Chapter 5.
Computation of Free Surface Profile (GVF Model)
Development of a Mathematical Simulation Model for GVF

Abstract

This chapter presents an efficient algorithm for quickly and accurately solving the implicit problem of Gradually Varied Flow (GVF) profile computations based on the standard step method. Equations have been derived to directly solve for flow conditions in prismatic channels without resorting to look-up tables or interpolation routines. The algorithm is applicable to all GVF regimes. The test results show that the procedure presented here meets the requirement guaranteed convergence and very high computational efficiency (accuracy and speed-wise). The findings will definitely be helpful in contributing directly to the development of flexible computer simulation model by providing an efficient algorithm for computing GVF profiles when necessary.

*Findings of this chapter have been reported in the paper entitled Smart Computational Algorithms in Open Channels submitted by Rathore & Sen (2001b)
Introduction

Almost all major hydraulic-engineering activities in open channel flow involve the computation of Gradually Varied Flow (GVF) profiles (Subramanya, 1990; Dey, 2000). Considerable computational effort is involved in the analysis of problems such as:

(a) inundation of lands due to a dam or weir construction,

(b) determination of the effect of a hydraulic structure on the channel,

(c) estimation of the flood zone.

Because of its practical importance, the computation of GVF has been a topic of continued interest to hydraulic engineers for the last 150 years. Dupuit (1848) was perhaps the first to attempt the integration of the differential equation of GVF.

As these complicated computations can be made easier with the aid of computer, a computer-compatible mathematical simulation model for GVF has been discussed in this chapter.

The GVF Model

The standard step method is one of the most widely accepted procedures for computing open-channel water surface profiles (Paine, 1992). It is used to compute gradually varied flow water surface profiles in open channels, solving for resultant flow depth at a specified position along a channel. The procedure solves an implicit equation that requires a trial & error solution. Because this procedure also involves solution of a differential equation, a boundary condition is required. When computing flow conditions from one channel cross-section to another, the depth of flow must be known at one end of the channel in order to complete the computations.
Sources are available in the literature that describe the standard step method (HEC-2 Water 1982). Texts on open channel hydraulics also address this method, as well as composite hydraulic profile logic (Chow, 1959); French (WSPRO) computer model research report (Shearman et. al. 1986) discusses many of the practical issues involved in developing a computer program based on the standard step method.

Properly applied, the standard step method is an effective approach to computing water surface profiles. Programs that use the standard step method, such as WSPRO and HEC-2, are somewhat bulky to use on simple analyses. These programs, however, have a wide range of options useful in analysing natural channels, which often have irregular and complicated channel cross-sections.

Many of the more involved flow profile models require detailed cross-section data to be input in coordinate form. Natural channels are accommodated in this manner. In most urban drainage problems, where channels are prismatic, this type of input data can be burdensome. In the standard step method, the wetted perimeter and area of flow must be calculated at each iteration. For natural channels, where these relationships can be fairly complicated, a significant amount of computer time can be spent determining flow area and wetted perimeter. When dealing with prismatic channels, however, the geometric relationships are straightforward. If the solution scheme, or algorithm, is simplified and made faster, additional processing effort can be used to check engineering logic. As the level of logic checking increases, programs become smarter (Paine, 1992).

In writing computer programs, there are a number of reasons for keeping the procedural portion of the code as compact as possible. If a well behaved portion of the compact code can be developed, it is much easier to add portions of code to check logic.
If the code becomes disjointed and sprawls over several subroutines, functions, and sections, it is much more difficult to check anything. Sprawling code usually stalls performance. Careful coding on the other hand can actually facilitate input data requirements while maximizing performance.

**Numerical Analysis**

In the standard step method, the equation being solved depends upon whether the free surface elevation is to be computed in an upstream direction, as is done in sub-critical flow, or in a downstream direction, as in super-critical flow.

**Sub-Critical Flow**

The energy equation applied to sections A (downstream section) and B (upstream section) gives:

\[
Y_a \cdot a_a + z_a + \hat{S}_r \Delta x + L_{e} = Y_b \cdot a_b + z_b \]

\[
\frac{V_a^2}{2g} + \frac{V_b^2}{2g} \]

where,

- \( Y \) = flow depth;
- \( a \) = energy coefficient;
- \( V \) = mean flow velocity;
- \( z \) = channel bed elevation with respect to a horizontal datum;
- \( \hat{S}_r \) = average friction slope between sections A and B,
- \( \Delta x \) = horizontal distance between sections A and B; and,
- \( L_e \) = eddy loss
Subscripts “a” and “b” refer to section A (downstream section) and B (upstream section), respectively.

