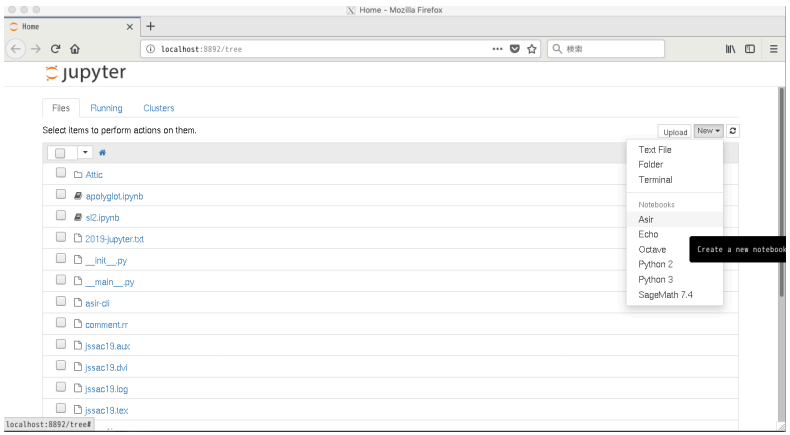
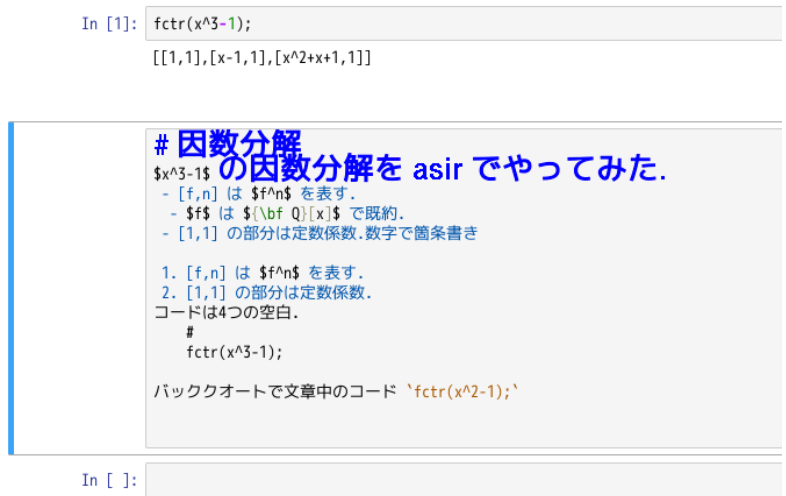
Jupyter notebook への asir カーネルの実装 (実装報告)

高山信毅 (神戸大)

Jupyter Notebook とは. まずはデモ





jupyter とカーネルは http でブラウザと通信するサーバー.
Notebook はブラウザで動く.

サーバの起動

```
[I 14:03:15.338 NotebookApp] The port 8888 is already in use, trying another port.
...
[I 14:03:15.348 NotebookApp] Serving notebooks from local directory: /amd/orange2b/usr/h
[I 14:03:15.348 NotebookApp] 0 active kernels
[I 14:03:15.348 NotebookApp] The Jupyter Notebook is running at: http://localhost:8892/
[I 14:03:15.348 NotebookApp] Use Control-C to stop this server and shut down all kernels
[I 14:04:03.397 NotebookApp] Creating new notebook in
[I 14:04:07.182 NotebookApp] Kernel started: 0c139a88-9f48-4519-bc96-a73641055ced
[I 14:04:12.810 NotebookApp] Adapting to protocol v5.1 for kernel 0c139a88-9f48-4519-bc9
[MetaKernelApp] ERROR | No such comm target registered: jupyter.widget.version
...
...
[I 14:24:07.177 NotebookApp] Saving file at /Untitled.ipynb
[I 14:25:32.077 NotebookApp] Kernel shutdown: 0c139a88-9f48-4519-bc96-a73641055ced

ctrl-C
[I 14:26:09.766 NotebookApp] interrupted
Serving notebooks from local directory: /amd/orange2b/usr/home/home01/taka/Texts/text-20
0 active kernels
The Jupyter Notebook is running at: http://localhost:8892/
Shutdown this notebook server (y/[n])? y
[C 14:26:11.151 NotebookApp] Shutdown confirmed
[I 14:26:11.152 NotebookApp] Shutting down kernels
```

