Notes on the Code Quality Culture on Jupyter (Notebooks)

21. WSRE, Bad-Honnef, 06.-08. Mai 2019

Daniel Speicher (dsp@bit.uni-bonn.de), Tiansi Dong, Olaf Cremers, Christian Bauckhage, Armin B. Cremers

Bonn-Aachen International Center for Information Technology, Universität Bonn
Outline

• Jupyter Notebooks are exciting
• ... and challenge everything we know about Quality

• Communicative Code
• Patterns = Solution to conflicting forces in a context
• Know the Context => Reason about solutions

• Further Reverse Engineering challenges
Observational basis for this talk


• Own notebooks at: [https://p3ml.github.io/](https://p3ml.github.io/)
  • Elaborating numerical recipes
  • Prototypical implementations for programming lab
• Students notebooks and our thorough review
• Notebooks of a course on Deep Learning (Coursera)
Jupyter Notebooks: Wow!

- A "Jupyter Notebook is [a] web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text." (https://jupyter.org/)

- Consists of text and code cells.

- The content of code cells is sent on demand to a Python session, executed and the output inserted below the cell.
Jupyter Notebooks: Wow!

**Frank-Wolfe for Minimum Enclosing Balls**

Accompanying notebook for two lecture notes on Machine Learning:

- Christian Bauckhage and Tiansi Dong: "Minimum Enclosing Balls", B-IT, University of Bonn, February 2019 (Download from ResearchGate)
- Abstract of the paper: The minimum enclosing ball problem is another example of a problem that can be cast as a constrained convex optimization problem. It thus provides us with another opportunity to work with the Karush-Kuhn-Tucker (KKT) conditions and to derive a Lagrangian dual which, as we shall see later, allows for a simple solution for the problem.
- Christian Bauckhage and Tiansi Dong: "Frank-Wolfe for Minimum Enclosing Balls", B-IT, University of Bonn, February 2019 (Download from ResearchGate)
- Abstract of the paper: In this note, we show how the Frank-Wolfe algorithm applies to the problem of computing the minimum enclosing ball of a set of data.

```python
In [2]:
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt
```
Jupyter Notebooks: Wow!

**Lagrangian Dual of the Minimum Enclosing Ball Problem**

For efficiency we calculate two constants upfront:

\[ X^T X \] and \( z = (x_j^T x_j)_{j=1,\ldots,n} \)

```python
def init_ingredients(X):
    XtX = X.T.dot(X)
    return XtX, np.diag(XtX)
```

The Lagrangian Dual of the Minimum Enclosing Ball Problem and its negative gradient:

\[
D(\mu) = \mu^T z - \mu^T X^T X \mu
\]

\[
\nabla (\nabla D(\mu)) = -\nabla D(\mu) = z - 2X^T X \mu
\]

```python
def lagrangian_dual(XtX, Z, mu):
    return mu.T.dot(Z) - mu.T.dot(XtX.dot(mu))

def neg_lagrangian_dual_gradient(XtX, Z, w):
    return -Z + 2 * XtX.dot(w)
```
Visualization of the Duality (3 Points only)

\[
X = \begin{bmatrix} 0.2, & 0.4, & 1.2, \\ 0.2, & 0.9, & 0.1 \end{bmatrix}
\]

\[\text{XtX}, \ Z = \text{init_ingredients}(X)\]

\[\text{mus} = \begin{bmatrix} 0.98, & 0.01, & 0.01, \\ 0.5, & 0.5, & 0, \end{bmatrix}, \ \begin{bmatrix} 1/3, & 1/3, & 1/3 \end{bmatrix}, \ \begin{bmatrix} 0.186, & 0.367, & 0.447 \end{bmatrix}\]

\[\text{Ls} = \text{values_over_simplex}(\text{partial(lagrangian_dual, XtX, Z)})\]

\[\text{axs} = \text{init_plots_3D}(\text{len(mus)}, \ \text{azim}=125, \ \text{elev}=15)\]

\[\text{for a, mu in zip(axs, mus):}\]
\[\hspace{1em} \text{plot_flat_simplex}(a)\]
\[\hspace{1em} \text{plot_over_simplex}(a, \ Ls)\]
\[\hspace{1em} \text{plot_point_over_simplex}(a, \ [\mu[0], \ \mu[1], \ \text{lagrangian_dual}(XtX, Z, \ mu)])\]
\[\text{done_plots}()\]

\[\text{axs} = \text{init_plots}(\text{len(mus)}, \ \text{lims}=[0, 1.5, 0, 1.1])\]