The expressions of $\hat{S}_r$ and eddy loss $L_e$ are given as:

$\hat{S}_r = 0.5(S_{fa} + S_{fb})$ ................................................................. (5.2)

$\alpha_a V_a^2 - \alpha_b V_b^2$

$L_e = \varepsilon \text{ abs} \frac{2}{2g}$ ................................................................. (5.3)

Where,

$S_r = \text{friction slope;}$ and

$\varepsilon = \text{coefficient of eddy loss.}$

Substituting equations (5.2) & (5.3) in equation (5.1), yields:

$\frac{V_b^2}{2g} \quad S_{fb} \quad \frac{V_a^2}{2g} \quad S_{fa}$

$Y_b + \alpha_b(1 + \varepsilon) - z_b - \Delta x - Y_a - \alpha_a(1 + \varepsilon) - z_a - \Delta x = 0$ ...................................................... (5.4)

The eddy loss $L_e$ is often included in the friction slope term in engineering practice by modifying the Manning coefficient. Thus, equation (5.4) reduces to:

$\frac{V_b^2}{2g} \quad S_{fb} \quad \frac{V_a^2}{2g} \quad S_{fa}$

$Y_b + \alpha_b - (Y_a + \alpha_a) - (z_a - z_b - \Delta x) = 0$ ...................................................... (5.5)

In the standard step method $\Delta x$ is established by the user and the terms inside the second parenthesis are known. The terms inside the first parenthesis are functions of $Y_b$.

Therefore, the solution of the equation is:

$f(Y_b) = 0$ ...................................................... (5.6)
Paine (1992) developed an algorithm for GVF profile computations (for prismatic channels) using Manning formula of flow resistance. Newton-Raphson technique was used for guaranteed convergence of the algorithm. Standard step functions $f(Y)$ and $P(Y)$ developed by Paine were tested for numerical convergence under a wide variety of flow conditions in subcritical, critical, horizontal, and adverse flow regimes. The Newton-Raphson technique has consistently converged.

However, there are two basic problems associated with Paine's algorithm:

(1) Manning's equation is accurate in an intermediate range of roughness ratios: it predicts unrealistically low friction and high discharge for both deep smooth and shallow rough channels, for which the friction-factor formulation would be preferred (White, 1986).

(2) Newton-Raphson method, as has already been established in the previous pages, is quite sensitive to the starting point and its rate of convergence is quite less.

(3) The algorithm is not applicable to natural streams.

It is therefore felt that the algorithm proposed by Paine can further be refined for maximizing its computational efficiency. Here, Chebyshev approximation can come to ones rescue.

In the Newton-Raphson technique, a value of $Y_b \approx \xi_k$ is used in equation (5.6) as a first trial solution, and then an improved value of $Y_b$ is estimated as $\xi_{k+1} = \xi_k - \Delta \xi_k$.

In the Chebyshev approximation, however, a value of $Y_b \approx \xi_k$ is used in equation (5.6) as a first trial solution, and then an improved value of $Y_b$ is estimated as $\xi_{k+1} = \xi_k$.
- \Delta \xi_k - \Delta^* \xi_k. The expressions of \Delta \xi_k and \Delta^* \xi_k obtained from the Taylor series are as follows.

\[ \Delta \xi_k = f(\xi_k)/f'(\xi_k) \] .......................................................... (5.7)

\[ \Delta^* \xi_k = 0.5f^2(\xi_k)f'(\xi_k)/f'^3(\xi_k) \] .......................................................... (5.8)

Where,

\[ f(\xi_k) = f(Y_b) \text{ at } Y_b = \xi_k; \] and

\[ f'(\xi_k) = f'(Y_b) \text{ at } Y_b = \xi_k; \] and

\[ f''(\xi_k) = f''(Y_b) \text{ at } Y_b = \xi_k. \]

Therefore, solutions to the equation require expressions of \( f(Y_b) \) and \( f''(Y_b) \).

Differentiating \( f(Y_b) \) once and then a second time, yields:

\[ f'(Y_b) = v_b^2 \Delta x \]
\[ gD_b \]

\[ f'(Y_b) = 1 - a_b - \frac{S'_{fb}}{2} \] .......................................................... (5.9)

\[ f''(Y_b) = \frac{v_b^2}{2} \Delta x \]
\[ gD_b^2 \]

\[ f''(Y_b) = a_b - \frac{D_b}{T_b} - \frac{T''_{fb}}{2} \] .......................................................... (5.10)

Where,

\[ D = \text{hydraulic depth } (A/T); \]

\[ A = \text{flow area}; \] and

\[ T = \text{Top width of flow}. \]

In equations (5.9) & (5.10), the terms \( S'_{fb} \) and \( S''_{fb} \) are determined using the flow-resistance equation of Manning, Chezy, or Colebrook-White.
Rearranging the Manning equation, $S_{fb}$ is given as:

\[ S_{fb} = n^2 V_b^2 / R_b^{4/3} \] ..................................................(5.11)

Where,

$n =$ Manning coefficient ;

$R =$ hydraulic radius ($A/P$); and

$P =$ wetted perimeter.

