Team Anyalgorithms - Major Feature Implementation

Deliverable 4

Ahmed Sirajuddin

Jon An

Ricky Chen

Robert Peng

Table of Contents

User Guide

Introduction:

Task: https://github.com/matplotlib/matplotlib/issues/13919

Our team was tasked with implementing a new feature in matplotlib to allow users to configure the grid styles in matplotlib independently from each other (major/minor). This allowed for more consistent use throughout the app and was a nice quality of life feature users were requesting.

New:

The following `rcParams` keys can be used to customize major gridlines:

- `rcParams['grid.major.color']`
- `rcParams['grid.major.linestyle']`
- `rcParams['grid.major.linewidth']`
- `rcParams['grid.major.alpha']`

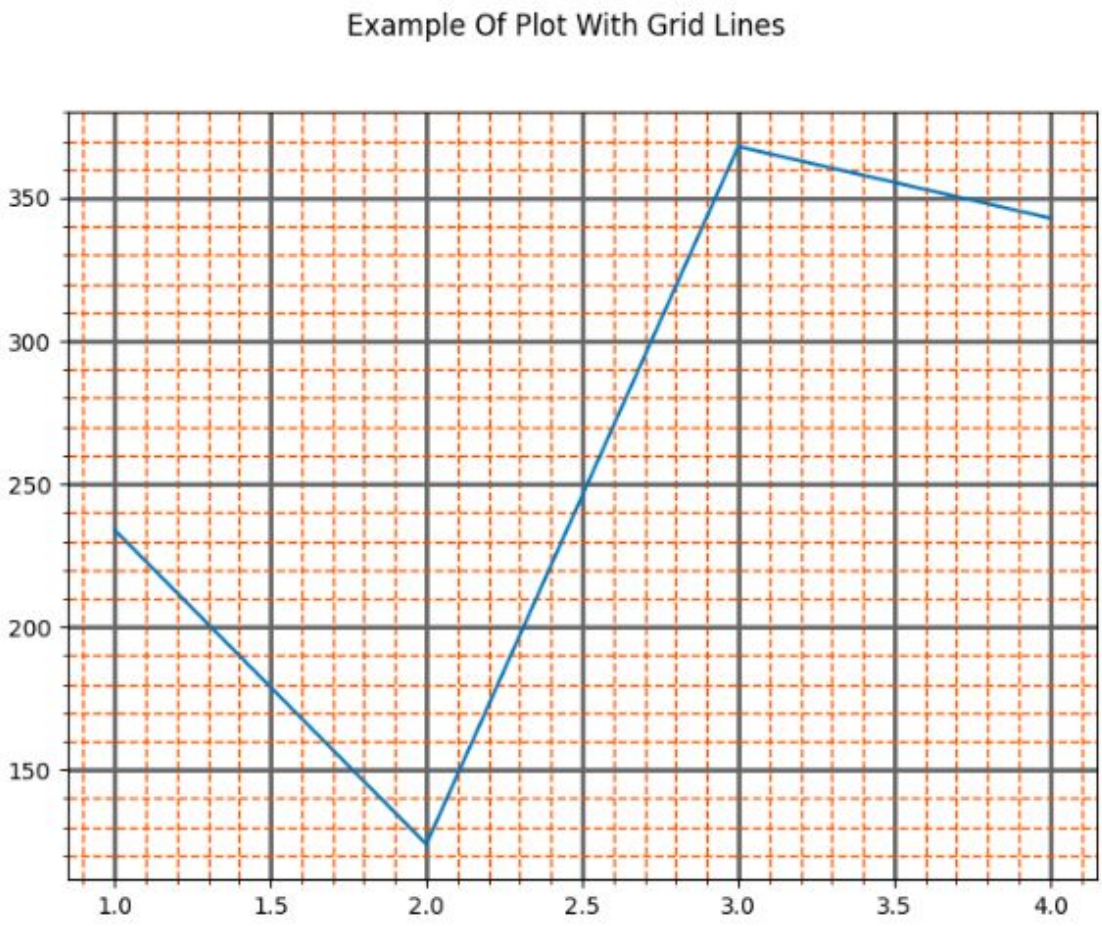
And customizing minor gridlines can be accomplished with the following `rcParams` keys:

- `rcParams['grid.minor.color']`
- `rcParams['grid.minor.linestyle']`
- `rcParams['grid.minor.linewidth']`
- `rcParams['grid.minor.alpha']`

