problem2

September 29, 2023

1 Foundations of Reinforcement Learning

Lab 2: Dynamic Programming

1.1 Content

1. Problem statement
2. Dynamic Programming algorithm

[ ]:
```
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.2 Problem Statement [P 29, Ch 1.3, Bertsekas]

A driver is looking for inexpensive parking on the way to his destination. The parking area contains N spaces, and a garage at the end. The driver starts at space 0 and traverses the parking spaces sequentially, i.e., from space k he goes next to space $k+1$, etc. Each parking space k costs $c(k)$ and is free (unoccupied) with probability $p(k)$ independently of whether other parking spaces are free or not. If the driver reaches the last parking space and does not park there, he must park at the garage, which costs C. The driver can observe whether a parking space is free only when he reaches it, and then, if it is free, he makes a decision to park in that space or not to park and check the next space. The problem is to find the minimum expected cost parking policy.

[ ]:
```
# write any global function here if needed

#Define a function to get list of optimal expected cost-to-go from list of␣
 ↪parking cost
def get_list_optimal_expected_cost(list_c):
  #Initialize an array reflects the expexted cost-to-go
  list_expected_cost = np.ones(N) * np.Inf
  #Initialize back-tracking array for DP
  list_V = np.ones(N) * np.Inf
  #Run DP
```

```
  k = N-1
  list_V[k] = p_k * min(list_c[k], C) + (1 - p_k) * C
  for k in range(N - 2, -1, -1):
    list_V[k] = p_k * min(list_c[k], list_V[k+1]) + (1 - p_k) * list_V[k+1]
  #Update the optimal expexted cost-to-go array
  for k in range(len(list_expected_cost)):
    list_expected_cost[k] = list_V[k]
  return list_expected_cost

#Define a function to get list of optimal action from list of optimal expected␣
 ↪cost-to-go
def get_list_optimal_action(list_expected_cost):
  #Initialize a binary matrix, where 0 means don't park, 1 mean park if possible
  list_actions = np.zeros(N)
  for k in range(len(list_expected_cost)):
    if k == N-1 and list_expected_cost[k] < C:
      list_actions[k] = 1
    elif k < N-1 and list_expected_cost[k] < list_expected_cost[k+1]:
      list_actions[k] = 1
  return list_actions

#Initialize global variables
N, C, p_k = 200, 100, 0.05
```

1.3 Problem 1. Dynamic Programming (DP) algorithm

1. (30 pts) Assume $N = 200$, $C = 100$; $p(k) \equiv 0.05$ and $c(k) = N - k$ for all $k \in [0, N-1]$, please: (a) Implement the Dynamic Programming algorithm to generate the expected cost-to-go upon arriving at each space k, if we always park at the first available space; (b) Implement the Dynamic Programming algorithm to generate the optimal expected cost-to-go upon arriving at each space k; (c) Plot and compare (a) and (b). (d) Plot the optimal action to do at each space k.

2. (10 pts) You now have the freedom to change C, $p(k)$ and $c(k)$. Please come up with different parameters such that: (a) parking at the first available space is the optimal policy; (b) not parking until reach the garage is the optimal policy.

   and plot optimal expected cost-to-go upon arriving at each space k repectively.

1.3.1 1.a

[ ]:
```
#Define a function to return an array refects the parking cost
def sample_parking_costs(N):
  list_c = np.zeros(N)
  for k in range(N):
    list_c[k] = N - k
  return list_c
```

```
list_c = sample_parking_costs(N)

#Initialize an array reflects the expexted cost-to-go
list_expected_cost_1a = np.ones(N) * np.Inf
#Initialize back-tracking array for DP
list_V = np.ones(N) * np.Inf

#Run DP if always park at the 1st place
k = N-1
list_V[k] = p_k * list_c[k] + (1 - p_k) * C
for k in range(N - 2, -1, -1):
  list_V[k] = p_k * list_c[k] + (1 - p_k) * list_V[k+1]

