Efficient resolution of the Colebrook equation

Didier Clamond

Efficient resolution of the Colebrook equation

Didier Clamond

Laboratoire J.-A. Dieudonné, 06108 Nice cedex 02, France.
E-Mail: didierc@unice.fr

Abstract

A robust, fast and accurate method for solving the Colebrook-like equations is presented. The algorithm is efficient for the whole range of parameters involved in the Colebrook equation. The computations are not more demanding than simplified approximations, but they are much more accurate. The algorithm is also faster and more robust than the Colebrook solution expressed in term of the Lambert W-function. Matlab© and FORTRAN codes are provided.

1 Introduction

Turbulent fluid flows in pipes and open channels play an important role in hydraulics, chemical engineering, transportation of hydrocarbons, for example. These flows induce a significant loss of energy depending on the flow regime and the friction on the rigid boundaries. It is thus important to estimate the dissipation due to turbulence and wall friction.

The dissipation models involve a friction coefficient depending on the flow regime (via a Reynolds number) and on the geometry of the pipe or the channel (via an equivalent sand roughness parameter). This friction factor if often given by the well-known Colebrook–White equation, or very similar equations.

The Colebrook–White equation estimates the (dimensionless) Darcy–Weisbach friction factor λ for fluid flows in filled pipes. In its most classical form, the Colebrook–White equation is

$$\frac{1}{\sqrt{\lambda}} \;=\; -\,2\,\log_{10}\!\left(\frac{K}{3.7} \,+\, \frac{2.51}{R}\,\frac{1}{\sqrt{\lambda}}\right), \tag{1}$$

where $R = UD/\nu$ is a (dimensionless) Reynolds number and $K = \epsilon/D$ is a relative (dimensionless) pipe roughness (U the fluid mean velocity in the pipe, D the pipe hydraulic diameter, ν the fluid viscosity and ϵ the pipe absolute roughness height). There exist several variants of the Colebrook equation, e.g.

$$\frac{1}{\sqrt{\lambda}} \;=\; 1.74 \,-\, 2\,\log_{10}\!\left(2\,K \,+\, \frac{18.7}{R}\,\frac{1}{\sqrt{\lambda}}\right), \tag{2}$$

$$\frac{1}{\sqrt{\lambda}} \;=\; 1.14 \,-\, 2\,\log_{10}\!\left(K \,+\, \frac{9.3}{R}\,\frac{1}{\sqrt{\lambda}}\right). \tag{3}$$

These variants can be recast into the form (1) with small changes in the numerical constants 2.51 and 3.7. Indeed, the latter numbers being obtained fitting experimental data, they are known with limited accuracy. Thus, the formulae (2) and (3) are not fundamentally different from (1). Similarly, there are variants of the Colebrook equations for open channels, which are very similar to (1). Thus, we shall focus on the formula (1), but it is trivial to adapt the resolution procedure introduced here to all variants, as demonstrated in this paper.

The Colebrook equation is transcendent and thus cannot be solved in terms of elementary functions. Some explicit approximate solutions have then been proposed [6, 10, 12]. For instance, the well-known Haaland formula [6] reads

$$\frac{1}{\sqrt{\lambda}} \;\approx\; -1.81 \times \log_{10}\left[\frac{6.9}{R} \;+\; \left(\frac{K}{3.7}\right)^{1.11}\right]. \tag{4}$$

Haaland's approximation is explicit but is not as simple as it may look. Indeed, this approximation involves one logarithm only, but also a non-integer power. The computation of the latter requires the evaluation of one exponential and one logarithm, since it is generally computed via the relation

$$x^{1.11} \;=\; \exp(1.11 \times \ln(x)),$$

where 'ln' is the natural (Napierian) logarithm. Hence, the overall evaluation of (4) requires the computation of three transcendant functions (exponentials and logarithms). We present in this paper much more accurate approximations requiring the evaluation of only two or three logarithms, plus some trivial operations $(+, -, \times, \div)$.

Only quite recently, it was noticed that the Colebrook–White equation (1) can be solved in closed form [8] using the long existing Lambert W-function [3]. However, when the Reynolds number is large, this exact solution in term of the Lambert function is not convenient for numerical computations due to overflow errors [11]. To overcome this problem, Sonnad and Goudar [11, 12] proposed to combine several approximations depending on the Reynolds number. These approaches are somewhat involve and it is actually possible to develop a simpler and more efficient strategy, as we demonstrate in this paper.

A fast, accurate and robust resolution of the Colebrook equation is, in particular, necessary for scientific intensive computations. For instance, numerical simulations of pipe flows require the computation of the friction coefficient at each grid point and for each time step. For long term simulations of long pipes, the Colebrook equation must therefore be solved a huge number of times and hence a fast algorithm is required. An example of such demanding code is the program OLGA [1] which is widely used in the oil industry.