\[\text{for a, mu in zip(axs, mus):}\]
\[\hspace{1em} a.set_title('S\text{\mu}_{S} = ({}).format(' ', \text{\join}([{{:.2}}}.format(m) \ \text{for m in mu))))\]
\[\hspace{1em} \text{plot_points}(a, X)\]
\[\hspace{1em} \text{plot_circle}(a, \text{center}(X, \ mu), \text{radius}(XtX, Z, \ mu))\]
\[\text{done_plots}()\]

\[\mu = (0.98, 0.01, 0.01)\]
\[\mu = (0.5, 0.5, 0.0)\]
\[\mu = (0.33, 0.33, 0.33)\]
\[\mu = (0.19, 0.37, 0.45)\]

The search for a minimum enclosing ball around three points is dual to the search for weights \(\mu = (\mu_1, \mu_2, \mu_3)\) that maximize \(D(\mu) = \mu^T z - \mu^T X^T X \mu\).
Jupyter Notebooks: Wow!

```
one = 1
two = 2

one + one
2

one + one == two
True
```

Inspired by Joel Grus: I Don’t Like Notebooks, JupyterCon 2018
Jupyter Notebooks: Arrrrrgh!

```
one = 1
two = 2
one + one
2
one + one == two
True
```

```
one = 1
two = 2
one + one
2
one + one == two
False
```

Inspired by Joel Grus: I Don’t Like Notebooks, JupyterCon 2018
Jupyter Notebooks: Arrrrrgh!

Inspired by Joel Grus: I Don’t Like Notebooks, JupyterCon 2018
Jupyter Notebooks: Arrrrgh!

Just one of many ways how execution order might spoil results.
Jupyter Notebooks: Arrrrrgh!

```python
jupiter = 'Jupiter is '
emphasis = ', '.join(7 * ['so']); jupiter += emphasis
jupiter += ' wonderful!
print(jupiter)
```

```
Jupiter is so, so, so, so, so, so, so, so wonderful!
```
Jupyter Notebooks: Arrrrgh!

```
jupyter = 'Jupyter is '
emphasis = ', '.join(7 * ['so']); jupiter += emphasis
jupyter += ' wonderful!'
print(jupyter)
```
Jupyter Notebooks: Arrrrgh!

```python
jupyter = 'Jupyter is '
emphasis = ', '.join(7 * ['so']); jupyter += emphasis
jupyter += ' wonderful!
print(jupyter)

Jupyter is wonderful!
```
Jupyter Notebooks: Arrrrgh!

```python
jupyter = 'Jupyter is ';
emphasis = ','.join(7 * ['so']);
jupyter += emphasis;
jupyter += ' wonderful!'
print(jupyter)

Jupyter is wonderful!
```

```python
print(jupyter)

Jupiter is so, so, so, so, so, so, so wonderful!so, so, so, so, so, so, so, so
```
We are on another Planet

- Manual Execution Order Matters
- Partial renaming “refactoring”
  => Old variable with old state still in the process
  => Silent errors or difficult to find errors
- Developer needs to maintain a mental model of the state of the calculation.

- Quirky: Global variables, top level statements, few functions (only in 37%), less objects (12%)
“[O]ur intellectual powers are rather geared to master static relations and [...] our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do [...] our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.”

Edsger W. Dijkstra, Letters to the Editor: Go To Statement Considered Harmful, 1968
Communicative Code
Quotes on Communicative Code

“A good code should read like a story, not like a puzzle.”

Venkat Subramaniam, 2018
Late imports

Accompanying notebook for the recipe:

- Abstract of the paper: In this note, we study least squares optimization for parameter estimation. By means of the basic example of a linear regression task, we explore different formulations of the ordinary least squares problem, show how to solve it using NumPy or SciPy, and provide suggestions for practical applications.

In this notebook you find now in addition to the technical report as lsq_solution_y0(x, y) a direct implementation of the formula $(X^T X)^{-1} X^T y$ based on numpy matrices reading $(X^T * X).I * X.T * y$. It comes with a slight performance penalty, especially for small matrices: 200% for n = 100, but only 3% for n = 100000. x and y in this case have to be instantiated with np.matrix(...) not np.array(...).

