed [m/sec]

BMEP Brake Mean Effective Pressure, a valuable measure of an engine's capacity to do work that is independent of engine displacement) [bar]

PMF *Available Mean Effective Pressure*, the maximum mean effective pressure calculated based on the energy content of the fuel [bar]

JSON-schema The JSON schema is an IETF draft that provides a *contract* for what JSON-data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. You can learn more about it from this excellent guide, and experiment with this on-line validator.

JSON-pointer JSON Pointer(**RFC 6901**) defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document. It aims to serve the same purpose as *XPath* from the XML world, but it is much simpler.

CHAPTER 4

Contribute

This project is hosted in **github**. To provide feedback about bugs and errors or questions and requests for enhancements, use github's Issue-tracker.

4.1 Sources & Dependencies

To get involved with development, you need a POSIX environment to fully build it (*Linux*, *OSX*, or *Cygwin* on *Windows*).

Liclipse IDE

Within the sources there are two sample files for the comprehensive LiClipse IDE:

- `eclipse.project`
- `eclipse.pydevproject`

Remove the `eclipse` prefix, (but leave the dot(`.`)) and import it as "existing project" from Eclipse's `File` menu.

Another issue is due to the fact that LiClipse contains its own implementation of *Git*, *EGit*, which badly interacts with unix *symbolic-links*, such as the `docs/docs`, and it detects working-directory changes even after a fresh checkout. To workaround this, Right-click on the above file *Properties* → *Team* → *Advanced* → *Assume Unchanged*

4.2 Development team

- Kostis Anagnostopoulos (software design & implementation)
- Georgios Fontaras (methodology inception, engineering support & validation)

4.2.1 Contributing Authors

- Stefanos Tsiakmakis
- Biagio Ciuffo
- Alessandro Marotta

Authors would like to thank experts of the SGS group for providing useful feedback.

CHAPTER 5

Frequently Asked Questions

5.1 General

5.1.1 Can I copy/extend it? What is its License, in practical terms?

I'm not a lawyer, but in a broad view, the core algorithm of the project is "copylefted" with the *EUPL-1.1+ license*, and it includes files from other "non-copyleft" open source licenses like *MIT MIT License* and *Apache License*, appropriately marked as such. So in an nutshell, you can study it, copy it, modify or extend it, and distrbute it, as long as you always distribute the sources of your changes.

5.2 Technical

5.2.1 I followed the instructions but i still cannot install/run/get *X*. What now?

If you have no previous experience in python, setting up your environment and installing a new project is a demanding, but manageable, task. Here is a checklist of things that might go wrong:

- Did you send each command to the **appropriate shell/interpreter**?

  You should enter sample commands starting `$` into your *shell* (**cmd** or **bash**), and those starting with >>> into the *python-interpreter* (but don't include the previous symbols and/or the *output* of the commands).

- Is **python contained in your PATH** ?

  To check it, type `python` in your console/command-shell prompt and press `[Enter]`. If nothing happens, you have to inspect `PATH` and modify it accordingly to include your python-installation.

  - Under *Windows* type `path` in your command-shell prompt. To change it, run **regedit.exe** and modify (or add if not already there) the `PATH` string-value inside the following *registry-setting*:

    `HKEY_CURRENT_USER\Environment\`

    You need to logoff and logon to see the changes.

    Note that *WinPython* **does not modify your path!** if you have registed it, so you definetely have to perform the the above procedure yourself.

  - Under *Unix* type `echo $PATH$` in your console. To change it, modify your "rc' files, ie: `~/.bashrc` or `~/.profile`.

- Is the correct **version of python** running? Of **fuefit**??

  Certain commands such as **pip** come in 2 different versions *python-2 & 3* (**pip2** and **pip3**, respectively). Most programs report their version-infos with `--version`. Use `--help` if this does not work.

- Have you **upgraded/downgraded the project** into a more recent/older version?

  This project is still in development, so the names of data and functions often differ from version to version. Check the *Changes* for point that you have to be aware of when upgrading.

