Appendix C

Computer Programs and Flow Charts

Computational flow chart for simulating flow over a curved bed
Appendix C. Computer Programs and Flow Charts

// SIMULATION OF FLOW OVER CURVED BEDS
COMPUTATION OF FLOW SURFACE AND BED PRESSURE PROFILES FOR STEADY FLOW OVER CURVED BED USING A ONE-DIMENSIONAL BOUSSINESQ-TYPE EQUATION. THE FLOW DEPTH AND SLOPE OF THE WATER SURFACE AT THE INFLOW SECTION, AND FLOW DEPTH AT THE OUTFLOW SECTION OF THE SOLUTION DOMAIN ARE SPECIFIED AS BOUNDARY CONDITIONS.

*****************************NOTATION**********************************
by = the specified flow depth at the inflow section of the computational domain;
ey = the specified flow depth at the outflow section of the computational domain;
q[i] = discharge per unit width;
l = length of the computational domain;
B = width of the channel;
m = the total number of nodes with unknown nodal values in the computational domain;
z[i] = depth of the flow at nodal point i;
r[i] = the right-hand side array of the Newton-Raphson linearising equation;
y[i] = the correction flow depth from the Newton-Raphson method at node i;
h = the size of the step;
k = the slope of the water surface at the inflow section;
xl and xr = the lower and upper interval values of the root for the bisection method;
oxo = the coordinate of the upstream end of the channel.

#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
#include<math.h>
#include "YTZ.h"

int main ()
{
int i,j,m,ite,kk,*dir[200],DIR[200],mm,nn;
double *sf[200],*df[200],*tf[200],SF[200],DF[200],TF[200],*y[200],Y[200],R[200],syn,*z[200],l,sumN,*pr[200],PR[200],LL,HH,Y2[200],*y2[200],ii, jj, Z[200],*z[200],h,ey,by,k,*a[20000],*b[200],*c[20000],*A[150][150],*r[200],error,ZZ[200],AA[20000],*sz[200],*dz[200],*tz[200],SZ[200],DZ[200],T2[200],*ps[200],PS[200],ks,ep[200],PSR[200],*Sff[200],SFF[200],xo,*wd[200],B[200],q[200],*QQ[200],xl,xr,Zb[200],*pf[200],FF[200],PPP[200],xx[200],*p[200],pb,aa,bb;
const double beta=1.0E-6;

//cout<<"ENTER THE VALUE OF 'h'"<<"\n";
cin>>&h;
cout<<"ENTER THE VALUE OF 'L' "<<"\n";
cin>>&l;
LL=l*1000.0;
HH=h*1000.0;
jj=LL/HH;
m=jj-1;
cout<<"ENTER THE BOUNDARY CONDITIONS AT THE END SECTIONS:"<<"\n";
cout<<"DEPTH OF THE FLOW 'by'"<<"\n";
cin>&by;
cout<<"DEPTH OF FLOW 'ey'"<<"\n";
cin>&ey;
cout<<"ENTER THE VALUE OF 'XL' "<<"\n";
Appendix C. Computer Programs and Flow Charts

```c
#include <iostream>
using namespace std;

// Function declarations...

// Function definitions...

int main() {
    double xl, xr, xo;
    cout << "ENTER THE VALUE OF 'XR' " << endl;
    cin >> xl;
    cout << "ENTER THE VALUE OF 'X0' " << endl;
    cin >> xr;
    cout << "ENTER THE VALUE OF 'X0' " << endl;
    cin >> xo;
    double Z[0] = by;
    double Z[m+1] = ey;

    //********************INITIALISING THE VARIABLES**************************
    for (j=1; j<=m; j++) {
        tf[j] = 0;
        df[j] = 0;
        sf[j] = 0;
        b[j] = 0;
        y[j] = 0;
        r[j] = 0;
        dir[j] = 0;
    }
    for (j=0; j<=m+1; j++) {
        z[j] = 0;
        sz[j] = 0;
        dz[j] = 0;
        tz[j] = 0;
        pr[j] = 0;
        ps[j] = 0;
        y2[j] = 0;
        Sff[j] = 0;
        wd[j] = 0;
        QQ[j] = 0;
        pf[j] = 0;
        x[j] = 0;
        p[j] = 0;
    }

    //********************ELEVATION AND DERIVATIVES OF THE BED PROFILE*********
    proLAT(pr, m, h, xo);
    for (j=0; j<=m+1; j++) {
        PR[j] = *pr[j];
    }

    slopLAT(sz, dz, tz, m, h, xo);
    for (i=0; i<=m+1; i++) {
        SZ[i] = *sz[i];
        DZ[i] = *dz[i];
        TZ[i] = *tz[i];
    }

    //********************THE WIDTH AND UNIT DISCHARGE OF THE CHANNEL*******
    comBqLAT(h, m, wd, QQ, xo);
    for (i=0; i<=m+1; i++) {
        B[i] = *wd[i];
        q[i] = *QQ[i];
    }

    //**********INITIALISING THE NODAL VALUES USING THE ENERGY EQUATION*****
    valintLAT(z, m, q, by, ey, PR, xl, xr, xo, h);
    for (j=1; j<=m; j++) {
        Z[j] = *z[j];
    }
    for (i=1; i<=m*m; i++) {
        a[i] = 0;
        c[i] = 0;
    }
    k = waterslopebs(SZ, q, Z, B);
    return 0;
}
```
// DETERMINATION OF THE FINAL WATER SURFACE ELEVATIONS
error=beta;
ite=0;
while (error>=beta && ite<=100){

//ROUGHNESS PARAMETERS FOR THE CHANNEL
frismoothbs(q,Z,B,m,Sff);
for (i=0;i<=m;i++)
    SFF[i]=Sff[i];

//DETERMINATION OF THE NON-LINEAR COEFFICIENTS OF THE EQUATION
for (i=1;i<=m;i++)
    SF[i]=cosf(SZ[i],Z[i]);
    DF[i]=coshf(SZ[i],Z[i],q[i]);
    TF[i]=cotf(SZ[i],DZ[i],TZ[i],Z[i],q[i],B[i],SFF[i]);
compf(pf,B,h,DZ,Z,m,SZ,xo);
for (i=1;i<=m;i++)
    PF[i]=pf[i];

//SOLVE THE VALUE OF SYN=Z[-1] AND THE RIGHT HAND SIDE EQUATIONS
solvbs(syn,Z,B,h,q,k,SZ,DZ,TZ,SFF);
comrbs(SF,DF,TF,m,h,r,syn,Z,PF,SZ,DZ,B);
for (i=1;i<=m;i++)
    R[i]=r[i];

//NUMERICAL DETERMINATION OF THE JACOBIAN MATRIX
linerbs (a,R,SF,DF,TF,h,Z,m,SZ,DZ,TZ,B,k,q,SFF,PF,xo);
kk=0;
for (i=1;i<=m;i++)
    for (j=1;j<=m;j++)
        A[i][j]=a[i+jkk];
kk=kk+m;

//SOLVE THE CORRECTION 'Y' FOR THE SOLUTION
decom (A,m,dir,c);
for (i=1;i<=m;i++)
    AA[i]=c[i];
for (j=1;j<=m;j++)
    DIR[j]=dir[j];
solu (AA,m,DIR,R,b);
for (i=1;i<=m;i++)
    Y[i]=b[i];
error=0.0;
for (i=1;i<=m;i++)
    error=error+fabs(Y[i]);
Z[i]=Z[i]+Y[i];
ite++;}
press(ps,q,h,DZ,Z,syn,m,SZ);
for (i=0;i<=m+1;i++)
    PS[i]=ps[i];

//OUTPUT THE SIMULATION RESULTS
cout<<"FLOW SURFACE PROFILE"<<endl;
for (i=0; i<=m+1; i++){
  cout<<Z[i]<<endl;
}
cout<<"BED PRESSURE PROFILE"<<endl;
for (j=0; j<=m+1; j++)
  cout<<PS[j]<<endl;
}
cout<<"BED PROFILE COORDINATES"<<endl;
for (j=0; j<=m+1; j++)
  cout<<PR[j]<<endl;
}
cout<<"error="<<error<<"\n"<<"number of iteration="<<ite<<"\n"<<"\n"<<"PRESS ANY KEY TO CONTINUE...";
getch();
return(0);
}

//**********************************************************************
// PROGRAM TO EVALUATE THE RIGHT HAND SIDE EQUATIONS FROM THE NEWTON-RAPHSON RULE
//**********************************************************************
double comrbs(double SF[200], double DF[200], double TF[200], int m, double h, double & syn, double Z[200], double PF[200], double SZ[200], double DZ[200], double B[200])
{
  int i, j;
  double u[200], v[200], w[200], x[200];
  for (j=1; j<=m; j++)
    r[j]=&x[j];
  w[1]=syn*(-6.0+(DF[1]*h*h))+(6.0*h*h*h*TF[1])+(6.0*h*h*h*PF[1]);
  u[1]=Z[0]*(18.0+(6.0*SF[1]*h)-(6.0*DF[1]*h*h));
  v[1]=Z[1]*(-18.0-(12.0*SF[1]*h)+(3.0*DF[1]*h*h))
    +Z[2]*(6.0+(6.0*SF[1]*h)+(2.0*DF[1]*h*h));
  for (i=2; i<=m; i++)
  {
    w[i]=Z[i-2]*(-6.0+(DF[i]*h*h))+(6.0*h*h*h*TF[i])+(6.0*h*h*h*PF[i]);
    u[i]=Z[i-1]*(18.0+(6.0*SF[i]*h)-(6.0*DF[i]*h*h));
    v[i]=Z[i]*(-18.0-(12.0*SF[i]*h)+(3.0*DF[i]*h*h))
      +Z[i+1]*(6.0+(6.0*SF[i]*h)+(2.0*DF[i]*h*h));
    *r[i]=-1.0*(u[i]+v[i]+w[i]);
  }
  return (0);
}