Guide:

In matplotlib, users can now quickly and efficiently customize major and minor grid lines by using `rcParams` and keywords. For example, set a value to `rcParams['grid.major.color']` to change the major grid line color, and set a value to `rcParams['grid.minor.color']` to change the minor grid color. The full customization options for making changes to major and minor grid lines are listed above. The following example demonstrates to create a graph and changing its minor and major grid line colors with this feature:

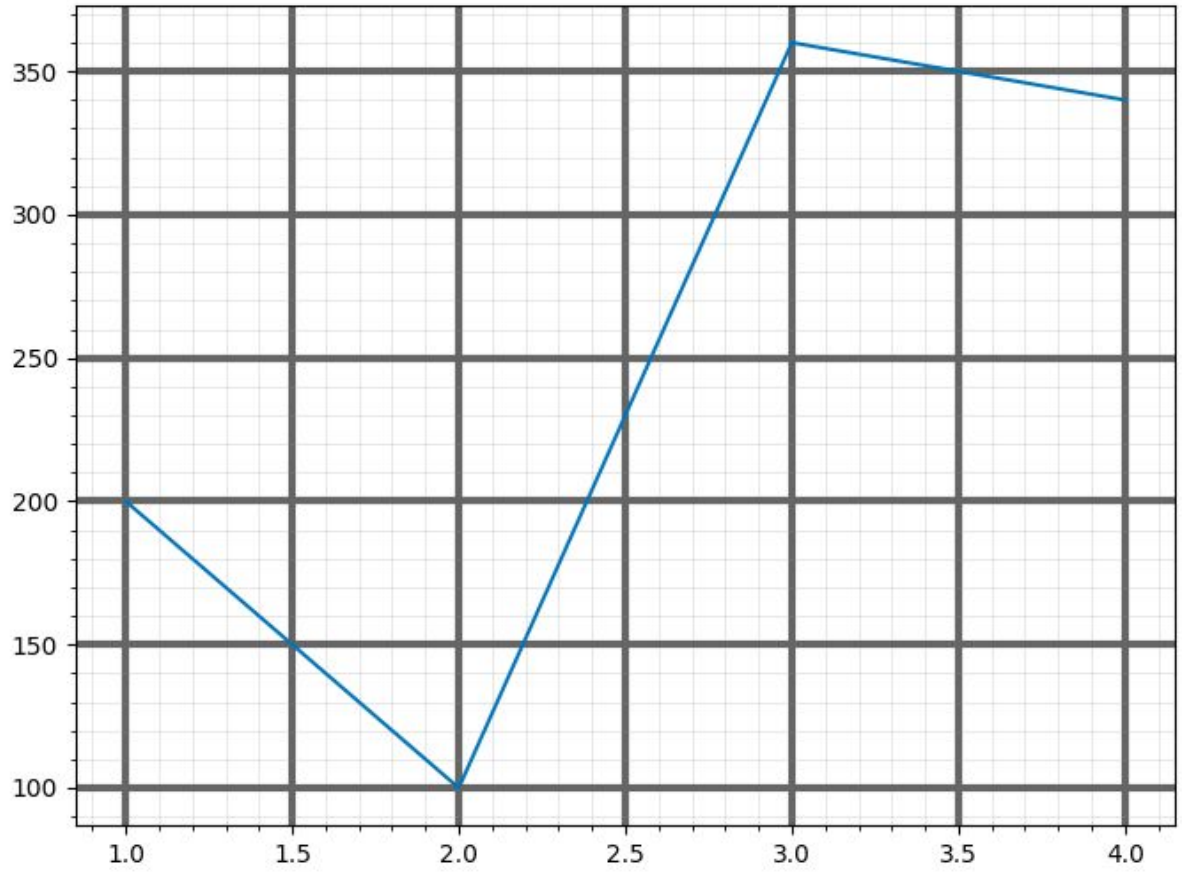
```
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['grid.minor.color'] = '#ff5d00'
rcParams['grid.minor.linestyle'] = '--'
rcParams['grid.minor.alpha'] = 1
rcParams['grid.minor.linewidth'] = 1
rcParams['grid.major.color'] = '#666666'
rcParams['grid.major.linestyle'] = '-'
rcParams['grid.major.alpha'] = 1
rcParams['grid.major.linewidth'] = 2
# The Data
x = [1, 2, 3, 4]
y = [234, 124,368, 343]
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Example Of Plot With Grid Lines')
plt.minorticks_on()
# Plot the data
ax.plot(x,y)
# Show the grid lines as dark grey lines
plt.grid(b=True, which='both')
plt.show()
```