Although the Colebrook formula itself is not very accurate, its accurate resolution is nonetheless an issue for numerical simulations because a too crude resolution may affect the repeatability of the simulations. Robustness is also important since one understandably wants an algorithm capable of dealing with all the possible values of the physical parameters involved in the model. The

method described in the present paper was developed to address all this issues. It is also very simple so it can be used for simple applications as well. The method proposed here aims at giving a definitive answer to the problem of solving numerically the Colebrook-like equations.

The paper is organized as follow. In section 2, a general Colebrook-like equation and its solution in term of the Lambert W-function are presented. For the sake of completeness, the Lambert function is briefly described in section 3, as well as a standard algorithm used for its computation. A severe drawback of using the Lambert function for solving the Colebrook equation is also pointed out. To overcome this problem, a new function is introduced in section 4 and an improved new numerical procedure is described. Though this function introduces a big improvement for the computation of the friction factor, it is still not fully satisfactory for solving the Colebrook equation. The reasons are explained in the section 5, where a modified function is derived to address the issue. The modified function is subsequently used in section 6 to solve the Colebrook equation efficiently. The accuracy and speed of the new algorithm is tested and compared with Haaland's approximation. For testing the method and for intensive practical applications, MATLAB© and FORTRAN implementations of the algorithm are provided in the appendices. The algorithm is so simple that it can easily be implemented in any other language and modified to be adapted to any variants of the Colebrook equation.

2 Generic Colebrook equation and its solution

We consider here a generic Colebrook-like equation as

$$\frac{1}{\sqrt{\lambda}} \;=\; c_0 \;-\; c_1 \,\ln\!\left(c_2 \,+\, \frac{c_3}{\sqrt{\lambda}} \right), \tag{5}$$

where the c_i are given constants such that $c_1 c_3 > 0$. The classical Colebrook–White formula (1) is obviously obtained as a special case of (5) with $c_0 = 0$, $c_1 = 2/\ln 10$, $c_2 = K/3.7$ and $c_3 = 2.51/R$.

The equation (5) has the exact analytical solution

$$\frac{1}{\sqrt{\lambda}} \;=\; c_1 \left[\mathrm{W}\!\left(\exp\left(\frac{c_0}{c_1} + \frac{c_2}{c_1 c_3} - \ln(c_1 c_3) \right) \right) \;-\; \frac{c_2}{c_1 c_3} \right], \tag{6}$$

which is real if $c_1 c_3 > 0$ and where W is the principal branch of the Lambert function, often denoted W_0 [3]. In this paper, only the principal branch of the Lambert function is considered because the other branches correspond to non-physical solutions of the Colebrook equations, so the simplified notation W is not ambiguous.

3 Brief introduction to the Lambert W-function

For the sake of completeness, we briefly introduce the Lambert function and its practical computation. Much more details can be found in [3, 4].

The Lambert W-function solves the equation

$$y \exp(y) \;=\; x \qquad \Longrightarrow \qquad y \;=\; \mathrm{W}(x), \tag{7}$$

where, here, x is real — more precisely $x \geqslant -\exp(-1)$ — and $\mathrm{W}(0) = 0$. The Lambert function cannot be expressed in terms of elementary functions. An efficient algorithm for its computation is based on Halley's iterations [3]

$$y_{j+1} \;=\; y_j \;-\; \frac{y_j \exp(y_j) \;-\; x}{(y_j + 1) \exp(y_j) \;-\; \frac{1}{2}(y_j + 2)\,(y_j \exp(y_j) - x)/(y_j + 1)}, \tag{8}$$

provided an initial guess y_0. Halley's method is cubic (c.f. Appendix A), meaning that the number of exact digits is (roughly) multiplied by three after each iteration. Today, programs for computing the Lambert function are easily found. For instance, an efficient implementation in MATLAB© (including complex argument and all the branches) is freely available [5].

The Taylor expansion around $x = 0$ of the Lambert function is

$$\mathrm{W}(x) \;=\; \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!}\, x^n, \qquad |x| < \exp(-1). \tag{9}$$

This expansion is of little interest to solve the Colebrook equation because, in this context, the corresponding variable x is necessarily large ($x \gg 1$). It is thus more relevant to consider the asymptotic expansion

$$\mathrm{W}(x) \;\sim\; \ln(x) \;-\; \ln(\ln(x)) \qquad \text{as} \quad x \to \infty. \tag{10}$$

This expansion reveals that W behaves logarithmically for large x, while we must compute $\mathrm{W}(\exp(x))$ to solve the Colebrook equation, c.f. relation (6). For our applications, x is large and $\exp(x)$ is therefore necessarily huge, to an extend that the computation of $\exp(x)$ cannot be achieved due to overflow. Even when the intermediate computations can be done, the result can be very inaccurate due to large round-off errors. Therefore, the resolution of the Colebrook equation via the Lambert function [8] is not efficient for the whole range of parameter of practical interest [11].

4 The ω-function

To overcome the numerical difficulties related to the Lambert W-function, when used for solving the Colebrook–White equation, we introduce here a new function: the ω-function.