//**********************************************************************
// PROGRAM TO EVALUATE THE NODAL VALUE AT THE IMAGINARY NODE USING THE BOUNDARY CONDITION AT THE INFLOW SECTION
//**********************************************************************
double solvbs(double & syn, double y[200], double B[200], double h, double q[200], double k, double SZ[200], double DZ[200], double TZ[200], double fs[200])
{
  double a, b, c, d, e, D, sf, df, tf, dd;
  const double g=9.81, om=0.95;
  sf=(2.0*SZ[0])/y[0];
  df=(4.0+(4.0*SZ[0]*SZ[0]))*((-1.0/(y[0]*y[0]))+((g*y[0])/(q[0]*q[0])));
  tf=(4.0+(4.0*SZ[0]*SZ[0]))*((g*y[0]*SZ[0])/(q[0]*q[0]))+
    ((g*y[0]*fs[0])/(q[0]*q[0]))+4.0*om*((TZ[0]/2.0)+(SZ[0]*DZ[0]/y[0]));
a=-6.0+(df*h*h);
b=(-6.0*df*h*h)+(6.0*sf*h)+18.0;
c=(3.0*df*h*h)-(12.0*sf*h)-18.0;
d=6.0+(6.0*sf*h)+(2.0*df*h*h);
syn=(-1.0/(b+6.0*a))*(y[1]*d+c*y[0]+a*(6.0*h*k-2.0*y[1]-3.0*y[0])+(6.0*h*h*tf));
return (0);
}

//**********************************************************************
// PROGRAM TO DECOMPOSE THE GIVEN MATRIX INTO UPPER AND LOWER
// TRIANGULAR MATRICES
// Adopted from Press et al. (2002, #2.3)
//**********************************************************************
double decom(double a[150][150],int m,int *indx[200],double *A[20000])
{
    int i,imax,j,k,yy[200],kk;
    double max,sos,sum,temp;
    double vv[200],b[20000],bb[20000];
    for (i=1;i<=m*m;i++)
        A[i]=&b[i];
    for (i=1;i<=m;i++)
        indx[i]=&yy[i];
    for (i=1;i<=m;i++)
        max=0.0;
        for (j=1;j<=m;j++)
            temp=fabs(a[i][j]);
                if (temp>max)
                    max=temp;
        if (max==0.0)
            cout<<"ERROR: NO NON-ZERO LARGEST ELEMENT FOR ROW"<<"\n";
                else
                    vv[i]=1.0/max;
                for (j=1;j<=m;j++)
                    for (i=1;i<=j-1;i++)
                        sum=a[i][j];
                        for (k=1;k<=i-1;k++)
                            sum=sum-a[i][k]*a[k][j];
                        a[i][j]=sum;
                    sos=(vv[i])*fabs(sum);
                        if (sos>=max)
                            max=sos;
                            imax=i;
        }
        if (j!=imax){
C. Computer Programs and Flow Charts

```c
for (k=1;k<=m;k++){
    sos=a[imax][k];
    a[imax][k]=a[j][k];
    a[j][k]=sos;
}

vv[imax]=vv[j];
}
*indx[j]=imax;
if (j!=m){
    sos=1.0/(a[j][j]);
    for (i=j+1;i<=m;i++){
        a[i][j]=a[i][j]*sos;
    }
}
}
k=0;
for (i=1;i<=m;i++){
    for (j=1;j<=m;j++){
        bb[j+kk]=a[i][j];
    }
    kk=kk+m;
}
for (i=1;i<=m^2;i++){
    *A[i]=bb[i];
}
return(0);
}

//**************************************************************************************
// PROGRAM FOR THE SOLUTIONS OF THE DECOMPOSED MATRIX EQUATIONS
// Adopted from Press et al. (2002, #2.3)
//**********************************************************************

double solu(double A[20000],int m,int indx[200],double R[200],
double *y[200])
{
    int i,ip,j,k;
    double sum,d[200],AA[150][150];
    int ii=0;
    for (j=1;j<=m;j++) {
        y[j]=&d[j];
    }
    k=0;
    for (i=1;i<=m;i++){
        for (j=1;j<=m;j++){
            AA[i][j]=A[j+k];
        }
        k=k+m;
    }
    for (i=1;i<=m;i++){
        ip=indx[i];
        sum=R[ip];
        R[ip]=R[i];
        if (ii!=0)
            for (j=ii;j<=i-1;j++){
                sum=sum-AA[i][j]*R[j];
            }
        else
            if (sum!=0.0){
                for (j=ii+1;j<=m;j++){
                    R[j]=R[j]-sum-AA[i][j]*y[j];
                }
                return(0);
            }
    }
    return(0);
}
```
ii=i;
}
R[i]=sum;
}
for (i=m;i>=1;i--){
    sum=R[i];
    for (j=i+1;j<=m;j++){
        sum=sum-AA[i][j]*R[j];
    }
    R[i]=sum/AA[i][i];
}
for (i=1;i<=m;i++){
    *y[i]=R[i];
}
return(0);
}
Appendix C. Computer Programs and Flow Charts

```c
solvbs(syn,Z,B,h,q,k,SZ,DZ,TZ,fs);
comrbs(SF,DF,TF,m,h,r,syn,Z,PF,SZ,DZ,B);
for (i=1;i<=m;i++){
    RR2[i]=(-1.0)*(*r[i]);
}
Z[j]=Z[j]-hh;
for (i=1;i<=m;i++){
    b[i][j]=(RR2[i]-RR1[i])/hh;
}
ll=0;
for (i=1;i<=m;i++) {
    for (j=1;j<=m;j++) {
        *a[j+ll]= b[i][j];
    }
    ll=ll+m;
    return(0);
}
//**********************************************************************
// PROGRAM TO PREDICT THE BED PRESSURE PROFILE
//**********************************************************************
double press(double *ps[200],double q[200],double h,double DZ[200],
               double Z[200],double syn,int m,double SZ[200]){
    int i,j;
    double BB[200],tt[200];
    const double g=9.81,om=0.95;
    for (i=0;i<=m+1;i++){
        ps[i]=&BB[i];
    }
    for (i=0;i<=m+1;i++){
        tt[i]=1.0+pow(SZ[i],2);
    }
    *ps[0]=(Z[0]/g)*(g+(((q[0]*q[0])/(Z[0]*Z[0]*tt[0]))*(om*DZ[0]+
                      ((1.0/(2.0*h*h))*(Z[1]-2.0*Z[0]+syn))));
    Z[m+1]=Z[m];
    for (j=1;j<=m;j++){
        *ps[j]=(Z[j]/g)*(g+(((q[j]*q[j])/(Z[j]*Z[j]*tt[j]))*(om*DZ[j]+
                      ((1.0/(2.0*h*h))*(Z[j+1]-2.0*Z[j]+Z[j-1]))));
    }
    return (0);
}
//**********************************************************************
//**********************************************************************
double cosf(double SZ,double y){
    double sf;
    sf=2.0*(SZ/y);   //BTMU model
    return (sf);
}
//**********************************************************************
//**********************************************************************
double codf(double SZ,double y,double q)
```
double df, aa;
const double g = 9.81;

    aa = (4.0 + (4.0 * SZ * SZ)); //BTMU model
    df = aa * (((g * y) / (q * q)) - (1.0 / (y * y)));
    return (df);
}

double cotf(double SZ, double DZ, double TZ, double y, double q, double B, double fs)
{
    double DD, tf, d, D;
    const double g = 9.81, om = 0.95;
    tf = (4.0 + 4.0 * (SZ * SZ)) * (((g * y * SZ) / (q * q)) + ((g * y * fs) / (q * q))) +
    4.0 * om * ((TZ / 2.0) + ((SZ * DZ) / y)); //BTMU model
    return (tf);
}

double compf(double *pf[200], double B[200], double h, double DZ[200], double Z[200], int m, double SZ[200], double xo)
{
    int i, j;
    double BB[200], tt[200], ii, x[200], SB[200], xx[200], zz[200];
    const double om = 0.95;
    for (i = 0; i <= m + 1; i++)
    { 
        pf[i] = &BB[i];
    }
    for (i = 1; i <= m; i++)
    {
        ii = i;
        x[i] = (ii * h) + xo;
        xx[i] = 5.5556 * x[i];
        zz[i] = 8281.0 - 40000.0 * (x[i] * x[i]) + 24000.0 * x[i];
    
        if (x[i] <= -0.2827433) SB[i] = 0.0;
        else if (0.2827433 < x[i] & & x[i] <= 0.2827433)
        SB[i] = 0.69333 * cos(xx[i]) * sin(xx[i]);
        else SB[i] = 0.0;
    }
    for (j = 1; j <= m; j++)
    {
        *pf[j] = (-4.0 * SB[j] / B[j]) * ((tt[j] / Z[j]) + (1.0 / (2.0 * h * h)) * (Z[j + 1] -
        2.0 * Z[j] + Z[j - 1])) + (om * DZ[j]); //BTMU model
    }
    return (0);
}

double waterslopebs(double SZ[200], double q[200], double y[200], double B[200])
{
    double R, Hw, Fr2, b, a, Re, e, fs;
    const double g = 9.81, nu = 1.0e-6, n = 0.01;
Appendix C. Computer Programs and Flow Charts

