7 References


Prigogine, I. (1967) Introduction to Thermodynamics of Irreversible Processes 3rd Ed. *Interscience John Wiley and Sons Ltd.*, USA.


8 Publications

Journal Paper


Conference Papers


Appendix A – Self-Organisation Model Code

This Appendix contains a copy of a sample code used to apply a cellular automata model to a field sized sandy lagoon case study. A similar code was used to model a channel constriction or obstruction. These case studies are discussed in Chapter Three.

```c
#include <stdio.h>               /*This header file initiates standard library */
#include <math.h>                /*This header file includes exp, pow and log */
#include <stdlib.h>               /*This header file has more standard library */
#include <string.h>               /*This header file includes string functions */
#include <time.h>                 /*This header file includes the current time */
#include <process.h>              /*This header file allows the WC exe to open*/
#include <conio.h>                /*This header file allows the WC exe to open*/
#include <io.h>                   /*This header file allows the WC exe to open*/
#include <fcntl.h>                /*This header file allows the WC exe to open*/
#include <iostream.h>             /*This header file includes c++ input/output*/
#include <errno.h>
#define a                100000   /*Number of bits (max string length) */
#define b                10000    /*Maximum population size */
#define PI               3.14159265
#define FALSE            0         /*Used by random number generating functions */
#define TRUE             1         /*Used by random number generating functions */
#define DENSITY          1000.0    /*density of water */
#define GRAVITY          9.81      /*acceleration due to gravity */
#define CO_FRICTION      0.01      /*coefficient of friction for bed drag */
#define STILL_DEPTH      1.5       /*still water level (ie do) */
#define FALL_VELOCITY    0.02      /*fall velocity in m/s */
#define SAND_DIAMETER_90 0.0006    /*d90 of the sand */
#define SAND_DIAMETER_50 0.00024   /*d50 of the sand */
#define KIN_VISCOSITY    0.000001  /*kinematic viscosity at 20C (m^2/s) */
#define VON_KARMAN       0.4       /*Von Karman constant */
#define GSX              32        /*grid size in x direction */
#define GSY              16        /*grid size in y direction */
#define X_STEP           50.0      /*size of dx */
#define Y_STEP           50.0      /*size of dy */
#define GCNU             1.0       /*maximum n constraint */
#define GCNL             (-1.0)    /*minimum n constraint */
#define AA               600       /*maximum length of (x) grid */
#define BB               600       /*maximum width (y) of grid */
#define SPECN            0.0       /*specified height above still water level */
#define SPECV            1.0       /*specified v */
#define SPECU            1.0       /*specified u */
#define SED_UNIT         0.01      /*sediment increase/decrease in each cell */
#define ANGLE_REPOSE     (35.0*PI/180.0)/*angle of repose in radians */
#define RELATIVE_DENSITY 2.65      /*relative density of soil and water */
#define POROSITY         0.6       /*porosity of sand */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <process.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <iostream.h>
#include <errno.h>

void mistake(void);
void other_mistake(int om_no);
```
void set_grid_size(int sgs_check1);
void set_grid_type(int sgt_check1);
void self_organisation(void);

void RandomInitialise(int ij,int kl);
double RandomUniform(void);
double RandomGaussian(double mean,double stddev);
int RandomInt(int lower,int upper);
double RandomDouble(double lower,double upper);

********************************************************************************
*The following set file prototypes, that are called by the main function
********************************************************************************/

FILE *grid_sizef;
FILE *grid_size2f;
FILE *grid_size3f;
FILE *grid_typef;
FILE *grid_type3f;
FILE *grid_vf;
FILE *grid_jf;
FILE *grid_nf;
FILE *grid_bitsf;
FILE *water_countf;

FILE *bestvf;
FILE *bestuf;
FILE *bestnf;
FILE *bestdef;
FILE *bestde2f;
FILE *small_arcef;
FILE *small_arce2f;
FILE *small_arcof;
FILE *ddepthf;
FILE *sed_movef;

********************************************************************************
******************************Main Function*************************************
********************************************************************************/

void main(){
    self_organisation();
}

/********************************************************************************
***********************Self-Organisation Function*******************************
********************************************************************************/

void self_organisation(){
    double so_answer;
    double (*so_v)[BB];
    double (*so_u)[BB];
    double (*so_n)[BB];
    double (*so_de)[BB];
    double (*so_vol)[BB];
    double (*so_energy)[BB];
    double (*so_histogram)[HBB]; // used to plot information entropy histogram
    double (*so_crit_vel)[BB]; // critical velocity at each point
    int (*so_gtype)[BB]; // grid type
    int (*so_all)[3]; // used in randomising points
    int (*so_random); // used in randomising points
    int so_ij; // used in for loops
    int so_jj; // used in for loops
    int so_kj; // used in for loops
    int so_pj; // used in for loops
    int so_qk; // used in for loops
    int so_rq; // used in loops
    int so_gsx; // grid size of x
    int so_gsy; // grid size of y
    int so_gsx;

}
double so_tempf;
char so_temps[50];
double so_water_en_st;
double so_water_en_end;
double so_delta_en;
double so_delta_v3;
double so_delta_u3;
double so_delta_esedv;
double so_delta_ewatv;
double so_delta_esedu;
double so_delta_ewatu;
double so_sed_pot;
double so_sed_vol;
double so_sweep_rowe;//sediment volume eroded in a row
double so_sweep_tote;//total sediment volume eroded before row
double so_sweep_rowd;//sediment volume deposited in a row
double so_sweep_totd;//total sediment volume deposited before row
double so_de_sum;//sum of de's
double so_de_sum_sq;//sum of squared de's
double so_standard_deviation;//standard deviation of de's
double so_bedr_sand;//bed roughness due to surface
double so_bedr_ripple;//bed roughness due to ripples
double so_bedr_total;//total bed roughness
double so_bedr_alpha;//ripple bed roughness proportional constant
double so_bed_shear_velocity;//bed shear velocity used to determine if hydraulically rough or smooth
double so_chezy_coeff;//chezy coefficient
double so_chezy_loss;//head loss due to bed roughness
double so_ripple_length;//representative ripple length
double so_ripple_height;//representative ripple height
double so_hydraulic_radius;//hydraulic radius
double so_ave_velocity = 0.0;//average velocity over grid (ignoring land points)
int so_sed_move;//if sediment movement in loop
double so_vector_velocity;//value of velocity vector
int so_sed_loop_no;//number of sediment movement loops
int so_rn1;//random number
int so_rn2;//random number
int so_tempd1;//temp int value
int so_tempd2;//temp int value
int so_pmax;//maximum value of so_all
double so_crit_del;//critical delta elevation
double so_bit;//used in calculations as interim value
double so_el1;//difference in elevation
double so_el2;//difference in elevation
double so_el3;//difference in elevation
double so_el4;//difference in elevation
int so_odd_even;//determine direction of velocity, even - left to right, odd - right to left
int so_oel_no;//odd and even number loop counter

/*Allocation of dynamic memory for arrays*/
if ((so_v = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_u = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_n = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_de = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_vol = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_histogram = new double[HAA][HBB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_energy = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_crit_vel = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((so_gtype = new int[AA][BB]) == NULL) {
```c
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((so_all = new int[(GSX*GSY)])) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((so_random = new int[(GSX*GSY)])) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
so_tempd = 0;
so_tempf = 0.0;
so_water_en_st = 0.0;
so_water_en_end = 0.0;
so_delta_en = 0.0;
so_rong3 = 0.0;
so_rong3 = 0.0;
so_delta_nede = 0.0;
so_delta_newater = 0.0;
so_delta_nedau = 0.0;
so_sed_pot = 0.0;
so_sed_vol = 0.0;
so_sweep_rose = 0.0;
so_sweep_tote = 0.0;
so_sweep_rowd = 0.0;
so_sweep_totd = 0.0;
so_de_sum = 0.0;
so_de_sum_sq = 0.0;
so_standard_deviation = 0.0;
so_bedr_sand = 0.0;
so_bedr_ripple = 0.0;
so_ripple_length = 0.0;
so_ripple_height = 0.0;
so_hydraulic_radius = 0.0;
so_ave_velocity = 0.0;
so_bedr_total = 0.0;
so_sed_move = 0;
so_vector_velocity = 0.0;
so_percentu = 0.0;
so_percentv = 0.0;
so_sed_unit = 0.0;
so_rn1 = 0;
so_rn2 = 0;
so_rn3 = 0.0;
so_tempd1 = 0;
so_tempd2 = 0;
so_pmax = 0;
so_q = 0;
so_crit_del = 0.0;
so_bit = 0.0;
so_el1 = 0.0;
so_el2 = 0.0;
so_el3 = 0.0;
so_el4 = 0.0;
so_sed_loop_no = 0;
so_odd_even = 0.0;
so_sed_unit = SED_UNIT;
for(so_j = 0; so_j < BB; so_j++){
  for(so_i = 0; so_i < AA; so_i++){
    so_v[so_i][so_j] = 0.0;
    so_u[so_i][so_j] = 0.0;
    so_n[so_i][so_j] = 0.0;
    so_de[so_i][so_j] = 0.0;
    so_vol[so_i][so_j] = 0.0;
    so_energy[so_i][so_j] = 0.0;
    so_crit_vel[so_i][so_j] = 0.0;
    so_type[so_i][so_j] = 0;
  }
}
for(so_j = 0; so_j < HBB; so_j++){
  for(so_i = 0; so_i < HAA; so_i++){
    so_histogram[so_i][so_j] = 0.0;
  }
}
for(so_j = 0; so_j < (GSX*GSY); so_j++)
  so_all[so_j][0] = 0;
  so_all[so_j][1] = 0;
```

so_all[so_j][2] = 0;
so_random[so_j] = 0;

} //change random number order
for(so_i=0; so_i < 85000; so_i++){
  so_rn1 = RandomInt(0,600);
}
so_crit_del = X_STEP*tan(ANGLE_REPOSE);
printf("\ncrit_el = %f", so_crit_del);
set_grid_size(1);
set_grid_type(1);
grid_size3f = fopen("g_size.txt", "r");
if( grid_size3f == NULL ){
  other_mistake(1);  //Can't open file
  printf("\ngrid_size\n");
}
else{
  fscanf(grid_size3f,"%d", &so_gsx);
  fscanf(grid_size3f,"%d", &so_gsy);
  fclose(grid_size3f);
}
grid_type3f = fopen("g_type.txt", "r");
if( grid_type3f == NULL ){
  other_mistake(1);  //Can't open file
  printf("\ngrid_type\n");
}
else{
  for(so_j=(so_gsy-1); so_j > (-1); so_j--){
    for(so_i=0; so_i < so_gsx; so_i++){
      fscanf(grid_type3f,"%d", &so_gtype[so_i][so_j]);
    }
  }
  fclose(grid_type3f);
}

/////////correct for constriction
for(so_j=0; so_j < so_gsy; so_j++){  //so_gsn
  for(so_i=0; so_i < so_gsx; so_i++){  //so_s
    if(so_gtype[so_i][so_j]==0){
      so_de[so_i][so_j] = 0;
    }
  }
}
so_i = 0;
so_j = 0;
so_k = 0;

while(1)://while sediment moving loop
  so_sed_move = 0;
  if(so_odd_even == 0.0){//if even loop
    ddepthf = fopen("deltadepth.dat", "w+");
    if( ddepthf == NULL ){
      other_mistake(1);  //Can't open file
    }
    else{
      for(so_i=0; so_i < so_gsx; so_i++){  //so_s
        for(so_j=0; so_j < so_gsy; so_j++){  //so_sn
          if(so_de[so_i][so_j]<0.0){
            fprintf(ddepthf,"%.3f ", so_de[so_i][so_j]);
          }
          else if((so_de[so_i][so_j]>=0.0) && (so_de[so_i][so_j]<10.0)){
            fprintf(ddepthf," %.3f ", so_de[so_i][so_j]);
          }
          else if(so_de[so_i][so_j]>=10.0){
            fprintf(ddepthf,"%.3f ", so_de[so_i][so_j]);
          }
          else{
            mistake();
            printf(" 10\n");
            fprintf (ddepthf, "\n");
            fclose(ddepthf);
          }
        }
      }
    }
  }
  spawnl(0,"hydra3jo.exe","help",NULL);
small_arcof = fopen("inlet.out", "r");
if( small_arcof == NULL ){
    other_mistake(1); //Can't open file
} else{
    fscanf(small_arcof, "%lf", &so_templf); //1020.398
    fscanf(small_arcof, "%d", &so_tempd); //12
    fscanf(small_arcof, "%lf", &so_templf); //0.1000000
    fscanf(small_arcof, "%s", &so_temps); //u
    for(so_i=0; so_i < so_gsx; so_i++){
        for(so_j=0; so_j < so_gsy; so_j++){
            fscanf(small_arcof, "%lf", &so_u[so_i][so_j]);
        }
    }
    fscanf(small_arcof, "%s", &so_temps); //v
    for(so_i=0; so_i < so_gsx; so_i++){
        for(so_j=0; so_j < so_gsy; so_j++){
            fscanf(small_arcof, "%lf", &so_v[so_i][so_j]);
        }
    }
    fscanf(small_arcof, "%s", &so_temps); //e
    for(so_i=0; so_i < so_gsx; so_i++){
        for(so_j=0; so_j < so_gsy; so_j++){
            fscanf(small_arcof, "%lf", &so_n[so_i][so_j]);
        }
    }
    fclose(small_arcof);
} //determine random order
so_p = 0;
for(so_i=0; so_i < (so_gsx-1); so_i++){
    for(so_j=1; so_j < (so_gsy-1); so_j++){
        so_all[so_p][0] = so_i;
        so_all[so_p][1] = so_j;
        so_random[so_p] = so_p;
        so_p = so_p + 1;
    }
} so_pmax = (so_gsx-1)*(so_gsy-2);
for(so_k=0; so_k < 5000; so_k++){
    so_rn1 = RandomInt(0, (so_pmax-1));
    so_rn2 = RandomInt(0, (so_pmax-1));
    so_templ = so_random[so_rn1];
    so_temps2 = so_random[so_rn2];
    so_random[so_rn1] = so_temps2;
    so_random[so_rn2] = so_templ;
} //move sediment
so_q = 0;
so_r = 0;
for(so_k=1; so_k < (so_gsx-1); so_k++){
    for(so_p=1; so_p < (so_gsy-1); so_p++){
        so_i = so_all[(so_random[so_q])][0];
        so_j = so_all[(so_random[so_q])][1];
        so_crit_vel[so_i][so_j] = 0.19* (pow(SAND_DIAMETER_50,0.1))*
        log10((4.0*(so_n[so_i][so_j]+STILL_DEPTH+so_de[so_i][so_j]))/SAND_DIAMETER_90);
        so_vector_velocity = pow((pow(so_u[so_i][so_j],2.0)+pow(so_v[so_i][so_j],2.0)),0.5);
        if(so_crit_vel[so_i][so_j]<=so_vector_velocity){//if critical velocity exceeded
            if(so_gtype[so_i][so_j]==0){//if land point
            } else if(so_gtype[so_i][so_j]!=0){//if water point
                so_percentu = pow(so_u[so_i][so_j],2.0)/pow(so_vector_velocity,2.0);
                so_percentv = pow(so_v[so_i][so_j],2.0)/pow(so_vector_velocity,2.0);
                if(so_u[so_i][so_j]>0.0){//flow in a positive direction
                    if(so_gtype[so_i+1][so_j]==0){//if next point is not a land point
                        so_de[so_i+1][so_j] = so_de[so_i][so_j] - so_sed_unit;
                    }
                }
            }
        }
    }
}
}
if(so_i==(so_gsx-2)) //if the next point is not a boundary
    so_de[so_i][so_j] = 0.0; //sediment is lost
else if(so_i==(so_gsx-2)) //if the next point is a boundary
    //sediment is lost
else if(so_v[so_i][so_j]==0) //if positive vertical velocity move sediment up
    if(so_gtype[so_i][so_j+1]==0) //if land point move directly forward
        if((so_j+1)!=0) //if sediment not moved out of the system over a boundary
            so_de[so_i][so_j+1] = so_de[so_i][so_j+1] - so_sed_unit*so_percentv;
        else
            so_de[so_i][so_j] = so_de[so_i][so_j] - so_sed_unit*so_percentv;
        //end sediment is lost
    else if(so_gtype[so_i][so_j]==0) //if negative vertical velocity move sediment down
        if((so_j-1)!=0) //if sediment not moved out of the system over a boundary
            so_de[so_i][so_j-1] = so_de[so_i][so_j-1] - so_sed_unit*so_percentv;
        else
            so_de[so_i][so_j] = so_de[so_i][so_j] - so_sed_unit*so_percentv;
        //end sediment is lost
    else if(u<0) //flow in a negative direction
        if(so_gtype[so_i][so_j]==0) //if the next point is not a boundary
            so_de[so_i][so_j] = so_de[so_i][so_j] - so_sed_unit;
if (so_v[so_i][so_j] >= 0) {// If positive vertical velocity, move sediment up
    if (so_gtype[so_i][so_j+1] == 0) {// If land point, move directly forward
        if (so_j != (so_gsy - 2)) {// If sediment not moved out of the system over a boundary
            so_de[so_i-1][so_j] = so_de[so_i-1][so_j] - so_sed_unit * so_percentv;
        } else if (so_j == (so_gsy - 2)) {// If sediment is lost over the boundary
            // Sediment is lost
        } else {// If next point is or isn't boundary mistake
            mistake();
        }
    } else if (no land point) {// If no land point
        so_de[so_i][so_j+1] = so_de[so_i][so_j+1] - so_sed_unit * so_percentv;
    } else {// If positive vertical velocity move sediment up
    }
} else if (so_v[so_i][so_j] < 0) {// If negative vertical velocity, move sediment down
    if (so_gtype[so_i][so_j-1] == 0) {// If land point, move directly forward
        if ((so_j-1) != 0) {// If sediment not moved out of the system over a boundary
            so_de[so_i-1][so_j] = so_de[so_i-1][so_j] - so_sed_unit * so_percentv;
        } else if ((so_j-1) == 0) {// If sediment is lost over the boundary
            // Sediment is lost
        } else {// If next point is or isn't boundary mistake
            mistake();
        }
    } else {// If no land point
        so_de[so_i][so_j-1] = so_de[so_i][so_j-1] - so_sed_unit * so_percentv;
    } else {// If negative vertical velocity move sediment down
    }
} else {// If vertical velocity mistake
    mistake();
}
so_de[so_i][so_j] = so_de[so_i][so_j] + so_sed_unit; // Remove sediment
so_sed_move = so_sed_move + 1;
}

if (u < 0) {// If u is negative
    mistake();
    // Flow in neither direction
    // Flow in neither direction
} else if (v < 0) {// If v is negative
    mistake();
    // Flow in neither direction
    // Flow in neither direction
} else if (critical velocity exceeded)
if (so_gtype[so_i][so_j] != 0) {
    so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
    so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
    so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
    so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
    if (so_el1 < 0) {
        if (((-1.0) * so_el1) > so_crit_del) {
            so_rn3 = RandomDouble(1.0, 1.5);
            so_bit = so_rn3 * (((-1.0) * so_el1) - so_crit_del) / 2.0;
            if (so_gtype[so_i][so_j+1] != 0) {// If not a land point
                so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
                so_de[(so_i+1)][so_j] = so_de[(so_i+1)][so_j] - so_bit;
                so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
                so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
                so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
                so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
                so_r = so_r + 1;
            }
        }
    }
    if (so_el2 < 0) {
        if (((-1.0) * so_el2) > so_crit_del) {
            so_rn3 = RandomDouble(1.0, 1.5);
            so_bit = so_rn3 * (((-1.0) * so_el2) - so_crit_del) / 2.0;
            if (so_gtype[so_i][so_j-1] != 0) {// If not a land point
                so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
                so_de[(so_i-1)][so_j] = so_de[(so_i-1)][so_j] - so_bit;
                so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
                so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
                so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
                so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
                so_r = so_r + 1;
            }
        }
    }
    if (so_el3 < 0) {
        if (((-1.0) * so_el3) > so_crit_del) {
            so_rn3 = RandomDouble(1.0, 1.5);
            so_bit = so_rn3 * (((-1.0) * so_el3) - so_crit_del) / 2.0;
            if (so_gtype[so_i][so_j] != 0) {// If not a land point
                so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
                so_de[(so_i-1)][so_j] = so_de[(so_i-1)][so_j] - so_bit;
                so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
                so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
                so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
                so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
                so_r = so_r + 1;
            }
        }
    }
    if (so_el4 < 0) {
        if (((-1.0) * so_el4) > so_crit_del) {
            so_rn3 = RandomDouble(1.0, 1.5);
            so_bit = so_rn3 * (((-1.0) * so_el4) - so_crit_del) / 2.0;
            if (so_gtype[so_i][so_j] != 0) {// If not a land point
                so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
                so_de[(so_i+1)][so_j] = so_de[(so_i+1)][so_j] - so_bit;
                so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
                so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
                so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
                so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
                so_r = so_r + 1;
            }
        }
    }
} else if (so_el2 < 0) {
    if (((-1.0) * so_el2) > so_crit_del) {
        so_rn3 = RandomDouble(1.0, 1.5);
        so_bit = so_rn3 * (((-1.0) * so_el2) - so_crit_del) / 2.0;
        if (so_gtype[so_i][so_j-1] != 0) {// If not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[(so_i-1)][so_j] = so_de[(so_i-1)][so_j] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
            so_r = so_r + 1;
        }
    }
} else if (so_el3 < 0) {
    if (((-1.0) * so_el3) > so_crit_del) {
        so_rn3 = RandomDouble(1.0, 1.5);
        so_bit = so_rn3 * ((-1.0) * so_el3) - so_crit_del) / 2.0;
        if (so_gtype[so_i][so_j] != 0) {// If not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[(so_i+1)][so_j] = so_de[(so_i+1)][so_j] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
            so_r = so_r + 1;
        }
    }
} else if (so_el4 < 0) {
    if (((-1.0) * so_el4) > so_crit_del) {
        so_rn3 = RandomDouble(1.0, 1.5);
        so_bit = so_rn3 * ((-1.0) * so_el4) - so_crit_del) / 2.0;
        if (so_gtype[so_i][so_j] != 0) {// If not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[(so_i+1)][so_j] = so_de[(so_i+1)][so_j] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][so_j-1];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][so_j+1];
            so_r = so_r + 1;
        }
    }
so_bit = so_rn3*(1.0-so_el3) - so_crit_del/2.0;
if(so_gtype[(so_i-1)][so_j] != 0) {//if not a land point
so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
so_de[(so_i-1)][so_j] = so_de[(so_i-1)][so_j] - so_bit;
so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
so_el4 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j+1];
so_r = so_r + 1;
}
if(so_el4<0){
if((-1.0)*so_el4)>so_crit_del){
so_rn3 = RandomDouble(1.0,1.5);
so_bit = so_rn3*((-1.0)*so_el4) - so_crit_del/2.0;
}
if(so_gtype[so_i][(so_j+1)] != 0) {//if not a land point
so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
so_de[so_i][(so_j+1)] = so_de[so_i][(so_j+1)] - so_bit;
so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
so_el4 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j+1];
so_r = so_r + 1;
}
}
so_q = so_q + 1;
//end for y
//end if even loop
else if(so_odd_even == 1.0) {//if odd loop
fdepthf = fopen("deltadepth.dat", "wt");
if( ddepthf == NULL ){
other_mistake(1); //Can't open file
}
else{
for(so_i=(so_gsx-1); so_i > (-1.0); so_i--){
for(so_j=0; so_j < so_gsy; so_j++){
if(so_de[so_i][so_j]<0.0){
fprintf(ddepthf, ".3f ", so_de[so_i][so_j]);
}
else if((so_de[so_i][so_j]>=0.0) && (so_de[so_i][so_j]<10.0)){
fprintf(ddepthf, " .3f ", so_de[so_i][so_j]);
}
else if(so_de[so_i][so_j]>=10.0){
fprintf(ddepthf, " .3f ", so_de[so_i][so_j]);
}
else{
mistake();
}
}
fprintf (ddepthf, 
);
fclose(ddepthf);
}
//call hydra.exe here
spawn(0,"hydra.exe","help",NULL);
small_arcof = fopen("inlet.out", "rt");
if( small_arcof == NULL ){
other_mistake(1); //Can't open file
}
else{
 fscanf(small_arcof, "%lf", &so_templf);
fscanf(small_arcof, "%lf", &so_templf);
fscanf(small_arcof, "%lf", &so_templf);
fscanf(small_arcof, "%lf", &so_templf);
for(so_i=(so_gsx - 1); so_i > (-1.0); so_i--){
for(so_j=0; so_j < so_gsy; so_j++){
if(so_gtype[so_i][so_j] != 0) {//if not a land point
so_u[so_i][so_j] = fscanf(small_arcof, "%lf", &so_u[so_i][so_j]);
}
}
for(so_i=(so_gsx - 1); so_i > (-1.0); so_i--){
for(so_j=0; so_j < so_gsy; so_j++){
so_v[so_i][so_j] = fscanf(small_arcof, "%lf", &so_v[so_i][so_j]);
}
}
for(so_i=(so_gsx - 1); so_i > (-1.0); so_i--){
for(so_j=0; so_j < so_gsy; so_j++){
so_e[so_i][so_j] = fscanf(small_arcof, "%lf", &so_e[so_i][so_j]);
}
}
}
for(so_j=0; so_j < so_gsy; so_j++){
    fscanf(small_arcof, "%lf", &so_n[so_i][so_j]);
}
fclose(small_arcof);

////////////////////////////////////////////////////////////////////////////////
//determine random order
////////////////////////////////////////////////////////////////////////////////
so_p = 0;
for(so_i=0; so_i < (so_gsx-1); so_i++){
    for(so_j=1; so_j < (so_gsy-1); so_j++){
        so_all[so_p][0] = so_i;
        so_all[so_p][1] = so_j;
        so_random[so_p] = so_p;
        so_p = so_p + 1;
    }
}
so_max = (so_gsx-1)*(so_gsy-2);
for(so_k=0; so_k < 5000; so_k++){
    so_random[so_k1] = RandomInt(0,(so_pmax-1));
    so_random[so_k2] = RandomInt(0,(so_pmax-1));
    so_tempd1 = so_random[so_k1];
    so_tempd2 = so_random[so_k2];
    so_random[so_k1] = so_tempd2;
    so_random[so_k2] = so_tempd1;
}
for(so_k=0; so_k < so_pmax; so_k++){
    //printf("%d %d", so_k, so_random[so_k]);
}
////////////////////////////////////////////////////////////////////////////////
//move sediment
////////////////////////////////////////////////////////////////////////////////
so_q = 0;
so_r = 0;
for(so_k=1; so_k < (so_gsx-1); so_k++){
    for(so_p=1; so_p < (so_gsy-1); so_p++){
        so_i = so_all[(so_random[so_q])][0];
        so_j = so_all[(so_random[so_q])][1];
        so_crit_vel[so_i][so_j] = 0.19* (pow(SAND_DIAMETER_50,0.1))*
            log10((4.0*(so_n[so_i][so_j]+STILL_DEPTH+so_de[so_i][so_j]))/SAND_DIAMETER_90);
        so_vector_velocity = pow((so_u[so_i][so_j] + so_v[so_i][so_j]),2.0);  
        if(so_crit_vel[so_i][so_j]<=so_vector_velocity){//if critical velocity exceeded
            if(so_gtype[so_i][so_j]==0){//if land point
                //sediment is lost
            }else if(so_gtype[so_i][so_j]!=0){//if water point
                so_percentu = pow((so_u[so_i][so_j]),2.0)/pow((so_u[so_i][so_j]+so_v[so_i][so_j]),2.0);  
                if(iso_v[so_i][so_j] > 0.0){//if positive vertical velocity move sediment up
                    if(so_gtype[so_i-1][so_j]==0){//if land point move directly backward
                        so_de[so_i-1][so_j] = so_de[so_i-1][so_j] - so_sed_unit;
                    }else{//if no land point
                        so_de[so_i-1][so_j+1] = so_de[so_i-1][so_j+1] - so_sed_unit*so_percentu;
                    }
                }else if(iso_v[so_i][so_j] < 0.0){//if negative vertical velocity move sediment down
                    if(so_gtype[so_i+1][so_j]==0){//if land point move directly backward
                        so_de[so_i+1][so_j] = so_de[so_i+1][so_j] - so_sed_unit*so_percentu;
                    }else{//if no land point
                        so_de[so_i+1][so_j-1] = so_de[so_i+1][so_j-1] - so_sed_unit*so_percentu;
                    }
                }else{//if no vertical component
                    if(so_i==1){//if the next point is a boundary
                        //sediment is lost
                    }else{//if the next point is or isn't boundary mistake
                        mistake();
                    }
                }
            }else{//if some vertical component
                if(iso_v[so_i][so_j] > 0.0){//if positive vertical velocity move sediment up
                    if(iso_v[so_i][so_j] == 0.0){//if land point move directly backward
                        so_de[so_i][so_j] = so_de[so_i][so_j] - so_sed_unit*so_per"
if(so_gtype[(so_i-1)][(so_j-1)]==0){//if land point move directly forward
    so_de[(so_i-1)][so_j] = so_de[(so_i-1)][so_j] - so_sed_unit*so_percentv;
} //end if land point move directly forward
else{//end if no land point
    so_de[(so_i-1)][(so_j-1)] = so_de[(so_i-1)][(so_j-1)] - so_sed_unit*so_percentv;
} //end if no land point
} ////end if negative vertical velocity move sediment down
else{//if vertical velocity mistake
    mistake();
} ////end if vertical velocity mistake
} //end if some vertical component
so_de[so_i][so_j] = so_de[so_i][so_j] + so_sed_unit;//remove sediment
so_sed_move = so_sed_move + 1;
} //end if next point is not a land point
else{//if water point
    mistake();
} //end if mistake(neither land nor water point
} //end if critical velocity exceeded
if(so_gtype != 0){
    so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
    so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
    so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
    so_el4 = so_de[so_i][so_j] - so_de[so_i][(so_j+1)];
    if([(-1.0)*so_el1]>so_crit_del){
        so_bit = (((-1.0)*so_el1) - so_crit_del)/2.0;
        if(so_gtype[(so_i+1)][so_j] != 0){//if not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[(so_i+1)][so_j] = so_de[(so_i+1)][so_j] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][(so_j+1)];
            so_r = so_r + 1;
        }
    }
    if([(-1.0)*so_el2]>so_crit_del){
        so_bit = (((-1.0)*so_el2) - so_crit_del)/2.0;
        if(so_gtype[so_i][(so_j-1)] != 0){//if not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[so_i][(so_j-1)] = so_de[so_i][(so_j-1)] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][(so_j+1)];
            so_r = so_r + 1;
        }
    }
    if([(-1.0)*so_el3]>so_crit_del){
        so_bit = (((-1.0)*so_el3) - so_crit_del)/2.0;
        if(so_gtype[(so_i-1)][so_j] != 0){//if not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[(so_i-1)][so_j] = so_de[(so_i-1)][so_j] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][(so_j+1)];
            so_r = so_r + 1;
        }
    }
    if([(-1.0)*so_el4]>so_crit_del){
        so_bit = (((-1.0)*so_el4) - so_crit_del)/2.0;
        if(so_gtype[so_i][(so_j+1)] != 0){//if not a land point
            so_de[so_i][so_j] = so_de[so_i][so_j] + so_bit;
            so_de[so_i][(so_j+1)] = so_de[so_i][(so_j+1)] - so_bit;
            so_el1 = so_de[so_i][so_j] - so_de[(so_i+1)][so_j];
            so_el2 = so_de[so_i][so_j] - so_de[so_i][(so_j-1)];
            so_el3 = so_de[so_i][so_j] - so_de[(so_i-1)][so_j];
            so_el4 = so_de[so_i][so_j] - so_de[so_i][(so_j+1)];
            so_r = so_r + 1;
        }
    }
}
so_q = so_q + 1;
} //end for y
} //end for x
} //end if odd loop
else{//if neither odd or even loop so mistake

mistake();
}  //end if neither odd or even loop so mistake
if( so_sed_loop_no < 1 ){ //print info and create file if first loop
  sed_movef = fopen( "sedmov.txt", "w+");
  if( sed_movef == NULL ){ other_mistake(1);  //Can't open file }
  else{
    fprintf(sed_movef, "%s ", "sed_loop_no");
    fprintf(sed_movef, "%s ", "sed_moved");
    fprintf(sed_movef, "angles_moved");
    fprintf(sed_movef, "\n");
    fprintf(sed_movef, "%d ", (so_sed_loop_no + 1));
    fprintf(sed_movef, "%d ", so_sed_move);
    fprintf(sed_movef, "%d ", so_r);
    fprintf(sed_movef, "\n");
    fclose(sed_movef);
  }
  bestde2f = fopen( "collde.txt", "w+");
  if( bestde2f == NULL ){ other_mistake(1);  //Can't open file }
  else{
    fprintf(bestde2f, "%s", "sed_loop_no_");
    fprintf(bestde2f, "%d ", (so_sed_loop_no + 1));
    fprintf(bestde2f, "\n");
    for(so_j=(so_gsy-1); so_j > (-1); so_j--){
      for(so_i=0; so_i < so_gsx; so_i++){
        fprintf(bestde2f, "%e ", so_de[so_i][so_j]);
      }
      fprintf(bestde2f, "$\n$" );
    }
    fprintf(bestde2f, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(bestde2f, "%e ", so_u[so_i][so_j]);
    }
    fprintf(bestde2f, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(bestde2f, "%e ", so_v[so_i][so_j]);
    }
    fprintf(bestde2f, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(bestde2f, "%e ", so_n[so_i][so_j]);
    }
    fprintf(bestde2f, "$\n$" );
    fclose(bestde2f);
  }
}
} //end print info and create file if first loop
else if( so_sed_loop_no >= 1 ){ //print info file
  sed_movef = fopen( "sedmov.txt", "a+");
  if( sed_movef == NULL ){ other_mistake(1);  //Can't open file }
  else{
    fprintf(sed_movef, "%d ", (so_sed_loop_no + 1));
    fprintf(sed_movef, "%d ", so_sed_move);
    fprintf(sed_movef, "%d ", so_r);
    fprintf(sed_movef, "\n");
    fclose(sed_movef);
  }
  bestde2f = fopen( "collde.txt", "a+");
  if( bestde2f == NULL ){ other_mistake(1);  //Can't open file }
  else{
    fprintf(bestde2f, "%s", "sed_loop_no_");
    fprintf(bestde2f, "%d ", (so_sed_loop_no + 1));
    fprintf(bestde2f, "\n");
    for(so_j=(so_gsy-1); so_j > (-1); so_j--){
      for(so_i=0; so_i < so_gsx; so_i++){
        fprintf(bestde2f, "%e ", so_de[so_i][so_j]);
      }
      fprintf(bestde2f, "$\n$" );
    }
    fprintf(bestde2f, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(bestde2f, "%e ", so_u[so_i][so_j]);
    }
    fprintf(bestde2f, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(bestde2f, "%e ", so_v[so_i][so_j]);
    }
    fprintf(bestde2f, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(bestde2f, "%e ", so_n[so_i][so_j]);
    }
    fprintf(bestde2f, "$\n$" );
    fclose(bestde2f);
  }
} //end print info file if first loop
else if( so_sed_loop_no >> 1 ){ //print info file
  sed_movef = fopen( "sedmov.txt", "a+");
  if( sed_movef == NULL ){ other_mistake(1);  //Can't open file }
  else{
    fprintf(sed_movef, "%s", "sed_loop_no_");
    fprintf(sed_movef, "%d ", (so_sed_loop_no + 1));
    fprintf(sed_movef, "\n");
    for(so_j=(so_gsy-1); so_j > (-1); so_j--){
      for(so_i=0; so_i < so_gsx; so_i++){
        fprintf(sed_movef, "%e ", so_de[so_i][so_j]);
      }
      fprintf(sed_movef, "$\n$" );
    }
    fprintf(sed_movef, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(sed_movef, "%e ", so_u[so_i][so_j]);
    }
    fprintf(sed_movef, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(sed_movef, "%e ", so_v[so_i][so_j]);
    }
    fprintf(sed_movef, "$\n$" );
    for(so_i=0; so_i < so_gsx; so_i++){
      fprintf(sed_movef, "%e ", so_n[so_i][so_j]);
    }
    fprintf(sed_movef, "$\n$" );
    fclose(sed_movef);
  }
} //end if neither odd or even loop so mistake
else {//mistake don’t print info file
    mistake();
} //end mistake don’t print info file

if(so_sed_move < 1) {//check if no sediment movement
    break; //end as reached a self-organised situation
} //end check if no sediment movement

so_sed_loop_no = so_sed_loop_no + 1;
so_oel_no = so_oel_no + 1.0;

} //end while sediment moving loop

//write the best configuration to file
bestvf = fopen(“bestv.txt”, “w+”);
bestuf = fopen(“bestu.txt”, “w+”);
bestnf = fopen(“bestn.txt”, “w+”);
bestdef = fopen(“bestde.txt”, “w+”);
if( bestvf == NULL ) {
    other_mistake(1); //Can’t open file
} else {
    for(so_j=(so_gsy-1); so_j > (-1); so_j--){
        for(so_i=0; so_i < so_gsx; so_i++){
            fprintf(bestvf, “%e “, so_v[so_i][so_j]);
            fprintf(bestuf, “%e “, so_u[so_i][so_j]);
            fprintf(bestnf, “%e “, so_n[so_i][so_j]);
            fprintf(bestdef, “%e “, so_de[so_i][so_j]);
        }
        fprintf(bestvf, “
”);
        fprintf(bestuf, “
”);
        fprintf(bestnf, “
”);
        fprintf(bestdef, “
”);
    }
    fclose(bestvf);
    fclose(bestuf);
    fclose(bestnf);
    fclose(bestdef);
} //end write the best configuration to file

