Universidad Nacional Autónoma de México

División de Estudios de Posgrado de la
Facultad de Ingeniería

Modelo de flujo bidimensional de fondo móvil de la bifurcación Mezcalapa Samaria Carrizal

Alejandro Mendoza Reséndiz

Tesis presentada como requisito para obtener el grado de
MAESTRO EN INGENIERÍA
(Hidráulica)

Tutor:
Dr. Jesús Gracia Sánchez

Diciembre 2004

Agradecimientos

Al Dr. Jesús Gracia por la asesoría y dedicación que me brindó para realizar este trabajo

Al Dr. Moisés Berezowsky por la asesoría proporcionada para hacer esta tesis

Al Dr. Abel Jiménez por la ayuda en las dudas que se me presentaron

A los sinodales Dr. Ramón Domínguez, Dr. Oscar Fuentes y al M.I. Víctor Franco por el tiempo que dedicaron para la revisión de este trabajo

Al Instituto de Ingeniería, por las facilidades que me dio para realizar este trabajo

Resumen

Esta tesis se desarrolló con el fin de determinar el comportamiento de la bifurcación del río Mezcalapa en los ríos Samaria y Carrizal a través de un modelo matemático. El problema en dicha zona es que con el paso del tiempo el caudal que deriva por el río Carrizal está aumentando.

Se presenta la deducción de las ecuaciones hidrodinámicas en dos dimensiones y se resuelven por medio del esquema de diferencias finitas de MacCormak. El transporte de sedimentos se calcula con la ecuacion de Engelund-Hansen.

Se simulan cuatro posibles causas que dan origen al aumento en el gasto del río Carrizal, de ellas, la única que produce dicho aumento es la condición de que existe un gradiente hidráulico más grande en el río Carrizal que en el río Samaria, esto produce que el flujo circule preferentemente por el río Carrizal y que la velocidad sobre el río Samaria dismunuya, lo que da origen a una zona de depósito en la entrada de dicho río.

Índice general

Capítulo 1

Introducción

La intención de desarrollar un modelo bidimensional con transporte de sedimentos es estudiar el comportamiento del río Mezcalapa en la bifurcación los ríos Samaria y Carrizal, ubicados en el estado de Tabasco.

En este trabajo se plantean las ecuaciones hidrodinámicas para flujo bidimensional a superficie libre interrelacionadas con ecuaciones de transporte de material de fondo, y se desarrolla una metodología para la solución del sistema formado por las dos ecuaciones dinámicas, la de continuidad del agua y la de sedimento.

En un río con fondo móvil las cotas z varían con el tiempo, dichos cambios están directamente relacionados con el transporte de sedimentos, es por ello que se utilizan ecuaciones de transporte para determinar la evolución del fondo.

El problema existente en dicha bifurcación es que con el paso del tiempo, el porcentaje del caudal proveniente del río Mezcalapa que ingresa por el Carrizal está aumentando, tal como se muestra en la figura 1.1, la cual fue tomada de la referencia 11. Esto puede deberse a la pérdida de capacidad en el río Samaria o a un aumento de capacidad del río Carrizal, o a una combinación de las dos. Lo anterior repercute en las inundaciones que se presentan en la ciudad de Villahermosa, ubicada en las orillas del río Carrizal. El incremento en el gasto de este último río puede deberse a la cantidad de sedimentos que se está depositando en la entrada del río Samaria (figura 1.2), lo cual obliga al agua a circular preferentemente por la entrada del río Carrizal.

El modelo desarrollado tiene el fin de simular el comportamiento de la bifurcación, probar en diferentes modelaciones, variando niveles de la superficie libre del agua, hidrogramas de entrada y condiciones topográficas, cuál de ellas es la que da origen a un aumento en el gasto que deriva por el río Carrizal. Uno de los objetivos principales es tratar de representar el fenómeno de depósito en la zona de la entrada al río Samaria

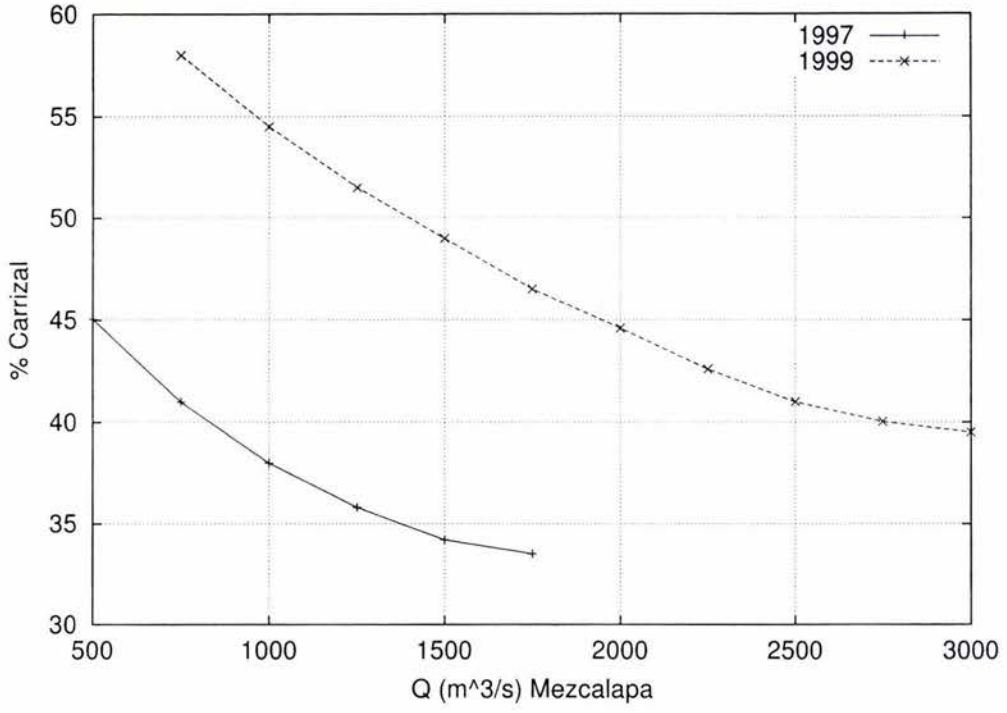


Figura 1.1: Porcentaje del gasto que deriva por el río Carrizal, del gasto proveniente del río Mezcalapa (tomada de la referencia 11)

que ocurre actualmente. Se espera que con este trabajo en el futuro podrán simularse diferentes opciones para revertir la pérdida de capacidad del río Samaria y disminuir el caudal que está derivando por el río carrizal.

En este trabajo se presentan las bases teóricas para calcular el flujo bidimensional a superficie libre, así como su implantación numérica, acopladas con las ecuaciones de transporte de sedimento, aquí se plantean para simular una bifurcación sin embargo, el modelo desarrollado puede ocuparse para cualquier otro caso, siempre que se cumplan con las condiciones descritas en los capítulos 2 y 3.

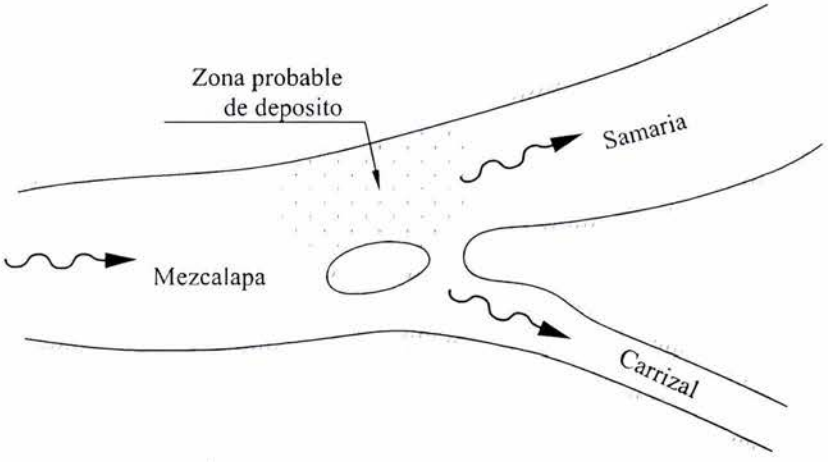


Figura 1.2: Zona de depósito en la bifurcación

Capítulo 2

Ecuaciones hidrodinámicas para flujo bidimensional

En este capítulo se deducen las ecuaciones diferenciales parciales necesarias para modelar un flujo bidimensional a superficie libre, se trata de tres ecuaciones, la primera es la de conservación de la masa y la otras dos corresponden a la conservación de momento en las direcciones de los ejes x y y.

Las ecuaciones antes mencionadas se deducen a partir del principo de conservación de propiedades físicas expresado primero como una ecuación integro diferencial, a partir de la cual se llega a las versiones diferenciales.

2.1. Hipótesis para las ecuaciones de flujo bidimensional a superficie libre

Las hipótesis para obtener de las ecuaciones hidrodinámicas, de acuerdo a la referencia 2, son las siguientes:

1. El flujo es incompresible esto es, $\rho = c$, c es constante.
2. Se supone una distribución hidrostática de presiones en la dirección vertical.
3. La pendiente del fondo es suave ($\cos\alpha_x \approx 0$ y $\cos\alpha_y \approx 0$).
4. El flujo ocurre predominantemente en un plano horizontal, y las velocidades en la dirección vertical (eje z) son muy pequeñas.
5. La fricción en el fondo puede calcularse con las ecuaciones de flujo permanente.

La deducción de la ecuación de conservación se realiza con ayuda de lo planteado en el capítulo 1 la referencia 8.

2.2. Ecuación de conservación

La cantidad total de una propiedad física Φ dentro de una masa de control (Ω_{MC}) se calcula con la integral siguiente

$$\Phi = \int_{\Omega_{MC}} \rho\phi d\Omega \tag{2.1}$$

Donde ϕ es la propiedad física analizada expresada por unidad de masa. El cambio de la propiedad física Φ en el tiempo está dada por la ecuación 2.2

$$\frac{d}{dt}\int_{\Omega_{MC}} \rho\phi d\Omega = \frac{d}{dt}\int_{\Omega_{VC}} \rho\phi d\Omega + \int_{S_{VC}} \rho\phi(\mathbf{v}\cdot\mathbf{n})dS \tag{2.2}$$

Donde

Ω_{MC}	masa de control
Ω_{VC}	volumen de control
S_{VC}	superficie del volumen de control
ρ	densidad específica del fluido
ϕ	propiedad física en estudio, expresada por unidad de masa
$\mathbf{v}$	vector de velocidad en el plano XY en la superficie de control
$\mathbf{n}$	vector normal a la superficie de control

La parte derecha de la ecuación 2.2 se obtiene al derivar Φ respecto al tiempo.

$$\frac{d\Phi}{dt} = \frac{d}{dt}\int_{\Omega_{MC}} \rho\phi d\Omega$$

De acuerdo con la referencia 8, la ecuación de conservación (2.2) puede enunciarse de la siguiente manera

El cambio en cantidad de la propiedad física Φ respecto al tiempo en una masa de control está dado por el cambio en el tiempo de la propiedad física dentro de un volumen de control (Ω_{VC}) más el flujo total de la propiedad física a través de las fronteras del volumen de control (S_{VC}) .

El último término de la ecuación 2.2 se denomina convectivo. En adelante al volumen de control Ω_{VC} se le denomina Ω y a la frontera del volumen de control S_{VC}, se le denomina S.

2.3. Ecuación de conservación de la masa

Si en la ecuación de conservación se hace $\phi = 1$, se llega a la ecuación integro diferencial 2.3

$$\frac{d}{dt}\int_{\Omega}\rho d\Omega + \int_{S}\rho(\mathbf{v}\cdot\mathbf{n})dS = \frac{d}{dt}\int_{\Omega MC}\rho d\Omega \tag{2.3}$$

Pero como el principio de conservación de la masa establece que una cantidad de masa (masa de control) dada no puede variar en el tiempo, puesto que la materia no se crea ni se destruye, entonces

$$\frac{d}{dt}\int_{\Omega_{MC}}\rho d\Omega = 0$$

por lo tanto

$$\frac{d}{dt}\int_{\Omega}\rho d\Omega + \int_{S}\rho(\mathbf{v}\cdot\mathbf{n})dS = 0 \tag{2.4}$$

La ecuación anterior establece que el cambio de la cantidad de masa dentro de un volumen de control más la cantidad de masa que sale por las fronteras debe ser igual a cero.

Por otra parte, el teorema de divergencia de Gauss permite expresar a una integral de superficie como una integral de volumen.

$$\int_{S}\rho(\mathbf{v}\cdot\mathbf{n})dS = \int_{\Omega}\nabla\cdot\mathbf{v}d\Omega$$

Entonces el principio de conservación de la masa puede escribirse de acuerdo con (2.5). Con dicha ecuación se puede obtener la versión diferencial de conservación de la masa para flujo bidimensional.

$$\frac{d}{dt}\int_{\Omega}\rho d\Omega + \int_{\Omega}\nabla\cdot\mathbf{v}d\Omega = 0 \tag{2.5}$$

En la figura 2.1 se presentan los elementos para evaluar la primera integral de la ecuación 2.5

$$\frac{d}{dt}\int_{\Omega}\rho d\Omega = \frac{d}{dt}(\rho\,\Delta x\,\Delta y\,h)$$

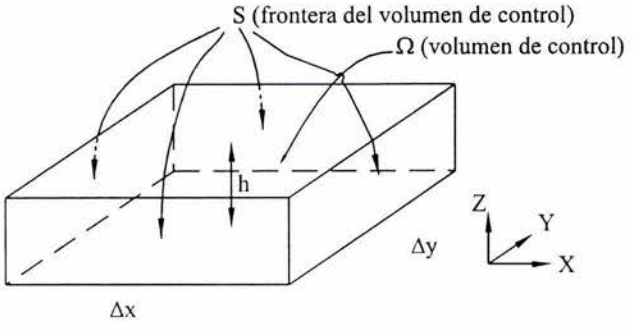


Figura 2.1: Volumen de control

para evaluar la segunda integral de (2.5)

$$\int_{\Omega} \nabla \cdot \mathbf{v} d\Omega = \nabla \cdot (\mathbf{v} \rho \Delta x \, \Delta y \, h)$$

Debido a que el término $\rho \, \Delta x \, \Delta y$ es constante, puede salir de las derivadas parciales, y sustituyendo los dos resultados anteriores se llega a la expresión diferencial.

$$\rho \, \Delta x \, \Delta y \frac{\partial h}{\partial t} + \rho \, \Delta x \, \Delta y \, \nabla \cdot (h\mathbf{v}) = 0$$

Simplificando términos

$$\frac{\partial h}{\partial t} + \nabla \cdot (h\mathbf{v}) = 0 \tag{2.6}$$

Y desarrollando el gradiente

$$\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial y} = 0 \tag{2.7}$$

La ecuación anterior es la ecuación diferencial de conservación de masa.

2.4. Ecuación de cantidad de movimiento

De acuerdo a la segunda ley de Newton, el cambio de momentum $(m\mathbf{v})$ puede calcularse como la suma de fuerzas que actúan sobre la masa del volumen de control

$$\frac{d(m\mathbf{v})}{dt} = \sum \mathbf{f} \tag{2.8}$$

donde

m	masa contenida en el volumen de control
$\sum \mathbf{f}$	suma de fuerzas que actúan en el volumen de control
$\mathbf{v}$	vector de velocidad en el plano XY en la superficie

Haciendo ahora que la propiedad física estudiada sea $\phi = \mathbf{v}$, sustituyendo en la parte derecha de la ecuación 2.2 y considerando que masa de control Ω es diferencial, y con la ecuación 2.8 se tiene

$$\frac{d}{dt}\int_{\Omega} \rho \mathbf{v} d\Omega = \sum \mathbf{f} \tag{2.9}$$

Sustituyendo el resultado anterior en la ecuación de conservación 2.2 se llega a la ecuación integro diferencial de conservación de cantidad de movimiento

$$\frac{d}{dt}\int_{\Omega} \rho \mathbf{v} d\Omega + \int_{S} \rho \mathbf{v}(\mathbf{v}\cdot\mathbf{n}) dS = \sum \mathbf{f} \tag{2.10}$$

La primera integral de la ecuación 2.10 puede desarrollarse de la siguiente manera, apoyándose en la figura 2.1 donde se muestra el volumen de control Ω dentro de un flujo bidimensional a superficie libre, como se trata de una integral de volumen entonces se puede evaluar como se muestra a continuación

$$\frac{d}{dt}\int_{\Omega} \rho \mathbf{v} d\Omega = \frac{\partial}{\partial t}(\mathbf{v}\rho \Delta x\, \Delta y\, h)$$

$$\frac{d}{dt}\int_{\Omega} \rho \mathbf{v} d\Omega = \rho \Delta x\, \Delta y \frac{\partial}{\partial t}(\mathbf{v}h) \tag{2.11}$$

donde

Δx	dimensión en la dirección del eje x del volumen de control
Δy	dimensión en la dirección del eje y del volumen de control
h	tirante en el flujo del volumen de control
$\mathbf{v}$	vector de velocidad en el plano XY en el volumen de control

Desarrollando la segunda integral de (2.10), que es el denominado término convectivo se llega a lo siguiente

$$\int_{S} \rho \mathbf{v}(\mathbf{v}\cdot\mathbf{n}) dS = \left[\int_{S} \rho u \mathbf{v}\cdot\mathbf{n} dS\right]\hat{i} + \left[\int_{S} \rho v \mathbf{v}\cdot\mathbf{n} dS\right]\hat{j}$$

y aplicando el teorema de divergencia de Gauss a la expresión anterior

$$\int_S \rho \mathbf{v}(\mathbf{v}\cdot\mathbf{n})dS = \left[\int_\Omega \nabla\cdot(\rho u\mathbf{v})d\Omega\right]\hat{i} + \left[\int_\Omega \nabla\cdot(\rho v\mathbf{v})d\Omega\right]\hat{j} \tag{2.12}$$

donde

- u componente de velocidad en la dirección x
- v componente de velocidad en la dirección y
- $\hat{i}$ vector unitario en la dirección x
- $\hat{j}$ vector unitario en la dirección x
- $\mathbf{v}$ vector de velocidad en el plano XY, $\mathbf{v} = u\hat{i} + v\hat{j}$

Las dos integrales de volumen de (2.12) pueden calcularse nuevamente con ayuda de la figura 2.1

$$\int_S \rho \mathbf{v}(\mathbf{v}\cdot\mathbf{n})dS = [\nabla\cdot(\rho u\mathbf{v}\Delta x\,\Delta y\,h)]\,\hat{i} + [\nabla\cdot(\rho v\mathbf{v}\Delta x\,\Delta y\,h)]\,\hat{j}$$

Agrupando terminos se llega a (2.13)

$$\int_S \rho \mathbf{v}(\mathbf{v}\cdot\mathbf{n})dS = \rho\Delta x\,\Delta y\left[[\nabla\cdot(uh\mathbf{v})]\hat{i} + [\nabla\cdot(vh\mathbf{v})]\hat{j}\right] \tag{2.13}$$

Las fuerzas a considerar en la ecuación 2.10 para determinar la magnitud de $\sum \mathbf{f}$ se dividen en dos grupos:

1. Fuerzas de superficie (presión, esfuerzos cortante y normal, tensión superficial, etc.)
2. Fuerzas de cuerpo (gravedad, fuerzas centrífugas y de coriolis, etc)

2.4.1. Fuerzas de superficie

Existen fuerzas que actúan sobre la superficie S que delimita el volumen de control, tales como la fricción, a estas fuerzas se les denomina *fuerzas de superficie.*

Las fuerzas de fricción se producen en el contacto del agua con el fondo del río, el esfuerzo cortante en esta área se calcula con la siguiente expresión.

$$\tau_0 = g\rho R_h S_f$$

donde:

τ_0 esfuerzo cortante en el fondo
S_f pendiente de fricción
R_h radio hidráulico para el volumen de control, en este caso $R_h = h$

La pendiente de fricción S_f es función de la velocidad, en los ejes x y y se tienen las velocidades u y v respectivamente, por lo tanto se tienen valores de pendiente de fricción en ambos ejes, S_{fx} y S_{fy} y puede expresarse como una cantidad vectorial.

$$\mathbf{S_f} = S_{fx}\hat{i} + S_{fy}\hat{j}$$

La fuerza de fricción que actúa en el fondo del cauce es igual al esfuerzo cortante multiplicado por el área A en la que actúa en el volumen de control, $A = \Delta x\, \Delta y$, por lo tanto

$$\mathbf{F_f} = -\Delta x\, \Delta y\, g\, \rho\, h\, \mathbf{S_f} \tag{2.14}$$

Tiene signo negativo porque actúa en dirección opuesta a la velocidad.

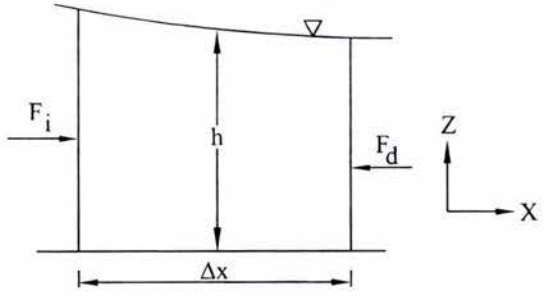


Figura 2.2: Fuerzas de presión en la dirección del eje x

El tirante h se presenta en el centro del volumen de control, como el nivel del tirante varía en la dirección del eje x, como se ve en la figura 2.2, entonces se tiene una diferencia de presiones entre las caras derecha e izquierda del volumen de control mostrado. la fuerza de presión que actúa en la frontera izquierda es

$$F_i = \frac{g\rho}{2}\left(h^2 - \frac{\partial h^2}{\partial x}\frac{\Delta x}{2}\right)\Delta y$$

y la fuerza en la frontera derecha

$$F_d = -\frac{g\rho}{2}\left(h^2 + \frac{\partial h^2}{\partial x}\frac{\Delta x}{2}\right)\Delta y$$

Sumando las dos fuerzas anteriores se tiene la fuerza de presión que actúa en el volumen de control debida a la variación del tirante en la dirección x.

$$F_x = -\frac{g\rho}{2}\frac{\partial h^2}{\partial x}\Delta x\,\Delta y$$

Haciendo un desarrollo análogo en la dirección y se obtiene

$$F_y = -\frac{g\rho}{2}\frac{\partial h^2}{\partial y}\Delta x\,\Delta y$$

Si la fuerza de presión se expresa como una cantidad vectorial se llega a lo siguiente

$$\mathbf{F_p} = F_x\hat{i} + F_y\hat{j} = -\frac{g\rho}{2}\Delta x\,\Delta y\left(\frac{\partial h^2}{\partial x}\hat{i} + \frac{\partial h^2}{\partial y}\hat{j}\right)$$

Expresando las derivadas de h^2 con respecto a x y y como un gradiente ∇h^2, se llega a la expresión para determinar las fuerzas en el volumen de control debidas a la presión.

$$\mathbf{F_p} = -\frac{g\rho}{2}\Delta x\,\Delta y\,\nabla h^2 \tag{2.15}$$

2.4.2. Fuerzas de cuerpo

Las fuerzas que actuan en cada punto del volumen de control Ω tales como el peso, se denominan *fuerzas de cuerpo.*

En las fuerzas de cuerpo se tienen las debidas al peso de la masa contenida dentro del volumen de control.