Markdown, MathJax

Cell, Cell type, Markdown. `https://github.github.com/gfm/`.
shift + enter (または play ボタン) で整形. double click で markdown 編集に.

```
# 因数分解
$x^2-1$ の因数分解を asir でやってみた.
 - [f,n] は $f^n$ を表す.
  - $f$ は ${\bf Q}[x]$ で既約.
 - [1,1] の部分は定数係数.
数字で箇条書き
 1. [f,n] は $f^n$ を表す.
 2. [1,1] の部分は定数係数.
コードは sharp tab
# tab
    fctr(x^3-1);
バッククオートで文章中のコード `fctr(x^2-1);`
```

行番号の表示は esc l. sl2.ipynb でデモ.

開発史: 最近

Asir 合宿, 2019.03.27 の午後–2019.03.28 の午前. 実質 6 時間程度で, とりあえず Jupyter, asir kernel が動く. ちなみに私はこの時 python をほとんどはじめて使った (sage から asir を呼ぶコードを書くのにソースを今までちょっとみたことあり.) すぐできた理由:

1. とりあえず動くものを作るには python は簡単, 検索ですぐわかる. Qiita, stackoverflow, フリーのテキスト, ...
2. octave kernel を再利用.
3. OpenXM の sm1 toolkit を使った (cf. `ox_toolkit` (小原)). "OpenXM の設計はよいものだった" の実証.
4. cfep/asir, texmacs/asir, sage/asir(2018.09), Qt/asir(未公開, 藤本アドバイス) の経験があった.

次のページからは動くものができてから覚えたことの紹介.

kernel の作成方法: 概要

以下は `https://jupyter-client.readthedocs.io/en/stable/kernels.html`. Making kernel for Jupyter の日本語要約.
kernel はユーザのコードを実行したり, テストして調べたりする (introspect) ためのプログラム. IPython* は python コードのための kernel を含む. その他いろんな言語のカーネルあり.
Jupyter が kernel をスタートすると kernel に connection file を渡す. このファイルにはどのように frontend が Jupyter と通信するかの情報が書かれている.
kernel を書くための 2 つの道.

1. IPython によるカーネルを再利用する.
2. カーネルを scratch から書く.

*NT. 今はこの部分は jupyter と呼ばれてる模様

kernel の作成方法: connection file

Jupyter は kernel に対して connection file を渡す. これは Jupyter のユーザーしか読めない. connection file は次の例のように辞書形式 (下記は python 記法) のデータである.

```
{
  "transport": "tcp",
  "ip": "127.0.0.1",
  "control_port": 50160,
  "shell_port": 57503,
  "stdin_port": 52597,
  "hb_port": 42540,
  "iopub_port": 40885,
  "signature_scheme": "hmac-sha256",
  "key": "a0436f6c-1916-498b-8eb9-e81ab9368e84"
}
```

5 個の port は ZeroMQ `http://zeromq.org` (Distributed Messaging) を用いて Jupyter と kernel が情報をやりとりするためのもの. 上の例の場合 shell socket は `tcp://127.0.0.1:57503` となる.

`signature_scheme` と `key` はメッセージの暗号化に使われる. 詳しくは Wire Protocol: `https://jupyter-client.readthedocs.io/en/stable/messaging.html#wire-protocol` を見よ.

kernel の作成方法: kernel の仕様 (spec) 1

kernel の仕様を格納しておくフォルダは下記のフォルダ (unix の場合, Env は略).

システム	`/usr/share/jupyter/kernels` `/usr/local/share/jupyter/kernels`
ユーザ	`~/.local/share/jupyter/kernels` `~/Library/Jupyter/kernels` (Mac)

検索の順番は ユーザ, システムの順である. kernel の仕様を書いたファイルの名前は url フィールドで使われるので英数字および '-', '.', '_' のみでファイル名を記述すること.

kernel folder の中には `kernel.json`, `kernel.js`, およびロゴのイメージファイル, をいれておく. 将来的にはさらに必要なファイルが増える予定.

kernel の作成方法: kernel の仕様 (spec) 2

kernel.json が最も重要なファイル. 下記のような key と 値の辞書.