/*Deletion of dynamic memory for arrays*/
delete [] so_v;
delete [] so_u;
delete [] so_n;
delete [] so_de;
delete [] so_vol;
delete [] so_histogram;
delete [] so_energy;
delete [] so_crit_vel;
delete [] so_gtype;
delete [] so_all;
delete [] so_random;
}

/**************************************************************************
***************************Mistake Function*******************************
***************************************************************************/

void mistake(void){
    printf(“
There has been a mistake”);
    return;
}

/********************************************************************************
**************************Set Grid Size Function******************************
*********************************************************************************/

void other_mistake(int om_no){
    char om_words[100];

    if(om_no==1){
        (void)strcpy(om_words,”Can’t open file “);
    } else if(om_no==6){
        (void)strcpy(om_words,”You entered an incorrect choice, please try again “);
    } else{
        mistake();
        printf(“
Wrong choice “, om_words);
        return;
    }
}

/**************************************************************************
**************************Set Grid Size Function******************************
*********************************************************************************/
void set_grid_size(int sgs_check1){
    char sgs_temps[100];
    int sgs_gsx;//number of points in x direction
    int sgs_gsy;//number of points in y direction
    
    sgs_gsx = GSX;//default number of points in x direction
    sgs_gsy = GSY;//default number of points in y direction
    
    small_arcef = fopen ("inlet.evt", "r");
    if( small_arcef == NULL ){  
        other_mistake(1);  //Can't open file
    }
    else{
        fscanf(small_arcef, "%s", &sgs_temps);
        fscanf(small_arcef, "%s", &sgs_temps);
        fscanf(small_arcef, "%s", &sgs_temps);
        fscanf(small_arcef, "%d", &sgs_gsx);
        fscanf(small_arcef, "%d", &sgs_gsy);
        fclose(small_arcef);
    }
    
    grid_sizef = fopen ("g_size.txt", "w+");
    if( grid_sizef == NULL ){  
        other_mistake(1);  //Can't open file
    }
    else{
        fprintf(grid_sizef, "%d", sgs_gsx);
        fprintf(grid_sizef, 
        "%d", sgs_gsy);
        fprintf(grid_sizef, 
        "%s", "x_value");
        fprintf(grid_sizef, 
        "%s", "y_value");
        fclose(grid_sizef);
    }
    return;
}

/**************************************************************************
**************************Set Grid Type Function**************************
**************************************************************************/

void set_grid_type(int sgt_check1){
    int sgt_check4;
    char sgt_line[500];
    char sgt_temps[500];
    int sgt_tempd;
    double sgt_templf;
    int sgt_gsx;//number of points in x direction
    int sgt_gsy;//number of points in y direction
    int sgt_i;
    int sgt_j;
    int sgt_k;
    int (*sgt_type)[BB];//type of point
    double (*sgt_v)[BB];//actual v values
    double (*sgt_u)[BB];//actual u values
    double (*sgt_n)[BB];//actual n values
    int *sgt_bits;//the type of each of the optimising values
    int sgt_countv;//count of v variables
    int sgt_countu;//count of u variables
    int sgt_countn;//count of n variables
    int sgt_count_tot;//count of n variables
    
    sgt_tempd = 0;
    sgt_templf = 0.0;
    
    /*Allocation of dynamic memory for arrays*/
    if ((sgt_type = new int[AA][BB]) == NULL) {  
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_v = new double[AA][BB]) == NULL) {  
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_u = new double[AA][BB]) == NULL) {  
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_n = new double[AA][BB]) == NULL) {  
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_bits = new int[a]) == NULL) {  
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
printf("Memory Allocation Failure for Array ar1\n");
exit(0);
}
}
for(sgt_i=0; sgt_i < AA; sgt_i++){
for(sgt_j=0; sgt_j < BB; sgt_j++){
 sgt_type[sgt_i][sgt_j] = 0;
 sgt_v[sgt_i][sgt_j] = 0.0;
 sgt_u[sgt_i][sgt_j] = 0.0;
 sgt_n[sgt_i][sgt_j] = 0.0;
}
}
for(sgt_i=0; sgt_i < a; sgt_i++){
 sgt_bits[sgt_i] = 0;
}
sgt_countv = 0;
sgt_countu = 0;
sgt_countn = 0;
sgt_count_tot = 0;
sgt_check4 = 1;
printf("\nbefore open grid size in function");
grid_size2f = fopen ( "g_size.txt", "r");
if( grid_size2f == NULL ){
 other_mistake(1);  //Can't open file
}
else{
 fscanf(grid_size2f, "%d", &sgt_gsx);//reads in x dimension
 fscanf(grid_size2f, "%d", &sgt_gsy);//reads in y dimension
 fclose(grid_size2f);
}
small_arce2f = fopen ( "inlet.evt", "r");
if( small_arce2f == NULL ){
 other_mistake(1);  //Can't open file
}
else{
 fscanf(small_arce2f, "%s", &sgt_temps);//Small
 fscanf(small_arce2f, "%s", &sgt_temps);//ARC
 fscanf(small_arce2f, "%s", &sgt_temps);//Channel
 fscanf(small_arce2f, "%d", &sgt_tempd);//38 - no of x points
 fscanf(small_arce2f, "%d", &sgt_tempd);//12 - no of y points
 fscanf(small_arce2f, "%d", &sgt_tempd);//1
 fscanf(small_arce2f, "%d", &sgt_tempd);//21
 fscanf(small_arce2f, "%d", &sgt_tempd);//6
 fscanf(small_arce2f, "%lf", &sgt_templf);//0.10
 fscanf(small_arce2f, "%lf", &sgt_templf);//0.10
 fscanf(small_arce2f, "%lf", &sgt_templf);//1000.0
 fscanf(small_arce2f, "%lf", &sgt_templf);//20.0
 fscanf(small_arce2f, "%d", &sgt_tempd);///1
 fscanf(small_arce2f, "%lf", &sgt_templf);//0.01
 fscanf(small_arce2f, "%lf", &sgt_templf);//0.01
 fscanf(small_arce2f, "%lf", &sgt_templf);//0.0
 fscanf(small_arce2f, "%lf", &sgt_templf);///21
 fscanf(small_arce2f, "%d", &sgt_tempd);///1
 fscanf(small_arce2f, "%d", &sgt_tempd);///38
 fscanf(small_arce2f, "%d", &sgt_tempd);///12
}
for(sgt_j=(sgt_gsy-1); sgt_j > (-1); sgt_j--){
 fscanf(small_arce2f, "%s", &sgt_temps);
 for(sgt_i=0; sgt_i < sgt_gsx; sgt_i++){
 if(sgt_temps[sgt_i] == '0'){
 sgt_tempd = 0;
 }
 else if(sgt_temps[sgt_i] == '1'){
 sgt_tempd = 1;
 }
 else if(sgt_temps[sgt_i] == '2'){
 sgt_tempd = 2;
 }
 else if(sgt_temps[sgt_i] == '4'){
 sgt_tempd = 4;
 }
 else if(sgt_temps[sgt_i] == '6'){
 sgt_tempd = 6;
 }
}
else if(sgt_temps[sgt_i] == '8') {
    sgt_tempd = 8;
} else {
    mistake();
    printf(" 9");
}
sgt_type[sgt_i][sgt_j] = sgt_tempd;
}
fclose(small_arce2f);
}

grid_typef = fopen ( "g_type.txt", "w+");
if( grid_typef == NULL ){
    other_mistake(1);  //Can't open file
} else{
    for(sgt_j=(sgt_gsy-1); sgt_j > (-1); sgt_j--){
        for(sgt_i=0; sgt_i < sgt_gsx; sgt_i++){
            fprintf(grid_typef, "%d ", sgt_type[sgt_i][sgt_j]);
        }
        fprintf(grid_typef, "\\n");
    }
    fprintf(grid_typef, "\land_0\n");
    fprintf(grid_typef, "\normal\water\n");
    fprintf(grid_typef, "\u=0\v=0\n");
    fprintf(grid_typef, "\specified\n\specified\u\specified\v\n");
    fprintf(grid_typef, "\specified\n\specified\v\n");
    fprintf(grid_typef, "\combination\sum\n");
}
fclose(grid_typef);
}

sgt_i = 0;
sgt_j = (sgt_gsy - 1);
sgt_k = 0;
while(1){
    if(sgt_type[sgt_i][sgt_j] == 0){
        sgt_v[sgt_i][sgt_j] = 0.0;
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_n[sgt_i][sgt_j] = 0.0;
    } else if(sgt_type[sgt_i][sgt_j] == 1){
        sgt_v[sgt_i][sgt_j] = 2.0;
        sgt_countv = sgt_countv + 1;
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 1;
        sgt_k = sgt_k + 1;
    } else if(sgt_type[sgt_i][sgt_j] == 2){
        sgt_v[sgt_i][sgt_j] = 2.0;
        sgt_countv = sgt_countv + 1;
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 2;
        sgt_k = sgt_k + 1;
    } else if(sgt_type[sgt_i][sgt_j] == 4){
        sgt_v[sgt_i][sgt_j] = 0.0;
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 4;
        sgt_k = sgt_k + 1;
    } else if(sgt_type[sgt_i][sgt_j] == 6){
        sgt_v[sgt_i][sgt_j] = 0.0;
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 6;
        sgt_k = sgt_k + 1;
    }
}
else if(sgt_type[sgt_i][sgt_j] == 8){
    sgt_v[sgt_i][sgt_j] = 2.0;
    sgt_countv = sgt_countv + 1;
    sgt_u[sgt_i][sgt_j] = 2.0;
    sgt_countu = sgt_countu + 1;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 8;
    sgt_k = sgt_k + 1;
}
else if(sgt_type[sgt_i][sgt_j] == 10){
    sgt_v[sgt_i][sgt_j] = 2.0;
    sgt_countv = sgt_countv + 1;
    sgt_u[sgt_i][sgt_j] = 0.0;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 10;
    sgt_k = sgt_k + 1;
}
else if(sgt_type[sgt_i][sgt_j] == 12){
    sgt_v[sgt_i][sgt_j] = 0.0;
    sgt_u[sgt_i][sgt_j] = 2.0;
    sgt_countu = sgt_countu + 1;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 12;
    sgt_k = sgt_k + 1;
}
else if(sgt_type[sgt_i][sgt_j] == 14){
    sgt_v[sgt_i][sgt_j] = 0.0;
    sgt_u[sgt_i][sgt_j] = 0.0;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_n[sgt_i][sgt_j] = 2.0;
    sgt_bits[sgt_k] = 16;
    sgt_k = sgt_k + 1;
}
else if(sgt_type[sgt_i][sgt_j] == 16){
    sgt_v[sgt_i][sgt_j] = 2.0;
    sgt_countv = sgt_countv + 1;
    if(sgt_check4 == 1){
        sgt_u[sgt_i][sgt_j] = SPECU;
    } else{
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
    }
    sgt_n[sgt_i][sgt_j] = 2.0;
    sgt_countn = sgt_countn + 1;
    sgt_bits[sgt_k] = 16;
    sgt_k = sgt_k + 1;
}
else if(sgt_type[sgt_i][sgt_j] == 20){
    sgt_v[sgt_i][sgt_j] = 0.0;
    if(sgt_check4 == 1){
        sgt_u[sgt_i][sgt_j] = SPECU;
    } else{
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
    }
    sgt_n[sgt_i][sgt_j] = 2.0;
else if(sgt_type[sgt_i][sgt_j] == 42){
    if(sgt_check4 == 1){
        sgt_v[sgt_i][sgt_j] = SPECV;
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter v for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_u[sgt_i][sgt_j] = 0.0;
} else if(sgt_type[sgt_i][sgt_j] == 48){
    if(sgt_check4 == 1){
        sgt_v[sgt_i][sgt_j] = SPECV;
        sgt_u[sgt_i][sgt_j] = SPECU;
    } else{
        printf("Enter v for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 48;
        sgt_k = sgt_k + 1;
    }
} else if(sgt_type[sgt_i][sgt_j] == 56){
    if(sgt_check4 == 1){
        sgt_v[sgt_i][sgt_j] = SPECV;
        sgt_u[sgt_i][sgt_j] = SPECU;
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter v for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
} else{
    mistake();
    printf("8");
}

sgt_i = sgt_i + 1;
if(sgt_i == sgt_gsx){
    sgt_i = 0;
    sgt_j = sgt_j - 1;
} if(sgt_j == 0){
    break;
}

grid_vf = fopen("g_v.txt", "w");
grid_uf = fopen("g_u.txt", "w");
grid_nf = fopen("g_n.txt", "w");
if(grid_vf == NULL){
    other_mistake(); //Can't open file
    printf("\ngrid_vf");
} else{
    for(sgt_j=(sgt_gsy-1); sgt_j > (-1); sgt_j--){
        for(sgt_i=0; sgt_i < sgt_gsx; sgt_i++){
            fprintf(grid_vf, "%f ", sgt_v[sgt_i][sgt_j]);
            fprintf(grid_uf, "%f ", sgt_u[sgt_i][sgt_j]);
            fprintf(grid_nf, "%f ", sgt_n[sgt_i][sgt_j]);
        }
        fprintf(grid_vf, "\n");
        fprintf(grid_uf, "\n");
        fprintf(grid_nf, "\n");
    }
}
fprintf(grid_vf, "\n%s", "velocity_v");
fprintf(grid uf, "\n%s", "velocity_u");
fprintf(grid nf, "\n%s", "water_elevation_n");
fclose(grid_vf);
fclose(grid uf);
fclose(grid nf);
}

grid_bitsf = fopen("grid_bits.txt", "w+");
if( grid_bitsf == NULL ){
  other_mistake(1);  //Can't open file
  fprintf("ingrid_bits");
} else{
  fprintf(grid_bitsf, "\d ", sgt_k);
  for(sgt_i=0; sgt_i < sgt_k; sgt_i++){
    fprintf(grid_bitsf, "\d ", sgt_bits[sgt_i]);
  }
  fprintf(grid_vf, "\n%s", "optimising_bits");
  fclose(grid_bitsf);
}

sgt_count_tot = sgt_countv + sgt_countu + sgt_countn;
water_countf = fopen("w_count.txt", "w+");
if( water_countf == NULL ){
  other_mistake(1);  //Can't open file
} else{
  fprintf(water_countf, "%d ", sgt_countn);//writes number of points to optimise
  fprintf(water_countf, "%d ", sgt_countv);//writes number of points to optimise
  fprintf(water_countf, "%d ", sgt_countu);//writes number of points to optimise
  fprintf(water_countf, "%d ", sgt_countn);//writes number of points to optimise
  fprintf(water_countf, "\n%s", "count_of_optimising_points");
  fprintf(water_countf, "\n%s", "count_of_v_optimising_points");
  fprintf(water_countf, "\n%s", "count_of_u_optimising_points");
  fprintf(water_countf, "\n%s", "count_of_n_optimising_points");
  fclose(water_countf);
}
/*Deletion of dynamic memory for arrays*/
delete [] sgt_type;
delete [] sgt_v;
delete [] sgt_u;
delete [] sgt_n;
delete [] sgt_bits;
return;
}

/*******************************************************************************
***************Random Number Generation Functions***************************
*******************************************************************************/

/* Globals */
double u[97];
double c;
double cd;
double cm;
int i97,j97;
int test = FALSE;

/*This is the initialisation routine for the random number generator. It*
*can generate 900 million different subsequences
*/
void RandomInitialise(int ij,int kl){
  double s,t;
  int ii,i,j,k,l,jj,m;

  /*This random number generator comes from a version published by George *
*Marsaglia and Arif Zaman, Florida State University and modified by the *
*Dept. of Computer Science at Fachhochschule Wiesbaden, Germany. It is *
*used as the Rand() function available in C/C++ is only pseudo random, *
*and this version has been found to be the best random number generator *
*known. It has a period of 2^144.                                    */

  /*Globals*/
  double u[97];
  double c;
  double cd;
  double cm;
  int i97,j97;
  int test = FALSE;

  /*This is the initialisation routine for the random number generator. It*/
  /*can generate 900 million different subsequences
  */
```c
/* Handle the seed range errors                        */
/* First random number seed must be between 0 and 31328*
* Second seed must have a value between 0 and 30081   */
if (ij < 0 || ij > 31328 || kl < 0 || kl > 30081) {
    ij = 1802;
    kl = 9373;
}
i = (ij / 177) % 177 + 2;
j = (ij % 177) + 2;
k = (kl / 169) % 178 + 1;
l = (kl % 169);
for (ii=0; ii<97; ii++) {
    s = 0.0;
    t = 0.5;
    for (jj=0; jj<24; jj++) {
        n = (((i * j) % 179) * k) % 179;
        i = j;
        j = k;
        k = m;
        l = (53 * l + 1) % 169;
        if (((l * m % 64)) >= 32)
            n *= t;
        t *= 0.5;
    }
    u[ii] = s;
}
c    = 362436.0 / 16777216.0;
cd   = 7654321.0 / 16777216.0;
cm   = 16777213.0 / 16777216.0;
i97  = 97;
j97  = 33;
test = TRUE;
}
/* This is the random number generator proposed by George Marsaglia*/
double RandomUniform(void){
    double uni;
    /* Make sure the initialisation routine has been called */
    if (!test)
        RandomInitialise(1802, time(NULL));
        RandomInitialise(1802, 9373);
    uni = u[i97-1] - u[j97-1];
    if (uni <= 0.0)
        uni++;
    u[i97-1] = uni;
    i97--;
    if (i97 == 0)
        i97 = 97;
    j97--;
    if (j97 == 0)
        j97 = 97;
    c -= cd;
    if (c < 0.0)
        c = cm;
    uni -= c;
    if (uni < 0.0)
        uni++;
    return(uni);
}
/* Algorithm from: Transactions on Mathematical Software,*
* Vol. 18, No. 4, (1992), pp. 434-435.                  */
double RandomGaussian(double mean, double stddev){
    double  q,u,v,x,y;
    /* Generate P = (u,v) uniform in rect. enclosing acceptance region */
    /* Make sure that any random numbers <= 0 are rejected, since *
    * gaussian() requires uniforms > 0, but RandomUniform() delivers <= 0. */
    do {
        u = RandomUniform();
        v = RandomUniform();
        if (u <= 0.0 || v <= 0.0) {
            u = 1.0;
            v = 1.0;
        }
        x = 2.0 * u - 1.0;
        y = 2.0 * v - 1.0;
        q = x * x + y * y;
        if (q >= 1.0) {
            x = x * sqrt(-2.0 * log(q) / q);
            y = y * sqrt(-2.0 * log(q) / q);
            if (x <= 0.0) {
                u = 1.0;
                v = 1.0;
            }
        }
    } while (true);
    /* Acceptance probability: */
    return(mean + stddev * x);
}
\begin{verbatim}
  v = 1.7156 * (v - 0.5);
  /*Evaluate the quadratic form*/
  x = u - 0.449871;
  y = fabs(v) + 0.386595;
  q = x * x + y * (0.19600 * y - 0.25472 * x);
  /*Accept P if inside inner ellipse*/
  if (q < 0.27597)
    break;
  /*Reject P if outside outer ellipse, or outside acceptance region*/
} while ((q > 0.27846) || (v * v > -4.0 * log(u) * u * u));
/*Return ratio of P's coordinates as the normal deviate*/
return (mean + stddev * v / u);
/*Return random integer within a range, lower <> upper INCLUSIVE*/
int RandomInt(int lower, int upper)
{
  // printf("%d", (int)(RandomUniform() * (upper - lower + 1)) + lower);
  return((int)(RandomUniform() * (upper - lower + 1)) + lower);
}
/*Return random float within a range, lower <> upper*/
double RandomDouble(double lower, double upper)
{
  return((upper - lower) * RandomUniform() + lower);
}
/*End of Program*/
\end{verbatim}
Appendix B – Comparison of Optimisation Methods

This Appendix contains a sample of selective results from sensitivity analyses of optimisation of the solution of flow patterns around a plate, using a binary genetic algorithm (BGA), real genetic algorithm (RGA), simulated annealing (SA) or a combination. These are discussed in Chapter Four, Section 4.2.

Table B.1 Sample of some GA sensitivity analyses.

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Type of Optimisation</th>
<th>Population</th>
<th>Generation</th>
<th>Crossover Type</th>
<th>Crossover Probability</th>
<th>Mutation Type</th>
<th>Mutation Probability</th>
<th>Best Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 6</td>
<td>RGA</td>
<td>200</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1983</td>
</tr>
<tr>
<td>6 6</td>
<td>RGA</td>
<td>200</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1984</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA</td>
<td>200</td>
<td>3324</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>3310</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA</td>
<td>300</td>
<td>720</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.1</td>
<td>571</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA</td>
<td>400</td>
<td>700</td>
<td>average</td>
<td>0.9</td>
<td>random</td>
<td>0.2</td>
<td>15</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA</td>
<td>400</td>
<td>10000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.05</td>
<td>7864</td>
</tr>
<tr>
<td>18 6</td>
<td>BGA</td>
<td>400</td>
<td>3000</td>
<td>uniform</td>
<td>0.8</td>
<td>uniform</td>
<td>0.01</td>
<td>698</td>
</tr>
<tr>
<td>18 6</td>
<td>BGA</td>
<td>400</td>
<td>3000</td>
<td>uniform</td>
<td>0.7</td>
<td>uniform</td>
<td>0.005</td>
<td>1698</td>
</tr>
<tr>
<td>18 6</td>
<td>BGA</td>
<td>400</td>
<td>10000</td>
<td>uniform</td>
<td>0.7</td>
<td>uniform</td>
<td>0.005</td>
<td>9422</td>
</tr>
<tr>
<td>18 6</td>
<td>BGA</td>
<td>400</td>
<td>3608</td>
<td>uniform</td>
<td>0.7</td>
<td>uniform</td>
<td>0.001</td>
<td>2710</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>20000</td>
<td>uniform</td>
<td>0.7</td>
<td>uniform</td>
<td>0.001</td>
<td>15915</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1872</td>
</tr>
<tr>
<td>6 6</td>
<td>RGA</td>
<td>200</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>2000</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>300</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>300</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>600</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>600</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1872</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1998</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1872</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1698</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1872</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>neigh</td>
<td>0.01</td>
<td>2000</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>neigh</td>
<td>0.01</td>
<td>1735</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>neigh</td>
<td>0.01</td>
<td>2000</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>real adj</td>
<td>0.01</td>
<td>1999</td>
</tr>
<tr>
<td>18 6</td>
<td>RGA/SA</td>
<td>400</td>
<td>2000</td>
<td>average</td>
<td>0.8</td>
<td>random</td>
<td>0.01</td>
<td>1872</td>
</tr>
</tbody>
</table>
Table B.2  Sample of some SA and combination sensitivity analyses.

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Type of Optimisation</th>
<th>Changes within constant temperature</th>
<th>Operators</th>
<th>Best Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>End Temperature</td>
<td>Cooling factor</td>
<td>Mutation Type</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 200</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 200</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SA 200</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SA 200</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SA 200</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SA 200</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>SA 600</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 600</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 600</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 600</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 500</td>
<td>0.0001</td>
<td>0.9</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.000001</td>
<td>0.98</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.000001</td>
<td>0.98</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.000204</td>
<td>0.995</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.00000991</td>
<td>0.995</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.001822</td>
<td>0.997</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 600</td>
<td>0.00019</td>
<td>0.997</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000006</td>
<td>0.997</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 600</td>
<td>0.00019</td>
<td>0.997</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>RGA/SA 600</td>
<td>0.0000001</td>
<td>0.998</td>
</tr>
</tbody>
</table>
Appendix C – Optimisation Model Code

This Appendix contains a copy of a sample code used to apply an optimisation model to a field sized sandy lagoon case study, as discussed in Chapter Four, Section 4.3. A similar code was used to model a lagoon with reversing flow (Chapter Four, Section 4.4), a laboratory sized lagoon (Chapter Four, Section 4.5) and a detached breakwater (Chapter Five).

```c
#include <stdio.h>                    /*This header file initiates standard library*/
#include <math.h>                     /*This header file includes exp, pow and log */
#include <stdlib.h>                   /*This header file has more standard library */
#include <string.h>                   /*This header file includes string functions */
#include <time.h>                     /*This header file includes the current time */
///*unixex*/ #include <process.h>     /*This header file allows the WC exe to open */
///*unixex*/ #include <conio.h>       /*This header file allows the WC exe to open */
///*unixex*/ #include <io.h>          /*This header file allows the WC exe to open */
#include <fcntl.h>                    /*This header file allows the WC exe to open */
#include <iostream.h>                 /*This header file includes c++ input/output */
/*unixin*/ #include <unistd.h>        /*This header file is needed for unix        */
/*unixin*/ #include <sys/wait.h>      /*This header file is needed for unix        */
#define a                10000        /*Number of bits (max string length)         */
#define b                1000         /*Maximum population size                    */
#define ge               20002        /*Maximum number of generations              */
#define po               50           /*default value for population number        */
#define ma               3000         /*default value for maximum generation number*/
#define px               0.8          /*default value for probability of crossover */
#define pm               0.01         /*default value for probability of mutation  */
#define CC               1            /*default value for type of crossover        */
#define MM               1            /*default value for type of mutation          */
#define DISC             0.1          /*default value for discresation interval     */
#define OPTYPE           1            /*default type of optimisation (RGA, BGA, SA) */
#define OPCOMB           0            /*default to combined GA and SA 1, 2 is for a* 
proscribed starting config with SA, else 0 */
#define PERTYPE          2            /*default type of SA perturbation            */
#define PERPROB          0.001        /*default SA perturbation probability        */
#define NEIGHBOUR_VAL    2000.0       /*value used to discritise                   */
#define NEIGHBOUR_USE    0            /*if 0 use random between, if one use value  */
#define STEP_SIZE        0.001        /*default value for lower random divide no   */
#define DIVL             0.01         /*default value for upper random divide no   */
#define DIVU             10000.0      /*default value for upper random divide no   */
#define PI               3.141
#define FALSE            0            /*Used by random number generating functions */
#define TRUE             1            /*Used by random number generating functions */
#define STRING_LENGTH    10           /*default number of bits in string           */
#define DENSITY          1000.0       /*density of water                           */
#define GRAVITY          9.81         /*acceleration due to gravity                */
#define MAXLOOPS         1            /*maximum number of loops used                */
#define LIMLOOPS         1            /*maximum number of loops possible            */
#define CO_FRICTION      0.01         /*coefficient of friction for bed drag        */
#define STILL_DEPTH      1.5          /*still water level (ie do)                   */
```
Appendix C  Morphology in Coastal and River Environments

#define FALL_VELOCITY 0.02 /*fall velocity in ms^-1 */
#define SAND_DIAMETER 90 0.0006 /*d90 of the sand */
#define SAND_DIAMETER_50 0.00024 /*d50 of the sand */
#define KIN_VISCOSITY 0.008001 /*kinematic viscosity at 20C (m^2/s) */
#define VON_KARMAN 0.4 /*Von Karman constant */
#define GSX 96 /*grid size in x direction */
#define GSY 16 /*grid size in y direction */
#define X_STEP 50.0 /*size of dx */
#define Y_STEP 50.0 /*size of dy */
#define GCYVU 1.6 /*maximum v constraint */
#define GCYL (-1.5) /*minimum v constraint */
#define GCUL (-1.5) /*minimum u constraint */
#define GCCVU 1.4 /*maximum n constraint */
#define GCCYL (-1.4) /*minimum n constraint */
#define NOM_LP 0 /*nomenclature of land point */
#define NOM_WN 1 /*nomenclature of normal wet point */
#define NOM_UZ 2 /*nomenclature of boundary condition u=0 */
#define NOM_VZ 4 /*nomenclature of boundary condition v=0 */
#define NOM_UV 6 /*nomenclature of boundary condition u=0, v=0*/
#define NOM_WM 8 /*nomenclature of specified water level */
#define NOM_SV 16 /*nomenclature of specified v */
#define NOM_NV 32 /*nomenclature of specified v */
#define AA 1000 /*maximum length of (x) grid */
#define BB 1000 /*maximum width (y) of grid */
#define SPECN 0.0 /*specified height above still water level */
#define SPECV 1.0 /*specified v */
#define SPECU 1.0 /*specified u */
#define WEIGHTC 1.0 /*weighting of continuity on objective function*/
#define WEIGHTM 1.0 /*weighting of momentum on objective function*/
#define SCALEG 1.0 /*scaling of momentum on objective function */
#define BLOCK1j 2 /*point where block1 is placed */
#define BLOCK1i 8 /*point where block1 is placed */
#define BLOCK2j 3 /*point where block2 is placed */
#define Ti 100.0 /*starting temperature for SA */
#define Tf 0.001 /*final temperature for SA */
#define COOL_FACT 0.988 /*temperature decrease rate for SA */
#define PCT_D_T 0.15 /*double temp if less than this accepted */
#define HAA 600 /*number of histogram intervals */
#define HBB 15 /*number of info entropy steps for matrix */
#define SCALEC 1.0 /*scaling of continuity on objective function*/
#define WEIGHTC 1.0 /*weighting of continuity on objective function*/
#define WEIGHTM 1.0 /*weighting of momentum on objective function*/
#define SPECN 0.0 /*specified height above still water level */
#define SPECV 1.0 /*specified v */
#define SPECU 1.0 /*specified u */
#define HYDRA_BW 1 /*if hydra3 used, 2 if breakwater used */
#define PERIOD 1.14 /*Wave period in seconds */
#define INITIAL_WAVE_HEIGHT 2.0 /*breakwater transmission coefficient */
#define HYDRA_BW 1.56 /*wave length at the breakwater (m) */
#define FNCALLNO 1 /*flow field calc. always (1) or x gen. (2) */
#define NOGENRC 100 /*no. of generations between fitness calc. */
The following set function prototypes, that are called by the main function:

```c
void calc_fitness(double cf_pop[a][b], double cf_L, int cf_m, int cf_type, int cf_bestpop, int cf_op_comb, double cf_qx[a][b], double cf_qy[a][b], double cf_flowx[a][b], double cf_flowy[a][b]);
void RandomInitialise(int ij, int kl);
double RandomUniform(void);
double RandomGaussian(double mean, double stddev);int RandomInt(int lower, int upper);
double RandomDouble(double lower, double upper);
void operation_file_write(int ofw_check1);
void fill_limits(double fl_ULgene[a], double fl_LLgene[a], int fl_L);
int default_check(void);
void print_results(int pr_itno, int pr_min, int pr_L, int pr_maxnogen);
int end_program(void);
void initial_defaults(int id_check1);
void prog_run_mess(void);
void mistake(void);
void other_mistake(int om_no);
void set_files(void);
void welcome_message(void);
void choose_values(void);
void loop_set(int ls_check1);
void set_grid_type(int sgt_check1);
void set_grid_size(int sgs_check1);
void optimisation_type(int ot_check1);
void discret_int(int di_check1);
void define_bit_no(double dbn_ULgene[a], double dbn_LLgene[a], int dbn_bit_string[a], double dbn_disc_string[a], int dbn_L, double dbn_disc);
void decode_binary(double db_l1_pop[a][b], double db_pop[a][b], int db_bit_string[a], double db_LLgene[a], double db_ULgene[a], int db_m, int db_L, int db_bno);
void simulated_annealing(double sa_population[a][b], double sa_fitness[b], double sa_min_population[ge], double sa_Lgene[a], double sa_ULgene[a], int sa_L, int sa_m, double sa_initial_temp, double sa_final_temp, double sa_cooling_factor, int sa_no_before_decrease, int sa_no_temp_decrease, double sa_acceptance_elasticity, int sa_per_type, double sa_probnut, int sa_check, double sa_qx[a][b], double sa_qy[a][b], double sa_flowx[a][b], double sa_flowy[a][b]);
void real_genetic_algorithm(void);
void binary_genetic_algorithm(void);
void crossover_type(int check5, int m, double probxover, int testL, double l4_population[a][b], double population[a][b]);
int mutation_type(int check6, int mutno, int m, double probmut, int testL, double l4_population[a][b], double population[a][b], double Lgene[a], double ULgene[a], int neighbour_use, double neighbour_val, double step_size);
int average_crossover(int ac_check5, int ac_m, double ac_probxover, int ac_testL, double ac_population[a][b], double ac_population[a][b]);
void one_point_crossover(int opc_m, double opc_probxover, int opc_testL, double opc_l4_population[a][b], double opc_population[a][b]);
void two_point_crossover(int tpc_check5, int tpc_m, double tpc_probxover, int tpc_testL, double tpc_l4_population[a][b], double tpc_population[a][b]);
int random_mutation(int rm_mutno, int rm_m, double rm_probmut, int rm_testL, double rm_Lgene[a], double rm_ULgene[a], int rm_neighbour_use, double rm_neighbour_val);
int true_adjacency_mutation(int tam_mutno, int tam_m, double tam_probmut, int tam_testL, double tam_l4_population[a][b], double tam_population[a][b], double tam_LLgene[a], double tam_ULgene[a], double tam_step_size);

int wang_zheng_mutation(int wzm_mutno, int wzm_m, double wzm_probmut, int wzm_testL, double wzm_l4_population[a][b], double wzm_population[a][b], double wzm_LLgene[a], double wzm_ULgene[a]);

void actual_fitness_value(void);

void GAmain(void);
void BWmain(void);

void initialise_flat_bed(double ifb_qx[a][b], double ifb_qy[a][b], double ifb_flowx[a][b], double ifb_flowy[a][b]);

void bwset_grid_type(int sgt_check1);
void bwset_grid_size(int sgs_check1);

void still_water_level(void);

void calc_fit_nc(double cfn_pop[a][b], double cfn_L, int cfn_m, int cfn_type, int cfn_bestpop, int cfn_op_comb, double cfn_qx[a][b], double cfn_qy[a][b], double cfn_flowx[a][b], double cfn_flowy[a][b]);

double find_fit_value(int ffv_gsx, int ffv_gsy, double ffv_u[a][b], double ffv_v[a][b], double ffv_n[a][b], double ffv_de[a][b], double ffv_qx[a][b], double ffv_qy[a][b], double ffv_flowx[a][b], double ffv_flowy[a][b]);

void change_results_name(int no_loops);

FILE *output;

FILE *outputf;
FILE *output2f;
FILE *GAops;
FILE *GAopsf;
FILE *GAlim;
FILE *GAlimf;
FILE *endresf;
FILE *endresf2;
FILE *pop_file;
FILE *pop_filef;
FILE *fit_file;
FILE *fit_filef;
FILE *intermin;
FILE *loopf;
FILE *loop;
FILE *grid_type;
FILE *grid_size;
FILE *grid_const;
FILE *water_count;

FILE *grid_typef;
FILE *grid_type2f;
FILE *grid_type3f;
FILE *grid_type4f;
FILE *grid_sizef;
FILE *grid_size2f;
FILE *grid_size3f;
FILE *grid_constf;
FILE *water_countf;
FILE *water_count2f;

FILE *grid_v;
FILE *grid_u;
FILE *grid_n;
FILE *grid_vf;
FILE *grid UF;
FILE *grid_nf;
FILE *grid_v2f;
FILE *grid_u2f;
FILE *grid_n2f;
FILE *grid_v3f;
FILE *grid_u3f;
FILE *grid_n3f;
FILE *grid_bitsf;
FILE *grid_bits2f;
FILE *grid_bits3f;
FILE *bestvf;
FILE *bestuf;
FILE *bestnf;
FILE *bestdef;
FILE *initialvf;
FILE *initialuf;
FILE *initialnf;
FILE *initialdef;
FILE *optitype3f;
FILE *optitype2f;
FILE *optitypef;
FILE *optitype;
FILE *discintf;
FILE *discint;
FILE *bitno;
FILE *bitnof;
FILE *bitmem;
FILE *bitmemf;
FILE *SAfit;
FILE *SAfitf;
FILE *startpopf;
FILE *startpop2f;
FILE *small_arcef;
FILE *small_arce2f;
FILE *small_arce3f;
FILE *small_arcof;
FILE *small_arco2f;
FILE *small_arco3f;
FILE *ddepthf;
FILE *ddepth2f;
FILE *ddepth3f;
FILE *wddepthf;
FILE *wddepth2f;
FILE *wddepth3f;
FILE *info_entf;
FILE *info_ent2f;
FILE *energy_totf;
FILE *still_waterf;
FILE *still_water2f;
FILE *still_water3f;

/**************************************************************************
****************************Main Function*********************************
/**************************************************************************