Here, the major and minor grid line appearances can now be conveniently be changed using `rcParams` instead of making the changes programmatically through functions. The previous way of customizing grid lines still works and is unaffected. This is an example of how to programmatically customize grid line appearances through functions (the previous way):

```
import matplotlib.pyplot as plt
# The Data
x = [1, 2, 3, 4]
y = [200, 100, 360, 340]
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Minor and Major Grid Lines')
# Plot the data
ax.plot(x,y)
# Show the major grid lines with dark grey lines
```

```
plt.grid(b=True, which='major', color='#666666', linestyle='-')
# Show the minor grid lines with very faint and almost transparent grey lines
plt.minorticks_on()
plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.3)
plt.show()
```



Design of New Code

The design of the new code entailed mimicking the current way that users have to programmatically change the defaults. Currently, users change the default values of major or minor gridlines using the grid function as shown below. Understanding how Grid properties changed required an investigation into its implementation.

```
grid(b=True, which='minor', color='r', linestyle='--')
grid(b=True, which='major', color='r', linestyle='--')
```

Matplotlib uses `rcParams` for default settings when plotting graphs and validates the data together with instantiating inside of the `rcsetup.py`.

```
    'grid.color':        ['#b0b0b0', validate_color],  # grid color
    'grid.linestyle':    ['-', _validate_linestyle],  # solid
    'grid.linewidth':    [0.8, validate_float],     # in points
    'grid.alpha':        [1.0, validate_float],
```

New keys were added into this object as it was necessary to differentiate between major and minor grid lines. The default configurations were kept so that users who do not want to use this feature can still use the old one.

```
    'grid.major.color':  ['#b0b0b0', validate_color], # grid color
    'grid.minor.color':  ['#b0b0b0', validate_color], # grid color
    'grid.major.linestyle':    ['-', _validate_linestyle],  # solid
    'grid.minor.linestyle':    ['-', _validate_linestyle],  # solid
    'grid.major.linewidth':    [3.0, validate_float],     # in points
    'grid.minor.linewidth':    [0.5, validate_float],     # in points
    'grid.major.alpha':        [1.0, validate_float],
    'grid.minor.alpha':        [0.5, validate_float],
```

So together it looks like this.

```
    'grid.color':              ['#b0b0b0', validate_color],  # grid color
```

```
    'grid.linestyle':          ['-', _validate_linestyle],  # solid
    'grid.linewidth':          [0.8, validate_float],     # in points
    'grid.alpha':              [1.0, validate_float],
     'grid.major.color':           ['#b0b0b0', validate_color], # major grid
color
     'grid.minor.color':           ['#b0b0b0', validate_color], # minor grid
color
    'grid.major.linestyle':    ['-', _validate_linestyle],  # major solid
    'grid.minor.linestyle':    ['-', _validate_linestyle],  # minor solid
    'grid.major.linewidth':    [3.0, validate_float],     # major in points
    'grid.minor.linewidth':    [0.5, validate_float],     # minor in points
    'grid.major.alpha':        [1.0, validate_float],
    'grid.minor.alpha':        [0.5, validate_float],
```

When `Grid` is initiated, grid arguments are passed into the `kwargs` parameter to be re-mapped, then sent to the `set_tick_params` function. Re-mapping is used to add "`grid_`" as a prefix to each key to allow Matplotlib to perform validations later in the process.

The `set_tick_params` function is used to set the appearance of ticks, tick labels, and gridlines. Within, the configurations get re-mapped again with a call to the `self._translate_tick_kw` function. However, this does not apply to all keys as gridline parameters are omitted from this process but still validated. As well, the `set_tick_params` function receives the new mapping and applies these new parameter configurations using the `_apply_params` function before translation. These functions are located in axis.py.

So the appropriate place in which to implement new default checks would be at the beginning of the `set_tick_params` function – before the translation occurs. It checks to see if the user provided their own gridline parameters which allows the code to be backwards compatible; if the user does not provide gridline parameters, then the default values from `rcParams` will be used. So the code will check and change values based on the new value of `rcParams` (default or changed) which will allow users change `rcParams` to change default values.