1. argv: kernel を start するための command line の引数. {connection_file} 文字列は connection file の path に置換される.
2. display_name:
3. language:
4. interrupt_mode: “signal” の時は SIGINT を送る. “message” の時は interrupt_request メッセージを control チャンネルに送る. “signal” が既定値.
5. env:
6. metadata:

IPython の場合の kernel.json .

```
{
 "argv": ["python3", "-m", "IPython.kernel",
          "-f", "{connection_file}"],
 "display_name": "Python 3",
 "language": "python"
}
```

kernel を動かしてみる

使える kernel の一覧は

```
jupyter kernelspec list
```

(Debian では (必要なら pip で install の後)

```
~/.local/bin/jupyter-kernelspec list
```

console で スタートするには

```
jupyter console --kernel カーネルの名前
```

(Debian では (必要なら pip install jupyter-console の後)

```
~/.local/bin/jupyter-console --kernel=カーネルの名前
...
ctrl-D
```

補足 (debian): pip でシステムが変になったら, `cd ~/.local ; rm -rf bin lib share` で消すと元のまっさら状態に.

kernel の作成方法: IPython カーネルを再利用して簡単な kernel を作る

ipykernel.kernelbase.Kernel の subclass として MyKernel を作り下記の class 変数, method を実装する.

1. implementation
2. implementation_version
3. banner
4. language_info
5. do_execute(code,silent,store_history=True,user_expressions=None, allow_stdin=False) は code を実行して結果を辞書形式で戻す. 出力を表示するには send_response() method を用いて jupyter に表示させる. message 形式については https://jupyter-client.readthedocs.io/en/stable/messaging.html 参照 (error message などもこちら, 図あり). silent=True の時は出力を表示しない.

Debian で試すには?

```
pip install echo_kernel
cd ~/.local/lib/phtyon2.7/site-packages/echo_kernel
python install.py    # 以上でインストール完了.
                     # jupyter-notebook にメニューが出る.
```

echo_kernel の kernel.py

入力をそのまま戻す kernel.

```
from ipykernel.kernelbase import Kernel

class EchoKernel(Kernel):
    implementation = 'Echo'
    # 略
    def do_execute(self, code, silent, store_history=True, user_expressions=None,
                   allow_stdin=False):
        if not silent:
            stream_content = {'name': 'stdout', 'text': code}
            self.send_response(self.iopub_socket, 'stream', stream_content)

        return {'status': 'ok',
                # The base class increments the execution count
                'execution_count': self.execution_count,
                'payload': [],
                'user_expressions': {},
               }
if __name__ == '__main__':
    from ipykernel.kernelapp import IPKernelApp
    IPKernelApp.launch_instance(kernel_class=EchoKernel)
```

練習: 'text':code を 'text':code.upper() とすると大文字に直して戻す.

歴史

1. OpenXM `http://www.openxm.org`, 1998–, 野呂, 田村, 小原.
2. `ox_texmacs`, 2004 (texmacs by Joris Van der Hoeven). `http://c1.math.kobe-u.ac.jp/cgi-bin/cvsweb.cgi/OpenXM/src/kxx/ox_texmacs.c`
3. `cfep/asir` for MacOS X. 2006. `http://c1.math.kobe-u.ac.jp/cgi-bin/cvsweb.cgi/OpenXM/src/cfep/MyOutputWindowController.h` 以後, 講義でずーっと利用. PC 必携化で Mac が講義で使えなくなるかも, が一つの動機.