//********************FOR SMOOTH BED CHANNEL**************************
R=(2.0*y[0])+B[0];
Re=(4.0*R*q[0])/(y[0]*nu);
b=6.90/Re;
a=log10(b);
f=(pow(a,-2))/(3.24);
fs= (f*pow(q[0],2))/(8.0*g*R*pow(y[0],2));

//***********************FOR ROUGH BED CHANNEL*************************
/*
R=y[0]);   // Wide channel assumption!
b=pow(R,4.0);
a=pow(b,1.0/3.0);
fs=(n*n*q[0]*q[0])/(y[0]*y[0]*a);
*/

//**********************************************************************
Fr2=(q[0]*q[0])/(g*pow(y[0],3.0));
Hw=(SZ[0]-fs)/(1.0-Fr2);
return (Hw);

//**********************************************************************
// PROGRAM TO EVALUATE THE FRICTION SLOPE AT DIFFERENT NODAL POINTS
//**********************************************************************
double frismoothbs(double q[200],double H[200],double B[200],int m,
double *Sff[200])
{
  int i,j;
double Re,R,f,a,b,SFF[200];
  const double g=9.81,nu=1.0e-6,n=0.0208;
  for (j=0;j<=m;j++){
    Sff[j]=&SFF[j];
  }
  //*************************FOR SMOOTH BED CHANNEL**********************
  for (i=0;i<=m;i++){
    R=(2.0*H[i])+B[i];
    Re=(4.0*R*q[i])/(H[i]*nu);
    b=6.90/Re;
    a=log10(b);
    f=(pow(a,-2))/(3.24);
    *Sff[i]= (f*pow(q[i],2))/(8.0*g*R*pow(H[i],2));
    /*
    //***********************FOR ROUGH BED CHANNEL*************************
    R=H[i]);   // Wide channel assumption!
b=pow(R,4.0);
a=pow(b,1.0/3.0);
    *Sff[i]=(n*n*q[i]*q[i])/(H[i]*H[i]*a);
    */
  }
  return (0);
}

//**********************************************************************
// PROGRAM TO EVALUATE THE BED PROFILE ELEVATIONS
//**********************************************************************
double proLAT(double *pr[200],int m,double h,double xo)
{
  int i,j;
double x[200],PR[200],ii,a,y[200],xx[200];
  const double ho=0.06,s=0.20;
  //********************FOR SMOOTH BED CHANNEL**************************
  for (i=0;i<=m;i++)
    R=(2.0*H[i])+B[i];
    Re=(4.0*R*q[i])/(H[i]*nu);
    b=6.90/Re;
    a=log10(b);
    f=(pow(a,-2))/(3.24);
    *Sff[i]= (f*pow(q[i],2))/(8.0*g*R*pow(H[i],2));
    /*
    //***********************FOR ROUGH BED CHANNEL*************************
    R=H[i]);   // Wide channel assumption!
b=pow(R,4.0);
a=pow(b,1.0/3.0);
    *Sff[i]=(n*n*q[i]*q[i])/(H[i]*H[i]*a);
    */
  }
  return (0);
}
for (j=0; j<=m+1; j++) {
    pr[j]=&PR[j];
}
for (i=0; i<=m+1; i++) {
    ii=i;
    x[i]=(ii*h)+xo;
    if (x[i]<-0.12) *pr[i]=0.0;
    else if (-0.12<=x[i]&&x[i]<=-0.05) *pr[i]=7945.56*pow(x[i],5.0)+
          380.55*pow(x[i],4.0)+551.116*pow(x[i],3.0)+43.6219*pow(x[i],2.0)+
          1.78941*x[i]+0.0356239;
    else if (-0.05<x[i]&&x[i]<=-0.05) *pr[i]=5.55556*x[i]-1.11111,
          *pr[i]=(0.003+(0.06*(cos(xx[i])*cos(xx[i]))));
    else if (0.45<x[i]&&x[i]<=0.55) *pr[i]=-1150.53*pow(x[i],5.0)+
          2903.56*pow(x[i],4.0)-2926.98*pow(x[i],3.0)+1473.75*pow(x[i],2.0)-
          370.804*x[i]+37.3204;
    else *pr[i]=0.0;
}
return (0);
}
Appendix C. Computer Programs and Flow Charts

xx[i]=5.5556*x[i]-1.1111;
if (x[i]<-0.12)*sz[i]=0.0,*dz[i]=0.0,*tz[i]=0.0;
else if (-0.12<x[i]&&x[i]<=-0.05)*sz[i]=39728.0*pow(x[i],4.0)+
13522.0*pow(x[i],3.0)+1653.3*pow(x[i],2.0)+87.244*x[i]+1.7894,
*dz[i]=3.7038-(7.4075*pow(cos(xx[i]),2.0)),
*tz[i]=82.306*cos(xx[i])*sin(xx[i]);
else if (0.45<x[i]&&x[i]<=0.55)*sz[i]=-5752.7*pow(x[i],4.0)+
11614.0*pow(x[i],3.0)-8780.90*pow(x[i],2.0)+2947.50*x[i]-370.80,
*dz[i]=-23011.0*pow(x[i],3.0)+34842.0*pow(x[i],2.0)-
17562.0*x[i]+2947.50,*tz[i]=-69033.0*pow(x[i],2.0)+69684.0*x[i]-
17562.0;
else *sz[i]=0,*dz[i]=0.0,*tz[i]=0.0;
}
return (0);
}

//**********************************************************************
// PROGRAM TO PREDICT THE INITIAL FLOW SURFACE PROFILE
//**********************************************************************

#include <math.h>

double valintLAT(double *y[200],int m,double q[200],double by,double ey,
double PR[200],double xl,double xr,double xo,double h)
{
int i,j,ite,n,nn,nr;
double x1,x2,x3,H,A,B,C,G[200],xint,fm,f,test,min,
root1[200],root2[200],fl,fr,xm,aa,bb,jj,stor[200][2],ii,yc,nnc;
double func(double A,double C,double x);
const double tol=1E-4, g=9.81;
const int Mite=20;

for (j=0;j<=m+1;j++)
{ y[j]=&G[j];
}
*y[0]=by;
*y[m+1]=ey;
aa=xl;
bb=xr;

H=(*y[0])+(0.5*(q[0]*q[0])/(2.0*g*(*y[0])*(q[0])))+(PR[0]);

for (i=1;i<=m;i++)
{ C=(q[i]*q[i])/(2.0*g);
jj=fabs(xl-xr)/(0.1);
n=jj;
A=PR[i]-H;
nr=1;
for (j=1;j<n;j++)
{ ii=j;

fl=func(A,C,xl);
fr=func(A,C,(xl+(ii*0.1)));

if (fl*fr<0.0){
  xr=xl+(ii*0.1);
  fm=func(A,C,xr);
  f=func(A,C,xl);
x1=xr-xl;
(xint=xl-xr,xr);
}
ite=0;
while (ite<=\text{Mite}){
    xint=xint/2.0;
    xm=x1+xint;
    fm=func(A,C,xm);
    if(fm<=0.0){
        x1=xm;
    }
    test=fabs(xint);
    if (test<=\text{tol}||fm==0.0){
        stor[i][nr]=xm;
        nr++;
        break;
    }
    ite++;
}

xl=xr;
xl=aa;
xr=bb;

for (j=1;j<=m;j++){
    root1[j]=stor[j][1];
    root2[j]=stor[j][2];
}

nnc=fabs(xo)/h;
nn=nnc;
yc=(q[nn]*q[nn])/g;
yc=pow(yc,1.0/3);
min=by;
for (i=1;i<=m;i++){
    if (root2[i]<=min){
        min=root2[i];
        nn=i;
    }
}
for (i=1;i<=m;i++){
    if(root2[i]==0.0||root2[i]<1e-10){
        root2[i]=yc;
    }
}
for (i=1;i<=m;i++){
    if(root1[i]==0.0||root1[i]<1e-10){
        root1[i]=yc;
    }
}
for(j=1;j<=nn;j++){
    *y[j]=root2[j];
}
for(j=nn+1;j<=m;j++){
    *y[j]=root1[j];
}
return(0);
/*********************************************************************************/

// PROGRAM THAT RETURNS THE VALUE OF THE FUNCTION
/*********************************************************************************/

double func(double A, double C, double x) {
    double f;
    f = pow(x, 3.0) + A * pow(x, 2.0) + C;
    return(f);
}
/*********************************************************************************/
Appendix C. Computer Programs and Flow Charts

Computational flow chart for the solution of the free overfall problem

*****************************NOTATION*****************************

byu = the specified flow depth at the inflow section of the computational domain;
eyl = elevation of the lower nappe at the outflow section of the computational domain;
byl = elevation of the lower nappe at the brink section;
q = discharge per unit width;
l = horizontal length of the channel;
zb[i] = elevation of the lower nappe at various nodal points;
y[i] = the correction flow depth from the Newton-Raphson method at node i;
lf = length of the fixed bed;
xo = the coordinate of the upstream end of the fixed bed = -lf;
Kf = the slope of the water surface at the inflow section;
mf and ml = number of nodes in the fixed bed and free jet portion of the computational domain respectively;
xl and xr = the lower and upper interval values of the root for the bisection method.