#Update expexted cost-to-go array
for k in range(len(list_expected_cost_1a)):
  list_expected_cost_1a[k] = list_V[k]
```

1.3.2 1.b

[ ]:
```
list_expected_cost_1b = get_list_optimal_expected_cost(list_c)
```

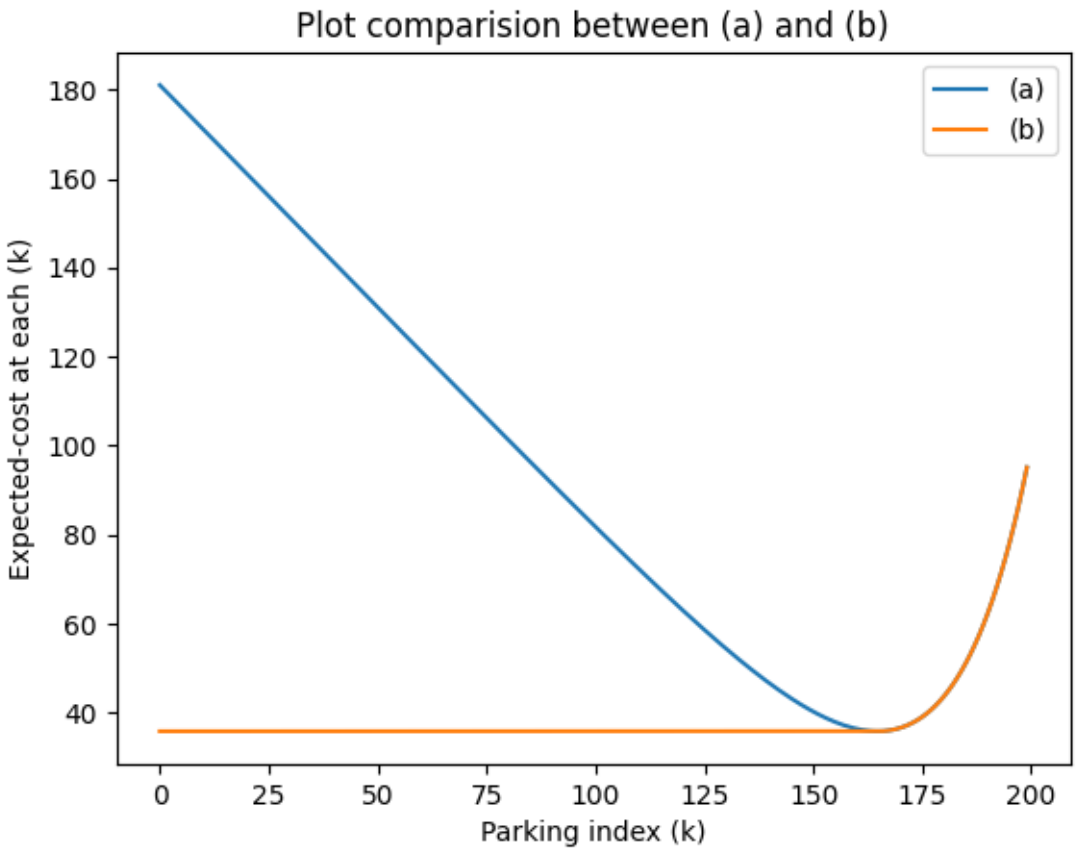
1.3.3 1.c

[ ]:
```
fig = plt.figure()
list_parking_index = np.arange(0, N)

plt.plot(list_parking_index, list_expected_cost_1a)
plt.plot(list_parking_index, list_expected_cost_1b)

plt.legend(["(a)", "(b)"])