As an experiment we write $y$ (Unicode 0177) instead of yhat to be closer to $\hat{y}$. This is not yet a recommendation. Try for yourself. There are different strongly diverging mathematical formula competes with the diffic

```
import numpy as np
import numpy.linalg as la
import numpy.random as rnd
import matplotlib.pyplot as plt
```
As an experiment we write \( \hat{y} \) (Unicode \( \text{0177} \)) instead of \( \hat{\text{y}} \). This is not yet a recommendation. Try for yourself. There are different strongly diverging options on this topic on the net. The benefit of close resemblance to the mathematical formula competes with the difficulty of typing the character.

Example data

```python
In [3]:

def create_data(n, xmin=-2, xmax=12, a=1.1, b=2.0):
    x = np.random.randn(n) * (xmax - xmin) + xmin
    
plt.show()
```

... much further down ...

```
In [12]:

import timeit
```

```
for v in [0, 1, 2, 3]:
    print('%s %s %s' % (v, v + v))
```
Late imports

What does the coder want to tell us?

```
Performance comparison

In [12]: import timeit

print(9*' ' + 'n', end='')
for v in [0, 1, 2, 3]:
    print(8*' ' + 'V{}'.format(v), end='')
print()

for n in [100, 1_000, 10_000, 100_000]:
    print('{{:10}}'.format(n), end='')
    for v in [0, 1, 2, 3):
        if v == 0:
            setup = 'x, y = create_data(n); X = np.matrix(np.vander(x, 2)); y = np.matrix(y).T'
        else:
            setup = 'x, y = create_data(n); X = np.vander(x, 2)'

    t = timeit.timeit(
        stmt = 'lsq_solution_V{}(X, y).format(v),
        setup = setup,
        number = 100,
        globals = globals()
    )

    print('{{:10.3f}}'.format(t), end='')
print()
```

<table>
<thead>
<tr>
<th>n</th>
<th>v0</th>
<th>v1</th>
<th>v2</th>
<th>v3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.007</td>
<td>0.003</td>
<td>0.010</td>
<td>0.008</td>
</tr>
<tr>
<td>1000</td>
<td>0.004</td>
<td>0.002</td>
<td>0.009</td>
<td>0.008</td>
</tr>
</tbody>
</table>
Late imports

**Goals:** Separate concerns – Share together – Know dependencies early

**Suggestion:** “This section covers a separate concern that I still want to share together with the rest of the notebook.”

What does the coder want to tell us?

```
In [12]: import timeit
print(9*' ' + 'n', end=''
for v in [0, 1, 2, 3]:
    print(8*' ' + 'V{v}'.format(v), end='
print()
for n in [100, 1_000, 10_000, 100_000):
    print(':10'.format(n), end=''
    for v in [0, 1, 2, 3]:
        if v == 0:
            setup = 'x, y = create_random(n, T'
        else:
            setup = 'x, y = create_random(n, T'
t = timeit.timeit(
    stmt = 'lsq_solution(data, setup=
    setup = setup,
    number = 100,
    globals = globals()
)
    print(':10.3f'.format()
print()
```
“To communicate effectively, the code must be based on the same language used to write the requirements - the same language that the developers speak with each other and with domain experts.”

Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, 2003
Universal Language:
Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^T X)^{-1} X^T y \]

• Implementation:

```python
# X and y created with numpy.array(..)
w = np.dot(np.dot(la.inv(np.dot(X.T, X)), (X.T)), y)
```
Universal Language: Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^T X)^{-1} X^T y \]

• Implementation:

```python
# X and y created with numpy.array(..)
w = la.inv(X.T.dot(X)).dot(X.T).dot(y)
```
Universal Language: 
Code ~ Domain

• Statistics, Ordinary Least Squares solution:

\[ w = (X^TX)^{-1}X^Ty \]

• Implementation:

```python
# X and y created with numpy.matrix(..)
w = (X.T * X).I * X.T * y
```
Identifier Length

• Shorter identifier names take longer to comprehend (See [Hofmeister 2019] and related work)

• For longer identifiers:
  • Observation: Bugs are found faster.
  • Hypothesis: Identifier meaning easier to be found.

• In mathematical contexts there are some short identifiers that have well established meaning:

  Established short >> longer unfamiliar
Length has still its value

```python
def MacQueen(X, k):
    n = X.shape[0]
    M = np.copy(X[:k])
    N = np.ones(k)

    for j in range(k, n):
        i = np.argmin(np.sum((M - X[j])**2, axis=1))
        N[i] += 1
        M[i] += 1./N[i] * (X[j] - M[i])

    return M

def MacQueen(points, k):
    means = np.copy(points[:k])
    sizes = np.copy(points[:k])

    for point in points[k:]:
        i = np.argmin(np.sum((means - point)**2, axis=1))
        sizes[i] += 1
        means[i] += 1./sizes[i] * (point - means[i])

    return means
```
Translating mathematical variables into code is difficult

• Statistics:
  \( \hat{y} \) means “estimated \( y \)”

• Best implementation?