The ω-function is defined such that it solves the equation

$$y \;+\; \ln(y) \;=\; x \qquad \Longrightarrow \qquad y \;=\; \omega(x), \tag{11}$$

where we consider only real x. The ω-function is related to the W-function as

$$\omega(x) \;=\; \mathrm{W}(\exp(x)).$$

Note that the Lambert W-function is also sometimes called the Omega function, that should not be confused with the ω-function defined here, where we follow the notation used in [9]. In terms of the ω-function, the solution of (5) is of course

$$\frac{1}{\sqrt{\lambda}} \;=\; c_1 \left[\omega\!\left(\frac{c_0}{c_1} + \frac{c_2}{c_1 c_3} - \ln(c_1 c_3) \right) \;-\; \frac{c_2}{c_1 c_3} \right]. \tag{12}$$

For large arguments $\omega(x)$ behaves like x, i.e. we have the asymptotic behavior

$$\omega(x) \;\sim\; x \;-\; \ln(x) \qquad \text{as} \quad x \to \infty, \tag{13}$$

which is an interesting feature for the application considered in this paper.

As noted by Corless *et al.* [4], the equation (11) is in some ways nicer than (7). In particular, its derivatives (with respect of y) are simpler, leading thus to algebraically simpler formulae for its numerical resolution. An efficient iterative quartic scheme (c.f. Appendix A) is thus

$$y_{j+1} \;=\; y_j \;-\; \frac{\left(1 + y_j + \frac{1}{2}\epsilon_j\right) \epsilon_j \, y_j}{\left(1 + y_j + \epsilon_j + \frac{1}{3}\epsilon_j^2\right)} \qquad \text{for} \quad j \geqslant 1, \tag{14}$$

with

$$\epsilon_j \;\equiv\; \frac{y_j \;+\; \ln(y_j) \;-\; x}{1 \;+\; y_j}, \qquad y_0 \;=\; x \;-\; \frac{1}{5}.$$

The computationally costless initial guess ($y_0 = x - \frac{1}{5}$) was obtained considering the asymptotic behavior (10), minus an empirically found correction (the term $-\frac{1}{5}$) to improve somewhat the accuracy of y_0 for small x without affecting the accuracy for large x. The relative error e_j of the j-th iteration, i.e.

$$e_j(x) \;\equiv\; \left| \frac{y_j(x) \;-\; \omega(x)}{\omega(x)} \right|,$$

is displayed on the figure 1 for $1 \leqslant x \leqslant 10^6$ and $j = 0, 1, 2$. (The accuracy of (14) were measured using the arbitrary precision capability of MATHEMATICA©.) We can see that with $j = 2$ we have already reached the maximum accuracy possible when computing in double precision, since $\max(e_2) \approx 4 \times 10^{-17}$ for $x \in [1; \infty[$. We note that the relative error continues to decay monotonically as x increases (even for $x > 10^6$) and that there are no overflow problems when computing y_j even for very large x (i.e. $x \gg 10^6$). We note also that for $x \gtrapprox 5700$ the machine double precision is obtained after one iteration only.

The scheme (14) is quartic, meaning that the number of exact digits is multiplied by four after each iteration (c.f. Appendix A). Hence, starting with an initial guess with one correct digit, four digits are exact after one iteration and sixteen after two iterations. That is to say that the machine precision (if working in double precision) is achieved after two iterations only (Fig. 1). Moreover, the scheme (14) has a comparable algebraic complexity per iteration than the scheme (8), i.e. the computational times per iteration are almost identical. However, the iterative quartic scheme (14) converges faster than the cubic

one (8), and there are no overflow problems as they appear when computing $\mathrm{W}(\exp(x))$ for large x. This algorithm could therefore be used to compute the solution of the Colebrook–White equation (1), but we will use instead an even better one defined in the next section. We note in passing that the iterations (14) are also efficient for computing the ω-function for any complex x, provided some changes in the initial guess y_0 depending on x.

Remarks:

i- With a more accurate initial guess y_0, such as $y_0 = x - \ln(x)$, the desired accuracy may be obtained with fewer iterations. However, the computation of such an improved initial guess requires the evaluation of transcendent functions. Thus, it cannot be significantly faster than the evaluation of y_1 with (14) from the simplest guess $y_0 = x - \frac{1}{5}$, and most likely less accurate.

ii- Higher-order iterations are generally more involved per iteration than the low-order ones. Higher-order iterations are thus interesting if the number of iterations is sufficiently reduced so that the total computation is faster to achieve the desired accuracy. This is precisely the case here.

iii- Intensive tests have convinced us that the choice of the simplest initial guess $y_0 = x - \frac{1}{5}$ together with the quartic iterations (14) is probably the best possible scheme for computing the ω-function in the interval $x \in [1; \infty[$, at least when working in double precision. If improvements can be found, they are thus most likely very minor in terms of both robustness, speed and accuracy.