plt.xlabel("Parking index (k)")
plt.ylabel("Expected-cost at each (k)")
plt.title("Plot comparision between (a) and (b)")
plt.show()
```

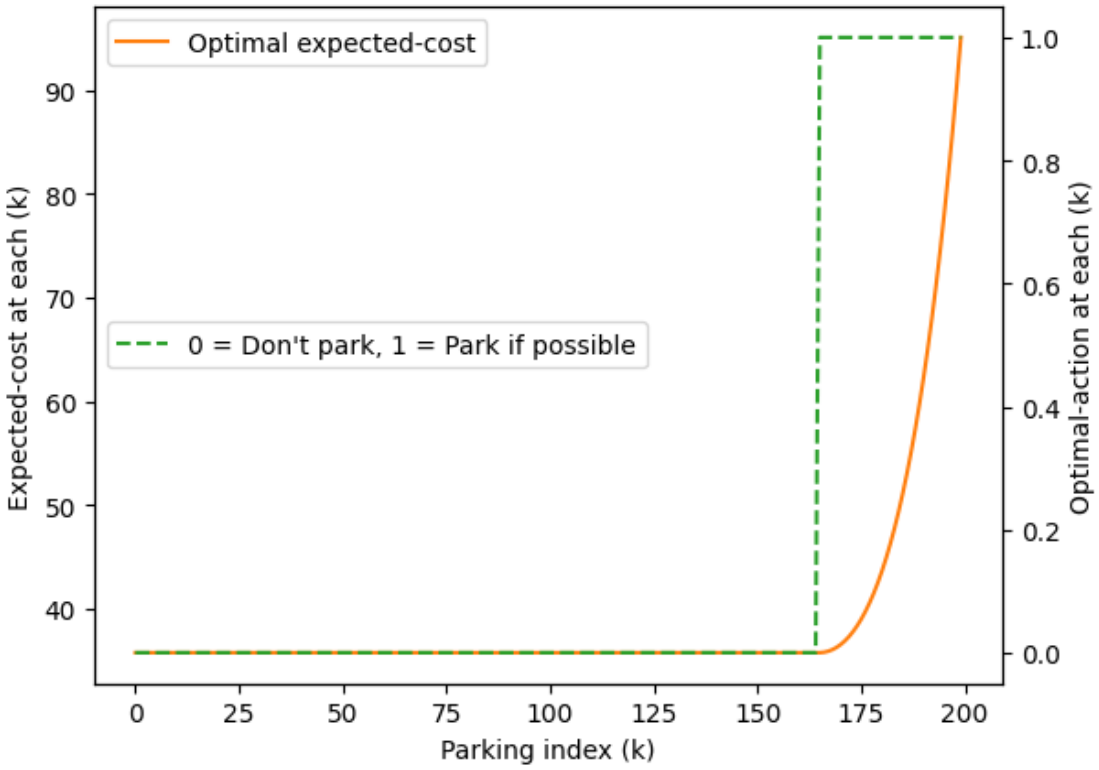


1.3.4 1.d

[ ]:
```
list_actions = get_list_optimal_action(list_expected_cost_1b)

plt.plot(list_parking_index, list_expected_cost_1b, color = "#ff7f0e")
plt.xlabel("Parking index (k)")
plt.ylabel("Expected-cost at each (k)")
plt.legend(["Optimal expected-cost"], loc = 0)