--- means --->

- \( y_{\text{hat}} \) \( \hat{y} \) “estimated \( y \)”
- \( \hat{y} \) \( \hat{y} \) “estimated \( y \)”
- \( y_{\text{est}} \) “estimated \( y \)”
- \( y_{\text{estimated}} \) “estimated \( y \)”
Design Patterns
Design Patterns

• Solution to conflicting forces in a context

• See e.g. Section 1.1 in [Gamma 1995]

• The context of a calculation presented as a linear narrative leads to solutions that differ substantially from solutions for other kinds of software.
Function Exemplification – Forces

• Notebooks present code and its result in a linear sequence

• Result of a function definition is a defined function and no immediate output.

• Self defined functions (let alone objects) are therefore used much less frequently in notebooks than in other software.

• Still, functions are helpful for internal reuse and to give structure to a longer calculation.
Function Exemplification – Solution

• Solution:
  • Illustrate the use of the function in the next cell.
  • (for functions without side effects, short runtime and easy to provide parameters)
Utility function guessing the schema from given JSON data

```python
def print_json_schema(json, indentation=-1):
    if isinstance(json, dict):
        print()
        indentation += 1
        for key, value in json.items():
            print(indentation * ' ', key, end=': ')
            print_json_schema(value, indentation)
        indentation -= 1
        print()
    elif isinstance(json, list):
        length = len(json)
        if length:
            print(length, end='x ')
            print_json_schema(json[0], indentation)
        else:
            print('0x ???')
    else:
        print(type(json))

print_json_schema({'countries': ['Germany', 'France'], 'politicians': [{'first': 'Angela', 'last': 'Merkel'}],

countries: 2x <class 'str'>
politicians: 3x
    first: <class 'str'>
    last: <class 'str'>
Function Exemplification – Ex. 2

Utility functions for naming files and extracting text from HTML

```python
def deputy_file_name_part(deputy):
    return '__'.join([deputy['meta']�이['username'], deputy['party'].lower().replace(' ', '-')])

def question_file_name_part(q, question):
    question_nr = 'Q{:04}'.format(q + 1) # Maximum in 12/18: 344 questions (Andrea Nahles)
    question_id = '__'.join([[question_nr, question['date']]])
    category = question['category'].lower().replace(' ', '-')
    return question_id, category

def answer_file_name_part(a, answer):
    answer_nr = 'A{:02}'.format(a + 1) # Maximum in 12/18: 2 answer for one question (often)
    return '__'.join([answer_nr, answer['date']])

print(deputy_file_name_part(deputy))

if questions:
    oldest = len(questions) - 1
    print(question_file_name_part(oldest, questions[oldest]))

    answers = questions[oldest]['answers']
    if answers:
        print(answer_file_name_part(0, answers[0]))
```

ulrich-wolfgang-kelber_spd
('Q0037_2017-07-24', 'inneres-und-justiz')
A01_2017-07-25
Updated Progress Line – Forces

• When executing the code while exploring own approaches or reproducing results of others, it is essential to get feedback about the progress of long running computations.

• Once the calculation is done, a larger part of the progress information in the notebook becomes uninteresting and distracting.
Updated Progress Line – Forces

Latent Dirichlet Allocation

We instantiate the LDA algorithm passing the configuration parameters. As the given text based does cover quite a lot topics (in the intuitive sense), we did indeed only get reasonable results after looking for at least 100 topics (in the sense of LDA). The results with batch learning were much better than with online learning, but we did not explore whether online learning could be rescued by parameter tuning. We search for the probabilities of the words within topics and the shares of the topics within the documents at the same time. "fit" without transform would do the same, but throw away the topic within document distribution. "transform" after a previous "fit" would find the topic shares without adapting the word in topic distribution. The latter could even be used for previously unseen documents.