5 The ϖ-function

Solving the Colebrook equation via the ω-function is a big improvement compared to its solution in term of the Lambert W-function. One can check that the numerical resolution of the Colebrook equation via the algorithm (14) is indeed very efficient when $K = 0$, even for very large R. However, when $K > 0$ the scheme (14) is not so effective for large R, meaning that not all the numerical shortcomings have been addressed introducing the ω-function. The cause for these numerical problems can be explained as follow.

The solution of the Colebrook equation requires the computation of an expression like $\omega(x_1 + x_2) - x_1$ where $x_1 \gg x_2$ when R is large and $K \neq 0$ (but $x_1 = 0$ if $K = 0$), see the relation (19) below. Assuming $x_2 \propto \ln(x_1)$, as is the case here, the asymptotic expansion as $x_1 \to \infty$, i.e.

$$\omega(x_1 + x_2) \; - \; x_1 \; \sim \; (x_1 + x_2 - \ln(x_1)) \; - \; x_1 \; = \; x_2 \; - \; \ln(x_1),$$

exhibits the source of the numerical problems. Indeed, when $K > 0$ and R is large, we have $x_1 \gg x_2$ and $x_1 \gg \ln(x_1)$. Therefore $|x_2 - \ln(x_1)|/x_1$ can be smaller than the accuracy used in the computation and we thus obtain numerically $x_1 + x_2 - \ln(x_1) \approx x_1$ due to round-off errors. Hence $\omega(x_1 + x_2) - x_1 \approx 0$ is computed instead of $\omega(x_1 + x_2) - x_1 \approx x_2 - \ln(x_1)$. To overcome this problem we introduce yet another function: the ϖ-function.

Introducing the change of variable $y = z + x_1$ into the equation (11), the ϖ-function is defined such that it solves the equation

$$z \;+\; \ln(x_1 + z) \;=\; x_2 \qquad \Longrightarrow \qquad z \;=\; \varpi(x_1 \,|\, x_2\,), \tag{15}$$

where the x_i are real. The ϖ-function is related to the ω- and W-functions as

$$\varpi(x_1 \,|\, x_2\,) \;=\; \omega(x_1 + x_2) \;-\; x_1 \;=\; \mathrm{W}(\exp(x_1 + x_2)) \;-\; x_1.$$

In terms of the ϖ-function, the solution of (5) is obviously

$$\frac{1}{\sqrt{\lambda}} \;=\; c_1\, \varpi\!\left(\frac{c_2}{c_1 c_3} \,\middle|\, \frac{c_0}{c_1} - \ln(c_1 c_3) \right). \tag{16}$$

The ϖ-function is nothing more than the ω-function shifted by the quantity x_1. This is a very minor analytic modification but this is a numerical significant improvement when x_1 is large.

An efficient numerical algorithm for computing the ϖ-function is directly derived from the scheme (14) used for the ω-function. We thus obtain at once

$$z_{j+1} \;=\; z_j \;-\; \frac{\left(1 + x_1 + z_j + \frac{1}{2}\epsilon_j\right) \epsilon_j \,(x_1 + z_j)}{\left(1 + x_1 + z_j + \epsilon_j + \frac{1}{3}\epsilon_j^{\,2}\right)} \qquad \text{for} \quad j \geqslant 1, \tag{17}$$

with

$$\epsilon_j \;\equiv\; \frac{z_j \;+\; \ln(x_1 + z_j) \;-\; x_2}{1 \;+\; x_1 \;+\; z_j}, \qquad z_0 \;=\; x_2 \;-\; \frac{1}{5}.$$

If $x_1 = 0$ the scheme (14) is recovered. The rate of convergence of (17) is of course identical to the scheme (14). Thus, the efficiency of (17) does not need to be re-discussed here (see section 4).

6 Resolution of the Colebrook–White equation

We test the new procedure with the peculiar Colebrook–White equation (1). Its general solution is

$$\begin{aligned}
\frac{1}{\sqrt{\lambda}} \;&=\; \frac{2}{\ell}\left[\mathrm{W}\left(\exp\left(\frac{\ell\, K\, R}{18.574} \;+\; \ln\!\left(\frac{\ell\, R}{5.02} \right) \right) \right) \;-\; \frac{\ell\, K\, R}{18.574} \right] && (18)\\
&=\; \frac{2}{\ell}\left[\omega\!\left(\frac{\ell\, K\, R}{18.574} \;+\; \ln\!\left(\frac{\ell\, R}{5.02} \right) \right) \;-\; \frac{\ell\, K\, R}{18.574} \right] && (19)\\
&=\; \frac{2}{\ell}\, \varpi\!\left(\frac{\ell\, K\, R}{18.574} \,\middle|\, \ln\!\left(\frac{\ell\, R}{5.02} \right) \right), && (20)
\end{aligned}$$

where $\ell = \ln(10) \approx 2.302585093$. All these analytic solutions are mathematically equivalent, but the relation (20) is more efficient for numerical computations if we use the scheme described in the previous section.