plt.twinx()
plt.plot(list_parking_index, list_actions, '--', color = "#2ca02c")
plt.ylabel("Optimal-action at each (k)")
plt.legend(["0 = Don't park, 1 = Park if possible"], loc = 6)
plt.show()
```



1.3.5 2.a

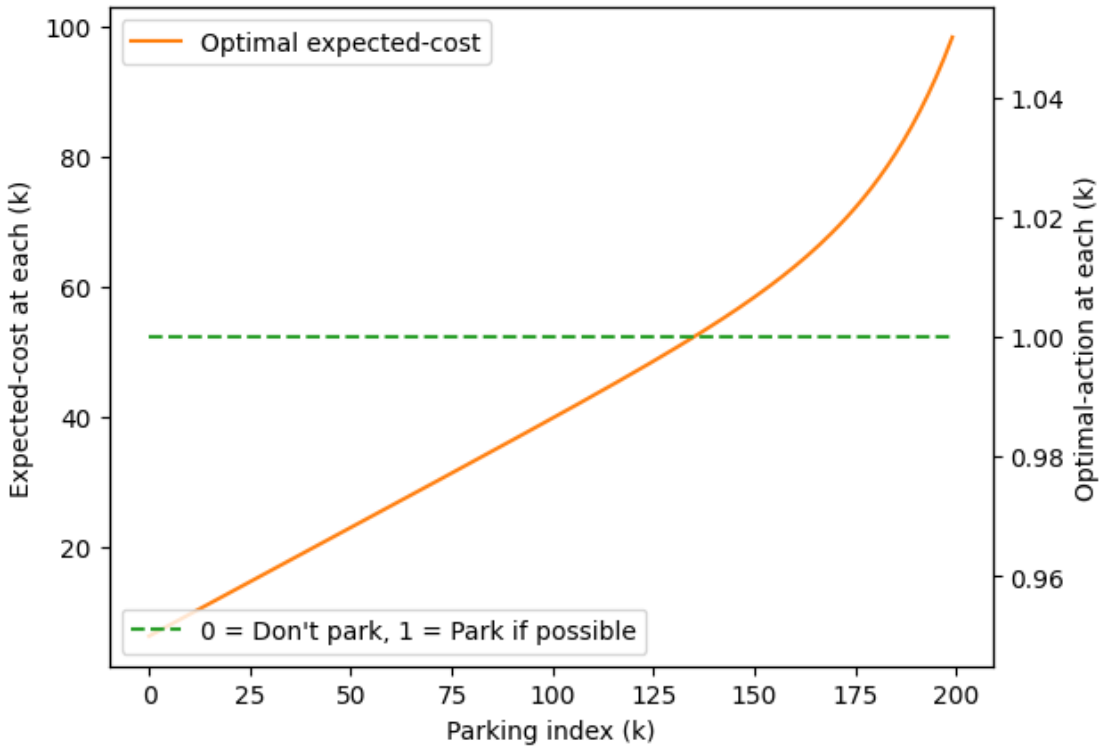
[ ]:
```
#Define a function to return an array refects the parking cost
def sample_parking_costs_2a(N):
  list_c = np.zeros(N)
  for k in range(N):
    # Change c(k) s.t. the parking cost is stricly monotonic increasing and␣
 ↪always smaller than C
    list_c[k] = k/3
  return list_c

list_expected_cost_2a =␣
 ↪get_list_optimal_expected_cost(sample_parking_costs_2a(N))
list_actions_2_a = get_list_optimal_action(list_expected_cost_2a)

plt.plot(list_parking_index, list_expected_cost_2a, color = "#ff7f0e")
plt.xlabel("Parking index (k)")
plt.ylabel("Expected-cost at each (k)")
plt.legend(["Optimal expected-cost"], loc = 0)

plt.twinx()
plt.plot(list_parking_index, list_actions_2_a, '--', color = "#2ca02c")
```

