Brief Introduction to the Most Convenient Water Filling Optimization Command Line Application with Plotting Support

Wuqiong Zhao, *Student Member, IEEE*
Chien-Shiung Wu College of Southeast University, Nanjing, China
National Mobile Communication Research Laboratory of Southeast University, Nanjing, China
Email: wqzhao@seu.edu.cn

***Abstract*—In convex optimization, the water filling algorithm is employed with the duality theory. This algorithm has a wide range of application especially in information science. In this paper, a brief introduction to the Water Filling (WF) application will guide you through the process of using it. WF is mainly composed of two modules, random data generator module and water filling problem optimization module. They can be implemented with command line options with great flexibility. One bonus of using WF is that it enables automatic plotting using Gnuplot. In addition to basic options useful in WF, getting and building the application will also be mentioned.**

***Index Terms*—Water Filling, Convex Optimization, Open Source Application, C++ Application**

I. Introduction

WATER filling algorithm, in the field of convex optimization, has been put into practice in information science as [1] has used. The ideas of the algorithm are elaborated in [2], where the main task is to solve the optimization problem of

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{n} \log(\alpha_i + x_i) \\ \text{subject to} & x \succeq 0, \quad \mathbf{1}^T x = 1. \end{array} \tag{1}$$

Using the theory of duality, the optimal solution x^* can be given as

$$x_i^* = \begin{cases} \frac{1}{\nu^*} - \alpha_i & , \nu^* < \frac{1}{\alpha_i} \\ 0 & , \nu^* \geq \frac{1}{\alpha_i} \end{cases} \tag{2}$$

which can be further simplified to

$$x_i^* = \max\{0, 1/\nu^* - \alpha_i\}, \tag{3}$$

where $1/\nu^*$ is actually the optimal water level shown in Fig. 1 with requirement

$$\sum_{i=1}^{n} \max\{0, 1/v^* - \alpha_i\} = 1. \tag{4}$$

So the key part of solving the optimization problem is to find the optimal water level.

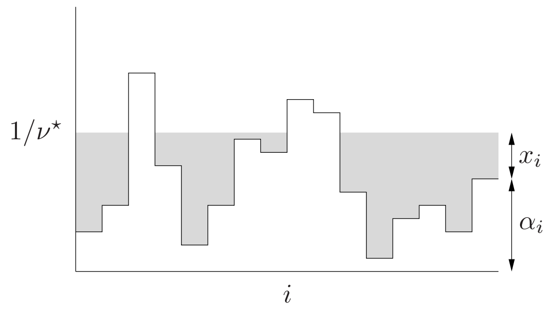


Fig. 1. Illustration of water-filling algorithm. The height of each patch is given by α_i. The region is flooded to a level $1/\nu^*$ which uses a total quantity of water equal to one. The height of the water (shown shaded) above each patch is the optimal value of x_i^*. © [2022] Cambridge University Press. Reprinted, with permission from Ref. [2].

The remainder of the paper is structured as follows. Section II shows the main process of algorithm implementation of finding the water level. Section III gives the basic application information of Water Filling (WF) Application. The random data generator module and the optimization module will be elaborated in section IV and V respectively. The conclusion will be given in section VI. Appendix A will give information on getting and building the application. Appendix B will present the API documentation guide.

II. Algorithm Choosing

There can be two methods of finding the optimal water level. First is to sort the vector α and fill water level by level. The good point of this method is that it can yield an accurate solution. Another method is using the dichotomy, which is detailed in Alg. 1. The main idea is to compare the current sum of water with expected sum and then adjust the upper

Manuscript received Jan 2, 2022; accepted Jan 2, 2022. Date of publication Jan 3, 2022; date of current version Jan 2, 2022.

This work is intended to give the users of Water Filling application a brief intriduction like a tutorial. Besides, this is also the course essay of Southeast University Convex Optimization 2021 Fall.