/**************************************************************************
******************************GA Function*********************************
**************************************************************************

void GAmain(){

}
char check1 = 'Y';                        /*While loop continuing variable        */
char check7 = 'D';
char check9 = 'D';

int check5, check6;                      /*Integer checks for user input        */
int check11;                             /*Integer checks for user input        */

int maxnogen;                            /*Maximum number of generations        */
int popnsize;                            /*Size of population                    */
double probxover;                        /*Probability of crossover              */
double probmut;                          /*Probability of mutation               */

int L;                                   /*Total integer no of bits for resol    */
int m;                                   /*Population number                     */

double min_fitness;                      /*Minimum fitness value                 */
int min_fitness_no;                      /*Population number of min fit value    */
int min_fitness_count;                    /*No of values with same min fitness    */

int itno;                                /*Iteration number                      */

int rni;                                 /*Random integer number for GA          */
int rn2;                                 /*Random integer number for GA          */

int *mask;                               /*Used for uniform crossover            */
double *xoave;                            /*Used for average crossover            */
double *limmid;                           /*Mid point between limits              */
int mutno;                               /*Number of mutations occurring in 1 gen*/

int *population)[b];                     /*Initial GA population                 */
int *l1_population)[b];                   /*GA popn used to compete in tournament */
int *l4_population)[b];                   /*GA popn at the end of a generation    */
int *min_population];                     /*Collection of min fitness for all gen */
int *mutno_all;                           /*Number of mutations in each generation*/

int total_min_fit;                       /*Total min fitness value for all gen   */
int min_fit_no;                           /*Gen in which total min fitness occurs */

int no_loops;
int max_loops;

int H2Ocount;
int mf_no;//number of generations at which absolute minimum objective function occurs for run
char temps1[40];//temporary storage
char tempd1;//temporary storage
double templf1;//temporary storage
int op_number;//optimisation type number (RGA=1, BGA=2, SA=3)
double disc_int;//discretisation interval
int bno;//number of bits required for BGA
int *bit_string;//number of bits that represent each variable in string
double neighbour_val;//value used to discritise between current value and bounds
int neighbour_use;//if 0 use random between, if one use value only
double step_size;//step size used in true adjacency mutation
char system_time[128];
char system_date[128];
double acceptance_elasticity;//used to determine initial starting temp
double initial_temp;//starting temperature for SA
double final_temp;//final temperature for SA
double cooling_factor;//temperature decrease rate for SA
int no_before_decrease;//number of iterations at current temperature before temperature is decreased
int no_temp_decrease;//number of temperature decreases without improvement before stopping criteria met

210
int per_type; // type of perturbation
double per_prob; // probability of perturbation

double (*qx)[b];  // volumetric transport difference in x direction
double (*flowx)[b];  // absolute volumetric transport difference in x direction
double (*qy)[b];  // volumetric transport difference in y direction
double (*flowy)[b];  // absolute volumetric transport difference in y direction

int hydra_bw; // used to determine if points written according to breakwater or hydra3

/*****************************************
* Allocation of dynamic memory for arrays *
*******************************************/

if ((ULgene = new double[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((LLgene = new double[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((mask = new int[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((xoave = new double[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((limmid = new double[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((population = new double[a][b]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((l1_population = new double[a][b]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((l4_population = new double[a][b]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((min_population = new double[ge]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((min_populationi = new int[ge]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((min_pop_value = new double[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((mutno_all = new int[ge]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((bit_string = new int[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((disc_string = new double[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((qx = new double[a][b]) == NULL) {
  printf("Memory Allocation Failure for Array qx\n");
  exit(0);
}
if ((flowx = new double[a][b]) == NULL) {
  printf("Memory Allocation Failure for Array flowx\n");
  exit(0);
}
if ((qy = new double[a][b]) == NULL) {
  printf("Memory Allocation Failure for Array qy\n");
  exit(0);
}
if ((flowy = new double[a][b]) == NULL) {
    printf("Memory Allocation Failure for Array flowy\n");
    exit(0);
}

H2Ocount = 0;
mf_no = 0;
op_number = 0;
disc_int = 0.0;
bno = 0;
testL = 0;
mistake_check = 0;
op_comb = 0;
initial_temp = 0.0;//starting temperature for SA
final_temp = 0.0;//final temperature for SA
cooling_factor = 0.0;//temperature decrease rate for SA
no_before_decrease = 0;//number of iterations at current temperature before temperature is decreased
no_temp_decrease = 0;//number of temperature decreases without improvement before stopping criteria met
acceptance_elasticity = 0.0;//used to determine initial starting temp
per_type = 0;
per_prob = 0.0;
neighbour_val = 0.0;//value used to discritise between current value and bounds
step_size = 0.0;//step size used in true adjacency mutation

no_loops = 1;
max_loops = MAXLOOPS;

/*rename files: */
int spec1, spec2, spec3, spec4;
double spec5, spec6;
char spec7[100];
int wccall;
wccall = 0;
mutno=0;                                 /*Sets number of mutations in a gen to 0*/
min_fit_no=0;                            /*Sets minimum fitness number to zero */
hydra_bw=0;

/*Initialisation of arrays*/
for(i=0; i < a; i++){
    ULgene[i] = 0.0;                     /*Upper limit of each variable */
    LLgene[i] = 0.0;                     /*Lower limit of each variable */
    mask[i] = 0;                         /*Used for uniform crossover */
    xoave[i] = 0.0;                      /*Used for average crossover */
    limmid[i] = 0.0;                     /*Mid point between limits */
    min_pop_value[i] = 0.0;              /*Collection of gene values for all mins*/
    bit_strIng[i] = 0;                   /*Bits for each variable in string */
    disc_string[i] = 0.0;                /*Discretisation interval in variable */
    for(j=0; j < b; j++){
        population[i][j] = 0.0;          /*Initial GA population */
        l1_population[i][j] = 0.0;       /*GA popn used to compete in tournament */
        l4_population[i][j] = 0.0;       /*GA popn at the end of a generation */
        qx[i][j] = 0.0;
        qy[i][j] = 0.0;
        flowx[i][j] = 0.0;
        flowy[i][j] = 0.0;
    }
}
for(j=0; j < b; j++){
    raw_fitness_population[j] = 0.0;     /*Raw fitness values of min function */
}
for(k=0; k < ge; k++){
    min_population[k] = 0.0;           /*Collection of min fitness for all gen */
    min_populationi[k] = 0;            /*Number of population with same fitness*/
    mutno_all[k] = 0;                  /*Number of mutations in each generation*/
}
for(i = 0; i < 6000; i++){                   /*change starting random numbers*/
tempf1 = RandomDouble(1.0, 10.0);
}

//no_keys welcome_message();//welcomes user to program, explains GA

212
set_files(); //writes all the necessary file information for correct output

/********************************************************************************************
* The following while loop allows the user to decide whether to repeat the GA and allows them to make this choice as many times as they want.  
********************************************************************************************/

while(1) {
    no_loops = 1;
    max_loops = MAXLOOPS;

    /*************************************************************************************
    * Variable Set Up  
    *************************************************************************************/
    no_keys = choose_values(); //choose default or own values
    loop = fopen( "loop.txt", "r" );
    if( loop == NULL ){
        other_mistake(1); //Can't open file
    } else{
        /* Set pointer to beginning of file: */
        fseek( loop, 0L, SEEK_SET);     //0
        fscanf(loop, "%d", &max_loops);
        fclose( loop );
    }

    optitype = fopen ( "optype.txt", "r" );
    if( optitype == NULL ){
        other_mistake(1); //Can't open file
    } else{
        fscanf(optitype, "%d", &op_number);
        fclose(optitype);
    }
    op_comb = OPCOMB;
    GAops = fopen( "GAops.txt", "r" );
    if( GAops == NULL ){
        other_mistake(1); //Can't open file
    } else{
        /* Set pointer to beginning of file: */
        fseek( GAops, 0L, SEEK_SET);     //0
        fscanf(GAops, "%s", &spec7);
        fscanf(GAops, "%d", &spec1);
        fscanf(GAops, "%s", &spec2);
        fscanf(GAops, "%d", &spec3);
        fscanf(GAops, "%s", &spec4);
        fscanf(GAops, "%d", &spec5);
        fscanf(GAops, "%s", &spec6);
        fclose( GAops );
    }
    check5=spec1;                         /* Crossover type code */
    check6=spec2;                         /* Mutation type code */
    popnsize=spec3;                       /* Population size */
    maxnogen=spec4;                       /* Maximum generation number */
    probxover=spec5;                      /* Probability of crossover */
    probmut=spec6;                        /* Probability of mutation */

    water_count = fopen ( "w_count.txt", "r" );
    if( water_count == NULL ){
        other_mistake(1); //Can't open file
    } else{
        fscanf(water_count, "%d ", &L); //writes number of points to optimise
        fclose(water_count);
    }
    fill_limits(ULgene, LLgene, L); //****need to change the limits

    if(op_number == 2){
        discint = fopen ( "discint.txt", "r" );
        if( discint == NULL ){
            other_mistake(1); //Can't open file
        } else{
            fscanf(discint, "%lf", &disc_int);
        }
    }
}
fclose(discint);
}
#define_bit_no(ULgene, LLgene, bit_string, disc_string, L, disc_int);
bitno = fopen("bit_no.txt", "r");
if(bitno == NULL){//Can't open file
else{
    fscanf(bitno, "%d ", &bno);//writes number of points to optimise
    fclose(bitno);
}
}
initial_temp = Ti;          /*starting temperature for SA                */
final_temp = Tf;           /*final temperature for SA                   */
cooling_factor = COOL_FACT; /*temperature decrease rate for SA           */
no_before_decrease = NBDE;  /*no it at current temp before temp decrease */
no_temp_decrease = NTDP;     /*st crit total temp decre w/o improve (3-7) */
acceptance_elasticity = ELASTICITY;//used to determine initial starting temp
per_type = PERTYPE;         /*type of perturbation*/
per_prob = PERPROB;         /*probability of perturbation*/
neighbour_val = NEIGHBOUR_VAL;//value used to discritise between current value and bounds
neighbour_use = NEIGHBOUR_USE;//if 0 use random between, if one use value only
step_size = STEP_SIZE;      /*step size used in true adjacency mutation*/
if((op_number == 1) || (op_number == 3)){
testL = L;
}
else if(op_number == 2){
testL = bno;
}
else{
    mistake();
    printf(" L mistake");
}
while(1){
    prog_run_mess();//tells user GA is running
    if((op_number == 1) || (op_number == 2)){
        m = popnsize;      /*Population size                            */
        /***************************************************************
        **********Population Initialisation*******************************
        ***************************************************************/
        if(op_number == 1){//initialisation of real GA population
            for(i = 0; i < L; i++){                   /*filling the population array*/
                for(j = 0; j < m; j++){
                    population[i][j] =  RandomDouble(LLgene[i], ULgene[i]);
                }
            }
        }
        else if(op_number == 2){//initialisation of binary GA population
            for(j = 0; j < m; j++){
                for(i = 0; i < bno; i++){
                    population[i][j] = double(RandomInt(0, 1));
                }
            }
        }
        else if(op_number == 2){{//initialisation of binary GA population
            for(j = 0; j < m; j++){
                for(i = 0; i < bno; i++){
                    population[i][j] = double(RandomInt(0, 1));
                }
            }
        }
        /*Decodes binary representation to a real number*/
decode_binary(l1_population, population, bit_string, LLgene, disc_string, m, L, bno);
}

else if (op_number == 3){//simulated annealing takes place
    printf("Simulated annealing not yet defined");
} else{
    mistake();
    printf("s 1");
}
/*Calculates the individual's raw fitness*/
pop_file = fopen("population.txt", "w+");
if( pop_file == NULL ){  //Can't open file
    other_mistake(1);  //Can't open file
} else{
    for(j=0; j < m; j++){
        for(i=0; i < L; i++){
            fprintf(pop_file, "%f ", population[i][j]);
        }
        if(i==(L-1)){
            fprintf(pop_file, "%f
", population[i][j]);
        } else{
            mistake();
            printf("s 2");
        }
    }
    fclose(pop_file);
}
calc_fitness(population, L, m, 1, 1, 0, qx, qy, flowx, flowy);
fit_file = fopen("fitness.txt", "r");
if( fit_file == NULL ){  //Can't open file
    other_mistake(1);  //Can't open file
} else{
    for(j=0; j < m; j++){
        fscanf(fit_file, "%lf", &raw_fitness_population[j]);
    }
    fclose(fit_file);
}
/*Check fittest member of population each generation*/
into=0;
min_fitness = raw_fitness_population[0]; /*Sets into to 0 as the first iteration*/
min_fitness_no = 0; /*Sets the initial min fit to be the first fitness value*/
min_fitness_count = 1; /*Sets initial count to 1 as for 1st member of popn*/
/*Search through population to find fittest member for current generation*/
for(j=0; j < m; j++){
    /*If the fitness value is the same as the min fitness value, the min
    *fitness value stays the same, and number of times this fitness occurs
    *is increased by 1*/
    if(raw_fitness_population[j]==min_fitness) {
        min_fitness_count = min_fitness_count + 1;
    } else if (raw_fitness_population[j]<min_fitness) {
        min_fitness = raw_fitness_population[j];
        min_fitness_no = j;
        min_fitness_count = 1;
    } else if (raw_fitness_population[j]>min_fitness) {
        min_fitness = raw_fitness_population[j];
        min_fitness_no = j;
        min_fitness_count = 1;
    }
}
min_fitness_count = min_fitness_count;
}
else {
    mistake(); /*Tells the user of a mistake*/
    printf("s 3\n");
}

/*The following lines assign the best fitness values to the array that contains values for all generations.*/
min_population[itno]=min_fitness;
min_populationi[itno]=min_fitness_count;
intermin = fopen("intermin.txt", "w+");
if( intermin == NULL ){
    other_mistake(1); /*Can't open file*/
}
else{

    fprintf (intermin, " %s %d %s %d", "Crossover_type", check5, "Mutation_type", check6);
    fprintf (intermin, " %s %d %s %d", "Population_Size", popsize, "Generation_no", maxnogen);
    fprintf (intermin, " %s %f %s %f", "Crossover_prob", probxover, "Mutation_prob", probmut);
    fprintf (intermin, " %s", "Generation");
    for(i=0; i < L; i++){
        fprintf( intermin, " %d", (i+1));
    }
    fprintf( intermin, " %s %s", "Objective_Function_Value", "No_of_times");
    fprintf(intermin, "%d ", (itno + 1));
    for(i=0; i < L; i++){
        fprintf( intermin, "%d", population[i][(min_fitness_no)];
    }
    fprintf( intermin, "%d
", (itno + 1));
    for(i=0; i < L; i++){
        fprintf(intermin, "%e %e %e %e %e %e
", population[i][j], l1_population[i][j], l2_population[i][j], l3_population[i][j], l4_population[i][j], l5_population[i][j];
    }
    for(i=0; i < L; i++){
        fprintf(intermin, "%e %e %e %e %e %e
", population[i][j], l1_population[i][j], l2_population[i][j], l3_population[i][j], l4_population[i][j], l5_population[i][j];
    }
}
if(op_number == 2){
    calc_fitness(population, L, m, 4, min_fitness_no, 0, qx, qy, flowx, flowy);//l4_population, L, m, 2, mf_no, 0, qx, qy, flowx, flowy type, bestpop, op_comb);
}

/************************************************************
***************************GA Loop**************************
************************************************************/
while(1){ /*Allows GA to perform operations for the required no of generations*/
    itno = itno + 1; /*Assigns the current generation number*/

    //*****************************************************************************************/
    Tournament
    /*********************************************************************************************/
    while(1){ /*Tournament is performed for m pairs of population members*/

        /*Determines the two members that will compete against each other to reproduce*/
        rn1=RandomInt(0,(m-1));
        rn2=RandomInt(0,(m-1));

        /*If the fitness of the rn1 member is less than the rn2 member, then it gets to continue on the mating pool. The rn2 member is lost from the gene pool.*/
        if (raw_fitness_population[rn1] <= raw_fitness_population[rn2]){ /*Transfer binary numbers*/
            for(i = 0; i < m/2; i++){
                population[i][i] = l1_population[i][i];
                population[i][i] = l2_population[i][i];
                population[i][i] = l3_generation[i][i];
                population[i][i] = l4_generation[i][i];
                population[i][i] = l5_generation[i][i];
            }
        }

        /*If the fitness of the rn1 member is equal to the rn2 member, then it does not matter which is lost from the gene pool and which gets to continue on to the mating pool, so the new population is assigned the rn1 values.*/
        if (raw_fitness_population[rn1] == raw_fitness_population[rn2]){ /*Filling the population array*/
            for(i = 0; i < m; i++){
                population[i][k] = population[i][k];
            }
        }

        /*If the fitness of the rn1 member is greater than the rn2 member, then it gets to continue on the mating pool. The rn2 member is lost from the gene pool.*/
        if (raw_fitness_population[rn1] > raw_fitness_population[rn2]){ /*Transfer binary numbers*/
            for(i = 0; i < m/2; i++){
                population[i][i] = l1_population[i][i];
                population[i][i] = l2_population[i][i];
                population[i][i] = l3_generation[i][i];
                population[i][i] = l4_generation[i][i];
                population[i][i] = l5_generation[i][i];
            }
        }
    }
}

if (FNCALLNO == 2){
    calc_fitness(population, L, m, 4, min_fitness_no, 0, qx, qy, flowx, flowy);//l4_population, L, m, 2, mf_no, 0, qx, qy, flowx, flowy type, bestpop, op_comb);
}

}
*is lost from the gene pool and the rn2 member gets to continue on to the *
*mating pool. /*

else if (raw_fitness_population[rn1] > raw_fitness_population[rn2]) {
  for(i = 0; i < testL; i++) {
    l4_population[i][k] = population[i][rn2];
  }
}

else {
  mistake();
  printf("s 4");
}

/**************************************************************
**************************Crossover**************************
/***************************************************************/
crossover_type(check5, m, probxover, testL, l4_population, population);
/**************************************************************
**************************Mutation***************************
/***************************************************************/
mutation_type(check6, mutno, m, probmut, testL, l4_population, population, LLgene, ULgene, neighbour_use, neighbour_val, step_size);
mutno_all[itno]=mutno; /*Records number of mutations*/
/**************************************************************
******************Convert Binary to Real********************
***************************************************************/
if(op_number == 2){//if using binary GA
  for(j = 0; j < m; j++) {
    for(i = 0; i < bno; i++){
      population[i][j] = l4_population[i][j];
    }
  }
  /*Decodes binary representation to a real number*/
decode_binary(population, l4_population, bit_string, LLgene, disc_string, m, L, bno);
}
/**************************************************************
*********************Population Evaluation*******************
***************************************************************/
/*Calculates the individual's raw fitness*/
pop_file = fopen( "population.txt", "w+");
if( pop_file == NULL ){
  other_mistake(1);  //Can't open file
  else{
  for(j=0; j < m; j++){
    for(i=0; i < L; i++){
      fprintf(pop_file, "%f ", l4_population[i][j]);
    }
    if(i<(L-1)){
      fprintf(pop_file, "\n");
    }
  }
}
fclose(pop_file);
}
if (FNCALLNO == 1){//if calling each generation
  calc_fitness(l4_population, L, m, i, 0, qx, qy, flowx, flowy);
}
else if (FNCALLNO == 2){
  if((itno % NOGEBC) == 0){//number of generations a factor of NOGEBC then calc proper fitness
    calc_fitness(l4_population, L, m, i, 0, qx, qy, flowx, flowy);
  }
}
else if ((itno % NOGEBC) != 0){//use fast OF calcs for most morphologies
  calc_fit_nc(l4_population, L, m, i, 0, qx, qy, flowx, flowy);
//end use fast OF calcs for most morphologies
else{//mistake can use neither fast or slow fitness calcs
mistake();
printf("\nwrong generation number calculations");
//end use neither fast or slow fitness calcs
}
else{
  mistake();
  printf("\nwrong function call number");
}

fit_file = fopen( "fitness.txt", "r");

if( fit_file == NULL ){ //Can't open file
  other_mistake(1);  //Can't open file
}else{
  for(j=0; j < m; j++){
    fscanf(fit_file, "%lf", &raw_fitness_population[j]);
  }
  fclose(fit_file);
}

/************************************************************
*************Determination of Least Fit Member**************
************************************************************/
max_fitness = raw_fitness_population[0];
max_fitness_no = 0;
/*Search through population to find least fittest member for current generation*/
for(j=0; j < m; j++){
  /*If the fitness value is more than or equal to the previous max fitness  *
   *(worst) value then it becomes the new max fitness value. This continues* *
   *until the maximum fitness for the whole of the current generation is    *
   *found. A note is made of the member of the population that gave the    *
   *fitness value.                                                          */
  if(raw_fitness_population[j]>max_fitness){
    max_fitness = raw_fitness_population[j];         /*ie if i=0 max fitness of the 1st member*/
    max_fitness_no = j;
  } /*If the fitness value is less than the max fitness, there is no change*/
}
else if(raw_fitness_population[j]<=max_fitness){
  max_fitness = max_fitness;
}
else{
  mistake();
  printf("s 6");
}

/************************************************************
***************Determination of Fittest Member**************
************************************************************/
/*Check fittest member of population each generation*/
min_fitness = raw_fitness_population[0];
min_fitness_no = 0;
min_fitness_count = 0;
/*Search through population to find fittest member for current generation*/
for(j=0; j < m; j++){
  /*If the fitness value is the same as the min fitness value, the min      *
   *fitness value stays the same, and number of times this fitness occurs   *
   *(min_fitness_count) is increased by 1                                   */
  if(raw_fitness_population[j]==min_fitness){
    min_fitness_count = min_fitness_count + 1;
  } /*If the fitness value is less than the previous min fitness value then it* *
   *becomes the new min fitness value and the number of times this value     *
   *occurs is returned to 1. A note is made of the member of the population* *
   *that gave this fitness value.                                           */
  else if(raw_fitness_population[j]<min_fitness){
    min_fitness = raw_fitness_population[j];
    min_fitness_no = j;
    min_fitness_count = 1;
  } /*If the fitness value is more than or equal to the previous min fitness */
min_fitness_no = j;
min_fitness_count = 1;
}
/*If the fitness value is greater than the min fitness, there is no change*/
else if (raw_fitness_population[j] > min_fitness)
  min_fitness_count = min_fitness_count;
}
else{
mistake();
printf("s 7");
}
/*The following lines assign the best fitness values to the array that contains values for all generations.*/
min_population[itno] = min_fitness;
min_populationi[itno] = min_fitness_count;

//calculate flows for minimum fitness morphology to use in fast calcs.

if (FNCALLNO == 2) {//if performing fast run
  if ((itno % NOGEBC) == 0) {//number of generations a factor of NOGEBC then calc proper flow
    calc_fitness(l4_population, L, m, 4, min_fitness_no, 0, qx, qy, flowx, flowy);
  } //end number of generations a factor of NOGEBC then calc proper flow
}

intermin = fopen("intermin.txt", "a+");  
if (intermin == NULL) 
  other_mistake(1);  //Can't open file 
  mistake_check = mistake_check + 1; 
else 
  fprintf(intermin, "%d ", (itno + 1)); 
  for (i=0; i < L; i++) 
    fprintf(intermin, "%e ", l4_population[i][min_fitness_no]); 
  fprintf(intermin, "%e ", min_fitness); 
  fprintf(intermin, "%d\n", min_fitness_count); 
  fclose(intermin);

/***************************************************************
***********Overall Fittest Member***************/
/***************************************************************
/*Check fittest member of all generations*/
total_min_fit = min_population[0]; 
/*Search through fittest member of each population from each generation to find fittest member from all generations*/
for (k=0; k < (itno+1); k++)
  /*If the fitness value is less than the previous min fitness value then it becomes the new min fitness value and the number of times this value occurs is returned to 1. A note is made of the member of the population that gave this fitness value.*/
  if (min_population[k] <= total_min_fit)
    total_min_fit = min_population[k]; 
    min_fit_no = k; 
  /*If the fitness value is greater than or equal to the min fitness, there is no change*/
else if (min_population[k] > total_min_fit)
  total_min_fit = total_min_fit; 
else{
mistake();
printf("s 8");
}
/*The following replaces the population member that has the worst fitness (ie *
*max fitness value) with the previous generation minimum fitness value, if *
*the current generation minimum fitness value is greater than the previous *
*generation value.*/

if (min_population[itno]>total_min_fit){
    intermin = fopen("intermin.txt", "r");
    if (intermin == NULL)
        other_mistake(1); //Can't open file
    else{
        fscanf (intermin, "%s", &temps1);//crossover_type
        fscanf (intermin, "%d", &tempd1);//crossover value
        fscanf (intermin, "%s", &temps1);//mutation_type
        fscanf (intermin, "%d", &tempd1);//mutation value
        fscanf (intermin, "%s", &temps1);//population_size
        fscanf (intermin, "%d", &tempd1);//population no
        fscanf (intermin, "%s", &temps1);//generation no
        fscanf (intermin, "%d", &tempd1);//crossover probability
        fscanf (intermin, "%d", &tempd1);//mutation probability
        fscanf (intermin, "%s", &temps1);//crossover_probability
        fscanf (intermin, "%d", &tempd1);//mutation_probability
        for(i=0; i < L; i+++){
            fscanf (intermin, "%d", &tempd1);//bit numbers
            fscanf (intermin, "%s", &temps1);//objective_function_value
            fscanf (intermin, "%s", &temps1);//number_of_times
            for(j=0; j < min_fit_no ; j+++){
                fscanf( intermin, "%d", &tempd1);//bit values
            }
        }
        fclose(intermin);
        raw_fitness_population[max_fitness_no]=min_population[min_fit_no];
    }
    else if (min_population[itno]<total_min_fit){
        min_population[itno]=min_population[itno];
    } else {
        mistake();
        printf("s 9");
    }
    /*Transfers the new population so it can be used in the new generation*/
    if (op_number == 1){
        for(j = 0; j < m; j++){                   /*filling the population array*/
            for(i = 0; i < L; i++)
                population[i][j] = l4_population[i][j];
        }
    } else if(op_number == 2){
        convert_binary(l1_population, l4_population, bit_string, LLgene, disc_string, m, L, bno);//convert
        l4_population into binary l1_population
    } else{
        mistake();
        printf("s 10");
    }
    mf_no = min_fit_no;//assigns min fitness population number to different value to be used overall
    /**************************Ending the GA loop*****************/
/* Determines if the required number of iterations have occurred. If the number of iterations is the same as the maximum number of required generations, then the GA loop terminated. (maxnogen - 1) is used as due to the specifications of arrays, the first itno is assigned a value of zero. */

// Flush output
/*unixin*/
fflush(stdout);//used in unix to flush output to screen

if(itno < (maxnogen-1)) {
  itno=itno;
} else if(itno == (maxnogen-1)) {
  break;  /*ends loop*/
} else {
  mistake();
  printf("s 11");
  break;  /*ends loop*/
}

}  //GA generation loop while

/******************************************************************************/
/*******************Overall Fittest Member*******************
******************************************************************************/
/*Check fittest member of all generations*/
total_min_fit = min_population[0];
/*Search through fittest member of each population* from each generation to find fittest member from * all generations*/
for(k=0; k < (itno+1); k++) {
  /*If the fitness value is less than the previous min fitness value then it * becomes the new min fitness value and the number of times this value * occurs is returned to 1. A note is made of the member of the population* that gave this fitness value. */
  if (min_population[k]<total_min_fit){
    total_min_fit= min_population[k];
    min_fit_no = k;
  }
  /*If the fitness value is greater than or equal to the min fitness, there * is no change*/
  else if (min_population[k]>=total_min_fit){
    total_min_fit = total_min_fit;
  }
  else{
    mistake();
    printf("s 12");
  }
}

mf_no = min_fit_no;  //assigns min fitness population number to different value to be used overall
printf("\nmf_no = %d", mf_no);
printf("\noverall fitness = %f", min_population[mf_no]);
output = fopen("allmin.txt", "w+ ");
if( output == NULL ){ other_mistake(1); //Can't open file }
else{
  fprintf(output, " %s %d %s %d", "Crossover_type", check5, "Mutation_type", check6);
  fprintf(output, " %s %d %s %d", "Population_Size", popnsize, "Generation_no", maxnogen);
  fprintf(output, " %s %f %s %f", "Crossover_prob", probxover, "Mutation_prob", probmut);
  fprintf(output, " %s %s %s %s", "Generation");
  for(i=0; i < L; i++){
    fprintf(output, " %d", (i+1));
  }
  fprintf(output, " %s %s
", "Objective_Function_Value", "No_of_times");
  intermin = fopen("intermin.txt", "r");
  if( intermin == NULL ){
    other_mistake(1); //Can't open file }
  else{
    fscanf (intermin, "%s", &temps1);
    fscanf (intermin, "%d", &tempd1);
Appendix C  Morphology in Coastal and River Environments

```c
fscanf(intermin, "%s", &temps1);
fscanf(intermin, "%d", &tempd1);
fscanf(intermin, "%s", &temps1);
fscanf(intermin, "%d", &tempd1);
fscanf(intermin, "%s", &temps1);
fscanf(intermin, "%d", &tempd1);
fscanf(intermin, "%s", &temps1);
fscanf(intermin, "%d", &tempd1);
fscanf(intermin, "%s", &temps1);
fscanf(intermin, "%d", &tempd1);
fscanf(intermin, "%s", &temps1);
fscanf(intermin, "%s", &temps1);
for(i=0; i < L; i++){
    fscanf(intermin, "%d", &tempd1);
    fscanf(intermin, "%lf", &templf1);
    for(k=0; k < maxnogen; k++){
        for(i=0; i < L; i++){
            fscanf(intermin, "%d", &tempd1);
            fscanf(intermin, "%lf", & population[i][0]);
        }
        fclose(intermin);
    }
    fscanf(intermin, "%d", & tempd1);
    fscanf(intermin, "%s", &temps1);
    fprintf( output, " %d", min_population[k], min_populationi[k]);
    fclose( output);
}
print_results(itno, min_fit_no, L, maxnogen);
calc_fitness(l4_population, L, m, 2, mf_no, 0, qx, qy, flowx, flowy);
```

simulated_annealing(population, raw_fitness_population, min_population, LLgene, ULgene, L, m, initial_temp, final_temp, cooling_factor, no_before_decrease, no_temp_decrease, acceptance_elasticity, per_type, per_prob, op_comb, 0, qx, qy, flowx, flowy);
}  //end GA if statement

/**************Simulated Annealing**********************/

else if(op_number == 3) {
    m = popnsize;  /*Population size */
    printf("m=%d",m);
    simulated_annealing(population, raw_fitness_population, min_population, LLgene, ULgene, L, m, initial_temp, final_temp, cooling_factor, no_before_decrease, no_temp_decrease, acceptance_elasticity, per_type, per_prob, op_comb, 0, qx, qy, flowx, flowy);
}  //end of SA else if statement (SA section)
else{ //if neither GA nor SA
    mistake();
} //end of neither GA or SA else statement

change_results_name(no_loops);
if (no_loops == max_loops){
    break;
} else if (no_loops < max_loops){
    no_loops = no_loops + 1;
} else{
    mistake();
}

}  //end whole GA while loop

/*The following allows the user to loop back and choose a different scenario*/
check11 = end_program(); //Needs to be linked to a button
if (check11 == 1){
    break;  /*Ends loop*/
}  //end whole program while loop

/*Deletion of dynamic memory for arrays*/
delete [] ULgene;
delete [] LLgene;
delete [] mask;
delete [] xoave;
delete [] limmid;
delete [] population;
delete [] raw_fitness_population;
delete [] ll_population;
delete [] l4_population;
delete [] min_population;
delete [] min_populationi;
delete [] min_pop_value;
delete [] mutno_all;
delete [] bit_string;
delete [] disc_string;
delete [] qx;
delete [] flowx;
delete [] qy;
delete [] flowy;
//delete [] printlimits;
}  /*End of GA Function*/
void simulated_annealing(double sa_population[a][b], double sa_fitness[b], double sa_min_population[ge], double sa_LLgene[a], double sa_ULgene[a], int sa_L, int sa_m, double sa_initial_temp, double sa_final_temp, double sa_delta_fitness, double sa_acceptance_elasticity, double sa_per_type, double sa_probmut, int sa_check, int sa_best_start, double sa_qx[a][b], double sa_qy[a][b], double sa_flowx[a][b], double sa_flowy[a][b]) {
    int sa_no_accepted; // number of perturbations accepted by Metropolis Criterion
    double sa_Metropolis_criterion; // Metropolis criterion
    double sa_current_temp; // current temperature
    double sa_delta_fitness; // the difference between current and previous fitness
    double sa_current_convergence; // number of solutions that have converged at current temperature
    double sa_overall_convergence; // number of solutions that have converged since minimum fitness in previous temperatures
    int sa_innerit; // inner iteration number of SA loop
    int sa_i; // for for loops
    double sa_k; // for for loops
    double sa_rn3; // random number
    double sa_rn4; // random number
    int sa_innerit; // for for loops
    double sa_qx[a][b]; // for for loops
    double sa_qy[a][b]; // for for loops
    double sa_flowx[a][b]; // for for loops
    double sa_flowy[a][b]; // for for loops

    int sa_no_accepted = 0; // number of perturbations accepted by Metropolis Criterion
    double sa_Metropolis_criterion = 0.0; // Metropolis criterion
    double sa_current_temp = 0.0; // current temperature
    double sa_delta_fitness = 0.0; // the difference between current and previous fitness
    double sa_current_convergence = 0.0; // number of solutions that have converged at current temperature
    double sa_overall_convergence = 0.0; // number of solutions that have converged since minimum fitness in previous temperatures
    int sa_innerit = 0; // inner iteration number of SA loop
    double sa_qx[a][b]; // for for loops
    double sa_qy[a][b]; // for for loops
    double sa_flowx[a][b]; // for for loops
    double sa_flowy[a][b]; // for for loops

    if (sa_check == 0) {
        for (sa_i = 0; sa_i < sa_L; sa_i++) {
            sa_population[sa_i][sa_innerit] = RandomDouble(sa_LLgene[sa_i], sa_ULgene[sa_i]);
        }
    } // end else if for sa_check==0
    else if (sa_check == 1) {
        // do nothing as take the best min value from GA
    } // end else if for sa_check==1
    else if (sa_check == 2) { // uses best values found from a previous GA or users choice
        startpopf = fopen("sastapop.txt", "r");
        if (startpopf == NULL) {
            other_mistake(1); // Can't open file
        } else {
            for (sa_i = 0; sa_i < sa_L; sa_i++) {
                fscanf(startpopf, "%lf", &sa_population[sa_i][sa_innerit]);
            }
            fclose(startpopf);
        } // end else if for sa_check==2
    } else if (sa_check does not equal 0, 1, or 2 then a mistake has occurred)
    } // end else for mistake if sa_check not equal to the right value

    calc_fitness(sa_population, sa_L, sa_m, sa_check, sa_qx, sa_qy, sa_flowx, sa_flowy);
SAfit = fopen("safit.txt", "r");
if (SAfit == NULL)
    other_mistake(1); //Can't open file
else
    fscanf(SAfit, "%lf", ssa_fitness[sa_innerit]);
close(SAfit);

/***************************************************************
********************Set initial values**************************
***************************************************************/

sa_no_accepted = 1;
sa_current_convergence = 0;
sa_overall_convergence = 0;
sa_itno = 0;

/***************************************************************
**************Calculate initial temperature******************
***************************************************************

sa_initial_temp = (0.1*sa_fitness[sa_innerit])/(log(sa_acceptance_elasticity));
sa_current_temp = sa_initial_temp;

while(1) //keep raising temperature until stopping criteria met loop
    sa_no_accepted = 0;

while(1) //initial increase temperature while loop
    sa_innerit = sa_innerit + 1;

/***************************************************************
****************Purturbate initial guess************************
***************************************************************

/***************************************************************
*********************Random Mutation*************************
***************************************************************

if(sa_per_type == 1) {
    for(sa_i = 0; sa_i < sa_L; sa_i++) {
        /*A random value is calculated to determine whether */
        /*mutation occurs for that gene.*/
        sa_rn3=RandomDouble(0.0, 1.0);
        /*If the value generated is less than or equal to the */
        /*user defined probability of mutation, then mutation */
        /*occurs as below.*/
        if (sa_rn3<=sa_probmut) {
            /*If the mutation occurs, the new random value */
            /*generated is within the specified limits of this */
            /*variable.*/
            /*ie child(gene) = random number*/
            sa_rn4=RandomDouble(sa_LLgene[sa_i], sa_ULgene[sa_i]);
            sa_population[sa_i][sa_innerit] = sa_rn4;
        }
        else if (sa_rn3>sa_probmut) {
            sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
        }
        else {
            mistake();
        }
    } /*end if sa_per_type is 1*/