```
plt.ylabel("Optimal-action at each (k)")
plt.legend(["0 = Don't park, 1 = Park if possible"], loc = 3)
plt.show()
```



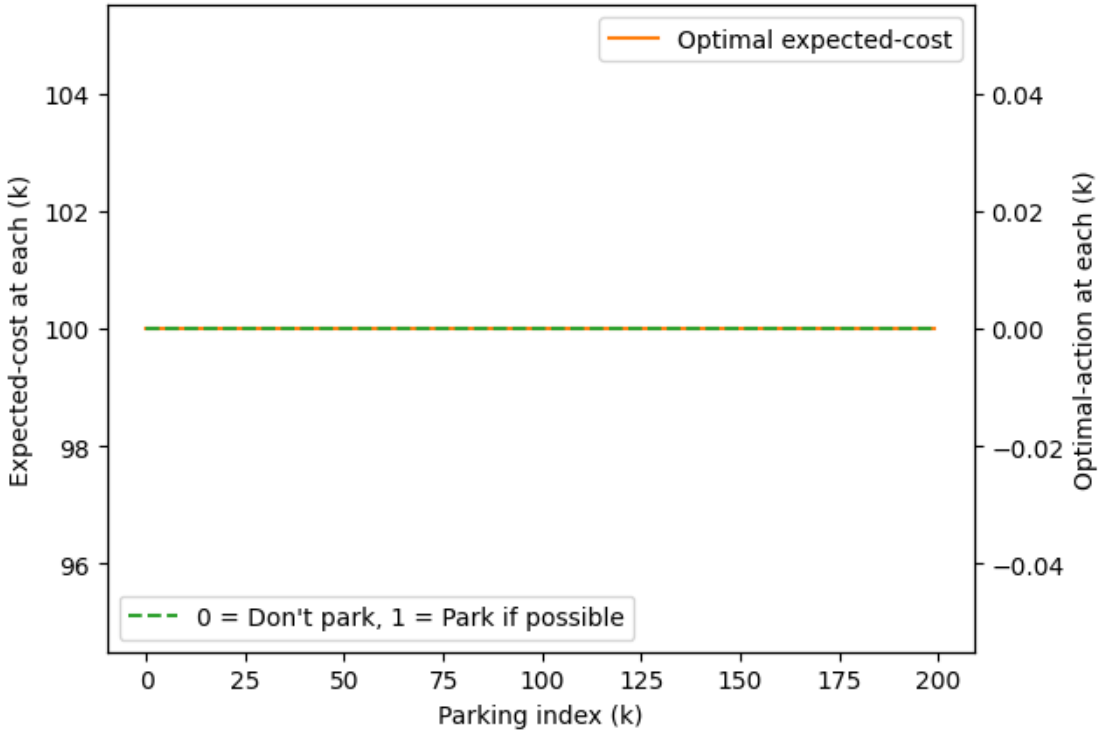
1.3.6 2.b

[ ]:
```
#Define a function to return an array refects the parking cost
def sample_parking_costs_2b(N):
  list_c = np.zeros(N)
  for k in range(N):
    #Change c(k) s.t. the parking cost is always smaller than C
    list_c[k] = N - k + 2 * C
  return list_c

list_expected_cost_2b =␣
 ↪get_list_optimal_expected_cost(sample_parking_costs_2b(N))
list_actions_2_b = get_list_optimal_action(list_expected_cost_2b)

plt.plot(list_parking_index, list_expected_cost_2b, color = "#ff7f0e")
plt.xlabel("Parking index (k)")
plt.ylabel("Expected-cost at each (k)")
plt.legend(["Optimal expected-cost"], loc = 0)
```

```
plt.twinx()
plt.plot(list_parking_index, list_actions_2_b, '--', color = "#2ca02c")
plt.ylabel("Optimal-action at each (k)")
plt.legend(["0 = Don't park, 1 = Park if possible"], loc = 3)
plt.show()
```



[9]:
```
!pip install nbconvert
!jupyter nbconvert --to pdf --output /content/problem2.pdf problem2.ipynb
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
packages (6.5.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (4.9.3)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (4.11.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (6.0.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-
```

```
packages (from nbconvert) (3.1.2)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (5.3.1)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.2.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.1.3)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.8.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.9.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (23.1)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.16.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.7.1)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert)
(3.10.0)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.10/dist-packages (from nbclient>=0.5.0->nbconvert)
(6.1.12)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-
packages (from nbformat>=5.1->nbconvert) (2.18.0)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (4.19.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
```

```
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.10.2)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (23.2.1)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.3.2)
[NbConvertApp] WARNING | pattern 'problem2.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
```

```
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
```

```
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook.
To recover
```

```
                                    previous default behaviour (outputting to the
current
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file
```

```
            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```