The website of the author is https://teddy-van-jerry.org and this project has the documentation in https://wf.teddy-van-jerry.org. The source code used in this paper can be found at https://github.com/Teddy-van-Jerry/Water_Filling.

```
$ ./waterfilling -h
Usage: ./waterfilling <command> [options]

Commands:
  optimize                         solve the optimization problem
  generate                         generate random test data
  (Leave empty)                    generic use

Allowed options:

Generic options:
  -v [ --version ]                 print version string
  -h [ --help ]                    produce help message

Configuration:
  -i [ --input-data ] arg          input data file name
  -o [ --output ] arg              output file name
  -p [ --precision ] arg (=6)      optimization precision, i.e. -log10(LINEAR_PRECISION)
  -k [ --iter-max ] arg (=1000)    maximum iteration times
  --print arg (=1)                 print results on console
  --plt arg (=0)                   plot results using GNU Plot
  --plt.title arg                  plot title
  --plt.file arg                   plot file name
  --plt.c.low arg (=lemonchiffon)  plot color of the lower bar
  --plt.c.up arg (=seagreen)       plot color of the upper bar
  --plt.w arg                      plot image width
  --plt.h arg                      plot image height
  -m [ --mode ] arg (=uniform)     random data mode ('uniform' or  'normal')
  -l [ --length ] arg              length of test data set
  --min arg                        min value for uniform distribution random data generation
  --max arg                        max value for uniform distribution random data generation
  --mean arg                       mean value for normal distribution random data generation
  --sigma arg                      standard deviation for normal distribution random data generation
  --abs arg (=1)                   abs generated data
  --norm arg                       normalize generated data
  --norm-dim arg (=2)              norm dimension
```

limit or lower limit. Therefore, the parameter ε is needed to specify the accuracy we expect, which works together with maximum iteration counts k to determine the termination of iteration update of the water level $1/\nu^*$.

Algorithm 1 Dichotomy Implementation of Water Filling

Input: α, k, ε.
Init: $n = \|\alpha\|_0, (1/\nu)_{\min} = 1/n, (1/\nu)_{\max} = (\|\alpha\|_1 + 1)/n$.

```
1:  for $i = 1 : k$ do
2:      $l = ((1/\nu)_{\min} + (1/\nu)_{\max})/2$;          ▷ Water level.
3:      for $j = 0 : n$ do
4:          $\hat{x}_i = \max\{0, l - \alpha_i\}$;
5:      end for
6:      if $|\|\hat{x}\|_1 - 1| < \varepsilon$ then
7:          break;                                   ▷ Accurate enough.
8:      else
9:          if $\|\hat{x}\|_1 < 1$ then
10:             $(1/\nu)_{\min} = l$;                    ▷ Update lower limit.
11:         else
12:             $(1/\nu)_{\max} = l$;                    ▷ Update upper limit.
13:         end if
14:     end if
15: end for
16: $\nu^* = \nu, x = \hat{x}$;
```

Output: ν^*, x.

Compared with the other approach, this method is especially efficient when the size of α is very large.

III. Application Information

The WF application is written using C++ with CMake, and has dependencies on Boost C++ Libraries and Gnuplot which will be elaborated on in Appendix A. This application can work on Linux, MacOS and Windows.

All commands and options used in the application can be found by using the option `-h` or `--help`, and the result is listed at the top of the page.

IV. Generating Random Data

The most basic random data generation is listed below. The command name is `generate`, followed by options `-o` specifies the location of the generated data file and `-l` specifying the length of test data.

```
$ ./waterfilling generate -odata.txt -l10
0.13047, 0.0612013, 0.787236, 0.655578, 0.500067,
 ↪ 0.684892, 0.178797, 0.651522, 0.287236,
 ↪ 0.0575645
```

Currently, WF provides two kinds of random distributions, including uniform distribution and normal distribution specified by option `-m` or `--mode`. Besides, you can get the absolute value by setting `-abs` to `true`. You can also normalize the data by using option `--norm` and `--norm-dim`, where the default of norm dimension is 2.