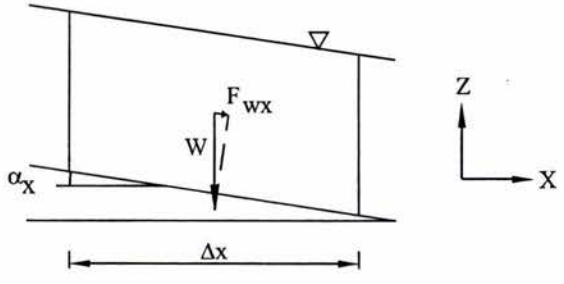


Figura 2.3: Fuerza debida al peso del volumen de contro

La fuerza que actúa en la dirección del eje x debida al peso, F_{wx}, se obtiene con ayuda de la figura 2.3, donde dicha fuerza es debida a la proyección del peso en la dirección del eje x.

$$F_{wx} = W\, sen\alpha_x$$

el peso W se obtiene de multiplicar el volumen por la densidad del fluido, $W = \Delta x\, \Delta y\, h\rho$, por otra parte, cuando α es pequeño puede considerase que $sen\alpha = tan\alpha$, y como $tan\alpha = S_0$ entonces

$$F_{wx} = \Delta x\, \Delta y\, h\, S_{0x}$$

Procediendo de manera analoga para la dirección del eje y se tiene

$$F_{wy} = \Delta x\, \Delta y\, h\, S_{0y}$$

La fuerza debida al peso que actúa en el volumen de control expresada en términos vectoriales queda

$$\mathbf{F_w} = \Delta x\, \Delta y\, h\, \rho\, g\, \left(S_{0x}\hat{i} + S_{0y}\hat{j}\right)$$

S_{0x} y S_{0y} son las pendientes del fondo en x y y respectivamente. Si la cota del fondo es z entonces las pendientes se pueden expresar como derivadas respecto a x y y

$$S_{0x} = -\frac{\partial z}{\partial x}$$

$$S_{0y} = -\frac{\partial z}{\partial y}$$

Finalmente la fuerza debida la peso queda expresada por la ecuacion 2.16

$$\mathbf{F_w} = \Delta x\, \Delta y\, h\, \rho\, g\, \nabla z \tag{2.16}$$

2.4.3. Ecuación diferencial de cantidad de movimiento

Sumando las fuerzas de fricción, presión y peso que actúan en el volumen de control se tiene

$$\sum \mathbf{f} = -\mathbf{F_f} - \mathbf{F_p} + \mathbf{F_w}$$

Sustituyendo las fuerzas dadas por las ecuaciones 2.14, 2.15 y 2.16 en la sumatoria de fuerzas se llega a la expresión 2.17

$$\sum \mathbf{f} = \Delta x\, \Delta y\, \rho\, g\, \left(-h\mathbf{S_f} - \frac{1}{2}\nabla h^2 + h\nabla z\right) \tag{2.17}$$

Sustituyendo (2.11), (2.13) y (2.17) en la ecuación de conservación de cantidad de movimiento (2.10) se tiene

$$\rho\,\Delta x\,\Delta y\frac{\partial}{\partial t}(h\mathbf{v})+\rho\,\Delta x\,\Delta y\left[[\nabla\cdot(uh\mathbf{v})]\hat{i}+[\nabla\cdot(vh\mathbf{v})]\hat{j}\right]=\Delta x\,\Delta y\,\rho\,g\left(-h\mathbf{S_f}-\frac{1}{2}\nabla h^2+h\nabla z\right)$$

simplificando términos

$$\frac{\partial}{\partial t}(h\mathbf{v})+\left[[\nabla\cdot(uh\mathbf{v})]\hat{i}+[\nabla\cdot(vh\mathbf{v})]\hat{j}\right]=g\left(-h\mathbf{S_f}-\frac{1}{2}\nabla h^2+h\nabla z\right)$$

reorganizando se llega a la ecuación diferencial parcial vectorial dada por (2.18)

$$\frac{\partial}{\partial t}(h\mathbf{v})+\left[[\nabla\cdot(uh\mathbf{v})]\hat{i}+[\nabla\cdot(vh\mathbf{v})]\hat{j}\right]+\frac{g}{2}\nabla h^2=g\,h\,(\nabla z-\mathbf{S_f}) \tag{2.18}$$

La expresión (2.18) es la ecuación de conservación de cantidad de movimiento en forma vectorial, desarrollándola se llega a las dos ecuaciones escalares (2.19), (2.20)

$$\frac{\partial hu}{\partial t}+\frac{\partial hu^2}{\partial x}+\frac{\partial huv}{\partial y}+\frac{g}{2}\frac{\partial h^2}{\partial x}=g\,h\left(-\frac{\partial z}{\partial x}-S_{fx}\right) \tag{2.19}$$

$$\frac{\partial hv}{\partial t}+\frac{\partial hv^2}{\partial y}+\frac{\partial huv}{\partial x}+\frac{g}{2}\frac{\partial h^2}{\partial y}=g\,h\left(-\frac{\partial z}{\partial y}-S_{fy}\right) \tag{2.20}$$

Las ecuaciones (2.7) (2.19) y (2.20) son un sistema hiperbólico de ecuaciones diferenciales parciales que al resolverse permiten conocer las velocidades u y v y el tirante h en un flujo bidimensional a superficie libre.

A veces es más práctico utilizar el nivel total del agua H, en lugar del tirante

$$H=h+z \tag{2.21}$$

donde

H	nivel total de la superficie libre del agua
h	tirante
z	cota del fondo

Para ello es necesario desarrollar las derivadas de h^2 respecto a x y y

$$\frac{\partial h^2}{\partial x}=2h\frac{\partial h}{\partial x} \tag{2.22}$$

sustituyendo el resultado anterior y reagrupando valores en (2.19)

$$\frac{\partial hu}{\partial t} + \frac{\partial hu^2}{\partial x} + \frac{\partial huv}{\partial y} + gh\left(\frac{\partial h}{\partial x} + \frac{\partial z}{\partial x}\right) = -g\,hS_{fx} \tag{2.23}$$

Uniendo las derivadas de h y z por derivada de H y procediendo de la misma forma para (2.20) se tienen finalmente las ecuaciones (2.24) y (2.25) de cantidad de movimiento en las direcciones de los ejes x y y respectivamente.

$$\frac{\partial hu}{\partial t} + \frac{\partial hu^2}{\partial x} + \frac{\partial huv}{\partial y} + gh\frac{\partial H}{\partial x} = -g\,hS_{fx} \tag{2.24}$$

$$\frac{\partial hv}{\partial t} + \frac{\partial hv^2}{\partial y} + \frac{\partial huv}{\partial x} + gh\frac{\partial H}{\partial y} = -g\,hS_{fy} \tag{2.25}$$

Es preferible utilizar el sistema formado por (2.7), (2.24) y (2.25), debido a que maneja el nivel total del agua, H, y los cálculos no se ven afectados si se tiene un fondo irregular. El tirante h se puede calcular a partir de H, $h = H - z$.

Finalmente el sistema a resolver para obtener la hidrodinámica del cuerpo de agua está conformado por (2.7), (2.24) y (2.25). Dicho sistema ya no es rigurosamente conservativo puesto que se abrió la derivada de h^2

Capítulo 3

Solución de las ecuaciones hidrodinámicas

En este capítulo se presenta el esquema de MacCormack para la solución de las ecuaciones hidrodinámicas de flujo bidimensional a superficie libre.

Las ecuaciones hidrodinámicas deducidas en el capítulo anterior pueden representarse como una ecuación diferencial vectorial

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ uh \\ vh \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} hu \\ hu^2 \\ hvu \end{pmatrix} + gh\frac{\partial}{\partial x}\begin{pmatrix} 0 \\ H \\ 0 \end{pmatrix} + \\ + \frac{\partial}{\partial y}\begin{pmatrix} hv \\ huv \\ hv^2 \end{pmatrix} + gh\frac{\partial}{\partial y}\begin{pmatrix} 0 \\ 0 \\ H \end{pmatrix} = \begin{pmatrix} 0 \\ -ghS_{fx} \\ -ghS_{fy} \end{pmatrix} \quad (3.1)$$

A cada término vectorial se le asignan los nombres de variables siguientes

$$\mathbf{U} = \begin{pmatrix} h \\ uh \\ vh \end{pmatrix} \quad (3.2)$$

$$\mathbf{E_1} = \begin{pmatrix} hu \\ hu^2 \\ hvu \end{pmatrix} \quad (3.3)$$

$$\mathbf{E_2} = \begin{pmatrix} 0 \\ H \\ 0 \end{pmatrix} \quad (3.4)$$

$$\mathbf{F_1} = \begin{pmatrix} hv \\ huv \\ hv^2 \end{pmatrix} \tag{3.5}$$

$$\mathbf{F_2} = \begin{pmatrix} 0 \\ 0 \\ H \end{pmatrix} \tag{3.6}$$

$$\mathbf{S} = \begin{pmatrix} 0 \\ -ghS_{fx} \\ -ghS_{fy} \end{pmatrix} \tag{3.7}$$

Con lo cual la ecuación 3.1 puede reescribirse de la siguiente manera

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{E_1} + gh\frac{\partial}{\partial x}\mathbf{E_2} + \frac{\partial}{\partial y}\mathbf{F_1} + gh\frac{\partial}{\partial y}\mathbf{F_2} = \mathbf{S} \tag{3.8}$$

La ecuación 3.8 es una ecuación diferencial parcial hiperbólica no lineal de primer orden.

3.1. Esquema de MacCormack

Existen diferentes versiones del esquema de diferencias finitas, como puede observarse de las referencias 2, 3 y 5. En el presente trabajo se emplea el esquema propuesto en la referencia 2.

El esquema se basa en diferencias finitas, la región donde se resuelve la ecuación 3.8 se discretiza en una malla rectangular, como en la figura 3.1. Los valores que se obtienen al resolver la ecuación diferencial son los componentes del vector $\mathbf{U}^{n+1}$, a partir de los cuales se obtienen los valores primarios u, v, h. Dichas variables se encuentran valuadas en el centroide de cada uno de los elementos (volúmenes) de la malla.

La forma de discretizar la ecuación 3.8 implica definir a los operadores de diferencias hacia adelante o hacia atras en la dirección x y y. Sea $\mathbf{G}$ una función vectorial, las diferencia hacia atras en x se define como:

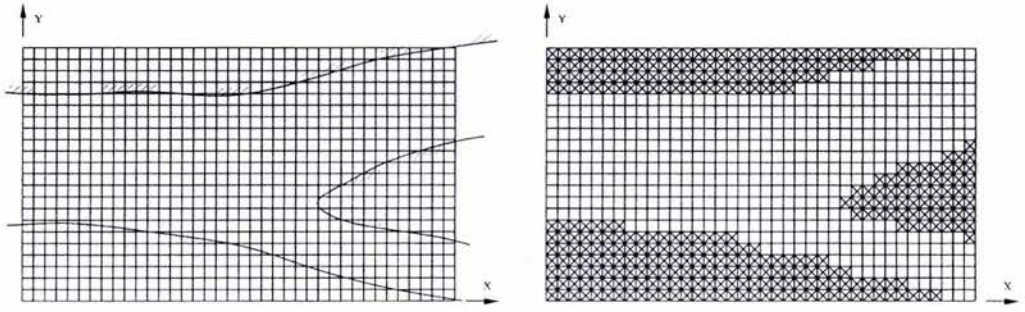
$$\nabla_x = \mathbf{G}_{i,j} - \mathbf{G}_{i,j-1} \tag{3.9}$$

y hacia adelante

$$\Delta_x = \mathbf{G}_{i,j+1} - \mathbf{G}_{i,j} \tag{3.10}$$

de igual manera, la diferencia hacia atras en la dirección del eje y

$$\nabla_y = \mathbf{G}_{i,j} - \mathbf{G}_{i-1,j} \tag{3.11}$$



a) *b)*

Figura 3.1: Malla de cálculo

y hacia adelante

$$\Delta_y = \mathbf{G}_{i+1,j} - \mathbf{G}_{i,j} \tag{3.12}$$

El esquema de MacCormack se divide en una fase de predicción y otra de corrección, las derivadas parciales en x y y se descomponen en diferencias hacia adelante o hacia atras, en la fase de predicción se utiliza una de ellas y en la corrección la complementaria. La ecuación 3.8, en la fase de predicción utilizando diferencias hacia atrás queda

$$\mathbf{U}^{*}_{i,j} = \mathbf{U}^{n}_{i,j} - \frac{\Delta t}{\Delta x}\left(\nabla_x(\mathbf{E_1})^{n}_{i,j} + gh^{n}_{i,j}\nabla_x(\mathbf{E_2})^{n}_{i,j}\right) + \\ - \frac{\Delta t}{\Delta y}\left(\nabla_y(\mathbf{F_1})^{n}_{i,j} + gh^{n}_{i,j}\nabla_y(\mathbf{F_2})^{n}_{i,j}\right) + \Delta t\mathbf{S}^{n}_{i,j} \tag{3.13}$$

La fase de corrección utilizando los operadores de diferencias hacia adelante es

$$\mathbf{U}^{**}_{i,j} = \mathbf{U}^{n}_{i,j} - \frac{\Delta t}{\Delta x}\left(\Delta_x(\mathbf{E_1})^{*}_{i,j} + gh^{*}_{i,j}\Delta_x(\mathbf{E_2})^{*}_{i,j}\right) + \\ - \frac{\Delta t}{\Delta y}\left(\Delta_y(\mathbf{F_1})^{n}_{i,j} + gh^{*}_{i,j}\Delta_y(\mathbf{F_2})^{*}_{i,j}\right) + \Delta t\mathbf{S}^{*}_{i,j} \tag{3.14}$$

El resultado final para el tiempo $n+1$ se obtiene de promediar los resultados de la fase de predicción (3.13) y la de corrección (3.14).

$$\mathbf{U}^{n+1}_{i,j} = \frac{1}{2}\left(\mathbf{U}^{*}_{i,j} + \mathbf{U}^{**}_{i,j}\right) \tag{3.15}$$

En las referencias 2, 3 y 5 se indica que es recomendable alternar el orden de las diferencias hacia adelante y hacia atrás en las fases de predicción y corrección de las

Cuadro 3.1: Combinaciones posibles para las ecuaciones de predicción y corrección

Combinación	Fase	Término E	Término F
1	predicción	∇_x	∇_y
	corrección	Δ_x	Δ_y
2	predicción	∇_x	Δ_y
	corrección	Δ_x	∇_y
3	predicción	Δ_x	Δ_y
	corrección	∇_x	∇_y
4	predicción	Δ_x	∇_y
	corrección	∇_x	Δ_y

ecuaciones 3.13 y 3.14, en la tabla 3.1 se muestran las cuatro posibles combinaciones de diferencias que se pueden usar.

Como ejemplo se tiene que usando la combinación 3 las ecuaciones quedarían de la siguiente manera

predicción:

$$\mathbf{U}^*_{i,j} = \mathbf{U}^n_{i,j} - \frac{\Delta t}{\Delta x}\left(\Delta_x(\mathbf{E_1})^n_{i,j} + gh^n_{i,j}\Delta_x(\mathbf{E_2})^n_{i,j}\right) + \\ - \frac{\Delta t}{\Delta y}\left(\Delta_y(\mathbf{F_1})^n_{i,j} + gh^n_{i,j}\Delta_y(\mathbf{F_2})^n_{i,j}\right) + \Delta t\mathbf{S}^n_{i,j} \quad (3.16)$$

corrección:

$$\mathbf{U}^{**}_{i,j} = \mathbf{U}^n_{i,j} - \frac{\Delta t}{\Delta x}\left(\nabla_x(\mathbf{E_1})^*_{i,j} + gh^*_{i,j}\nabla_x(\mathbf{E_2})^*_{i,j}\right) + \\ - \frac{\Delta t}{\Delta y}\left(\nabla_y(\mathbf{F_1})^n_{i,j} + gh^*_{i,j}\nabla_y(\mathbf{F_2})^*_{i,j}\right) + \Delta t\mathbf{S}^*_{i,j} \quad (3.17)$$

En una sucesión de cálculos de tiempo $n+1$, $n+2$, $n+3$ y $n+4$ se alternaría el orden de los operadores de diferencias usando la combinación 1, 2, 3 y 4 respectivamente para cada tiempo, iniciando con la combinación 1 para el tiempo $n+5$.

3.2. Condiciones de frontera

Se tienen dos tipos de frontera, las cerradas corresponden a fronteras por las cuales no hay un flujo de salida, por ejemplo las márgenes de un río, y las fronteras abiertas, es por donde se presenta un flujo que está entrando o saliendo de la región analizada.

3.2.1. Fronteras cerradas

Existen tres formas de considerar las fronteras cerradas, la primera es utilizar la *ley de pared*, es decir, aplicar la teoría de la capa límite para conocer la distribución de velocidades en las regiones cercanas a la pared. La segunda es considerar una *velocidad nula en la pared*, se hace la hipótesis de que no hay velocidad tangencial en la frontera. Las dos formas anteriores permiten determinar la distribución de velocidades en las celdas que colindan con las paredes y con ello evaluar las derivadas de la ecuación 3.8. La tercera forma, que es la empleada en este trabajo, se denomina de *libre deslizamiento* o *pared reflejante*, consiste en suponer que existe una celda con las mismas características del otro lado de la frontera, con igual velocidad en la dirección de la frontera y velocidades de igual magnitud pero sentido contrario en la dirección perpendicular a la frontera, como se ve en la figura 3.2 a y b. Cuando la frontera es paralela al eje x, y la celda en la frontera tiene velocidades u y v en las direcciones de los ejes x y y respectivamente, las velocidades en la celda ficticia serán u, y $-v$, la velocidad perpendicular a la frontera, v, es la que cambia de signo, y la velocidad paralela a la pared, u, se mantiene igual. Sucede lo mismo con una frontera paralela al eje y, en este caso la velocidad que cambia de signo en la celda ficticia es u.

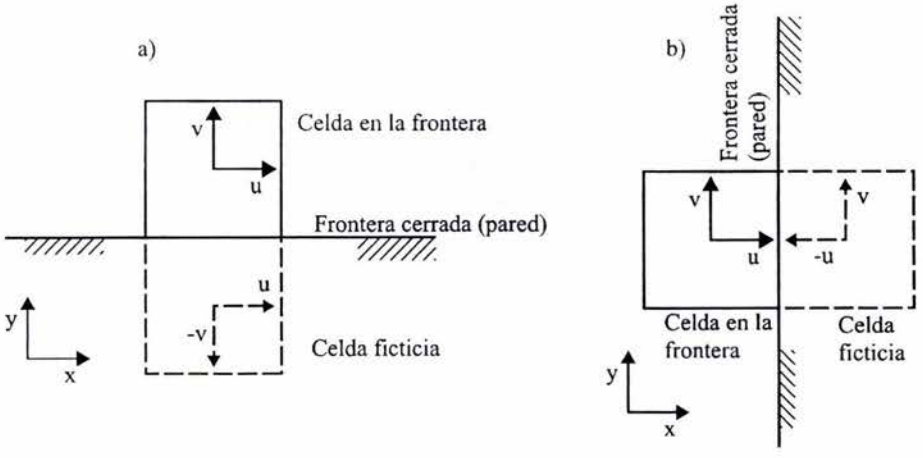


Figura 3.2: Pared reflejante: a) paralela al eje x b) paralela al eje y

Cada celda puede tener diez posibles estados referentes a las fronteras, ocho de los cuales se muestran en la figura 3.3; los otros dos corresponden al caso combinado de fronteras tipo 1 y 2 y también a la combinación de fronteras tipo 3 y 4, estas dos situaciones no se consideran en este trabajo. En cada uno de estos casos se evalúan los operadores de diferencias necesarios, con lo cual se obtienen los resultados mostrados en los cuadros 3.2 y 3.3. Allí se muestran los valores resultantes de los operadores de

diferencias para $\mathbf{E_1}$ y $\mathbf{F_1}$, los resultados para $\mathbf{E_2}$ y $\mathbf{F_2}$ no se muestran debido a que valen cero cuando se evalúan de manera perpendicular a una frontera cerrada.

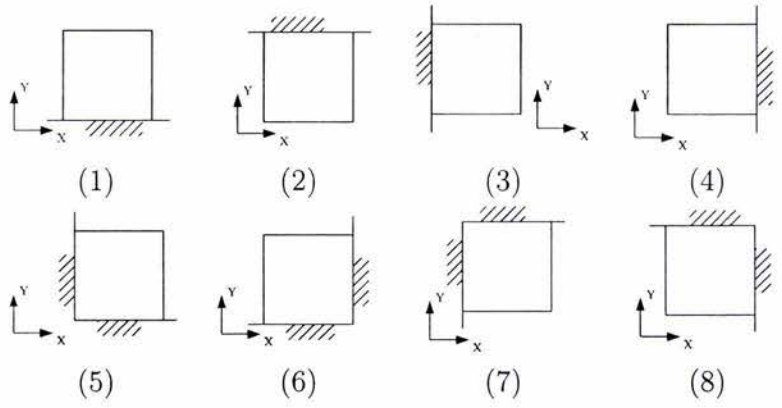


Figura 3.3: Posibles posiciones de celdas en fronteras cerradas

3.2.2. Fronteras abiertas

En la frontera por la cual está entrando el flujo al dominio donde se resuelven las ecuaciones hidrodinámicas, el número de las variables que deben ser conocidas depende del tipo de régimen que se tenga; si el régimen es subcrítico se requiere conocer dos variables, lo común es conocer el gasto en todo tiempo (Q^{n+1}) y la otra variable se infiere de suponer que la velocidad transversal a la frontera es igual a cero.

Las celdas que se encuentran en esta frontera no se calculan con las ecuaciones 3.13 y 3.14 debido a que ya es conocida una velocidad y es posible determinar el gasto unitario, sólo es necesario calcular una variable más. En este caso se emplea la ecuación de continuidad. Suponiendo que el flujo entra en la dirección del eje x, la velocidad en la dirección del eje y es cero ($v = 0$) por lo cual la ecuación de continuidad se reduce a la siguiente expresión

$$\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} = 0 \tag{3.18}$$

Expresando la ecuación anterior en diferencias finitas y despejando $h_{i,0}^{n+1}$ se tiene

$$h_{i,j}^{n+1} = h_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left((hu)_{i,j+1}^{n} - (hu)_{i,j}^{n+1}\right) \tag{3.19}$$

El término $(hu)_{i,j}^{n+1}$ se obtiene de dividir el gasto Q^{n+1} entre el ancho de la sección transversal de la frontera de entrada. $(hu)_{i,j}^{n+1} = q^{n+1} = Q^{n+1}/B$. En los valores ante-

Cuadro 3.2: Operadores de diferencias en celdas con una frontera cerrada

Tipo de Celda	Operación	Resultado
1	$\nabla_y(\mathbf{F_1})_{i,j}$	$\begin{pmatrix} 2hv \\ 2huv \\ 0 \end{pmatrix}_{i,j}$
2	$\Delta_y(\mathbf{F_1})_{i,j}$	$\begin{pmatrix} -2hv \\ -2huv \\ 0 \end{pmatrix}_{i,j}$
3	$\nabla_x(\mathbf{E_1})_{i,j}$	$\begin{pmatrix} 2hu \\ 0 \\ 2huv \end{pmatrix}_{i,j}$
4	$\Delta_x(\mathbf{E_1})_{i,j}$	$\begin{pmatrix} -2hu \\ 0 \\ -2huv \end{pmatrix}_{i,j}$

Cuadro 3.3: Operadores de diferencias en celdas con dos fronteras cerradas

Tipo de Celda	Operación	Resultado
5	$\nabla_x(\mathbf{E_1})_{i,j}$	Igual que en la celda tipo 3
	$\nabla_y(\mathbf{F_1})_{i,j}$	igual que en la celda tipo 1
6	$\Delta_x(\mathbf{E_1})_{i,j}$	Igual que en la celda tipo 4
	$\nabla_y(\mathbf{F_1})_{i,j}$	igual que en la celda tipo 1
7	$\nabla_x(\mathbf{E_1})_{i,j}$	Igual que en la celda tipo 3
	$\Delta_y(\mathbf{F_1})_{i,j}$	igual que en la celda tipo 2
8	$\Delta_x(\mathbf{E_1})_{i,j}$	Igual que en la celda tipo 4
	$\Delta_y(\mathbf{F_1})_{i,j}$	igual que en la celda tipo 2

riores $j = 0$, debido a que se trata de la frontera de entrada.

Si el flujo entrara en dirección paralela al eje y, la velocidad paralela al eje x en las celdas de esa frontera sería cero ($u = 0$) y para determinar $h^n_{i,j}$ se procedería de una manera análoga a como se hizo con las ecuaciones 3.18 y 3.19.