The function `Grid()`:

```
def grid(self, b=None, which='major', **kwargs):
        """
        Configure the grid lines.

        Parameters
        ----------
        b : bool or None
            Whether to show the grid lines. If any *kwargs* are supplied,
            it is assumed you want the grid on and *b* will be set to True.

            If *b* is *None* and there are no *kwargs*, this toggles the
            visibility of the lines.

        which : {'major', 'minor', 'both'}
            The grid lines to apply the changes on.

        **kwargs : `.Line2D` properties
            Define the line properties of the grid, e.g.::

                grid(color='r', linestyle='-', linewidth=2)

        """
        if len(kwargs):
            if not b and b is not None:  # something false-like but not None
                cbook._warn_external('First parameter to grid() is false, '
                                     'but line properties are supplied. The '
                                     'grid will be enabled.')
            b = True
        which = which.lower()
        cbook._check_in_list(['major', 'minor', 'both'], which=which)
        gridkw = {'grid_' + item[0]: item[1] for item in kwargs.items()}

        if which in ['minor', 'both']:
            if b is None:
                self._gridOnMinor = not self._gridOnMinor
            else:
                self._gridOnMinor = b
```

```
            self.set_tick_params(which='minor', gridOn=self._gridOnMinor,
                                 **gridkw)
        if which in ['major', 'both']:
            if b is None:
                self._gridOnMajor = not self._gridOnMajor
            else:
                self._gridOnMajor = b
            self.set_tick_params(which='major', gridOn=self._gridOnMajor,
                                 **gridkw)
        self.stale = True
```

The modified function `set_tick_params()`:

```
    def set_tick_params(self, which='major', reset=False, **kw):
        """
        Set appearance parameters for ticks, ticklabels, and gridlines.

        For documentation of keyword arguments, see
        :meth:`matplotlib.axes.Axes.tick_params`.
        """
        cbook._check_in_list(['major', 'minor', 'both'], which=which)

        # Check for defaults
        if which=='major':
          if (not "grid_color" in kw):
            kw['grid_color'] = mpl.rcParams['grid.major.color']
          if (not 'grid_linestyle' in kw):
            kw['grid_linestyle'] = mpl.rcParams['grid.major.linestyle']
          if (not 'grid_linewidth' in kw):
            kw['grid_linewidth'] = mpl.rcParams['grid.major.linewidth']
          if (not 'grid_alpha' in kw):
            kw['grid_alpha'] = mpl.rcParams['grid.major.alpha']
        elif which=='minor':
          if (not "grid_color" in kw):
            kw['grid_color'] = mpl.rcParams['grid.minor.color']
          if (not 'grid_linestyle' in kw):
```

```
            kw['grid_linestyle'] = mpl.rcParams['grid.minor.linestyle']
          if (not 'grid_linewidth' in kw):
            kw['grid_linewidth'] = mpl.rcParams['grid.minor.linewidth']
          if (not 'grid_alpha' in kw):
            kw['grid_alpha'] = mpl.rcParams['grid.minor.alpha']

        kwtrans = self._translate_tick_kw(kw)

        # the kwargs are stored in self._major/minor_tick_kw so that any
        # future new ticks will automatically get them
        if reset:
            if which in ['major', 'both']:
                self._major_tick_kw.clear()
                self._major_tick_kw.update(kwtrans)
            if which in ['minor', 'both']:
                self._minor_tick_kw.clear()
                self._minor_tick_kw.update(kwtrans)
            self.reset_ticks()
        else:
            if which in ['major', 'both']:
                self._major_tick_kw.update(kwtrans)
                for tick in self.majorTicks:
                    tick._apply_params(**kwtrans)
            if which in ['minor', 'both']:
                self._minor_tick_kw.update(kwtrans)
                for tick in self.minorTicks:
                    tick._apply_params(**kwtrans)
            # special-case label color to also apply to the offset text
            if 'labelcolor' in kwtrans:
                self.offsetText.set_color(kwtrans['labelcolor'])

        self.stale = True
```