6.1 Numerical procedure

The solution of the Colebrook–White equation is obtained computing the ϖ-function with

$$x_1 \;=\; \frac{\ell\, K\, R}{18.574}, \qquad x_2 \;=\; \ln\!\left(\frac{\ell\, R}{5.02}\right),$$

and using the iterative scheme (17) with $j = 0, 1, 2$. An approximation of the friction factor is eventually

$$\lambda_j \;\approx\; (\,\ell \,/\, 2\, z_j\,)^2.$$

This way, the whole computation of λ_j requires the evaluation of $j+1$ logarithms only,[1] i.e. one logarithm per iteration.

A MATLAB© implementation of this algorithm is given in the appendix B. This (vectorized) code was written with clarity in mind, so that one can test and modify easily the program. This program is also fast, accurate and robust, so it can be used in real intensive applications developed in MATLAB.

A FORTRAN implementation of this algorithm is given in the appendix C. This program was written with speed in mind, so there are no checks of the input parameters. The code is clear enough that it should be easy to modify and to translate into any programming language.

6.2 Accuracy

For the range of Reynolds numbers $10^3 \leqslant R \leqslant 10^{13}$ and for four relative roughness $K = \{0, 10^{-3}, 10^{-2}, 10^{-1}\}$, the accuracy of $\lambda_j^{-1/2}$ — obtained from the iterations (17) with $j = \{0, 1, 2\}$ — and of Haaland's approximation $\lambda_{\mathrm{H}}^{-1/2}$ — given by (4) — are compared with the exact friction coefficient $\lambda^{-1/2}$. The relative errors are displayed on the figure 2.

It appears clearly that $\lambda_2^{-1/2}$ is accurate to machine double-precision (at least) for all Reynolds numbers and for all roughnesses (in the whole range of physical interest, and beyond).

It also appears that $\lambda_1^{-1/2}$ is more accurate than Haaland's approximation, specially for large R and K. Moreover, the computation of λ_1 requires the evaluation of only two logarithms, so it is faster than Haaland's formula. Note that other explicit approximations having more or less the same accuracy as Haaland's formula, $\lambda_1^{-1/2}$ is significantly more accurate than these approximations.

Finally, we note that $\lambda_0^{-1/2}$ is a too poor approximation to be of any practical interest.

6.3 Speed

Testing the actual speed of an algorithm is a delicate task because the running time depends of many factors independent of the algorithm itself (imple-

[1] The numerical constant ln(10) is not counted because it can be explicitly given in the program and does not need to be computed each time.

mentation, system, compiler, hardware, etc.), specially on multi-tasking and multi-users computers. In order to estimate the speed of our scheme as fairly as possible, the following methodology was used.

The speeds of the computation of λ_1 and λ_2 are compared with the Haaland approximation λ_{H}. The MATLAB environnement and its built-in cputime function is used, for simplicity.

Two vectors of N components, with $1 \leqslant N \leqslant 10^5$, are created for R and K. The values are chosen randomly in the intervals $10^3 \leqslant R \leqslant 10^9$ and $0 \leqslant K < 1$. The computational times are measured several times, the different procedures being called in different orders. For each value of N, the respective timings are averaged and divided by the averaged time used by the Haaland approximation (the latter having thus a relative computational time equal to one for all N). The result of this test are displayed on the figure 3. (The whole procedure was repeated several times and the corresponding graphics were similar.)

For small N, say $N < 2000$, the computations are so fast that the function cputime cannot measure the times. For larger values of N, we can see on the figure 3 that the computations of λ_1 are a bit faster than the Haaland formula, while the computations of λ_2 are a bit slower, in average. This is in agreement with the number of evaluations of transcendent functions needed for each approximations, as mentioned above.

These relative times may vary depending on the system, hardware and software, but we believe that the results would not be fundamentally different from the ones obtained here. The important result is that the procedure presented in this paper is comparable, in term of speed, to simplified formulae such as the Haaland approximation. The new procedure being much more accurate, it should thus be preferred.

7 Conclusion

We have introduced a simple, fast, accurate and robust algorithm for solving the Colebrook equation. The formula used is the same for the whole range of the parameters. The accuracy is around machine double precision (around sixteen digits). The present algorithm is more efficient than the solution of the Colebrook equation expressed in term of the Lambert W-function and than simple approximations, such as the Haaland formula.

We have also provided routines in MATLAB and FORTRAN for its practical use. The algorithm is so simple that it can easily be implemented in any other language and can be adapted to any variant of the Colebrook equation.

To derive the algorithm, we introduced two special functions: the ω- and ϖ-functions. These functions could also be useful in other contexts than the Colebrook equation. The efficient algorithms introduced in this paper for their numerical computation could then be used, perhaps with some modifications of the initial guesses, specially if high accuracy is needed.