En la frontera por donde sale el flujo de la región, en flujo subcrítico, es necesario conocer dos variables, lo común es conocer el tirante y suponer que la velocidad transversal es cero, si el flujo es paralelo al eje x entonces $v = 0$, en este caso se emplea la ecuación dinámica en la dirección del eje x; con la consideración de que la velocidad transversal es cero, se tiene

$$\frac{\partial hu}{\partial t} + \frac{\partial hu^2}{\partial x} + gh\frac{\partial H}{\partial x} = -ghS_{fx} \tag{3.20}$$

Expresando en diferencias finitas la ecuación anterior y despejando $(hu)^{n+1}_{i,j}$ se tiene

$$(hu)^{n+1}_{i,j} = (hu)^n_{i,j} - \frac{\Delta t}{\Delta x}\left((hu^2)^n_{i,j} - (hu^2)^n_{i,j-1}\right) + \\ - \frac{\Delta t}{\Delta x} gh^n_{i,j}\left(H^{n+1}_{i,j} - H^n_{i,j-1}\right) - \Delta t\, gh^{n+1}_{i,j} S_{fx} \tag{3.21}$$

Si en la frontera de salida el flujo es paralelo al eje y, se tiene que $u = 0$ se emplea la ecuación dinámica en la dirección y y se hace un desarrollo similar al de las ecuaciones 3.21 y 3.20.

3.3. Procedimiento de cálculo

En las secciones anteriores de este capítulo se obtuvieron las ecuaciones en forma discretizada para resolver las ecuaciones hidrodinámicas empleando el esquema de MacCormack. Ahora se describe el procedimiento a seguir para hacer una programa computacional que implemente el esquema descrito.

La información requerida considerando que el flujo es subcrítico es el gasto en la entrada, los niveles en la salida, si se trata de una bifurcación es necesario conocer las elevaciones en los dos brazos, los dos valores de frontera anteriores deben conocerse para cualquier tiempo.

En lo que sigue se considera que el flujo de entrada a la región del problema es por la frontera izquierda perpendicular al eje x (frontera oeste), y la salida es por la frotera derecha (frontera este).

El procedimiento para preparar la información requerida es el siguiente.

1. En la región de estudio se genera una malla rectangular, con M celdas en la dirección x y N celdas en la dirección y, (ver figura 3.1 a). Las celdas que queden fuera de las márgenes del río se marcan como inactivas (ver figura 3.1 b).

2. Se identifican las celdas que corresponden a la frontera de entrada y las celdas de la frontera de salida. Una nomenclatura para identificar celdas activas o inactivas, o de frontera es la siguiente

Tipo de celda	**Identificador**
Celda inactiva	0
Celda activa (que no es frontera de entrada o salida)	1
Frontera de entrada	2
Frontera de salida	3

   Debido a que las celdas que se encuentran junto a las fronteras cerradas se pueden identificar fácilmente verificando si existen celdas inactivas a su alrededor o si se encuentran en la frontera de la malla, pude programarse el algoritmo para que identifique esta celdas y les asigne el tipo correspondiente de los mostrados en la figura 3.3.

3. Se especifican los niveles de cota $z_{i,j}$ en las celdas activas y el gasto en la frontera de entrada, ya sea con un hidrograma o asignando un valor constante; en la frontera de salida se especifica el nivel $h_{i,j}$. Con lo anterior se inician las iteraciones en el tiempo.

Los valores iniciales de velocidad y tirante, h^0, u^0 y v^0 deben ser conocidos y para calcular los valores en el tiempo $n+1$ el procedimiento de cálculo es el siguiente:

1. Se calculan los valores de predicción en las celdas activas, que no corresponden a celdas de entrada o salida empleando la ecuación de predicción 3.13. Se identifica si correponde a una celda de frontera, si es así debe definirse a qué tipo de celda corresponde de las mostradas en la figura 3.3, y de acuerdo con ello se utiliza el resultado adecuado de los mostrados en los cuadros 3.2 o 3.3 para evaluar los operadores de diferencias.

2. En las celdas de la frontera de entrada ya es conocido $v^{n+1} = 0$ y el valor $(hu)_{i,j}^{n+1}$ se obtiene de dividir Q^{n+1} entre el ancho transversal de la frontera de entrada, $h_{i,j}^{n+1}$ se obtiene a partir de la ecuación 3.19 con lo cual ya se conocerían todos los valores del vector $\mathbf{U}_{i,j}^{n+1}$ en el cual $(hv)_{i,j}^{n+1} = 0$.

3. En la frontera de salida $h_{i,j}^{n+1}$ ya es conocido y considerando que $v_{i,j}^{n+1} = 0$ sólo resta calcular $(hu)_{i,j}^{n+1}$, esto se hace con la ecuación 3.21 con lo cual ya son conocidos todos los valores del vector $\mathbf{U}_{i,j}^{n+1}$.

4. Se calculan los valores de corrección con la ecuación 3.14. Para las celdas que se encuentren en fronteras cerradas se utilizan los resultados de los cuadros 3.2 y 3.3 para evaluar los operadores de diferencias. En este paso y en anterior se debe alternar en cada tiempo de cálculo el orden de los operadores de las ecuaciones 3.13 y 3.14 como se indica en el cuadro 3.1.

5. Se calculan los valores de $\mathbf{U}_{\mathbf{i,j}}^{\mathbf{n+1}}$ promediando los valores predichos y corregidos con la ecuación 3.15.

6. En este punto ya se tiene el cálculo de los vectores $\mathbf{U}_{i,j}^{n+1}$ en todas las celdas activas, (con identificador mayor o igual a 1) y con dichos vectores es posible determinar las variables primitivas:

$$h_{i,j}^{n+1} = h_{i,j}^{n+1}$$

$$u_{i,j}^{n+1} = \frac{(hu)_{i,j}^{n+1}}{h_{i,j}^{n+1}}$$

$$v_{i,j}^{n+1} = \frac{(hv)_{i,j}^{n+1}}{h_{i,j}^{n+1}}$$

La manera de evaluar la pendiente de fricción en dirección de x y de y es empleando la ecuación de Chezy.

$$S_{fx} = \frac{1}{hC^2} u\sqrt{u^2 + v^2} \tag{3.22}$$

$$S_{fy} = \frac{1}{hC^2} v\sqrt{u^2 + v^2} \tag{3.23}$$

Donde C es el coeficiente de Chezy. Si éste coeficiente se evalúa con la fórmula de Manning, las ecuaciones 3.22 y (3.23) quedan

$$S_{fx} = \frac{n^2}{h^{4/3}} u\sqrt{u^2 + v^2} \tag{3.24}$$

$$S_{fy} = \frac{n^2}{h^{4/3}} v\sqrt{u^2 + v^2} \tag{3.25}$$

Donde n es el coeficiente de fricción.

3.4. Condiciones de estabilidad

Para que el esquema de MacCormack en dos dimensiones sea estable el número de Courant C_n debe cumplir con la siguiente condición

$$\Delta t < C_n \frac{1}{\frac{(u+\sqrt{gh})}{\Delta x} + \frac{(v+\sqrt{gh})}{\Delta y}} \tag{3.26}$$

El número de Courant debe ser siempre $C_n \leq 1{,}0$.

Por otra parte, se recomienda utilizar un filtro numérico debido a que el esquema presentado produce algunas oscilaciones en los resultados. El filtro consiste en afectar el vector $\mathbf{U}$ (descrito por la ecuación 3.2) de la celda i, j por los valores de los vectores $\mathbf{U}$ de las celdas que se encuentran alrededor.

$$\mathbf{R} = \mathbf{U}_{i-1,j} + \mathbf{U}_{i+1,j} + \mathbf{U}_{i,j-1} + \mathbf{U}_{i,j+1} \tag{3.27}$$

$$\mathbf{U}^f_{i,j} = (1 - 4\alpha)\mathbf{U}_{i,j} + \alpha\mathbf{R} \tag{3.28}$$

El valor del factor de peso α se recomienda entre $0 \leq \alpha \leq 0{,}25$. El filtro se aplica cada cierto número de iteraciones, el valor final filtrado es $\mathbf{U}^f_{i,j}$.

3.5. Verificación del modelo

Con lo descrito en las secciones anteriores de este capítulo se generó un programa de cómputo; una manera de verificar los resultados que produce dicho programa es compararlos realizando cálculos por otra metodología que ya sea conocida.

3.5.1. Comparación con un modelo unidimensional

Se calcula el perfil de un canal en 1-D, a través de la ecuación dinámica para flujo permanente en una dimensión (3.29). La deducción de esta ecuación se encuentra en la referencia 12.

$$\frac{dh}{dx} = \frac{S_0 - S_f}{1 - F^2} \tag{3.29}$$

donde F es el número de Froude.

Se emplea un canal con ancho $B = 100m$, gasto $Q = 50m^3/s$, longitud $L = 200m$ y pendienete $S_0 = 0{,}001$. En la figura 3.4 se muestra el perfil obtenido de resolver la ecuación 3.29 y el obtenido por medio del equema de MacCormack, dividiendo el canal en una malla de 200 elementos, con $\Delta x = 10m$ y $\Delta y = 10m$. Para Las condiciones mencionadas el tirante normal es $h_n = 0{,}422m$ y el crítico es $h_c = 0{,}295m$.

Se observa que el resultado del modelo bidimensional es prácticamente el mismo que el obtenido al resolver la ecuación unidimensional.

3.5.2. Comparación con un modelo bidimensional

Para verificar el comportamiento del modelo en un problema bidimensional se emplean los datos del el experimento reportado en la referencia 9, donde se presentan resultados de mediciones realizados en un canal rectangular con un espigón.

Los datos a emplear corresponden al experimento A2 de la referencia 9 y se presentan en el cuadro 3.4. Se indica que el flujo es uniforme y el tirante igual a $h = 0{,}223m$, como no se especifica la pendiente ni la rugosidad, se fija la pendiente en $S_0 = 0{,}0001$, para lo cual el coeficiente de rugosidad debe ser $n = 0{,}16565$. En la figura 3.5.2 se muestra el arreglo del canal para las pruebas.

Cuadro 3.4: Datos para la prueba de flujo bidimensional

tirante h	0.223 m
Ancho del canal B	0.915 m
Longitud de espigón	0.15 m
Gasto Q	0.0453 m^3/s

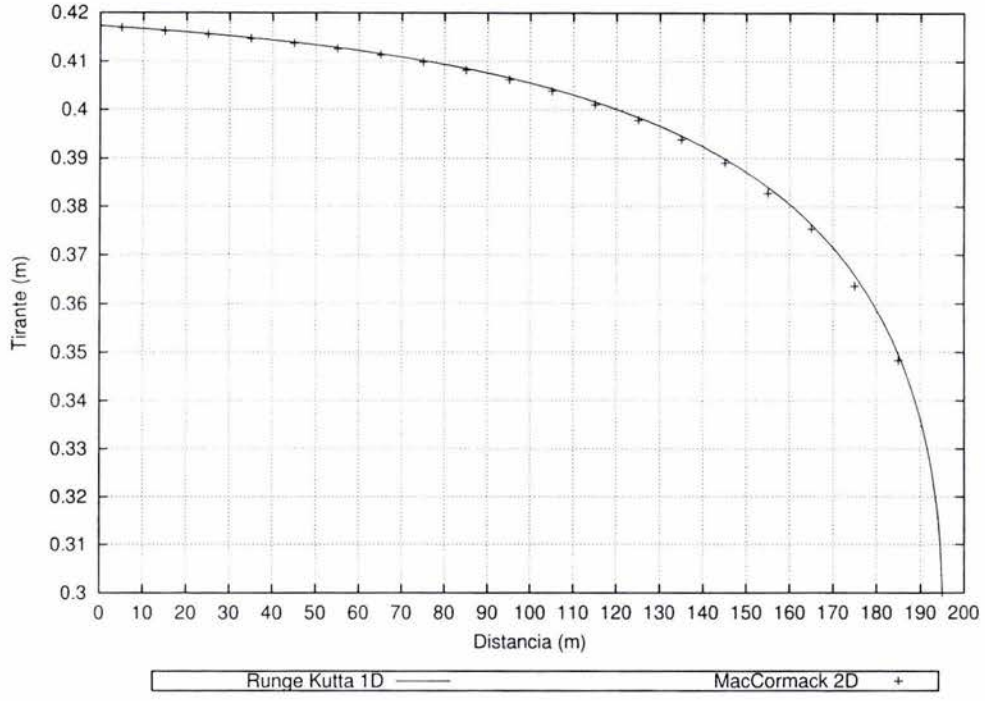


Figura 3.4: Comparación de los resultados optenidos por el esquema de MacCormack planteado y por la ecuación de flujo permanente resuelta con el esquema de Runge Kutta de cuarto orden

La malla empleada se compone de 360 celdas en la dirección x y 61 celdas en la dirección y, las celdas son cuadradas con dimensiones $\Delta x = \Delta y = 0{,}015m$. El itervalo de tiempo empleado fue $\Delta t = 2s$.

Como condición inical se especificó que las velocidades fueran cero, y el nivel de agua constante, se dio un hidrograma que parte de cero y llega al gasto especificado en 20 segundos, el modelo llega a una condición estable después de simular aproximadamente 20 minutos. El campo de velocidades se muestra en la figura 3.6. La componente de velocidad en la dirección x, u, se muestra en diferentes secciones transversales en la figura 3.7.

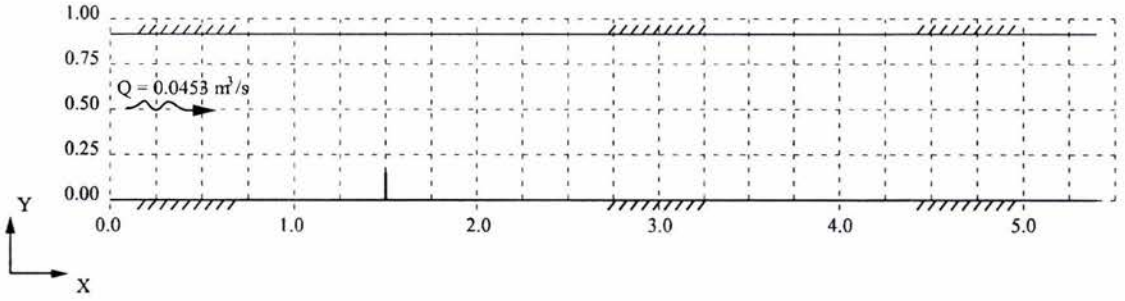


Figura 3.5: Canal con espigón empleado para la prueba, las dimensiones están en metros

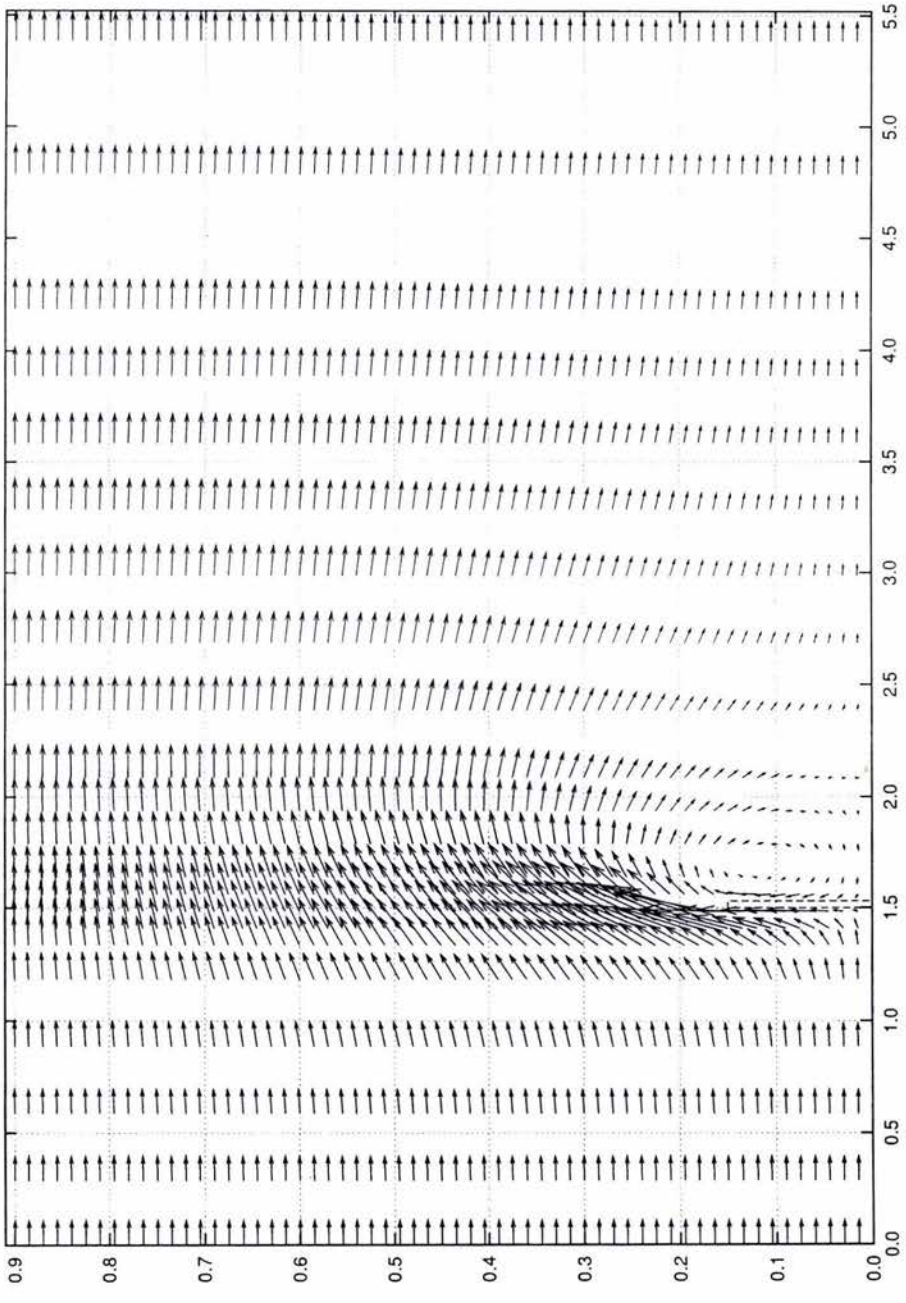
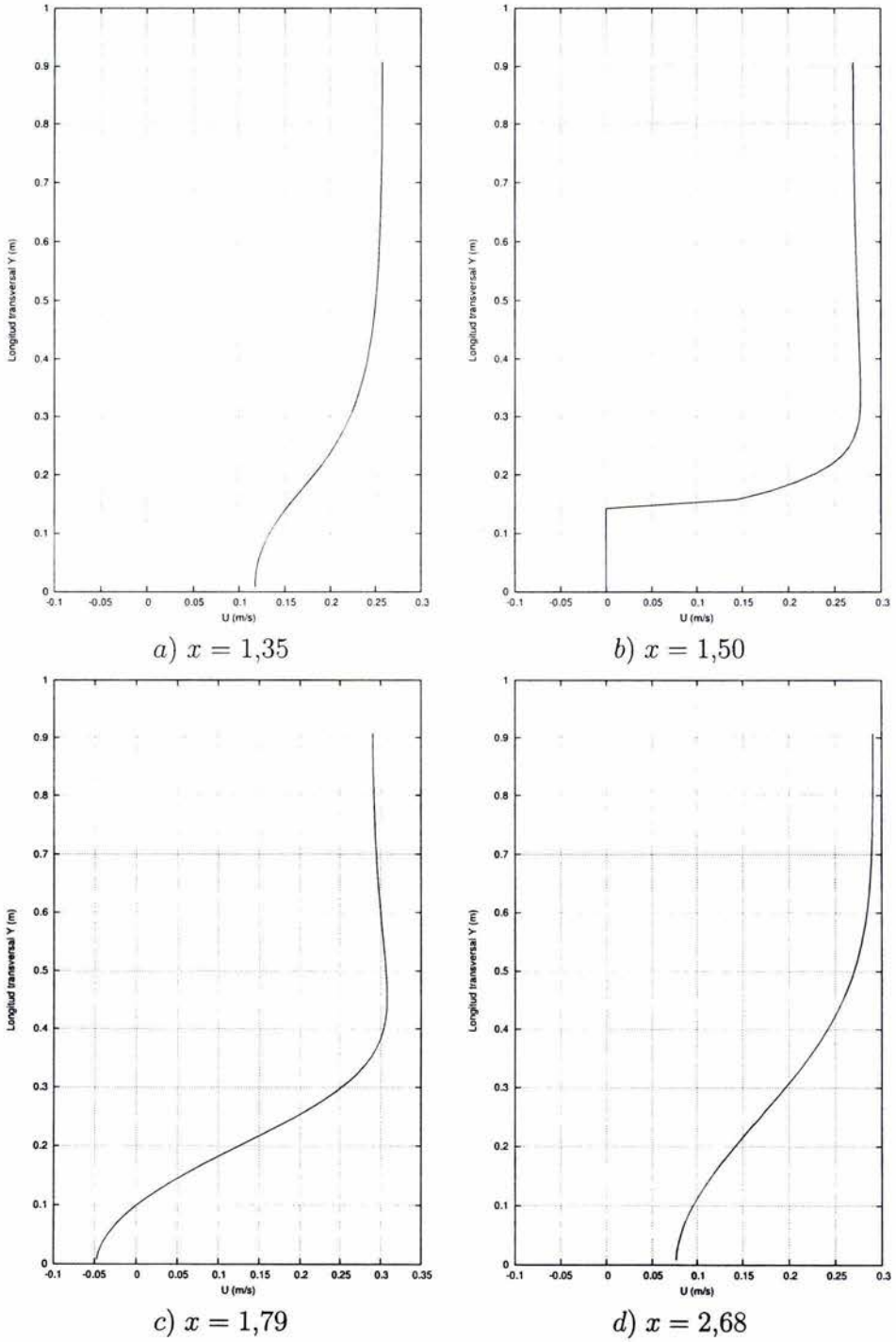


Figura 3.6: Campo de valocidades en el canal con espigón



a) $x = 1{,}35$ *b*) $x = 1{,}50$

c) $x = 1{,}79$ *d*) $x = 2{,}68$

Figura 3.7: Comparación de la componente u calculada

Capítulo 4

Transporte de sedimento

En los capítulos anteriores se plantearon las ecuaciones hidrodinámicas para flujo bidimensional a superficie libre, ahora en este capítulo se revisa el planteamiento para determinar el arrastre de sedimentos, y determinar cómo evoluciona el fondo de un cauce ocasionado por el transporte total de fondo.

El sedimento transportado se cuantifica en peso o volumen de manera unitaria, si se designa en peso se denomina g_x y sus unidades son $kg/(m \cdot s)$, si es en volumen entonces se utiliza q_x con unidades $m^3/(m \cdot s)$; el subíndice x corresponde al tipo de transporte, como se verá más adelante.

4.1. Tipos de transporte

La forma en que se transporta el sedimento puede clasificarse en seis tipos que dependen del material y del lugar donde ocurre el transporte. Los grupos de materiales transportados se dividen en material del fondo y de lavado; el primero es el mismo material que conforma el lecho del cauce, el material de lavado es sedimento fino, como limos o arcillas y se transporta siempre en suspensión. El material de fondo se transporta en la región denominada capa de fondo o suspendido. El material de lavado se transporta siempre suspendido.

4.1.1. Arrastre en la capa de fondo

El material de fondo del cauce es arrastrado por la corriente dentro de la denominada *capa de fondo*, la cual, según Einstein, tiene un espesor de dos veces el diámetro de las partículas del fondo, aunque otros autores han propuesto un espesor diferente.

El arrastre de sedimento es función de las características hidráulicas de la corriente, de la geometría del cauce y de las características físicas del material del fondo. Se

denomina con el subíndice B.

4.1.2. Transporte de material del fondo en suspensión

Está formado por el material del fondo que es transportado en suspensión por la corriente, se encuentra en movimiento arriba de la capa de fondo. Este tipo de transporte se origina por la velocidad y turbulencia de la corriente que levanta las partículas del fondo y las mantiene en suspensión, la concentración disminuye cuando la velocidad y la turbulecia son pequeñas. Las fuerzas que tratan de mover las partículas del fondo son las de arrastre y sustentación originadas por la corriente, las fuerzas que impiden su movimiento son el peso propio y la fricción de las partículas con las que se encuentra en contacto; una vez que la partícula se encuentra en movimiento, la única fuerza que impide su movimiento es su peso. A este tipo de transporte se denomina con el subíndice BS

4.1.3. Transporte total de fondo

Es el material de fondo que se transporta tanto en la capa de fondo como de forma suspendida, por lo cual este tipo de transporte es igual a la suma del material de arrastre de fondo mas el material de fondo en suspensión. Se denomina con el subíndice BT.

$$g_{BT} = g_B + g_{BS} \tag{4.1}$$

La expresión (4.1) permite calcular el peso unitario de material de fondo total transportado, de la misma forma se podría calcular el volumen unitario q_{BT}.

4.1.4. Transporte de lavado

Es material muy fino que se transporta en suspensión y no corresponde al material del fondo del cauce. Su origen es debido a la erosión ocasionada por las lluvias en las cuencas de aportación al río o por la erosión de las márgenes del mismo. El transporte de este tipo de material no es función de las características de la corriente o de la geometría del cauce. Si no son conocidas las características del materiál de fondo, se considera material de lavado aquellas que tengan un diámetro menor o igual a $0{,}062mm$. Este tipo de transporte se denomina con el subíndice l.

4.1.5. Transporte en suspensión

Es el material total que se transporta en suspensión, la suma del material de fondo en suspensión mas el material de lavado, se denomina con el subíndice S

$$g_S = g_L + g_{BS} \tag{4.2}$$

4.1.6. Transporte total

Está constituido por el total de las partículas que son transportadas, la suma del material total de fondo mas el de lavado, se denomina con el subíndice T

$$g_T = g_L + g_{BT} \tag{4.3}$$

o también puede evaluarse de la siguiente manera

$$g_T = g_B + g_S \tag{4.4}$$

4.2. Cuantificación del transporte total de fondo

El cambio en el tiempo de las características del fondo de un cauce se debe a la sedimentación o la erosión del lecho, los fenómenos anteriores se pueden estimar cuando se calcula por el transporte total de fondo.

Para este trabajo se propone utilizar el criterio de Engelund - Hansen. El volumen unitario de material trasportado se calcula con la siguiente expresión

$$q_{BT} = \frac{0{,}05\, U^5}{\Delta^2 \sqrt{g}\, d_{50}\, C^3} \tag{4.5}$$

donde

q_{BT}	transporte total del material de fondo en $m^3/(m \cdot s)$
U	velocidad media del flujo em m/s
d_{50}	diámetro medio de las partículas del fondo en m
Δ	densidad relativa del material, adimensional $\Delta = \frac{\gamma_s - \gamma}{\gamma}$
g	aceleración de la gravedad en m/s^2
C	coeficiente de Chezy en $m^{1/2}/s$

Utilizando la fórmula de Manning y considerando que el radio hidráulico es igual al tirante $R_h = h$, el coeficiente de Chezy se puede calcular con la expresión siguiente

$$C = \frac{h^{1/6}}{n} \tag{4.6}$$

Sustituyendo (4.6) en (4.5) y agrupando los términos constantes se llega a

$$q_{BT} = K \frac{U^5}{h^{1/2}} \tag{4.7}$$

donde K es un coeficiente de términos constantes

$$K = \frac{0{,}05\, n^3}{\Delta^2 \sqrt{g}\, d_{50}} \tag{4.8}$$

4.3. Determinación del depósito o de la erosión

De aquí en adelante el gasto de material de fondo se denomina con q_s.

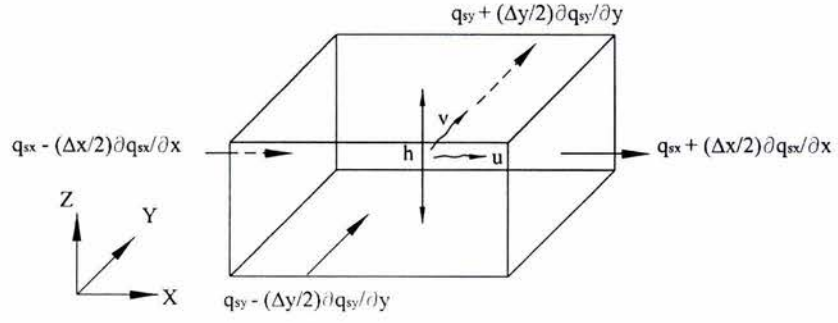


Figura 4.1: Volumen de control para cuantificar el depósito de sedimento

El volumen de control mostrado en la figura 4.1, permite determinar la cantidad de material que entra y sale del volumen de control. El flujo de material que entra al volumen de control por la frontera oeste, en un intervalo de tiempo Δt es

$$\left(q_{sx} - \frac{\Delta x}{2}\frac{\partial q_{sx}}{\partial x}\right)\Delta y \Delta t \tag{4.9}$$

El material que sale por la frontera este es

$$\left(q_{sx} + \frac{\Delta x}{2}\frac{\partial q_{sx}}{\partial x}\right)\Delta y \Delta t \tag{4.10}$$

En la dirección y el flujo de material que entra por la frontera sur es

$$\left(q_{sy} - \frac{\Delta y}{2}\frac{\partial q_{sy}}{\partial y}\right)\Delta x \Delta t \tag{4.11}$$

En la frontera norte el material que sale es

$$\left(q_{sy} + \frac{\Delta y}{2}\frac{\partial q_{sy}}{\partial y}\right)\Delta x \Delta t \tag{4.12}$$

Por lo tanto empleando los resultados anteriores se puede determinar el volumen del material almacenado dentro del volumen de control, dicho volumen se ve afectado

por la porosidad p; la cantidad de material que se almacena dentro de dicho volumen es proporcional a la variación de la cota z, en un intervalo de tiempo Δt por lo tanto

$$(1-p)\Delta x\Delta y\Delta z = \left(q_{sx} - \frac{\Delta x}{2}\frac{\partial q_{sx}}{\partial x}\right)\Delta y\Delta t - \left(q_{sx} + \frac{\Delta x}{2}\frac{\partial q_{sx}}{\partial x}\right)\Delta y\Delta t + \left(q_{sy} - \frac{\Delta y}{2}\frac{\partial q_{sy}}{\partial y}\right)\Delta x\Delta t - \left(q_{sy} + \frac{\Delta y}{2}\frac{\partial q_{sy}}{\partial y}\right)\Delta x\Delta t \quad (4.13)$$

Simplificando la ecuación anterior se llega a una versión diferencial para transporte de sedimentos, los incrementos Δz y Δt se llevan al limite cercano a cero para que sean ∂z y ∂t respectivamente.

$$\frac{\partial z}{\partial t} + \frac{1}{1-p}\left(\frac{\partial q_{sx}}{\partial x} + \frac{\partial q_{sy}}{\partial y}\right) = 0 \quad (4.14)$$

Debido a que la porosidad p es un valor constante, el valor $1/(1-p)$ puede incorporarse a la ecuación 4.5, con lo cual, la constante K se redefine como (4.15) y la ecuación diferencial 4.14 queda definida por (4.16).

$$K' = \frac{0{,}05\, n^3}{(1-p)\Delta^2\sqrt{g}\, d_{50}} \quad (4.15)$$

$$\frac{\partial z}{\partial t} + \frac{\partial q_{sx}}{\partial x} + \frac{\partial q_{sy}}{\partial y} = 0 \quad (4.16)$$

4.4. Integración con las ecuaciones hidrodinámicas

La ecuación 4.16 se puede integrar a la ecuación vectorial 3.1 vista en el capítulo anterior.

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ uh \\ vh \\ z \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} hu \\ hu^2 \\ hvu \\ q_{sx} \end{pmatrix} + gh\frac{\partial}{\partial x}\begin{pmatrix} 0 \\ H \\ 0 \\ 0 \end{pmatrix} + \frac{\partial}{\partial y}\begin{pmatrix} hv \\ huv \\ hv^2 \\ q_{sy} \end{pmatrix} + gh\frac{\partial}{\partial y}\begin{pmatrix} 0 \\ 0 \\ H \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -ghS_{fx} \\ -ghS_{fy} \\ 0 \end{pmatrix} \quad (4.17)$$

Con lo cual se pueden definir las siguientes variables

$$\mathbf{U_s} = \begin{pmatrix} h \\ uh \\ vh \\ z \end{pmatrix} \tag{4.18}$$

$$\mathbf{E_{s1}} = \begin{pmatrix} hu \\ hu^2 \\ hvu \\ q_{sx} \end{pmatrix} \tag{4.19}$$

$$\mathbf{E_{s2}} = \begin{pmatrix} 0 \\ H \\ 0 \\ 0 \end{pmatrix} \tag{4.20}$$

$$\mathbf{F_{s1}} = \begin{pmatrix} hv \\ huv \\ hv^2 \\ q_{sy} \end{pmatrix} \tag{4.21}$$

$$\mathbf{F_{s2}} = \begin{pmatrix} 0 \\ 0 \\ H \\ 0 \end{pmatrix} \tag{4.22}$$

$$\mathbf{S_s} = \begin{pmatrix} 0 \\ -ghS_{fx} \\ -ghS_{fy} \\ 0 \end{pmatrix} \tag{4.23}$$

Los gastos unitarios de material sólido q_{sx} y q_{sy} se evaluan con la ecuación 4.7 empleando las velocidades de en la dirección x y y respectivamente.

$$q_{sx} = K' \frac{u^5}{h^{1/2}} \tag{4.24}$$

y

$$q_{sy} = K' \frac{v^5}{h^{1/2}} \tag{4.25}$$

donde K' se evalua con (4.15).

Finalmente, con lo anterior se llega a una ecuación similar a la 3.8.

Cuadro 4.1: Operadores de diferencias en celdas con una frontera cerrada

Tipo de Celda	Operación	Resultado
1	$\nabla_y \mathbf{F_{s1}}_{i,j}$	$\begin{pmatrix} 2hv \\ 2huv \\ 0 \\ 2q_{sy} \end{pmatrix}_{i,j}$
2	$\Delta_y \mathbf{F_{s1}}_{i,j}$	$\begin{pmatrix} -2hv \\ -2huv \\ 0 \\ -2q_{sy} \end{pmatrix}_{i,j}$
3	$\nabla_x \mathbf{E_{s1}}_{i,j}$	$\begin{pmatrix} 2hu \\ 0 \\ 2huv \\ 2q_{sx} \end{pmatrix}_{i,j}$
4	$\Delta_x \mathbf{E_{s1}}_{i,j}$	$\begin{pmatrix} -2hu \\ 0 \\ -2huv \\ -2q_{sx} \end{pmatrix}_{i,j}$

$$\frac{\partial}{\partial t}\mathbf{U_s} + \frac{\partial}{\partial x}\mathbf{E_{s1}} + \frac{\partial}{\partial x}\mathbf{E_{s2}} + \frac{\partial}{\partial y}\mathbf{F_{s1}} + \frac{\partial}{\partial y}\mathbf{F_{s2}} = \mathbf{S_s} \tag{4.26}$$

La ecuación 4.26 se resuelve con el esquema de MacCormak, de manera idéntica a como se describió en el capítulo anterior. Sin embargo falta definir las condiciones de frontera para la ecuación 4.16.

4.4.1. Fronteras cerradas

Se emplea la hipótesis de pared reflejante definida en la sección 3.2.1 del capítulo anterior para determinar las condiciones de flujo de material sólido transportado en este tipo de fronteras. En el cuadro 4.1 se tienen evaluados los operadores de diferencias aplicados a la ecuación 4.26 para los tipos de celdas definidos en la figura 3.3, las celdas con frontera tipo 5, 6, 7 y 8 se evalúan como se indica en el cuadro 3.3.

4.4.2. Fronteras abiertas

En el caso de fronteras abiertas donde está entrando un gasto líquido que se espefica por medio de un hidrograma, se debe establecer como condición de frontera un sedimentograma, como el indicado en la figura 4.2; si se estan modelando condiciones de flujo permanente tanto el hidrograma como el sedimentograma son líneas horizontales.

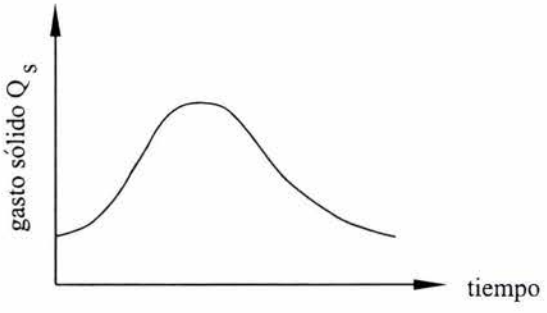


Figura 4.2: Sedimentograma proporcionado como condición de frontera

Como se puede observar en el apartado 3.2.1, se estableció como condición que la velocidad en dirección del eje y es cero, por lo anterior se tiene que el gasto unitario de material sólido q_{sx} es cero también, lo que permite simplificar la ecuación 4.14.

$$\frac{\partial z}{\partial t} + \frac{\partial q_{sx}}{\partial x} = 0 \tag{4.27}$$

En la ecuación 4.27 el valor q_{sx} se conoce en todo tiempo $n+1$ a partir del sedimentograma especificado como condición de frontera, por lo ello puede discretizarse de la siguiente manera

$$z_{i,j}^{n+1} = z_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left((q_{sx})_{i,j+1}^{n} - (q_{sx})_{i,j}^{n+1}\right) \tag{4.28}$$

Las ecuaciones 3.19 y 4.28 permiten evaluar al vector $\mathbf{U_s}$ en las fronteras de entrada en el tiempo $n+1$.

En la frontera de salida, debido a que el flujo es paralelo al eje x, se tiene también la hipótesis de flujo paralelo a dicho eje, por lo cual v es cero, por lo cual la ecuación 4.16 se reduce nuevamente a (4.27), que discretizandose queda de la siguiente manera

$$z_{i,j}^{n+1} = z_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left((q_{sx})_{i,j}^{n} - (q_{sx})_{i,j-1}^{n}\right) \tag{4.29}$$

En (4.29) los términos $(q_{sx})^n_{i,j}$ y $(q_{sx})^n_{i,j-1}$ se evalúan con la ecuación 4.7 emplenado los datos de las celdas (i,j) y $(i,j-1)$ respectivamente.

4.5. Procedimiento de cálculo

El procedimiento de cálculo es similar al descritp en el apartado 3.3 del capítulo anterior, sin embargo se describe aquí nuevamente.

Conocidos los valores h^0, u^0, y v^0, los valores en el tiempo $(n+1)\Delta t$ se calculan con el siguiente procedimiento:

1. Se determinan los valores de predicción, empleando la ecuación de predicción (3.13), pero en lugar de emplear los vectores $\mathbf{F_1}$, $\mathbf{F_2}$, $\mathbf{E_1}$, $\mathbf{E_2}$ definidos en el capítulo anterior, se emplean los vectores $\mathbf{F_{s1}}$, $\mathbf{F_{s2}}$, $\mathbf{E_{s1}}$, $\mathbf{E_{s2}}$ definidos por (4.21), (4.22), (4.19) y (4.20) respectivamente. Las celdas ubicadas en fronteras se evalúan con ayuda de los valores definidos en el cuadro 4.1.

2. En la frontera de entrada, ya es conocido el valor $(hu)^{n+1}_{i,j}$ y el tirante h^{n+1} se obtiene de evaluar (3.19). La cota $z^{n+1}_{i,j}$ se determina con (4.28) empleando la información proporcionada por el sedimentograma.

3. En la frontera de salida se evalúa $(hu)^{n+1}_{i,j}$ con (3.21). La cota $z^{n+1}_{i,j}$ se determina con (4.28).

4. Los valores corregidos en cada celda se determinan con (3.14), las celdas de las fronteras cerradas se evalúan con ayuda de los resultados mostrados en cuadro 4.1.

5. $(\mathbf{U_s})^{n+1}_{i,j}$ se calcula promediando los valores predichos y corregidos empleando (3.15).

6. Conocidos los valores $(\mathbf{U_s})^{n+1}_{i,j}$ en todas las celdas, se determinan las variables primitivas h, u y v como se indica en el procedimiento descrito en el capítulo anterior. El valor de la cota z es directamente el cuarto elemento del vector $\mathbf{U_s}$.

Las condiciones de estabilidad que deben cumplirse son las descritas en el apartado 3.4 del capítulo anterior. En esta versión también se emplea un filtro numérico (ecuaciones 3.27 y 3.28) para determinar un vector filtrado $\mathbf{U_s}^f$, dicho filtro sólo se aplica a los valores h, hu y hv, pero no a z.

Capítulo 5

Modelo de la bifurcación

Con el procedimiento descrito en los capítulos 3 y 4 se implantó un programa que emplea los esquemas de solución planteados, con dicho programa se realizaron 4 modelaciones que se describen a continuación.

Como se mencionó en el capítulo 1, se pretende encontrar las condiciones que obligaron a que con el paso del tiempo, el gasto que circula por el río Carrizal aumentara.

5.1. Primera modelación: desde la bifurcación hasta los puentes Samaria I y II

La primera modelación realizada cubre un área que va desde la bifurcación hasta los puentes Samaria I y II, la malla generada es de 106 celdas en la dirección x y 102 celdas en la dirección y. Cada celda es de 50×50 metros. En la figura 5.1 se muestra la malla de cálculo.

Se consideró un coeficiente de rugosidad $n = 0{,}023$, y el diámetro de las partículas del fondo es $d_{50} = 1{,}6mm$, densidad relativa del material es $\Delta = 1{,}65$ y porosidad $p = 0{,}4$. Con los valores anteriores la constante K' empleada en la ecuación 4.7 es:

$$K' = (0{,}05(0{,}023)^3)/((1 - 0{,}4)1{,}65^2\sqrt{9{,}81}(0{,}0008)) = 0{,}000074$$

En la figura 5.2 se muestra la topografía del fondo de la zona modelada, se distingue un canal más profundo que viene del río Mezcalapa y llega a la entrada del río Carrizal.

Antes de realizar una modelación de un periodo largo de tiempo primero se determinaron las condiciones existentes en la bifurcación para un gasto $Q = 600m^3/s$, y una

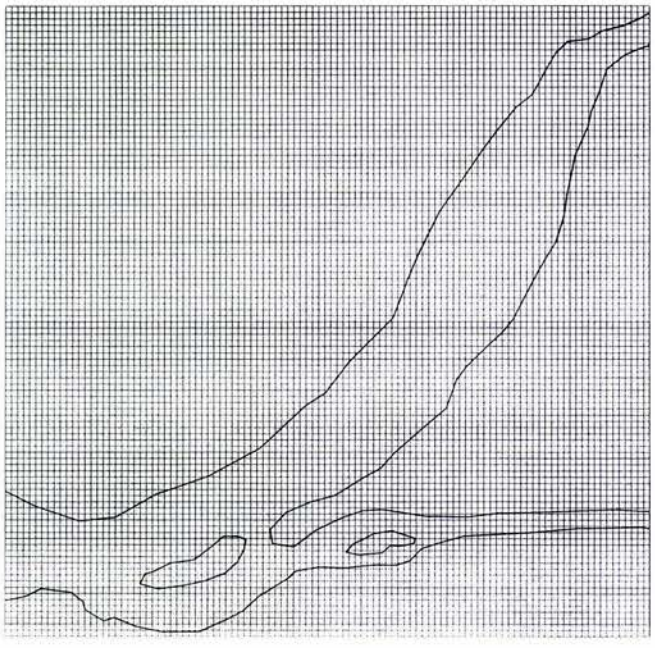


Figura 5.1: Malla de cálculo para la primera modelación

vez conseguido esto se realizó una modelación de 30 días.

La condición de frontera impuesta en la salida es que por el lado del río Carrizal la elevación de la superficie libre del agua es $H = 14{,}9$ metros y por el lado del río Mezcalapa de $H = 15{,}2$ metros, se buscó con los niveles anteriores aproximarse a los medidos en campo. Al ser en el río Carrizal mas bajo el nivel del agua en la salida se produce un gradiente hidráulico más fuerte y favorece que un mayor gasto circule por él.

La elevación de la superficie libre del agua se muestra en la figura 5.3, se buscó que los niveles se aproximaran lo más posible a elevaciones registradas en las escalas colocadas a lo largo de la zona de la bifurcación, sin embargo no fue posible hacerlas concordar completamente debido a que se encontraron algunas características que hacen diferente la topografía empleada para los cálculos de la real. Otro fenómeno que no se logró reproducir con esta modelación fue la distribución de gastos, de acuerdo con la figura 1.1, cuando en el río Mezcalapa llega un gasto $Q_M = 600m^3/s$ por el Carrizal deriva de un 55 % a un 60 % y el resto por el río Samaria. En la modelación realizada resulta que sólo un 43.5 % deriva por el río Carrizal.

Con las condiciones descritas anteriormente se encontró la condición estable de funcionamiento para la bifurcación con el gasto mencionado, es decir, que después de un

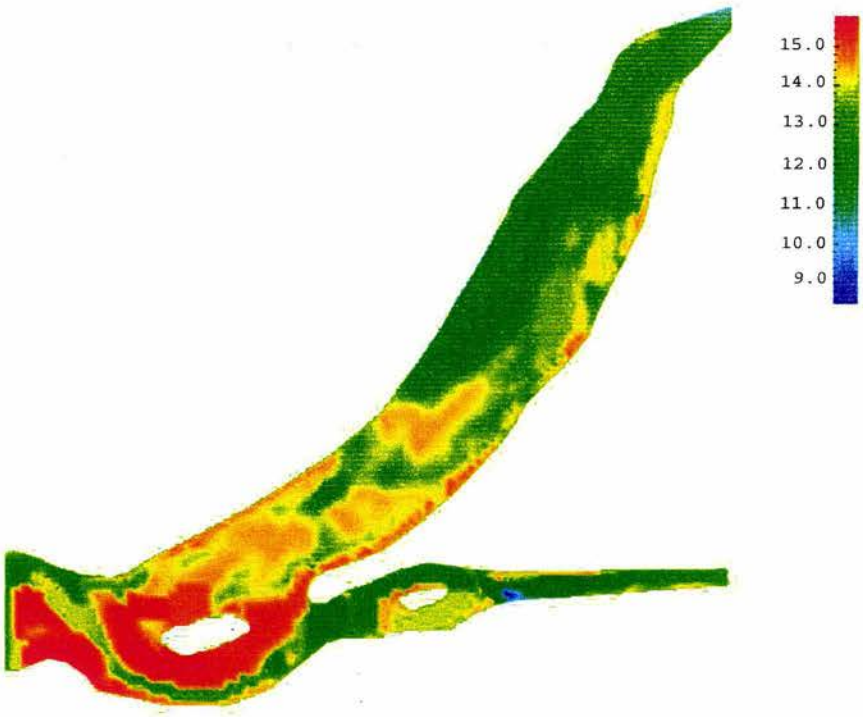


Figura 5.2: topografía del fondo para la primera modelación, en metros sobre el nivel del mar

intervalo de tiempo de cálculo, ya no variaban las velocidades o los tirantes. La dirección del flujo se representa en la figura 5.4.a, en la figura 5.4.b se muestra la magnitud de la velocidad. En este primer cálculo se consideró el fondo fijo, en los cálculos posteriores se considera móvil.

Una vez encontradas las condiciones iniciales, se realizó una modelación de 30 días durante los cuales el hidrograma de entrada diario fue el mostrado en la figura 5.5. La forma del hidrograma se debe a que aguas arriba de la bifurcación se encuentra una hidroeléctrica que genera durante aproximadamente 4 horas todas las noches, por lo cual se libera un gasto mayor durante ese tiempo.

Después de los 30 días de simulación la erosión y depósitos desarrollados se muestran en la figura 5.6. Se observa que con las condiciones empleadas hay poco transporte de sedimento a lo largo del río Samaria, donde las velocidades son muy bajas prácticamente no se modificó el fondo. En la zona de la bifurcación se tienen velocidades más grandes y hay procesos de erosión y sedimentación, con las condiciones dadas el canal profundo que parte del río Mezcalapa inicia a llenarse de sedimento como puede verse

en la figura 5.6.

En la frontera de aguas arriba el flujo está entrando paralelo al eje x, como se puede ver en la figura 5.4.a. En el prototipo ésta frontera se encuentra después de una curva, por lo cual el flujo viene más cargado hacia la margen derecha, y al chocar con ésta la corriente principal se dirige hacia la entrada del río Carrizal. El hecho de que el modelo no represente bien esta condición puede ser la causa de que tenga un comportamiento diferente al del prototipo. Sería conveniente representar un tramo mayor aguas arriba pero no existe información de campo.

Con esta primera modelación no se obtuvieron los resultados deseados y como se mencionó anteriormente, la parte del río Samaria después de la bifurcación hasta los puentes no tiene transporte, la malla de cálculo se puede reducir exclusivamente a la zona de la bifurcación en una segunda simulación. Con la nueva malla se logró que el flujo entrara con una componente dirigida hacia la margen derecha para representar el el fenómeno de que la corriente choque contra la margen izquierda y luego derive hacia la margen derecha en la entrada del río Carrizal.

5.2. Segunda modelación: Zona de la bifurcación

En esta simulación la malla se reduce al área mostrada en la figura 5.7, En este caso la malla es de 60 celdas en la dirección x y 34 celdas en la dirección y, con dimensiones de 50 × 50 metros. La topografía es la misma que se empleó en la primera simulación. Ahora el gasto se consideró constante $Q = 1200m^3/s$, pero se aumentó respecto de la primera modelación para acelerar los procesos de erosión y sedimentación.

La condición de frontera en la salida es que en el río Samaria se tiene el nivel de la superficie libre del agua $H = 17m$ y en el río Carrizal $H = 16{,}84m$, es decir, hay un desnivel de 0.16 m entre los dos ríos. Con estas condiciones la distribución de gastos es que por el río Samaria circula un 51 % y por el Carrizal un 49 %, esto es aproximadamente lo que sucede en la realidad, como se puede ver al comparar estos porcentajes con la figura 1.1. Como se hizo con la primera modelación, inicialmente se determinó el funcionamiento de la bifurcación con las condiciones de frontera especificadas manteniendo el fondo fijo, y una vez encontradas las velocidades y niveles del agua para las condiciones anteriores se realizó una simulación de 30 días permitiendo que el fondo se moviera. El campo de velocidades calculado se muestra en la figura 5.9.a y la magnitud de las velocidades en la figura 5.9.b.

Después de 30 días de simulación las distribución de gastos quedo de la siguiente

manera: por el río Carrizal deriva un 48 % y por el Samaria un 52 %. Con estos resultados se tiene que la capacidad del río Carrizal disminuyó en 1 %, contrario a lo que sucede en la realidad. Las zonas de erosión y depósito son muy similares a las calculadas en la primera simulación, es por ello que no se incluye aquí esa figura. Aunque en esta segunda simulación se obligó que en la frontera de entrada la corriente entrara con una componente dirigida hacia la margen izquierda nuevamente no se logró reproducir el fenómeno del aumento de gasto en el río Carrizal con el paso del tiempo.

En las dos modelaciones anteriores se ha empleado la topografía disponible del lugar, en ella se aprecia un canal más profundo en que viene del río Mezcalapa y desemboca en el río Carrizal. Una nueva prueba a realizar es generar una topografía artifical, con un canal más profundo que el que tiene actualmente la topografía del fondo, y verificar si con estas condiciones se logra que con el paso del tiempo aumente el gasto que pasa por el río Carrizal.

5.3. Tercera modelación: topografía artificial con un canal profundo dirigido hacia el río Carrizal

Para esta modelación se deja la misma geometría de la bifurcación y por lo tanto se emplea la misma malla, pero el fondo se fija en la cota de $z = 15$ metros, en la figura 5.8 se muestra la topografía generada. El canal profundo está en la cota $z = 13$ metros, también se deja un canal profundo pegado a la margen izquierda que llega al río Samaria. Lo anterior es para dejar una zona con una capacidad hidráulica menor en el lado izquierdo de la isla y ver si esto ayuda a producir sedimentación en esta zona.

Para esta modelación se emplea un gasto constante de $Q = 600m^3/s$, los niveles de la superficie libre del agua en las dos salidas es la misma y es $H = 15{,}5$ metros. Con las condiciones anteriores por el río Carrizal deriva un 51 % el gasto y por el Samaria un 49 %, el campo de velocidades calculado y las magnitudes se muestran en la figura 5.10.

En este caso se simularon también 30 días, después de este tiempo el gasto que circula por el río Carrizal es 45 % y por el río Samaria es 55 %. Nuevamente el gasto del río Carrizal disminuye con el paso del tiempo.

El canal profundo que desemboca en la entrada del río Carrizal tampoco es la causa de que esté aumentando el gasto en dicho río, en la figura 5.11 se muestran las zonas de erosión y depósito, se observa que en el río Samaria hay más erosión que en el río Carrizal, es por ello que aumenta la capacidad del río Samaria, y con ello el gasto que circula por él.

5.4. Cuarta modelación: mayor gradiente hidráulico en el río Carrizal

Empleando la misma malla se consideró que la topografía es plana, sin los canales profundos empleados en la simulación anterior, la cota del fondo es $Z = 14$ metros, el nivel de la superficie libre del agua en los dos ríos de salida es diferente, en el río Samaria es $H = 15{,}5$ metros y en el río Carrizal $H = 15{,}3$ metros. Con estos niveles el gradiente hidráulico en el río Carrizal es mayor. El gasto nuevamente es $Q = 600m^3/s$.

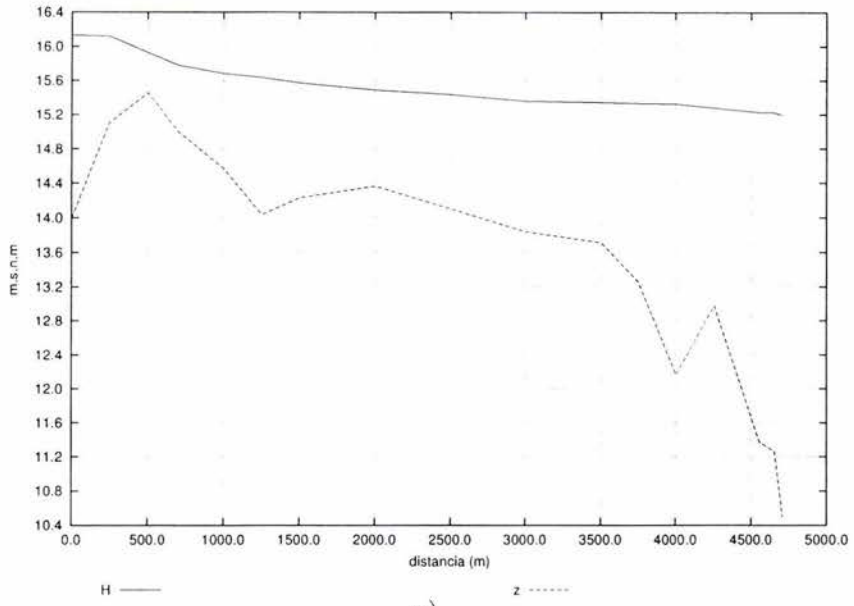
Con las condiciones descritas en el párrafo anterior, el gasto que deriva por el río Carrizal es 57 % y por el Samaria 43 %, las velocidades calculadas se muestran en la figura 5.13. Después de 30 días de simulación la distribución de gastos es 62 % por el Carrizal y 38 % por el río Samaria. En la figura 5.12 se observa que en el río Samaria hay zonas de depósito, aunque no es en la magnitud que se esperaba, ya que en el prototipo se produjeron grandes bancos de depósito en esta zona. En la entrada del río Carrizal el fondo se erosiona en la zona donde es estrecha la sección transversal.

5.5. Resultados

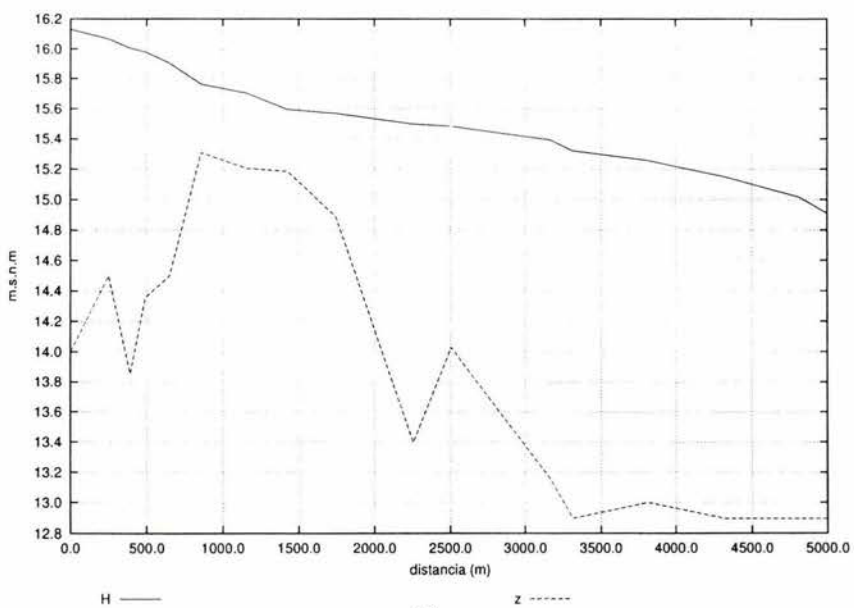
Los porcentajes del gasto que proviene del río Mezcalapa y desemboca en el río Carrizal antes y después de las simulaciones se muestran en el cuadro 5.1, se tiene que sólo dando un gradiente hidráulico más grande en el río Carrizal, como se hizo en la cuarta simulación, se logra que con el paso del tiempo aumente el gasto que circula por dicho río.

Cuadro 5.1: Distribución de los gastos por el río Carrizal antes y despues de 30 días de simulación

Modelación	Q inicial %	Q final %
Primera	43.5	43.5
Segunda	49.0	48.0
Tercera	51.0	45.0
Cuarta	57.0	62.0

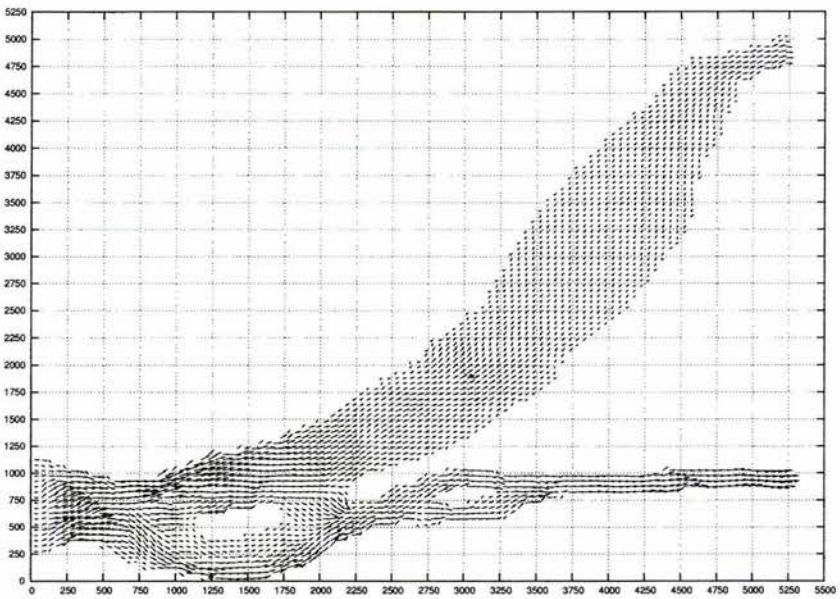


a)

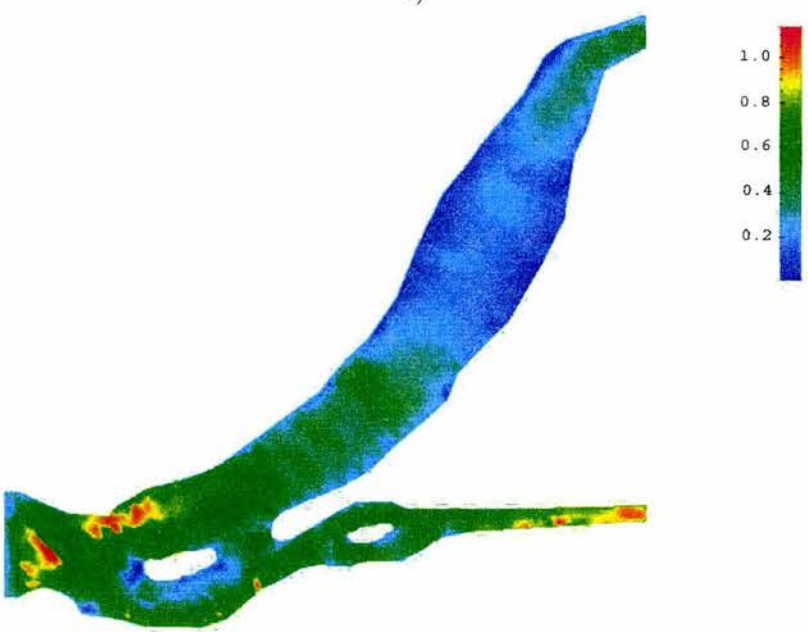


b)

Figura 5.3: Perfiles en los ríos a) Samaria y b) Carrizal para un gasto $Q = 600m^3/s$ (Primera modelación).



a)



b)

Figura 5.4: Velocidades para $Q = 600m^3/s$. a) Campo de velocidades. b) Magnitudes de las velocidades, en m/s.

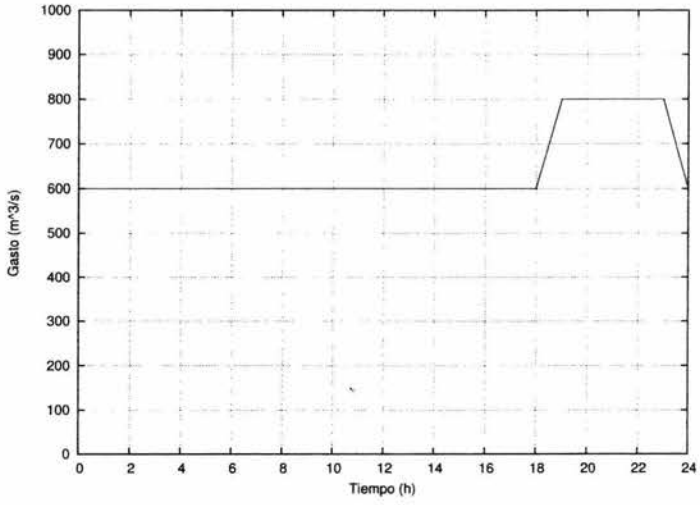


Figura 5.5: Hidrograma diario de entrada empleado para la primera simulación

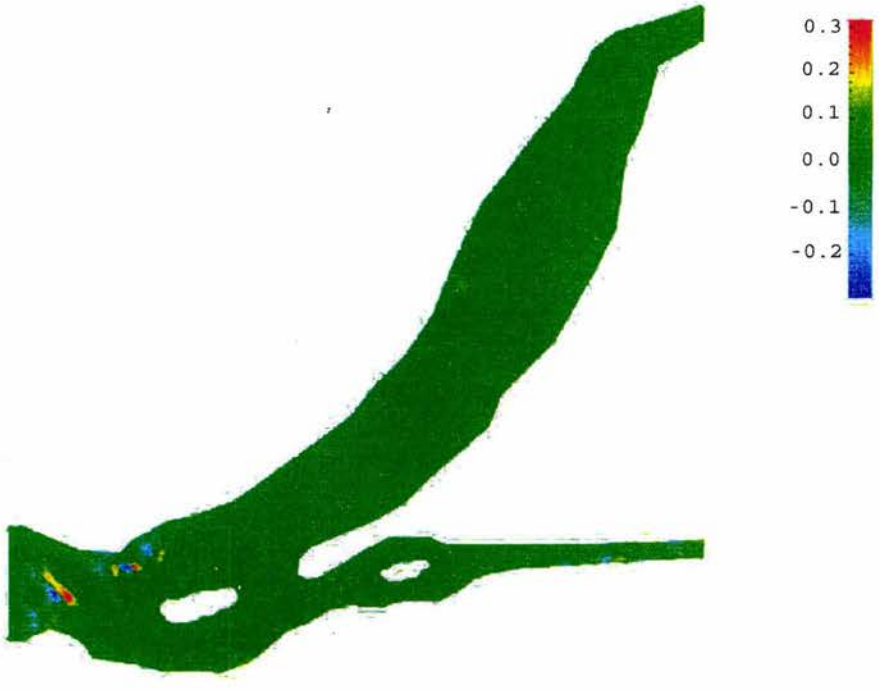


Figura 5.6: Erosión y depósito producidos con la primera simulación, en metros, el signo negativo representa erosión y el positivo depósito

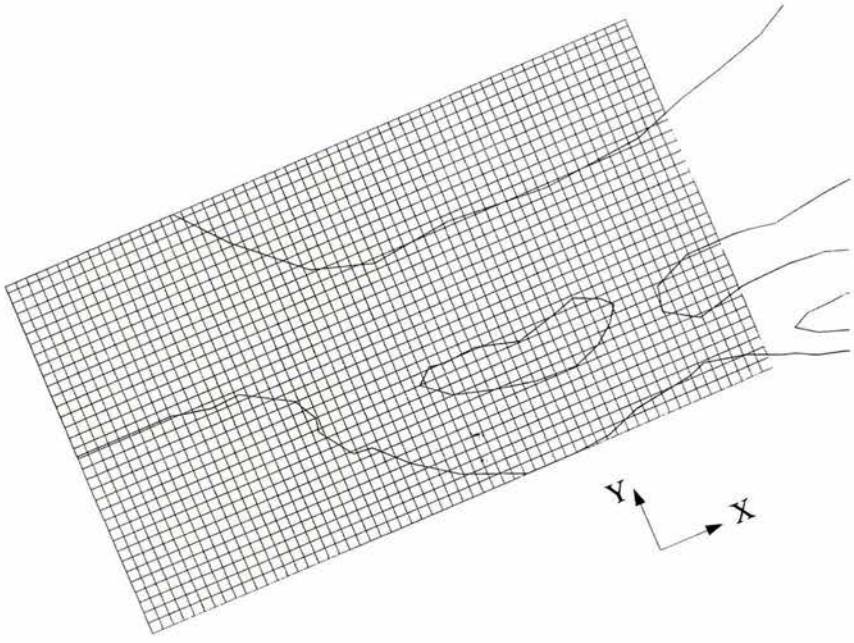


Figura 5.7: Malla de cálculo para la segunda simulación

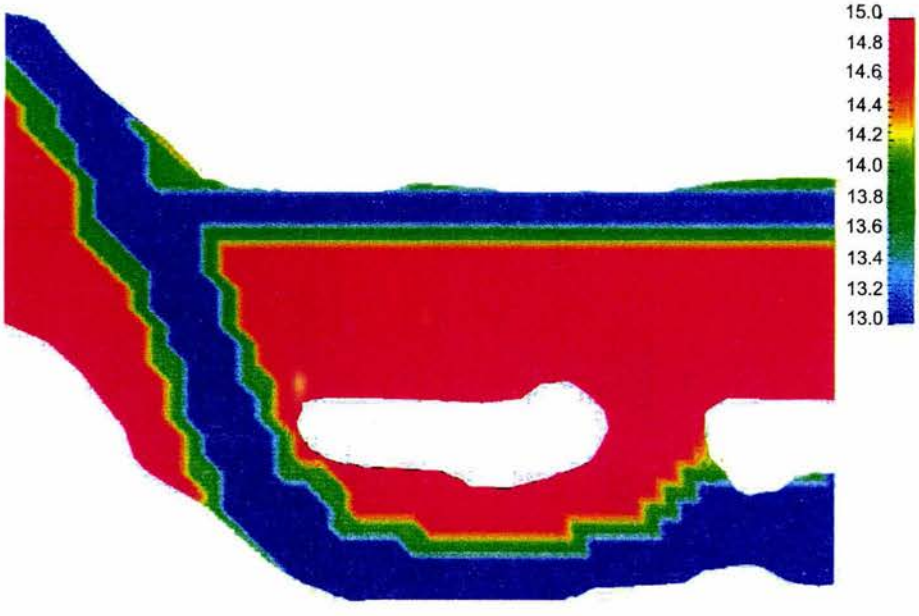
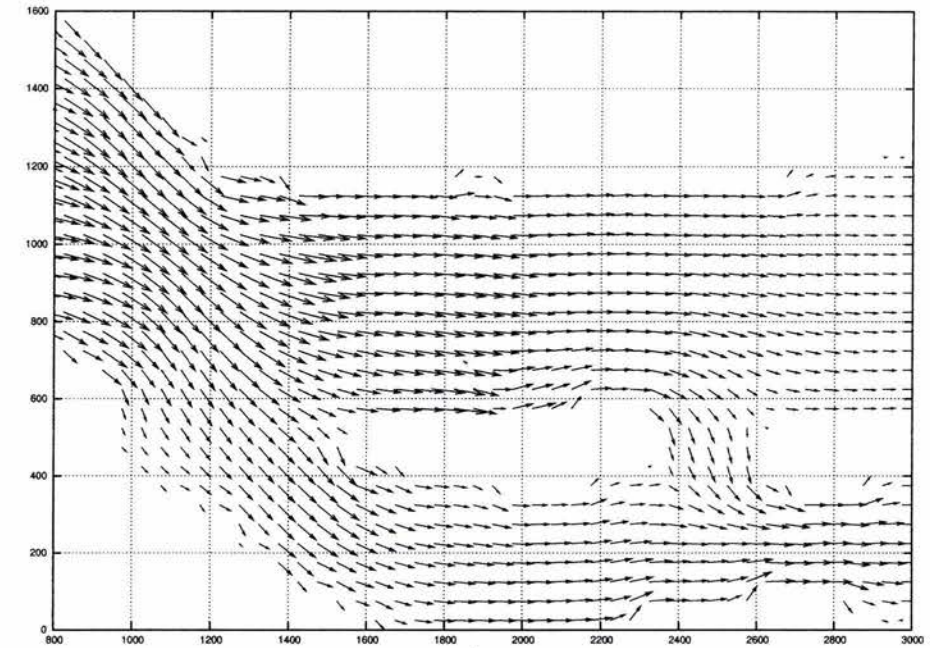
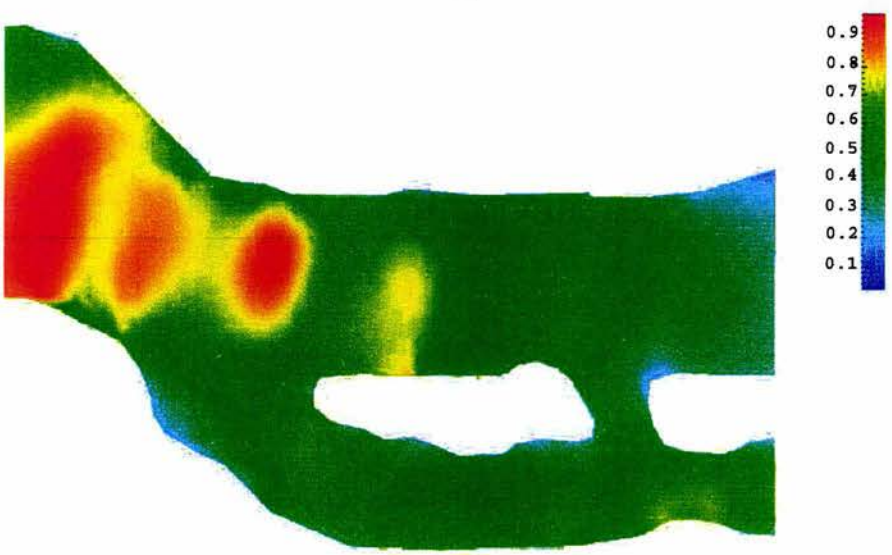


Figura 5.8: Topografía del fondo empleada para la tercera simulación, en metros sobre el nivel del mar

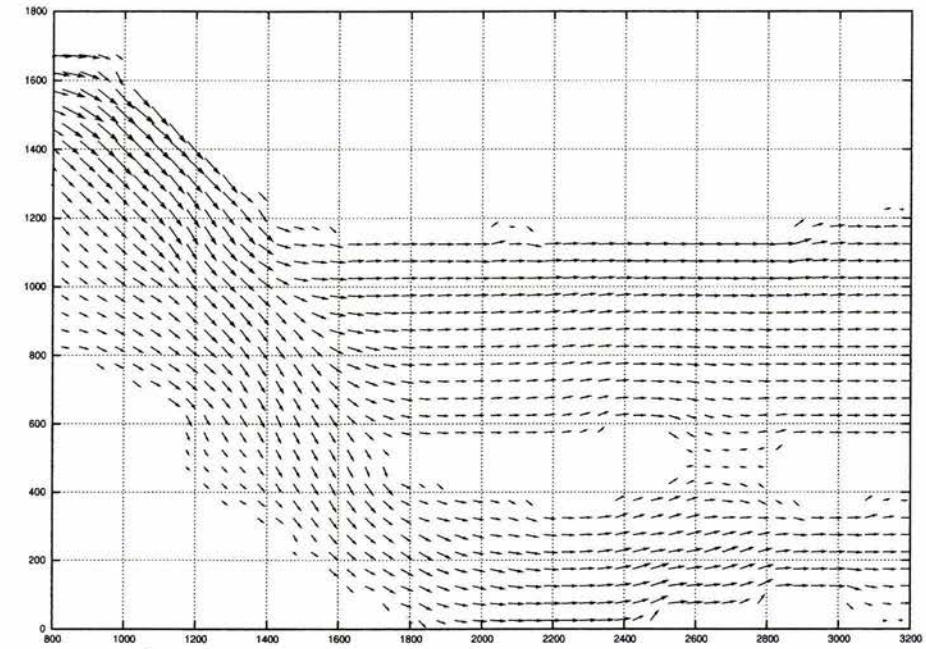


a)

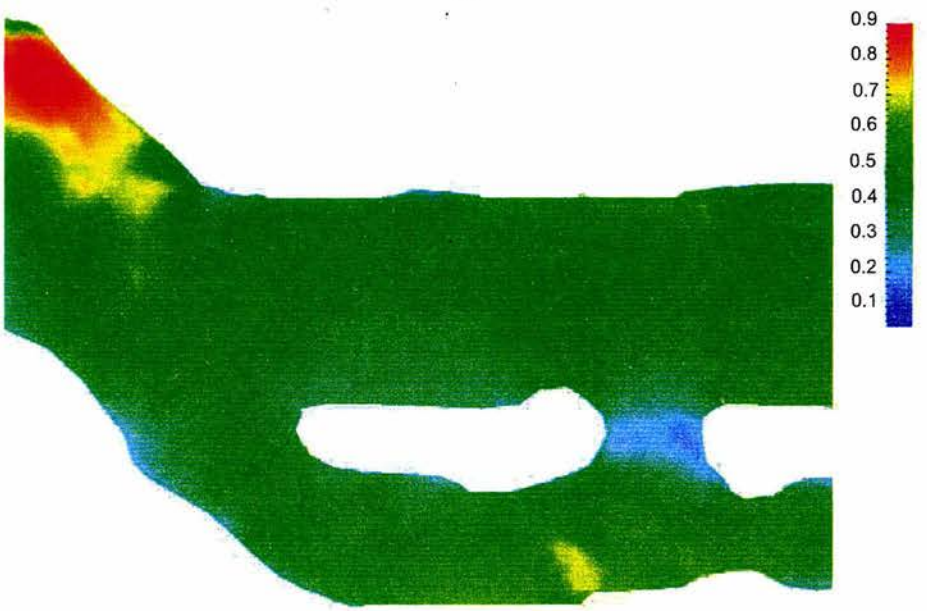


b)

Figura 5.9: Velocidades para $Q = 1200m^3/s$ de la segunda modelación. a) Campo de velocidades. b) Magnitud de las velocidades, en m/s.



a)



b)

Figura 5.10: Velocidades para $Q = 600m^3/s$ de la tercera modelación. a) Campo de velocidades. b) Magnitud de las velocidades, en m/s.

Figura 5.11: Erosión producida con lá cuarta simulación, en metros, los valores positivos indican depósito y los negativos erosión

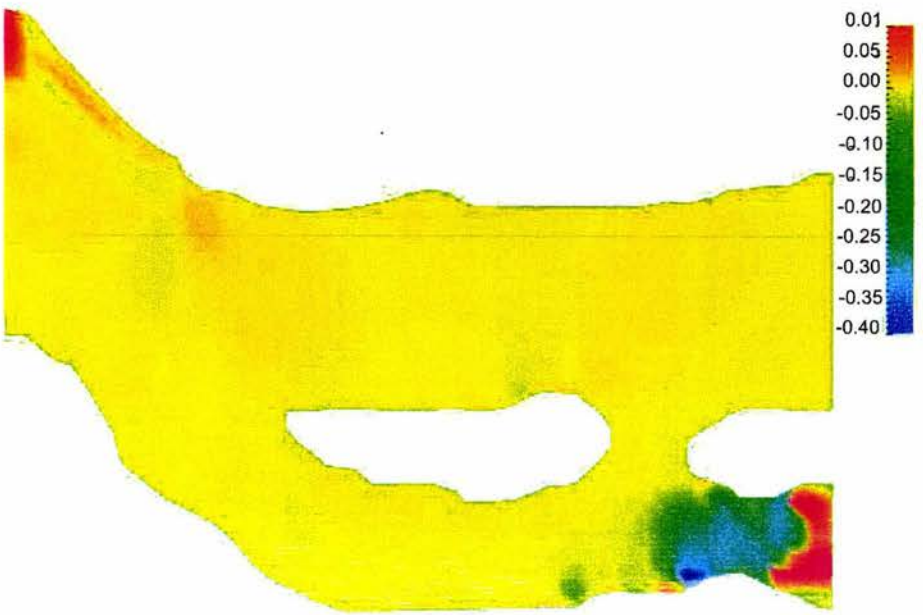
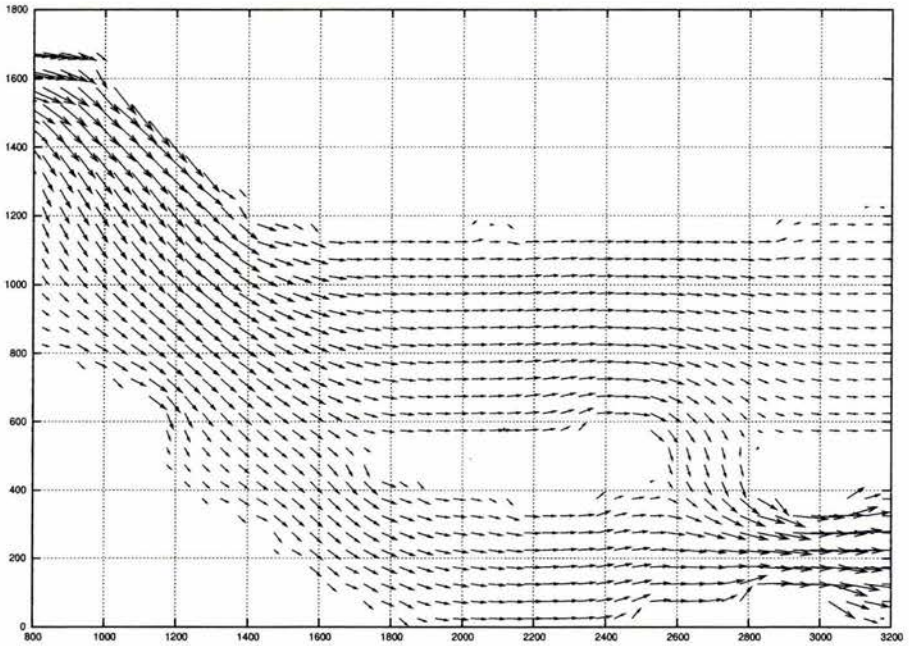
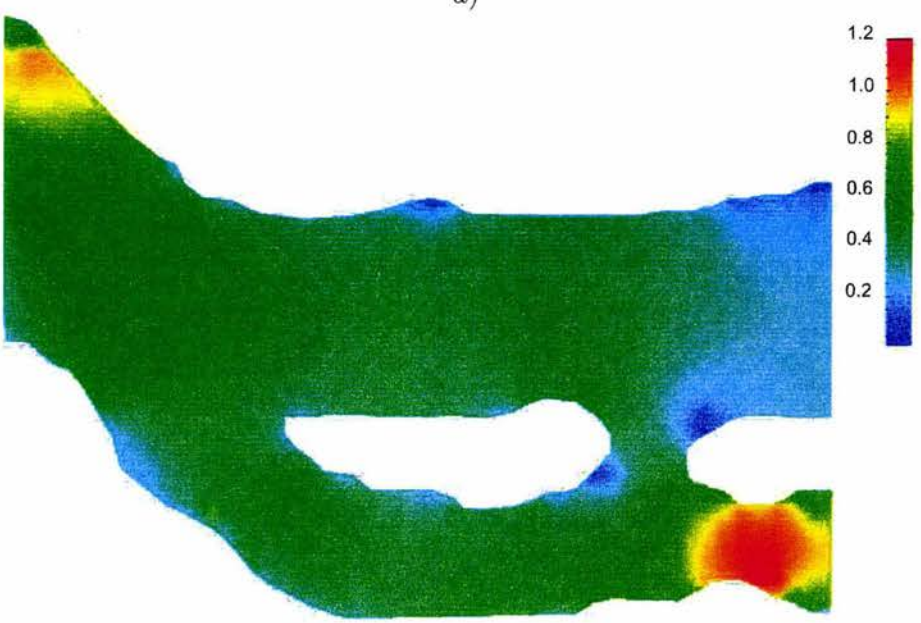


Figura 5.12: Erosión producida con la cuarta simulación, en metros, los valores positivos indican depósito y los negativos erosión



a)



b)

Figura 5.13: Velocidades para $Q = 600m^3/s$ de la cuarta modelación. a) Campo de velocidades. b) Magnitud de las velocidades, en m/s.

Capítulo 6

Conclusiones

6.1. Resultados de las simulaciones

El modelo bidimensional implantado en esta tesis permitió realizar cuatro modelaciones del flujo y arrastre de sedimentos en la bifurcación, solo en la última de ellas se logró representar el aumento del gasto líquido con el tiempo del río Carrizal. A continuación se describen los resultados obtenidos en cada una de las simulaciones realizadas.

La primera simulación fue con una malla que abarca desde la zona de la bifurcación, en el río Mezcalapa, hasta los puentes Samaria I y II ubicados en el río Samaria. Esta fue la única simulación realizada con un hidrograma diario (figura 5.5), todas las demás simulaciones se realizaron con gasto constante. Debido a que la malla fue muy grande el cálculo tomó mucho tiempo, los 30 días se simularon en 36 horas de cómputo. Al final se obtuvo que después de la bifurcación, en los ríos Samaria y Carrizal hay muy poco o nulo transporte de sedimento. El canal profundo que va de la entrada del río Mezcalapa al río Carrizal tiende a taparse, por lo cual no se representó el aumento de gasto en el río Carrizal.

En la segunda modelación la malla de cálculo se restringió a la zona de la bifurcación, esto permitió reducir el tiempo de cálculo, en estas condiciones 30 días se simularon en cuatro horas. El nivel de la superficie libre del agua en la salida del río Carrizal se especificó por debajo del nivel en la salida del río Samaria para favorecer el flujo por el primero. Con las condiciones anteriores tampoco se logró que aumentara el gasto que deriva por el río Carrizal.

Con la tercera modelación se sustituyó la topografía de los ríos por otra artificial; se dejó el fondo plano en la cota $z = 15$ metros, con un canal más que parte de la margen izquierda del río Mezcapala y desemboca en la entrada del río Carrizal, también se generó un canal en la margen izquierda, hacia el río Samaria pero menos ancho. El

canal que desemboca en el río Carrizal tiene el fin de favorecer el flujo por dicho río, al ser mayor el gasto en él las velocidades en el río Samaria serían menores, lo que favorecería el depósito en esta zona. Después de los 30 días simulados el gasto que deriva por el río Carrizal disminuyó, por lo cual tampoco se logró representar el fenómeno deseado.

En la última modelación se dejó el fondo plano con la cota $z = 14$ metros, en este caso en nivel de la superficie libre del agua en la salida del río Carrizal fue 0.3 metros por debajo del nivel del agua en el río Samaria para generar una gradiente hidráulico mayor y favorecer el flujo por el río Carrizal. Después de 30 días, con un gasto constante de 600 m^3/s el gasto que deriva por el río Carrizal aumentó en un 5 %.

Con la cuarta modelación se logró representar el aumento del gasto que circula por el río Carrizal, sin embargo, como puede verse en la figura 5.12, el sedimento depositado en la entradada del río Samaria es poco comparado con el depósito que se ha generado en el prototipo, sin embargo el depósito calculado se produjo con 30 días de simulación, tal vez con un tiempo mayor se logre representar los bancos de deposito que hay en la realidad.

La condición que logró aumentar el gasto en el río Carrizal con el paso del tiempo fue que en éste existiera un gradiente hidráulico mayor que en el río Samaria, el gradiente menor en este último río puede deberse a que esté siendo remanzado por algún obstáculo aguas abajo de la bifurcación. Otra posiblilidad es que el río Carrizal tenga una pendiente del fondo mayor, lo que da origen también a un gradiente hidráulico mayor.

Las curvas en el río Mezcalapa que hay antes de llegar a la zona de la bifurcación producen que el flujo esté más concentrado en la margen izquierda, para las modelaciones realizadas no se dispuso de información aguas arriba por lo cual no se pudieron prepresentar las curvas ni las características topográficas, por ello se manipularon artificialmente las condiciones de frontera para tratar de reproducir el flujo por la margen izquierda en la entrada a la bifurcación. Los gastos considerados fueron constantes en tres de los casos estudiados y las simulaciones se realizaron durante 30 días, sin embargo en la realidad el proceso de sedimentación y erosión está determinado también por las variaciones del gasto con el paso de los meses y los años, en épocas de avenidas el gasto en el río Mezcalapa a sido superior a los 2000 m^3/s y en estiaje el gasto es alrededor de 400 m^3/s.

6.2. Resultados de la implantación del modelo

En relación a la implementación de este modelo y en la solución de las ecuaciones hidrodinámicas, los puntos destacables son los siguientes:

1. En un principio se intentó resolver el sistema formado por la ecuación de conservación de masa (2.7) y las de cantidad de movimiento (2.19) y (2.20) ya que estas son las tratadas comúnmente en la literatura (referencias 1, 2 y 3), en ellas las variables a determinar son las velocidades en las direcciones x y y, y el tirante h. Con el modelo implementado se determinó que las ecuaciones anteriores trabajan bien cuando es completamente horizontal, sin embargo, cuando se probaron con la topografía de la bifurcación presentaron inestabilidades, lo anterior se determinó dejando la superficie libre del agua completamente horizontal y sin que entrara gasto a la malla de cálculo, con estas condiciones no debería existir movimiento, pero por las características del método de solución con el paso del tiempo comenzaba a haber movimiento en la dirección del gradiente del fondo; revisando el esquema de solución empleado se determinó que dicho movimiento era de esperarse. Por lo anterior se manipularon las ecuaciones para que manejaran en lugar del tirante h, el nivel total de la superficie del agua H, con lo que el sistema a resolver quedó conformado por las ecuaciones (2.24) y (2.25).

2. El esquema implantado no permite números de Courant grandes, en las referencias se indica que dicho número no debe ser mayor a uno, sin embargo con las simulaciones realizadas se tiene que para valores de $C_n > 0{,}6$ ya se presentaban inestabilidades. El valor empleado para los cálculos realizados fue de $C_n \approx 0{,}3$.

3. Con el esquema planteado la ecuación de conservación de la masa no se cumple completamente, en cálculos realizados con gasto constante hay diferencias en el gasto de $\pm 8\,\%$ en las secciones de salida respecto al gasto de entrada.

En el modelo desarrollado se empleó la ecuación de transporte total de fondo propuesta por Engelund y Hansen, sin embargo puede emplearse cualquier otro método que permita evaluar dicho transporte en función de las características del material y de las condiciones hidrodinámicas del flujo, es decir, que no necesite gráficas o tablas auxiliares.

Referencias

1. Carrillo Sosa J.J. (2000) *Modelo matemático bidimensional para el cálculo de la hidrodinámica de cuerpos de agua (flujo bidimensional)*, Tesis de Maestría DEPFI-UNAM, México D.F.

2. Chaudhry, M.H. (1993) *Open channel flow*, Prentince Hall, New Jersey

3. García R. y Kahawita (1986) *Numerical Solution of the Saint-Venant Equations with the MacCormack Finite-Diference Scheme*, International Journal for Numerical Methods in Fluids, Vol. 6, pp 259 -274.

4. Gracia Sánchez J. (1994) *La erosión de suelos arenosos, causados por flujos superficiales con tirantes pequeños (overland flow)*, CENAPRED Coordinación de investigación RH/02/94, México D.F.

5. Jiménez Castañeda A.A. (1999) *Cálculo de la hidrodinámica en cuerpos de agua con fronteras irregulares*, Tesis de Doctorado DEPFI-UNAM, México D.F.

6. Maza Álvarez J.A. García Flores M. (1996) *Transporte de sedimentos*, Capítulo 10 del manual de ingeniería de ríos. Series del Instituto de Ingeniería 584, México D.F.

7. Mendoza Reséndiz A. (2001) *Comparación de un modelo numérico y otro físico de la bifurcación Mezcalapa Samaria Carrizal*, Tesis de licenciatura FI-UNAM, México D.F.

8. Peric M. Ferziger J.H. (2002) *Computational methods for fluid dynamics*, Springer Verlang, Berlin

9. Rajaratnam, N.y Nwachukwu, B. (1983) *Flow near groin-like structures*, ASCE Journal of Hydraulic Engineering, Vol 109, No. 3 pp. 463-480

10. Rivera Trejo J.G.F. (2002) *Modelo Bidimensional de Fondo Movil*, Tesis de Doctorado DEPFI-UNAM, México D.F.

11. Sánchez Bribiesca J.L., Franco V. (2001) *Modelo Físico del río Samaria en la bifurcación Carrizal-Samaria en el estado de Tabasco* Informe dei Instituto de Ingeniería, UNAM para CNA.

12. Sotelo Ávila G. (1996) *Hidráulica General.* Editorial Limusa, México D.F.

13. Tingsanchali T. y Maheswuaran S. (1990) *2-D Depth-averaged Flow Computation Near Groyne*, ASCE, Journal of Hydraulic Engineering, Vol 116, No. 1 pp 71-85

Apéndice: Programa para cálculo de flujo bidimensional con fondo móvil

El programa se escribió en lenguaje C++ empleando programación orientada a objetos, es por ello que cada objeto está en un archivo separado. A continuación se describe brevemente la tarea de cada objeto.

1. Cell
   Este objeto representa una celda de cálculo, contiene la informacion necesaria para los cálculos, como las velocidades, tirantes, cota del fondo, coeficiente de rugosidad, etc.

2. Grid
   Grid es un objeto que agrupa a todas las celdas de la malla, sabe la ubicación de cada una de ellas y detecta cuando a las que se encuentran en una frontera para el trato diferente que requieren, como se describió en el capítulo 3.

3. EvalEQ
   Este objeto evalúa los términos $\mathbf{U_s}$, $\mathbf{E_{s1}}$, $\mathbf{E_{s2}}$, $\mathbf{F_{s1}}$ $\mathbf{F_{s2}}$ y $\mathbf{S_s}$ de una celda, como se especifica en las ecuaciones de la (4.16) a la (4.21)

4. DiferenceE y DiferenceF
   Este objeto aplica los operadores de diferencias explicados en el apartado 3.1 para los vectores $\mathbf{E_{s1}}$, $\mathbf{F_{s1}}$, $\mathbf{E_{s2}}$, $\mathbf{F_{s2}}$

5. MacCormack
   Este objeto implementa el métido de MacCormack, emplea a todos los objetos anteriores

6. ReadGrid
   Este objeto tiene el fin de leer todos loar achivos de datos

7. Vec4
   Este objeto define un vector de 4 elementos, su funcion es que permite sumarse restarse y multiplicarse como con los símbolos convensionales $+$, $-$, $\times$

8. Tablesign
   Este objeto tiene la función de saber en qué dirección se aplican los operadores de diferencias definidas en el apartado 3.1, de acuerdo ai cuadro 3.1

9. Hidrogram
   Este objeto lee un archivo con los datos de un hidrograma y tiene la funcion de proporcionar el gasto para el tiempo que se le especifique

El programa principal está en el archivo MainMc.cpp, a continuación se listan los archivos del programa.

archivo: mainMc.cpp:

```
#include "MacCormack.h"
#include "Hidrogram.h"
#include <iostream>

using namespace std;

int main(int argc, char **argv)
{

   if( argc != 2)
   {
      cerr << "uso: mac.exe entrada.txt\n";
      return 0;
   }

   ifstream in( argv[1] );
   if ( !in )
   {
      cerr << "no se puede abrir " << argv[1] << '\n';
      return 0;
   }

   string n1, n2, tmp;
   double t0, tf, dt, dtw;

   in >> n1;
   in >> n2;
   in >> tmp >> t0;
   in >> tmp >> tf;
   in >> tmp >> dt;
   in >> tmp >> dtw;

   in.close();

   ReaderCell rd( n1.c_str() );

   Grid grd( rd );

   cerr << "malla construida \n";

   Hidrogram hdr( n2.c_str() );

   cerr << "hidrograma construido\n";

   MacCormack mc( grd, hdr );

   cerr << "Mackormac construido, calculando...\n";

   mc.calc( t0, tf, dt, dtw );

   cerr << "calculo terminado \n";
   grd.writeM();

   return 0;

}
```

archivo: Cell.h:

```
// clase que contiene los atributos de una celda de calculo
#ifndef CELL_H
#define CELL_H
#include "Vec4.h"

class Cell
{
 public:
   double u,v,h;   // variables de velocidad y tirante
   double n,z;   // coeficiente de rugosidad y cota del fondo
   double uc, vc, hc, zc;   // variables empleadas para calculo
   int tc, tf;   // tipo de celda y tipo de frontera
   double sx, sy;   // pendientes en ambas direcciones;
   double k;   // coefciente de terminos const. para qs.
   double ht, htc; // nivel total del agua

   //constructor
   Cell(const double u, const double v, const double ht,
      const double n, const double z, const double kk,
      const int tc);

   // metodo para recibir al vector U y traducirlo a
   // las variables primitivas
   void getU(Vec4 &v);


   // metodo para hacer la operacion equivalente a promediar
   // los vectores U pedichos y corregidos el resultado se
   // almacena en las variables primarias u, v y h y z
   void prom();
};
#endif
```

archivo: Cell.cpp :

```
#include "Cell.h"

//constructor, recibe datos iniciales y datos que describen
// las caracteristicas de la celda.
Cell::Cell(const double uu, const double vv, const double hht,
   const double nn, const double zz, const double  kk,
   const int tcc):u(uu), v(vv), ht(hht), n(nn), z(zz), k(kk), tc(tcc)
{
   uc = u;
   vc = v;
   htc = ht;
   zc = z;

   h = ht - z;
   hc = h;
   tf = 0;
}

//metodo para recibir un vector U y transformarlo a las
// variables primitivas u, v, h.
void Cell::getU(Vec4 &vU)
{
   // cada vez que se asigna un nuevo valor U se pasan
   // se pasan las variables de calculo a variables
   // de la celda.
   h = hc;
   u = uc;
   v = vc;
   z = zc;
   ht = htc;

   hc = vU(0);
   uc = vU(1) / hc;
   vc = vU(2) / hc;
   zc = vU(3);
   htc = hc + zc;
}

// metodo correspondiente a calcular el valor
// ultimo de las variables primitivas, empleando
// el valor predicho y corregido de U.
void Cell::prom()
{
   double hp, uhp, vhp, hco, uhc, vhc;
```

```
   hp = h;
   hco = hc;

   uhp = u * h; // variable predicha
   uhc = uc * hco; // variable corregida

   vhp = v * h;
   vhc = vc * hco;

   h = 0.5 * ( hp + hco);
   u = 0.5 * (uhp + uhc) / h;
   v = 0.5 * (vhp + vhc) / h;
   z = 0.5 * (z + zc);
   ht = h + z;

   hc = h;
   htc = ht;
   uc = u;
   vc = v;
   zc = z;
}
```

archivo: Diference.h:

```
// clase base para calcular diferencias de E y F
#ifndef DIFFERENCE_H
#define DIFFERENCE_H

#include "EvalEQ.h"
#include "Grid.h"
#include "TableSign.h"

class Difference
{
 protected:
   EvalEQ eval;
   TableSign ts;
   Difference()
   {
   };
   /// prediccion
   virtual Vec4 difP(Grid &gr, const int h, const int w,
         const int iter) = 0;
   /// correcion
   virtual Vec4 difC(Grid &gr, const int h, const int w,
         const int iter) = 0;

};
#endif
```

archivo: DiferenceE.h:

```
/// implementacion de la clase de diferencias para
// el vector E
#ifndef DIFFERENCE_E_H
#define DIFFERENCE_E_H

#include "Difference.h"

class DifferenceE: public Difference
{
 private:
   //evalua la condicion de frontera 3 de la tabla 3.2
    Vec4 fronE3(Grid &gr, const int h, const int w,
         const int iter);

   //evalua la condicion de frontera 4 de la tabla 3.2
    Vec4 fronE4(Grid &gr, const int h,
      const int w, const int iter);

 public:
    DifferenceE();
   // diferencia de prediccion
   Vec4 difP(Grid &gr, const int h, const int w, const int iter);

   // diferencia de correcion
   Vec4 difC(Grid &gr, const int h, const int w, const int iter);
};

#endif
```

archivo: DiferenceE.cpp:

```
#include "DifferenceE.h"

DifferenceE::DifferenceE():Difference()
{
}

// frontera en celda tipo 3
Vec4 DifferenceE::fronE3(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   double u,v,hh, u_2;
   double v1, v2, v3, v4;

   u = cel1.uc;
   v = cel1.vc;
   hh = cel1.hc;
   u_2 = u * u;

   double uv = u * v;

   v1 = 2.0 * hh * u;
   v2 = 0.0;
   v3 = 2.0 * hh * uv;

   v4 = 2.0 * cel1.k * u_2 * u_2 * u;
   v4 /= sqrt(hh);

   Vec4 vecr(v1, v2, v3, v4);

   return vecr;
}

// frontera en celda tipo 4
Vec4 DifferenceE::fronE4(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   double u,v,hh, u_2;
   double v1, v2, v3, v4;

   u = cel1.uc;
   v = cel1.vc;
   hh = cel1.hc;
   u_2 = u * u;

   double uv = u * v;

   v1 = -2.0 * hh * u;
   v2 = 0.0;
   v3 = -2.0 * hh * uv;

   v4 = -2.0 * cel1.k * u_2 * u_2 * u;
   v4 /= sqrt(hh);

   Vec4 vecr(v1, v2, v3, v4);
   return vecr;
}

/// fase de prediccion
Vec4 DifferenceE::difP(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   int tf = cel1.tf;   // tipo de frontera
   int sig = ts.dirpE(iter); // direccion de la diferencia
   double hg = cel1.hc * 9.81;   // tirante en la celda actual

   /// si no esta en una frontera, o si esta en la
   // frontera este pero no interfiere
   if( tf == 0 || tf == 2 || tf == 1 || (tf == 3 && sig > 0) ||
      (tf == 5 && sig > 0) || (tf == 7 && sig > 0) )
   {
      Vec4 vec;
      Cell &cel2 = gr(h, w + sig );

      Vec4 vt1 = eval.eE2(cel1) * hg;
      vt1 += eval.eE1(cel1);
```

```
      Vec4 vt2 = eval.eE2(cel2) * hg;
      vt2 += eval.eE1(cel2);

      if(sig > 0)
      {
         //vec = eval.eE(cel2) - eval.eE(cel1);
         vec = vt2 - vt1;
      }
      else
      {
         //vec = eval.eE(cel1) - eval.eE(cel2);
         vec = vt1 - vt2;
      }

      return vec;
   }
   // si esta en la frontera este pero no interfiere
   else if( (tf == 4 && sig < 0) || (tf == 6 && sig < 0)
      || (tf == 8 && sig < 0) )
   {
      Cell &cel2 = gr(h, w + sig );

      Vec4 vt1 = eval.eE2(cel1) * hg;
      vt1 += eval.eE1(cel1);

      Vec4 vt2 = eval.eE2(cel2) * hg;
      vt2 += eval.eE1(cel2);

      //Vec4 vec( eval.eE(cel1) - eval.eE(cel2) );
      Vec4 vec( vt1 - vt2 );

      return vec;
   }

   // si tiene una frontera en el oeste
   if( tf == 3 || tf == 5 || tf == 7)
   {
      return fronE3(gr,h,w,iter);
   }
   else if( tf == 4 || tf == 6 || tf == 8 )
   {
      return fronE4(gr,h,w,iter);
   }

   cerr << " no se proceso tf= " << tf << " sig= " << sig << '\n';

   Vec4 vecr;
   return vecr;
}

/// fase de correccion
Vec4 DifferenceE::difC(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   int tf = cel1.tf;   // tipo de frontera
   int sig = ts.dircE(iter); // direccion de la diferencia
   double hg = cel1.hc * 9.81; // tirante celda central

   /// si no esta en una frontera, o si esta en la
   // frontera este pero no interfiere
   if( tf == 0 || tf == 2 || tf == 1 || (tf == 3 && sig > 0) ||
      (tf == 5 && sig > 0) || (tf == 7 && sig > 0) )
   {
      Vec4 vec;
      Cell &cel2 = gr(h, w + sig );

      Vec4 vt1 = eval.eE2(cel1) * hg;
      vt1 += eval.eE1(cel1);

      Vec4 vt2 = eval.eE2(cel2) * hg;
      vt2 += eval.eE1(cel2);

      if(sig > 0)
      {
         //vec = eval.eE(cel2) - eval.eE(cel1);
         vec = vt2 - vt1;
      }
      else
      {
         //vec = eval.eE(cel1) - eval.eE(cel2);
         vec = vt1 - vt2;
```

```
    }

    return vec;
  }
  // si esta en la frontera este pero no interfiere
  else if( (tf == 4 && sig < 0) || (tf == 6 && sig < 0)
    || (tf == 8 && sig < 0) )
  {
    Cell &cel2 = gr(h, w + sig );

    Vec4 vt1 = eval.eE2(cel1) * hg;
    vt1 += eval.eE1(cel1);

    Vec4 vt2 = eval.eE2(cel2) * hg;
    vt2 += eval.eE1(cel2);

    //Vec4 vec( eval.eE(cel1) - eval.eE(cel2) );
    Vec4 vec( vt1 - vt2 );

    return vec;
  }

  // si tiene una frontera en el oeste
  if( tf == 3 || tf == 5 || tf == 7)
  {
    return fronE3(gr,h,w,iter);
  }
  else if( tf == 4 || tf == 6 || tf == 8 )
  {
    return fronE4(gr,h,w,iter);
  }

  cerr << " no se proceso tf= " << tf << " sig= " << sig << '\n';

  Vec4 vecr;
  return vecr;
}
```

archivo: DiferenceF.h:

```
/// implementacion de la clase de diferencias para
// el vector F
#ifndef DIFFERENCE_F_H
#define DIFFERENCE_F_H

#include "Difference.h"

class DifferenceF: public Difference
{
 private:
   //evalua la condicion de frontera 3 de la tabla 3.2
    Vec4 fronF1(Grid &gr, const int h, const int w,
         const int iter);

   //evalua la condicion de frontera 4 de la tabla 3.2
    Vec4 fronF2(Grid &gr, const int h,
      const int w, const int iter);

 public:
    DifferenceF();
   // diferencia de prediccion
   Vec4 difP(Grid &gr, const int h, const int w, const int iter);

   // diferencia de correcion
   Vec4 difC(Grid &gr, const int h, const int w, const int iter);
};

#endif
```

archivo: DiferenceF.cpp:

```
#include "DifferenceF.h"

DifferenceF::DifferenceF():Difference()
{
}

// frontera en celda tipo 1
Vec4 DifferenceF::fronF1(Grid &gr, const int h, const int w,
```

```
        const int iter)
{
   Cell &cel1 = gr(h,w);

   double u,v,hh, v_2;
   double v1, v2, v3, v4;

   u = cel1.uc;
   v = cel1.vc;
   hh = cel1.hc;
   v_2 = v * v;

   double uv = u * v;

   v1 = 2.0 * hh * v;
   v2 = 2.0 * hh * uv;
   v2 = 0.0;

   v4 = 2.0 * cel1.k * v_2 * v_2 * v;
   v4 /= sqrt(hh);

   Vec4 vecr(v1, v2, v3, v4);
   return vecr;
}

// frontera en celda tipo 2
Vec4 DifferenceF::fronF2(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   double u,v,hh, v_2;
   double v1, v2, v3, v4;

   u = cel1.uc;
   v = cel1.vc;
   hh = cel1.hc;
   v_2 = v * v;

   double uv = u * v;

   v1 = -2.0 * hh * v;
   v2 = -2.0 * hh * uv;
   v3 = 0.0;

   v4 = -2.0 * cel1.k * v_2 * v_2 * v;
   v4 /= sqrt(hh);

   Vec4 vecr(v1, v2, v3, v4);
   return vecr;
}

/// fase de prediccion
Vec4 DifferenceF::difP(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   int tf = cel1.tf;   // tipo de frontera
   int sig = ts.dirpF(iter); // direccion de la diferencia
   double hg = cel1.hc * 9.81;

   /// si no esta en una frontera, o si esta en la
   // frontera este pero no interfiere
   if( tf == 0 || tf == 3 || tf == 4 || (tf == 1 && sig > 0) ||
      (tf == 5 && sig > 0) || (tf == 6 && sig > 0) )
   {
      Vec4 vec;
      Cell &cel2 = gr(h + sig, w);

      Vec4 vt1 = eval.eF2(cel1) * hg;
      vt1 += eval.eF1(cel1);

      Vec4 vt2 = eval.eF2(cel2) * hg;
      vt2 += eval.eF1(cel2);

      if(sig > 0)
      {
         //vec = eval.eF(cel2) - eval.eF(cel1);
         vec = vt2 - vt1;
      }
      else
      {
         //vec = eval.eF(cel1) - eval.eF(cel2);
```

```
         vec = vt1 - vt2;
      }

      return vec;
   }
   // si esta en la frontera este pero no interfiere
   else if( (tf == 2 && sig < 0) || (tf == 7 && sig < 0)
      || (tf == 8 && sig < 0) )
   {
      Cell &cel2 = gr(h + sig, w);

      Vec4 vt1 = eval.eF2(cel1). * hg;
      vt1 += eval.eF1(cel1);

      Vec4 vt2 = eval.eF2(cel2) * hg;
      vt2 += eval.eF1(cel2);

      //Vec4 vec( eval.eF(cel1) - eval.eF(cel2) );
      Vec4 vec( vt1 - vt2 );

      return vec;
   }

   // si tiene una frontera en el sur
   if( tf == 1 || tf == 5 || tf == 6)
   {
      return fronF1(gr,h,w,iter);
   }
   // si tiene una frontera en el norte
   else if( tf == 2 || tf == 7 || tf == 8 )
   {
      return fronF2(gr,h,w,iter);
   }

   cerr << " no se proceso tf= " << tf << " sig= " << sig << '\n';

   Vec4 vecr;
   return vecr;
}

/// fase de correccion
Vec4 DifferenceF::difC(Grid &gr, const int h, const int w,
        const int iter)
{
   Cell &cel1 = gr(h,w);

   int tf = cel1.tf;    // tipo de frontera
   int sig = ts.dircF(iter); // direccion de la diferencia
   double hg = cel1.hc * 9.81;

   /// si no esta en una frontera, o si esta en la
   // frontera este pero no interfiere
   if( tf == 0 || tf == 3 || tf == 4 || (tf == 1 && sig > 0) ||
      (tf == 5 && sig > 0) || (tf == 6 && sig > 0) )
   {
      Vec4 vec;
      Cell &cel2 = gr(h + sig, w);

      Vec4 vt1 = eval.eF2(cel1) * hg;
      vt1 += eval.eF1(cel1);

      Vec4 vt2 = eval.eF2(cel2) * hg;
      vt2 += eval.eF1(cel2);


      if(sig > 0)
      {
         //vec = eval.eF(cel2) - eval.eF(cel1);
         vec = vt2 - vt1;
      }
      else
      {
         //vec = eval.eF(cel1) - eval.eF(cel2);
         vec = vt1 - vt2;
      }

      return vec;
   }
   // si esta en la frontera este pero no interfiere
   else if( (tf == 2 && sig < 0) || (tf == 7 && sig < 0)
      || (tf == 8 && sig < 0) )
   {
      Cell &cel2 = gr(h + sig, w);
```

```
      Vec4 vt1 = eval.eF2(cel1) * hg;
      vt1 += eval.eF1(cel1);

      Vec4 vt2 = eval.eF2(cel2) * hg;
      vt2 += eval.eF1(cel2);

      //Vec4 vec( eval.eF(cel1) - eval.eF(cel2) );
      Vec4 vec( vt1 - vt2 );

      return vec;
   }

   // si tiene una frontera en el sur
   if( tf == 1 || tf == 5 || tf == 6)
   {
      return fronF1(gr,h,w,iter);
   }
   // si tiene una frontera en el norte
   else if( tf == 2 || tf == 7 || tf == 8 )
   {
      return fronF2(gr,h,w,iter);
   }

   cerr << " no se proceso tf= " << tf << " sig= " << sig << '\n';

   Vec4 vecr;
   return vecr;
}
```

archivo: EvalEQ.h:

```
// clase funcion para evaluar los terminos U, E, F y S de la
// ecuacion diferencial vectorial
#ifndef EVALEQ_H
#define EVALEQ_H
#include "Cell.h"
#include <math.h>

class EvalEQ
{
 public:
   EvalEQ();
    Vec4 eU(Cell &c) const;
    Vec4 eE1(Cell &c) const;
    Vec4 eE2(Cell &c) const;
    Vec4 eF1(Cell &c) const;
    Vec4 eF2(Cell &c) const;
    Vec4 eS(Cell &c) const;
};

#endif
```

archivo: EvalEQ.cpp:

```
#include "EvalEQ.h"
#include <iostream>

using namespace std;

// constructor de default
EvalEQ::EvalEQ()
{
}

// metodo para evaluar el vector U de una celda que se
// pasa como argumento
Vec4 EvalEQ::eU(Cell &c) const
{
   double u1, u2, u3, u4;

   u1 = c.h;
   u2 = c.u * u1;
   u3 = c.v * u1;
   u4 = c.z;

   Vec4 uu(u1, u2, u3, u4);

   return uu;
}
```

```
//evalua el vector E con los datos de la celda
Vec4 EvalEQ::eE1(Cell &c) const
{
   double u, v, h, ht, u2;
   double v1, v2, v3, v4;

   u = c.uc;
   v = c.vc;
   h = c.hc;
   u2 = u*u;
   ht = c.htc;

   //// vector E ////
   v1 = h*u;
   v2 = v1 * u;
   v3 = v1 * v;

   v4 = c.k * u2 * u2 * u;
   v4 /= sqrt(h);

   Vec4 ee(v1, v2, v3, v4);

   return ee;
}

//evalua el vector E2 con los datos de la celda
Vec4 EvalEQ::eE2(Cell &c) const
{
   double ht;
   double v1, v2, v3, v4;

   ht = c.htc;

   //// vector E ////
   v1 = 0.0;
   v2 = ht;
   v3 = 0.0;
   v4 = 0.0;

   Vec4 ee(v1, v2, v3, v4);

   return ee;
}


// evalua el vector F con los datos de la celda
Vec4 EvalEQ::eF1(Cell &c) const
{
   double u, v, h, ht, v_2;
   double v1, v2, v3, v4;

   u = c.uc;
   v = c.vc;
   h = c.hc;
   ht = c.htc;
   v_2 = v * v;

   //// vector F ////
   v1 = h*v;
   v2 = v1 * u;
   v3 = v1 * v;

   v4 = c.k * v_2 * v_2 * v;
   v4 /= sqrt(h);

   Vec4 ff(v1, v2, v3, v4);

   return ff;
}

// evalua el vector F con los datos de la celda
Vec4 EvalEQ::eF2(Cell &c) const
{
   double ht;
   double v1, v2, v3, v4;

   ht = c.htc;

   //// vector F ////
   v1 = 0.0;
   v2 = 0.0;
   v3 = ht;
   v4 = 0.0;
```

```
   Vec4 ff(v1, v2, v3, v4);

   return ff;
}

// evalua el vector S con los datos de la celda.
// emplea la formula de Manning para calcular la
// pendiente de friccion
Vec4 EvalEQ::eS(Cell &c) const
{
   double u, v, h, sfx, sfy, n;
   double v1, v2, v3, v4;

   u = c.uc;
   v = c.vc;
   h = c.hc;
   n = c.n;

   // pendientes de friccion ///

   sfx = n * n / pow(h, 1.3333) * sqrt(u*u + v*v);
   sfy = sfx;

   sfx *= u;
   sfy *= v;

   //// vector S ////
   v1 = 0.0;
   v2 = - 9.81 * h * sfx;
   v3 = - 9.81 * h * sfy;
   v4 = 0.0;

   Vec4 ss(v1, v2, v3, v4);

   return ss;
}
```

archivo: Grid.h:

```
// objeto que contiene los datos de la malla rectangular
// en la cual se aplica el esquema de macCormack a las
// ecuaciones de Saint Venant
#ifndef GRID_H
#define GRID_H
#include "ReaderCell.h"
#include <math.h>
#include <iostream>

using namespace std;

class Grid
{

 protected:
    int width, height; // numero de celdas de la malla en x y y

   double dx, dy;   // tamanio de las celdas en x y y

   Cell **cells; // apuntador a las celdas de la malla;

 public:
    Grid( ReaderCell &rc );
   int getW() const;   // celdas en x
   int getH() const;   // celdas en y

   double getDx() const;
   double getDy() const;

   // regresa la celda especificada
   // se ubican las celdas como en las matrices
   Cell &operator()(const int row, const int col);

   // calcula las pendientes de la plantilla en x y y
   void calcSO();

   // determina el tipo de las fronteras cerradas
   void setTF();

   void write(const char *);
   void writeM();

   void filter(const double alfa);
```

```
};

#endif
```

archivo: Grid.cpp:

```
#include "Grid.h"

// constructor, recibe el lector
Grid::Grid( ReaderCell &rc )
{
   if( (!rc) == false )
   {
      cerr << "Grid::Grid: el lector no funciona\n";
      return;
   }

   // dimensiones de la malla
   rc.header(width, height, dx, dy);

   int size = width * height; // numero de celdas a leer

   cells = new (Cell*)[size];

   for(int i = 0; i < height; ++i)
   {
      for(int j = 0; j <  width; ++j)
      {
         cells[(height-i-1)*width + j] = rc.readCell();
      }
   }

   // se determina el tipo de fronteras
   setTF();

   // se calculan las pendientes de la plantilla
   calcSO();
}

Cell &Grid::operator()(const int row, const int col)
{
   return *cells[row * width + col];
}

// determina el tipo de frontera cerrada de acuerdo a la
// nomenclatura de ls figura x.3
void  Grid::setTF()
{
   Cell *cel;
   // se procesan las celdas tc = 1
   for(int i = 0; i < height; ++i)
   {
      for(int j = 0; j < width; ++j)
      {
         cel = &(*this)(i,j);

         // si es inactiva o frontera abierta
         if(cel->tc == 0)
            continue;
         else if( cel->tc == 2 || cel->tc == 3)
         {
            if( i == 0)
               cel->tf = 1;
            else if( (*this)(i-1,j).tc == 0)
               cel->tf = 1;
            else if( i == height-1 )
               cel->tf = 2;
            else if( (*this)(i+1,j).tc == 0)
               cel->tf = 2;
            continue;
            cerr << cel->tf << '\n';
         }

         //// esta en el borde ...
         if( i == height-1 )   // norte o...
            cel->tf = 2;
         else if(i == 0)       // ... sur
            cel-> tf = 1;

         /// o tiene una celda bloqueda en el ...
         else if( (*this)(i-1,j).tc == 0) // ...sur
```

```
            cel->tf = 1;
         else if( (*this)(i+1,j).tc == 0) // ...norte
            cel->tf = 2;

         /// y/o tiene una celda bloqueada en el ...
         if((*this)(i,j-1).tc == 0) // ... oeste
         {
            if(cel->tf == 1) // y ya es tipo 1
               cel->tf = 5;
            else if( cel->tf == 2) // y ya es tipo 2
               cel->tf = 7;
            else
               cel->tf = 3;
         }
         else if( (*this)(i,j+1).tc == 0) // ... este
         {
            if(cel->tf == 1) // y ya es tipo 1
               cel->tf = 6;
            else if( cel->tf == 2) // y ya es tipo 2
               cel->tf = 8;
            else
               cel->tf = 4;
         }
         //else if( (*this)(i,j+1).tf == 0) // ... este
         //cerr << i << ',' << j << " tf="
          //  << cel->tf << '\n';
      }
   }

   ofstream ts("testTF.txt");
   for(int i = 0; i < height; ++i)
   {
      for(int j = 0; j < width; ++j)
      {
         ts << (*this)(height-i-1,j).tf << '\t';
      }
      ts << '\n';
   }
   ts.close();
}

// calcula las pendientes a partir de las cotas del fondo
void Grid::calcSO()
{
   Cell *cel;
   double s1, s2;

   for(int i = 0; i < height; ++i)
   {
      for(int j = 0; j < width; ++j)
      {
         cel = &(*this)(i,j);
         if(cel->tc == 0) continue;

         //////// SX ////////////

         // frontera de entrada o tf = 3, 5 o 7
         if( cel->tf == 3 || cel->tc == 2 ||
            cel->tf == 5 || cel->tf == 7 )
         {
            cel->sx = (cel->zc - (*this)(i,j+1).zc)
                  / dx;
         }
         // front. de salida o tipo 4, 7 o 8
         else if( cel->tc == 3 || cel->tf == 4
            || cel->tf == 6 || cel->tf == 8)
         {
            cel->sx = ( (*this)(i,j-1).zc - cel->zc)
                  / dx;
         }
         else
         {
            s1 = ( (*this)(i,j-1).zc - cel->zc)
                  / dx;
            s2 = (cel->zc - (*this)(i,j+1).zc)
                  / dx;
            if(s1 == 0.0 || s2 == 0.0)
            {
               cel->sx = ( (*this)(i,j-1).zc -
                  (*this)(i,j+1).zc ) / dx;
            }
            else
            {
               cel->sx = 0.5 * ( s1 + s2 );
```

```
            }
         }

         //////// SY ///////////

         /// fronteras en el sur
         if(cel->tf == 1 || cel->tf == 5 || cel->tf == 6)
         {
            cel->sy = (cel->zc - (*this)(i+1,j).zc) / dy;
         }
         /// fronteras en el norte
         else if(cel->tf == 2 || cel->tf == 7
            || cel->tf == 8)
         {
            cel->sy = ( (*this)(i-1,j).zc - cel->zc ) / dy;
         }
         else
         {
            s1 = ( (*this)(i-1,j).zc - cel->zc) / dy;
            s2 = (cel->zc - (*this)(i+1,j).zc) / dy;
            if(s1 == 0.0 || s2 == 0.0)
            {
               cel->sy = ( (*this)(i-1,j).zc -
                  (*this)(i+1,j).zc ) / dx;
            }
            else
            {
               cel->sy = 0.5 * (s1 + s2);
            }
         }
      }
   }
}

void Grid::write( const char *nm)
{
   ofstream out( nm );

   out << "#x \t y \t u \t v \t h \t z \t Cn/dt \n";
   double x,y;
   double cel, cr;

   for(int i = 0; i < height; ++i)
   {
      y = (0.5 + (double)i) * dy;
      for(int j = 0; j < width; ++j)
      {
         x = (0.5 + (double)j) * dx;

         if( (*this)(i,j).tc > 0)
         {
            cel = sqrt( 9.81 * (*this)(i,j).h );
            cr = ( (*this)(i,j).u + cel ) / dx +
                 ( (*this)(i,j).v + cel ) / dy;
            out << x << '\t' << y << '\t' <<
               (*this)(i,j).u << '\t' <<
               (*this)(i,j).v << '\t' <<
               (*this)(i,j).h << '\t' <<
               (*this)(i,j).z << '\t' <<
               cr << '\n';
         }
      }
   }

   out.close();
}

int Grid::getW() const { return width; }

int Grid::getH() const { return height; }

double Grid::getDx() const { return dx; }

double Grid::getDy() const { return dy; }

void Grid::filter(const double alfa)
{

   double rfh, rfu, rfv;
   double n;

   Cell *cel;

   for(int i = 0; i < height; ++i)
```

```
{
   for(int j = 0; j < width; ++j)
   {
      cel = &(*this)(i,j);
      if( cel->tc == 1 )
      {
         if( cel->tf == 0) // no hay fronteras
         {
            // promedio del tirante
            rfh = (*this)(i-1,j).ht + (*this)(i+1,j).ht +
                  (*this)(i,j+1).ht + (*this)(i,j-1).ht;

            // promedio de velocidad u
            rfu = (*this)(i-1,j).u + (*this)(i+1,j).u +
                  (*this)(i,j+1).u + (*this)(i,j-1).u;

            // promedio de velocidad v
            rfv = (*this)(i-1,j).v + (*this)(i+1,j).v +
                  (*this)(i,j+1).v + (*this)(i,j-1).v;
            n = 4.0;
         }
         else if( cel->tf == 1 )
         {
            // promedio del tirante
            rfh = (*this)(i+1,j).ht +
                  (*this)(i,j+1).ht + (*this)(i,j-1).ht;

            // promedio de velocidad u
            rfu = (*this)(i+1,j).u +
                  (*this)(i,j+1).u + (*this)(i,j-1).u;

            // promedio de velocidad v
            rfv = (*this)(i+1,j).v +
                  (*this)(i,j+1).v + (*this)(i,j-1).v;

            n = 3.0;
         }
         else if( cel->tf == 2 )
         {
            // promedio del tirante
            rfh = (*this)(i-1,j).ht +
                  (*this)(i,j+1).ht + (*this)(i,j-1).ht;

            // promedio de velocidad u
            rfu = (*this)(i-1,j).u +
                  (*this)(i,j+1).u + (*this)(i,j-1).u;

            // promedio de velocidad v
            rfv = (*this)(i-1,j).v +
                  (*this)(i,j+1).v + (*this)(i,j-1).v;

            n = 3.0;
         }
         else if( cel->tf == 3)
         {
            // promedio del tirante
            rfh = (*this)(i-1,j).ht + (*this)(i+1,j).ht +
                  (*this)(i,j+1).ht;

            // promedio de velocidad u
            rfu = (*this)(i-1,j).u + (*this)(i+1,j).u +
                  (*this)(i,j+1).u;

            // promedio de velocidad v
            rfv = (*this)(i-1,j).v + (*this)(i+1,j).v +
                  (*this)(i,j+1).v;
            n = 3.0;
         }
         else if( cel->tf == 4)
         {
            // promedio del tirante
            rfh = (*this)(i-1,j).ht + (*this)(i+1,j).ht +
                  (*this)(i,j-1).ht;

            // promedio de velocidad u
            rfu = (*this)(i-1,j).u + (*this)(i+1,j).u +
                  (*this)(i,j-1).u;

            // promedio de velocidad v
            rfv = (*this)(i-1,j).v + (*this)(i+1,j).v +
                  (*this)(i,j-1).v;
            n = 3.0;
         }
         else if( cel->tf == 5)
```

```
            {
               // promedio del tirante
               rfh = (*this)(i+1,j).ht + (*this)(i,j+1).ht;

               // promedio de velocidad u
               rfu = (*this)(i+1,j).u + (*this)(i,j+1).u;

               // promedio de velocidad v
               rfv = (*this)(i+1,j).v + (*this)(i,j+1).v;
               n = 2.0;
            }
            else if( cel->tf == 6)
            {
               // promedio del tirante
               rfh = (*this)(i+1,j).ht + (*this)(i,j-1).ht;

               // promedio de velocidad u
               rfu = (*this)(i+1,j).u + (*this)(i,j-1).u;

               // promedio de velocidad v
               rfv = (*this)(i+1,j).v + (*this)(i,j-1).v;
               n = 2.0;
            }
            else if( cel->tf == 7)
            {
               // promedio del tirante
               rfh = (*this)(i-1,j).ht + (*this)(i,j+1).ht;

               // promedio de velocidad u
               rfu = (*this)(i-1,j).u + (*this)(i,j+1).u;

               // promedio de velocidad v
               rfv = (*this)(i-1,j).v + (*this)(i,j+1).v;
               n = 2.0;
            }
            else if( cel->tf == 8)
            {
               // promedio del tirante
               rfh = (*this)(i-1,j).ht + (*this)(i,j-1).ht;

               // promedio de velocidad u
               rfu = (*this)(i-1,j).u + (*this)(i,j-1).u;

               // promedio de velocidad v
               rfv = (*this)(i-1,j).v + (*this)(i,j-1).v;
               n = 2.0;
            }

            cel->ht = cel->ht*(1.0 - n*alfa) + alfa * rfh;
            cel->u = cel->u*(1.0 - n*alfa) + alfa * rfu;
            cel->v = cel->v*(1.0 - n*alfa) + alfa * rfv;

            cel->h = cel->ht - cel->z;
            cel->hc = cel->h;
            cel->hc = cel->h;
            cel->uc = cel->u;
            cel->vc = cel->v;
         }
      }
   }
}

void Grid::writeM()
{

   ofstream velU("velUFin.txt");
   ofstream velV("velVFin.txt");
   ofstream tir("hFin.txt");
   ofstream hto("htFin.txt");
   ofstream ele("elevFin.txt");
   for(int i = 0; i < height; ++i)
   {
      for(int j = 0; j < width; ++j)
      {
         if( (*this)(height-i-1,j).tc > 0)
         {
             velU << (*this)(height-i-1,j).u << '\t';
             velV << (*this)(height-i-1,j).v << '\t';
             tir << (*this)(height-i-1,j).h << '\t';
             hto << (*this)(height-i-1,j).ht << '\t';
             ele << (*this)(height-i-1,j).z << '\t';
         }
         else
         {
```

```
            velU << 0 << '\t';
            velV << 0 << '\t';
            tir << 0 << '\t';
            hto << 0 << '\t';
            ele << 0 << '\t';
         }

      }
      velU << '\n';
      velV << '\n';
      tir << '\n';
      hto << '\n';
      ele << '\n';
   }

}
```

archivo: Hidrogram.h:

```
// Clase a la que en un archivo se le indican
// puntos (t,Q) de un hidrograma e interpola linealmente
// entre ellos para obtener Q en un tiempo t cualquiera
#ifndef HIDROGRAM_H
#define HIDROGRAM_H

#include <fstream>

class Hidrogram
{
protected:
   double x[200], y[200],xmin, xmax;
   int np;

public:
   Hidrogram(const char *name)
   {
     ifstream in(name);
     int i=1;
     double xx, yy;
     in>>xx >> yy;

     xmax=xmin=xx;
     x[0]=xx;
     y[0]=yy;

     while (in >> xx) // lectura
     {
       in >> yy;

       if(xx < xmin) xmin =xx;
       if (xx > xmax) xmax=xx;

       x[i] = xx;
       y[i] = yy;

       i++;
     }

     np = i;
     in.close ();
   }

   double interp(const double &xx) const // funcion que interpola
   {
        if (xx < xmin) return 0;
        if (xx > xmax) return 0;

        int pos;

        for (pos=0;pos<np; ++pos)
        {
           if (x[pos] > xx) break;
        }

        double x0, x1, y0, y1, yi, p;

        x1 = x[pos];
        y1 = y[pos];
        x0 = x[pos-1];
        y0 = y[pos-1];

        p = (xx-x0)/(x1-x0);
```

```
        yi = y0+p*(y1-y0);

        return yi;
    }
};

#endif
```

archivo: MacCormack.h:

```
// implementacion del esquema de maccormack para la solucion de
// las ecuaciones de Saint Venant
#ifndef MACCORMACK_H
#define MACCORMACK_H

#include "DifferenceE.h"
#include "DifferenceF.h"
#include "Grid.h"
#include "Hidrogram.h"

#include <stdlib.h>

class MacCormack
{
 protected:
   Grid &grd;   //malla de calculo

   Hidrogram &hdr;   //hidrograma de entrada

   DifferenceE dE;   //diferencias en E
   DifferenceF dF;   //diferencias en F

   EvalEQ eval;   //evaluador de terminos de 3.12

   //double t0, tf, dt; //tiempos de calculo

 public:
   MacCormack( Grid &gr, Hidrogram &hd );

   //calculoe en el tiempo espeificado
   void calc(const double t0, const double tf, const double dt,
      const double dtw);

};
#endif
```

archivo: MacCormack.cpp:

```
#include "MacCormack.h"
#include <iostream>

MacCormack::MacCormack( Grid &gr, Hidrogram &hd ): grd(gr), hdr(hd)
{
}

void MacCormack::calc(const double t0, const double tf, const double dt,
        const double dtw)
{
   ////////////////////////////////////////////////////////
   /// se determinan parametros de calculo

   double dtx = dt / grd.getDx();
   double dty = dt / grd.getDy();
   double t = t0;    // tiempo actual de calculo

   int iter = 0;      // contador de iteraciones
   int iterw = (int)(dtw / dt);      // iteraciones antes de escribir
   int itermax = (int)((tf - t0) / dt);   //iteraciones maximas
   int tiw; // tiempo que se escribe en el archivo

   double b;       // ancho en la seccion de entrada
   double q;       // gasto en la entrada
   double hu, hh, ht, u, v, zz;
   double fs; // factor de sedimento ( porcentaje de Qs resp. a Q)
   fs = 0.0; // se fija temporalmente

   int w = grd.getW();
   int h = grd.getH();

   /// calculo del ancho de la entrada
```

```
b = 0.0;
for(int i = 0; i < h; ++i)
{
   for(int j = 0; j < w; ++j)
   {
      if( grd(i,j).tc == 2 )
     {
        b += grd.getDy();
     }
   }
}

//// vectores para almacenamiento temporal de resultados
Vec4 vect[h][w];

for(int ti = 0; ti < itermax; ++ti)
{

   t = t0 + ti*dt;   // tiempo actual

   if(ti % iterw == 0) // se guarda malla
   {
      string name("iter_");
      char text[10];

     tiw = (int)(ti*dt + 0.5);
      sprintf(text, "%d", tiw);
      name += text;
      name += ".txt";

      grd.write( name.c_str() );
   }

   ////////////////////////////////////////////////
   /////// Fase de prediccion //////

   for(int i = 0; i < h; ++i)
   {
      for(int j = 0; j < w; ++j)
      {
         Cell &cel = grd(i,j);
        if( cel.tc == 1)
        {
            Vec4 v2, v3;

            // se evalua cada termino de la ec 3.11

            // termino U
             //v1 = eval.eU( grd(i,j) );
            Vec4 v1( eval.eU( grd(i,j) ) );

            // diferencia de E
            v2 = dE.difP( grd, i, j, ti);

            // diferencia de F
            v3 = dF.difP( grd, i, j, ti );

            // vector S
            Vec4 v4( eval.eS( grd(i,j) ) );


            // multiplicacion por constantes

            v2 *= -dtx;
            v3 *= -dty;
            v4 *= dt;

            v1 += v2;
            v3 += v4;

           /// se almacena resultado
            vect[i][j] = v1 + v3;


        }
      }
   } // fin de fase de prediccion

   ////////////////////////////////////////////////
   //// frontera de entrada ////////
   q = hdr.interp( t + dt ); // revisar si el hidrog. esta en hrs.

   hu = q / b;
```

```
for(int i = 0; i < h; ++i)
{
   for(int j = 0; j < w; ++j)
   {
     Cell &cel = grd(i,j);
     if( cel.tc == 2)
     {
        Cell &cel2 = grd(i,j+1);

        hh = cel.h - dtx * ( cel2.h * cel2.u - hu);

        ht = hh + cel.z;

        //sedimento

        // se considera que el sedimento que entra es igual
        // al que sale
        /*
        double qsxt = cel2.uc;
        qsxt *= qsxt * qsxt * qsxt * qsxt;
        qsxt *= cel2.k / sqrt(cel.h);

        zz = cel.z - dtx * ( qsxt - hu * fs);
        */

        cel.h = hh;
        cel.hc = hh;

        cel.ht = ht;
        cel.htc = ht;

        cel.u = hu / hh;
        cel.uc = cel.u;

        cel.v = 0.0;
        cel.vc = 0.0;

        //cel.z = zz;
        //cel.zc = zz;
     }

   }
}

//////////////////////////////////////////////////
////// frontera de salida /////

//// en este momento se deja el nivel de
////  los tirantes fijo

for(int i = 0; i < h; ++i)
{
   for(int j = 0; j < w; ++j)
   {
     Cell &cel = grd(i,j);
     Cell &cel2 = grd(i,j-1);
     if( cel.tc == 3)
     {
        u = cel.u;
        v = cel.v;
        hh = cel.h;
        ht = cel.ht;
        Vec4 vec( eval.eS( cel ) );

        hu = hh * u - dtx * ( hh * u * u -
            cel2.h * cel2.u * cel2.u ) - 9.81 * dtx *
           ( ht - cel2.ht ) + dt * vec(1) ;

        u = hu / hh;

        // sedimento

        double qsxt1 = cel.u;
        qsxt1 *= qsxt1 * qsxt1 * qsxt1 * qsxt1;
        qsxt1 *= cel.k / sqrt(cel.h);

        double qsxt2 = cel2.uc;
        qsxt2 *= qsxt2 * qsxt2 * qsxt2 * qsxt2;
        qsxt2 *= cel2.k / sqrt(cel2.h);

         zz = cel.z - dtx * ( qsxt1 - qsxt2);

        cel.u = u;
        cel.uc = u;
```

```
        cel.v = v;
        cel.vc = v;

        cel.ht = ht;
        cel.htc = ht;

        cel.h = ht - zz;
        cel.hc = cel.h;

        cel.z = zz;
        cel.zc = zz;

    }
   }
}

//////////////////////////////////////////
//// se actualizan valores en la celdas
for(int i = 0; i < h; ++i)
{
   for(int j = 0; j < w; ++j)
  {
     Cell &cel = grd(i,j);
     if( cel.tc == 1 )
     {
        cel.getU( vect[i][j] );
     }
  }
}

grd.calcS0(); // se recauclula la pendiente

//////////////////////////////////////
///// fase de correcion
for(int i = 0; i < h; ++i)
{
   for(int j = 0; j < w; ++j)
   {
     Cell &cel = grd(i,j);
     if( cel.tc == 1)
     {
         Vec4 v1, v2, v3;

         // se evalua cada termino de la ec 3.11

         // termino U
         //Vec4 v1( eval.eU( grd(i,j) ) );
        v1 = eval.eU( grd(i,j) );

         // diferencia de E
         v2 = dE.difC( grd, i, j, ti);

         // diferencia de F
         v3 = dF.difC( grd, i, j, ti );

         // vector S
         Vec4 v4( eval.eS( grd(i,j) ) );

         // multiplicacion por constantes

         v2 *= -dtx;
         v3 *= -dty;
         v4 *= dt;

         // sumas parciales

         v1 += v2;
         v3 += v4;

        /// se almacenan resultados
         vect[i][j] = v1 + v3;

     }
   }
} // fin de fase de correcion

/// se alctualizan valores de las celdas
for(int i = 0; i < h; ++i)
{
   for(int j = 0; j < w; ++j)
  {
     Cell &cel = grd(i,j);
     if( cel.tc == 1 )
     {
```

```
            cel.getU( vect[i][j] );
         }
      }
   }

   ////////////////////////////////////////////
   ///// promedio de valores predichos y corregidos

   for(int i = 0; i < h; ++i)
   {
      for(int j = 0; j < w; ++j)
      {
         Cell &cel = grd(i,j);
         if( cel.tc == 1)
         {
            cel.prom();
         }
      }
   }

   // filtro numerico
   if( ti %  10 == 0 && ti > 0) grd.filter(0.20);

  } // fin de iteraciones
}
```

archivo: ReaderCell.h:

```
// clase para leer los datos de una malla rectangular
// para aplicar el metodo de MacCormack a las ecuaciones
// de Saint Venant
#ifndef READERCELL_H
#define READERCELL_H
#include "Cell.h"
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class ReaderCell
{
 private:
    ifstream *tcel, *elev, *rug, *uu, *vv, *hh, *kk;
   bool done;   // bandera de control
   int w, h;   // dimensiones de la malla a leer
   double dx, dy;   // deltas de celdas

 public:
   // en el constructor se lee el nombre del archivo de
   // cabecera de la malla
   ReaderCell( const char *n);


   // metodo para leer los datos de cabecera
   void header(int &w, int &h,
      double &dx, double &dy) const;

   // metodo para regresar la direccion de una celda que
   // se acaba de leer
   Cell *readCell();

   bool operator!() const;

};

#endif
```

archivo: ReaderCell.cpp:

```
#include "ReaderCell.h"

ReaderCell::ReaderCell( const char *name )
{
   done = false;

   ifstream hdr( name );

   if( !hdr )
```

```
{
   cerr << "no se abrio el archivo de cabecera\n";
   return;
}

hdr >> w >> h >> dx >> dy;
cout << "w= " << w << " h= " << h << '\n';

string text;

// se abre el archivo con el tipo de celda

hdr >> text;
tcel = new ifstream( text.c_str() );

if( !(*tcel) )
{
   cerr << " no se abrio archivo de tipos\n";
   return;
}

// se abre el archivo de elevaciones

hdr >> text;
elev = new ifstream( text.c_str() );

if( !(*elev) )
{
   cerr << " no se abrio archivo de elevaciones\n";
   return;
}

//// se abre archivo de rugosidades
hdr >> text;
rug = new ifstream( text.c_str() );

if( !(*rug) )
{
   cerr << " no se abrio archivo de rugosidades\n";
   return;
}


// se abre el archivo de velocidades iniciales en x

hdr >> text;
uu = new ifstream( text.c_str() );

if( !(*uu) )
{
   cerr << " no se abrio archivo de vel. U\n";
   return;
}

// se abre el archivo de velocidades iniciales en y

hdr >> text;
vv = new ifstream( text.c_str() );

if( !(*vv) )
{
   cerr << " no se abrio archivo de vel. V\n";
   return;
}

// se habre el archivo de tirantes

hdr >> text;
hh = new ifstream( text.c_str() );

if( !(*hh) )
{
   cerr << " no se abrio archivo de vel. h\n";
   return;
}

// se abre el archivo de coeficientes k

hdr >> text;
kk = new ifstream( text.c_str() );

if( !(*kk) )
{
   cerr << " no se abrio archivo de k. h\n";
```

```
      return;
   }

   hdr.close();

   done = true; //se abrio el archivo de cabecera correctamente
   cout << "archivos de datos abiertos\n";
}

// regresa dimensiones de la malla en referencias
void ReaderCell::header(int &ww, int &hh,
   double &dxx, double &dyy) const
{
   ww = w;
   hh = h;
   dxx = dx;
   dyy = dy;
}

bool ReaderCell::operator!() const
{
   return done;
}

// lee una celda y regresa un apuntador la lectura
// es avanzando primero por columna y despues por
// renglon
Cell *ReaderCell::readCell()
{
   double u,v,h;
   double z,n,k;
   int tc;

   (*tcel) >> tc;
   (*elev) >> z;
   (*rug) >> n;
   (*uu) >> u;
   (*vv) >> v;
   (*hh) >> h;
   (*kk) >> k;

   return new Cell( u, v, h, n, z, k, tc);
}
```

archivo: TableSign.h:

```
/// objeto funcion que alterna los signos de las diferencias
/// de acuerdo a la tabla 3.3
#ifndef TABLESIGN_H
#define TABLESIGN_H

class TableSign
{
 private:
    int spe[4];   //signo para operando E de la ec 3.3
    int spf[4];   //signo para operando F de la ec 3.4
   int start;
 public:
    TableSign( const int st = 0 );
   // direcciones de los operadores de diferencias en ...
   //... vector funcion E en prediccion
   int dirpE(const int iter) const;

   //... vector funcion E en correcion
   int dircE(const int iter) const;

   //... vector funcion F en prediccion
   int dirpF(const int iter) const;

   //... vector funcion F en correcion
   int dircF(const int iter) const;
};
#endif
```

archivo: TableSign.cpp:

```
#include "TableSign.h"

TableSign::TableSign(const int st):start(st)
```

```
{
   spe[0] = 1;
   spe[1] = 1;
   spe[2] = -1;
   spe[3] = -1;

   spf[0] = -1;
   spf[1] = 1;
   spf[2] = 1;
   spf[3] = -1;
}

// prediccion para E
int TableSign::dirpE(const int iter) const
{
   return -spe[iter%4];
}

// correcion para E
int TableSign::dircE(const int iter) const
{
   return spe[iter%4];
}

// prediccion para F
int TableSign::dirpF(const int iter) const
{
   return spf[iter%4];
}

// correcion para F
int TableSign::dircF(const int iter) const
{
   return -spf[iter%4];
}
```

archivo: Vec4.h:

```
#ifndef VEC4_H
#define VEC4_H
class Vec4
{
 protected:
    double vc[4];
 public:
   // constructores
    Vec4();
    Vec4(const double a, const double b, const double c,
         const double d);
   Vec4( const Vec4 &v);

   // operaciones
   double operator()(const int i) const;
   Vec4 operator=( const Vec4 &v);
   Vec4 operator*(const double f) const;
   Vec4 operator+(const Vec4 &v) const;
   Vec4 operator-(const Vec4 &v) const;

   Vec4 operator*=(const double f);
   Vec4 operator+=(const Vec4 &v);
   Vec4 operator-=(const Vec4 &v);
};
#endif
```

archivo: Vec4.cpp:

```
#include "Vec4.h"

/// constructores
Vec4::Vec4()
{
   for(int i = 0; i < 4; ++i)
   {
      vc[i] = 0.0;
   }
}

Vec4::Vec4(const double a, const double b, const double c,
      const double d)
{
```

```
   vc[0] = a;
   vc[1] = b;
   vc[2] = c;
   vc[3] = d;
}

Vec4::Vec4( const Vec4 &v)
{
   for(int i = 0; i < 4; ++i)
   {
      vc[i] = v.vc[i];
   }
}

/// operadores ////
double Vec4::operator()(const int i) const
{
   return vc[i%4];
}

Vec4 Vec4::operator=(const Vec4 &v)
{
   if(this == &v)
   {
      return *this;
   }

   for(int i = 0; i < 4; ++i)
   {
      vc[i] = v.vc[i];
   }

   return *this;
}

Vec4 Vec4::operator*(const double f) const
{
   Vec4 vn;

   for(int i = 0; i < 4; ++i)
   {
      vn.vc[i] = this->vc[i] * f;
   }

   return vn;
}

Vec4 Vec4::operator*=(const double f)
{
   for(int i = 0; i < 4; ++i)
   {
      this->vc[i] *= f;
   }

   return *this;
}

Vec4 Vec4::operator+(const Vec4 &v) const
{
   Vec4 vn;

   for(int i = 0; i < 4; ++i)
   {
      vn.vc[i] = this->vc[i] + v.vc[i];
   }

   return vn;
}

Vec4 Vec4::operator+=(const Vec4 &v)
{
   for(int i = 0; i < 4; ++i)
   {
      this->vc[i] += v.vc[i];
   }

   return *this;
}

Vec4 Vec4::operator-(const Vec4 &v) const
{
   Vec4 vn;

   for(int i = 0; i < 4; ++i)
```

```
    {
        vn.vc[i] = this->vc[i] - v.vc[i];
    }

    return vn;
}

Vec4 Vec4::operator-=(const Vec4 &v)
{
    for(int i = 0; i < 4; ++i)
    {
        this->vc[i] -= v.vc[i];
    }

    return *this;
}
```