By differentiating, one gets:

\[ S'_{fb} = \frac{2}{3} \frac{5}{D_b} \frac{2}{P_b} S_{fb} \] ..................................................(5.12)

\[ S''_{fb} = \frac{4}{3} \frac{65}{D_b^2} \frac{5}{2A_b} \frac{20}{3D_b P_b} \frac{1}{3P_b^2} \frac{1}{P_b} S_{fb} \] ...........................................(5.13)

Chezy Equation:

Using the Chezy equation, $S_{fb}$ can be expressed as:

\[ S_{fb} = \frac{V_b^2}{C R_b} \] ..................................................(5.14)

Where,

$C =$ Chezy coefficient.

Differentiating (5.14) with respect to $V_b$ once and then a second time, one can write:
Colebrook-White equation

The advantage of using the Colebrook–White equation of flow-resistance is that it covers the regime of flow from the hydraulic smooth zone to the hydraulic rough zone. The Colebrook–White equation is given below:

\[

c_s = - \frac{3}{D_b} \left( \frac{1}{P_b} \right) - \frac{1}{P_b} \left( 1 - \frac{T_p}{P_b} \right) \left( \frac{1}{P_b} + \frac{1}{P''_b} \right) \left( \frac{1}{P_b} + \frac{1}{P''_b} \right)^{-1} \quad (5.15)
\]

\[

c_s = \frac{12}{D_b^2} \left( \frac{3}{A_b} - \frac{6}{D_b P_b} + \frac{1}{P''_b} \right) \left( \frac{1}{P_b} + \frac{1}{P''_b} \right)^{-1} \quad (5.16)
\]

where,

\[\lambda = \text{friction factor; \quad } k_s = \text{Nikuradse equivalent sand roughness; \quad and } \nu = \text{kinematic viscosity.}\]

The friction factor can be extracted from the equation of bed shear stress:

\[\tau_b = 0.125 \lambda_b \rho V_b^2 = \gamma R_b S_{fb}\]

where,

\[\tau = \text{bed shear stress; \quad } \rho = \text{mass density of fluid; \quad and } \gamma = \text{specific weight of fluid.}\]
Using equations (5.17) & (5.18), one can write:

\[
V_b = -2.457\sqrt{gR_b S_{fb}} \ln\left(\frac{ks}{14.8R_b} + \frac{0.222v}{R_b \sqrt{gR_b S_{fb}}}\right) \tag{5.18}
\]

The terms \(S'_{fb}\) and \(S''_{fb}\) are obtained by differentiating the above equation once and then a second time with respect to \(Y_b\). One gets

\[
S'_{fb} = -\frac{1}{V_b + \beta_3} \frac{1}{D_b + \beta_3} \frac{1}{D_b} \frac{1}{P_b} \left[(\beta_3 + 2\beta_1) + V_b \left(P' - \frac{V_b}{P_b}\right)\right] \tag{5.20}
\]

\[
S''_{fb} = -\frac{1}{V_b + \beta_3} \frac{1}{S^2_{fb}} \frac{1}{S_{fb}} \frac{1}{d_b} \left[1 - (\beta_3 + 2\beta_1) + (\beta_3 + 2\beta_1)\right] \tag{5.21}
\]

Where,

\[
\beta_1 = 2.457 (gR_b S_{fb})^{0.5},
\]

\[
\beta_2 = \frac{k_s}{(14.8R_b)} + 0.545v / (\beta_1 R_b);
\]

\[
\beta_3 = 0.545v / (\beta_2 R_b),
\]
\[
\beta'_{1} = \frac{\beta_{1}}{2} \left( \frac{1}{A_{b}} - \frac{1}{S_{fb}} \right) (5.22)
\]

\[
\beta'_{2} = -\beta_{2} \left( \frac{1}{D_{b}} + \frac{1}{\beta_{1}^{2}} - \frac{1}{\beta_{1}^{2} P_{b}} \right) (5.23)
\]

\[
\beta'_{3} = -\beta_{3} \left( \frac{1}{D_{b}} + \frac{1}{\beta_{2}} - \frac{1}{P_{b}} \right) (5.24)
\]

The Colebrook-White equation is an implicit equation of \( S_{fb} \). Therefore, the Chebyshev approximation can

\[
f(S_{fb}) = V_{b} + \beta_{1} \ln \beta_{2} \quad \text{..........................................................} (5.25)
\]

For given flow and channel conditions, the values of \( Y_{b}, R_{b} \) and \( V_{b} \) are constant.