A High-order schemes for solving a single nonlinear equation

Let be a single nonlinear equation $f(y) = 0$, where f is a sufficiently regular given function and y is unknown. This equation can be solved iteratively via the numerical scheme [7]

$$y_{j+1} \;=\; y_j \;+\; (p+1)\left[\frac{(1/f)^{(p)}}{(1/f)^{(p+1)}}\right]_{y=y_j}, \tag{21}$$

where p is a non-negative integer and $F^{(p)}$ denotes the p-th derivative of F with $F^{(0)} = F$.

The scheme (21) is of order $p+2$, meaning that the number of exact digits is roughly multiplied by $p+2$ after each iteration (when the procedure converges, of course). For $p = 0$ and $p = 1$, one obtains Newton's and Halley's schemes, respectively. The scheme (14) for solving the Colebrook equation is obtained with $p = 2$ together with the function f given by the equation (11), plus some elementary algebra. Intensive tests have convinced us that it is most probably the best choice for the problem at hand here.

B MATLAB code

The MATLAB© function below is a vectorized implementation of the algorithm described in this paper. This code can also be freely downloaded [2]. We hope that the program is sufficiently documented so that one can easily test and modify it.

```
function F = colebrook(R,K)
% F = COLEBROOK(R,K) fast, accurate and robust computation of the
% Darcy-Weisbach friction factor according to the Colebrook formula:
%                                    _                    _
%       1                           |    K         2.51    |
%  ----------  =  -2 * Log10 (  ----- + ------------  )
%   sqrt(F)                     |   3.7     R * sqrt(F)   |
%                                    -                    -
% INPUT:
%   R : Reynolds' number (should be > 2300).
%   K : Equivalent sand roughness height divided by the hydraulic
%       diameter (default K=0).
%
% OUTPUT:
%   F : Friction factor.
%
% FORMAT:
%   R, K and F are either scalars or compatible arrays.
%
% ACCURACY:
%   Around machine precision for all R > 3 and for all 0 <= K,
%   i.e. in an interval exceeding all values of physical interest.
%
% EXAMPLE: F = colebrook([3e3,7e5,1e100],0.01)

% Check for errors.
if any(R(:)<=0) == 1,
```

```
    error('The Reynolds number must be positive (R>2000).');
end,
if nargin == 1,
   K = 0;
end,
if any(K(:)<0) == 1,
    error('The relative sand roughness must be non-negative.');
end,

% Initialization.
X1 = K .* R * 0.123968186335417556;      % X1 <- K * R * log(10) / 18.574.
X2 = log(R) - 0.779397488455682028;      % X2 <- log( R * log(10) / 5.02 );

% Initial guess.
F = X2 - 0.2;                            % F <- X2 - 1/5;

% First iteration.
E = ( log(X1+F) + F - X2 ) ./ ( 1 + X1 + F );
F = F - (1+X1+F+0.5*E) .* E .* (X1+F) ./ (1+X1+F+E.*(1+E/3));

% Second iteration (remove the next two lines for moderate accuracy).
E = ( log(X1+F) + F - X2 ) ./ ( 1 + X1 + F );
F = F - (1+X1+F+0.5*E) .* E .* (X1+F) ./ (1+X1+F+E.*(1+E/3));

% Finalized solution.
F = 1.151292546497022842 ./ F;           % F <- 0.5 * log(10) / F;
F = F .* F;                              % F <- Friction factor.
```

C FORTRAN code

The FORTRAN function below was written with maximum speed in mind, so some trivial arithmetic simplifications were used and there are no check for errors in the input parameters.

```
      DOUBLE PRECISION FUNCTION COLEBROOK(R,K)

C  F = COLEBROOK(R,K) computes the Darcy-Weisbach friction
C  factor according to the Colebrook-White formula.
C
C  R : Reynold's number.
C  K : Roughness height divided by the hydraulic diameter.
C  F : Friction factor.

      IMPLICIT NONE
      DOUBLE PRECISION R, K, F, E, X1, X2, T
      PARAMETER ( T = 0.33333333333333333D0 )

C Initialization.
      X1 = K * R * 0.123968186335417556D0
      X2 = LOG(R) - 0.779397488455682028D0

C Initial guess.
      F = X2 - 0.2D0

C First iteration.
      E = (LOG(X1+F)-0.2D0) / (1.0D0+X1+F)
      F = F - (1.0D0+X1+F+0.5D0*E)*E*(X1+F) / (1.0D0+X1+F+E*(1.0D0+E*T))

C Second iteration (if needed).
      IF ((X1+X2).LT.(5.7D3)) THEN
         E = (LOG(X1+F)+F-X2) / (1.0D0+X1+F)
         F = F - (1.0D0+X1+F+0.5D0*E)*E*(X1+F) / (1.0D0+X1+F+E*(1.0D0+E*T))
      ENDIF
```

```
C Finalized solution.
      F = 1.151292546497022842D0 / F
      COLEBROOK = F * F

      RETURN
      END
```

Note that, depending on the FORTRAN version and on the compiler, the command LOG may have to be replaced by DLOG to ensure that the logarithm is computed with a double-precision accuracy.