/***************************************************************
**************Random Mutation for one decision variable only********
***************************************************************

else if(sa_per_type == 2) {
    sa_rn1 = RandomInt(0, (sa_L - 1));
    for(sa_i = 0; sa_i < sa_rn1; sa_i++) {
        sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
    }
    sa_i = sa_rn1;
    sa_rn4=RandomDouble(sa_LLgene[sa_i], sa_ULgene[sa_i]);
    sa_population[sa_i][sa_innerit] = sa_rn4;
}
for(sa_i = (sa_rn1 + 1); sa_i < sa_L; sa_i++){
    sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
}
//end if sa_per_type is 2
******************************************************************************
**********Random Mutation**********
******************************************************************************
else if(sa_per_type == 3){
    //end if sa_per_type is 3
******************************************************************************
**********Random Mutation**********
******************************************************************************
else if(sa_per_type == 4){
    //end if sa_per_type is 4
else{
    //mistake();
} //end else mistake if sa_per_type is not valid
******************************************************************************
***********Calculate new fitness***********
******************************************************************************
calc_fitness(sa_population, sa_L, sa_m, 3, sa_innerit, sa_check, sa_gx, sa_gy, sa_flowx, sa_flowy);
    SAIfit = fopen( "safit.txt", "r" );
    if( SAIfit == NULL ){
        other_mistake(1); //Can't open file
    } else{
        fscanf(SAIfit, "%lf", &sa_fitness[sa_innerit]);
        fclose(SAIfit);
    }
******************************************************************************
**************Compare fitness**************
******************************************************************************
sa_delta_fitness = sa_fitness[sa_innerit] - sa_fitness[(sa_innerit-1)];
    if(sa_delta_fitness <= 0.0){
        sa_Metropolis_criterion = 1.0;
    } else if(sa_delta_fitness > 0.0){
        sa_Metropolis_criterion = exp((-1.0)*sa_delta_fitness)/sa_current_temp;
    } else{
        mistake();
    }
******************************************************************************
**********Accept or Reject Using Metropolis Criterion**********
******************************************************************************
sa_rn3=RandomDouble(0.0, 1.0);
    if(sa_rn3 <= sa_Metropolis_criterion){
        sa_no_accepted = sa_no_accepted + 1;
    } else if(sa_rn3 > sa_Metropolis_criterion){
        for(sa_i = 0; sa_i < sa_L; sa_i++){
            sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
            sa_fitness[sa_innerit] = sa_fitness[(sa_innerit-1)];
        }
    }
******************************************************************************
**********Check Convergence at Current Temperature**********
******************************************************************************
if(sa_innerit < (sa_no_before_decrease - 1)){
    else if(sa_innerit == (sa_no_before_decrease - 1)){
        sa_min_population[sa_itno] = sa_fitness[sa_innerit];
    if(sa_itno == 0){
        intermin = fopen( "sainmin.txt", "w" );
        if( intermin == NULL ){
            other_mistake(1); //Can't open file
        } else{
            //Intermin file opened
        }
    } else{
fprintf (intermin, " %s %d %s %d", "No_before_decrease", sa_no_before_decrease, "No_temp_decrease", sa_no_temp_decrease);
fprintf (intermin, " %s %f %s %f", "Initial_temp", sa_initial_temp, "Final_temp", sa_final_temp);
fprintf (intermin, " %s %f %s %f", "Cooling_factor", sa_cooling_factor, "Acceptance_elasticity", sa_acceptance_elasticity);
for (sa_i=0; sa_i < sa_L; sa_i++){
    fprintf (intermin, " %d", {sa_i+1});
}
fprintf (intermin, " %s %s %s
", "Objective_Function_Value", "No_accepted", *increase_decrease);
fprintf (intermin, "%d ", {sa_itno + 1});
for (sa_i=0; sa_i < sa_L; sa_i++){
    fprintf (intermin, "%e ", sa_population[sa_i][sa_innerit]);
}
fprintf (intermin, "%e ", sa_fitness[sa_innerit]);
fprintf (intermin, "%d ", sa_no_accepted);
fprintf (intermin, "%d
", i);
fclose (intermin);
}
else if (sa_itno>0){
    intermin = fopen( "sainmin.txt", "a+" );
    if ( intermin == NULL ){
        other_mistake(1); //Can't open file
    }
    else{
        fprintf (intermin, "%d ", {sa_itno + 1});
        for (sa_i=0; sa_i < sa_L; sa_i++){
            fprintf (intermin, "%e ", sa_population[sa_i][sa_innerit]);
        }
        fprintf (intermin, "%e ", sa_fitness[sa_innerit]);
        fprintf (intermin, "%d ", sa_no_accepted);
        fclose (intermin);
    }
    else{
        mistake();
    }
    sa_itno=sa_itno+1;
    break;
}
else{
    mistake();
}
}//end while loop for current generation at increasing temperature

/***************************************************************
************Increase or Decrease Temperature?****************
***************************************************************
for (sa_i=0; sa_i < sa_L; sa_i++){
    sa_population[sa_i][0] = sa_population[sa_i][sa_innerit];
} sa_fitness[0] = sa_fitness[sa_innerit];
sa_innerit = 0;
if ( (double{sa_no_accepted})/(double{sa_no_before_decrease})<PERCENT_D_T ){
    sa_current_temp = 2.0*sa_current_temp;
} else if ( (double{sa_no_accepted})/(double{sa_no_before_decrease})>=PERCENT_D_T ){
    break;//start decreasing temperature
} else{
    mistake();
}
}//end while loop for keep raising temperature until stopping criteria met

/***************************************************************
****************Decrease Temperature***************************
***************************************************************
while(1);//keep decreasing temperature until stopping criteria met
sa_current_temp = sa_current_temp*sa_cooling_factor;
printf("\ncurrenttemp = %f", sa_current_temp);
sa_innerit = 0;
sa_no_accepted = 0;
while(1);//current generation at decreasing temperature
sa_innerit = sa_innerit + 1;
if(sa_per_type == 1){
    for(sa_i = 0; sa_i < sa_L; sa_i++){
        /*A random value is calculated to determine whether the mutation occurs for that gene.*/
        sa_rn3 = RandomDouble(0.0, 1.0);
        /*If the value generated is less than or equal to the user defined probability of mutation, then mutation occurs as below.*/
        if (sa_rn3 <= sa_probmut){
            /*If the mutation occurs, the new random value generated is within the specified limits of this variable.*/
            sa_rn4 = RandomDouble(sa_LLgene[sa_i], sa_ULgene[sa_i]);
            sa_population[sa_i][sa_innerit] = sa_rn4;
        }
        else if (sa_rn3 > sa_probmut){
            sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
        }
        else{
            mistake();
        }
    }
} //end if sa_per_type is 1

else if(sa_per_type == 2){
    sa_rn1 = RandomInt(0, (sa_L - 1));
    for(sa_i = 0; sa_i < sa_rn1; sa_i++){
        sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
    }
    sa_i = sa_rn1;
    sa_rn4 = RandomDouble(sa_LLgene[sa_i], sa_ULgene[sa_i]);
    sa_population[sa_i][sa_innerit] = sa_rn4;
    for(sa_i = (sa_rn1 + 1); sa_i < sa_L; sa_i++){
        sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
    }
} //end if sa_per_type is 2

else if(sa_per_type == 3){
} //end if sa_per_type is 3

else if(sa_per_type == 4){
} //end if sa_per_type is 4

else{
    mistake();
} //end else mistake if sa_per_type is not valid

calc_fitness(sa_population, sa_L, sa_m, 3, sa_innerit, sa_check, sa_qx, sa_qy, sa_flowx, sa_flowy);
SAfit = fopen("safit.txt", "r");
if (SAfit == NULL) {
    other_mistake(1); //Can't open file
} else {
    fscanf(SAfit, "%lf", &sa_fitness[sa_innerit]);
    fclose(SAfit);
}

/******************************************************************************
***************Compare fitness*********************************
*******************************************************************************/

sa_delta_fitness = sa_fitness[sa_innerit] - sa_fitness[(sa_innerit-1)];

if(sa_delta_fitness <= 0.0) {
    sa_Metropolis_criterion = 1.0;
} else if(sa_delta_fitness > 0.0) {
    sa_Metropolis_criterion = exp((-1.0)*sa_delta_fitness)/sa_current_temp;
} else {
    mistake();
}

/******************************************************************************
***********Accept or Reject Using Metropolis Criterion**********************
*******************************************************************************/

sa_rn3 = RandomDouble(0.0, 1.0);

if(sa_rn3 <= sa_Metropolis_criterion) {
    sa_no_accepted = sa_no_accepted + 1;
} else if(sa_rn3 > sa_Metropolis_criterion) {
    for(sa_i = 0; sa_i < sa_L; sa_i++) {  
        sa_population[sa_i][sa_innerit] = sa_population[sa_i][(sa_innerit-1)];
        sa_fitness[sa_innerit] = sa_fitness[(sa_innerit-1)];
    }
}

/******************************************************************************
***************Check Convergence at Current Temperature**********************
*******************************************************************************/

if(sa_innerit<(sa_no_before_decrease -1)) {
} else if(sa_innerit == (sa_no_before_decrease - 1)) {
    sa_min_population[sa_itno] = sa_fitness[sa_innerit];
    break;
} else {
    mistake();
}
}

/******************************************************************************
*****************Check Convergence Overall*******************************
*******************************************************************************/

if(((sa_min_population[(sa_itno-1)] - sa_min_population[sa_itno])<0.0001) &&
    ((sa_min_population[(sa_itno-1)] - sa_min_population[sa_itno]) > (-0.0001))) {
    sa_overall_convergence = sa_overall_convergence + 1;
    printf("\noverall_convergence=%d", sa_overall_convergence);
    printf("\nfit=%e", sa_min_population[sa_itno]);
} else if(((sa_min_population[(sa_itno-1)] - sa_min_population[sa_itno])>=0.0001) ||
    ((sa_min_population[(sa_itno-1)] - sa_min_population[sa_itno])<=(-0.0001))) {
    sa_overall_convergence = 0;
} else {
    mistake();
}
if((sa_overall_convergence >= sa_no_temp_decrease) || (sa_current_temp <= sa_final_temp)){
  output = fopen( "saallmin.txt", "w+" );//May want to print different things to output file
  if( output == NULL ){
    other_mistake(1);  //Can't open file
  }
  fprintf (output, " %s %d %s %d", "No_before_decrease", sa_no_before_decrease,
"No_temp_decrease", sa_no_temp_decrease);
  fprintf (output, " %s %f %s %f", "Initial_temp", sa_initial_temp, "Final_temp", sa_final_temp);
  fprintf (output, " %s %f %s %f
", "Cooling_factor", sa_cooling_factor, "Acceptance_elasticity",
sa_acceptance_elasticity);
  fprintf (output, " %s", "Generation");
  for(sa_i=0; sa_i < sa_L; sa_i++){
    fprintf (output, " %d", (sa_i+1));
  }
  fprintf (output, " %s %s %s
", "Objective_Function_Value", "No_accepted", "increase_decrease");
  intermin = fopen( "sainmin.txt", "r" );
  if( intermin == NULL ){  //Can't open file
    other_mistake(1);  //Can't open file
  }
  else{
    fscanf (intermin, "%s", &sa_temps1);//no_before_decrease
    fscanf (intermin, "%d", &sa_tempd1);//no_before_decrease
    fscanf (intermin, "%s", &sa_temps1);//no_temp_decrease
    fscanf (intermin, "%d", &sa_tempd1);//no_temp_decrease
    fscanf (intermin, "%s", &sa_temps1);//Initial_temp
    fscanf (intermin, "%d", &sa_tempd1);//Initial_temp
    fscanf (intermin, "%lf", &sa_temps1);//initial_temp
    fscanf (intermin, "%lf", &sa_tempd1);//initial_temp
    fscanf (intermin, "%s", &sa_temps1);//final_temp
    fscanf (intermin, "%lf", &sa_tempd1);//final_temp
    fscanf (intermin, "%s", &sa_temps1);//cooling_factor
    fscanf (intermin, "%lf", &sa_tempd1);//cooling_factor
    fscanf (intermin, "%s", &sa_temps1);//acceptance_elasticity
    fscanf (intermin, "%lf", &sa_tempd1);//acceptance_elasticity
    fscanf (intermin, "%s", &sa_temps1);//generation
    for(sa_i=0; sa_i < sa_L; sa_i++){
      fscanf (intermin, "%d", &sa_tempd1);//variable number
      fscanf(intermin, "%e", &sa_temps1);//objective function value
      fscanf(intermin, "%s", &sa_temps1);//no_accepted
      fscanf(intermin, "%s", &sa_temps1);//increase_decrease
      for(sa_k=0; sa_k < (sa_itno+1); sa_k++){
        if(sa_i==0){
          fprintf( output, " %d", (sa_k+1));
        }
        fscanf(intermin, "%e", &sa_temps1);//variable number
        if(sa_i==sa_L-1){
          fscanf(intermin, "%e", &sa_temps1);//objective function value
          fprintf( output, " %e", sa_temps1);
        }
        fprintf(output, " %e", sa_temps1);
      }
      fscanf(intermin, "%d", &sa_temps1);//generation no
      fclose(intermin);
      fclose(output);
    }
    }/*Check Overall Fittest Member***********/
    sa_ov_minfit_no = 0;
    sa_ov_minfit = sa_min_population[0];
    for(sa_i=1; sa_i < (sa_itno + 1); sa_i++){
      if(sa_min_population[sa_i] < sa_ov_minfit){
        sa_ov_minfit_no = sa_i;
        sa_ov_minfit = sa_min_population[sa_i];
      }
    }
    }/*Write best fitness combination files***********/
    calc_fitness(sa_population, sa_L, sa_m, 2, sa_ov_minfit_no, sa_check, sa_qx, sa_qy, sa_flowx, sa_flowy);
    break;
}
else if(sa_overall_convergence < sa_no_temp_decrease){
    sa_itno = sa_itno + 1;
    for(sa_i=0; sa_i < sa_L; sa_i++){
        sa_population[sa_i][0] = sa_population[sa_i][sa_innerit];
    }
    sa_fitness[0] = sa_fitness[sa_innerit];
} //end if not converged statement
else{
    mistake();
} //end while loop for keep decreasing temperature until stopping criteria met 
} //end SA function

/*******************************************************************************
************************Welcome Message Function**************************
******************************************************************************/

void welcome_message(void){
    int wm_check1;
    char wm_line[100];
    printf("Welcome to Jo's trial Program.");
    printf("\n\nThis is an optimisation program based on entropy principles.");
    printf("\n\nThe user has the flexibility of using default settings, or entering system specific settings that enable sensitivity analysis to be performed.");
    printf("\nNow, answer a few simple questions, sit back and sip a cup of tea while the GA performs its job.");
    printf("\n\nPlease press any key to continue (or ctrl c to quit)): ");
    (void) fgets(wm_line, sizeof(wm_line), stdin);
    (void) sscanf(wm_line, "%c", &wm_check1);
    return;
}

/*******************************************************************************
************************User Interface Function***************************
******************************************************************************/

void choose_values(void){
    int cv_check1;
    char cv_line[100];
    cv_check1 = 0;
    while(1){
        printf("\n\n\nTo check or change any default values for the items below, enter the corresponding integer number:");
        printf("\n1. GA Parameters (crossover type/probability etc)\n2. The dimensions of the required grid\n3. The type of each grid point\n4. Optimisation type (eg RGA, BGA, SA)\n5. Discretisation interval\n6. Maximum and minimum depths\n7. Number of loops\n8. To run the GA press 9\n\nTo run the GA press 9\n\nPlease enter your choice: ");
        (void) fgets(cv_line, sizeof(cv_line), stdin);
        (void) sscanf(cv_line, "%d", &cv_check1);
        if(cv_check1 == 1){
            initial_defaults(cv_check1);
            operation_file_write(1);
        } if(cv_check1 == 2){
            initial_defaults(cv_check1);
            set_grid_size(1);
        } if(cv_check1 == 3){
            initial_defaults(cv_check1);
            set_grid_type(1);
        } if(cv_check1 == 4){
            initial_defaults(cv_check1);
            optimisation_type(1);
        } if(cv_check1 == 5){
            initial_defaults(cv_check1);
            discret_int(1);
        }
    }
}
Appendix C  Morphology in Coastal and River Environments

// if(cv_check1 == 6){
//   initial_defaults(cv_check1);
// }
// if(cv_check1 == 7){
//   initial_defaults(cv_check1);
// }
if(cv_check1 == 8){
  initial_defaults(cv_check1);
  loop_set(1);
}
else if(cv_check1 == 9){
  break;
} else{
  other_mistake(6);
}
}

/**************************************************************************
**************************Set Grid Size Function**************************
**************************************************************************/

void set_grid_size(int sgs_check1){
  char sgs_temps[100];
  int sgs_gsx;//number of points in x direction
  int sgs_gsy;//number of points in y direction
  sgs_gsx = GSX;//default number of points in x direction
  sgs_gsy = GSY;//default number of points in y direction
  if( small_arcef == NULL ){  //Can't open file
    other_mistake(1);
  } else{
    fscanf(small_arcef, "%s", &sgs_temps);
    fscanf(small_arcef, "%s", &sgs_temps);
    fscanf(small_arcef, "%s", &sgs_temps);
    fscanf(small_arcef, "%d", &sgs_gsx);
    fscanf(small_arcef, "%d", &sgs_gsy);
    fclose(small_arcef);
  }
  grid_sizef = fopen ( "g_size.txt", "w+" );
  if( grid_sizef == NULL ){  //Can't open file
    other_mistake(1);
  } else{
    fprintf(grid_sizef, "%d", sgs_gsx);
    fprintf(grid_sizef, "\n%"d", sgs_gsy);
    fprintf(grid_sizef, "\n%s", "x_value" );
    fprintf(grid_sizef, "\n%s", "y_value" );
    fclose(grid_sizef);
  }
  return;
}

/**************************************************************************
**************************Set Grid Size Function**************************
**************************************************************************/

void bwset_grid_size(int sgs_check1){
  char bsgs_temps[100];
  int bsgs_gsx;//number of points in x direction
  int bsgs_gsy;//number of points in y direction
  bsgs_gsx = GSX;//default number of points in x direction
  bsgs_gsy = GSY;//default number of points in y direction
  small_arcef = fopen ( "detach.adi", "r" );
  if( small_arcef == NULL ){  //Can't open file
    other_mistake(1);
  } else{
    fscanf(small_arcef, "%s", &bsgs_temps);//detached
    fscanf(small_arcef, "%s", &bsgs_temps);//breakwater
    fscanf(small_arcef, "%d", &bsgs_gsx);
    fscanf(small_arcef, "%d", &bsgs_gsy);
    fclose(small_arcef);
  }
}
grid_sizef = fopen ("g_size.txt", "w+");
if( grid_sizef == NULL ){
    other_mistake(1);  //Can't open file
} else{
    fprintf(grid_sizef, "%d", bsgs_gsx);
    fprintf(grid_sizef, "\n%d", bsgs_gsy);
    fprintf(grid_sizef, "\n%s", "x_value");
    fprintf(grid_sizef, "\n%s", "y_value");
    fclose(grid_sizef);
}
return;
}

/**************************************************************************
**************************Set Grid Type Function**************************
**************************************************************************/
void set_grid_type(int sgt_check1){
    int sgt_check4;
    char sgt_line[100];
    char sgt_temps[100];
    int sgt_tempd;
    double sgt_templf;
    int sgt_gsx; //number of points in x direction
    int sgt_gsy; //number of points in y direction
    int sgt_i;
    int sgt_j;
    int sgt_k;
    int (*sgt_type)[BB];//type of point
    double (*sgt_v)[BB];//actual v values
    double (*sgt_u)[BB];//actual u values
    double (*sgt_n)[BB];//actual n values
    int *sgt_bits;//the type of each of the optimising values
    int sgt_countv;//count of v variables
    int sgt_countu;//count of u variables
    int sgt_countn;//count of n variables
    int sgt_count_tot;//count of n variables
    sgt_tempd = 0;
    sgt_templf = 0.0;
    /*Allocation of dynamic memory for arrays*/
    if ((sgt_type = new int[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_v = new double[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_u = new double[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_n = new double[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((sgt_bits = new int[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    for(sgt_i=0; sgt_i < AA; sgt_i++)
        for(sgt_j=0; sgt_j < BB; sgt_j++){
            sgt_type[sgt_i][sgt_j] = 0;
            sgt_v[sgt_i][sgt_j] = 0.0;
            sgt_u[sgt_i][sgt_j] = 0.0;
            sgt_n[sgt_i][sgt_j] = 0.0;
        }
    for(sgt_i=0; sgt_i < a; sgt_i++)
        sgt_bits[sgt_i] = 0;
    sgt_countv = 0;
    sgt_countu = 0;
    sgt_countn = 0;
    sgt_count_tot = 0;
    sgt_check4 = 1;
grid_size2f = fopen("g_size.txt", "r");
if(grid_size2f == NULL)
    other_mistake(1); //Can't open file
else{
    fscanf(grid_size2f, "%d", &sgt_gsx);//reads in x dimension
    fscanf(grid_size2f, "%d", &sgt_gsy);//reads in y dimension
    fclose(grid_size2f);
}

small_arce2f = fopen("inletevt", "r");
if(small_arce2f == NULL)
    other_mistake(1); //Can't open file
else{
    fscanf(small_arce2f, "%s", &sgt_temps);//Small
    fscanf(small_arce2f, "%s", &sgt_temps);//ARC
    fscanf(small_arce2f, "%d", &sgt_temps);//38 - no of x points
    fscanf(small_arce2f, "%d", &sgt_temps);//12 - no of y points
    fscanf(small_arce2f, "%d", &sgt_tempd);//1
    fscanf(small_arce2f, "%d", &sgt_tempd);//1
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_templf);//0.10
    fscanf(small_arce2f, "\lf", &sgt_temps);//full
    fscanf(small_arce2f, "%d", &sgt_temps);//1
    fscanf(small_arce2f, "%d", &sgt_temps);//1
    for(sgt_j=(sgt_gsy-1); sgt_j > (-1); sgt_j--){
        for(sgt_i=0; sgt_i < sgt_gsx; sgt_i++)
            if(sgt_temps[sgt_i] == '0')
                sgt_tempd = 0;
            else if(sgt_temps[sgt_i] == '1')
                sgt_tempd = 1;
            else if(sgt_temps[sgt_i] == '2')
                sgt_tempd = 2;
            else if(sgt_temps[sgt_i] == '4')
                sgt_tempd = 4;
            else if(sgt_temps[sgt_i] == '6')
                sgt_tempd = 6;
            else if(sgt_temps[sgt_i] == '8')
                sgt_tempd = 8;
            else
                mistake();
        sgt_type[sgt_i][sgt_j] = sgt_tempd;
    }
    fclose(small_arce2f);
}

grid_typef = fopen("g_type.txt", "w+");
if(grid_typef == NULL)
    other_mistake(1); //Can't open file
else{
    for(sgt_j=(sgt_gsy-1); sgt_j > (-1); sgt_j--){
        for(sgt_i=0; sgt_i < sgt_gsx; sgt_i++)
            fprintf(grid_typef, "%d ", sgt_type[sgt_i][sgt_j]);
        fprintf(grid_typef, "\n");
    }
}
fprintf(grid_typef, "\n%s", "land_0");
fprintf(grid_typef, "\n%s", "normal_water_1");
fprintf(grid_typef, "\n%s", "u=0_2");
fprintf(grid_typef, "\n%s", "v=0_4");
fprintf(grid_typef, "\n%s", "u=0_v=0_6");
fprintf(grid_typef, "\n%s", "specified_n_8");
fprintf(grid_typef, "\n%s", "specified_u_16");
fprintf(grid_typef, "\n%s", "specified_v_32");
fprintf(grid_typef, "\n%s", "combination_sum");
fclose(grid_typef);
}

sgt_i = 0;
sgt_j = (sgt_gsy - 1);
sgt_k = 0;
while(1){
    if(sgt_type[sgt_i][sgt_j] == 0){
        sgt_v[sgt_i][sgt_j] = 0.0;
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_n[sgt_i][sgt_j] = 0.0;
    }
    else if(sgt_type[sgt_i][sgt_j] == 1){
        sgt_v[sgt_i][sgt_j] = 2.0;
        sgt_countv = sgt_countv + 1;
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 1;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 2){
        sgt_v[sgt_i][sgt_j] = 2.0;
        sgt_countv = sgt_countv + 1;
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_countu = sgt_countu + 1;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 2;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 4){
        sgt_v[sgt_i][sgt_j] = 0.0;
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 4;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 6){
        sgt_v[sgt_i][sgt_j] = 0.0;
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 6;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 8){
        sgt_v[sgt_i][sgt_j] = 2.0;
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        if(sgt_check4 == 1){
            sgt_n[sgt_i][sgt_j] = SPECN;
        } else{
            printf("Enter n for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
        }
        sgt_bits[sgt_k] = 8;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 10){
        sgt_v[sgt_i][sgt_j] = 2.0;
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        if(sgt_check4 == 1){
            sgt_n[sgt_i][sgt_j] = SPECN;
        } else{
            printf("Enter n for point %d,%d: ", sgt_i, sgt_j);
        }
    }
    else{
        printf("\nEnter n for point %d,%d: ", sgt_i, sgt_j);
    }
(void) fgets(sgt_line, sizeof(sgt_line), stdin);
(void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);

sgt_bits[sgt_k] = 10;
sgt_k = sgt_k + 1;

else if(sgt_type[sgt_i][sgt_j] == 12){
    sgt_v[sgt_i][sgt_j] = 0.0;
    sgt_u[sgt_i][sgt_j] = 2.0;
    sgt_countu = sgt_countu + 1;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 12;
    sgt_k = sgt_k + 1;
}

else if(sgt_type[sgt_i][sgt_j] == 14){
    sgt_v[sgt_i][sgt_j] = 0.0;
    sgt_u[sgt_i][sgt_j] = 0.0;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 14;
    sgt_k = sgt_k + 1;
}

else if(sgt_type[sgt_i][sgt_j] == 16){
    sgt_v[sgt_i][sgt_j] = 2.0;
    sgt_countv = sgt_countv + 1;
    if(sgt_check4 == 1){
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 16;
    sgt_k = sgt_k + 1;
}

else if(sgt_type[sgt_i][sgt_j] == 20){
    sgt_v[sgt_i][sgt_j] = 0.0;
    if(sgt_check4 == 1){
        sgt_u[sgt_i][sgt_j] = SPECU;
    } else{
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
    }
    sgt_n[sgt_i][sgt_j] = 2.0;
    sgt_countn = sgt_countn + 1;
    sgt_bits[sgt_k] = 20;
    sgt_k = sgt_k + 1;
}

else if(sgt_type[sgt_i][sgt_j] == 24){
    sgt_v[sgt_i][sgt_j] = 2.0;
    sgt_countv = sgt_countv + 1;
    if(sgt_check4 == 1){
        sgt_u[sgt_i][sgt_j] = SPECU;
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else{
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
    sgt_bits[sgt_k] = 24;
    sgt_k = sgt_k + 1;
}

else if(sgt_type[sgt_i][sgt_j] == 28){
    sgt_v[sgt_i][sgt_j] = 0.0;
    if(sgt_check4 == 1){
        sgt_u[sgt_i][sgt_j] = SPECU;
    }
sgt_n[sgt_i][sgt_j] = SPECN;
else{
    printf("Enter u for point %d,%d: ", sgt_i, sgt_j);
    (void) fgets(sgt_line, sizeof(sgt_line), stdin);
    (void) scanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
    printf("Enter n for point %d,%d: ", sgt_i, sgt_j);
    (void) fgets(sgt_line, sizeof(sgt_line), stdin);
    (void) scanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
}
}
else if(sgt_type[sgt_i][sgt_j] == 32){
    if(sgt_check4 == 1){
        sgt_v[sgt_i][sgt_j] = SPECV;
    }
    else{
        printf("Enter v for point %d,%d: ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) scanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countv = sgt_countv + 1;
        sgt_n[sgt_i][sgt_j] = 2.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 32;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 34){
        if(sgt_check4 == 1){
            sgt_v[sgt_i][sgt_j] = SPECV;
        }
        else{
            printf("Enter v for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
            printf("Enter n for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
        }
        sgt_u[sgt_i][sgt_j] = 0.0;
        sgt_countn = sgt_countn + 1;
        sgt_bits[sgt_k] = 34;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 40){
        if(sgt_check4 == 1){
            sgt_v[sgt_i][sgt_j] = SPECV;
            sgt_n[sgt_i][sgt_j] = SPECN;
        }
        else{
            printf("Enter v for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
            printf("Enter u for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
        }
        sgt_u[sgt_i][sgt_j] = 2.0;
        sgt_countu = sgt_countu + 1;
        sgt_bits[sgt_k] = 40;
        sgt_k = sgt_k + 1;
    }
    else if(sgt_type[sgt_i][sgt_j] == 42){
        if(sgt_check4 == 1){
            sgt_v[sgt_i][sgt_j] = SPECV;
            sgt_n[sgt_i][sgt_j] = SPECN;
        }
        else{
            printf("Enter v for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
            printf("Enter n for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
        }
        sgt_u[sgt_i][sgt_j] = 0.0;
    }
    else if(sgt_type[sgt_i][sgt_j] == 48){
        if(sgt_check4 == 1){
            sgt_v[sgt_i][sgt_j] = SPECV;
            sgt_u[sgt_i][sgt_j] = SPECU;
        }
        else{
            printf("Enter v for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
            (void) scanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
            printf("Enter u for point %d,%d: ", sgt_i, sgt_j);
            (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        }
    }
}
Appendix C Morphology in Coastal and River Environments

```c
(void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
}
sgt_n[sgt_i][sgt_j] = 2.0;
sgt_countn = sgt_countn + 1;
sgt_bits[sgt_k] = 48;
sgt_k = sgt_k + 1;
} else if (sgt_type[sgt_i][sgt_j] == 56) {
    if (sgt_check4 == 1) {
        sgt_v[sgt_i][sgt_j] = SPECV;
        sgt_u[sgt_i][sgt_j] = SPECU;
        sgt_n[sgt_i][sgt_j] = SPECN;
    } else {
        printf("Enter v for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_v[sgt_i][sgt_j]);
        printf("Enter u for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_u[sgt_i][sgt_j]);
        printf("Enter n for point %d,%d:  ", sgt_i, sgt_j);
        (void) fgets(sgt_line, sizeof(sgt_line), stdin);
        (void) sscanf(sgt_line, "%lf", &sgt_n[sgt_i][sgt_j]);
    }
} else {
    mistake();
}
sgt_i = sgt_i + 1;
if (sgt_i == sgt_gsx) {
    sgt_i = 0;
    sgt_j = sgt_j - 1;
} if (sgt_j == 0) {
    break;
}
}
grid_vf = fopen("g_v.txt", "w+");
grid_uf = fopen("g_u.txt", "w+");
grid_nf = fopen("g_n.txt", "w+");
if (grid_vf == NULL) {
    other_mistake(1);  //Can't open file
    printf("grid_v");
} else {
    for (sgt_j = sgt_gsy - 1; sgt_j > (-1); sgt_j--) {
        for (sgt_i = 0; sgt_i < sgt_gsx; sgt_i++) {
            fprintf(grid_vf, "%f ", sgt_v[sgt_i][sgt_j]);
            fprintf(grid_uf, "%f ", sgt_u[sgt_i][sgt_j]);
            fprintf(grid_nf, "%f ", sgt_n[sgt_i][sgt_j]);
        }
        fprintf(grid_vf, "\n");
        fprintf(grid_uf, "\n");
        fprintf(grid_nf, "\n");
    }
    fprintf(grid_vf, "\n\$\$", "velocity_v");
    fprintf(grid_uf, "\n\$\$", "velocity_u");
    fprintf(grid_nf, "\n\$\$", "water_elevation_n");
    fclose(grid_vf);
    fclose(grid_uf);
    fclose(grid_nf);
}
grid_bitsf = fopen("g_bits.txt", "w+");
if (grid_bitsf == NULL) {
    other_mistake(1);  //Can't open file
    printf("grid_bits");
} else {
    for (sgt_i = 0; sgt_i < sgt_k; sgt_i++) {
        fprintf(grid_bitsf, "%d ", sgt_bits[sgt_i]);
    }
    fprintf(grid_vf, "\n\$\$", "optimising_bits");
    fclose(grid_bitsf);
}
```
Morphology in Coastal and River Environments

Appendix C

sgt_count_tot = sgt_countv + sgt_countu + sgt_countn;
water_countf = fopen ("w_count.txt", "w+");
if( water_countf == NULL ){  
other_mistake(1);  //Can't open file
} else{
  fprintf(water_countf, "%d ", sgt_countn);//writes number of points to optimise
  fprintf(water_countf, "%d ", sgt_countv);//writes number of points to optimise
  fprintf(water_countf, "%d ", sgt_countu);//writes number of points to optimise
  fprintf(water_countf, "%d ", sgt_countn);//writes number of points to optimise
  fprintf(water_countf, "\n%s", "count_of_optimising_points");
  fprintf(water_countf, "\n%s", "count_of_v_optimising_points");
  fprintf(water_countf, "\n%s", "count_of_u_optimising_points");
  fprintf(water_countf, "\n%s", "count_of_n_optimising_points");
fclose(water_countf);  
} /*Deletion of dynamic memory for arrays*/
delete [] sgt_type;
delete [] sgt_v;
delete [] sgt_u;
delete [] sgt_n;
delete [] sgt_bits;
return;
} /***************************************************************************
**************************Plot Still Water Level**************************
**************************************************************************/

void still_water_level(void){
  char swl_temps[100];
  int swl_tempd;
  double swltemplf;
  int swl_gsx;//number of points in x direction
  int swl_gsy;//number of points in y direction
  int swl_i;
  int swl_j;
  int swl_k;
  double (*swl_waterel)[BB];//water elevations
  int (*swl_type)[BB];//type of point
  double *swl_wbits;//the type of each of the optimising values
  double swl_slope_inc;//increment that depth increases as move to next point from shore
  double swl_stop_el;//final depth when end of slope reached
  double swl_start_el;//initial depth at shoreline
  int swl_stop_no;//number of grid points that beach is sloped
  /*Allocation of dynamic memory for arrays*/
  if ((swl_waterel = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
  }
  if ((swl_type = new int[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
  }
  if ((swl_wbits = new double[a]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
  }
  for(swl_i=0; swl_i < AA; swl_i++){
    for(swl_j=0; swl_j < BB; swl_j++){
      swl_waterel[swl_i][swl_j] = 0.0;
      swl_type[swl_i][swl_j] = 0;
    }
  }
  swl_tempd = 0;
  swl_template = 0.0;
  swl_slope_inc = 0.0;//increment that depth increases as move to next point from shore
  swl_stop_el = 0.0;//final depth when end of slope reached
  swl_start_el = 0.0;//initial depth at shoreline
  swl_stop_no = 0;//number of grid points that beach is sloped
  grid_size2f = fopen ("g_size.txt", "r");
Appendix C Morphology in Coastal and River Environments

```c
if (grid_size2f == NULL) {
    other_mistake(1); // Can't open file
} else{
    fscanf(grid_size2f, "%d", &swl_gsx); // reads in x dimension
    fscanf(grid_size2f, "%d", &swl_gsy); // reads in y dimension
    fclose(grid_size2f);
}
Small_arce2f = fopen("Detach.adw", "r");
if (small_arce2f == NULL) {
    other_mistake(1); // Can't open file
} else{
    fscanf(small_arce2f, "%s", &swl_temps); // Detached
    fscanf(small_arce2f, "%s", &swl_temps); // Breakwater
    fscanf(small_arce2f, "%d", &swl_temps); // 50 - no of x points
    fscanf(small_arce2f, "%d", &swl_temps); // 90 - no of y points
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.04 - DX
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.01 - DT
    fscanf(small_arce2f, "%d", &swl_temps); // 10 - point coordinate
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.05 -
    fscanf(small_arce2f, "%lf", &swl_temps); // 1.16 -
    fscanf(small_arce2f, "%lf", &swl_temps); // 180.0 -
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.001 -
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.04 -
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.88 -
    fscanf(small_arce2f, "%s", &swl_temps); // I
    while(swl_temps[0]!='e'){
        fscanf(small_arce2f, "%d", &swl_temps); // 20 - I1
        fscanf(small_arce2f, "%d", &swl_temps); // 23 - I2
        fscanf(small_arce2f, "%d", &swl_temps); // 37 - I3
        fscanf(small_arce2f, "%d", &swl_temps); // 8 - I4
        fscanf(small_arce2f, "%s", &swl_temps); // I
    } // read in of slope, need to expand if more than one slope
    fscanf(small_arce2f, "%d", &swl_temps); // 99 - empty
    fscanf(small_arce2f, "%d", &swl_temps); // 99 - empty
    fscanf(small_arce2f, "%d", &swl_temps); // 99 - empty
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.02 - depth at start of slope
    fscanf(small_arce2f, "%lf", &swl_temps); // 0.2 - depth after slope
    fscanf(small_arce2f, "%d", &swl_temps); // 5 - point at which slope stops
    fclose(small_arce2f);
}
swl_slope_inc = (swl_stop_el - swl_start_el) / ((double)swl_stop_no) - 1.0;
for (swl_i = 0; swl_i < (swl_stop_no-1)/2; swl_i++){
    for (swl_j = 0; swl_j < swl_gsy; swl_j++){
        swl_waterel[swl_i][swl_j] = swl_start_el + swl_slope_inc * swl_i * X_STEP;
    }
}
for (swl_i = (swl_stop_no-1)/2; swl_i < swl_gsx; swl_i++){
    for (swl_j = 0; swl_j < swl_gsy; swl_j++){
        swl_waterel[swl_i][swl_j] = swl_stop_el;
    }
}
still_waterf = fopen("swater.txt", "w+");
if (still_waterf == NULL) {
    other_mistake(1); // Can't open file
} else{
    for (swl_j = (swl_gsy-1); swl_j > (-1); swl_j--){
        fprintf(still_waterf, "%e ", swl_waterel[swl_i][swl_j]);
    }
    fprintf(still_waterf, "\n");
    fclose(still_waterf);
}
grid_typef = fopen("g_type.txt", "r");
if (grid_typef == NULL) {
    other_mistake(1); // Can't open file
} else{
    for (swl_j = (swl_gsy-1); swl_j > (-1); swl_j--){
        for (swl_i = 0; swl_i < swl_gsx; swl_i++){
            fprintf(still_waterf, "%e ", swl_waterel[swl_i][swl_j]);
        }
        fprintf(still_waterf, "\n");
    }
    fclose(still_waterf);
}
```
```c
fscanf(grid_typef, "%d ", &swl_type[swl_i][swl_j]);
}
fclose(grid_typef);

swl_i = 0;
swl_j = 0;
swl_k = 0;

while(1){
  if(swl_type[swl_i][swl_j] == 0){
  }
  else if(swl_type[swl_i][swl_j] == 1){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 2){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 4){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 6){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 8){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 10){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 12){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 14){
  }
  else if(swl_type[swl_i][swl_j] == 16){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 20){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 24){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 28){
  }
  else if(swl_type[swl_i][swl_j] == 32){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 34){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 40){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 42){
  }
  else if(swl_type[swl_i][swl_j] == 48){
    swl_wbits[swl_k] = swl_waterel[swl_i][swl_j];
    swl_k = swl_k + 1;
  }
  else if(swl_type[swl_i][swl_j] == 56){
  }
  else{
    mistake();
  }
  swl_i = swl_i + 1;
  if(swl_i == swl_gsx){
    swl_i = 0;
    swl_j = swl_j + 1;
  }
  if(swl_j == swl_gsy){
    } 
```
Appendix C  Morphology in Coastal and River Environments

break;
}
}

still_water2f = fopen("swater_bits.txt", "w+");
if( still_water2f == NULL ){
    other_mistake(1);  //Can't open file
    printf("nswater_bits");
} else{
    fprintf(still_water2f, "%d ", swl_k);
    for(swl_i=0; swl_i < swl_k; swl_i++){
        fprintf(still_water2f, "%e ", swl_wbits[swl_i]);
    }
    fprintf(still_water2f, 
            "water_depth_bits");
    fclose(still_water2f);
}
/*Deletion of dynamic memory for arrays*/
delete [] swl_waterel;
delete [] swl_type;
delete [] swl_wbits;
return;
}