So, for the Chebyshev.

\[
f'(S_{fb}) = 0.5 \left( \beta_{1} \ln \beta_{2} - \beta_{2} \right) S \quad \text{..........................................................} (5.26)
\]

\[
f''(S_{fb}) = 0.5 \left( \beta_{2} \ln \beta_{2} - \beta_{2} + 0.5 \beta_{2}^{2} \right) S_{fb}^{2} \quad \text{..........................................................} (5.27)
\]

The first trial solution of \( S_{fb} \) can be obtained using the explicit equation of Jain (1976) as given below:

\[
V_{b} = 2828 \sqrt{gR_{b}S_{fb} \left[ 1.14 - 0.86 \ln \left( \frac{1}{R_{b}^{0.9}} \right) \right]} \quad \text{..................................................} (5.28)
\]

\[
4R_{b} \quad R_{b}^{0.9} \quad V_{b}^{0.9}
\]

- 192 -
However, the Von Karman equation of flow resistance, obtained using \( \nu = 0 \) in equation (5.17) on the other hand, the complete smooth regime is seldom obtained in practice. The Nikuradse equation of flow resistance found using \( K_s = 0 \) in equation (5.17) can be applicable in a smooth regime.

**Super-Critical Flow**

Applying the energy equation between sections B (downstream section) and A (upstream section), yields:

\[
R(Y_b) = \left( \frac{V_b^2}{2g} + \frac{S_{fb}}{2} - \frac{\Delta x}{2g} \right) - \left( \frac{V_a^2}{2g} + z_a - \frac{a_b}{2} - \frac{\Delta x}{2g} \right) = 0 \quad (5.29)
\]

and differentiating \( R(Y_b) \) once and then a second time, one gets:

\[
f'(Y_b) = \frac{V_b^2}{gD_b} \frac{\Delta x}{2} + \frac{S'_{fb}}{2} \quad (5.30)
\]

\[
f''(Y_b) = a_b \frac{V_b^2}{gD_b^2} \frac{\Delta x}{2T_b} + \frac{S''_{fb}}{2} \quad (5.31)
\]

The terms \( S'_{fb} \) and \( S''_{fb} \) are determined as before.

**Common Channel Sections**

The channel parameters and their differential forms of some commonly available channels in practice, required for the direct use in the developed equations of the preceding section, are furnished in Table (5.1).
The two extreme cases of trapezoidal channel section are rectangular \((Z_1 = Z_2 = 0)\) and triangular \((B = 0)\) sections. The proposed method can also be applicable to natural channels, if \(A\), \(P\) and \(T\) of a natural channel are expressed as functions of \(Y\) using appropriate polynomial fittings.

**Summary & Conclusions**

A modified algorithm has been presented for solution of GVF profile problems based on the standard step method. Equations and derivatives have been derived to directly solve for flow conditions in prismatic channels without resorting to look-up tables or interpolation routines. As the Manning’s equation is accurate in an intermediate range of roughness ratios only; it predicts unrealistically low friction and high discharge for both deep smooth and shallow rough channels. Therefore, the requisite derivatives have been developed for Colebrook-White equation as well. Chebyshev approximation was used for faster convergence of the algorithm instead of Newton-Raphson technique. The algorithm is applicable to all GVF regimes. Moreover, it is reasonable to assume that the algorithm is sufficiently fast to be used in a robust hydraulic routing model.

There are a number of physical conditions that may preclude successful computation of a flow profile using the standard step method. For example, the length of flow profile may be shorter than the length of the channel reach, or a hydraulic jump may occur on the reach. While using this algorithm in computer models, checks can be built into the computer code that leads to a controlled termination of the program run in all these cases. Strictly speaking, these conditions are not related to convergence.