References

[1] Bendiksen, K, Malnes, D, Moe, R & Nuland, S. 1991. Dynamic two-fluid model OLGA. Theory and application. *SPE Prod. Engin.* **6**, 171-180.

[2] Clamond, D. 2008. colebrook.m. Matlab Central File Exchange.

[3] Corless, R. M., Gonnet, G. H., Hare, D. E. G., Jeffrey, D. J. & Knuth, D. E. 1996. On the Lambert W function. *Adv. Comput. Math.* **5**, 329–359.

[4] Corless, R. M., Jeffrey, D. J. & Knuth, D. E. 1997. A sequence of series for the Lambert W function. *Proc. Int. Symp. Symb. Alg. Comp., Maui, Hawaii.* ACM Press, 197–204.

[5] Getreuer, P. 2005. lambertw.m. Matlab Central File Exchange.

[6] Haaland, S. E. 1983. Simple and explicit formulas for the friction factor in turbulent pipe flow. *J. Fluids Eng.* **105**, 89–90.

[7] Householder. 1970. *The Numerical Treatment of a Single Nonlinear Equation.* McGraw-Hill.

[8] Keady, G. 1998. Colebrook–White formula for pipe flows. *J. Hydr. Engrg.* **124**, **1**, 96–97.

[9] http://www.orcca.on.ca/LambertW

[10] Romeo, E., Royo, C. & Monzón, A. 2002 Improved explicit equations for estimation of the friction factor in rough and smooth pipes. *Chem. Eng. J.* **86**, **3**, 369–374.

[11] Sonnad, J. R. & Goudar, C. T. 2004. Constraints for using Lambert W function-based explicit Colebrook–White equation. *J. Hydr. Engrg.* **130**, **9**, 929–931.

[12] Sonnad, J. R. & Goudar, C. T. 2007. Explicit reformulation of the Colebrook–White equation for turbulent flow friction factor calculation. *Ind. Eng. Chem. Res.* **46**, 2593–2600.

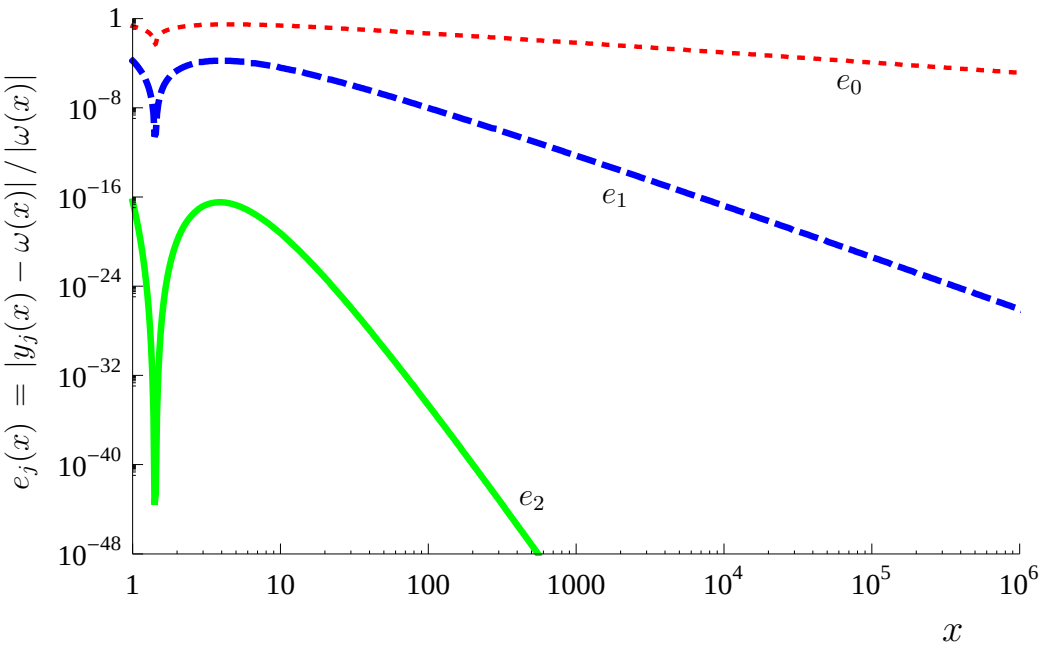

Figure 1: *Relative errors e_j of the ω-function computed via the iterations (14).*

Dotted red line: e_0; Dashed blue line: e_1; Solid green line: e_2.