```
In [7]:
    lda_start_time = time.perf_counter()

    lda_algorithm = LatentDirichletAllocation(n_components = n_topics, learning_method='batch', max_iter = max_iter,
                                             n_jobs=n_jobs, evaluate_every=evaluate_every, verbose=verbosity_level)

    topic_model = lda_algorithm.fit_transform(word_counts)

    lda_end_time = time.perf_counter()
    print('Latent Dirichlet Allocation took {:.2f}s.'.format(lda_end_time - lda_start_time))

iteration: 1 of max_iter: 200
iteration: 2 of max_iter: 200
iteration: 3 of max_iter: 200, perplexity: 6834.2921
iteration: 4 of max_iter: 200
```
We instantiate the `LDA algorithm` passing the configuration parameters. As the given text-based does cover quite a lot topics (in the intuitive sense), we did only get reasonable results after looking for at least 100 topics (in the sense of LDA). The results with batch learning were much better than with online learning, but we did not explore whether online learning could be rescued by parameter tuning. We search for the probabilities of the words in topics and the share of topics within the documents at the same time. "Fit" without transform would do the same, but throw away the topic within-document distribution. "Transform" a previous "fit" would find the topic shares without adapting the word in topic distribution. The latter would could even be used for previously unseen documents.

```
In [7]:

    lda_start_time = time.perf_counter()
    lda_algorithm = LatentDirichletAllocation(n_components=n_topics, learning_method='batch', max_iter=max_iter, n_jobs=n_jobs, evaluate_every=evaluate_every, verbose=verbosity_level)
    topic_model = lda_algorithm.fit_transform(word_counts)
    lda_end_time = time.perf_counter()
    print('Latent Dirichlet Allocation took {:.2f}s' . format(lda_end_time - lda_start_time))

    iteration: 1 of max_iter: 200
    iteration: 2 of max_iter: 200
    iteration: 3 of max_iter: 200, perplexity: 6834.2921
    iteration: 4 of max_iter: 200
    iteration: 5 of max_iter: 200
    iteration: 6 of max_iter: 200, perplexity: 4896.6652
    iteration: 7 of max_iter: 200
    iteration: 8 of max_iter: 200
    iteration: 9 of max_iter: 200, perplexity: 4277.6493
    iteration: 10 of max_iter: 200
    iteration: 11 of max_iter: 200
    iteration: 12 of max_iter: 200, perplexity: 3989.7329
    iteration: 13 of max_iter: 200
    iteration: 14 of max_iter: 200
    iteration: 15 of max_iter: 200, perplexity: 3831.3961
    iteration: 16 of max_iter: 200
    iteration: 17 of max_iter: 200
    iteration: 18 of max_iter: 200
    iteration: 19 of max_iter: 200
    iteration: 20 of max_iter: 200, perplexity: 3454.9064
```
Updated Progress Line

Latent Dirichlet Allocation

We instantiate the LDA algorithm passing the configuration parameters only get reasonable results after looking for at least 100 topics but we did not explore whether online learning could be rescued. We explore whether online learning could be rescued.

The topics within the documents at the same time. "fit" without training a previous "fit" would find the topic shares without adapting the weight.

```python
In [7]:
lda_start_time = time.perf_counter()
lda_algorithm = LatentDirichletAllocation(n_components=n_components, n_jobs=n_jobs)
topic_model = lda_algorithm.fit_transform(word_counts)
lda_end_time = time.perf_counter()
print('Latent Dirichlet Allocation took {:.2f}s'.format(lda_end_time - lda_start_time))

iteration: 1 of max_iter: 200
iteration: 2 of max_iter: 200
iteration: 3 of max_iter: 200, perplexity: 6384.292
iteration: 4 of max_iter: 200
iteration: 5 of max_iter: 200
iteration: 6 of max_iter: 200, perplexity: 4596.665
iteration: 7 of max_iter: 200
iteration: 8 of max_iter: 200
iteration: 9 of max_iter: 200, perplexity: 4277.649
iteration: 10 of max_iter: 200
iteration: 11 of max_iter: 200
iteration: 12 of max_iter: 200, perplexity: 3989.73
iteration: 13 of max_iter: 200
iteration: 14 of max_iter: 200
iteration: 15 of max_iter: 200, perplexity: 3831.39
iteration: 16 of max_iter: 200
iteration: 17 of max_iter: 200
iteration: 18 of max_iter: 200
iteration: 19 of max_iter: 200
iteration: 20 of max_iter: 200, perplexity: 3454.90
```
Updated Progress Line – Solution

• Let the calculation repeatedly overwrite only temporarily interesting progress information in the same line.