**************************************************************************************/

#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
#include<math.h>
#include "YTZ.h"

int main ()
{
  int i,j,m,ite,kk,*dir[100],DIR[100],mm,mf,ml,nn;
double q,*sf[100],*df[100],*tf[100],B,SF[100],DF[100],TF[100],*y[100],
    Y[100],R[100],syn,*x[70],xx[70],lf,*pr[100],PR[100],xl,xy,HH, jj, kf,
    Z[100],*z[100],h,ey,by,k,*a[10000],*b[100],*c[10000],*y[100],
    ZZ[100],AA[10000],*sz[100],*dz[100],*tz[100],*sz[100],*dz[100],*tz[100],
    *ps[100],PS[100],DX[100],*dx[100],PDR[100],PSL[100],yc,ZZ[100],*zf[100],
    zf[100],xo,*Zbi[100],zb[100],*Zdb[100],db[100],erroru,errorl,k1,eyl,by
l,ku,byu,eyu,ss,zsl[100],ss[100],sym,pb,*p[70],aa,bb,cc,PPP[70],*pk[100],
    PK[100],nm,synz,*cp[100],CP[100],uu;
const double beta=1.0E-6;

cout<<"ENTER THE VALUE OF 'h'"<<"\n"; cin>>h;
cout<<"ENTER THE VALUE OF 'B'"<<"\n"; cin>>B;
cout<<"ENTER THE VALUE OF 'q'"<<"\n"; cin>>q;
cout<<"ENTER THE VALUE OF 'Lf'"<<"\n"; cin>>lf;
cout<<"ENTER THE BOUNDARY CONDITIONS AT THE END SECTIONS:"<<"\n";
cout<<"DEPTH OF THE FLOW 'byu'"<<"\n"; cin>>byu;
cout<<"ELEVATION OF THE LOWER NAPPE 'eyl'"<<"\n";
cin>>eyl;
cout<<"ENTER THE VALUE OF 'XL' "<<"\n";
cin>>xl;
cout<<"ENTER THE VALUE OF 'XR' "<<"\n";
cin>>xr;
cout<<"ENTER THE VALUE OF 'XO' "<<"\n";
cin>>xo;
cout<<"ENTER THE HORIZONTAL LENGTH OF THE LOWER NAPPE 'l' "<<"\n";
cin>>l;

LL=l*1000.0;
HH=h*1000.0;
jj=LL/HH;
ml=jj-1;
Z[0]=byu;
LL=lf*1000.0;
jj=(LL/HH);
mf=jj;
m=mf+ml+1;

for (j=1;j<=m;j++){
tf[j]=0;
df[j]=0;
sf[j]=0;
b[j]=0;
y[j]=0;
r[j]=0;
dir[j]=0;
}
for (j=0;j<=m+1;j++){
z[j]=0;
sz[j]=0;
dx[j]=0;
tz[j]=0;
pr[j]=0;
ps[j]=0;
px[j]=0;
Zf[j]=0;
Zbi[j]=0;
DZb[j]=0;
p[j]=0;
pk[j]=0;
x[j]=0;
cp[j]=0;
}

z[0]=0;
sz[0]=0;
dz[0]=0;
tz[0]=0;
pr[0]=0;
ps[0]=0;
px[0]=0;
Zf[0]=0;
Zbi[0]=0;
DZb[0]=0;
p[0]=0;
pk[0]=0;
x[0]=0;
cp[0]=0;

proftra(pr,mf,h,dx,l,xo);
for (j=0;j<=mf;j++){
    zb[j]=*pr[j];
}
slopchan(sz,dz,tz,mf,h,l,xo);
for (i=0;i<=mf;i++){
    SZ[i]=*sz[i];
    DZ[i]=*dz[i];
    TZ[i]=*tz[i];
}

byl=zb[mf];
intilower(Zbi,ml,h,byl);
for (i=1;i<=ml;i++){
zb[mf+i]=Zbi[i];
}
zb[m]=eyl;
valintF(z,m,q,byu,zb,xl xr);
for (j=1;j<=m;j++){
    Z[j]=z[j];
}
for (i=1;i<=m*m;i++) {
    a[i]=0;
    c[i]=0;
}
kf=waterslope (SZ,q,Z,B);
**************************************************************
erroru=beta;
errorl=beta;
ite=0;
while ((erroru>=beta)&&(errorl>=beta)&&(ite<=400)) {
    derlowernappe(sz,dz,tz,ml,h,zb);
    for (i=1;i<=ml+1;i++) {
        SZ[i+mf]=sz[i];
        DZ[i+mf]=dz[i];
        TZ[i+mf]=tz[i];
    }
    //******************COMPUTATION OF THE UPPER NAPPE PROFILE**************
    for (i=1;i<=m;i++) {
        if (i<=mf) nm=0.0;  //smooth boundary resistance formula
            else nm=0.0284;  //rough bed channel
        SF[i]= cosf(SZ[i],Z[i]);
        DF[i]= codf(SZ[i],Z[i],q);
        TF[i]= cotf(SZ[i],DZ[i],TZ[i],Z[i],q,B,nm);
    }
    //******SOLVE THE VALUE OF SYN = Z[-1] AND THE RIGHT HAND SIDE EQUATIONS
    solvF(syn,Z,B,h,q,kf,SZ,DZ,TZ);
    comrF(SF,DF,TF,m,h,r,syn,Z,q,mf,DZ);
    for (i=1;i<=m;i++) {
        R[i]= r[i];
    }
    //*******************NUMERICAL DETERMINATION OF THE JACOBIAN MATRIX*****
    linerF(a,R,SF,DF,TF,h,Z,m,SZ,DZ,TZ,B,kf,q,mf);
    kk=0;
    for (i=1;i<=m;i++) { 
        for (j=1;j<=m;j++){
            A[i][j]=a[j+kk];
        }
        kk=kk+m;
    }
    //***********************SOLVE THE CORRECTION 'Y' FOR THE SOLUTION**************
    decom (A,m,dir,c);
    for (i=1;i<=m*m;i++) {
        AA[i]=c[i];
    }
    for (j=1;j<=m;j++){
        DIR[j]=dir[j];
    }
solu ( AA,m,DIR,R,b);
    for (i=1;i<=m;i++) {
        Y[i]=b[i];
    }
}
erroru=0.0;
for (i=1;i<=m;i++){
    erroru=erroru+fabs(Y[i]);
    Z[i]=Z[i]+Y[i];
}

//***********COMPUTATION OF THE LOWER NAPPE PROFILE**************
zb[m]=eyl;
for (i=0;i<=ml+1;i++){
    zz[i]=Z[mf+i];
    zzb[i]=zb[mf+i];
}
curlower(DZb,ml,h,q,zz);
for (i=1;i<=ml;i++){
    dzb[i]=*DZb[i];
}
comrlower(dzb,ml,h,r,zzb);
for (i=1;i<=ml;i++){
    R[i]=*r[i];
}
linerlower(a,R,dzb,h,zzb,ml);
kk=0;
for (i=1;i<=ml;i++) {
    for (j=1;j<=ml;j++){
        A[i][j]=*a[j+kk];
    }
    kk=kk+ml;
}
decon (A,ml,dir,c);
    for (i=1;i<=ml*ml;i++){  
        AA[i]=*c[i];
    }
for (j=1;j<=ml;j++){
    DIR[j]=*dir[j];
}
solu (AA,ml,DIR,R,b);
    for (i=1;i<=ml;i++){
        Y[mf+i]=*b[i];
    }
errorl=0.0;
for (i=1;i<=ml;i++){
    errorl=errorl+fabs(Y[mf+i]);
    zb[mf+i]=zb[mf+i]+Y[mf+i];
}
ite++;
}

//***********THE BED PRESSURE PROFILE**************
pressF(ps,q,h,DZ,Z,syn,m-1,SZ,mf,zb);
for (i=0;i<=m-1;i++){
    PS[i]=*ps[i];
}

//***********OUTPUT THE SIMULATION RESULTS*************
cout<<"BED PROFILE FOR THE FIXED BED PORTION"
for (j=0;j<=mf;j++){
    cout<<zb[j]<<endl;
}
cout<<"THE UPPER FLOW SURFACE PROFILE"
for (j=0;j<=m;j++){
    cout<<Z[j]<<endl;
}
cout<<"THE LOWER NAPPE PROFILE";
for (j=mf+1;j<=m;j++) { 
cout<<zb[j]<<endl; }
cout<<"THE BED PRESSURE UPSTREAM OF THE END SECTION"<<"\n" ;
for (j=0;j<=m-1;j++) { 
cout<<PS[j]<<"\n" ;
}
cout<<"errorl="<<errorl1<<"\t"<<"number of iteration="<<ite<<"\n" ;
cout<<"erroru="<<erroru<<"\t"<<"number of iteration="<<ite<<"\n" ;
//**********************************************************************
cout<<"\n" <<"PRESS ANY KEY TO CONTINUE..."; getch(); return(0);
//**********************************************************************
// PROGRAM TO EVALUATE THE JACOBIAN MATRIX ELEMENTS NUMERICALLY
//**********************************************************************
double linerF(double *a[100],double R[100],double SF[100],
double DF[100],double TF[100],double h,double Z[100],int &m,
double SZ[100],double DZ[100],double TZ[100],double B,double k,double q,
int mf) {
  #include "YTZ.h"
  int i,1l,1m,j;jj;
double f[10000],b[150][150],RR1[100],RR2[100],*r[100],ii,*sf[100],
df[100],*tf[100],syn,hh,nm;
  const double gama=1E-4;
  for (j=1;j<=m;j++) { 
    tf[j]=0;
    df[j]=0;
    sf[j]=0;
    r[j]=0;
  }
  for (j=1;j<=m*m;j++) { 
    a[j]=&f[j];
  }
  for (i=1;i<=m;i++) { 
    RR1[i]=-1.0*R[i];
    for (j=1;j<=m;j++)
      b[i][j]=0.0 ;
  }
  for (j=1;j<=m;j++) { 
    hh=0.001*Z[j];
    if(fabs(Z[j])<gama ) {
      hh=gama;
    }
    Z[j]=hh+Z[j];
    for (i=1;i<=m;i++) { 
      if (i<=mf) nm=0.0;     //SMOOTH BED CHANNEL
      if (i<mf) nm=0.0284;   //ROUGH BED CHANNEL
      else nm=0.0;
      SF[i]= cosf(SZ[i],Z[i]);
      DF[i]= cofs(SZ[i],Z[i],q);
      TF[i]= cotf(SZ[i],DZ[i],TZ[i],Z[i],q,B,nm);
    }
  }
solvF(syn,Z,B,h,q,k,SZ,DZ,TZ);
comrF(SF,DF,TF,m,h,r,syn,Z,q,mf,DZ);
for (i=1;i<=mf;i++) { 
  RR2[i]=(-1.0)*(*r[i]);
}
314 Appendix C. Computer Programs and Flow Charts