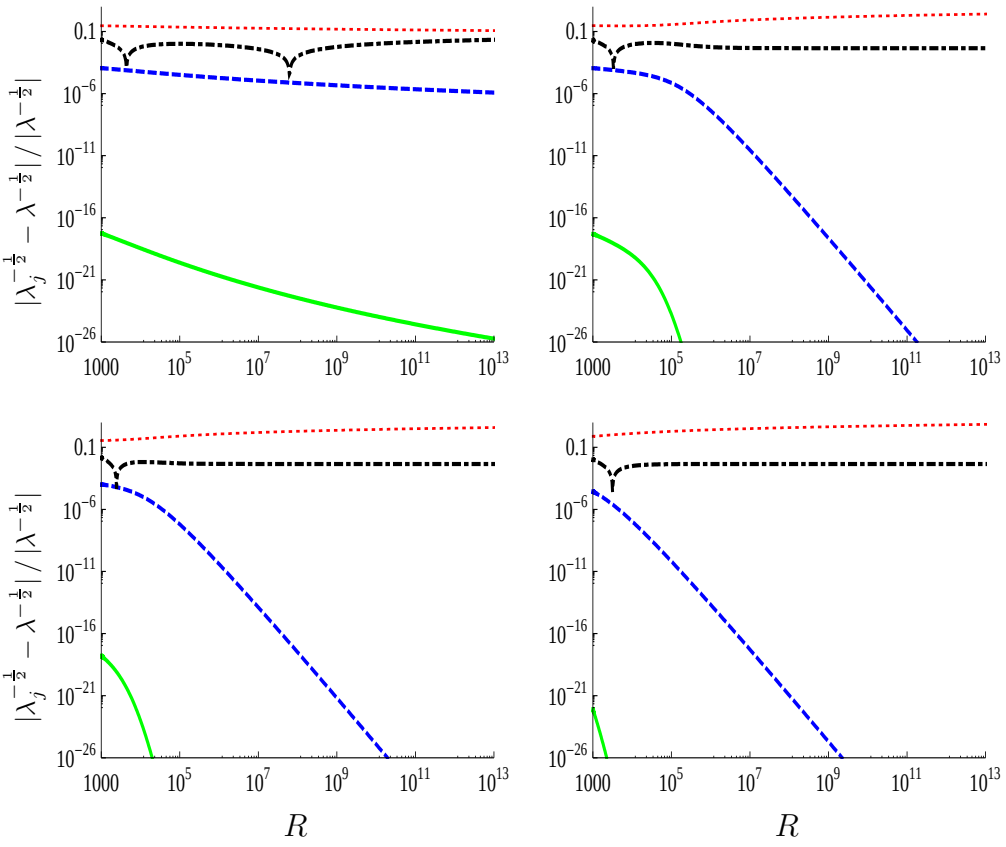

Figure 2: *Relative errors of* $\lambda^{-\frac{1}{2}}$*, computed via the iterations (17) and the Haaland formula (4), as functions of the Reynolds number R. Upper-left:* $K = 0$*; Upper-right:* $K = 10^{-3}$*; Lower-left:* $K = 10^{-2}$*; Lower-right:* $K = 10^{-1}$*.*

Dotted red line: $\lambda_0^{-\frac{1}{2}}$; Dashed blue line: $\lambda_1^{-\frac{1}{2}}$; Solid green line: $\lambda_2^{-\frac{1}{2}}$;
Dashed-dotted black line $\lambda_{\mathrm{H}}^{-\frac{1}{2}}$ (Haaland's approximation).

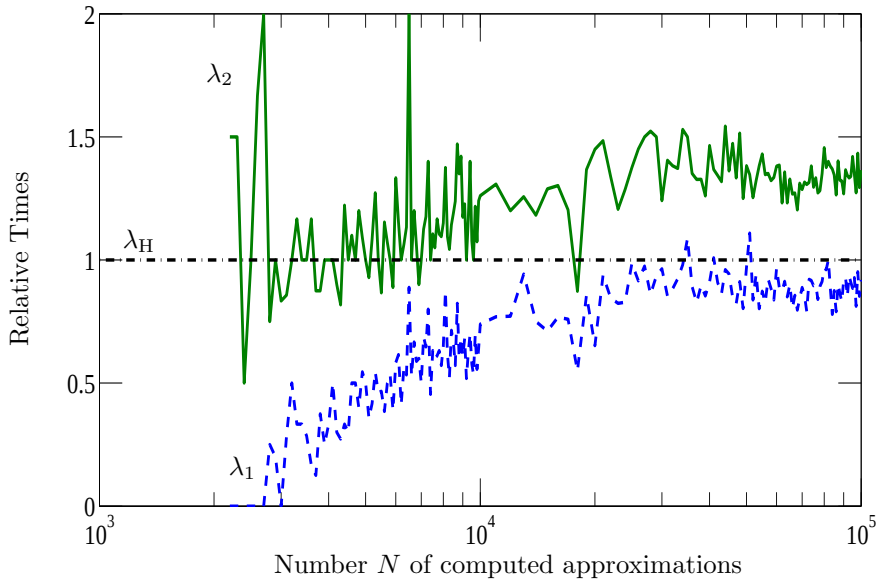

Figure 3: *Computational times of* λ_1 *(dashed blue line) and* λ_2 *(solid green line) with respect of the Haaland approximation* λ_H *(dashed-dotted black line).*