* * *
TABLE (5.1). Parameters of Different Channel Sections

<table>
<thead>
<tr>
<th>A</th>
<th>P</th>
<th>T</th>
<th>( \frac{dP}{dY} )</th>
<th>( \frac{d^2P}{dY^2} )</th>
<th>( \frac{dT}{dY} )</th>
</tr>
</thead>
</table>

**Trapezoidal Channel**

\[
Y \left[ B + \frac{Y}{2} (Z_1 + Z_2) \right] \quad B + Y \left( \sqrt{1 + Z_1^2} \right) \quad B + Y (Z_1 + Z_2) \quad \sqrt{1 + Z_1^2} \quad O \quad Z_1 + Z_2
\]

**Circular Channels**

\[
0.25d^2 \left[ \cos^{-1} \left( 1 - \frac{2Y}{d} \right) \right] \quad d \cos^{-1} \left( 1 - \frac{2Y}{d} \right) \quad 2d - \left( 1 - \frac{2Y}{d} \right) \quad \sqrt{Y(d-Y)} \quad 2|Y(d-Y)|^{1/3} \quad \sqrt{Y(d-Y)}
\]

**Exponential Channels**

\[
\zeta Y^n \quad 2Y \left( 1 + \zeta^2 \right) \quad \eta \zeta Y^{n-1} \quad 2Y \left( 1 + \zeta^2 \right) \quad O \quad \eta(\eta-1) \zeta \gamma^{n-2}
\]

*Note: For exponential channels (Dey, 1998), the channel profile can be expressed as \( Y = KZ^p \), where \( Y = \) vertical axis, \( Z = \) horizontal axis, \( K = \) coefficient; \( p = \) exponent; \( \zeta = 2K^n (1 - p) (1 - 2p); \) and \( \eta = (1 - p)^p. \)"
Closure
CLOSURE

It is expected that the various computations of constantly recurring problems in open channel hydraulics would greatly be simplified by the proposed (or modified) computational algorithms. As such, special attention has been paid towards the simplicity, clock-time efficiency, and practical utility of the proposed algorithms by applying the techniques of statistical and applied mathematics so as to break the barriers of stultification in the adaptability of available implicit solutions in computer aided designs. Every attempt, as such, has been made to present this dissertation as an acceptable, referable, and useful document for the hydraulic engineers.

The work may further be extended by other researchers as there are many more such problems in open channel hydraulics which are still awaiting their computational simplifications.

• • • • • •• • •

- 196 -
Annexure
ANNEXURE I

Easa’s, Pillai’s, & Shirley’s Iterative Solutions for the Computation of Normal Depth in Trapezoidal Channels

(A) Easa (1992) presented the following iterative equation:

\[ W = \frac{f(Y)}{B + C_1 Y} \] \hspace{1cm} (A-1)*

Where,

\[ W = (K_2 + y) K_1 f^{1.3} \] (for the Chezy equation)

Or,

\[ W = (K_2 + y) K_2 f^{1.5} \] (for the Manning equation)

Here,

\[ K_1 = C_2 \frac{Q}{\sqrt{S_0}} \]
\[ K_2 = C_2 \frac{n Q}{\sqrt{S_0}} \]
\[ K_3 = \frac{B}{C_2} \]

* Equation (A-1) can also be used for rectangular channels by substituting:

\[ C_2 = 2, \text{ and } C_1 = 0 \]
(B) Pillai et al. (1989) presented for the Manning equation the following form of
iterative algorithm:

\[
 f(Y) = \frac{K_3 (K_1 + K_2 Y)^{0.40}}{K_1 + Y} \quad \text{(A-2)}
\]

Here,

\[
 K_1 = \frac{B}{C_1}
\]

\[
 K_2 = \frac{C_2}{C_1}
\]

\[
 K_3 = \frac{(NQ)}{(C_1 \sqrt{S_0})^{0.60}}
\]

(C) Shirley et al. (1991) presented the requisite algorithm in the following form
using the Chezy equation:

\[
 f(Y) = \frac{W}{K_1 - (K_2 + W)^{1/3}} \quad \text{(A-3)}
\]

where, \( W = [(K_3 - Y) K_5]^{1.3} \)

\[
 K_1 = \frac{B}{2C_1}, \quad K_2 = K_3^{1/3}, \quad K_3 = \frac{B}{C_2}
\]

and,
For the Manning equation:

\[ W = \left[ (K_3 + Y) K_4 \right]^{2/5} \]

Where,

\[ K_3 = \left( \frac{C_2}{C_1 S_0} \right) \left( \frac{Q}{C_1 C} \right)^2 \]

\[ K_4 = \left( \frac{C_2}{C_1} \right) \left( \frac{nQ}{C_1 S_0^{1/2}} \right)^{3/2} \]

* * *
ANNEXURE II

Saatci’s Equation for the Computation of Normal Depth in Circular Channels

Saatci (1990) has given an interesting equation for flow-depth calculations in partially-filled circular pipes which, as such, eliminates the need for iterative methods or the use of nomographs and fits with enough accuracy to the results obtained using numerical techniques or tables. However, this equation is valid only in the range $0^\circ < \phi \leq 265^\circ$. In the range $265^\circ < \phi < 302^\circ$, yet another equation is required to be used. For the range $\phi > 302^\circ$, no equation has been proposed by Saatci.