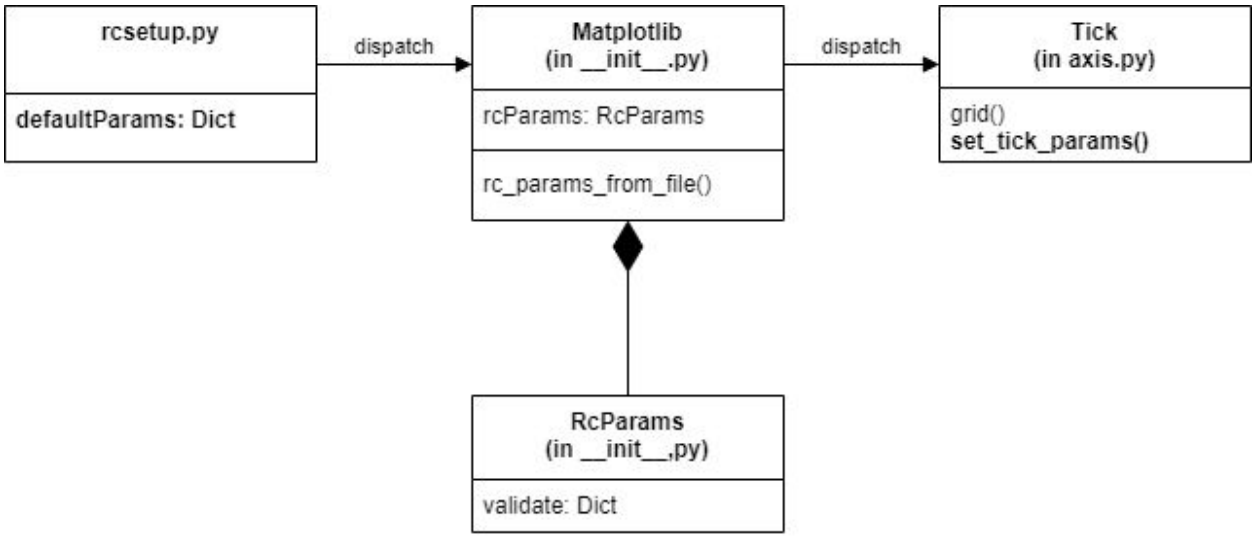
The function `_translate_tick_kw()`:

```
def _translate_tick_kw(kw):
        # The following lists may be moved to a more accessible location.
```

```
        kwkeys = ['size', 'width', 'color', 'tickdir', 'pad',
                  'labelsize', 'labelcolor', 'zorder', 'gridOn',
                  'tick1On', 'tick2On', 'label1On', 'label2On',
                  'length', 'direction', 'left', 'bottom', 'right', 'top',
                  'labelleft', 'labelbottom', 'labelright', 'labeltop',
                  'labelrotation'] + _gridline_param_names
        kwtrans = {}
        if 'length' in kw:
            kwtrans['size'] = kw.pop('length')
        if 'direction' in kw:
            kwtrans['tickdir'] = kw.pop('direction')
        if 'rotation' in kw:
            kwtrans['labelrotation'] = kw.pop('rotation')
        if 'left' in kw:
            kwtrans['tick1On'] = kw.pop('left')
        if 'bottom' in kw:
            kwtrans['tick1On'] = kw.pop('bottom')
        if 'right' in kw:
            kwtrans['tick2On'] = kw.pop('right')
        if 'top' in kw:
            kwtrans['tick2On'] = kw.pop('top')
        if 'labelleft' in kw:
            kwtrans['label1On'] = kw.pop('labelleft')
        if 'labelbottom' in kw:
            kwtrans['label1On'] = kw.pop('labelbottom')
        if 'labelright' in kw:
            kwtrans['label2On'] = kw.pop('labelright')
        if 'labeltop' in kw:
            kwtrans['label2On'] = kw.pop('labeltop')
        if 'colors' in kw:
            c = kw.pop('colors')
            kwtrans['color'] = c
            kwtrans['labelcolor'] = c
        # Maybe move the checking up to the caller of this method.
        for key in kw:
            if key not in kwkeys:
                raise ValueError(
```

```
                    "keyword %s is not recognized; valid keywords are %s"
                    % (key, kwkeys))
            kwtrans.update(kw)
        return kwtrans
```

Below is a UML Diagram representing the parts of Matplotlib that are involved with the current feature with the bolded text indicating where changes were made to implement it. Compared to expectations from the previous deliverable, the `defaultParams` dictionary in `rcsetup.py` was correctly identified as needing modification as that is where the major and minor gridline keys were added. However, the statement that gridlines were handled in the `__init__` function of `axis.py` proved to be incorrect as modifications were, instead, made in the `set_tick_params` function.