The following example uses the normal distribution, where $\alpha_i \sim \mathcal{N}(0.5, 2^2)$ and then it is normalized by 1-norm to 1.

```
$ ./waterfilling generate -odata.txt -l10 -mnormal
 ↪ --mean 0.5 --sigma 2 --abs true --norm 1
 ↪ --norm-dim 1
0.0422217, 0.1123, 0.00212039, 0.0480777, 0.131173,
 ↪ 0.231008, 0.057158, 0.160638, 0.0224425, 0.19286
```

The uniform distribution can implemented similarly. Here by specifying `--print` to `false`, no data will be printed on the terminal.

```
$ ./waterfilling generate -odata.txt -l10 -muniform
 ↪  --min 0 --max 1 --norm 1.5 --norm-dim 2
 ↪ --print false
```

It should be noticed if option `-o` is not specified, the generated data will not be saved.

V. Optimization Implementation

With data in file `data.txt`, use command `optimize` and then the result will be printed on the terminal.

```
$ ./waterfilling optimize -idata.txt
0.394069, 0.119994, 0, 0.0157279, 0.180826, 0,
 ↪ 0.0189609, 0, 0.270417, 0
```

You can set the output file using `-o` and set `-print` to `false`. Precision is set using `-p` or `--precision`, max iteration is set using `-k` or `--iter-max`.

```
$ ./waterfilling optimize -idata.txt -oresult.txt
 ↪ --print false -p8 -k500
```

To plot, make sure you have Gnuplot installed on your PC. You can install it on Linux or MacOS easily.

```
$ sudo apt-get install gnuplot # Ubuntu or Debian
$ sudo pacman -Syu gnuplot     # Arch
$ sudo port install gnuplot    # MacOS with MacPort
```

For Windows users, you need to add system path after installing Gnuplot.

WF provides a wide range of image output format, including `png`, `jpg`, `jpeg`, `eps`, `pdf`, `svg`, and even `tex` TikZ plot, where the `--plt` option should be set to `true`. `--plt.file` sets the output file name and this should be distinguished from the option `--plt.title` which adds a title above the graph. The following example creates an eps image with the default settings. The y-axis of the image is auto scaled so it always looks good.

```
$ ./waterfilling optimize -idata.txt --plt true
 ↪ --plt.file plot.eps
0.394069, 0.119994, 0, 0.0157279, 0.180826, 0,
 ↪ 0.0189609, 0, 0.270417, 0
```

The output image is shown in Fig. 2[1].

[1]All these images are generated on MacOS Big Sur (M1 chip) with Gnuplot 5.4. The generated images can be different on different platforms or with different versions of Gnuplot.

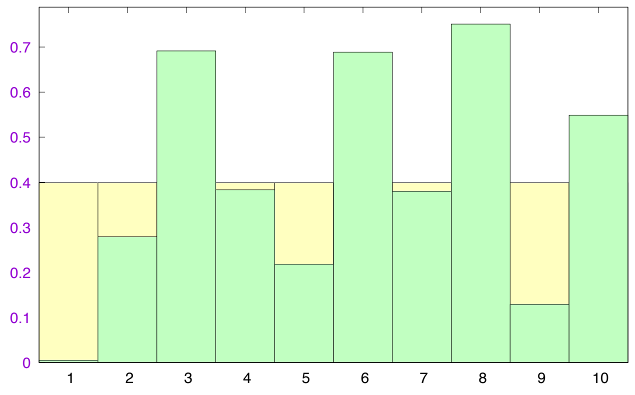


Fig. 2. Eps plot with the default settings.