For,

\[ K = \frac{Q \cdot n \cdot d^3}{S^{1/2}} = \frac{1}{\pi} \left[ 1 - \frac{2\phi}{3\pi} \right] \left[ 1 - \left( 1 - \frac{2\phi}{3\pi} \right)^2 \right]^2 \] ..........................(AII-1)

it was found by Saatci (1990) that conveyance (K) values, as calculated from eq. (AII-1) in the range $\phi = 265^\circ$, can be used to calculate $\phi$ (in radians) with a reasonable degree of accuracy using the following explicit relationship:

\[ \phi = \frac{3\pi}{2} \sqrt{1 - \sqrt{1 - \sqrt{K}}} \] ..........................(AII-2)

For the range $265^\circ < \phi < 302^\circ$, Saatci (1992) has given yet another equation:

\[ \phi = 302 \left[ 1 - \sqrt{0.2765 - 0.824K} \right] \] ..........................(AII-3)

in which $0.31758 < K < 0.33528$
Now, from the known value of $\phi$ from equations (AII-2) or (AII-3), $Y/d$ can be directly calculated from the equation:

$$\frac{Y}{d} = \frac{1}{2} \frac{\phi}{\cos(\frac{\phi}{2})} \quad \text{(AII-4)}$$

Equation (AII-1) can easily be modified to fit observed hydraulic elements if conservative design is preferred. It can give a reasonably good fit to Escritt (1984) formula if the coefficient $(1/\pi)$ is replaced by 0.275.

$$K = 0.275 \left[ 1 - \left(1 - (\frac{2\phi}{3\pi})^2\right)^2 \right] \quad \text{(AII-5)}$$
ANNEXURE III  
Newton-Raphson Method

Let "a" be the given (first) approximation to a root of the equation f(x) = 0 and suppose (a + h) is its closer approximation.

Since f(x) = 0 is approximately satisfied by x = a + h,

\[ f(a + h) = 0 \] (approximately)

Also by Taylor's theorem,

\[ f(a + h) = f(a) + h f'(a) + \frac{h^2}{2!} f''(a) + \ldots = 0 \]

Since h is small, neglecting squares and higher powers, one gets:

\[ f(a) + h f'(a) = 0 \]

or,

\[ h = -\frac{f(a)}{f'(a)} \]

Hence,

\[ x = a - \frac{f(a)}{f'(a)} \]

is the second approximation to the root.

Similarly, taking b = a + h as the new approximate value, the third approximation to the root = b - f(a)/f'(a).

Proceeding in this way, a root of f(x) = 0 can be found to a desired degree of accuracy.

Limitations:

1. This method is not suitable when f'(a) is very small.

2. Sometimes it so happens that the second approximation to the root may even be worse than the first approximation. The behaviour of f(a) and f'(a) must be investigated before using this method.

3. The method is sensitive to the starting point.
ANNEXURE IV

An Iterative Algorithm for the Computation of Critical depth in Trapezoidal Channels as Proposed by Pillai et. al. (1989)

Pillai et. al. (1989) have presented the fastest known algorithm as hereunder:

\[ Y_c = \sqrt{\left[ K_1^2 + K_2 (Y_c + K_1)^{1/3} \right]} - K_1 \]  
(AIV-1)

Where,

\[ K_1 = \frac{B}{2C_1}, \text{ and} \]
\[ K_2 = \left( \frac{2Q^2}{gZ^2} \right)^{1/3} \]

Normalised version of equation (AIV-1) can also be presented as hereunder:

\[ \frac{Y_c}{B} = \sqrt{\left[ K_1^2 + K_2 \left( \frac{Y_c}{B} + K_1 \right)^{1/3} \right]} - K_1 \]  
(AIV-2)

Where,

\[ Y_c = \frac{Y_c}{B} \]
\[ K_1 = \frac{1}{2C_1}, \text{ and} \]
\[ K_2 = \left( \frac{2Q^2}{gZ^2B^5} \right)^{1/3} \]
ANNEXURE V