Acceptance Test Suite

```
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['grid.minor.color'] = '#ff5d00'
rcParams['grid.minor.linestyle'] = '--'
rcParams['grid.minor.alpha'] = 1
rcParams['grid.minor.linewidth'] = 1
rcParams['grid.major.color'] = '#666666'
rcParams['grid.major.linestyle'] = '-'
rcParams['grid.major.alpha'] = 1
rcParams['grid.major.linewidth'] = 2
# The Data
x = [1, 2, 3, 4]
y = [234, 124,368, 343]
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Example Of Plot With Grid Lines')
plt.minorticks_on()
# Plot the data
ax.plot(x,y)
# Show the grid lines as dark grey lines
plt.grid(b=True, which='both')
plt.show()
```

1. Setup Matplotlib.
2. Create a .py file, then copy the program above into the file.
3. Run the file (for example, by opening terminal and typing `python gridExample.py`)
4. Verify that the minor grid lines are dashed lines (e.g. - - - -), are of orange color, with low opacity and thin width.
5. Verify that the major grid lines are solid lines (.e.g. -----), are of dark gray color, with high opacity and a width of 2.

The reason for choosing acceptance testing -- as opposed to unit or any other kind of testing -- is because these changes are made more specifically as a feature that users can use. Therefore it makes more sense that a user uses it in a case rather than a unit test that makes sure it works. It is more of a coding preference and making unit tests to make sure it “works” is not really testing the usefulness of this feature.

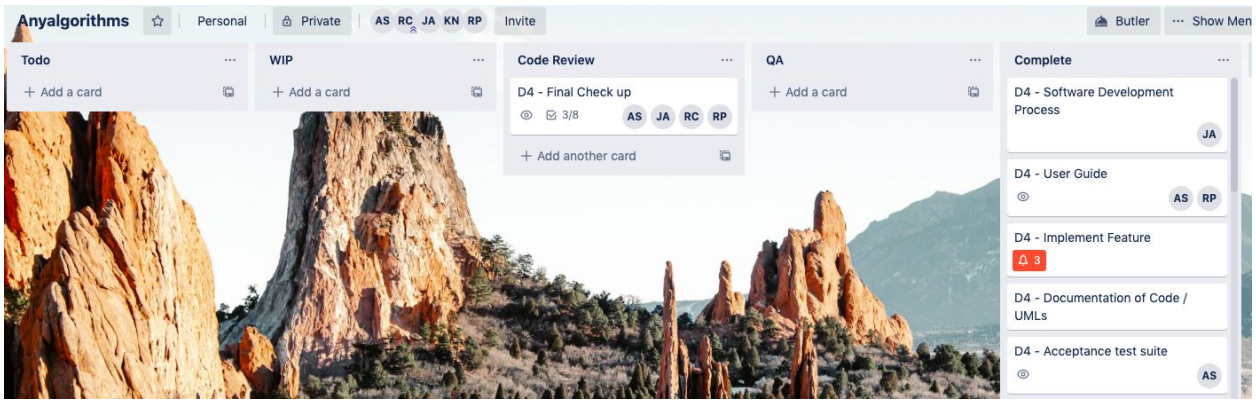
Minor modifications needed to be made to this acceptance test from the previous deliverable. Namely, the user should call the `grid` function by passing `both` as an argument, to make sure both grid lines will be visible.