You can also set its size and color with option `--plt.w` for width, `--plt.h` for height, `--plt.c.up` for upper bar color and `--plt.c.low` for lower bar color. The following example uses the TikZ plot with `tex` extension. Notice that the naming for colors should follow the requirements of Gnuplot.

```
$ ./waterfilling optimize -idata.txt --print false
 ↪ --plt true --plt.file plot.pdf --plt.title
 ↪ "Water Filling Optimization Result" --plt.w 4
 ↪ --plt.h 3 --plt.c.up "#dda0dd" --plt.c.low
 ↪ royalblue
```

The output image is shown in Fig. 3.

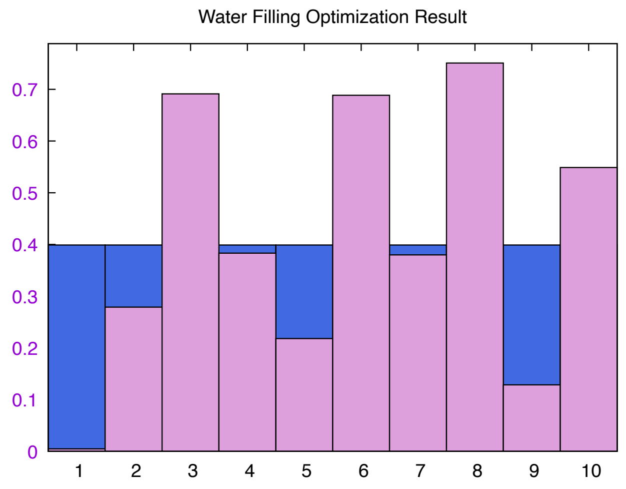


Fig. 3. Pdf plot with customized settings.

The WF application can deal with more data and therefore an even more magnificent image is generated shown in Fig. 4 with the following commands.

```
$ ./waterfilling generate -odata.txt -l100 --norm
 ↪ 1.5 --print false
$ ./waterfilling optimize -idata.txt --plt true
 ↪ --plt.file plot.eps --plt.h 2 --plt.w 20
 ↪ --print false
```

Most options concerning the optimization implementation has been convered and more options will be aded in the future update to make the WF application even more powerful.

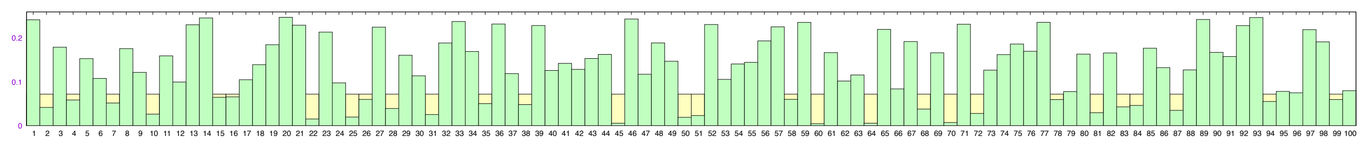


Fig. 4. Eps plot with generated data size of 100.

VI. Conclusion

This paper gives a brief intriductinon to the Water Filling (WF) application. The main contribution of the paper is that the author designs an open source application with command line interface, making it easy to optimize the water filling problem. Besides, the support of plotting adds a new dimension to the application which is now intersting to use and easy to view the optimized result. In the future, more options will be added to make the application to have a even wider range of use.

Appendix A
Get and Build the Application

The project of WF is open source licensed under the MIT License at GitHub, and the repository is Teddy-van-Jerry/Water_Filling [3]. You can clone or download the build binary package or source file in the release page. Since the build version may not be compatible with your system so it is strongly recommended that you build the project yourself with CMake which is open source available at [4]. To compile, you need a C++ compiler supporting at least C++ 17 standard [5] and the Boost library available at [6]. To use the plotting function of WF, you need to have Gnuplot available at [7].