Source Code of the Computer Program (Christened as Regress.C by the Author) used for the Development of Regression Models in Chapter 4

```c
/* ****** REGRESSION ANALYSIS AND PLOTTING OF GRAPH ******
 * WITH DIFFERENT GRAPH FACILITIES AND  *
 * DESIRED DEGREE OF REGRESSION RELATIONSHIPS *
 * FOR DEVELOPING THE EXPLICIT SOLUTIONS  *
 * OF IMPLICIT HYDRAULIC EQUATIONS  *
 */

#include <stdio.h>
#include <math.h>
#include <graphics.h>

main()
{
    float a[50][50], b[50], a1[50][50], b1[50], a2[50][50], dx[50], dy[50],
        b2[50], x[50], y[50], x1[50], y1[50], r[50][50], s[50], xx[50],
        yy[50], cof[50], ry[50], e, eee, error, sy, id, il, pererr[50],
        ddy, per, err[50], fk, xg[50], yg[50], yyy[50], ygg[50];

    int i, j, k, l, n, m, mm, gd, gm, gr;
    FILE *fp1, *fp2;

    fp1 = fopen("regr1.i","r");
    fp2 = fopen("regr1.o","w");
    i=0;
    fprintfl(fp2,"\nwelcome to solve regression analysis\n");
    printf("\ngive the no. of data\n");
    fscanf(fp1,"%d", &nd);
    printf("\ngive all values of x\n");
    for (i=1; i<=nd; i++) {
        fscanf(fp1,"%f", &xr[i]);
        xr[i] = xr[i];
    }

    /* entering " y " data ****** */
    printf("\ngive the degree of regression desired\n");
    scanf("%d", &n);

    /* entering " x " data ****** */
    printf("\ngive all values of x\n");
    for (i=1; i<=n; i++) {
        fscanf(fp1,"%f", &x1[i]);
        x1[i] = x1[i];
    }

    /* entering " y " data ****** */
    yy=0;
```
printf("give all values of y ");
for (i=1; i<=nd; i++) {
    fscanf(fp1,"%f", &yr[i]),
}

/* entering graph type */

graph: printf(" 1. X Vs. Y \n");
printf(" 2. log X Vs. Y \n");
printf(" 3. X Vs. log Y\n");
printf(" 4. log X Vs. log Y\n");
printf(" \n\nGive the graph no. as given above : ");
scanf("%d", &gr);
if((gr<0) || (gr>4)) goto graph;

switch (gr) {

    case 1 : for (i=1; i<=nd; i++) {
        x[i] = xr[i];
        y[i] = yr[i];
    }
    break;

    case 2 : for (i=1; i<=nd; i++) {
        x[i] = log(xr[i]);
        y[i] = yr[i];
    }
    break;

    case 3 : for (i=1; i<=nd; i++) {
        x[i] = xr[i];
        y[i] = log(yr[i]);
    }
    break;

    case 4 : for (i=1; i<=nd; i++) {
        x[i] = log(xr[i]);
        y[i] = log(yr[i]);
    }
    break;

}

for (i=1; i<=nd; i++) {
    yy = yy + y[i];
}

nn=2*n-2;
for (k=1; k<=nn; k++) {
    xx[k]=0.0;
    for (i=1; i<=nd; i++)

    - 205 -
\[ xx[k] = xx[k] + (\text{pow}(x[i], k)) \]

for (k=2; k<=n; k++) {
    xy[k-1]=0;
    for (i=1; i<=nd; i++) {
        xy[k-1]=xy[k-1]+(\text{pow}(x[i],(k-1)))*y[i];
    }
}

xx [0] = nd;
xy [0] = yy;

for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++) {
        a[i][j]=xx[i+j-2],
    }
    b[i]=xy[i-1];
}

for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++) {
        r[i][j]=a[i][j];
    }
    s[i]=b[i];
}

fprintf(fp2,"\n");
/*
for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++) {
        fprintf(fp2,"%f ",a[i][j]);
    }
    fprintf(fp2,"%f ",b[i]);
    fprintf(fp2,"\n");
}
*/

a5000. l=!+1;
if(l == n) goto a10000;
/*
fprintf(fp2,"\nIteration no. %d\n", l),
fprintf(fp2,"---------\n").
*/

- 206 -
for (i=1; i<=n; i++) {
    if(a[i][l] == 0.) b1[i]=b[i];
    if(a[i][l] == 0.) goto a33;
    b1[i]=b[i]/a[i][l];
    a33:
        for (j=1; j<=n; j++) {
            if(a[i][j] == 0.) a1[i][j]=a[i][j];
            if(a[i][j] == 0.) continue;
            a1[i][j]=a[i][j]/a[i][l];
        }
    /* fprintf(fp2, "%f ",a1[i][j]); */
}
/*
    fprintf(fp2,"%f ",b1[i]);
    fprintf(fp2,"\n");
*/
}
/*
fprintf(fp2,"\n");
*/
for (i=1+1; i<=n; i++) {
    b2[i]=b1[i]-b1[1];
    for (j=1; j<=n; j++) {
        if(a1[i][j] == 0.) {
            a2[i][j]=a1[i][j];
            b2[i]=b1[i];
            continue;
        }
        a2[i][j]=a1[i][j]-a1[1][j];
    }
}
    b2[1]=b1[1];
    for (j=1; j<=n; j++) {
        a2[1][j]=a1[1][j];
    }
/*
for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++) {
        fprintf(fp2,"%f ",a2[i][j]);
    }
    fprintf(fp2,",%f ",b2[i]);
    fprintf(fp2,"\n");
}*/
    fprintf(fp2,"\n");
for (i=1; i<=n; i++) {
    b[i]=b2[i];
    for (j=1; j<=n; j++) {
        a[i][j]=a2[i][j];
    }
}
goto a5000;