/**************************************************************************
**************************Set Grid Type Function**************************
**************************************************************************/

void bwset_grid_type(int sgt_check1){
int bsgt_check4;
char bsgt_line[100];
char bsgt_temps[100];
int bsgt_tempd;
double bsgt_templf;
int bsgt_gsx; //number of points in x direction
int bsgt_gsy; //number of points in y direction
int bsgt_i;
int bsgt_j;
int bsgt_k;
int (*bsgt_type)[BB];//type of point
double (*bsgt_v)[BB];//actual v values
double (*bsgt_u)[BB];//actual u values
int (*bsgt_bits)[BB];//the type of each of the optimising values
int bsgt_countu;//count of u variables
int bsgt_countn;//count of n variables
int bsgt_count_tot;//count of n variables
int bsgt_I1;//used in reading in of breakwater position
int bsgt_I2;//used in reading in of breakwater position
int bsgt_I3;//used in reading in of breakwater position
int bsgt_I4;//used in reading in of breakwater position

bsgt_tempd = 0;
bsgt_templf = 0.0;
bsgt_I1 = 0;
bsgt_I2 = 0;
bsgt_I3 = 0;
bsgt_I4 = 0;
/*Allocation of dynamic memory for arrays*/
if ((bsgt_type = new int[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
} else{
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
} else{
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
} else{
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
} else{
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
} else{
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
for (bsgt_i=0; bsgt_i < AA; bsgt_i++){
    for (bsgt_j=0; bsgt_j < BB; bsgt_j++){
        bsgt_type[bsgt_i][bsgt_j] = 0;
        bsgt_v[bsgt_i][bsgt_j] = 0.0;
        bsgt_u[bsgt_i][bsgt_j] = 0.0;
        bsgt_n[bsgt_i][bsgt_j] = 0.0;
    }
}
for (bsgt_i=0; bsgt_i < a; bsgt_i++){
    bsgt_bits[bsgt_i] = 0;
}
bsgt_countv = 0;
bsgt_countu = 0;
bsgt_countn = 0;
bsgt_count_tot = 0;
bsgt_check4 = 1;
grid_size2f = fopen ( "g_size.txt", "r" );
if( grid_size2f == NULL ){
    other_mistake(1);  //Can't open file
} else{
    fscanf(grid_size2f, "%d", &bsgt_gsx);//reads in x dimension
    fscanf(grid_size2f, "%d", &bsgt_gsy);//reads in y dimension
    fclose(grid_size2f);
}
for (bsgt_j=0; bsgt_j < bsgt_gsy; bsgt_j++)//set all grid points to water
    for (bsgt_i=0; bsgt_i < bsgt_gsx; bsgt_i++){
        bsgt_type[bsgt_i][bsgt_j] = 1;
    }
small_arce2f = fopen ( "detach.adi", "r" );
if( small_arce2f == NULL ){
    other_mistake(1);  //Can't open file
} else{
    fscanf(small_arce2f, "%s", &bsgt_temps);//Detached
    fscanf(small_arce2f, "%s", &bsgt_temps);//Breakwater
    fscanf(small_arce2f, "%d", &bsgt_tempd);//30 - no of x points
    fscanf(small_arce2f, "%d", &bsgt_tempd);//60 - no of y points
    fscanf(small_arce2f, "%d", &bsgt_tempd);//5 - IXX
    fscanf(small_arce2f, "%d", &bsgt_tempd);//5 - IYY
    fscanf(small_arce2f, "%lf", &bsgt_templf);//5.0 - DX
    fscanf(small_arce2f, "%lf", &bsgt_templf);//5.0 - DY
    fscanf(small_arce2f, "%lf", &bsgt_templf);//2.0 - DT
    fscanf(small_arce2f, "%lf", &bsgt_templf);//600.0 - TLIM
    fscanf(small_arce2f, "%lf", &bsgt_templf);//100.0 - GRITS
    fscanf(small_arce2f, "%lf", &bsgt_templf);//0.010 - CF
    fscanf(small_arce2f, "%lf", &bsgt_templf);//0.005 - XH
    fscanf(small_arce2f, "%lf", &bsgt_templf);//1000.0 - RHO
    fscanf(small_arce2f, "%lf", &bsgt_templf);//6.60 - TP
    fscanf(small_arce2f, "%lf", &bsgt_templf);//1.0 - GHO
    fscanf(small_arce2f, "%s", &bsgt_temps);//I
    while(bsgt_temps[0]!="E"){
        fscanf(small_arce2f, "%d", &bsgt_I1);//20 - I1
        fscanf(small_arce2f, "%d", &bsgt_I2);//23 - I2
        fscanf(small_arce2f, "%d", &bsgt_I3);//37 - I3
        fscanf(small_arce2f, "%d", &bsgt_I4);//8 - I4
        for(bsgt_j = (bsgt_I2-1); bsgt_j < bsgt_I3; bsgt_j++){
            bsgt_type[bsgt_I1][bsgt_j] = bsgt_I4;
        }
        fscanf(small_arce2f, "%s", &bsgt_temps);//I
    }
    fclose(small_arce2f);
}
grid_typef = fopen ( "g_type.txt", "w+" );
if( grid_typef == NULL ){
    other_mistake(1);  //Can't open file
} else{
Appendix C Morphology in Coastal and River Environments

for(bsgt_j=(bsgt_gsy-1); bsgt_j > (-1); bsgt_j--){
    for(bsgt_i=0; bsgt_i < bsgt_gsx; bsgt_i++){
        fprintf(grid_typef, "%d", bsgt_type[bsgt_i][bsgt_j]);
    }
    fprintf(grid_typef, "\n"
};

fprintf(grid_typef, "\n\n\n%s", "land_0"
);
fprintf(grid_typef, "\n\n\n%s", "normal_water_1"
);
fprintf(grid_typef, "\n\n\n%s", "u=0_2"
);
fprintf(grid_typef, "\n\n\n%s", "v=0_4"
);
fprintf(grid_typef, "\n\n\n%s", "u=0_v=0_6"
);
fprintf(grid_typef, "\n\n\n%s", "specified_n_8"
);
fprintf(grid_typef, "\n\n\n%s", "specified_u_16"
);
fprintf(grid_typef, "\n\n\n%s", "specified_v_32"
);
fprintf(grid_typef, "\n\n\n%s", "combination_sum"
);
fclose(grid_typef);
}

bsgt_i = 0;
bsgt_k = 0;
bsgt_j = (bsgt_gsy-1);
while(1){
    if(bsgt_type[bsgt_i][bsgt_j] == 0){
        bsgt_v[bsgt_i][bsgt_j] = 0.0;
        bsgt_u[bsgt_i][bsgt_j] = 0.0;
        bsgt_n[bsgt_i][bsgt_j] = 0.0;
    }
    else if(bsgt_type[bsgt_i][bsgt_j] == 1){
        bsgt_v[bsgt_i][bsgt_j] = 2.0;
        bsgt_countv = bsgt_countv + 1;
        bsgt_u[bsgt_i][bsgt_j] = 2.0;
        bsgt_countu = bsgt_countu + 1;
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = bsgt_countn + 1;
        bsgt_bits[bsgt_k] = 1;
        bsgt_k = bsgt_k + 1;
    }
    else if(bsgt_type[bsgt_i][bsgt_j] == 2){
        bsgt_v[bsgt_i][bsgt_j] = 2.0;
        bsgt_countv = bsgt_countv + 1;
        bsgt_u[bsgt_i][bsgt_j] = 0.0;
        bsgt_countu = bsgt_countu + 1;
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = bsgt_countn + 1;
        bsgt_bits[bsgt_k] = 2;
        bsgt_k = bsgt_k + 1;
    }
    else if(bsgt_type[bsgt_i][bsgt_j] == 4){
        bsgt_v[bsgt_i][bsgt_j] = 0.0;
        bsgt_u[bsgt_i][bsgt_j] = 2.0;
        bsgt_countu = bsgt_countu + 1;
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = bsgt_countn + 1;
        bsgt_bits[bsgt_k] = 4;
        bsgt_k = bsgt_k + 1;
    }
    else if(bsgt_type[bsgt_i][bsgt_j] == 6){
        bsgt_v[bsgt_i][bsgt_j] = 0.0;
        bsgt_u[bsgt_i][bsgt_j] = 0.0;
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = bsgt_countn + 1;
        bsgt_bits[bsgt_k] = 6;
        bsgt_k = bsgt_k + 1;
    }
    else if(bsgt_type[bsgt_i][bsgt_j] == 8){
        bsgt_v[bsgt_i][bsgt_j] = 2.0;
        bsgt_countv = bsgt_countv + 1;
        bsgt_u[bsgt_i][bsgt_j] = 2.0;
        bsgt_countu = bsgt_countu + 1;
        if(bsgt_check4 == 1){
            bsgt_n[bsgt_i][bsgt_j] = SPECN;
        }
        else{
            printf("\nEnter n for point %d,%d:  ", bsgt_i, bsgt_j);
            (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
            (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
        }
        bsgt_bits[bsgt_k] = 8;
        bsgt_k = bsgt_k + 1;
    }
    else if(bsgt_type[bsgt_i][bsgt_j] == 10){
        bsgt_v[bsgt_i][bsgt_j] = 2.0;
        bsgt_countv = bsgt_countv + 1;
        bsgt_u[bsgt_i][bsgt_j] = 2.0;
        bsgt_countu = bsgt_countu + 1;
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = 2.0;
        bsgt_bits[bsgt_k] = 10;
        bsgt_k = bsgt_k + 1;
    }
}
bsgt_u[bsgt_i][bsgt_j] = 0.0;
if(bsgt_check4 == 1){
    bsgt_n[bsgt_i][bsgt_j] = SPECN;
} else{
    printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
    (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
    (void) sscanf(bsgt_line, "\lf", &bsgt_n[bsgt_i][bsgt_j]);
    bsgt_bits[bsgt_k] = 10;
    bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 12){
    bsgt_v[bsgt_i][bsgt_j] = 0.0;
    bsgt_u[bsgt_i][bsgt_j] = 2.0;
    bsgt_countu = bsgt_countu + 1;
    if(bsgt_check4 == 1){
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "\lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
    bsgt_bits[bsgt_k] = 12;
    bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 14){
    bsgt_v[bsgt_i][bsgt_j] = 0.0;
    if(bsgt_check4 == 1){
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    } else{
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "\lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
    bsgt_bits[bsgt_k] = 14;
    bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 16){
    bsgt_v[bsgt_i][bsgt_j] = 0.0;
    bsgt_countv = bsgt_countv + 1;
    if(bsgt_check4 == 1){
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
    } else{
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "\lf", &bsgt_u[bsgt_i][bsgt_j]);
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = bsgt_countn + 1;
        bsgt_bits[bsgt_k] = 16;
        bsgt_k = bsgt_k + 1;
    }
else if(bsgt_type[bsgt_i][bsgt_j] == 20){
    bsgt_v[bsgt_i][bsgt_j] = 0.0;
    if(bsgt_check4 == 1){
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
    } else{
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "\lf", &bsgt_u[bsgt_i][bsgt_j]);
        bsgt_n[bsgt_i][bsgt_j] = 2.0;
        bsgt_countn = bsgt_countn + 1;
        bsgt_bits[bsgt_k] = 20;
        bsgt_k = bsgt_k + 1;
    }
else if(bsgt_type[bsgt_i][bsgt_j] == 24){
    bsgt_v[bsgt_i][bsgt_j] = 2.0;
    bsgt_countv = bsgt_countv + 1;
    if(bsgt_check4 == 1){
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
    } else{
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "\lf", &bsgt_u[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "\lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
    bsgt_bits[bsgt_k] = 24;
bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 28){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECU;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 32){
    bsgt_u[bsgt_i][bsgt_j] = 2.0;
    bsgt_v[bsgt_i][bsgt_j] = SPECV;
    bsgt_countv = bsgt_countv + 1;
    bsgt_n[bsgt_i][bsgt_j] = 2.0;
    bsgt_countn = bsgt_countn + 1;
    bsgt_bits[bsgt_k] = 32;
    bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 34){
    bsgt_u[bsgt_i][bsgt_j] = 2.0;
    bsgt_v[bsgt_i][bsgt_j] = SPECV;
    bsgt_countv = bsgt_countv + 1;
    bsgt_n[bsgt_i][bsgt_j] = 2.0;
    bsgt_countn = bsgt_countn + 1;
    bsgt_bits[bsgt_k] = 34;
    bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 40){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 42){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 48){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 50){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 52){
    bsgt_k = bsgt_k + 1;
    bsgt_v[bsgt_i][bsgt_j] = SPECV;
    bsgt_u[bsgt_i][bsgt_j] = SPECU;
    bsgt_n[bsgt_i][bsgt_j] = SPECN;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 56){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 60){
    bsgt_k = bsgt_k + 1;
    bsgt_u[bsgt_i][bsgt_j] = 2.0;
    bsgt_v[bsgt_i][bsgt_j] = SPECV;
    bsgt_n[bsgt_i][bsgt_j] = SPECN;
    bsgt_bits[bsgt_k] = 60;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 62){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 64){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 68){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else if(bsgt_type[bsgt_i][bsgt_j] == 70){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
    }
    else{
        printf("Enter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("Enter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
    }
}
else{
    printf("\nEnter v for point %d,%d:  ", bsgt_i, bsgt_j);
    (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
    (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
    printf("\nEnter u for point %d,%d:  ", bsgt_i, bsgt_j);
    (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
    (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
    bsgt_n[bsgt_i][bsgt_j] = 2.0;
    bsgt_countn = bsgt_countn + 1;
    bsgt_bits[bsgt_k] = 48;
    bsgt_k = bsgt_k + 1;
}
else if(bsgt_type[bsgt_i][bsgt_j] == 56){
    if(bsgt_check4 == 1){
        bsgt_v[bsgt_i][bsgt_j] = SPECV;
        bsgt_u[bsgt_i][bsgt_j] = SPECU;
        bsgt_n[bsgt_i][bsgt_j] = SPECN;
    }
    else{
        printf("\nEnter v for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_v[bsgt_i][bsgt_j]);
        printf("\nEnter u for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_u[bsgt_i][bsgt_j]);
        printf("\nEnter n for point %d,%d:  ", bsgt_i, bsgt_j);
        (void) fgets(bsgt_line, sizeof(bsgt_line), stdin);
        (void) sscanf(bsgt_line, "%lf", &bsgt_n[bsgt_i][bsgt_j]);
    }
}
else{
    mistake();
}
bsgt_i = bsgt_i + 1;
if(bsgt_i == bsgt_gsx){
    bsgt_i = 0;
    bsgt_j = bsgt_j - 1;
} else if(bsgt_j == 0){
    break;
}
}
grid_vf = fopen( "g_v.txt", "w+" );
grid_uf = fopen( "g_u.txt", "w+" );
grid_nf = fopen( "g_n.txt", "w+" );
if( grid_vf == NULL ){ other_mistake(); //Can't open file
    printf("\ngrid_v
");
}
else{
    for(bsgt_j=(bsgt_gsy-1); bsgt_j > (-1); bsgt_j--){
        for(bsgt_i=0; bsgt_i < bsgt_gsx; bsgt_i++){
            fprintf(grid_vf, "%f ", bsgt_v[bsgt_i][bsgt_j]);
            fprintf(grid_uf, "%f ", bsgt_u[bsgt_i][bsgt_j]);
            fprintf(grid_nf, "%f ", bsgt_n[bsgt_i][bsgt_j]);
        }
        fprintf(grid_vf, "\n");
        fprintf(grid_uf, "\n");
        fprintf(grid_nf, "\n");
    }
    fprintf(grid_vf, "\n\nvelocity_v\n");
    fprintf(grid_uf, "\n\nvelocity_u\n");
    fprintf(grid_nf, "\n\nwater_elevation_n\n");
    fclose(grid_vf);
    fclose(grid_uf);
    fclose(grid_nf);
}
grid_bitsf = fopen( "g_bits.txt", "w+" );
if( grid_bitsf == NULL ){ other_mistake(); //Can't open file
    printf("\ngrid_bits
");
}
else{
    fprintf(grid_bitsf, "%d ", bsgt_k);
    for(bsgt_i=0; bsgt_i < bsgt_k; bsgt_i++){
        fprintf(grid_bitsf, "%d ", bsgt_bits[bsgt_i]);
    }
}
fprintf(grid_bitsf, "\n\n\n\noptimising_bits\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nThe content of the image is a page from a document discussing the morphology in coastal and river environments. The page contains a section of code written in C programming language, which involves operations such as processing bit vectors, deleting dynamic memory, and setting loop and optimisation parameters. It also includes file operations using functions like `fprintf` and `fclose`. The code snippet includes error handling for file I/O operations. The document seems to be part of an academic or technical report.
printf("\nor C if you wish to use a different type: ");
(void) fgets(ot_line, sizeof(ot_line), stdin);
(void) sscanf(ot_line, "%c", &ot_check2);
if ((ot_check2 != 'D') && (ot_check2 != 'd')) {
    printf("\n\nEnter the number corresponding to the optimisation type required\n");
    (void) fgets(ot_line, sizeof(ot_line), stdin);
    (void) sscanf(ot_line, "%d", &ot_optype);
}
}
}
//optitypef = fopen ("optype.txt", "w+);
if (optitypef == NULL) {
    other_mistake(1);  //Can't open file
} else {
    fprintf(optitypef, "%d", ot_optype);
    fprintf(optitypef, "optimisation_type"),
    fclose(optitypef);
}
return;

/**************************************************************************
******************Set Discretisation Interval Function********************
**************************************************************************

void discret_int(int di_check1){
    char di_check2;
    char di_line[100];
    double di_disc;//optimisation type used (RGA=1, BGA=2, SA=3)
    di_disc = DISC;//assigns default optimisation type
    if(di_check1 == 1) {
        printf("\n\nEnter D if you wish to use the default discretisation interval,"),
        printf("\nor C if you wish to use a different interval: ");
        (void) fgets(di_line, sizeof(di_line), stdin);
        (void) sscanf(di_line, "%c", &di_check2);
        if ((di_check2 != 'D') && (di_check2 != 'd')) {
            printf("\nEnter the required discretisation interval: ");
            (void) fgets(di_line, sizeof(di_line), stdin);
            (void) sscanf(di_line, "%lf", &di_disc);
        }
    }
    discintf = fopen ("discint.txt", "w+"),
    if (discintf == NULL) {
        other_mistake(1);  //Can't open file
    } else {
        fprintf(discintf, "%lf", di_disc);
        fprintf(discintf, "discretisation_interval"),
        fclose(discintf);
    }
    return;
}

/**************************************************************************
*********************Define Number of Bits Function***********************
**************************************************************************

void define_bit_no(double dbn_ULgene[a], double dbn_LLgene[a], int dbn_bit_string[a], double dbn_disc_string[a], int dbn_L, double dbn_disc){
    double dbn_templf1;
    double dbn_templf2;
    int dbn_tempd;
    int dbn_i;
    int dbn_bno;
    dbn_tempd = 0;
    dbn_i = 0;
    dbn_bno = 0;
    dbn_templf1 = 0.0;
    dbn_templf2 = 0.0;
    /*The following lines calculate real bit values that correspond to resol*/
    for(dbn_i = 0; dbn_i < dbn_L; dbn_i++){
        dbn_templf1 = (log(((dbn_ULgene[dbn_i]-dbn_LLgene[dbn_i])/(dbn_disc))+1.0))/(log(2.0));
        dbn_tempd = int(dbn_templf1);
        dbn_bit_string[dbn_i] = dbn_tempd;
        dbn_disc_string[dbn_i] = ((dbn_ULgene[dbn_i] - dbn_LLgene[dbn_i])/(pow(2.0, double(dbn_tempd))))) - 1.0);
    }
Appendix C
Morphology in Coastal and River Environments

```c
/*printf(\"discstring %f\", dbn_disc_string[dbn_i]);
dbn_bno = dbn_bno + dbn_bit_string[dbn_i];
*/

bitnof = fopen(\"bit_no.txt\", \"w\");
if( bitnof == NULL ){
    other_mistake(1);  //Can't open file
} else{
    fprintf(bitnof, \"%d\", dbn_bno);
    fprintf(bitnof, \"\n\%s\", \"Number_of_bits_BGA\%");
    fclose(bitnof);
}

bitmemf = fopen(\"bit_mem.txt\", \"w\");
if( bitmemf == NULL ){
    other_mistake(1);  //Can't open file
} else{
    for(dbn_i = 0; dbn_i < dbn_L; dbn_i++){
        fprintf(bitmemf, \"%d\n\", dbn_bit_string[dbn_i]);
    }
    fprintf(bitmemf, \"\n\%s\", \"number_of_bits_per_member\%");
    fclose(bitmemf);
}

return;
}

/**************************************************************************
******************Decode Binary to Real Values Function*******************
**************************************************************************/

void decode_binary(double db_l1_pop[a][b], double db_pop[a][b], int db_bit_string[a], double
db_LLgene[a], double db_disc_string[a], int db_m, int db_L, int db_bno){
    int db_j;//used for arrays and loops
    int db_i;//used for arrays and loops
    int db_k;//used for arrays and loops
    int db_bs_part;//the number of bits representing each variable in string
    double db_two_pow;
    double db_count;
    double db_realno;
    double db_binary_convert;
    int db_spot;//spot along binary string
    db_bs_part = 0;
    db_two_pow = 0.0;
    db_count = 0.0;
    db_realno = 0.0;
    db_binary_convert = 0.0;
    db_spot = 0;
    for(db_j = 0; db_j < db_m; db_j++){
        db_spot = (db_bno - 1);
        for(db_i = (db_L - 1); db_i >= 0; db_i--){
            db_count = 0.0;
            db_realno = 0.0;
            db_binary_convert = 0.0;
            db_bs_part = db_bit_string[db_i];
            for(db_k = db_spot; db_k > (db_spot - db_bs_part); db_k--){
                db_two_pow = pow(2.0, db_count);
                db_binary_convert = (db_l1_pop[db_k][db_j]*db_two_pow);
                db_realno = db_realno + db_binary_convert;
                db_count = db_count + 1.0;
            }
            db_pop[db_i][db_j] = db_LLgene[db_i] + db_disc_string[db_i]*db_realno;
            db_spot = db_spot - db_bs_part;
        }
    }
    return;
}

/**************************************************************************
******************Convert Real Values to Binary Function******************
**************************************************************************/

void convert_binary(double cb_l1_pop[a][b], double cb_l4_pop[a][b], int cb_bit_string[a], double
cb_LLgene[a], double cb_disc_string[a], int cb_m, int cb_L, int cb_bno){
    int cb_j;//used for arrays and loops
    int cb_i;//used for arrays and loops
    int cb_k;//used for arrays and loops
    int cb_bs_part;//the number of bits representing each variable in string
    double cb_two_pow;
    double cb_count;
    double cb_realno;
    double cb_binary_convert;
    int cb_spot;//spot along binary string
    cb_bs_part = 0;
    cb_two_pow = 0.0;
    cb_count = 0.0;
    cb_realno = 0.0;
    cb_binary_convert = 0.0;
    cb_spot = 0;
    for(cb_j = 0; cb_j < cb_m; cb_j++){
        cb_spot = (cb_bno - 1);
        for(cb_i = (cb_L - 1); cb_i >= 0; cb_i--){
            cb_count = 0.0;
            cb_realno = 0.0;
            cb_binary_convert = 0.0;
            cb_bs_part = cb_bit_string[cb_i];
            for(cb_k = cb_spot; cb_k > (cb_spot - cb_bs_part); cb_k--){
                cb_two_pow = pow(2.0, cb_count);
                cb_binary_convert = (cb_l1_pop[cb_k][cb_j]*cb_two_pow);
                cb_realno = cb_realno + cb_binary_convert;
                cb_count = cb_count + 1.0;
            }
            cb_pop[cb_i][cb_j] = cb_LLgene[cb_i] + cb_disc_string[cb_i]*cb_realno;
            cb_spot = cb_spot - cb_bs_part;
        }
    }
    return;
}
```

250
double cb_binary_convert; //
int cb_spot; // spot along binary string
double cb_tot_sum; // sum of binary values

cb_bs_part = 0;
cb_two_pow = 0.0;
cb_count = 0.0;
cb_realno = 0.0;
cb_binary_convert = 0.0;
cb_tot_sum = 0.0;
for(cb_j = 0; cb_j < cb_m; cb_j++) {
    cb_spot = 0;
    for(cb_i = 0; cb_i < cb_L; cb_i++) {
        cb_count = double((cb_bit_string[cb_i] - 1));
        cb_realno = 0.0;
        cb_binary_convert = 0.0;
        cb_bs_part = cb_bit_string[cb_i];
        cb_tot_sum = (cb_l4_pop[cb_i][cb_j] - cb_LLgene[cb_i])/cb_disc_string[cb_i];
        for(cb_k = cb_spot; cb_k < (cb_spot + cb_bs_part); cb_k++) {
            cb_two_pow = pow(2.0, cb_count);
            if (cb_tot_sum >= cb_two_pow) {
                cb_l1_pop[cb_k][cb_j] = 1.0;
                cb_tot_sum = cb_tot_sum - cb_two_pow;
            } else if (cb_tot_sum < cb_two_pow) {
                cb_l1_pop[cb_k][cb_j] = 0.0;
            } else {
                mistake();
            }
            cb_count = cb_count - 1.0;
        }
        cb_spot = cb_spot + cb_bs_part;
    }
}
return;

/**************************************************************************
****************************Set Files Function****************************
**************************************************************************/

void set_files(void) {
    // writes default data into required files //
    char sf_check1;
    // no_keys char sf_line[100];
    int sf_hydra_bw;
    sf_hydra_bw = 0;
    sf_hydra_bw = HYDRA_BW;
    sf_check1 = 'D';
    // no_keys printf("\nEnter P if you wish to use some data from a run immediately\n");
    // no_keys printf("\npreceding this one.  (You must have run the GA at least once)\n");
    // no_keys printf("\non this computer, or copied all txt files from a previous run\n");
    // no_keys printf("\non a similar computer.\n");
    // no_keys printf("\nYou will be able to change some of your previous data if required.\n");
    // no_keys printf("\nthe GA for the first time on this computer: \n");
    // no_keys (void) fgets(sf_line, sizeof(sf_line), stdin);
    // no_keys (void) sscanf(sf_line, "%c", &sf_check1);
    if ((sf_check1 == 'P') && (sf_check1 == 'p')) {
        operation_file_write(2);
        if (sf_hydra_bw == 1) {
            set_grid_size(2);
            set_grid_type(2);
        } else if (sf_hydra_bw == 2) {
            bwset_grid_size(2);
            bwset_grid_type(2);
            still_water_level();
        } else {
            mistake();
        }
        optimisation_type(2);
        discret_int(2);
        loop_set(2);
    }
    endresf2 = fopen("endres.txt", "w+");
Appendix C

Morphology in Coastal and River Environments

if (endresf2 == NULL) {
    printf( "The file endres.txt was not opened\n" );
} else {
    fprintf (endresf2, "%s", "run_not_completed");
    fclose(endresf2);
}

/**************************************************************************
**********************Initial Defaults Function***************************
**************************************************************************

void initial_defaults(int id_check1)
{
    if(id_check1==1){
        printf("The following default values are embedded into the program");
        printf("\nPopulation * Generation * Crossover * Mutation *");
        printf("Size * Size * Probability * Probability *");
        printf("\n%4d * %4d * %.1f * %.1f", po, ma, px, pm);
        printf("\nCrossover type: ");
        if(CC == 1){
            printf("Average\n");
        } else if(CC == 2){
            printf("One Point\n");
        } else if(CC == 3){
            printf("Two Point\n");
        } else if(CC == 4){
            printf("Uniform\n");
        } else if(CC == 5){
            printf("Average-Uniform\n");
        } else{
            mistake();
            printf("\nMutation type: ");
            if(MM == 1){
                printf("Random\n");
            } else if(MM == 2){
                printf("Adjacency\n");
            } else{
                mistake();
            }
        }
    }
    else if(id_check1==2){
        printf("The default grid size is %4d (x) by %4d (y)", GSX, GSY);
    } else if(id_check1==3){
    } else if(id_check1==4){
        printf("The default optimisation type is %d", OPTYPE);
        printf("(RGA=1, BGA=2, SA=3)\n");
    } else if(id_check1==5){
        printf("The default discretisation interval is %f", DISC);
    } else if(id_check1==8){
        printf("The number of loops is %d",MAXLOOPS);
    } else{
        mistake();
    }
    return;
}

void prog_run_mess(void){
    printf("Please wait a few moments for the program to run\n");
    return;
}
/**************************************************************************
***************************Mistake Function*******************************
**************************************************************************
void mistake(void){
    printf("There has been a mistake");
    return;
}

**************************************************************************
************************Other Mistake Function****************************
**************************************************************************
void other_mistake(int om_no){
    char om_words[100];
    if(om_no==1){
        (void)strcpy(om_words,"Can't open file ");  /*Copies a string of characters into the array*/
    } else if(om_no==2){
        (void)strcpy(om_words,"Average Three Point Crossover is not possible, use uniform ");  /*Copies a string of characters into the array*/
    } else if(om_no==3){
        (void)strcpy(om_words,"Two point crossover is not possible, use uniform crossover ");  /*Copies a string of characters into the array*/
    } else if(om_no==4){
        (void)strcpy(om_words,"Assuming xover points ");  /*Copies a string of characters into the array*/
    } else if(om_no==5){
        (void)strcpy(om_words,"System too big for program to optimise ");  /*Copies a string of characters into the array*/
    } else if(om_no==6){
        (void)strcpy(om_words,"You entered an incorrect choice, please try again ");  /*Copies a string of characters into the array*/
    } else{
        mistake();
        printf("\nks", om_words);
        return;
    }

**************************************************************************
************************Print Results Function****************************
**************************************************************************
void print_results(int pr_itno, int pr_min, int pr_L, int pr_maxnogen){
    int pr_i;
    int pr_k;
    char pr_temps[50];
    int pr_tempd;
    double pr_tempf;
    double pr_min_popvm;
    double pr_min_popv;
    int pr_min_popim;
    double *pr_min_popvt;
    double pr_min_popt;
    int pr_min_popit;
    int pr_op_number;

    /*Allocation of dynamic memory for arrays*/
    if ((pr_min_popvm = new double[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if ((pr_min_popvt = new double[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar\n");
        exit(0);
    }

    /*initialise the arrays*/
    for(pr_i = 0; pr_i < a; pr_i++){
        pr_min_popvm(pr_i) = 0.0;
        pr_min_popvt(pr_i) = 0.0;
    }
    pr_op_number = 0;
optitype3f = fopen( "optype.txt", "r" );
if ( optitype3f == NULL ){
    other_mistake(1); //Can't open file
}
else{
    fscanf(optitype3f, "%d", &pr_op_number);
    fclose(optitype3f);
}
outputf = fopen( "allmin.txt", "r" );
if ( outputf == NULL ){
    other_mistake(1); //Can't open file
}
else{
    fscanf (outputf, "%s", &pr_temps); //crossover type
    fscanf (outputf, "%d", &pr_tempd); //crossover value
    fscanf (outputf, "%s", &pr_temps); //mutation type
    fscanf (outputf, "%d", &pr_tempd); //mutation value
    if (pr_op_number == 3){
        fscanf (outputf, "%s", &pr_temps); //population size
        fscanf (outputf, "%lf", &pr_templf); //population size value
        fscanf (outputf, "%s", &pr_temps); //generation size
        fscanf (outputf, "%lf", &pr_templf); //generation size value
    }
    else{
        fscanf (outputf, "%s", &pr_temps); //population size
        fscanf (outputf, "%d", &pr_tempd); //population size value
        fscanf (outputf, "%s", &pr_temps); //generation size
        fscanf (outputf, "%d", &pr_tempd); //generation size value
    }
    fscanf (outputf, "%s", &pr_temps); //crossover prob
    fscanf (outputf, "%lf", &pr_templf); //crossover prob value
    fscanf (outputf, "%s", &pr_temps); //mutation prob
    fscanf (outputf, "%lf", &pr_templf); //mutation prob value
    fscanf (outputf, "%s", &pr_temps); //generation
    for(pr_i=0; pr_i < pr_L; pr_i++){
        fscanf (outputf, "%d", &pr_tempd); //variable no value
    }
    fscanf (outputf, "%s", &pr_temps); //objective function
    fscanf (outputf, "%s", &pr_temps); //no of times
    if (pr_op_number == 3){
        fscanf (outputf, "%d", &pr_tempd); //increase_decrease
    }
    for(pr_k=0; pr_k < pr_min; pr_k++){
        for(pr_i=0; pr_i < pr_L; pr_i++){
            if (pr_i == 0){
                fscanf (outputf, "%d", &pr_tempd); //variable no
            }
            fscanf (outputf, "%lf", &pr_templf); //variable values
            if (pr_i == (pr_L - 1)){
                fscanf (outputf, "%lf", &pr_templf); //objective function value
            }
            if (pr_k == (pr_min + 1) & (pr_op_number == 3)){
                fscanf (outputf, "%d", &pr_tempd); //no of times
            }
            for (pr_k = (pr_min + 1); pr_k < (pr_maxnogen - 1); pr_k++){
                for (pr_i = 0; pr_i < pr_L; pr_i++){
                    if (pr_i == 0){
                        fscanf (outputf, "%d", &pr_tempd); //variable no
                    }
                    fscanf (outputf, "%lf", &pr_templf); //variable values
                    if (pr_i == (pr_L - 1)){
                        fscanf (outputf, "%lf", &pr_templf); //objective function value
                    }
                    if (pr_op_number == 3){
                        fscanf (outputf, "%d", &pr_tempd); //no of times
                    }
                }
            }
        }
    }
}
for(pr_i=0; pr_i < pr_L; pr_i++){
    fscanf( outputf, "%d", &pr_tempd);//generation no
    fscanf( outputf, "%lf", &pr_min_popvt[pr_i]);//variable values
    if(pr_i==(pr_L - 1)){
        fscanf( outputf, "%lf", &pr_min_popt);//objective function value
        fscanf( outputf, "%d", &pr_min_popit);//no of times
    }
}

fclose( outputf );

printf("\n\n\n       The Absolute Minimum Fitness Value from all generations\n\n");
printf("\n******************************************************************************\n\n*   Variable No.    * Variable Value  *   Fitness Value    * First Occurred*\n******************************************************************************\n*         1         *\nif(pr_min_popvm[0]<0.0){
    printf("% .2e **", pr_min_popvm[0]);
} else{
    printf(" % .2e **", pr_min_popvm[0]);
}
printf("% .2e **", pr_min_popvm);
if((pr_min + 1)<10){
    printf(" %d **", (pr_min + 1));
} else if((pr_min + 1)<100){
    printf("%d **", (pr_min + 1));
} else{
    printf(" %d **", (pr_min + 1));
}
printf("\n******************************************************************************\n\n");
//no keys for(pr_i=1; pr_i < pr_L; pr_i++){
//no keys   printf("\n*         %d         *", (pr_i+1));
//no keys   else if((pr_i+1)<100){
//no keys    printf("\n*        %d         *", (pr_i+1));
//no keys   }
//no keys   else{
//no keys    printf("\n*       %d         *", (pr_i+1));
//no keys   }
//no keys   if(pr_min_popvm[pr_i]<0.0){
//no keys   printf("% .2e **", pr_min_popvm[pr_i]);
//no keys   }
//no keys   else{
//no keys   printf("% .2e **", pr_min_popvm[pr_i]);
//no keys   }
//no keys   printf("\n******************************************************************************\n\n");
//no keys if(pr_min<pr_itno){
printf("\n\n\n       The minimum fitness that occured in the final generation (%d)\n\n");
printf("\n******************************************************************************\n\n*   Variable No.    * Variable Value  *   Fitness Value    *   Occurance   *\n******************************************************************************\n*         1         *
if(pr_min_popvt[0]<0.0){
    printf("% .2e **", pr_min_popvt[0]);
} else{
    printf("% .2e **", pr_min_popvt[0]);
}
printf("% .2e **", pr_min_popvt);
if(pr_min_popt<10){
    printf(" %d **", pr_min_popt);
} else if(pr_min_popt<100){
    printf("%d **", pr_min_popt);
} else{
    printf(" %d **", pr_min_popt);
}
printf("\n******************************************************************************\n\n");
//no keys for(pr_i=1; pr_i < pr_L; pr_i++){
Appendix C Morphology in Coastal and River Environments

```c
// no keys    if((pr_i+1)<10){
// no keys        printf("%d
", (pr_i+1));
// no keys    } else if((pr_i+1)<100){
// no keys        printf("%d
", (pr_i+1));
// no keys    } else{
// no keys        printf("%d
", (pr_i+1));
// no keys    }
// no keys    if(pr_min_popvt[pr_i]<0.0){
// no keys        printf("%.2e
", pr_min_popvt[pr_i]);
// no keys    } else{
// no keys        printf("%.2e
", pr_min_popvt[pr_i]);
// no keys    }
// no keys    printf("***************************************");
// no keys   }
else if(pr_min=pr_itno){
    printf("\n\nAbsolute Minimum occured in final generation (%d)\n", (pr_itno + 1));
else{
    mistake();
}
endresf = fopen("endres.txt", "w+");
if( endresf == NULL ){
    printf("The file endres.txt was not opened\n");
} else{
    for(pr_i=0; pr_i < pr_L; pr_i++){
        fprintf(endresf, "%.2e
", pr_min_popvm[pr_i]);
    }
    fprintf(endresf, "run_completed\n");
    fclose(endresf);
} /*Deletion of dynamic memory for arrays*/
delete[] pr_min_popvm;
delete[] pr_min_popvt;
return;
} /*End Program Function***************************
**************************************************************************
int end_program(void){
    /***********************************************************************
    char ep_line[100];
    char ep_check1 = 'Y';
    int ep_check2;
    /***********************************************************************
    printf("\n\nContinue to run program?:  Y/n   ");
    printf("\n\nEnter D if you wish to use all the default program values\n")
    printf("\n\nEnter P if you wish to use the same values as a previous run\n")
    printf("\n\n(you must have already run through the GA once, without exiting)\n");
    if ((ep_check1 != 'Y') & (ep_check1 != 'y')) {
        ep_check2=1;
    }
    else{
        ep_check2=2;
    }
    return(ep_check2);
} /*Check User Entry Function*******************************
**************************************************************************
int default_check(void){
    char dc_line[100];
    char dc_check1;
    int dc_check2;
    printf("\n\nEnter D if you wish to use all the default program values\n")
    printf("\n\nEnter P if you wish to use the same values as a previous run\n")
    printf("\n\n(you must have already run through the GA once, without exiting)\n");
    
```
Morphology in Coastal and River Environments