Fig. 5. WF Logo

With the requirements met, you can compile with the command `cmake` or to specify the Boost root directory with the command listed below.

```
$ cmake . -DBOOST_ROOT=<dir>
```

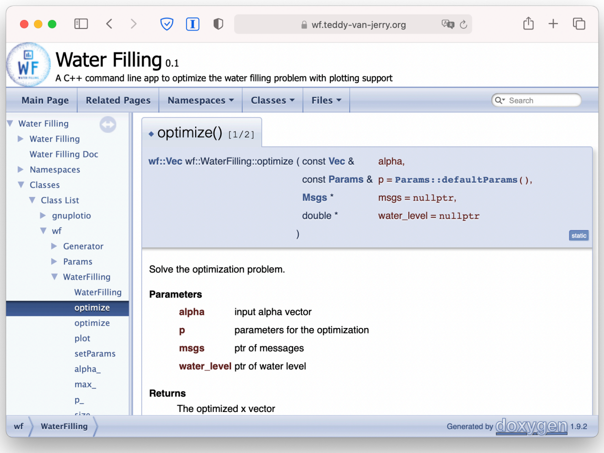
If this process is successful, the executable will appear in directory `bin`.

Appendix B
Application Documentation

WF has the API documentation generated by Doxygen at https://wf.teddy-van-jerry.org. The example Doxygen page is shown in Fig. 6

All classes and functions related to WF are in the namespace `wf`, which distinguishes them from dependency repository classes and functions like those of `iostream-gnuplot` open source at [8]. Three main classes `WaterFilling`, `Generator` and `Params` work separately, making them easy to maintain.

Since this project is open source, you are welcome to pull request at GitHub or publish issues.



◆ optimize() [1/2]

```
wf::Vec wf::WaterFilling::optimize ( const Vec &     alpha,
                                     const Params &  p = Params::defaultParams(),
                                     Msgs *          msgs = nullptr,
                                     double *        water_level = nullptr
                                   )
```

static

Solve the optimization problem.

Parameters

alpha	input alpha vector
p	parameters for the optimization
msgs	ptr of messages
water_level	ptr of water level

Returns
The optimized x vector

Fig. 6. API documentation generated using Doxygen.

References

[1] W. Yu, W. Rhee, S. Boyd *et al.*, “Iterative water-filling for Gaussian vector multiple-access channels,” *IEEE Transactions on Information Theory*, vol. 50, no. 1, pp. 145–152, 2004.
[2] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[3] T. v. Jerry, “Water filling,” GitHub, 2022. [Online]. Available: https://github.com/Teddy-van-Jerry/Water_Filling
[4] CMake, “Build with cmake. build with confidence.” 2022. [Online]. Available: https://cmake.org
[5] ISO, *ISO/IEC 14882:2017 — Programming languages C++*. ISO, 2017.
[6] Boost, “Boost C++ libraries,” 2022. [Online]. Available: https://www.boost.org
[7] Gnuplot, “Gnuplot,” 2022. [Online]. Available: http://www.gnuplot.info
[8] D. Stahlke, “Gnuplot iostream,” GitHub, 2022. [Online]. Available: https://github.com/dstahlke/gnuplot-iostream

Wuqiong Zhao (Student Member, IEEE) is an undergraduate student pursuing the Bachelor’s Degree in information science, working at Lab of Efficient Architectures for Digital-communication and Signal-processing (LEADS) and National Mobile Communications Research Laboratory, Southeast University. He is the honors (number one) student of Chien-Shiung Wu College which is a pilot college and training ground in Southeast University to cultivate top-notch undergraduate students selected from multiple science and engineering departments. He earned the National Scholarship and Cyrus Tang Scholarship in 2021. His research interest includes channel estimation, Bayesian algorithms, and the intelligent reflecting surface (IRS) in wireless communication of 5G and 6G. He assisted editing the book ‘Channel Codes for 5G Wireless Systems’ and the chapter ‘Stochastic Computation for Baseband Processing’. He is now working on the research ‘OMPL-SBL Algorithm for Intelligent Reflecting Surface Channel Estimation’.