\[ Z[j] = Z[j] - hh; \]
for (i=1; i<=m; i++) {
    b[i][j] = (RR2[i] - RR1[i]) / hh;
}
}
ll=0;
for (i=1; i<=m; i++) {
    for (j=1; j<=m; j++) {
        a[j+ll] = b[i][j];
    }
    ll = ll + m;
}
return(0);
}

//**********************************************************************
// PROGRAM TO EVALUATE THE RIGHT HAND SIDE EQUATIONS FROM THE NEWTON-// RAPHSON RULE
//**********************************************************************
double comrF(double SF[100], double DF[100], double TF[100], int &m, double h, double*r[100], double &syn, double Z[100], double q, int mf, double DZ[100])
{
    int i, j;
    double  u[100], v[100], w[100], x[100];
    const double g=9.81, om=0.95;
    for (j=1; j<=m; j++) {
        r[j] = &x[j];
    }
    w[1] = syn*(-6.0+(DF[1]*h*h))+(6.0*h*h*h*TF[1]);
    u[1] = Z[0]*(18.0+(6.0*SF[1]*h)-(6.0*DF[1]*h*h));
    v[1] = Z[1]*(-18.0-(12.0*SF[1]*h)+(3.0*DF[1]*h*h))
        + Z[2]*(6.0+(6.0*SF[1]*h)+(2.0*DF[1]*h*h));
    for (i=2; i<m; i++) {
        if (i!=mf) {
            w[i] = Z[i-2]*(-6.0+(DF[i]*h*h))+(6.0*h*h*h*TF[i]);
            u[i] = Z[i-1]*(18.0+(6.0*SF[i]*h)-(6.0*DF[i]*h*h));
            v[i] = Z[i]*(-18.0-(12.0*SF[i]*h)+(3.0*DF[i]*h*h))
                + Z[i+1]*(6.0+(6.0*SF[i]*h)+(2.0*DF[i]*h*h));
            *r[i] = -1.0*(u[i]+v[i]+w[i]);
        }
        if (i==mf) {
            u[i] = (1.0/60.0)*((-50.0*Z[i-5])+(305.0*Z[i-4])-(780.0*Z[i-3])+(1070.0*Z[i-2])-(770.0*Z[i-1])+(225.0*Z[i]));
            v[i] = (2.0*h*h)*((q*Z[i]*Z[i])/(g*q q))+(2.0*h*h*om*DZ[i]);
            *r[i] = -1.0*(u[i]+v[i]+w[i]);
        }
    }
    w[m] = Z[m-3]*((-32.0*DF[m]*h*h)-(112.0*SF[m]*h)-(168.0))
        + Z[m-2]*((72.0*DF[m]*h*h)+288.0)+(228.0*SF[m]*h*h))
        +(24.0*h*h*h*TF[m]);
    u[m] = Z[m-4]*((6.0*DF[m]*h*h)+(22.0*SF[m]*h)+36.0);
    v[m] = Z[m]*((50.0*DF[m]*h*h)+(50.0*DF[m]*h*h)+60.0)+Z[m-1]*(-216.0-
        (208.0*SF[m]*h)-(96.0*DF[m]*h*h));
    *r[m] = -1.0*(u[m]+v[m]+w[m]);

return (0);
Appendix C. Computer Programs and Flow Charts

// **********************************************************************
// PROGRAM TO EVALUATE PART OF THE CURVATURE TERM FOR THE LOWER NAPPE
// PROFILE
//**********************************************************************

double curlower(double *DZb[100], int &m, double h, double q, double y[100])
{
    int i, j;
    double x[100], aa;
    const double g = 9.81, om = 0.95;

    for (j = 1; j <= m; j++)
    {
        DZb[j] = &x[j];
    }

    for (i = 1; i <= m; i++)
    {
        if (i < m)
            aa = y[i - 1] - (2.0 * y[i]) + y[i + 1];
        else
            aa = y[i - 2] - (2.0 * y[i - 1]) + y[i];
        *DZb[i] = (-1.0 / om) * (((g * y[i] * y[i]) / (q * q)) + ((0.50 / (h * h)) * (aa)));
    }
    return (0);
}

//**********************************************************************
// INITIAL FLOW DEPTHS FOR THE LOWER NAPPE PROFILE
//**********************************************************************

double intilower(double *Zbi[100], int &m, double h, double Zbf)
{
    int i, j;
    double x[100], vx[100], ii, mm;
    for (j = 1; j <= m; j++)
    {
        Zbi[j] = &vx[j];
    }

    for (i = 1; i <= m; i++)
    {
        *Zbi[i] = Zbf;
    }
    return(0);
}

//**********************************************************************
// PROGRAM TO EVALUATE THE JACOBIAN MATRIX ELEMENTS NUMERICALLY FOR
// THE LOWER NAPPE PROFILE
//**********************************************************************

double linerlower(double *a[100], double R[100], double DZb[100], double h, double Z[100], int&m)
{
    int i, ll, j, mm, jj;
    double f[2000], b[150][150], RR1[100], RR2[100], *r[100], ii, *dzb[100], hh, synz;
    const double gama = 1E-4;
    comrlower(DZb[], m, h, r[]);

double comrlower(double DZb[], int m, double h, double* r[]);
{
    for (j = 1; j <= m; j++)
    {
        r[j] = 0;
    }
    for (j = 1; j <= m * m; j++)
    {
        a[j] = &f[j];
    }
    for (i = 1; i <= m; i++)
    {
        RR1[i] = -1.0 * R[i];
        for (j = 1; j <= m * m; j++)
            b[i][j] = 0.0;
    }
    for (j = 1; j <= m; j++)
    {
        hh = 0.001 * Z[j];
if(fabs(Z[j])<gama ){
    hh=gama;
}

Z[j]=hh+Z[j];

comrlower(DZb,m,h,r,Z);
for (i=1;i<=m;i++){
    RR2[i]=(-1.0)*(*r[i]);
}  
Z[j]=Z[j]-hh;

for (i=1;i<=m;i++){
    RR2[i]=(-1.0)*(*r[i]);
}

for (i=1;i<=m;i++) {
    j=1;j<=m;j++){
    *a[j+ll] = b[i][j];
}
    ll=ll+m;
}

return(0);
}

//**********************************************************************
// PROGRAM TO COMPUTE THE DERIVATIVES OF THE LOWER NAPPE PROFILE
//**********************************************************************
double derlowernappe(double *sz[100],double *dz[100],double *tz[100],
int m,double h,double Zb[100])
{
    int i,j,ld,M,nl,nr,np,ii;
    double u[100],v[100],w[100],x[100];
    double *c[100],cc[100],PR[100],l,suml,sumr,DB[10];
    double dercof (double *c[],int np,int nl,int nr,int ld,int m);

    for (i=0;i<=m+1;i++){
        sz[i]=&u[i];
        dz[i]=&v[i];
        tz[i]=&w[i];
}

    for (i=1;i<=m+3;i++){
        PR[i]=Zb[i];
}

    for (i=2;i<=6;i++){
        PR[m+i]=PR[m+1];
}

    for (j=1;j<=m+1;j++){
        if (j=1){
            nl=1;
            nr=3;
        }
        if (j>=2){
Appendix C. Computer Programs and Flow Charts

nl=2;
nr=2;
}
np=nr+nl+1;
for (ld=1;ld<=3;ld++){
M=4;
dercof(c,np,nl,nr,ld,M);
for (i=1;i<=np;i++){
    cc[i]=*c[i];
}
suml=0.0;
for (i=2;i<=nl+1;i++){
    suml=suml+cc[i]*PR[j-i+1];
}
sumr=0.0;
i=1;
for (i=np;i>=nl+2;i--){
    sumr=sumr+cc[i]*PR[j+ii];
    ii=ii+1;
}
DB[ld]=sumr+suml+PR[j]*cc[1];
}
//**********************************************************************
//sz[j]=DB[1]/h;
*dz[j]=2.0*(DB[2])/(h*h);
*tz[j]=6.0*(DB[3])/(h*h*h);
} return (0); 
//**********************************************************************
// PROGRAM TO EVALUATE THE SAVITZKY-GOLAY FILTER COEFFICIENTS FOR
// ESTIMATING THE DERIVATIVES
// Adopted from Press et al. (2002, #14.8)
//**********************************************************************