Appendix C

```c
(void) fgets(dc_line, sizeof(dc_line), stdin);
(void) sscanf(dc_line, "%c", &dc_check1);
if ((dc_check1 != 'D') && (dc_check1 != 'd') && (dc_check1 != 'P') && (dc_check1 != 'p')) {
    dc_check2=2;
} else if ((dc_check1 == 'P') || (dc_check1 == 'p')) {
    dc_check2=3;
} else {
    dc_check2=1;
}
return(dc_check2);

/**************************************************************************
**********Write file containing operation information Function************
**************************************************************************/

void operation_file_write(int ofw_check1){
char ofw_line[100];
char ofw_choice_cc1[50];
char ofw_choice_cc2[50];
char ofw_choice_cc3[50];
char ofw_choice_cc4[50];
char ofw_choice_cc5[50];
char ofw_choice_mm1[50];
char ofw_choice_mm2[50];
int ofw_cc;
int ofw_mm;
int ofw_po;
int ofw_ma;
double ofw_px;
double ofw_pm;
int ofw_check;
ofw_check=1;
if(ofw_check1==1){
ofw_check = default_check();
}
(void)strcpy(ofw_choice_cc1,"Average crossover "); /*Copies a string of characters into the array*/
(void)strcpy(ofw_choice_cc2,"One point crossover "); /*Copies a string of characters into the array*/
(void)strcpy(ofw_choice_cc3,"Two point crossover "); /*Copies a string of characters into the array*/
(void)strcpy(ofw_choice_cc4,"Uniform crossover "); /*Copies a string of characters into the array*/
(void)strcpy(ofw_choice_cc5,"Average-Uniform crossover "); /*Copies a string of characters into the array*/
if(ofw_check==1){
ofw_cc=CC;
ofw_mm=MM;
ofw_po=po;
ofw_ma=ma;
ofw_px=px;
ofw_pm=pm;
} else if (ofw_check==2){
printf("\n%s = 1, %s = 2, %s = 3, %s = 4 and %s = 5 ",
ofw_choice_cc1, ofw_choice_cc2, ofw_choice_cc3);printf("\n%s = 1, %s = 2, %s = 3, %s = 4 and %s = 5 ",
ofw_choice_cc4, ofw_choice_cc5);
printf("The default crossover type is %d", CC);

(void) fgets(ofw_line, sizeof(ofw_line), stdin); /*Reads line of input*/
(void) sscanf(ofw_line, "%d", &ofw_cc); /*Converts text to number and assigns it*/
printf("\n%s = 1, %s = 2, %s = 3, %s = 4 and %s = 5 ",
ofw_choice_mm1, ofw_choice_mm2);printf("\n%s = 1, %s = 2, %s = 3, %s = 4 and %s = 5 ",
ofw_choice_mm3, ofw_choice_mm4, ofw_choice_mm5);
printf("The default mutation type is %d", MM);

(void) fgets(ofw_line, sizeof(ofw_line), stdin); /*Reads line of input*/
(void) sscanf(ofw_line, "%d", &ofw_mm); /*Converts text to number and assigns it*/
printf("\n%s = 1, %s = 2, %s = 3, %s = 4 and %s = 5 ",
ofw_choice_mm1, ofw_choice_mm2, ofw_choice_mm3);printf("\n%s = 1, %s = 2, %s = 3, %s = 4 and %s = 5 ",
ofw_choice_mm4, ofw_choice_mm5);
printf("The default population size is %d", po);
printf("\nPlease choose the population size you wish to use: ");

(void) fgets(ofw_line, sizeof(ofw_line), stdin); /*Reads line of input*/
(void) sscanf(ofw_line, "%d", &ofw_po); /*Converts text to number and assigns it*/
printf("\nThe default number of generations is %d", ge);
printf("\nPlease choose the number of generations you wish to use: ");
```

257
Appendix C  Morphology in Coastal and River Environments

```c
(void) fgets(ofw_line, sizeof(ofw_line), stdin); /*Reads line of input                   */
(void) sscanf(ofw_line, "%d", &ofw_ma);         /*Converts text to number and assigns it*/
printf("The default probability of crossover is %f", px);
printf("Please enter the probability of crossover required (max 1.0):  ");
(void) fgets(ofw_line, sizeof(ofw_line), stdin); /*Reads line of input                   */
(void) sscanf(ofw_line, "%lf", &ofw_px);         /*Converts text to number and assigns it*/
printf("The default probability of mutation is %f", pm);
printf("Please enter the probability of mutation required (max 1.0):  ");
(void) fgets(ofw_line, sizeof(ofw_line), stdin); /*Reads line of input                   */
(void) sscanf(ofw_line, "%lf", &ofw_pm);         /*Converts text to number and assigns it*/
}
else if (ofw_check==3){
  return;
}
else{
  mistake();
}
GAopsf = fopen( "GAops.txt", "w+" );
if( GAopsf == NULL ){
  printf( "The file GAops.txt was not opened\n" );
}
else{
  fprintf (GAopsf, "%s
", "Crossover_type"); //14
  fprintf (GAopsf, "%d
", ofw_cc);
  fprintf (GAopsf, "%s
", "Mutation_type");    //13
  fprintf (GAopsf, "%d
", ofw_mm);
  fprintf (GAopsf, "%s
", "Population_Size");  //15
  fprintf (GAopsf, "%d
", ofw_po);
  fprintf (GAopsf, "%s
", "Generation_no");    //13
  fprintf (GAopsf, "%d
", ofw_ma);
  fprintf (GAopsf, "%s
", "Crossover_prob");   //14
  fprintf (GAopsf, "%f
", ofw_px);
  fprintf (GAopsf, "%s
", "Mutation_prob");    //13
  fprintf (GAopsf, "%f
", ofw_pm);
  fclose( GAopsf );
}
return;
}
**************************************************************************
************************Write Limit File Function************************
**************************************************************************

void fill_limits(double fl_ULgene[a], double fl_LLgene[a], int fl_L){
    int fl_i;
    int fl_k;
    int fl_j;
    int fl_q;
    int *fl_bits;
    double *fl_wbits;
    int fl_end;
    int nasty;
    double fl_gcnl;

    fl_gcnl = 0.0;
    /*Allocation of dynamic memory for arrays*/
    if (((fl_bits = new int[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    if (((fl_wbits = new double[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    nasty = fl_L;
    for(fl_i = 0; fl_i < a; fl_i++) {
        fl_ULgene[fl_i] = 0.0;
        fl_LLgene[fl_i] = 0.0;
        fl_bits[fl_i] = 0;
    }
    fl_j=0;
    fl_k = 0;
```
fl_j = 0;
fl_q = 0;
fl_end = 0;
grid_bits2f = fopen( "g_bits.txt", "r");
if( grid_bits2f == NULL ){
    other_mistake(1);  //Can't open file
} else{
    fscanf(grid_bits2f, "%d", &fl_end);
    for(fl_i=0; fl_i < fl_end; fl_i++){
        fscanf(grid_bits2f, "%d", &fl_bits[fl_i]);
    }
fclose(grid_bits2f);
}

if(HYDRA_BW==2){
    still_water3f = fopen( "swater_bits.txt", "r");
    if( still_water3f == NULL ){
        other_mistake(1);  //Can't open file
    } else{
        fscanf(still_water3f, "%d", &fl_end);
        for(fl_i=0; fl_i < fl_end; fl_i++){
            fscanf(still_water3f, "%lf", &fl_wbits[fl_i]);
        }
    fclose(grid_bits2f);
    }
}

fl_k = 0;
fl_i = 0;
while(fl_k < fl_end){
    if(HYDRA_BW==2){
        if((fl_wbits[fl_k]+GCNL) < 0.02){
            fl_gcnl = (-1.0)*fl_wbits[fl_k] + 0.02;
        } else if((fl_wbits[fl_k]+GCNL) >= 0.02){
            fl_gcnl = GCNL;
        } else{
            mistake();
            printf(" m1");
        }
    } else if(HYDRA_BW == 1){
        fl_gcnl = GCNL;
    } else{
        mistake();
        printf(" not breakwater or hydra");
    }
    if(fl_bits[fl_k] == 1){
        fl_ULgene[fl_i] = GCNU;
        fl_LLgene[fl_i] = fl_gcnl;
        fl_i=fl_i + 1;
    } else if(fl_bits[fl_k] == 2){
        fl_ULgene[fl_i] = GCNU;
        fl_LLgene[fl_i] = fl_gcnl;
        fl_i=fl_i + 1;
    } else if(fl_bits[fl_k] == 4){
        fl_ULgene[fl_i] = GCNU;
        fl_LLgene[fl_i] = fl_gcnl;
        fl_i=fl_i + 1;
    } else if(fl_bits[fl_k] == 6){
        fl_ULgene[fl_i] = GCNU;
        fl_LLgene[fl_i] = fl_gcnl;
        fl_i=fl_i + 1;
    } else if(fl_bits[fl_k] == 8){
    } else if(fl_bits[fl_k] == 10){
    } else if(fl_bits[fl_k] == 12){
    } else if(fl_bits[fl_k] == 16){
}
Appendix C  Morphology in Coastal and River Environments

```c
fl_ULgene[fl_i] = GCNU;
fl_LLgene[fl_i] = fl_gcnl;
fl_i = fl_i + 1;
}
else if(fl_bits[fl_k] == 20){
fl_ULgene[fl_i] = GCNU;
fl_LLgene[fl_i] = fl_gcnl;
fl_i = fl_i + 1;
}
else if(fl_bits[fl_k] == 24){
}
else if(fl_bits[fl_k] == 32){
fl_ULgene[fl_i] = GCNU;
fl_LLgene[fl_i] = fl_gcnl;
fl_i = fl_i + 1;
}
else if(fl_bits[fl_k] == 34){
fl_ULgene[fl_i] = GCNU;
fl_LLgene[fl_i] = fl_gcnl;
fl_i = fl_i + 1;
}
else if(fl_bits[fl_k] == 40){
}
else if(fl_bits[fl_k] == 48){
fl_ULgene[fl_i] = GCNU;
fl_LLgene[fl_i] = fl_gcnl;
fl_i = fl_i + 1;
}
else{
mistake();
printf("nasty ");
}
fl_k = fl_k + 1;
}
GAlimf = fopen( "GAlim.txt", "w+" );
if( GAlimf == NULL ){ printf("The file GAlim.txt was not opened\n");
}
else{
fprintf(GAlimf, "%s", "Variable_no");
fprintf(GAlimf, " %s", "Upper_limit");
fprintf(GAlimf, " %s\n", "Lower_limit");
for(fl_j = 0; fl_j < nasty; fl_j++){
fprintf(GAlimf, "%d", (fl_j + 1));
fprintf(GAlimf, " %f", fl_ULgene[fl_j]);
fprintf(GAlimf, " %f\n", fl_LLgene[fl_j]);
}
fclose(GAlimf);
}
/*Deletion of dynamic memory for arrays*/
delete [] fl_bits;
delete [] fl_wbits;
return;
}
/**************************************************************************
************************Calculate Fitness Function************************
**************************************************************************/
void calc_fitness(double cf_pop[a][b], double cf_L, int cf_m, int cf_type, int cf_bestpop, int
                     cf_op_comb, double cf_qx[a][b], double cf_qy[a][b], double cf_flowx[a][b], double cf_flowy[a][b])(
    double cf_answer;
    double (*cf_v)[BB];
    double (*cf_u)[BB];
    double (*cf_n)[BB];
    double (*cf_de)[BB];
    double (*cf_vol)[BB];
    double (*cf_energy)[BB];
    double (*cf_histogram)[HBB]; //used to plot information entropy histogram
    int (*cf_pt)[BB];
    int *cf_bits;
    double (*cf_crit vel)[BB]; //critical velocity
    double (*cf_stream_power)[BB]; //stream power
    int cf_no;
    int cf_jj; //used in for loops
    int cf_j; //used in for loops
    int cf_kj; //used in for loops
    int cf_check;
    int cf_gsx; //grid size of x
```
int cf_gsy; // grid size of y
double cf_Mx;
double cf_My;
double cf_deltax;
double cf_deltay;
double cf_conbit;
double cf_weightm;
double cf_weightc;
double cf_scalem;
double cf_scalec;
double cf_continuity;
double cf_momentum;
int cf_op_number;
int cf_tempd;
double cf_templf;
char cf_temps[50];
double cf_water_en_st;
double cf_water_en_end;
double cf_delta_en;
double cf_delta_v3;
double cf_delta_u3;
double cf_delta_esedv;
double cf_delta_ewatv;
double cf_delta_esedu;
double cf_delta_ewatu;
double cf_water_en_st;
double cf_water_en_end;
double cf_delta_en;
double cf_delta_v3;
double cf_delta_u3;
double cf_delta_esedv;
double cf_delta_ewatv;
double cf_delta_esedu;
double cf_delta_ewatu;
double cf_sed_pot;
double cf_sed_vol;
double cf_sweep_rowe; // sediment volume eroded in a row
double cf_sweep_tote; // total sediment volume eroded before row
double cf_sweep_rowd; // sediment volume deposited in a row
double cf_sweep_totd; // total sediment volume deposited before row
double cf_de_sum; // sum of de's
double cf_de_sum_sq; // sum of squared de's
double cf_standard_deviation; // standard deviation of de's
double cf_bedr_total; // total bed roughness
double cf_bedr_alpha; // ripple bed roughness proportional constant
double cf_bedr_ripple; // bed roughness due to surface
double cf_bedr_total; // total bed roughness
double cf_bedr_alpha; // ripple bed roughness proportional constant
double cf_bedr_ripple; // bed roughness due to surface
double cf_chezy_coeff; // chezy coefficient
double cf_chezy_loss; // bed loss due to bed roughness
double cf_ripple_length; // representative ripple length
double cf_ripple_height; // representative ripple height
double cf_hydraulic_radius; // hydraulic radius
double cf_ave_velocity = 0.0; // average velocity over grid (ignoring land points)
double cf_vector_velocity = 0.0; // vector velocity
int cf_check9;
int wccount;
double cf_sediment_balance; // sediment balance, should be zero else penalty applied
double cf_balance_penalty; // penalty if sediment balance not zero
double cf_vel_answer; // penalty if velocity greater than critical velocity
double cf_aor_answer; // penalty if angle of repose greater than allowable

double cf_crit_del; // critical delta elevation
double cf_el1; // differenve in elevation
double cf_el2; // differenve in elevation
double cf_el3; // differenve in elevation
double cf_el4; // differenve in elevation

double cf_vel_penalty_factor; // factor used to multiply velocity error to determine relevance
double cf_aor1_penalty_factor; // factor used to multiply angle of repose error to determine relevance
double cf_aor2_penalty_factor; // factor used to multiply angle of repose error to determine relevance
double cf_aor3_penalty_factor; // factor used to multiply angle of repose error to determine relevance
double cf_aor4_penalty_factor; // factor used to multiply angle of repose error to determine relevance

double cf_wavebed; // used to extrapolate for wave model bottom elevations

double cf_exner_error; // error between sediment eroded in a time step and rate at which it gets eroded
double cf_exner_errorabs; // error between sediment eroded in a time step and absolute rate

double cf_water_height; // height of water column
double cf_D_star; // used in the calc of suspended sed transport in van Rijn eqn
double cf_sed_trans_suspended; // calc suspended sed transport based on van Rijn
double cf_sed_trans_bed; // calc bedload sed transport based on van Rijn
double cf_sed_trans_total; // calculate total transport rate based on van Rijn

/* unix */
pid_t pid; // used in calling hydra3jo from unix
/* unix */
int status; // used in calling hydra3jo from unix

/* Allocation of dynamic memory for arrays*/
if ((cf_v = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
if ((cf_u = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_n = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_de = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_vol = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_bits = new int[AA]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_histogram = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_energy = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_crit_vel = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_pt = new int[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cf_stream_power = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}

cf_tempd = 0;
cf_templf = 0.0;
cf_check9 = 0;
cf_check = 1;
wccount = 0;
cf_Mx = 0.0;
cf_My = 0.0;
cf_deltax = 0.0;
cf_deltay = 0.0;
cf_deltax = 1.0;
cf_deltay = 1.0;
cf_continuity = 0.0;
cf_momentum = 0.0;
cf_weightm = 0.0;
cf_weightc = 0.0;
cf_scalem = 0.0;
cf_scalec = 0.0;
cf_conbit = 0.0;
cf_op_number = 0;
cf_water_en_st = 0.0;
cf_water_en_end = 0.0;
cf_delta_en = 0.0;
cf_delta_u3 = 0.0;
cf_delta_esedv = 0.0;
cf_delta_ewatv = 0.0;
cf_delta_esedu = 0.0;
cf_delta_ewatu = 0.0;
cf_sed_pot = 0.0;
cf_sed_vol = 0.0;
cf_sweep_rowe = 0.0;
cf_sweep_tote = 0.0;
cf_sweep_rowd = 0.0;
cf_sweep_totd = 0.0;
cf_de_sum = 0.0;//sum of de's
cf_de_sum_sq = 0.0;//sum of squared de's
cf_standard_deviation = 0.0;//standard deviation of de's
cf_bedr_sand = 0.0;//bed roughness due to surface
cf_bedr_ripple = 0.0;//bed roughness due to ripples
cf_bedr_total = 0.0;//total bed roughness
cf_bedr_alpha = 0.0;//ripple bed roughness proportional constant
cf_bedr_shear = 0.0;//bed shear velocity used to determine if hydraulically rough or smooth
cf_chezy_coeff = 0.0;//chezy coefficient
cf_chezy_loss = 0.0;//head loss due to bed roughness
cf_ripple_length = 0.0;//representative ripple length
Morphology in Coastal and River Environments

Appendix C

cf_ripple_height = 0.0;//representative ripple height
cf_hydraulic_radius = 0.0;//hydraulic radius
cf_ave_velocity = 0.0;//average velocity over grid (ignoring land points)
cf_sediment_balance = 0.0;//sediment balance, should be zero else penalty applied
cf_hydraulic_radius = 0.0;//penalty if sediment balance not zero
cf_vel_answer = 0.0;//penalty if velocity greater than critical velocity
cf_aor_answer = 0.0;//penalty if angle of repose greater than allowable

cf_vector_velocity = 0.0;//value of velocity vector

cf_crit_del = 0.0;//critical delta elevation

cf_el1 = 0.0;//difference in elevation

cf_el2 = 0.0;//difference in elevation

cf_el3 = 0.0;//difference in elevation

cf_el4 = 0.0;//difference in elevation

cf_vel_penalty_factor = 0.0;//factor used to multiply velocity error to determine relevance

cf_aor1_penalty_factor = 0.0;//factor used to multiply angle of repose error to determine relevance

cf_aor2_penalty_factor = 0.0;//factor used to multiply angle of repose error to determine relevance

cf_aor3_penalty_factor = 0.0;//factor used to multiply angle of repose error to determine relevance

cf_aor4_penalty_factor = 0.0;//factor used to multiply angle of repose error to determine relevance

cf_wavede = 0.0;

cf_exner_error = 0.0;
cf_exner_errorabs = 0.0;

cf_water_height = 0.0;//height of water column

cf_D_star = 0.0;//used in the calc of suspended sed transport in van Rijn eqn

cf_sed_trans_suspend = 0.0;//calc suspended sed transport based on van Rijn

cf_sed_trans_bed = 0.0;//calc bedload sed transport based on van Rijn

cf_sed_trans_total = 0.0;//calculate total transport rate based on van Rijn

cf_sed_trans_diverge = 0.0;//calculate sed trans divergence into and out of a cell.

for(cf_i=0; cf_i < a; cf_i++){
    cf_bits[cf_i] = 0;
    }

for(cf_j=0; cf_j < BB; cf_j++){
    for(cf_i=0; cf_i < AA; cf_i++){
        cf_v[cf_i][cf_j] = 0.0;
        cf_u[cf_i][cf_j] = 0.0;
        cf_n[cf_i][cf_j] = 0.0;
        cf_de[cf_i][cf_j] = 0.0;
        cf_vol[cf_i][cf_j] = 0.0;
        cf_energy[cf_i][cf_j] = 0.0;
        cf_crit_vel[cf_i][cf_j] = 0.0;
        cf_stream_power[cf_i][cf_j] = 0.0;
        cf_pt[cf_i][cf_j] = 0;
    }
    }

cf_weightm = WEIGHTM;
cf_weightc = WEIGHTC;
cf_scalem = SCALEM;
cf_scalec = SCALEC;
cf_crit_del = X_STEP*tan(ANGLE_REPOSE);//critical elevation for allowable angle of repose

cf_vel_penalty_factor = VEL_PENALTY_FACTOR;//factor used to multiply velocity error to determine relevance

cf_aor1_penalty_factor = AOR1_PENALTY_FACTOR;//factor used to multiply angle of repose error to determine relevance

cf_aor2_penalty_factor = AOR2_PENALTY_FACTOR;//factor used to multiply angle of repose error to determine relevance

cf_aor3_penalty_factor = AOR3_PENALTY_FACTOR;//factor used to multiply angle of repose error to determine relevance

cf_aor4_penalty_factor = AOR4_PENALTY_FACTOR;//factor used to multiply angle of repose error to determine relevance

grid_size3f = fopen( "g_size.txt", "r" );
if( grid_size3f == NULL ){
    other_mistake(1); //Can't open file
    printf("Can't open grid_size.txt!\n");
}
else{
    fscanf(grid_size3f, "%d", &cf_gsx);
    fscanf(grid_size3f, "%d", &cf_gsy);
    fclose(grid_size3f);
}

grid_v3f = fopen( "g_v.txt", "r" );
grid_u3f = fopen( "g_u.txt", "r" );
grid_n3f = fopen( "g_n.txt", "r" );
if( grid_v3f == NULL ){
    other_mistake(1); //Can't open file
}
printf("\ngrid_v");
}
else{
for(cf_j=(cf_gsy-1); cf_j > (-1); cf_j--){
  for(cf_i=0; cf_i < cf_gsx; cf_i++){
    fscanf(grid_v3f, "%lf", &cf_v[cf_i][cf_j]);
    fscanf(grid_u3f, "%lf", &cf_u[cf_i][cf_j]);
    fscanf(grid_n3f, "%lf", &cf_n[cf_i][cf_j]);
  }
}
fclose(grid_v3f);
fclose(grid_u3f);
fclose(grid_n3f);
}
grid_bits3f = fopen("g_bits.txt", "r");
if( grid_bits3f == NULL ){
  other_mistake(1);  //Can't open file
  printf("\ngrid_bits");
}
else{
  fscanf(grid_bits3f, "%d ", &cf_k);
  for(cf_i=0; cf_i < cf_k; cf_i++){
    fscanf(grid_bits3f, "%d ", &cf_bits[cf_i]);
  }
}
close(grid_bits3f);
}

optitype2f = fopen("optype.txt", "r");
if( optitype2f == NULL ){
  other_mistake(1);  //Can't open file
}
else{
  fscanf(optitype2f, "%d", &cf_op_number);
}
close(optitype2f);
}
for(cf_no=0; cf_no < cf_m; cf_no++){
  if(cf_type == 2){//if final run
    printf("cf_bestpop = %d", cf_bestpop);
    if((cf_op_number == 3) || (cf_op_comb == 1)){
      output2f = fopen("saallmin.txt", "r");
      for(cf_k=0; cf_k < cf_n; cf_k++){
        fscanf(output2f, "%d", &cf_tempd);
      }
    }
  }
}
for(cf_i=0; cf_i < cf_L; cf_i++){
  fscanf(output2f, "%d", &cf_tempd);
}
for(cf_k=0; cf_k < cf_no; cf_k++){
  for(cf_i=0; cf_i < cf_L; cf_i++){
    fscanf(output2f, "%d", &cf_tempd);
  }
}
}
fclose(output2f);
}
startpop2f = fopen("sastapop.txt", "w+"); //write best starting values to file
if(startpop2f == NULL){
    other_mistake(1);  //Can't open file
}
else{
    for(cf_i=0; cf_i < cf_L; cf_i++){
        fprintf(startpop2f, "%e ", cf_pop[cf_i][cf_no]);//variable values
    }
    fprintf(startpop2f, "\n");  
    fclose(startpop2f);  
   } //stop writing best starting values to file
} //end if final run
if((cf_type == 3) || (cf_type == 4)){ //if SA run or fast run
    cf_no=cf_bestpop;  
} //end if SA run or fast run

for(cf_i=0; cf_i < cf_L; cf_i++){
    if(cf_i==0){
        fscanf(output2f, "%d", &cf_tempd);//generation no
    }
    fscanf(output2f, "%lf", &cf_pop[cf_i][cf_no]);//variable values
    if(cf_i==(cf_L - 1)){
        fscanf(output2f, "%lf", &cf_templf);//objective function value
        printf("fitness function of = %f", cf_templf);
        fscanf(output2f, "%d", &cf_tempd);//no of times
    }
}
cfclose(output2f);

grid_type3f = fopen("g_type.txt", "r");
if(grid_type3f == NULL){
    other_mistake(1);  //Can't open file
}
else{
    while(1){
        fscanf(grid_type3f, "%d ", &cf_pt[cf_i][cf_j]);
        if(cf_pt[cf_i][cf_j] == 0){
        } else if(cf_pt[cf_i][cf_j] == 1){
            cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
            cf_k = cf_k + 1;
        } else if(cf_pt[cf_i][cf_j] == 2){
            cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
            cf_k = cf_k + 1;
        } else if(cf_pt[cf_i][cf_j] == 4){
            cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
            cf_k = cf_k + 1;
        } else if(cf_pt[cf_i][cf_j] == 6){
            cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
            cf_k = cf_k + 1;
        } else if(cf_pt[cf_i][cf_j] == 8){
        } else if(cf_pt[cf_i][cf_j] == 10){
        } else if(cf_pt[cf_i][cf_j] == 12){
        } else if(cf_pt[cf_i][cf_j] == 14){
        } else if(cf_pt[cf_i][cf_j] == 16){
            cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
            cf_k = cf_k + 1;
        } else if(cf_pt[cf_i][cf_j] == 18){
        } else if(cf_pt[cf_i][cf_j] == 20){
            cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
            cf_k = cf_k + 1;
        }
    }
} //end while
}

fscanf(output2f, "@f", &cf_templf);//objective function value
fscanf(output2f, "@d", &cf_tempd);//no of times
if(cf_op_number == 3) || (cf_op_comb == 1)){
    fscanf(output2f, "%d", &cf_tempd);//increase_decrease
}
else if(cf_pt[cf_i][cf_j] == 24)
{
}
else if(cf_pt[cf_i][cf_j] == 28)
{
}
else if(cf_pt[cf_i][cf_j] == 32)
{
  cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
  cf_k = cf_k + 1;
}
else if(cf_pt[cf_i][cf_j] == 34)
{
  cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
  cf_k = cf_k + 1;
}
else if(cf_pt[cf_i][cf_j] == 40)
{
}
else if(cf_pt[cf_i][cf_j] == 42)
{
}
else if(cf_pt[cf_i][cf_j] == 48)
{
  cf_de[cf_i][cf_j] = cf_pop[cf_k][cf_no];
  cf_k = cf_k + 1;
}
else if(cf_pt[cf_i][cf_j] == 56)
{
  else{
    mistake();
  }
  cf_i = cf_i + 1;
  if(cf_i == cf_gsx)
{
    cf_i = 0;
    cf_j = cf_j - 1;
    if(cf_j == 0)
    {
      break;
    }
  }
}
fclose(grid_type3f);

//   ///////////////////////////////////////////////
//   //convert elevations, so that only every second
//   //one is considered and the points inbetween
//   //are averaged linearly
//   ///////////////////////////////////////////////
//   for(cf_j=0; cf_j < cf_gsy; cf_j++)
//    for(cf_i=0; cf_i < cf_gsx; cf_i++)
//     cf_de[(cf_i+1)][cf_j] = cf_de[cf_i][cf_j] + (cf_de[(cf_i+2)][cf_j] - cf_de[cf_i][cf_j])/2.0;
//    cf_i = cf_i + 1;
//   }
//   ///////////////////////////////////////////////
//   //convert elevations passed the breakwater
//   //to zero so that passed wave breaker zone
//   //no sediment is transported
//   ///////////////////////////////////////////////
//   for(cf_j=0; cf_j < cf_gsy; cf_j++)
//    for(cf_i=8; cf_i < cf_gsx; cf_i++)
//    for(cf_i=16; cf_i < cf_gsx; cf_i++)
//    for(cf_i=0; cf_i < cf_gsx; cf_i++)
//     cf_de[cf_i][cf_j] = 0.0;
//    }
//   ///////////////////////////////////////////////
//   //convert elevations to zero at start and end
//   //of channel
//   ///////////////////////////////////////////////
//   for(cf_j=0; cf_j < cf_gsy; cf_j++)
//    for(cf_i=0; cf_i < 3; cf_i++)
//     cf_de[cf_i][cf_j] = 0.0;
//    for(cf_i=35; cf_i < cf_gsx; cf_i++)
//     cf_de[cf_i][cf_j] = 0.0;
//   }
//   ///////////////////////////////////////////////
//convert elevations to zero at start and end
//of channel for inlet evt
//   /////////////////////////////////////////////////
for(cf_j=0; cf_j < cf_gsy; cf_j++){  
  for(cf_i=0; cf_i < 6; cf_i++){  
    cf_de[cf_i][cf_j] = 0.0;  
  }
}

for(cf_j=0; cf_j < 6; cf_j++){  
  for(cf_i=22; cf_i < 27; cf_i++){  
    cf_de[cf_i][cf_j] = 0.0;  
  }
}

if(HYDRA_BW == 2){  
  wddepthf = fopen( "wdeltadepth.dat", "w+" );//write delta depth file for waves module  
  if( wddepthf == NULL ){  
    other_mistake(1);  //Can't open file  
  }else{  
    for(cf_i=0; cf_i < cf_gsx; cf_i++){  
      for(cf_j=0; cf_j < cf_gsy; cf_j++){  
        if(cf_de[cf_i][cf_j]<0.0){  
          fprintf(wddepthf, "%.3f ", cf_de[cf_i][cf_j]);  
        }else if((cf_de[cf_i][cf_j]>=0.0) && (cf_de[cf_i][cf_j]<10.0)){  
          fprintf(wddepthf, " %.3f ", cf_de[cf_i][cf_j]);  
        }else if(cf_de[cf_i][cf_j]>=10.0){  
          fprintf(wddepthf, "%.3f ", cf_de[cf_i][cf_j]);  
        }else{  
          mistake();  
        }
      }
    }
  }
}

if( ddepthf == NULL ){  
  ddepthf = fopen( "deltadepth.dat", "w+" );  
  if( ddepthf == NULL ){  
    other_mistake(1);  //Can't open file  
  }else{  
    for(cf_i=0; cf_i < cf_gsx; cf_i++){  
      for(cf_j=0; cf_j < cf_gsy; cf_j++){  
        if(cf_de[cf_i][cf_j]<0.0){  
          fprintf(ddepthf, "%.3f ", cf_de[cf_i][cf_j]);  
        }else if((cf_de[cf_i][cf_j]>=0.0) && (cf_de[cf_i][cf_j]<10.0)){  
          fprintf(ddepthf, " %.3f ", cf_de[cf_i][cf_j]);  
        }else if(cf_de[cf_i][cf_j]>=10.0){  
          fprintf(ddepthf, "%.3f ", cf_de[cf_i][cf_j]);  
        }else{  
          mistake();  
        }
      }
    }
  }
}
other_mistake(1); //Can't open file
} else{
    for(cf_i=0; cf_i < cf_gsx; cf_i++){
        for(cf_j=0; cf_j < cf_gsy; cf_j++) { //need to add extra if greater than or less than 10, 100, etc
            if(cf_de[cf_i][cf_j] < 0.0) {
                fprintf(ddepthf, "%.3f ", cf_de[cf_i][cf_j]);
            } else if((cf_de[cf_i][cf_j] >= 0.0) && (cf_de[cf_i][cf_j] < 10.0)) {
                fprintf(ddepthf, " %.3f ", cf_de[cf_i][cf_j]);
            } else if(cf_de[cf_i][cf_j] >= 10.0) {
                fprintf(ddepthf, "%.3f ", cf_de[cf_i][cf_j]);
            } else {
              mistake();
          }
        }
      }fprintf (ddepthf, "\n");
} fclose(ddepthf);
}
if(HYDRA_BW == 2) {
    //call wave module here
    /*unixin*/ if ((pid = fork()) == 0) execl("adw2_short", NULL);
    /*unixin*/ waitpid(pid, NULL, 0);
    //unixex*/
    spawnl(0,"adw2_short.exe","help",NULL);
    //call current module here
    /*unixin*/ if ((pid = fork()) == 0) execl("adi2_short", NULL);
    /*unixin*/ waitpid(pid, NULL, 0);
    //unixex*/
    spawnl(0,"adi2_short.exe","help",NULL);
} else if (HYDRA_BW == 1) {
    //call hydra3jo.exe here
    /*unixin*/ if ((pid = fork()) == 0) execl("hydra3jo", NULL);
    /*unixin*/ waitpid(pid, NULL, 0);
    //unixex*/
    spawnl(0,"hydra3jo.exe","help",NULL);
} else{
    mistake();
    printf(" not hydra or detach");
} if( small_arcof == NULL ){
    other_mistake(1); //Can't open file
} else{
    fscanf(small_arcof, "%lf", &cf_templf);//100.
    fscanf(small_arcof, "%d", &cf_tempd);//30
    fscanf(small_arcof, "%d", &cf_tempd);//60
    fscanf(small_arcof, "%lf", &cf_templf);//5.
    fscanf(small_arcof, "%lf", &cf_templf);//5.
    fscanf(small_arcof, "%s", &cf_temps);//u
    for(cf_i=0; cf_i < cf_gsx; cf_i++){
        for(cf_j=0; cf_j < cf_gsy; cf_j++) { //need to add extra if greater than or less than 10, 100, etc
            fscanf(small_arcof, "%lf", &cf_u[cf_i][cf_j]);
        }
    } fscanf(small_arcof, "%s", &cf_temps);//v
    for(cf_i=0; cf_i < cf_gsx; cf_i++){
        for(cf_j=0; cf_j < cf_gsy; cf_j++) { //need to add extra if greater than or less than 10, 100, etc
            fscanf(small_arcof, "%lf", &cf_v[cf_i][cf_j]);
        }
    } fscanf(small_arcof, "%s", &cf_temps);//e
    for(cf_i=0; cf_i < cf_gsx; cf_i++){
        for(cf_j=0; cf_j < cf_gsy; cf_j++) { //need to add extra if greater than or less than 10, 100, etc
            fscanf(small_arcof, "%lf", &cf_e[cf_i][cf_j]);
        }
    }
fscanf(small_arcof, "%lf", &cf_n[cf_i][cf_j]);
}
fclose(small_arcof);

if(cf_type == 2){//if final run
    //write the best configuration to file
    if((cf_op_number == 3) || (cf_op_comb == 1)){
        bestvf = fopen("sabestv.txt", "w+");
        bestuf = fopen("sabestu.txt", "w+");
        bestnf = fopen("sabestn.txt", "w+");
        bestdef = fopen("sabestde.txt", "w+");
    } else{
        bestvf = fopen("bestv.txt", "w+");
        bestuf = fopen("bestu.txt", "w+");
        bestnf = fopen("bestn.txt", "w+");
        bestdef = fopen("bestde.txt", "w+");
    }
    if(bestvf == NULL){
        other_mistake(1); //Can't open file
    } else{
        for(cf_j=(cf_gsy-1); cf_j > (-1); cf_j--){
            for(cf_i=0; cf_i < cf_gsx; cf_i++){
                fprintf(bestvf, "%e \n" , cf_v[cf_i][cf_j]);
                fprintf(bestuf, "%e \n" , cf_u[cf_i][cf_j]);
                fprintf(bestnf, "%e \n" , cf_n[cf_i][cf_j]);
                fprintf(bestdef, "%e \n" , cf_de[cf_i][cf_j]);
            }
        }
    }
    fclose(bestvf);
    fclose(bestuf);
    fclose(bestnf);
    fclose(bestdef);
    //end write the best configurations to file
    break;
} //end if final run

cf_answer = 0.0;

///////////////////////////////////////////
//work out fitness objective function here\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
} //end if GA
else if(cf_no==0){
    fit_filef = fopen("fitness.txt", "w+");
    if( fit_filef == NULL ){
        printf("\nfitness\n");
        other_mistake(1); //Can't open file
    } else{
        fprintf(fit_filef, "%e \n", cf_answer);
        fclose(fit_filef);
    }
} else if((cf_no<cf_m) && (cf_no>0)){
    fit_filef = fopen("fitness.txt", "a+");
    if( fit_filef == NULL ){
        printf("\nfitness x 2\n");
        other_mistake(1); //Can't open file
    } else{
        fprintf(fit_filef, "%e \n", cf_answer);
        fclose(fit_filef);
    }
} else{
    mistake();
} //end if GA
else if((cf_op_number == 3) || (cf_op_comb == 1) || (cf_type == 3)){//if SA
    SAfitf = fopen("safit.txt", "w+");
    if( SAfitf == NULL ){
        other_mistake(1); //Can't open file
    } //end if SA
} //end if GA
else{
  fprintf(SAfitf, "%e\n", cf_answer);
  fclose(SAfitf);
}
break;//escape from population loop
}//end if SA
else{//mistake as did not break for best population loop
  mistake();
}
}

/*Deletion of dynamic memory for arrays*/
delete [] cf_v;
delete [] cf_u;
delete [] cf_n;
delete [] cf_de;
delete [] cf_vol;
delete [] cf_bits;
delete [] cf_energy;
delete [] cf_histogram;
delete [] cf_crit_vel;
delete [] cf_stream_power;
delete [] cf_pt;
return;
}

/**************************************************************************
************************Calculate Fitness Function************************
**************************************************************************/

void calc_fit_nc(double cfn_pop[a][b], double cfn_L, int cfn_m, int cfn_type, int cfn_bestpop, int cfn_op_comb, double cfn_qx[a][b], double cfn_qy[a][b], double cfn_flowx[a][b], double cfn_flowy[a][b]){

double cfn_answer;

double (*cfn_v)[BB];
double (*cfn_u)[BB];
double (*cfn_n)[BB];
double (*cfn_de)[BB];
int (*cfn_pt)[BB];
int *cfn_bits;
int cfn_no;
int cfn_i;//used in for loops
int cfn_j;//used in for loops
int cfn_k;//used in for loops
int cfn_check;
int cfn_gsx;//grid size of x
int cfn_gsy;//grid size of y
int cfn_op_number;
int cfn_tempd;
double Cfn_tempf;
char Cfn_temps[50];
int cfn_check9;
int wccount;

double cfn_wavede;//used to extrapolate for wave model bottom elevations

double cfn_water_height;//height of water column

double cfn_D_star;//used in the calc of suspended sed transport in van Rijn eqn
double cfn_sed_trans_bed;//calc bedload sed transport based on van Rijn
double cfn_sed_trans_total;//calc total transport rate based on van Rijn
double cfn_sed_trans_diverge;//calc sed transport divergence into and out of a cell.