ちがった方向では

1. cgi-asir, 2013. `http://c1.math.kobe-u.ac.jp/cgi-bin/cvsweb.cgi/OpenXM/src/asir-port/cgi/` デモ: `http://www.math.kobe-u.ac.jp/HOME/taka/2018/data-b`
2. emacs asir mode (小原). `http://c1.math.kobe-u.ac.jp/cgi-bin/cvsweb.cgi/OpenXM_contrib2/windows/post-msg-asirgui/`. 2013. 自分が一番良く使う環境.

jupyter asir kernel を試す

Install (debian): OpenXM の snapshot(`http://air.s.kanazawa-u.ac.jp/ohara/software.html`) を download, 展開.

```
cd OpenXM/src
make configure ; make install
cd jupyter
make install-debian-for-debug
```

Mac, anaconda2 では, 最後で,

`make install-mac-anaconda2-for-debug`

以下は, 開発者用の覚書.

jupyter asir kernel の設計と作成メモ 1

1. octave kernel を再利用. octave のふりをする, OpenXM で簡単にできた `ox_texmacs.c`.
2. `base_prompt()` で Prompt を変更するふり. IPython カーネルは pexpect を使ってる.
3. `eval` は `version()` を呼ぶ.
4. code は `do_execute_direct` で実行される.
5. make_figures で return None とすれば'NoneType' object has no attribute '__getitem__' エラーが例外でおきるのを抑制.
6. asir_kernel/ にも __init__.py 必要.
7. `ctrl("debug_window",0);  jupyter-init.rr`
8. 改行で prompt を戻すが buffering するのみ. ';;' で計算開始とした.

jupyter asir kernel の設計と作成メモ 2

1. ox_texmacs.c で DEBUG2 を def して /tmp/debug-texmacs.txt.
2. kernel.py では

```
        stream_handler = None if silent else self.stream_handler
        if self.logger:
            self.logger.setLevel(logging.DEBUG) <-- これを加えた.
            self.logger.debug('Asir eval:')
            self.logger.debug(code)

        f = open('tmptmp.txt','a');f.write(str(code));f.close()
        self._octave_engine.logger.debug(str(code)) <-- これを加えた.
        val = ProcessMetaKernel.do_execute_direct(self, code, silent=silent)
```

3. mathlibre-2019 iso boot の仮想環境. snapshot を活用.
4. ox_texmacs のテスト

```
openxm ox_texmacs --view jupyter
This is Risa/Asir, full GMP Version 20190318 (Kobe Distribution).
....

PEXPECT_PROMPT>1+2;
PEXPECT_PROMPT>2+3;;
5

PEXPECT_PROMPT> ctrl-D
```

jupyter asir kernel の設計と作成メモ 3

1. pexpect module

```
import pexpect
import sys
mac = pexpect.spawn('asir')
mac.logfile = sys.stdout
mac.expect(r'\[[0-9]*\] ');
print('before [0] ',mac.before)
print('after [0] ',mac.after)
print('buffer [0] ',mac.buffer)
mac.sendline('2^100;')
print('before [1] ',mac.before)
print('after [1] ',mac.after)
print('buffer [1] ',mac.buffer)
```

2. kernel.json

```
{
    "argv": ["python",
             "-m", "asir_kernel",
             "-f", "{connection_file}"],
    "display_name": "Asir",
    "mimetype": "text/x-octave",
    "language": "asir",
    "name": "asir"
}
```

jupyter asir kernel の設計と作成メモ 4

1. `export OX_XTERM_GEOMETRY=80x20+0+0`
としておくと util/ox_pathfinder が -icon option を加えない. Mac の dock のバグ対策 (xterm -icon を元に戻せない).
[(getServerEnv) (bin/ox_sm1)] extension
2. `http://pygments.org` (get_pygments_lexer) で入力の構文を検査. asir_kernel/kernel.py で

```
def language_info(self):
    return {'mimetype': 'text/x-octave',
            'name': 'c',
```

name のところに言語名を与える. C の場合第一 cell のみ構文エディタとなる.
3. jupyter から呼ぶ shell script は `#!/bin/bash` が必要. さもなければ Exec format error が出る.

Todo

1. BUG: Mac で debug mode へ入る件.
2. BUG: 孫の ox_plot が crash する件.
3. `status:error` を戻す. エラー行へジャンプ.
4. 2 番目以降の cell でも pygments/C 言語 を有効に.
5. 画像を jupyter notebook へはりつける.
6. pip で asir kernel をインストールできるように commit.
7. Windows 対応 (計算数学 I の実習で使えるようにする).