double dercof(double *c[150],int np,int nl,int nr ,int ld,int m)
{
    int imj,ipj,j,k,kk,*dir[100],DIR[100],i;
    double d,fac,sum,a[150][150],b[150],*CC[2000],AA[2000],*bb[100],D[100];

double decom(double a[150],int m,int *indx[],double *A[]);
double solu(double A[],int m,int indx[],double R[],double *y[]);

for (i=1;i<=(m+1)*(m+1);i++){
    CC[i]=0;
}
for (i=1;i<=m+1;i++){
    dir[i]=0;
    bb[i]=0;
}
for (i=1;i<=np+1;i++){
    c[i]=&D[i];
}
if (np<nl+nr+1||nl<0||nr<0||ld>m||nl+nr<m ){
    cout<<"ERROR"<<endl;
}
for (ipj=0;ipj<=(m<<1);ipj++){
    sum=(ipj?0.0:1.0);
    for (k=1;k<=nr;k++){
        sum+=pow((double)k,(double)ipj);
    }
    for (k=1;k<=nl;k++){
318 Appendix C. Computer Programs and Flow Charts

```c
sum+=pow((double)-k,(double)ipj);
}
mm=ipj;
if(mm>(2.0*m-ipj)){
  mm=2.0*m-ipj;
}
for (imj=-mm;imj<=mm;imj+=2)
  a[1+(ipj+imj)/2][1+(ipj-imj)/2]=sum;
}
decom(a,m+1,dir,CC);
for (i=1;i<=(m+1)*(m+1);i++){
  AA[i]=*CC[i];
}
for (i=1;i<=m+1;i++){
  DIR[i]=*dir[i];
}
for (j=1;j<=m+1;j++){
  b[j]=0.0;
}
b[ld+1]=1.0;
solu(AA,m+1,DIR,b,bb);
for (i=1;i<=m+1;i++){
  b[1]=*bb[i];
}
for (kk=1;kk<=np;kk++){
  *c[kk]=0.0;
}
for (k=-nl;k<=nr;k++){
  sum=b[1];
  fac=1.0;
  for (mm=1;mm<=m;mm++){
    sum+=b[mm+1]*(fac*=k);
    kk=((np-k)%np)+1;
    *c[kk]=sum;
  }
return(0);
}
//*********************************************************************************
// PROGRAM TO EVALUATE THE FIXED BED ELEVATIONS
***********************************************************************************/
double proftra0(double *pr[200],int m,double h,double *dx[200],
double L,double xo)
{
  int i,j;
  double x[200],PR[200],ii,a,y[200],DX[200];
  for (j=0;j<=m+1;j++){
    pr[j]=&PR[j];
    dx[j]=&DX[j];
  }
  for (i=0;i<=m+1;i++){
    ii=i;
    x[ii]=xo+(ii*h);
    *dx[ii]=x[ii];
    *pr[ii]=0.0;
  }
  return (0);
}
```
Appendix C. Computer Programs and Flow Charts

//**********************************************************************
// PROGRAM TO EVALUATE THE DERIVATIVES OF THE FIXED BED
//**********************************************************************
double slopchan0(double *sz[200], double *dz[200], double *tz[200], int m,
double h, double L, double xo)
{
int i, j;
double u[200], v[200], w[200], x[200], ii, aa, bb;
for (i=0; i<=m+1; i++){
    sz[i] = &u[i];
    dz[i] = &v[i];
    tz[i] = &w[i];
}
for (i=0; i<=m+1; i++){
    ii = i;
x[i] = xo + (ii*h);
    *sz[i] = 0.0;
    *dz[i] = 0.0;
    *tz[i] = 0.0;
}
return (0);
}

//**********************************************************************
// PROGRAM TO EVALUATE THE BED PRESSURE PROFILE
//**********************************************************************
double pressF(double *ps[100], double q, double h, double DZ[100],
double Z[100], double syn, int m, double SZ[100], int mf, double zb[100])
{
int i, j;
double BB[100], u, tt[100];
const double g = 9.81, om = 0.95;
for (i=0; i<=m+1; i++){
    ps[i] = &BB[i];
}
for (i=0; i<=m+1; i++){
    tt[i] = 1.0 + pow(SZ[i], 2);
}
*ps[0] = (1.0/g) * (g + (((q*q)/(Z[0]*Z[0]))*(DZ[0] + ((1.0/(2.0*h*h))*(Z[1] -
2.0*Z[0] + syn)))));
for (j=mf+1; j<=m; j++){
    DZ[j] = (zb[j-1] - 2.0*zb[j] + zb[j+1])/(h*h);
    if (j<mf) u = (Z[j+1] - 2.0*Z[j] + Z[j-1]);
    else if (j==mf) u = (1.0/60.0) * ((-50.0*Z[j-5]) + (305.0*Z[j-4]) -
(780.0*Z[j-3]) + (1070.0*Z[j-2]) - (770.0*Z[j-1]) + (225.0*Z[j]));
*ps[j] = (1.0/g) * ((q*g)/(Z[j]*Z[j]*tt[j])) * (om*DZ[j] +
((1.0/(2.0*h*h))*(u))));
}
return (0);
}

//**********************************************************************
// PROGRAM TO PREDICT THE NODAL VALUE AT j=-1 USING THE SECOND
// BOUNDARY CONDITION AT THE INFLOW SECTION
//**********************************************************************
double solvF(double &syn, double y[100], double B, double h, double q,
double k, double SZ[100], double DZ[100], double TZ[100])
{
{
double a,b,c,d,e,D,sf,df,tf,fs,dd;
const double g=9.81,n=0.0284,om=0.95;
//**********************************************************************
sf=(2.0*SZ[0])/y[0];
df=(4.0+(4.0*SZ[0]*SZ[0]))*((-1.0/(y[0]*y[0]))+((g*y[0])/(q*q)));
tf=(4.0+(4.0*SZ[0]*SZ[0]))*(((g*y[0]*SZ[0])/(q*q))+((g*y[0]*fs)/(q*q)))+
  4.0*om*((T2[0]/2.0)+(SZ[0]*DZ[0]/y[0]));

dd=y[0];
D=pow(dd,4.0);
D=pow(D,1.0/3.0);
fs=(n*n*q*q)/(y[0]*y[0]*D);

a=-6.0+(df*h*h);
b=(-6.0*df*h*h)+(6.0*sf*h)+18.0;
c=(3.0*df*h*h)-(12.0*sf*h)-18.0;
d=6.0+(6.0*sf*h)+(2.0*df*h*h);
syn=(-1.0/(b+6.0*a))*(y[1]*d+c*y[0]+a*(6.0*h*k-2.0*y[1]-3.0*y[0]))+
  (6.0*h*h*h*tf));
return (0);
}
//**********************************************************************
// PROGRAM TO EVALUATE THE RIGHT HAND SIDE EQUATIONS FROM THE NEWTON-
// RAPHSON RULE FOR THE LOWER NAPPE PROFILE
//**********************************************************************
double comrlower(double DZb[100],int &m,double h,double*r[100],
double Zb[100])
{
int i,j;
double  u[100],x[100],w[100];
for (j=1;j<=m;j++){
  r[j]=&x[j];
}
for (i=1;i<=m;i++){
  u[i]=Zb[i-1]-2.0*Zb[i]+Zb[i+1]-(DZb[i]*h*h);
  *r[i]=-1.0*(u[i]);
}
return (0);
}
//**********************************************************************
Appendix C. Computer Programs and Flow Charts

Start

Read input data

Evaluation of the initial flow surface profile, boundary conditions, and the geometric properties of the trapezoidal profile weirs

Calculation of the nonlinear coefficients associated with the derivative terms of the flow equation

Calculation of the Jacobian matrix elements using a numerical method

Update the flow depths

Calculation of the solution matrix using LU-decomposition method

Calculation of the correction depths

Check the convergence of the solution

No

Yes

Determine the location of the gauge station at maximum discharge and compute the flow depths at this station

Output the predicted overflow depths

Stop

Computational flow chart for the development of discharge rating curves
Appendix C. Computer Programs and Flow Charts

// SIMULATION OF FLOW OVER TRAPEZOIDAL PROFILE WEIRS FOR DEVELOPING DISCHARGE RATING CURVES

// OVERFLOW DEPTHS AND FLOW SURFACE PROFILE COMPUTATIONS FOR FLOW OVER TRAPEZOIDAL PROFILE WEIRS USING A BOUSSINESQ-TYPE FLOW EQUATION. THE FLOW DEPTH, SLOPE AND CURVATURE OF THE WATER SURFACE AT THE INFLOW SECTION OF THE FLOW DOMAIN ARE SPECIFIED AS BOUNDARY CONDITIONS.

*****************************NOTATION*****************************
by = the specified flow depth at the inflow section of the computational domain;
q = discharge per unit width;
l = length of the computational domain;
B = width of the channel;
z[i] = depth of the flow at the nodal point i;
k = the slope of the water surface at the inflow section;
kk = curvature of the water surface at the inflow section.