- Did you try **verbose reporting** for the command-line tool?
  - Use `-v` of `--vv` to receive log-messages.
  - Use `-d` to enable debug-checks.
- Did you search whether **a similar issue** has already been reported?
- Did you **ask google** for an answer??
- If the above suggestions still do not work, feel free to **open a new issue** and ask for help. Write down your platform (Windows, OS X, Linux), your exact python distribution and version, and include the *print-out of the failed command along with its error-message.*

  This last step will improve the documentation and help others as well.

CHAPTER 6

API reference

Content below is automatically produced from `docstrings` in the sources, and needs more work...

6.1 Core

pdcalc
datamodel
processor

6.2 ExcelRunner

FuefitExcelRunner

6.3 Tests

cmdline_test

6.4 Module: fuefit.datamodel

6.5 Module: fuefit.processor

6.6 Module: fuefit.pdcalc

6.7 Module: fuefit.excel.FuefitExcelRunner

6.8 Module: fuefit.test.cmdline_test

CHAPTER 7

Changes

Contents

7.1 Releases

7.1.1 v0.0.6, X-X-X – Maintenance release

- build: Untrack exclipse-project files.
- docs: Improve installation instructions and review of scientific content.
- model: Move /params/is_robust –> ./fitting/is_robust
- model: Rename fit-coefficient PMF –> BMEP.

7.1.2 v0.0.5, 12-Noe-2014 – 3rd public (Rosetta) release

- core: Use `lmfit` library for enforcing limits on fitted coefficients, etc.
- data: Updated Excel file with more engines.
- docs: Fix math-formulas and improve instructions.
- WARN: ExcelRunner fails on *OS X*.

7.1.3 v0.0.4, 10-Noe-2014 – 2nd public (beta) release

- core: FIX calclulations.
- core: Possible to specify whether to Robust-fit or not.
- core: Pin `b0` coefficient to 0.
- excel: Enhance excel-runner code to support any python-code.

- excel: FIX parsing of ExcelRefs and their syntax documentation.
- test: Improve tests and Doctest code in README.
- test, ci: Use TravisCI/Anaconda Continuous-integration to check project health.
- docs: Add "API-reference" section.
- docs: Add some "Anaconda" help.
- NOTE: Various renames of modules, files and model properties.

7.1.4 v0.0.3, 03-Noe-2014 – 1st public (beta) release

- excel: Add excel-runner for running batch of experiments.
- cmd: Rename fuefitcmd –> fuefit (back again)
- cmd: Add StartMenu item in *Windows*.
- build: Distribute on Wheels and Docs-archive.
- build: Upload to Github/RTD/PyPi.

7.1.5 v0.0.2, 28-Oct-2014 – Beta release

- Add Excel-UI.
- cmd: Rename fuefit –> fuefitcmd
- core,model: Rename rpm_XXX –> n_XXX, etc.
- docs: Update README with excel capability, copy sections from wltp project.
- build: Stop building as EXE.
- build: Add WinPython-deps as a requirments.txt.
- Add sphinx documentation.
- Relicense from AGPL –> EUPL.

7.1.6 v0.0.1, 25-Jul-2014 – Alpha release

- Implemented algorithm using `pdcalc`.
- **pdcalc: Implemented library that decides what to calculate with a topological sorting of** required calculations from Input –> Output, ala-Excel.
- Packaged as EXE.

7.1.7 v0.0.0, 15-Apr-2014 – Alpha release

- Project administerial: README, INSTALL, setup.py mostly transcopied from wtlc

CHAPTER 8

Indices

CM Mean Piston Speed, a measure for the engines operating speed [m/sec]

BMEP Brake Mean Effective Pressure, a valuable measure of an engine's capacity to do work that is independent of engine displacement) [bar]

PMF *Available Mean Effective Pressure*, the maximum mean effective pressure calculated based on the energy content of the fuel [bar]

JSON-schema The JSON schema is an IETF draft that provides a *contract* for what JSON-data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. You can learn more about it from this excellent guide, and experiment with this on-line validator.

JSON-pointer JSON Pointer(**RFC 6901**) defines a string syntax for identifying a specific value within a JavaScript Object Notation (JSON) document. It aims to serve the same purpose as *XPath* from the XML world, but it is much simpler.

8.1 *Index*

B

C

D

E

J

P

R