/*unixin*/  pid_t pid;//used in calling hydra3jo from unix
/*unixin*/  int status;//used in calling hydra3jo from unix

/*Allocation of dynamic memory for arrays*/
if ((cfn_v = new double[AA][BB]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((cfn_u = new double[AA][BB]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((cfn_n = new double[AA][BB]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((cfn_de = new double[AA][BB]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}
if ((cfn_bits = new int[a]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((cfn_pt = new int[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
cfn_tempd = 0;
cfn_tempfl = 0.0;
cfn_check8 = 0;
cfn_check = 1;
wccount = 0;
c_fn_op_number = 0;
c_fn_wavea = 0.0;
cfn_water_height = 0.0;//height of water column
cfn_D_star = 0.0;//used in the calc of suspended sed transport in van Rijn eqn
cfn_sed_trans_suspend = 0.0;//calc suspended sed transport based on van Rijn
cfn_sed_trans_bed = 0.0;//calc bedload sed transport based on van Rijn
cfn_sed_trans_total = 0.0;//calculate total transport rate based on van Rijn
cfn_sed_trans_diverge = 0.0;//calc sed trans divergence into and out of a cell.
for(cfn_i=0; cfn_i < a; cfn_i++){
    cfn_bits[cfn_i] = 0;
}
for(cfn_j=0; cfn_j < BB; cfn_j++){
    for(cfn_i=0; cfn_i < AA; cfn_i++){
        cfn_v[cfn_i][cfn_j] = 0.0 -
        cfn_u[cfn_i][cfn_j] = 0.0 -
        cfn_de[cfn_i][cfn_j] = 0.0;
        cfn_pt[cfn_i][cfn_j] = 0;
    }
}
g grid_size3f = fopen( "g_size.txt", "r");
if( grid_size3f == NULL ){
    other_mistake(1);  //Can't open file
    printf("Can't open file\n");
} else{
    fscanf(grid_size3f, "%d", &cfn_gsx);
    fscanf(grid_size3f, "%d", &cfn_gsy);
    fclose(grid_size3f);
}
g grid_v3f = fopen( "g_v.txt", "r");
g grid_u3f = fopen( "g_u.txt", "r");
g grid_n3f = fopen( "g_n.txt", "r");
if( grid_v3f == NULL ){
    other_mistake(1);  //Can't open file
    printf("Can't open file\n");
} else{
    for(cfn_j=(cfn_gsy-1); cfn_j > (-1); cfn_j--)
        for(cfn_i=0; cfn_i < cfn_gsx; cfn_i++){
            fscanf(grid_v3f, "%lf", &cfn_v[cfn_i][cfn_j]);
            fscanf(grid_u3f, "%lf", &cfn_u[cfn_i][cfn_j]);
            fscanf(grid_n3f, "%lf", &cfn_n[cfn_i][cfn_j]);
        }
    fclose(grid_v3f);
    fclose(grid_u3f);
    fclose(grid_n3f);
}
g grid_bits3f = fopen( "g_bits.txt", "r");
if( grid_bits3f == NULL ){
    other_mistake(1);  //Can't open file
    printf("Can't open file\n");
} else{
    fscanf(grid_bits3f, "%d ", &cfn_k);
    for(cfn_i=0; cfn_i < cfn_k; cfn_i++){
        fscanf(grid_bits3f, "%d ", &cfn_bits[cfn_i]);
    }
    fclose(grid_bits3f);
}
optitype2f = fopen ( "optype.txt", "r");
if( optitype2f == NULL ){
    other_mistake(1);  //Can't open file
}
else{
    fscanf(optitype2f, "kd", &cfn_op_number);
    fclose(optitype2f);
}

for(cfn_no=0; cfn_no < cfn_m; cfn_no++){
    if(cfn_type == 3){//if SA run
        cfn_no=cfn_bestpop;////what it is
    } //end if SA run
    cfn_i = 0;
    cfn_j = (cfn_gsy - 1);
    cfn_k = 0;
    grid_type3f = fopen( "g_type.txt", "r");
    if( grid_type3f == NULL ){
        other_mistake(1);  //Can't open file
        printf("ingrid_type");
        return;
    } else{
        while(1){
            fscanf(grid_type3f, "%d", &cfn_pt[cfn_i][cfn_j]);
            if(cfn_pt[cfn_i][cfn_j] == 0){
                } else if(cfn_pt[cfn_i][cfn_j] == 1){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 2){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 4){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 6){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 8){
                    } else if(cfn_pt[cfn_i][cfn_j] == 10){
                    } else if(cfn_pt[cfn_i][cfn_j] == 12){
                    } else if(cfn_pt[cfn_i][cfn_j] == 14){
                    } else if(cfn_pt[cfn_i][cfn_j] == 16){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 20){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 22){
                    } else if(cfn_pt[cfn_i][cfn_j] == 28){
                    } else if(cfn_pt[cfn_i][cfn_j] == 32){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 34){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 40){
                    } else if(cfn_pt[cfn_i][cfn_j] == 42){
                    } else if(cfn_pt[cfn_i][cfn_j] == 48){
                    cfn_de[cfn_i][cfn_j] = cfn_pop[cfn_k][cfn_no];
                    cfn_k = cfn_k + 1;
                } else if(cfn_pt[cfn_i][cfn_j] == 56){
                    } else{
                        mistake();
                        printf(" 1");
                    }
        }
    }
}
cfn_i = cfn_i + 1;
if(cfn_i == cfn_gsx){
  cfn_i = 0;
  cfn_j = cfn_j - 1;
if(Cfn_j == 0){
    break;
  }
}
close(grid_type3f);
}

///////////////////////////////////////////////
//convert elevations to zero at start and end
//of channel for inlet.evt
///////////////////////////////////////////////
for(cfn_j=0; cfn_j < cfn_gsy; cfn_j++){
  for(cfn_i=0; cfn_i < 6; cfn_i++){
    cfn_de[cfn_i][cfn_j] = 0.0;
  }
  for(cfn_i=22; cfn_i < 27; cfn_i++){
    cfn_de[cfn_i][cfn_j] = 0.0;
  }
}
if(HYDRA_BW == 2){
  small_arcof = fopen( "detach.vel", "r" );
} else if(HYDRA_BW == 1){
  small_arcof = fopen( "inlet.out", "r" );
} else{
  mistake();
  printf(" not hydra or detach");
}
if( small_arcof == NULL ){  //Can't open file
  other_mistake(1);  //Can't open file
} else{
  fscanf(small_arcof, "%lf", &cfn_templf);//100.
  fscanf(small_arcof, "%d", &cfn_tempd);//30
  fscanf(small_arcof, "%d", &cfn_tempd);//60
  fscanf(small_arcof, "%lf", &cfn_templf);//5.
  fscanf(small_arcof, "%lf", &cfn_templf);//5.
  fscanf(small_arcof, "%lf", &cfn_temps);//u
  for(cfn_i=0; cfn_i < cfn_gsx; cfn_i++){
    fscanf(small_arcof, "%lf", &cfn_templf);
  }
  fscanf(small_arcof, "%lf", &cfn_temps);//v
  for(cfn_i=0; cfn_i < cfn_gsx; cfn_i++){
    fscanf(small_arcof, "%lf", &cfn_temps);
  }
  fscanf(small_arcof, "%s", &cfn_temps);//e
  for(cfn_i=0; cfn_i < cfn_gsx; cfn_i++){
    fscanf(small_arcof, "%lf", &cfn_n[cfn_i][cfn_j]);
  }
  fclose(small_arcof);
}

//calculate velocities from flows
for(cfn_j=0; cfn_j < cfn_gsy; cfn_j++){
  for(cfn_i=0; cfn_i < cfn_gsx; cfn_i++){
    cfn_water_height = cfn_n[cfn_i][cfn_j]+STILL_DEPTH+cfn_de[cfn_i][cfn_j];
    cfn_u[cfn_i][cfn_j] = cfn_flowx[cfn_i][cfn_j]/cfn_water_height;
    cfn_v[cfn_i][cfn_j] = cfn_flowy[cfn_i][cfn_j]/cfn_water_height;
  }
  cfn_answer = 0.0;
}

///////////////////////////////////////////////////
//work out fitness objective function here/////////
///////////////////////////////////////////////////
cfn_answer = find_fit_value(cfn_gsx, cfn_gsy, cfn_u, cfn_v, cfn_n, cfn_de, cfn_qx, cfn_qy, cfn_flowx, cfn_flowy);
if((cfn_op_number == 1) || (cfn_op_number == 2)){//if GA
if(cfn_no==0){
  fit_filef = fopen( "fitness.txt", "w+");
  if( fit_filef == NULL ){
    other_mistake(1);  //Can't open file
    printf("\nfitness");
  }
  else{
    fprintf(fit_filef, "%e\n", cfn_answer);
    fclose(fit_filef);
  }
} else if((cfn_no>0) && (cfn_no>0)){
  fit_filef = fopen( "fitness.txt", "a+");
  if( fit_filef == NULL ){
    other_mistake(1);  //Can't open file
    printf("\nfitness x 2")
  }
  else{
    fprintf(fit_filef, "%e\n", cfn_answer);
    fclose(fit_filef);
  }
} else{
  mistake();
  printf(" 2");
}
} //end if GA
else if((cfn_op_number == 3) || (cfn_op_comb == 1) || (cfn_type ==3)){//if SA
SAfitf = fopen( "safit.txt", "w+");
if( SAfitf == NULL ){ //Can't open file
  other_mistake(1);
  fprintf(SAfitf, "%e\n", cfn_answer);
  fclose(SAfitf);
} break;//escape from population loop
} //end if SA
else if(mistake as did not break for best population loop
  printf(" 3");
}

/*Deletion of dynamic memory for arrays*/
delete [] cfn_v;
delete [] cfn_u;
delete [] cfn_n;
delete [] cfn_de;
delete [] cfn_bits;
delete [] cfn_pt;
return;
*/

/**************************************************************************
****************************Find Fitness Value***************************
**************************************************************************

double find_fit_value(int ffv_gsx, int ffv_gsy, double ffv_u[a][b], double ffv_v[a][b], double ffv_n[a][b], double ffv_de[a][b], double ffv_qx[a][b], double ffv_qy[a][b], double ffv_flowx[a][b], double ffv_flowy[a][b])
{
  double ffv_answer;
  double (*ffv_vol)[BB];
  double (*ffv_energy)[BB];
  double (*ffv_histogram)[HBB];//used to plot information entropy histogram
  int (*ffv_pt)[BB];
  int *ffv_bits;
  double (*ffv_crit_vel)[BB];//critical velocity
  double (*ffv_crit_vel)[BB];//critical velocity
  int ffv_i://used in for loops
  int ffv_j;//used in for loops
  int ffv_p;//used in for loops
  int ffv_check;
  double ffv_Nx;
  double ffv_Ny;
  }
double ffv_deltax;
double ffv_deltay;
double ffv_conbit;
double ffv_weightm;
double ffv_weightc;
double ffv_scalem;
double ffv_continuity;
double ffv_momentum;
int ffv_op_number;
int ffv_tempd;
double ffv_tempf;
double ffv_water_en_st;
double ffv_water_en_end;
double ffv_delta_en;
double ffv_delta_u3;
double ffv_delta_esedv;
double ffv_delta_ewatv;
double ffv_delta_esedu;
double ffv_delta_ewatu;
double ffv_sed_pot;
double ffv_sed_vol;
double ffv_sweep_rowe;//sediment volume eroded in a row
double ffv_sweep_rowd;//total sediment volume deposited in a row
double ffv_sweep_tote;//total sediment volume deposited before row
double ffv_de_sum;//sum of de's
double ffv_de_sum_sq;//sum of squared de's
double ffv_standard_deviation;//standard deviation of de's
double ffv_bedd_sand;//bed roughness due to surface
double ffv_bedd_ripple;//bed roughness due to ripples
double ffv_bedd_total;//total bed roughness
double ffv_bedd_alpha;//ripple bed roughness proportional constant
double ffv_bed_shear_velocity;//bed shear velocity used to determine if hydraulically rough or smooth
double ffv_chezy_coef;//chezy coefficient
double ffv_chezy_loss;//head loss due to bed roughness
double ffv_ripple_length;//representative ripple length
double ffv_ripple_height;//representative ripple height
double ffv_ripple_water_height;//representative ripple height
double ffv_ave_velocity = 0.0;//average velocity over grid (ignoring land points)
double ffv_vector_velocity = 0.0;//vector velocity
int ffv_check9;
int wccount;
double ffv_sediment_balance;//sediment balance, should be zero else penalty applied
double ffv_balance_penalty;//penalty if sediment balance not zero
double ffv_vel_answer;//penalty if velocity greater than critical velocity
double ffv_aor_answer;//penalty if angle of repose greater than allowable
double ffv_crit_del1;//critical delta elevation
double ffv_el1;//difference in elevation
double ffv_el2;//difference in elevation
double ffv_el3;//difference in elevation
double ffv_el4;//difference in elevation

double ffv_vel_penalty_factor;//factor used to multiply velocity error to determine relevance
double ffv_aor1_penalty_factor;//factor used to multiply angle of repose error to determine relevance
double ffv_aor2_penalty_factor;//factor used to multiply angle of repose error to determine relevance
double ffv_aor3_penalty_factor;//factor used to multiply angle of repose error to determine relevance
double ffv_aor4_penalty_factor;//factor used to multiply angle of repose error to determine relevance
double ffv_wavede;//used to extrapolate for wave model bottom elevations

double ffv_exner_error;//error between sediment eroded in a time step and rate at which it gets eroded
double ffv_exner_errorabs;//error between sediment eroded in a time step and absolute rate at which it gets eroded

double ffv_water_height;//height of water column
double ffv_D_star;//used in the calc of suspended sed transport in van Rijn eqn
double ffv_sed_trans_suspended;//calc suspended sed transport based on van Rijn
double ffv_sed_trans_load;//calc bedload sed transport based on van Rijn
double ffv_sed_trans_total;//calculate total transport rate based on van Rijn
double ffv_sed_trans_diverge;//calculate sed trans divergence into and out of a cell.

/*unix*/  pid_t pid;//used in calling hydra3jo from unix
/*unix*/  int status;//used in calling hydra3jo from unix

/*Allocation of dynamic memory for arrays*/
if (ffv_vol = new double[AA][BB]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
} else {
  ffv_bits = new int[a] = NULL;
}
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((ffv_histogram = new double[AA][BB]) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((ffv_energy = new double[AA][BB]) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((ffv_crit_vel = new double[AA][BB]) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((ffv_pt = new int[AA][BB]) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
if ((ffv_stream_power = new double[AA][BB]) == NULL) {
printf("Memory Allocation Failure for Array ar1\n")
exit(0);
}
ffv_tempd = 0;
ffv_tempif = 0.0;
ffv_check9 = 1;
wccount = 0;
ffv_Mx = 0.0;
ffv_My = 0.0;
ffv_deltax = 0.0;
ffv_deltay = 0.0;
ffv_deltax = 1.0;
ffv_deltay = 1.0;
ffv_energy = 0.0;
ffv_momentum = 0.0;
ffv_weightm = 0.0;
ffv_weightc = 0.0;
ffv_scalem = 0.0;
ffv_scalec = 0.0;
ffv_momentum = 0.0;
ffv_chezy_loss = 0.0;
ffv_chezy_coeff = 0.0;
ffv_ripple_length = 0.0;
ffv_ripple_height = 0.0;
ffv_hydraulic_radius = 0.0;
ffv_ave_velocity = 0.0;
ffv_sediment_balance = 0.0;
ffv_balance_penalty = 0.0;
ffv_vei1_answer = 0.0;
ffv_veil2_answer = 0.0;
ffv_aor_answer = 0.0;
ffv_vector_velocity = 0.0;
ffv_critical_del = 0.0;
ffv_el1 = 0.0;
ffv_el2 = 0.0;
ffv_el3 = 0.0;
ffv_el4 = 0.0;
ffv_sediment_balance = 0.0;
ffv_velu_penalty_factor = 0.0;
ffv_aor1_penalty_factor = 0.0;
ffv_aor2_penalty_factor = 0.0;
ffv_aor3_penalty_factor = 0.0;//factor used to multiply angle of repose error to determine relevance
ffv_aor4_penalty_factor = 0.0;//factor used to multiply angle of repose error to determine relevance
ffv_wavede = 0.0;
ffv_exner_error = 0.0;
ffv_exner_errorabs = 0.0;
ffv_water_height = 0.0;//height of water column
ffv_D_star = 0.0; //used in the calc of suspended sed transport in van Rijn eqn
ffv_sed_trans_suspend = 0.0; //calc suspended sed transport based on van Rijn
ffv_sed_trans_bed = 0.0; //calc bedload sed transport based on van Rijn
ffv_sed_trans_total = 0.0; //calculate total transport rate based on van Rijn
ffv_sed_trans_diverge = 0.0; //calculate sed trans divergence into and out of a cell.

for(ffv_i=0; ffv_i < a; ffv_i++)
{
  ffv_bits[ffv_i] = 0;
}
for(ffv_j=0; ffv_j < BB; ffv_j++)
{
  for(ffv_i=0; ffv_i < AA; ffv_i++)
  {
    ffv_vol[ffv_i][ffv_j] = 0.0;
    ffv_energy[ffv_i][ffv_j] = 0.0;
    ffv_crit_vel[ffv_i][ffv_j] = 0.0;
    ffv_stream_power[ffv_i][ffv_j] = 0.0;
    ffv_pt[ffv_i][ffv_j] = 0;
  }
}
for(ffv_j=0; ffv_j < HBB; ffv_j++)
{
  for(ffv_i=0; ffv_i < HAA; ffv_i++)
  {
    ffv_histogram[ffv_i][ffv_j] = 0.0;
  }
}
ffv_weightm = WEIGHTM;
ffv_weightc = WEIGHTC;
ffv_scalem = SCALEM;
ffv_scalec = SCALEC;
ffv_crit_del = X_STEP*tan(ANGLE_REPOSE); //critical elevation for allowable angle of repose

//check that rezero each time
ffv_answer = 0.0;
ffv_water_en_st = 0.0;
ffv_water_en_end = 0.0;
ffv_delta_en = 0.0;
ffv_delta_v3 = 0.0;
ffv_delta_u3 = 0.0;
ffv_delta_esedv = 0.0;
ffv_delta_ewatv = 0.0;
ffv_delta_esedu = 0.0;
ffv_delta_ewatu = 0.0;
ffv_sed_pot = 0.0;
ffv_sed_vol = 0.0;
ffv_sweep_rowe = 0.0;
ffv_sweep_tote = 0.0;
ffv_sweep_rowd = 0.0;
ffv_sweep_totd = 0.0;
ffv_de_sum = 0.0;//sum of de's
ffv_de_sum_sq = 0.0;//sum of squared de's
ffv_standaId_deviation = 0.0;//standard deviation of de's
ffv_bedr_sand = 0.0;//bed roughness due to surface
ffv_bedr_ripple = 0.0;//bed roughness due to ripples
ffv_bedr_total = 0.0;//total bed roughness
ffv_bedr_alpha = 0.0;//ripple bed roughness proportional constant
ffv_bedr_shear_velocity = 0.0;//bed shear velocity used to determine if hydraulically rough or smooth
ffv_chezy_coeff = 0.0;//chezy coefficient
ffv_chezy_loss = 0.0;//head loss due to bed roughness
ffv_ripple_length = 0.0;//representative ripple length
ffv_ripple_height = 0.0;//representative ripple height
ffv_hydraulic_radius = 0.0;//hydraulic radius
ffv_ave_velocity = 0.0;//average velocity over grid (ignoring land points)
for(ffv_p=0; ffv_p < HAA; ffv_p++)
{
  ffv_histogram[ffv_p][1] = 0.0;
}
ffv_sediment_balance = 0.0;//sediment balance, should be zero else penalty applied
ffv_balance_penalty = 0.0;//penalty if sediment balance not zero
ffv_vel_answer = 0.0;//penalty if velocity greater than critical velocity
ffv_vel2_answer = 0.0;//penalty if velocity greater than critical velocity
ffv_aor_answer = 0.0;//penalty if angle of repose greater than allowable
ffv_vector_velocity = 0.0;//value of velocity vector
ffv_el1 = 0.0;//difference in elevation
ffv_el2 = 0.0;//difference in elevation
ffv_el3 = 0.0;//difference in elevation
ffv_el4 = 0.0;//difference in elevation
ffv_enner_error = 0.0;  
ffv_enner_errorabs = 0.0; 

ffv_sed_trans_diverge = 0.0; //calculate sed trans divergence into and out of a cell. 

///////////////////////////////////////////////////  
//work out fitness objective function here/////////  
/////////////////////////////////////////////////// 

///////////////////////////////////////  
//calculation of energy difference  
/////////////////////////////////////// 

ffv_answer = 0; 
ffv_water_en_st = 0.0;  
ffv_water_en_end = 0.0; 

for(ffv_j=12; ffv_j < (ffv_gsy - 2); ffv_j++) { //inlet.evt 
  ffv_i = 3;  
  ffv_water_en_st = ffv_water_en_st + (pow(((ffv_u[ffv_i][ffv_j]+ffv_u[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (pow(((ffv_v[ffv_i][ffv_j]+ffv_v[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (ffv_n[ffv_i][ffv_j]); 
  ffv_i = 4; 
  ffv_water_en_st = ffv_water_en_st + (pow(((ffv_u[ffv_i][ffv_j]+ffv_u[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (pow(((ffv_v[ffv_i][ffv_j]+ffv_v[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (ffv_n[ffv_i][ffv_j]); 
  ffv_i = 5; 
  ffv_water_en_st = ffv_water_en_st + (pow(((ffv_u[ffv_i][ffv_j]+ffv_u[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (pow(((ffv_v[ffv_i][ffv_j]+ffv_v[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (ffv_n[ffv_i][ffv_j]); 
}

for(ffv_i=23; ffv_i < (ffv_gsx - 6); ffv_i++) { //inlet.evt 
  ffv_j = 3; //inlet.evt 
  ffv_water_en_end = ffv_water_en_end + (pow(((ffv_u[ffv_i][ffv_j]+ffv_u[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (pow(((ffv_v[ffv_i][ffv_j]+ffv_v[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (ffv_n[ffv_i][ffv_j]); 
  ffv_j = 4; //inlet.evt 
  ffv_water_en_end = ffv_water_en_end + (pow(((ffv_u[ffv_i][ffv_j]+ffv_u[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (pow(((ffv_v[ffv_i][ffv_j]+ffv_v[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (ffv_n[ffv_i][ffv_j]); 
  ffv_j = 5; //inlet.evt 
  ffv_water_en_end = ffv_water_en_end + (pow(((ffv_u[ffv_i][ffv_j]+ffv_u[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (pow(((ffv_v[ffv_i][ffv_j]+ffv_v[(ffv_i-1)][ffv_j])/2.0,2.0))/2.0*GRAVITY) + (ffv_n[ffv_i][ffv_j]); 
}

ffv_water_en_st = ffv_water_en_st/(6.0); //inlet.evt 
ffv_water_en_end = ffv_water_en_end/(6.0); //inlet.evt 
ffv_delta_en = ffv_water_en_st - ffv_water_en_end; 
ffv_answer = ffv_delta_en*1000.0; 

///end energy difference 

/////////////////////////////////////////////////////////////////// 
///////////calculation of crit vel deviation 
/////////////////////////////////////////////////////////////////// 

ffv_vel_answer = 0.0;  
ffv_vel2_answer = 0.0;  

for(ffv_i=5; ffv_i < (ffv_gsx-1); ffv_i++) { 
  for(ffv_j=5; ffv_j < (ffv_gsy-1); ffv_j++) { 
    ffv_crit_vel[ffv_i][ffv_j] = 0.19*(pow(SAND_DIAMETER_50,0.1))*log10((4.0*(ffv_n[ffv_i][ffv_j]+STILL_DEPTH+ffv_de[ffv_i][ffv_j]))/SAND_DIAMETER_90); 
    ffv_vector_velocity = pow(pow(ffv_u[ffv_i][ffv_j],2.0)+pow(ffv_v[ffv_i][ffv_j],2.0),0.5); 
    if((ffv_de[ffv_i][ffv_j]}}> 0.0) { 
      if((ffv_pt[ffv_i][ffv_j])!=0) {//if erosion hole and water point 
        ffv_vel_answer = ffv_vel_answer + (ffv_crit_vel[ffv_i][ffv_j] - ffv_vector_velocity); 
      } } 
    if((ffv_de[ffv_i][ffv_j])< 0.0) { 
      if((ffv_pt[ffv_i][ffv_j])!=0) {//if deposition mound and water point 
        ffv_vel2_answer = ffv_vel2_answer + (ffv_vector_velocity - ffv_crit_vel[ffv_i][ffv_j]); 
      } } 
  } 
}

if((ffv_vel_answer > 0.1)) { 
  if((ffv_vel2_answer > 0.1)) { 
    ffv_answer = 100.0*ffv_vel_answer*ffv_vel2_answer; //multiply the sed trans divergence with the global energy loss for answer 
  } 
  if((ffv_vel_answer > 0.1)) { 
    ffv_answer = 10.0*ffv_vel_answer; //multiply the sed trans divergence with the global energy loss for answer 
  } 
}
```c
//end if there is no penalty associated with deposition
//end if there is a penalty associated with deposition only
if (ffv_vel2_answer > 0.1) {//if there is a penalty associated with deposition only
    ffv_answer = 10.0*ffv_answer*ffv_vel2_answer; //multiply the sed trans divergence with the global energy loss for answer
} else if (ffv_vel2_answer > 0.1) {//if there is a penalty associated with erosion holes
    //do nothing
} else if (ffv_vel2_answer > 0.1) {//if there is no mound or erosion crit vel penalties
    //do nothing
} else {//if there are no mound or erosion crit vel penalties
    //do nothing
} //end calculation of crit vel deviation

/*Deletion of dynamic memory for arrays*/
delete [] ffv_vol;
delete [] ffv_bits;
delete [] ffv_energy;
delete [] ffv_histogram;
delete [] ffv_crit_vel;
delete [] ffv_stream_power;
delete [] ffv_pt;
return(ffv_answer);
}

/**************************************************************************
**************************Real Genetic Algorithm**************************
**************************************************************************
void real_genetic_algorithm(void){
    double (*rga_v)[BB];
    int rga_i = 0; //used in for loops
    int rga_j = 0; //used in for loops
    int rga_k = 0; //used in for loops
    int rga_check = 0;
    int rga_tempd = 0;
    double rga_tempdf = 0.0;
    /*Allocation of dynamic memory for arrays*/
    if ((rga_v = new double[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    /*Deletion of dynamic memory for arrays*/
    delete [] rga_v;
    return;
}

/**************************************************************************
**************************Binary Genetic Algorithm************************
**************************************************************************
void binary_genetic_algorithm(void){
    double (*bga_v)[BB];
    int bga_i = 0; //used in for loops
    int bga_j = 0; //used in for loops
    int bga_k = 0; //used in for loops
    int bga_check = 0;
    int bga_tempd = 0;
    double bga_tempdf = 0.0;
    /*Allocation of dynamic memory for arrays*/
    if ((bga_v = new double[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    /*Deletion of dynamic memory for arrays*/
    delete [] bga_v;
    return;
}
```
int mutation_type(int check6, int mutno, int m, double probmut, int testL, double l4_population[a][b], double population[a][b], double LLgene[a], double ULgene[a], int neighbour_use, double neighbour_val, double step_size)
{
    if (check6 == 1){  /*If the user predefined to use random mutation, *  
        the following is performed. */
        /********************************************************************************
        **************************Mutation Type****************************************
        *********************************************************************************/
        /************************************************************
        *******************Random Mutation***************************
        *************************************************************/
        /*Mutation can occur on any gene in any population member*/
        /****//won't work for binary GA
        mutno = random_mutation(mutno, m, probmut, testL, l4_population, population, LLgene, ULgene);
    }
    else if {check6 == 2}{ /*If the user predefined to use adjacency      
        *mutation, the following is performed. */
        /********************************************************************************
        **************************Random Adjacency Mutation***************************
        *********************************************************************************/
        /*Mutation can occur on any gene in any population member*/
        /****//won't work for binary GA
        mutno = random_adjacency_mutation(mutno, m, probmut, testL, l4_population, population, LLgene, ULgene);
    }
    else if {check6 == 3}{ /*If the user predefined to use adjacency      
        *mutation, the following is performed. */
        /********************************************************************************
        *****************************Uniform Mutation*******************************
        *********************************************************************************/
        /*Mutation can occur on any gene in any population member*/
        /****//works for binary GA only
        mutno = uniform_mutation(mutno, m, probmut, testL, l4_population, population);
    }
    else if {check6 == 4}{ /*If the user predefined to use adjacency      
        *mutation, the following is performed. */
        /********************************************************************************
        ****************************Neighbour Mutation******************************
        *********************************************************************************/
        /*Mutation can occur on any gene in any population member*/
        /****//won't work for binary GA
        mutno = neighbour_mutation(mutno, m, probmut, testL, l4_population, population, LLgene, ULgene, neighbour_use, neighbour_val);
    }
    else if {check6 == 5}{ /*If the user predefined to use adjacency      
        *mutation, the following is performed. */
        /********************************************************************************
        ********************True Adjacency Mutation*******************************
        *********************************************************************************/
        /*Mutation can occur on any gene in any population member*/
        /****//won't work for binary GA
        mutno = true_adjacency_mutation(mutno, m, probmut, testL, l4_population, population, LLgene, ULgene, step_size);
    }
    else if {check6 == 6}{ /*If the user predefined to use adjacency      
        *mutation, the following is performed. */
        /********************************************************************************
        ********************Wang and Zheng Mutation*******************************
        *********************************************************************************/
        /*
/*Mutation can occur on any gene in any population member*/

void wang_zheng_mutation(int mutno, int m, double probmut, int testL, double l4_population[a][b], double population[a][b])
{
    if (mutno > 0)
    {
        mutno = wang_zheng_mutation(mutno, m, probmut, testL, l4_population, population, LLgene, ULgene);
    }
    else{
        mistake();
    }
    return(mutno);
}

/**************************************************************************
*******************************Crossover Type*****************************
***************************************************************************/

void crossover_type(int check5, int m, double probxover, int testL, double l4_population[a][b], double population[a][b])
{
    if (check5 == 1){ /*If the user predefined to use average crossover,*
        *the following is performed                      */
        //****************************************************
        //********Average Crossover*****************************
        //****************************************************
        check5=average_crossover(check5, m, probxover, testL, l4_population, population);
    } //end ave xover
    else if (check5 == 2){ /*If the user predefined to use one point         *
        *crossover, the following is performed           */
        //****************************************************
        //********One Point Crossover**************************
        //****************************************************
        one_point_crossover(m, probxover, testL, l4_population, population);
    } //end 1pt xover
    else if (check5 == 3){ /*If the user predefined to use two point         *
        *crossover, the following is performed           */
        //****************************************************
        //********Two Point Crossover**************************
        //****************************************************
        check5=two_point_crossover(check5, m, probxover, testL, l4_population, population);
    } //end 2pt xover
    else if (check5 == 4){ /*If the user predefined to use uniform crossover,*
        *the following is performed                      */
        //****************************************************
        //*******Uniform Crossover****************************
        //****************************************************
        uniform_crossover(m, probxover, testL, l4_population, population);
    } //end uniform xover
    else if (check5 == 5){ /*If the user predefined to use average-uniform crossover,*
        *the following is performed                              */
        //****************************************************
        //********Average-Uniform Crossover**********************
        //****************************************************
        average_uniform_crossover(m, probxover, testL, l4_population, population);
    } //end ave-uni xover
    else{
        mistake();
    }
    return;
}

/**************************************************************************
*****************************Average Crossover****************************
***************************************************************************/
int average_crossover(int ac_check5, int ac_m, double ac_probxover, int ac_testL, double ac_14_population[a][b], double ac_population[a][b])
{
    int ac_i = 0; // used in for loops
    int ac_k = 0; // used in for loops
    int ac_rn1 = 0;
    int ac_rn2 = 0;
    double ac_rn3 = 0.0;
    int ac_rn5 = 0;
    int ac_rn6 = 0;
    int ac_rn7 = 0;
    int ac_pt1 = 0;
    int ac_pt2 = 0;
    int ac_pt3 = 0;
    double *ac_xoave;

    /* Allocation of dynamic memory for arrays*/
    if ((ac_xoave = new double[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }

    /* Crossover can occur m/2 times as there are only m/2 pairs available to crossover. */
    for(ac_k=0; ac_k < (ac_m/2); ac_k++){
        /* The population member numbers that will crossover are chosen randomly, along with a random percentage value that crossover will even occur. */
        ac_rn1=RandomInt(0, (ac_m-1));
        ac_rn2=RandomInt(0, (ac_m-1));
        ac_rn3=RandomDouble(0.0, 1.0);
        ac_rn5=RandomInt(0, (ac_testL-1));
        ac_rn6=RandomInt(0, (ac_testL-1));
        ac_rn7=RandomInt(0, (ac_testL-1));