Software Development Process

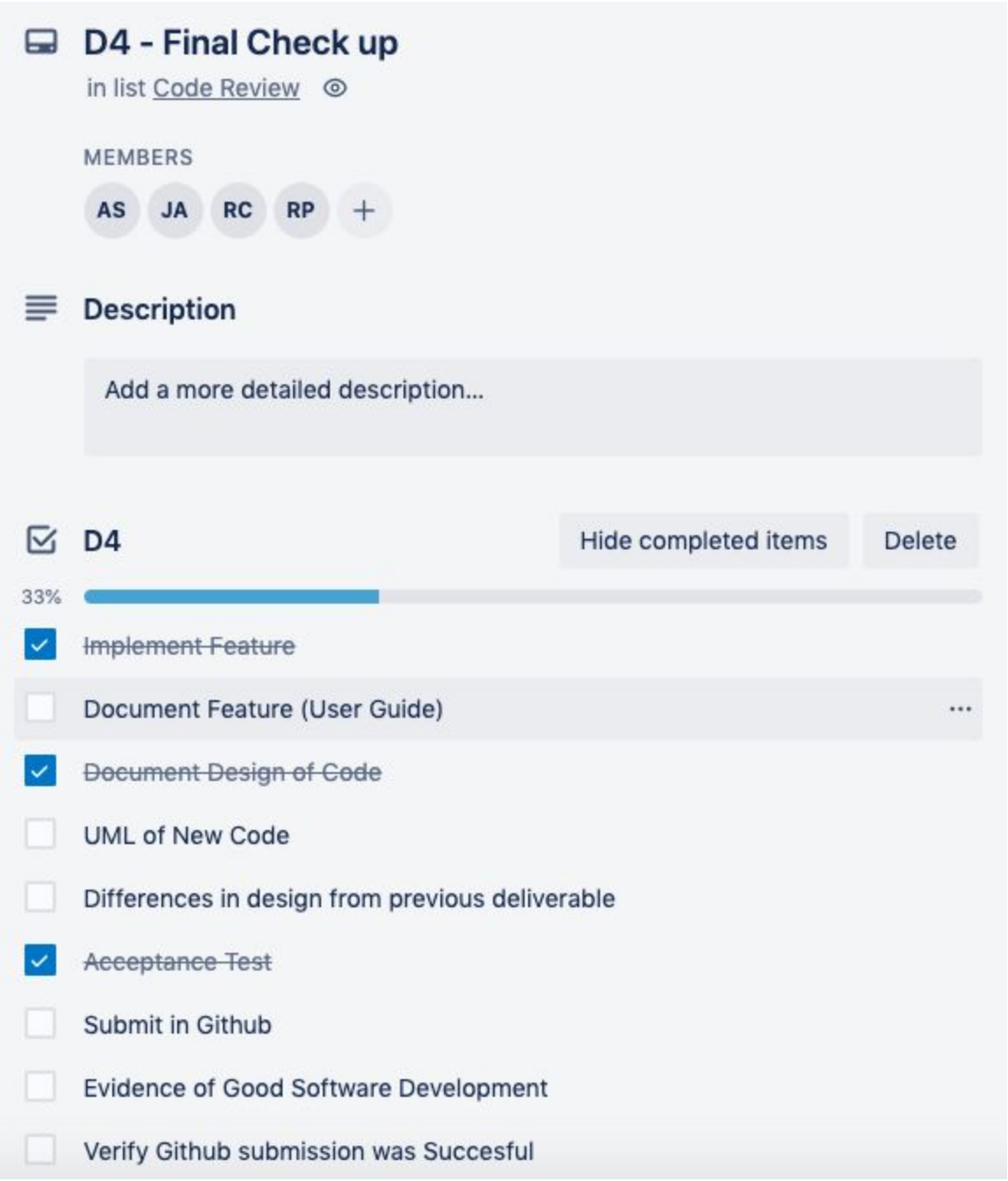
Each card on the board is associated with a deliverable 4 task and they are all initially placed in the Todo column. When one or several individuals have taken responsibility for a task, the card would be moved to the WIP column to indicate that its completion is in progress. When a task has been completed, its corresponding card gets moved to the Complete column. Specifically in the case of a card that corresponds to coding a feature, it gets moved to the Code Review column to show that the activity can take place. After that is completed, the card is moved to the QA column indicating that the implemented feature is ready for testing. For our particular feature, acceptance testing was used. We ran code and verified that the generated graph visually looked like how it should look like. For this reason, unit testing and running `pytest` wasn't necessarily needed, as our feature was more "visual" than code based. Also, as mentioned earlier, our changes are made more specifically as a feature that users can use. Therefore it makes more sense that a user uses it in a case rather than a unit test that makes sure it works. It is more of a coding preference and making unit tests to make sure it "works" is not really testing the usefulness of this feature. After that activity is completed, the card is finally considered complete and is moved to the corresponding column.

There is a small difference in the way cards were used in this deliverable compared to deliverable 2. In deliverable 2, cards were categorized under various sub-phases that were shown on the team's Google Docs that showed the dependencies of tasks. That was forgone in this deliverable as the team made use of only five cards; the small number of cards made it easy to be aware of task dependencies.

Here is a screenshot of our Kanban / Trello board near the end of the deliverable, where most of the cards are complete:



We also created a Final Checkup card that had a final checklist of everything that should be completed for this deliverable. This card was moved from Todo to the Review column near the end of the project, and made it simple for everyone to verify that the deliverable was complete. Here is a screenshot of the Final Checkup card and the checklist:



One aspect of Kanban that was not of much use was the way in which the board would reveal bottlenecks if a card limit was reached in the work-in-progress columns; because only six cards were used in this deliverable, bottlenecks did not arise.

The meetings conducted online tended to be short. They involved members of the team summarizing their progress on a given task and how they planned to go about completing it. For the most part, however, the Kanban board was used throughout the weeks and days to keep the group updated on the progress of the deliverable.

In summary, the Kanban process we followed remained largely unchanged from previous deliverables. It has continued to be a useful tool in coordinating the work done on this deliverable. It was helpful in very quickly visualizing the work that needed to be done, who was assigned to a given task, what tasks were completed, and what tasks were left to be done.

Summary of Team Meetings

Monday, 30 March (first deliverable 4 meeting)
Team members present:

- Ahmed
- Jon
- Ricky
- Robert

New items:

- Team agrees on D4 extension to 7 April.
- Team to test code for the purpose of tracing.

```
import pdb; pdb.set_trace()
```

- Team agrees to postpone further progress on D4 to finish assignments in other courses.

Saturday, 4 April
Team members present:

- Ahmed
- Jon
- Ricky

New items:

- The team is to continue its investigation of Matplotlib by continuing investigation on relevant code found in deliverable 3. The following code is to be traced:

```
import matplotlib.pyplot as plt# The Data
x = [1, 2, 3, 4]
y = [200, 100, 360, 340]# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(8, 6))
fig.suptitle('Minor and Major Grid Lines')# Plot the data
ax.plot(x,y)# Show the major grid lines with dark grey lines
plt.grid(b=True, which='major', color='#666666', linestyle='-')# Show the
minor grid lines with very faint and almost transparent grey lines
```

```
plt.minorticks_on()
plt.grid(b=True,    which='minor',    color='#999999',    linestyle='-',
alpha=0.3)plt.show()
```

Kanban board changes by the time of meeting:

- D4 - Implement Feature: Assigned to the entire team and put into the WIP column.
- D4 - Documentation of Code / UMLs: Initialized in Todo column.
- D4 - User Guide: Initialized in Todo column.
- D4 - Acceptance Test Suite: Initialized in Todo column.
- D4 - Software Development Process: Initialized in Todo column.
- D4 - Final Check up: Initialized in Todo column.

Sunday, 5 April.

Team members present:

- Ahmed
- Jon
- Ricky
- Robert

Progress report since last meeting:

- Jon attempted to change the rcParams keys. As grid.color, grid.linestyle, grid.linewidth, grid.alpha are associated with major gridlines, an attempt was made to see if simply changing them to grid.major.color, grid.major.linestyle, grid.major.linewidth, grid.major.alpha would work.
    - Resulted in major gridlines not appearing.
- Ricky investigated changing gridlines by running commands.
    - Has managed to make lines thicker by default.
- Ahmed discovered that `XAxis.grid()` sent the arguments for customizing the grid lines to `XAxis.set_tick_params` which linked to `self._major_tick_kw`

New items:

- Team to continue investigation of code.
    - One specific item is to continue on with Ahmed's findings to find equivalent functionality for minor gridlines.

Kanban board changes by the time of meeting:

No change from last meeting.

Monday, 6 April

Team members present:

- Jon
- Ricky

Progress report since last meeting

- Ricky finalizes feature implementation.

New items:

- Ricky to write down details of the implementation.
- Ahmed and Jon to review the code.
- Ahmed to do the acceptance testing.
    - Expected to be mostly unchanged from previous deliverable.

Kanban board changes by the time of meeting:

- D4 - Implement Feature: Re-assigned to Ahmed and Jon, moved to Code-Review column.
- D4 - Documentation of Code / UMLs: Assigned to Ricky, moved to WIP column.
- D4 - Acceptance Test Suite: Assigned to Ahmed, moved to WIP column.
- D4 - Software Development Process: Assigned to Jon, moved to WIP column.

Tuesday, 7 April.

Team members present:

- Ahmed
- Jon
- Ricky
- Robert

Progress report since last meeting

- Feature implementation was successfully replicated by other team members.
- Ricky completed code documentation.

New items

- Ahmed continues to work on acceptance testing.
- Robert to work on User Guide.
- Final review of deliverable 4 submission to take place.

Kanban board changes by the time of meeting:

- D4 - Implement Feature: Moved to the QA then the Complete column.
- D4 - Documentation of Code / UMLs: Moved to the Complete column.
- D4 - User Guide: Assigned to Robert, moved to the WIP column.