```python
print('Progress: {} of {}.{}'.format(i, n), end='\r')
```
Updated Progress Line – Example

```python
Create or update deputy files (JSON) and question files (URL)

success = []
failure = []

for d, deputy in enumerate(deputies):
    deputy_prefix = deputy_file_name_part(deputy)
    deputy_file = corpus_dir / (deputy_prefix + '.json')

try:
    if update_only_missing_deputies and deputy_file.exists():
        continue
    questions = deputy_json['profile']['questions']
    for q, question in enumerate(reversed(questions)):
        question_infix, question_suffix = question_file_name_parts(q, question)
        success.append(url_file.name)

except Exception as exception:
    failure.append((deputy_file.name, exception))

finally:
    print('{:0d} of {:d}. {:d} files successfully created. {:d} files failed. Latest: {}'.format(d + 1, len(deputies), len(success), len(failure), deputy_file.stem), end='')

716 of 716. 73 files successfully created. 0 files failed. Latest: gyde-jensen_fdp
```
Visualization Callback – Forces

• Algorithm, implementation should not be influenced by other concerns
• We often want to show intermediate state of the algorithm.
• Same implementation should be usable with or without visualization. (If it is not visualized it should be fast.)
• It is often interesting to visualize algorithms in varying detail and with respect to different aspects.
Visualization Callback – Solution

• We pass a function as a parameter to the function that implements the algorithm.
• Default value this parameter gets an anonymous function doing nothing
• The algorithm function calls the parameter function passing all potentially interesting information in.
• Visualization functions that actually show something may have additional parameters that can be "frozen" by creating a partial function.
• ~ Strategy + Null Object as Default Strategy
Visualization Callback – Ex. 1a

```python
def MacQueen(points, k, show=lambda state, i, sizes, means: None):
    means = np.copy(points[:k])
    sizes = np.ones(k)
    for point in points[k:]:
        i = np.argmin(np.sum((means - point)**2, axis=1))
        sizes[i] += 1
        means[i] += 1./sizes[i] * (point - means[i])
    return means

def plot_some_macqueen_state(n_rows, n_cols, points, k, state, i, sizes, means):
    plot_some = partial(plot_some_macqueen_state, 1, 6, points, 3)
    means = MacQueen(points, 3, show=plot_some)
```

3 points  | 35 points  | 66 points  | 98 points  | 129 points  | 160 points
---|---|---|---|---|---
3 points | 35 points | 66 points | 98 points | 129 points | 160 points
Visualization Callback – Ex. 1a

```python
def MacQueen(points, k, show=lambda state, i, sizes, means: None):
    
    means = np.copy(points[:k])
    sizes = np.ones(k)

    for point in points[k:]
        i = np.argmin(np.sum((means - point)**2, axis=1))
        sizes[i] += 1
        means[i] += 1./sizes[i] * (point - means[i])

    return means
```

```python
def plot_some_macqueen_state(n_rows, n_cols, points, k, state, i, sizes, means):
```

```
plot_some = partial(plot_some_macqueen_state, 1, 6, points, 3)
```

```
means = MacQueen(points, 3, show=plot_some)
```

Default: show nothing. Exemplifies signature.

Calls not part of the Gestalt of the algorithm

Visualization function with additional arguments

Partial function with „frozen“ arguments

Call to the algorithm passing the visualization function

Progress visualization as “small multiples”
Visualization Callback – Ex. 1b

Plot intermediate means and final clusters

```python
def store_macqueen_state(k, cluster, means, state, j, i, N, M):

def plot_cluster_and_means(cluster, means):
    k=3; cluster = list(range(k)); means = list()
    store_state = partial(store_macqueen_state, k, cluster, means)
    M = MacQueen_func(X, k, show=store_state)

    Initialized. cluster = [0, 1, 2] , means = []
    Clustering 160 points. Cluster sizes: 47, 71, 42

    plot_cluster_and_means(cluster, means)
```
More Reverse Engineering

• For example, notebooks that have served to explore data and calculations often need thorough clean-up before they may be passed on to explain findings.
  • Unroll exploration history -> Duplication Detection -> Function Definitions
• Meaningful identifiers
• Dead code
• Data flow analysis
• Jupyter notebooks are interesting software
• ≥ 1,000,000 computational notebooks on GitHub!
• Code Quality Culture on Jupyter:
  • Code quality guidelines need to be adapted for the context of “calculations as a linear narrative”. (M2)
  • Searching for “solutions to conflicting forces in a context” is still a helpful practice. (M3)
• Software Engineering and Reverse Engineering can help to make better notebooks.
• Ours: https://p3ml.github.io/ (far from perfect)
Vielen Dank
für Ihre Aufmerksamkeit