clip-max-ignore-zeros-asymmetric

August 13, 2020

[1]:
```
# rank the obtained results using the *.log files
import os
import pandas as pd
import numpy as np
```

[2]:
```
source = "3c"
targetdir = '../../data/' + source + "/"
filelist = sorted(os.listdir(targetdir))
```

[3]:
```
filelist
```

[3]:
```
['1.data', '2.data', '3.data', '4.data', '5.data', '6.data', '7.data']
```

[4]:
```
# Create dataframe from files
df = pd.DataFrame()

for file in filelist:
    filename = targetdir+file
    col_name = [file]
    temp_df = pd.read_csv(filename,names=col_name)
    df = pd.concat([df, temp_df], axis=1)

# Look at the data
df.head()
```

[4]:
```
   1.data  2.data  3.data  4.data  5.data  6.data  7.data
0    0.88    1.00    1.00    1.00    0.73     1.0    0.80
1    0.93    1.15    0.00    1.20    0.73     0.0    0.89
2    1.00    1.22    0.00    1.33    0.73     0.0    0.93
3    0.73    0.83    0.91    0.83    0.62     0.9    0.59
4    0.73    0.83    0.91    0.83    0.62     0.0    0.59
```

[5]:
```
# Clip values > 1 with 1 and ignore 0s
df.mask(df > 1, 1, inplace=True)
df.mask(df <= 0, np.NaN, inplace=True)
```

[6]:

```
# Count NaN values
df.isna().sum()
```

[6]:

```
1.data      0
2.data      0
3.data    324
4.data    288
5.data      0
6.data    261
7.data      0
dtype: int64
```

[7]:

```
df
```

[7]:

```
     1.data  2.data  3.data  4.data  5.data  6.data  7.data
0      0.88    1.00    1.00    1.00    0.73    1.00    0.80
1      0.93    1.00     NaN    1.00    0.73     NaN    0.89
2      1.00    1.00     NaN    1.00    0.73     NaN    0.93
3      0.73    0.83    0.91    0.83    0.62    0.90    0.59
4      0.73    0.83    0.91    0.83    0.62     NaN    0.59
..      …      …      …      …      …      …      …
724    0.81    0.87     NaN     NaN    0.64    0.88    0.61
725    0.81    0.87     NaN     NaN    0.64     NaN    0.61
726    0.69    0.74    0.99    1.00    0.53    0.50    0.52
727    0.69    0.74    0.99    1.00    0.53    0.50    0.52
728    0.69    0.74    1.00    1.00    0.53    0.50    0.52

[729 rows x 7 columns]
```

[8]:

```
# Save processed dataframe as csv file
df.to_csv("../../data/processed/asymmetric/" + source +".csv",index=False)
```

[9]:

```
# Creating ranked dataframe
ranked_df = pd.DataFrame()
stats_df = pd.DataFrame()
```

[10]:

```
# going through every column
for column in df:
    wwtp = column[0]

    # In every column, drop na values
    asym_column = df[column].dropna()

    # and calculate individual tao
    tao = len(asym_column)

    # calculate mean
    avg_eff = round(asym_column.mean(),3)

    # calculate max
    max_eff = round(asym_column.max(),3)

    # calculate min
    min_eff = round(asym_column.min(),3)

    # calculate amplitude
    amplitude = round((max_eff - min_eff)*100,2)

    amp_str = "Amplitude (max-min)(%)"

    # print stats results
    print("WWTP",wwtp,
          "\nMean =",avg_eff,"Maximum =",max_eff,"Minimum =",min_eff, amp_str,"=",amplitude)

    stats_df = stats_df.append({ 'WWTP': wwtp,
        "Mean": avg_eff, "Maximum" : max_eff, "Minimum": min_eff, amp_str: amplitude},ignore_index=True)

    # Calculating Sk sum of factors
    Sk = round(asym_column.sum(),3)

    # Calculating ek sum of factors of 1 (or above if errors in calculation)
    ek = asym_column >= 1
    ek = ek.sum()

    # Calculating R1k ek/tao
    R1k = round(ek/tao,3)

    # Calculate R2k
    if tao != ek:
        R2k = (Sk - ek)/(tao - ek)
    elif R1k == 1:
        R2k = 0

    R2k = round(R2k,3)

    # Printing results
    print("tao =",tao,"| ek =",ek,"| R1k =",R1k, "| Sk =",Sk, "| R2k =",R2k,"\n")

    # Populate ranking dataframe using pd.df.append
    # Using unicode to name columns with super and subscripts
    R1k_col = 'R\u00B9\u2096\u2080'
    R2k_col = 'R\u00B2\u2096\u2080'
    ranked_df = ranked_df.append({ R2k_col:R2k, R1k_col: R1k,'WWTP':␣
↪wwtp},ignore_index=True)
```

```
WWTP 1
Mean = 0.795 Maximum = 1.0 Minimum = 0.61 Amplitude (max-min)(%) = 39.0
tao = 729 | ek = 125 | R1k = 0.171 | Sk = 579.89 | R2k = 0.753

WWTP 2
Mean = 0.855 Maximum = 1.0 Minimum = 0.69 Amplitude (max-min)(%) = 31.0
tao = 729 | ek = 223 | R1k = 0.306 | Sk = 623.61 | R2k = 0.792

WWTP 3
Mean = 0.919 Maximum = 1.0 Minimum = 0.76 Amplitude (max-min)(%) = 24.0
tao = 405 | ek = 102 | R1k = 0.252 | Sk = 372.31 | R2k = 0.892

WWTP 4
Mean = 0.884 Maximum = 1.0 Minimum = 0.68 Amplitude (max-min)(%) = 32.0
tao = 441 | ek = 144 | R1k = 0.327 | Sk = 390.06 | R2k = 0.828

WWTP 5
Mean = 0.633 Maximum = 0.78 Minimum = 0.5 Amplitude (max-min)(%) = 28.0
tao = 729 | ek = 0 | R1k = 0.0 | Sk = 461.16 | R2k = 0.633

WWTP 6
Mean = 0.675 Maximum = 1.0 Minimum = 0.48 Amplitude (max-min)(%) = 52.0
tao = 468 | ek = 61 | R1k = 0.13 | Sk = 316.12 | R2k = 0.627

WWTP 7
Mean = 0.652 Maximum = 1.0 Minimum = 0.49 Amplitude (max-min)(%) = 51.0
tao = 729 | ek = 2 | R1k = 0.003 | Sk = 475.29 | R2k = 0.651
```