**********************************************************************

#include<iostream.h>
#include<iomanip.h>
#include<conio.h>
#include<math.h>
#include "YTZ.h"

int main ()
{
  int i,j,m,ite,kk,*dir[200],DIR[200];
double q,*sf[200],*df[200],*tf[200],B,SF[200],DF[200],TF[200],*y[200],Y[200],R[200],syn,x[200],l,*pr[200],PR[200],xl,xr,LL,HH,Y2[200],*y2[200],xx,ii,Z[200],*z[200],h,ey,by,k,*a[20000],*b[200],*c[20000],A[150][150],*r[200],error,ZZ[200],AA[20000],*sz[200],*dz[200],*tz[200],SZ[200],DZ[200],T2[200],PSR[200],kkk,dyn,*dx[200],xo,BB,HH[100],P,h1,*Sff[200],SFF[200],f,2b[200],
  const double beta=1.0E-6;

  cout<<"ENTER THE VALUE OF 'h'"<<"\n";
cin>>h;
  cout<<"ENTER THE VALUE OF 'B'"<<"\n";
cin>>B;
  cout<<"ENTER THE VALUE OF 'q'"<<"\n";
cin>>q;
  cout<<"ENTER THE VALUE OF 'L'"<<"\n";
cin>>l;
  LL=l*100.0;
  HH=h*100.0;
  ii=LL/HH;
m=ii;
m=m-1;
  cout<<"ENTER THE BOUNDARY CONDITIONS AT THE INFLOW SECTION:"<<"\n";
  cout<<"DEPTH OF THE FLOW 'by'"<<"\n";
cin>>by;
  cout<<"ENTER THE VALUE OF 'XL'"<<"\n";
cin>>xl;
  cout<<"ENTER THE VALUE OF 'XR'"<<"\n";
cin>>xr;
  Z[0]=by;
  cout<<"ENTER THE VALUE OF 'X0'"<<"\n";
cin>>xo;
  cout<<"ENTER THE HEIGHT OF THE FLOW CONTROL STRUCTURE 'F'"<<"\n";
cin>>F;
//**************************INITIALISING THE VARIABLES******************
for (j=1;j<=m;j++){
    tf[j]=0;
    df[j]=0;
    sf[j]=0;
    b[j]=0;
    y[j]=0;
    r[j]=0;
    dir[j]=0;
}
for (j=0;j<=m+1;j++){
    z[j]=0;
    sz[j]=0;
    dz[j]=0;
    tz[j]=0;
    pr[j]=0;
    y2[i]=0;
    dx[i]=0;
    Sff[i]=0;
}
//*********************ELEVATION OF THE BED PROFILE **********************
profembanMM(pr,m,h,xo,BB);
for (j=0;j<=m;j++){
    PR[j]=*pr[j];
}
slopembanMM(sz,dz,tz,m,h,xo,BB);
for (i=0;i<=m+1;i++){
    SZ[i]=*sz[i];
    DZ[i]=*dz[i];
    TZ[i]=*tz[i];
}
//***************INITIALISING THE NODAL VALUES USING THE ENERGY EQUATION
valint(z,m,q,by,PR,xl,xr);
for (j=1;j<=m;j++){
    Z[j]=*z[j];
}
for (i=1;i<=m*m;i++){
    a[i]=0;
    c[i]=0;
}
//****PARAMETERS RELATED TO BOUNDARY CONDITIONS AT THE INFLOW SECTION***
k=waterslope (SZ,q,Z,B);
kkk=curwat (by,q,SZ,B);
//********DETERMINATION OF THE FINAL WATER SURFACE ELEVATIONS**********
error=beta;
ite=0;
while (error>=beta && ite<=100){
    frismooth(q,Z,B,m,Sff);
    for (i=0;i<=m;i++){
        SFF[i]=*Sff[i];
    }
}
//********DETERMINATION OF THE NON-LINEAR COEFFICIENTS OF THE EQUATION
for (i=1;i<=m;i++){
    SF[i]= cosf(SZ[i],Z[i]);
    DF[i]= codf(SZ[i],Z[i],q);
    TF[i]= cotf(SZ[i],DZ[i],TZ[i],Z[i],q,B,SFF[i]);
}
//*****SOLVE THE VALUES OF SYN, DYN AND THE RIGHT HAND SIDE EQUATIONS***
solvR(syn,Z,B,h,q,k,SZ,DZ,TZ,kkk,SFF);
dyn=funcdyn(kkk,Z,syn,k,h);
comR(SF,DF,TF,m,h,r,syn,Z,dyn);
for (i=1;i<=m;i++)
{   
    R[i]=r[i];
}

laughs***********NUMERICAL DETERMINATION OF THE JACOBIAN MATRIX*****
linerR(a,R,SF,DF,TF,h,Z,m,SZ,DZ,TZ,B,k,q,kkk,SFF);
kk=0;
for (i=1;i<=m;i++) {
    for (j=1;j<=m;j++){
    A[i][j]=a[j+kk];
    }
    kk=kk+m;
}

laughs***********SOLVE THE CORRECTION 'Y' FOR THE SOLUTION************
decon (A,m,dir,c);
for (i=1;i<=m*m;i++)
{   
    AA[i]=c[i];
}
for (j=1;j<=m;j++){
    DIR[j]=dir[j];
}
solu ( AA,m,DIR,R,b);
for (i=1;i<=m;i++)
{   
    Y[i]=b[i];
}

error=0.0;
for (i=1;i<=m;i++)
{   
    error=error+fabs(Y[i]);
    Z[i]=Z[i]+Y[i];
    ite++;
}
}
h1=OverDep(P,Z,BB,h,xo);

laughs************************OUTPUT THE SIMULATION RESULTS***************
cout<<"THE OVERFLOW DEPTH"<<endl;
cout<<h1<<endl;
cout<<"FLOW SURFACE PROFILE"<<endl;
for (i=0;i<=m;i++)
{   
    cout<<Z[i]<<endl;
}
cout<<"error="<<error<<"t"<<"number of iteration="<<ite<<endl;
cout<<endl<<"PRESS ANY KEY TO CONTINUE...";
getch();
return(0);

laughs**************************************************************************
laughs**************************************************************************
*****************************************************************************
double linerR(double *a[200],double R[200],double SF[200],
double DF[200],double TF[200],double h,double Z[200],int m,
double SZ[200],double DZ[200],double TZ[200],double B,double k,double q,
double kk,double fs[200])
{
    #include "YTZ.h"
    int i,il,j,mm,jj;
    double f[20000],b[150][150],RR1[200],RR2[200],*r[200],ii,*sf[200],
    *df[200],*tf[200],syn,sh,dy;
    const double gama=1E-4;
for (j=1;j<=m;j++)
{
    tf[j]=0;
    df[j]=0;
    sf[j]=0;
    r[j]=0;
}

for (j=1;j<=m*m;j++)
{
    a[j]=&f[j];
}

for (i=1;i<=m;i++)
{
    RR1[i]=-1.0*R[i];
    for (j=1;j<=m;j++)
    {
        b[i][j]=0.0;
    }
}

for (j=1;j<=m;j++)
{
    hh=0.001*Z[j];
    if(fabs(Z[j])<gama ){
        hh=gama;
    }
    Z[j]=hh+Z[j];
    for (i=1;i<=m;i++)
    {
        SF[i]=cosf(SZ[i],Z[i]);
        DF[i]=codf(SZ[i],Z[i],q);
        TF[i]=cotf(SZ[i],DZ[i],TZ[i],Z[i],q,B,fs[i]);
    }
    solvR(syn,Z,B,h,q,k,SZ,DZ,TZ,kk,fs);
    dyn=funcdyn(kk,Z,syn,k,h);
    comR(SF,DF,TF,m,h,r,syn,Z,dyn);
}

for (i=1;i<=m;i++)
{
    RR2[i]=(-1.0)*(*r[i]);
}

for (i=1;i<=m;i++)
{
    b[i][j]=(RR2[i]-RR1[i])/hh;
}

ll=0;
for (i=1;i<=m;i++)
{
    for (j=1;j<=m;j++)
    {
        *a[j+ll]= b[i][j];
    }
    ll=ll+m;
}
return(0);

 double comR(double SF[200],double DF[200],double TF[200],int m,
 double h,double*r[200],double &syn,double Z[200],double &dyn)
{
    int i,j;
    double u[200],v[200],w[200],x[200];
    for (j=1;j<=m;j++)
    {
        r[j]=&x[j];
    }
\[ w[1] = \text{dyn} * ((-2.0*DF[1]*h*h)-(2.0*SF[1]*h)+(12.0)) + \text{syn} * ((12.0*DF[1]*h*h)-72.0+8.0*SF[1]*h))+(24.0*h*h*TF[1]); \]
\[ u[1] = Z[2] * ((6.0*DF[1]*h*h)+(2.0*SF[1]*h)+36.0); \]
\[ v[1] = Z[1] * ((-40.0*SF[1]*h)+(20.0*DF[1]*h*h)-120.0)+Z[0] *(144.0+(12.0*SF[1]*h)-(36.0*DF[1]*h*h)); \]
\[ w[2] = \text{syn} * ((-2.0*DF[2]*h*h)-(2.0*SF[2]*h)+(12.0)) + \text{syn} * ((-10.0*SF[2]*h)+(20.0*DF[2]*h*h)-120.0)+Z[1] *(144.0+(12.0*SF[2]*h)-(36.0*DF[2]*h*h)); \]
\[ u[2] = Z[3] * ((6.0*DF[2]*h*h)+(2.0*SF[2]*h)+36.0); \]
\[ v[2] = Z[2] * ((-40.0*SF[2]*h)+(20.0*DF[2]*h*h)-120.0)+Z[1] *(144.0+(12.0*SF[2]*h)-(36.0*DF[2]*h*h)); \]
\[ \text{for } (i=3;i<=m-1;i++){ \]
\[ w[i] = Z[i-3] * ((-2.0*DF[i]*h*h)-(2.0*SF[i]*h)+(12.0)) + Z[i-2] * ((12.0*DF[i]*h*h)-72.0+8.0*SF[i]*h))+(24.0*h*h*TF[i]); \]
\[ u[i] = Z[i+1] * ((6.0*DF[i]*h*h)+(2.0*SF[i]*h)+36.0); \]
\[ v[i] = Z[i] * ((-40.0*SF[i]*h)+(20.0*DF[i]*h*h)-120.0)+Z[i-1] *(144.0+(12.0*SF[i]*h)-(36.0*DF[i]*h*h)); \]
\[ *r[i] = -1.0*(u[i]+v[i]+w[i]); \]
\[ \}
\]
\[ w[m] = Z[m-3] * ((-32.0*DF[m]*h*h)-(112.0*SF[m]*h)-(168.0)) + Z[m-2] * ((72.0*DF[m]*h*h)+288.0+(228.0*SF[m]*h)+4.0*TF[m]); \]
\[ u[m] = Z[m-4] * ((6.0*DF[m]*h*h)+(22.0*SF[m]*h)+36.0); \]
\[ v[m] = Z[m] * ((70.0*SF[m]*h)+(50.0*DF[m]*h*h)+60.0)+Z[m-1] * ((-216.0-(208.0*SF[m]*h)-(96.0*DF[m]*h*h)); \]
\[ *r[m] = -1.0*(u[m]+v[m]+w[m]); \]
\]
\[ \text{return (0); } \]

// ***********************************************
// PROGRAM TO EVALUATE THE NODAL VALUE AT j=-2 USING THE BOUNDARY
// CONDITION AT THE INFLOW SECTION
// ***********************************************

double funcdyn(double kk, double Z[200], double syn, double k, double h)
{
    double dyn;
    dyn=6.0*h*k+12*syn-15.0*Z[0]+4.0*Z[1]-6.0*h*h*kk;
    return(dyn);
}

// ***********************************************
// PROGRAM TO EVALUATE THE NODAL VALUE AT j=-1 USING THE BOUNDARY
// CONDITION AT THE INFLOW SECTION
// ***********************************************

double solvR(double &syn, double y[200], double B, double h, double q, double k, double SZ[200], double DZ[200], double TZ[200], double kk, double fs[200])
{
    double a, b, c, d, e, D, sf, df, tf, dd, aa, bb, cc;
    const double g=9.81, om=0.95;
    sf=(2.0*SZ[0])/y[0];
    df=(4.0+(4.0*SZ[0]*SZ[0]))*((-1.0/(y[0]*y[0]))+(g*y[0]/(q*q)));
    tf=(4.0+(4.0*SZ[0]*SZ[0]))*((g*y[0]*SZ[0])/(q*q))+((g*y[0]*fs[0])/(q*q))+(4.0*om*((TZ[0]/2.0)+(SZ[0]*DZ[0]/y[0])));
    a=12.0-(2.0*sf*h)-(2.0*df*h*h);
    b=(12.0*df*h*h)+(8.0*sf*h)-72.0;
c = (-36.0*df*h*h) + (12.0*sf*h) + 144.0;
d = -120.0 - (40.0*sf*h) + (20.0*df*h*h);
e = 36.0 + (22.0*sf*h) + (6.0*df*h*h);
aa = a* ((24.0*h*k) - (80.0*y[0]) + (27.0*y[1]) - (36.0*h*h*kk));
bb = b* ((6.0*h*k) - (15.0*y[0]) + (4.0*y[1]) - (6.0*h*h*kk));
cc = -54.0*a - 12.0*b - c;
syn = (1.0/(cc)) * (y[1]*e + d*y[0] + bb + aa + (24.0*h*h*h*tf));
return (0);
}

//****************************************************************************
// PROGRAM TO EVALUATE THE CURVATURE OF THE FLOW SURFACE
//****************************************************************************
double curwat (double yo, double q, double SZ[200], double B)
{
    double Hxx, R, fs, Re, a, b, K, A, P, Cf;
    const double g = 9.81, nu = 1.0e-6, n = 0.0208, BB = 0.0;
    //***********SMOOTH BED CHANNEL*****************************************
    /*
    R = (yo*B) / ((2.0*yo)+B);
    Re = (4.0*R*q) / (yo*nu);
    b = 6.90/Re;
    a = log10(b);
    f = (pow(a, -2)) / (3.24);
    fs = (f*pow(q, 2)) / (8.0*g*R*pow(yo, 2));
    Hxx = ((3.0*f*pow(q, 2)) / (8.0*g*pow(yo, 4.0))) * (SZ[0] - fs);
    */
    //***********ROUGH BED CHANNEL******************************************
    A = B*yo;
    P = B;  //wide channel assumption
    K = pow(A, 5.0/3.0) / (n*pow(P, 2.0/3.0));
    R = A/P;
    Cf = (2.0*q*q*B*B) / (3.0*n*pow(K, 3.0));
    fs = (n*n*q*q) / (yo*yo*pow(R, 4.0/3.0));
    Hxx = Cf * (5.0*B*pow(R, 2.0/3.0) - 4.0*pow(R, 5.0/3.0)) * (BB - fs);
    return (Hxx);
}

//****************************************************************************
// PROGRAM TO ESTIMATE THE OVERFLOW DEPTHS
//****************************************************************************
double OverDep (double P, double Z[200], double BB, double h, double xo)
{
    int i, ii, mm, mmm;
    double h1, PP[100], Hv[100];
    ii = (1000.0*fabs(xo)) / (1000.0*h);
    mmm = ii;
    for (i = 0; i <= mmm; i++){
        PP[i] = P - BB * (xo + i*h);
    }
    ii = 3.0*(Z[mm] - PP[mm]);
    ii = (fabs(ii)*1000.0) / (1000.0*h);
    if (ii >= mmm) mm = Z[mm] - PP[mm];
    else if (ii < mmm){
        mm = mm - ii;
        for (i = mmm-3; i <= mmm; i++){
            Hv[i] = Z[i] - PP[i];
        }
        h1 = 0.0;
        mmm = 0;
        for (i = mmm-3; i <= mmm; i++){
            h1 = min + Hv[i];
        }
    }
328

Appendix C. Computer Programs and Flow Charts

```c
mmm = mmm + 1;
}
h1 = h1 / mmm;
return (h1);

FILE: YTZ.h
HEADER FILE FOR PROVIDING THE SPECIFICATIONS OF SUBPROGRAMS AND
FUNCTIONS THAT ARE USED IN THE MAIN PROGRAMS

FILE: YTZ.h
HEADER FILE FOR PROVIDING THE SPECIFICATIONS OF SUBPROGRAMS AND
FUNCTIONS THAT ARE USED IN THE MAIN PROGRAMS

FILE: YTZ.h
HEADER FILE FOR PROVIDING THE SPECIFICATIONS OF SUBPROGRAMS AND
FUNCTIONS THAT ARE USED IN THE MAIN PROGRAMS

double cosf(double y, double q);
double codf(double y, double q);
double cotf(double y, double q);
double compf(double *pf[], double h, double DZ[], double Z[],
int m, double SZ[]);
double decom(double a[150], int m, double *indx[], double *A[]);
double solu(double A[], double y[], double R[], double *y[]);
double solvbs(double *y[], double B[], double h, double k,
double q[], double k, double SZ[], double DZ[], double T2[], double fs[]);
double comrbs(double SF[], double DF[], double TF[], int m, double
h, double r[], double Z[], double B[], double DZ[], double B[]);
double linerbs(double *a[], double R[], double SF[], double DF[], double
TF[], double h, double Z[], int m, double SZ[], double DZ[], double T2[],
double B[], double k, double q[], double fs[]);
double proLAT(double *pr[], int m, double h, double xo);
double comBqLAT(double h, int m, double B[], double q[]);
double slopLAT(double *sz[], double *dz[], double *tz[], int m, double
h, double xo);
double valintLAT(double *y[], int m, double q[], double by, double ey,
double PR[], double xl, double xr, double xo, double h);
Appendix C. Computer Programs and Flow Charts

double linerF(double *a[], double R[], double SF[], double DF[], double TF[], double h, double Z[], int &m, double SZ[], double DZ[], double TZ[], double B, double k, double q, double mf);
double pressF(double *ps[], double q, double h, double DZ[], double Z[], double syn, int m, double SZ[], int mf, double zb[]);
double derlowernappe(double *sz[], double *dz[], double *tz[], int m, double h, double Zb[]);
double linerlower(double *a[], double R[], double DZb[], double h, double Z[][], int &m);
double comrlower(double DZb[], int &m, double h, double *r[], double Zb[]);
double curlower(double *DZb[], int &m, double h, double q, double y[]);
double eylower(double ZB[], double y[], int ml, double q, double h);
double funcdyn(double kk, double Z[], double syn, double k, double h);
double solvR(double &syn, double y[], double B, double h, double q, double k, double *r[], double &syn, double Z[][], double &dyn);
double linerR(double *a[], double R[], double SF[], double DF[], double TF[], double h, double Z[], int m, double SZ[], double DZ[], double TZ[], double B, double k, double q, double kkk, double fs[]);
double valint(double *y[], int m, double q, double by, double PR[], double xl, double xr);
double curwat(double yo, double q, double SZ[], double B);
double profembanMM(double *pr[], int m, double h, double xo, double BB);
double slopembanMM(double *sz[], double *dz[], double *tz[], int m, double h, double xo, double BB);
double OverDep(double P, double Z[200], double BB, double h, double xo);
Author/s: Zerihun, Yebegaeshet Tsegaye

Title: A one-dimensional Boussinesq-type momentum model for steady rapidly varied open channel flows

Date: 2004-11


Publication Status: Unpublished

Persistent Link: http://hdl.handle.net/11343/39476

File Description: Appendix C

Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.