a10000: for (i=1; i<=n; i++) {
    m=n-i+1;
    e=0.0;
    for (j=m+1; j<=n; j++) {
        if (i==0) goto a27;
        e=e+a2[m][j]*cofUj;
    }
a27:     cof[m]=(b2[m]-e)/a2[m][m];
}
printf("n");
for (mm=1; mm<=n; mm++) {
    fprintf(fp2,"COF %d = %f
", mm, cof[mm]);
    printf("COF %d = %f
", mm, cof[mm]);
}

/* ************ justification of answers ************ */
    fprintf(fp2,"n justification of answers \n");
    fprintf(fp2," ------------------------ 
");
for (i=1; i<=n; i++) {
    eee = 0.0;
    for (j=1; j<=n; j++) {
        eee=eee+r[i][j]*cofUj;
    }
    fprintf(fp2,"Eq. no. %d l.h.s. = %f   r.h.s = %f\n", i, eee, s[i]);
}

/* ************ comparison of actual and regression " y " values ************ */

fprintf(fp2," x y r y err % error\n"),
fprintf(fp2," comparison of actual & regression values y values\n"),
fprintf(fp2," ------------------------------------------\n"),
fprintf(fp2," x y r y err % error\n"),

- 208 -
fprintf(fp2,"--------- --------- --------- ---------------------
"),

for (i=1; i<=nd; i++) {
    ry[i]=cof[1];
    for (k=2; k<=n; k++) {
        ry[i] = ry[i] + ( cof[k] * ( pow (x[i],(k-1))));
    }
}

/* Converting x and y into originals */

switch (gr) {
    case 1: for (i=1; i<=nd; i++) {
        rx[i] = x[i];
        rry[i] = ry[i];
    } break;
    case 2: for (i=1; i<=nd; i++) {
        rx[i] = exp(x[i]);
        rry[i] = ry[i];
    } break;
    case 3: for (i=1; i<=nd; i++) {
        rx[i] = x[i];
        rry[i] = exp(ry[i]);
    } break;
    case 4: for (i=1; i<=nd; i++) {
        rx[i] = exp(x[i]);
        rry[i] = exp(ry[i]);
    } break;
}

/**************** error & curve fitting calculation **************/

error=0.;

for (i=1; i<=nd; i++) {
    err[i]=rry[i]-yr[i];
    pererr[i] = err[i] * 100 / yr[i];
}

fprintf(fp2,"%f  %f  %f  %f  %s  %f  %s  %f  %d  %f  %f  %s  %f  %f  
", xr[i], yr[i], rry[i], err[i], pererr[i],
    if(err[i] < 0.) err[i]=-err[i];
    if(ry[i] < 0.) yr[i]=-yr[i];
    error+=error=err[i];

- 209 -
sy = sy + yr[i];

\/* ******************* % fitting ******************* */

ddy = sy - error;
per = (ddy * 100) / sy;
fprintf(fp2, "\n");
fprintf(fp2, "uncurve fitting is \%f", per);
printf("%f
", per);

fprintf("\n\n");
fprintf(fp2, "\n\n");
gd = DETECT;
initgraph (&gd, &gm, "");
for (i = 1; i <= nd; i++) {
    yyy[i] = 100 - y[i];
    circle (x[i], yyy[i], 2);
}
for (i = 1; i <= nd; i++) {
    if (i == nd) il = x[nd];
}
id = il / 50.0;

for (i = 0; i <= 50; i++) {
    xg[i] = id * i;
yg[i] = cof[1];
    for (k = 2, k <= n; k++) {
        yg[i] = yg[i] + (cof[k] * (pow(xg[i], (k - 1))));
    }
ygg[i] = 100 - yg[i];
fprintf(fp2, "\n\n%f \%f\n", xg[i], yg[i]);
putpixel (xg[i], ygg[i], 5);
}
printf("\nuncurve fitting is \%f", per);

getch();
closegraph();

* * *