0.1 Ranking of WWTP

[11]:

```
# Reorder columns to be usable as a results table
ranked_df = ranked_df.reindex(columns=['WWTP',R1k_col, R2k_col])
```

[12]:

```
ranked_df
```

[12]:

```
   WWTP    R¹     R²
0     1  0.171  0.753
1     2  0.306  0.792
2     3  0.252  0.892
3     4  0.327  0.828
4     5  0.000  0.633
```

```
5    6  0.130  0.627
6    7  0.003  0.651
```

[13]:

```
import os

path = "../../results/" + source + "/asymmetric"

# Save rankings dataframe as csv file

try:
    ranked_df.to_csv(path + "/ranking.csv",index=False)
    print("Save succesful")
except:
    print("Creating folder and saving")
    os.mkdir(path)
    ranked_df.to_csv(path + "/ranking.csv",index=False)
```

```
Save succesful
```

0.2 Calculate Descriptive Statistics

[14]:

```
# Calculate the mean of every column
mean_mean = round(stats_df.Mean.mean(),3)
mean_max = round(stats_df.Maximum.mean(),3)
mean_min = round(stats_df.Minimum.mean(),3)
mean_amp = round(stats_df[amp_str].mean(),3)
```

[15]:

```
# Add means to stats dataframe
stats_df = stats_df.append({ 'WWTP': "Mean", "Mean" : mean_mean, "Maximum" :␣
 ↪mean_max,
                            "Minimum" : mean_min, amp_str :␣
 ↪mean_amp},ignore_index=True)
```

[16]:

```
# Calculate the standard deviation of every column
sd_mean = round(stats_df.Mean.std(),3)
sd_max = round(stats_df.Maximum.std(),3)
sd_min = round(stats_df.Minimum.std(),3)
sd_amp = round(stats_df[amp_str].std(),3)
```

[17]:

```
# Add means to stats dataframe
stats_df = stats_df.append({ 'WWTP': "SD", "Mean" : sd_mean, "Maximum" : sd_max,
                            "Minimum" : sd_min, amp_str :␣
 ↪sd_amp},ignore_index=True)
```

[18]:

```
# Reorder columns
```

```
stats_df = stats_df.reindex(columns=["WWTP", "Mean", "Maximum", "Minimum",␣
 ↪amp_str])
```

```
[19]: stats_df
```

```
[19]:    WWTP   Mean  Maximum  Minimum  Amplitude (max-min)(%)
      0     1  0.795    1.000    0.610                  39.000
      1     2  0.855    1.000    0.690                  31.000
      2     3  0.919    1.000    0.760                  24.000
      3     4  0.884    1.000    0.680                  32.000
      4     5  0.633    0.780    0.500                  28.000
      5     6  0.675    1.000    0.480                  52.000
      6     7  0.652    1.000    0.490                  51.000
      7  Mean  0.773    0.969    0.601                  36.714
      8    SD  0.110    0.077    0.105                  10.250
```

```
[20]: # Save statistics dataframe as csv file
      stats_df.to_csv(path + "/statistics.csv",index=False)
```

```
[21]: # Convert Jupyter Notebook to PDF LaTeX file
      !jupyter-nbconvert --to pdf "clip-max-ignore-zeros-asymmetric" --output-dir "../
       ↪../results/3c/asymmetric"
```

```
[NbConvertApp] Converting notebook clip-max-ignore-zeros-custom.ipynb to pdf
[NbConvertApp] Writing 45170 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 47339 bytes to ../../results/3c/asymmetric/clip-max-
ignore-zeros-custom.pdf
```

```
[23]: !jupyter-nbconvert --to pdf --output-dir "../../results/3c/asymmetric"
```

```
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.
```

```
--debug
    set log level to logging.DEBUG (maximize logging output)
--generate-config
    generate default config file
-y
    Answer yes to any questions instead of prompting.
--execute
    Execute the notebook prior to export.
--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.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
--stdout
    Write notebook output to stdout instead of files.
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--no-prompt
    Exclude input and output prompts from converted document.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL')
    Set the log level by value or name.
--config=<Unicode> (JupyterApp.config_file)
    Default: ''
    Full path of a config file.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
    'python', 'rst', 'script', 'slides'] or a dotted object name that represents
    the import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: ''
    Name of the template file to use
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: ''
    PostProcessor class used to write the results of the conversion
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    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.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: ''
    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.
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.

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

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb

    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
```

```
can specify the flavor of the format used.

> jupyter nbconvert --to html --template basic 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
```

[ ]: