**Program DamBreak and Subroutines**

Note by Neil Coleman: This program and associated subroutines developed to solve dam breach equations described in a paper by J. Walder & J. O’Connor (1997), published in the journal Water Resources Research. That paper must be consulted to supply appropriate data for the program variables for specific dam breach scenarios. The following program and subroutines were sent to me in 2012 by the US Geological Survey (former contact, J. Walder, USGS, now retired). Full citation of paper:

**Walder, J. S., & J. E. O’Connor (1997). Methods for predicting peak discharge of floods caused by failure of natural and constructed earthen dams. *Water Resour. Res. 33*, 2337–2348.**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

module declarations

!

integer::i !loop index

integer::numsteps !number of time steps to take

integer, parameter::N=2 !number of simultaneous equations

integer::outfilestatus !flag

integer::Fl !flag set to 1 if breach is above base, 0 once breach has reached base

!

character(len=20) outfile !output file name

!

real, parameter::g=3.72 !Martian acceleration of gravity in SI units

real, parameter::alpha=0.544 !constant in hydraulic relation for breach flow

real, parameter::beta=0.404 !constant in hydraulic relation for breach flow

!real, parameter::H0=3. !initial value of water level relative to breach floor

real, parameter::Q0=0. !initial value of discharge

real, parameter::T0=0. !initial value of time

!

real::Dt !time step

real::r

real::phi

real::phirad

real::D

real::k

real::V0

real::V0mks

real::p

real::tau

real::kmks

real::taumks

real::BW0

real::B0

real::step

real::T

real::Dc

real::WL0

!

!real::B !breach bottom width

!real::H !water level relative to breach floor

real::Q !discharge through the breach

real, dimension(N)::Y !the solution vector

real, dimension(N)::Yold

real, dimension(N)::Ynew

real, dimension(N)::Dydt

real, dimension(N)::Yout

real, dimension(N)::Yt

!

end module declarations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

program dambreak

!

! program based on Walder & O'Connor analysis

! calculates DIMENSIONAL outflow hydrograph at dam breach

! This implementation solves simultaneously for breach bottom width B,

! water depth h, and discharge Q. Initial values of these variables are

! specified, and the program then steps forward in time. A natural scale for

! time is the time for the breach to cut down to the dam base, and this scale

! is used in choosing the size of the time step.

!

! user has to specify:

!

! lake hypsometry (volume proportional to depth to some power)

! lake drawdown (commonly taken as the initial water level at the dam)

! breach shape factors

! breach downcutting rate

!

! Specify numerical constants. These come from the critical flow assumption:

!

use declarations

implicit none

real::term1

real::term2

!real, external::derivs

!

write(\*,\*) 'Spillover depth (m) = '

read(\*,\*) Dc

write(\*,\*) 'Distance of initial breach floor below dam crest (m) = '

read(\*,\*) D

write(\*,\*) 'Breach bottom-width/depth ratio (commonly 1 to 5) = '

read(\*,\*) r

write(\*,\*) 'Slope angle of breach sides (degrees) = '

read(\*,\*) phi

write(\*,\*) 'Breach downcutting rate (m/h) = '

read(\*,\*) k

write(\*,\*) 'Reservoir volume in million cubic meters = '

read(\*,\*) V0

write(\*,\*) 'Hypsometric exponent (V~h\*\*p) = '

read(\*,\*) p

write(\*,\*) 'Time step as fraction of breach downcutting time = '

read(\*,\*) step

write(\*,\*) 'Number of time steps = '

read(\*,\*) numsteps

write(\*,\*) 'Name of output file (<=20 characters) : '

read(\*,\*) outfile

!

tau=(Dc-D)/k !time for breach to erode to base, in h.

kmks=k/3600. !erosion rate in m/s

taumks=(Dc-D)/kmks !downcutting time in s

phirad=phi/57.29 !slope angle in radians

V0mks=1.e6\*V0

!

!

BW0=r\*D !initial breach bottom width

Fl=1 !flag value

B0=Dc-D !initial value of breach elevation

WL0=Dc !initial water level for spillover case

!

! Calculate magnitude of time step

!

Dt=taumks\*step

!

! open the output file

!

open(unit=10, file=outfile,status='new',action='write',iostat=outfilestatus)

!

! write basic parameter values into the file header

!

write(10,10) Dc

10 format(1x, 'Spillover depth (m) = ',F4.1)

write(10,40) D

40 format(1x,'Height of initial breach floor above dam base (m) = ',F5.1)

write(10,20) r

20 format(1x, 'Ratio of breach bottom width to depth = ',F4.1)

write(10,30) phi

30 format(1x, 'Slope angle of the breach slides (deg) = ',F4.1)

write(10,50) k

50 format(1x,'Breach downcutting rate (m/h) = ',F5.1)

write(10,60) V0

60 format(1x, 'Reservoir volume in million cubic meters = ',E8.2)

write(10,70) p

70 format(1x, 'Hypsometric exponent = ',F4.2,//)

!

! create column headings and write initial values to first line following

!

write(10,80)

80 format(1x, 'time (s)',7x,'breach floor elevation (m)',3x,'lake level (m)',5x,'discharge (cms)',2x,'Flag',/)

write(10,90) T0,B0,WL0,Q0,Fl

90 format(3x,F7.1,12x,F5.1,24x,F5.1,12x,F6.1,12x,I1)

!

! Start the time steps

!

Y(1)=B0 !first element of the vector Y is breach floor elevation

Y(2)=WL0 !second element of the vector Y is lake level

T=T0 !starting value of time

Fl=1 !flag set to 1 because breach is initially above the dam base

!

stepintime: Do i=1,numsteps

Yold(:)=Y(:) !Yold is set to the value of Y determined at end of last time step

Call Derivs(T,Yold,N,Dydt) !returns the starting values of the derivatives

Call Rk4(Yold,Dydt,N,T,Dt,Ynew) !Ynew contains the solution stepped forward by timestep Dt

!

! Calculate discharge

!

term1=alpha/tan(phirad)+beta\*r\*(Dc-Ynew(1)\*Fl)/(Ynew(2)-Ynew(1)\*Fl)

term2=sqrt(g)\*((Ynew(2)-Ynew(1))\*\*2.5)

Q=term1\*term2

!

! Write the results to output file

!

T=T+Dt

If (T>=taumks) then

Fl=0

Endif

Y(:)=Ynew(:) !Set the vector Y to the values returned at end of step

write(10,90) T,Y(1),Y(2),Q,Fl !flag becomes zero when breach has eroded to base

End Do stepintime

!

! close output file

!

close(unit=10)

!

end program dambreak

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Subroutine Derivs(Tt,F,Nn,Dfdt)

!

! defines the form of the derivatives for the Runge-Kutta solver

!

use declarations

implicit none

integer::Nn

real::term1

real::term2

real::term3

!integer, intent(in)::Flag

real, intent(in)::Tt

real, intent(in), dimension(Nn)::F

real, intent(out),dimension(Nn)::Dfdt

!

Dfdt(1)=-kmks\*Fl

term1=(WL0\*\*p)/(p\*V0mks)

term2=(F(2)\*\*(1.-p))\*sqrt(g)\*((F(2)-F(1))\*\*2.5)

term3=alpha/tan(phirad)+beta\*r\*(Dc-F(1)\*Fl)/(F(2)-F(1)\*Fl)

Dfdt(2)=-term1\*term2\*term3

!

Return

End

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Subroutine Rk4(F,Dfdt,N,T,S,Fout)

!

!Runge-Kutta solver per "Numerical Recipes"

! Given values for N variables in the vector F and their derivatives in the vector Dfdt, known at T,

! use the fourth-order Runge-Kutta method to advance the solution over an interval S and return the

! incremented variables as Fout. The user supplies the subroutine Derivs, which returns the derivatives Dfdt.

!

implicit none

integer::j !loop index

integer, intent(in)::N !number of elements in each vector

!integer, intent(in)::Flag !flags whether breach has reached base

real, intent(in), dimension(N)::F

real, intent(in), dimension(N)::Dfdt

real, intent(out), dimension(N)::Fout

real, dimension(N)::Ft

real, dimension(N)::Dft

real, dimension(N)::Dfm

real, intent(in)::S

real, intent(in)::T

real::Sh

real::S6

real::Th

!

Sh=S/2.

S6=S/6.

Th=T+Sh

Do j=1,N

Ft(j)=F(j)+Sh\*Dfdt(j)

End Do

Call Derivs(Th,Ft,N,Dft)

Do j=1,N

Ft(j)=F(j)+Sh\*Dft(j)

End Do

Call Derivs(Th,Ft,N,Dfm)

Do j=1,N

Ft(j)=F(j)+S\*Dfm(j)

Dfm(j)=Dft(j)+Dfm(j)

End Do

Call Derivs(T+S,Ft,N,Dft)

Do j=1,N

Fout(j)=F(j)+S6\*(Dfdt(j)+Dft(j)+2.\*Dfm(j))

End Do

Return

End