        /* if the probability that crossover will occur defined by *random number generator is less than or equal to the user *defined probability, then crossover occurs. */
        while(1){
            if(ac_testL<3){
                other_mistake(2); // Average Three Point Crossover is not possible, use uniform
                ac_check5=4;
                break;
            }
            else if((ac_rn5<ac_rn6) && (ac_rn6<ac_rn7)){
                ac_pt1=ac_rn5;
                ac_pt2=ac_rn6;
                ac_pt3=ac_rn7;
                break;
            }
            else if((ac_rn5<ac_rn7) && (ac_rn7<ac_rn6)){
                ac_pt1=ac_rn5;
                ac_pt2=ac_rn7;
                ac_pt3=ac_rn6;
                break;
            }
            else if((ac_rn6<ac_rn5) && (ac_rn5<ac_rn7)){
                ac_pt1=ac_rn6;
                ac_pt2=ac_rn5;
                ac_pt3=ac_rn7;
                break;
            }
            else if((ac_rn6<ac_rn7) && (ac_rn7<ac_rn5)){
                ac_pt1=ac_rn6;
                ac_pt2=ac_rn7;
                ac_pt3=ac_rn5;
                break;
            }
            else if((ac_rn7<ac_rn6) && (ac_rn6<ac_rn5)){
                ac_pt1=ac_rn7;
                ac_pt2=ac_rn6;
                ac_pt3=ac_rn5;
                break;
            }
            else if((ac_rn7<ac_rn5) && (ac_rn5<ac_rn6)){
                ac_pt1=ac_rn7;
                ac_pt2=ac_rn5;
                ac_pt3=ac_rn6;
                break;
            }
        }
    }
}
else if((ac_rn5==ac_rn6) || (ac_rn5==ac_rn7) || (ac_rn6==ac_rn7)){
    ac_rn5=RandomInt(0, (ac_testL-1));
    ac_rn6=RandomInt(0, (ac_testL-1));
    ac_rn7=RandomInt(0, (ac_testL-1));
}
else{
    mistake();
}
}

if (ac_rn3<=ac_probxover){
    /*An average of the first variables in each of the genes*
    *to be crossed over is taken as well as an average of *
    *the second variable. The average value for variable 1*
    *then replaces the variable 1 value of the first parent*
    *member, while the child formed retains the parent *
    *value of its second variable. The opposite occurs *
    *with the second child from the second parent. *
    *
    *ie child 1 = average parent 1 and 2, parent 1 *
    *   child 2 = parent 2, average of parent 1 and 2      */
    for(ac_i=0; ac_i < ac_testL; ac_i++){
        ac_xoave[ac_i]=(ac_l4_population[ac_i][ac_rn1] + ac_l4_population[ac_i][ac_rn2])/2.0;
    }
    for(ac_i=0; ac_i < ac_pt1; ac_i++){
        ac_population[ac_i][ac_k]=ac_xoave[ac_i];
        ac_population[ac_i][ac_k+(ac_m/2)]=ac_l4_population[ac_i][ac_rn2];
    }
    for(ac_i=ac_pt1; ac_i < ac_pt2; ac_i++){
        ac_population[ac_i][ac_k]=ac_l4_population[ac_i][ac_rn1];
        ac_population[ac_i][ac_k+(ac_m/2)]=ac_xoave[ac_i];
    }
    for(ac_i=ac_pt2; ac_i < ac_pt3; ac_i++){
        ac_population[ac_i][ac_k]=ac_l4_population[ac_i][ac_rn1];
        ac_population[ac_i][ac_k+(ac_m/2)]=ac_xoave[ac_i];
    }
    for(ac_i=ac_pt3; ac_i < ac_testL; ac_i++){
        ac_population[ac_i][ac_k]=ac_l4_population[ac_i][ac_rn1];
        ac_population[ac_i][ac_k+(ac_m/2)]=ac_xoave[ac_i];
    }
}
else if (ac_rn3>ac_probxover){
    for(ac_i = 0; ac_i < ac_testL; ac_i++){                   /*filling the population array*/
        ac_population[ac_i][ac_k] =  ac_l4_population[ac_i][ac_rn1];
        ac_population[ac_i][ac_k+(ac_m/2)] =  ac_l4_population[ac_i][ac_rn2];
    }
}
else {
    mistake();
}

/*Deletion of dynamic memory for arrays*/
delete [] ac_xoave;
return(ac_check5);
}

**************************************************************************
***************************One-Point Crossover*****************************
**************************************************************************
void one_point_crossover(int opc_m, double opc_probxover, int opc_testL, double opc_l4_population[a][b], double opc_population[a][b][]{
    int opc_i = 0; //used in for loops
    int opc_j = 0; //used in for loops
    int opc_k = 0; //used in for loops
    int opc_rn1 = 0;
    int opc_rn2 = 0;
    double opc_rn3 = 0.0;
    int opc_rn5 = 0;

    /*Crossover can occur m/2 times as there are only m/2 pairs */
*/
for(opc_k=0; opc_k < (opc_m/2); opc_k++)
{
/*The opc_population member numbers that will crossover are
chosen randomly, along with a random percentage value that
will crossover will even occur. */

opc_rn1=RandomInt(0, (opc_m-1));
opc_rn2=RandomInt(0, (opc_m-1));
opc_rn3=RandomDouble(0.0, 1.0);
opc_rn5=RandomInt(0, (opc_testL-1));

/*If the probability that crossover will occur defined by
random number generator is less than or equal to the user
defined probability, then crossover occurs. */

if (opc_rn3<=opc_probxover){
/*Child 1 is given the value of variable 1 from parent 1
and the value of variable 2 from parent 2. Child 2 is
* given the value of variable 1 from parent 2 and the
* value of variable 2 from parent 1.
* ie child 1 = parent 1, parent 2
* child 2 = parent 2, parent 1*/

for(opc_i=0; opc_i < opc_rn5; opc_i++){
opc_population[opc_i][opc_k] = opc_l4_population[opc_i][opc_rn1];
opc_population[opc_i][(opc_k+(opc_m/2))] = opc_l4_population[opc_i][opc_rn2];
}

for(opc_i=opc_rn5; opc_i < opc_testL; opc_i++){
opc_population[opc_i][opc_k] = opc_l4_population[opc_i][opc_rn2];
opc_population[opc_i][(opc_k+(opc_m/2))] = opc_l4_population[opc_i][opc_rn1];
}

/*If crossover does not occur then the children have the
* genes as the parents. */
else if (opc_rn3>opc_probxover){
for(opc_i = 0; opc_i < opc_testL; opc_i++){/*filling the opc_population array*/
opc_population[opc_i][opc_k] = opc_l4_population[opc_i][opc_rn1];
opc_population[opc_i][(opc_k+(opc_m/2))] = opc_l4_population[opc_i][opc_rn2];
}
else {
mistake();
}

return;
}

/*******************************************************************************
****************************Two-Point Crossover****************************
*******************************************************************************/

int two_point_crossover(int tpc_check5, int tpc_m, double tpc_probxover, int tpc_testL, double
tpc_l4_population[a][b], double tpc_population[a][b])
{
int tpc_i = 0;//used in for loops
int tpc_k = 0;//used in for loops
int tpc_rn1 = 0;
int tpc_rn2 = 0;
double tpc_rn3 = 0.0;
int tpc_rn5 = 0;
int tpc_rn6 = 0;
int tpc_pt1 = 0;
int tpc_pt2 = 0;

/*Crossover can occur m/2 times as there are only m/2 pairs
available to crossover. */

for(tpc_k=0; tpc_k < (tpc_m/2); tpc_k++)
/*The tpc_population member numbers that will crossover are
chosen randomly, along with a random percentage value that
will crossover will even occur. */

tpc_rn1=RandomInt(0, (tpc_m-1));
tpc_rn2=RandomInt(0, (tpc_m-1));
tpc_rn3=RandomDouble(0.0, 1.0);
tpc_rn5=RandomInt(0, (tpc_testL-1));
tpc_rn6=RandomInt(0, (tpc_testL-1));

while(1){
if (tpc_testL<2){
other_mistake(3); //Two point crossover is not possible, use uniform crossover
    tpc_check5=4;
    break;
}
else if (tpc_rn5<tpc_rn6){
    tpc_pt1 = tpc_rn5;
    tpc_pt2 = tpc_rn6;
    break;
}
else if (tpc_rn5>tpc_rn6){
    tpc_pt1 = tpc_rn6;
    tpc_pt2 = tpc_rn5;
    break;
}
else if (tpc_rn5 == tpc_rn6){
    tpc_rn5=RandomInt(0, (tpc_testL-1));
    tpc_rn6=RandomInt(0, {tpc_testL-1});
}
else{
    mistake();
    other_mistake(4);  //Assuming xover points
    tpc_pt1 = 1;
    tpc_pt2 = 2;
    break;
}
}

/*If the probability that crossover will occur defined by   *
*random number generator is less than or equal to the user *
*defined probability, then crossover occurs.               */
if (tpc_rn3<=tpc_probxover){
    /*Child 1 is given a mixture of values from parent 1    *
    *and parent 2. Child 2 is given a mixture of values   *
    *from parent 2 and parent 1.                          *
    *                                                         *
    *ie child 1 = parent 1, parent 2, parent 1             *
    * child 2 = parent 2, parent 1, parent 2             */
    for(tpc_i=0; tpc_i < tpc_pt1; tpc_i++){
        tpc_population[tpc_i][tpc_k] = tpc_l4_population[tpc_i][tpc_rn1];
        tpc_population[tpc_i][(tpc_k+(tpc_m/2))] = tpc_l4_population[tpc_i][tpc_rn2];
    }
    for(tpc_i=tpc_pt1; tpc_i < tpc_pt2; tpc_i++){
        tpc_population[tpc_i][tpc_k] = tpc_l4_population[tpc_i][tpc_rn2];
        tpc_population[tpc_i][(tpc_k+(tpc_m/2))] = tpc_l4_population[tpc_i][tpc_rn1];
    }
    for(tpc_i=tpc_pt2; tpc_i < tpc_testL; tpc_i++){
        tpc_population[tpc_i][tpc_k] = tpc_l4_population[tpc_i][tpc_rn1];
        tpc_population[tpc_i][(tpc_k+(tpc_m/2))] = tpc_l4_population[tpc_i][tpc_rn2];
    }
}

/*If crossover does not occur then the children have the    *
*genes as the parents.                                     */
else if (tpc_rn3>tpc_probxover){
    for(tpc_i = 0; tpc_i < tpc_testL; tpc_i++){
        /*filling the tpc_population array*/
        tpc_population[tpc_i][tpc_k] = tpc_l4_population[tpc_i][tpc_rn1];
        tpc_population[tpc_i][(tpc_k+(tpc_m/2))] = tpc_l4_population[tpc_i][tpc_rn2];
    }
}
else {
    mistake();
}
}
return(tpc_check5);

/**************************************************************************
*****************************Uniform Crossover*****************************
**************************************************************************/
void uniform_crossover(int uc_m, double uc_probxover, int uc_testL, double uc_l4_population[a][b],
double uc_population[a][b]){
    int *uc_mask;
    int uc_i = 0;//used in for loops
    int uc_k = 0;//used in for loops
    int uc_rn1 = 0;
    int uc_rn2 = 0;
double uc_rn3 = 0.0;

/*Allocation of dynamic memory for arrays*/
if ((uc_mask = new int[a]) == NULL) {
  printf("Memory Allocation Failure for Array ar1\n");
  exit(0);
}

for(uc_k=0; uc_k < (uc_m/2); uc_k++){
  uc_rn1=RandomInt(0, (uc_m-1));
  uc_rn2=RandomInt(0, (uc_m-1));
  uc_rn3=RandomDouble(0.0, 1.0);

  if (uc_rn3<uc_probxover){
    for(uc_i=0; uc_i < uc_testL; uc_i++){
      uc_mask[uc_i] = RandomInt(0, 1);
    }
    for(uc_i=0; uc_i < uc_testL; uc_i++){
      uc_population[uc_i][uc_k] = uc_l4_population[uc_i][uc_rn1];
      uc_population[uc_i][(uc_k+(uc_m/2))] = uc_l4_population[uc_i][uc_rn2];
    }
  } else if (uc_rn3==uc_probxover){
    for(uc_i=0; uc_i < uc_testL; uc_i++){
      uc_population[uc_i][uc_k] = uc_l4_population[uc_i][uc_rn2];
      uc_population[uc_i][(uc_k+(uc_m/2))] = uc_l4_population[uc_i][uc_rn1];
    }
  } else{
    mistake();
  }
}

else if (uc_rn3>uc_probxover){
  for(uc_i=0; uc_i < uc_testL; uc_i++){
    uc_population[uc_i][uc_k] = uc_l4_population[uc_i][uc_rn1];
    uc_population[uc_i][(uc_k+(uc_m/2))] = uc_l4_population[uc_i][uc_rn2];
  }
}
else{
  mistake();
}

/*Deletion of dynamic memory for arrays*/
delete [] uc_mask;
return;

**************************************************************************
************************Average Uniform Crossover*************************
**************************************************************************

void average_uniform_crossover(int auc_m, double auc_probxover, int auc_testL, double auc_l4_population[a][b], double auc_population[a][b]){ double *auc_xoave;
  int *auc_mask;

  int auc_i = 0; //used in for loops
  int auc_j = 0; //used in for loops
  int auc_k = 0; //used in for loops
  int auc_rn1 = 0;
  int auc_rn2 = 0;
  double auc_rn3 = 0.0;

  /*Allocation of dynamic memory for arrays*/
  if ((auc_xoave = new double[a]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
  }
  if ((auc_mask = new int[a]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
  }

  for(auc_k=0; auc_k < (auc_m/2); auc_k++){
    auc_rn1=RandomInt(0, (auc_m-1));
    auc_rn2=RandomInt(0, (auc_m-1));
    auc_rn3=RandomDouble(0.0, 1.0);
    if (auc_rn3<auc_probxover){
      for(auc_i=0; auc_i < auc_testL; auc_i++){
        auc_mask[uc_i] = RandomInt(0, 1);
      }
      for(auc_i=0; auc_i < auc_testL; auc_i++){
        auc_population[uc_i][uc_k] = auc_l4_population[uc_i][auc_rn1];
        auc_population[uc_i][(uc_k+(auc_m/2))] = auc_l4_population[uc_i][auc_rn2];
      }
    } else if (auc_rn3==auc_probxover){
      for(auc_i=0; auc_i < auc_testL; auc_i++){
        auc_population[uc_i][uc_k] = auc_l4_population[uc_i][auc_rn2];
        auc_population[uc_i][(uc_k+(auc_m/2))] = auc_l4_population[uc_i][auc_rn1];
      }
    } else{
      mistake();
    }
  }
}
auc_xoave[auc_i] = (auc_l4_population[auc_i][auc_rn1] + auc_l4_population[auc_i][auc_rn2]) / 2.0;
}
for (auc_i = 0; auc_i < auc_testL; auc_i++) {
    /*filling the auc_population array*/
    auc_mask[auc_i] = RandomInt(0, 6);
}
for (auc_i = 0; auc_i < auc_testL; auc_i++) {
    /*filling the auc_population array*/
    if (auc_mask[auc_i] == 0) {
        auc_population[auc_i][auc_k] = auc_l4_population[auc_i][auc_rn1];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_l4_population[auc_i][auc_rn2];
    } else if (auc_mask[auc_i] == 1) {
        auc_population[auc_i][auc_k] = auc_xoave[auc_i];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_xoave[auc_i];
    } else if (auc_mask[auc_i] == 2) {
        auc_population[auc_i][auc_k] = auc_xoave[auc_i];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_l4_population[auc_i][auc_rn2];
    } else if (auc_mask[auc_i] == 3) {
        auc_population[auc_i][auc_k] = auc_l4_population[auc_i][auc_rn1];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_xoave[auc_i];
    } else if (auc_mask[auc_i] == 4) {
        auc_population[auc_i][auc_k] = auc_l4_population[auc_i][auc_rn1];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_l4_population[auc_i][auc_rn2];
    } else if (auc_mask[auc_i] == 5) {
        auc_population[auc_i][auc_k] = auc_xoave[auc_i];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_xoave[auc_i];
    } else if (auc_mask[auc_i] == 6) {
        auc_population[auc_i][auc_k] = auc_xoave[auc_i];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_l4_population[auc_i][auc_rn2];
    } else { mistake(); }
}
else if (auc_rn3 > auc_probxover) {
    for (auc_i = 0; auc_i < auc_testL; auc_i++) {
        /*filling the auc_population array*/
        auc_population[auc_i][auc_k] = auc_l4_population[auc_i][auc_rn1];
        auc_population[auc_i][auc_k + (auc_m/2)] = auc_l4_population[auc_i][auc_rn2];
    }
} else { mistake(); }
/*Deletion of dynamic memory for arrays*/
delete [] auc_mask;
delete [] auc_xoave;
return;
}
if (rm_rn3<=rm_probmut) {
    rm_mutno+= rm_mutno + 1; /*counts how many mutations occur */
    /*If the mutation occurs, the new random value * 
    *generated is within the specified limits of this * 
    *variable. * 
    *ie child(gene) = random number */
    rm_rn4=RandomDouble(rm_LLgene[rm_i], rm_ULgene[rm_i]);
    rm_l4_population[rm_i][rm_j] = rm_rn4;
    /*If the random probability generated is greater than * 
    *the user specified value, then the child gene is the * 
    *same as the parent gene. * 
    *ie child(gene) = parent(gene) */
    else if (rm_rn3>rm_probmut){
        rm_l4_population[rm_i][rm_j] = rm_population[rm_i][rm_j];
    }
    else{
        mistake();
    }
}
return(rm_mutno);

int random_adjacency_mutation(int ram_mutno, int ram_m, double ram_probmut, int ram_testL, double ram_l4_population[a][b], double ram_population[a][b], double ram_LLgene[a], double ram_ULgene[a])
{ //returns mutation number
    double *ram_limmid;
    int ram_i = 0;//used in for loops
    int ram_j = 0;//used in for loops
    int ram_rn1 = 0;
    double ram_rn3 = 0.0;
    double ram_rn4 = 0.0;
    double ram_rn8 = 0.0;
    double ram_step = 0.0;
    /*Allocation of dynamic memory for arrays*/
    if ((ram_limmid = new double[a]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    /*Mutation can occur on any gene in any population member*/
    //****/"won't work for binary GA
    for(ram_i=0; ram_i < ram_testL; ram_i++){
        ram_limmid[ram_i]=ram_LLgene[ram_i] + ((ram_ULgene[ram_i] - ram_LLgene[ram_i])/2.0);
    }
    for(ram_j=0; ram_j < ram_m; ram_j++){
        for(ram_i = 0; ram_i < ram_testL; ram_i++){
            /*A random value is calculated to determine whether * 
            *mutation occurs for that gene. */
            ram_rn1=RandomInt(0, 1);
            ram_rn3=RandomDouble(0.0, 1.0);
            ram_rn8=RandomDouble(DIVL, DIVU);
            /*If the value generated is less than or equal to the * 
            *user defined probability of mutation, then mutation * 
            *occurs as below. */
            if (ram_rn3<=ram_probmut){
                ram_rn4=RandomDouble(rm_LLgene[rm_i], rm_ULgene[rm_i]);
                if(ram_rn4<=ram_limmid[ram_i]){  
                    ram_step=(ram_ULgene[ram_i] - ram_LLgene[ram_i])/ram_rn8;
                }
                else if(ram_rn4>ram_limmid[ram_i]){  
                    ram_step=(-1.0)*((ram_ULgene[ram_i] - ram_LLgene[ram_i])/ram_rn8);
                }
            }
        }
    }
/*If the mutation occurs in variable 1, the new gene is calculated using a predefined ram_step size (SS1) and a random number (RND) within specified limits namely (0,1). ie child(gene) = parent(gene) + SS1(2*RND - 1) */

else{
  mistake();
  ram_l4_population[ram_i][ram_j] = ram_population[ram_i][ram_j] + ram_step*(2.0*ram_rn4-1.0);
  if ((ram_l4_population[ram_i][ram_j]<ram_LLgene[ram_i]) || (ram_l4_population[ram_i][ram_j]>ram_ULgene[ram_i])){
    ram_l4_population[ram_i][ram_j] = ram_rn4;
  }
}

/*If the random probability generated is greater than the user specified value, then the child gene is the same as the parent gene. ie child(gene) = parent(gene)*/
else if (ram_rn3>ram_probmut){
  ram_l4_population[ram_i][ram_j] = ram_population[ram_i][ram_j];
} else{
  mistake();
}

/*Deletion of dynamic memory for arrays*/
delete [] ram_limmid;
return(ram_mutno);
}

**************************************************************************
*****************************Uniform Mutation****************************
**************************************************************************

int uniform_mutation(int um_mutno, int um_m, double um_probmut, int um_testL, double um_l4_population[a][b], double um_population[a][b]){

  //returns mutation number
  int um_i = 0;//used in for loops
  int um_j = 0;//used in for loops
  double um_rn3 = 0.0;

  for(um_j=0; um_j < um_m; um_j++){
    for(um_i = 0; um_i < um_testL; um_i++){
      um_rn3=RandomDouble(0.0, 1.0);
      if (um_rn3<=um_probmut){
        um_mutno= um_mutno + 1;
        if (um_population[um_i][um_j] == 0.0){
          um_l4_population[um_i][um_j] = 1.0;
        } else if (um_population[um_i][um_j] == 1.0){
          um_l4_population[um_i][um_j] = 0.0;
        } else {
          printf("There has been a mistake 5 ");
        }
      } else if (um_rn3>um_probmut){
        um_l4_population[um_i][um_j] = um_population[um_i][um_j];
      } else{
        printf("There has been a mistake 6 ");
      }
    }
  }

  return(um_mutno);
}

**************************************************************************
*****************************Neighbour Mutation****************************
**************************************************************************

int neighbour_mutation(int nm_mutno, int nm_m, double nm_probmut, int nm_testL, double nm_l4_population[a][b], double nm_population[a][b], double nm_LLgene[a], double nm_ULgene[a], int nm_neighbour_use, double nm_neighbour_val){

  //returns mutation number

  for(nm_j=0; nm_j < nm_m; nm_j++){
    for(nm_i = 0; nm_i < nm_testL; nm_i++){
      if (nm_population[nm_i][nm_j] == 0.0){
        nm_l4_population[nm_i][nm_j] = 1.0;
      } else if (nm_population[nm_i][nm_j] == 1.0){
        nm_l4_population[nm_i][nm_j] = 0.0;
      } else {
        printf("There has been a mistake 5 ");
      }
    } /*for n_j*/
  } /*for n_i*/

  return(nm_mutno);
}
int nm_i = 0; //used in for loops
int nm_j = 0; //used in for loops
int nm_rn1 = 0;
double nm_rn3 = 0.0;
double nm_rn8 = 0.0;
double nm_step = 0.0;

/*Mutation can occur on any gene in any population member*/
/***/ //won't work for binary GA

for(nm_j=0; nm_j < nm_m; nm_j++){
  for(nm_i = 0; nm_i < nm_testL; nm_i++){
    /*A random value is calculated to determine whether */
    /*mutation occurs for that gene. */
    nm_rn1=RandomInt(0, 1);
    nm_rn3=RandomDouble(0.0, 1.0);

    /*If the value generated is less than or equal to the */
    /*user defined probability of mutation, then mutation */
    /*occurs as below. */
    if (nm_rn3<=nm_probmut){
      if(nm_neighbour_use == 0){
        nm_rn8=RandomDouble(2.0, nm_neighbour_val);
      }
      else if(nm_neighbour_use == 1){
        nm_rn8=nm_neighbour_val;
      }
      else{
        mistake();
      }
      nm_mutno= nm_mutno + 1; /*counts how many mutations occur */
      if(nm_rn==0){
        nm_l4_population[nm_i][nm_j] = nm_population[nm_i][nm_j] + nm_step;
      }
      else if(nm_rn==1){
        nm_l4_population[nm_i][nm_j] = nm_population[nm_i][nm_j] - nm_step;
      }
      else{
        mistake();
      }
    }
    /*If the random probability generated is greater than */
    /*the user specified value, then the child gene is the */
    /*same as the parent gene. */
    /*ie child(gene) = parent(gene) */
    else if (nm_rn3>nm_probmut){
      nm_l4_population[nm_i][nm_j] = nm_population[nm_i][nm_j];
    }
    else{
      mistake();
    }
  }
}

return(nm_mutno);
}

**************************************************************************
**************************True Adjacency Mutation*************************
**************************************************************************

int true_adjacency_mutation(int tam_mutno, int tam_m, double tam_probmut, int tam_testL, double	am_l4_population[a][b], double tam_population[a][b], double tam_LLgene[a], double tam_ULgene[a], double tam_step_size){
  //returns mutation number
  int tam_i = 0; //used in for loops
  int tam_j = 0; //used in for loops
  double tam_rn3 = 0.0;
  double tam_rn4 = 0.0;
  double tam_rn8 = 0.0;

  /*Mutation can occur on any gene in any population member*/
  /***/ //won't work for binary GA

  for(tam_j=0; tam_j < tam_m; tam_j++){
    for(tam_i = 0; tam_i < tam_testL; tam_i++){
/*A random value is calculated to determine whether mutation occurs for that gene.*/
tam_rn3=RandomDouble(0.0, 1.0);

/*If the value generated is less than or equal to the user defined probability of mutation, then mutation occurs as below.*/
if (tam_rn3<=tam_probmut){
    tam_mutno= tam_mutno + 1; /*counts how many mutations occur*/
tam_rn8=RandomDouble(0.0, 1.0);

    /*If the mutation occurs in variable 1, the new gene is calculated using a predefined step size (SS1) and a random number (RND) within specified limits namely (0,1). ie child(gene) = parent(gene) + SS1(2* RND - 1)*/
    tam_l4_population[tam_i][tam_j] = tam_population[tam_i][tam_j] + tam_step_size*((2.0*tam_rn8)-1.0);
    if ((tam_l4_population[tam_i][tam_j]<tam_LLgene[tam_i]) || (tam_l4_population[tam_i][tam_j]>tam_ULgene[tam_i])){
        tam_rn4=RandomDouble(tam_LLgene[tam_i], tam_ULgene[tam_i]);
        tam_l4_population[tam_i][tam_j] = tam_rn4;
    }
}
else if (tam_rn3>tam_probmut){
    tam_l4_population[tam_i][tam_j] = tam_population[tam_i][tam_j];
}
else{
    mistake();
}
}
return(tam_mutno);

/**************************************************************************
*******************************Wang and Zheng Mutation**************************
**************************************************************************/
wzm_14_population[wzm_i][wzm_j] = wzm_population[wzm_i][wzm_j] + (2.0*(wzm_rn8-0.5));
if (wzm_14_population[wzm_i][wzm_j] > wzm_ULgene[wzm_i]) {
    wzm_14_population[wzm_i][wzm_j] = wzm_ULgene[wzm_i];
}
else if (wzm_rn4<wzm_LLgene[wzm_i]) {
    wzm_14_population[wzm_i][wzm_j] = wzm_rn4;
}
/*If the random probability generated is greater than the user specified value, then the child gene is the same as the parent gene. ie child(gene) = parent(gene)*/
else if (wzm_rn3<wzm_probmut) {
    wzm_14_population[wzm_i][wzm_j] = wzm_population[wzm_i][wzm_j];
} else {
    mistake();
}
return(wzm_mutno);
}
/**************************************************************************
**************************Actual Fitness Value**************************
**************************************************************************/void actual_fitness_value(void){
    double (*afv_v)[BB];
    // char afv_temps[50];
    int afv_i = 0;//used in for loops
    int afv_j = 0;//used in for loops
    int afv_k = 0;//used in for loops
    int afv_check = 0;
    int afv_tempd = 0;
    double afv_templf = 0.0;
    /*Allocation of dynamic memory for arrays*/
    if ((afv_v = new double[AA][BB]) == NULL) {
        printf("Memory Allocation Failure for Array ar1\n");
        exit(0);
    }
    /*Deletion of dynamic memory for arrays*/
    delete [] afv_v;
    return;
}
/**************************************************************************
*********************Initialise Flat Bed Function*************************
**************************************************************************/void initialise_flat_bed(double ifb_qx[a][b], double ifb_qy[a][b], double ifb_flowx[a][b], double ifb_flowy[a][b]){double (*ifb_v)[BB];
    double (*ifb_u)[BB];
    double (*ifb_n)[BB];
    double (*ifb_de)[BB];
    double (*ifb_vol)[BB];
    double (*ifb_energy)[BB];
    int (*ifb_pt)[BB];
    int *ifb_bits;
    double (*ifb_crit_vel)[BB];//critical velocity
    double (*ifb_q_bedload)[BB];//volumetric bedload transport rate
    double (*ifb_q_bedloadu)[BB];//volumetric bedload transport rate
    double (*ifb_q_bedloadv)[BB];//volumetric bedload transport rate
    int ifb_i;//used in for loops
    int ifb_j;//used in for loops
    int ifb_k;//used in for loops
    int ifb_gx;//grid size of x
    int ifb_gy;//grid size of y
    int ifb_tempt;
    double ifb_tempf;
    char ifb_temps[50];
    /*unixin*/
    pid_t pid;//used in calling hydra3jo from unix
    /*unixin*/
    int status;//used in calling hydra3jo from unix
/* Allocation of dynamic memory for arrays */
if ((ifb_v = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_u = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_n = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_de = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_vol = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_bits = new int[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_energy = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_crit_vel = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_q_bedload = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_q_bedloadu = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_q_bedloadv = new double[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
if ((ifb_pt = new int[AA][BB]) == NULL) {
    printf("Memory Allocation Failure for Array ar1\n");
    exit(0);
}
ifb_tempd = 0;
ifb_tempfl = 0.0;
for(ifb_i=0; ifb_i < a; ifb_i++){
    ifb_bits[ifb_i][ifb_j] = 0;
}
for(ifb_j=0; ifb_j < b; ifb_j++){
    for(ifb_i=0; ifb_i < AA; ifb_i++){
        ifb_v[ifb_i][ifb_j] = 0.0;
        ifb_u[ifb_i][ifb_j] = 0.0;
        ifb_n[ifb_i][ifb_j] = 0.0;
        ifb_de[ifb_i][ifb_j] = 0.0;
        ifb_vol[ifb_i][ifb_j] = 0.0;
        ifb_energy[ifb_i][ifb_j] = 0.0;
        ifb_crit_vel[ifb_i][ifb_j] = 0.0;
        ifb_q_bedload[ifb_i][ifb_j] = 0.0;
        ifb_q_bedloadu[ifb_i][ifb_j] = 0.0;
        ifb_q_bedloadv[ifb_i][ifb_j] = 0.0;
        ifb_pt[ifb_i][ifb_j] = 0;
    }
}
grid_size3f = fopen("g_size.txt", "r");
if( grid_size3f == NULL ) { other_mistake(1); // Can't open file
    printf("\ngrid_size\n");
}
else{
    fscanf(grid_size3f, "%d", &ifb_gsx);
    fscanf(grid_size3f, "%d", &ifb_gsy);
    fclose(grid_size3f);
}
grid_v3f = fopen("g_v.txt", "r");
grid_u3f = fopen("g_u.txt", "r");
grid_n3f = fopen("g_n.txt", "r");
if( grid_n3f == NULL ){
    other_mistake(1); //Can't open file
    printf("
grid_v");
}
else{
    for(ifb_j=(ifb_gsy-1); ifb_j > (-1); ifb_j--){
        for(ifb_i=0; ifb_i < ifb_gsx; ifb_i++){
            fscanf(grid_n3f, "%lf", &ifb_n[ifb_i][ifb_j]);
        }
    }
}
fclose(grid_n3f);
grid_bits3f = fopen("g_bits.txt", "r");
if( grid_bits3f == NULL ){
    other_mistake(1); //Can't open file
    printf("
grid_bits");
}
else{
    fscanf(grid_bits3f, "%d ", &ifb_k);
    for(ifb_i=0; ifb_i < ifb_k; ifb_i++){
        fscanf(grid_bits3f, "%d ", &ifb_bits[ifb_i]);
    }
    fclose(grid_bits3f);
}
ifb_i = 0;
ifb_j = (ifb_gsy - 1);
ifb_k = 0;
grid_type3f = fopen("g_type.txt", "r");
if( grid_type3f == NULL ){
    other_mistake(1); //Can't open file
    printf("
grid_type");
}
else{
    while(1){
        fscanf(grid_type3f, "%d", &ifb_pt[ifb_i][ifb_j]);
        if(ifb_pt[ifb_i][ifb_j] == 0){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 1){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 2){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 4){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 6){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 8){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 10){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 12){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 14){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 16){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 20){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 24){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 28){
            break;
        }
        else if(ifb_pt[ifb_i][ifb_j] == 32){
            ifb_de[ifb_i][ifb_j] = 0.0;// ifb_pop[ifb_k][ifb_no];
            ifb_k = ifb_k + 1;
        }
    }
}
else if(ifb_pt[ifb_i][ifb_j] == 34){
    ifb_de[ifb_i][ifb_j] = 0.0;  // ifb_pop[ifb_k][ifb_no];
    ifb_k = ifb_k + 1;
}
else if(ifb_pt[ifb_i][ifb_j] == 40){
}
else if(ifb_pt[ifb_i][ifb_j] == 42){
    ifb_de[ifb_i][ifb_j] = 0.0;  // ifb_pop[ifb_k][ifb_no];
    ifb_k = ifb_k + 1;
}
else if(ifb_pt[ifb_i][ifb_j] == 48){
    ifb_de[ifb_i][ifb_j] = 0.0;  // ifb_pop[ifb_k][ifb_no];
    ifb_k = ifb_k + 1;
}
else if(ifb_pt[ifb_i][ifb_j] == 56){
}
else{
    mistake();
    printf("1");
}

ifb_i = ifb_i + 1;
if ((ifb_i == ifb_gsx) || (ifb_j == ifb_gsy)) {  // need to add extra if greater than or less than 10, 100,
    ifb_i = 0;
    ifb_j = ifb_j - 1;
}
fclose(grid_type3f);

ddepthf = fopen( "deltadepth.dat", "w+" );
if (ddepthf == NULL) {
    other_mistake(1);  // Can't open file
}
else{
    for(ifb_i=0; ifb_i < ifb_gsx; ifb_i++){
        for(ifb_j=0; ifb_j < ifb_gsy; ifb_j++)// need to add extra if greater than or less than 10, 100,
            if (ifb_de[ifb_i][ifb_j] < 0.0){
                fprintf(ddepthf, "%.3f ", ifb_de[ifb_i][ifb_j]);
            }
            else if(ifb_de[ifb_i][ifb_j] >= 0.0 && (ifb_de[ifb_i][ifb_j] < 10.0)){
                fprintf(ddepthf, " %.3f ", ifb_de[ifb_i][ifb_j]);
            }
            else if(ifb_de[ifb_i][ifb_j] >= 10.0){
                fprintf(ddepthf, "%.3f ", ifb_de[ifb_i][ifb_j]);
            }
            else{
                mistake();
                printf("2");
            }
        fprintf (ddepthf, 
        close(ddepthf);
        // call hydra3jo.exe here
    }
}
fclose(grid_type3f);

/* unixin*/
if ((pid = fork()) == 0) execl("hydra3jo",NULL);
/* unixin*/
waitpid(pid, NULL, 0);
/* unixex*/   /****************/   spawnl(0,"hydra3jo.exe","help",NULL);
small_arcof = fopen( "inlet.out", "r" );
if (small_arcof == NULL) {
    other_mistake(1);  // Can't open file
}
else{
    fscanf(small_arcof, "%lf", &ifb_templf);//1020.398
    fscanf(small_arcof, "%d", &ifb_tempd);//38
    fscanf(small_arcof, "%d", &ifb_tempd);//12
    fscanf(small_arcof, "%lf", &ifb_templf);//0.1000000
    fscanf(small_arcof, "%lf", &ifb_templf);//0.1000000
    fscanf(small_arcof, "%s", &ifb_temps);//u
    for(ifb_i=0; ifb_i < ifb_gsx; ifb_i++){
        for(ifb_j=0; ifb_j < ifb_gsy; ifb_j++) {
            fscanf(small_arcof, "%lf", &ifb_u[ifb_i][ifb_j]);
        }
    }
    fscanf(small_arcof, "%s", &ifb_temps);//v
    for(ifb_i=0; ifb_i < ifb_gsx; ifb_i++){
        for(ifb_j=0; ifb_j < ifb_gsy; ifb_j++) {
            fscanf(small_arcof, "%lf", &ifb_v[ifb_i][ifb_j]);
        }
    }
Appendix C  Morphology in Coastal and River Environments

```c
fprintf(small_arcof, "%s", &ifb_temps);//e
for(ifb_i=0; ifb_i < ifb_gsx; ifb_i++){
    for(ifb_j=0; ifb_j < ifb_gsy; ifb_j++){
        fscanf(small_arcof, "%lf", &ifb_n[ifb_i][ifb_j]);
    }
}
fclose(small_arcof);
}
for(ifb_i=0; ifb_i < ifb_gsx; ifb_i++){
    for(ifb_j=0; ifb_j < ifb_gsy; ifb_j++){
        ifb_crit_vel[ifb_i][ifb_j] = 0.19*(((SAND_DIAMETER_50)/0.1))log10((4.0*(ifb_n[ifb_i][ifb_j]+STILL_DEPTH+ifb_de[ifb_i][ifb_j]))/(SAND
        _DIAMETER_90));
        ifb_vector_velocity = pow(pow(ifb_u[ifb_i][ifb_j],2.0)+pow(ifb_v[ifb_i][ifb_j],2.0)), 0.5);
        if(ifb_crit_vel[ifb_i][ifb_j]<=ifb_vector_velocity){//if critical velocity exceeded can calculate
            ifb_q_bedload[ifb_i][ifb_j] = ifb_q_bedload[ifb_i][ifb_j]*((ifb_u[ifb_i][ifb_j])/(pow((RELATIVE_DENSITY-
            1.0)*GRAVITY*SAND_DIAMETER_50),0.5)),(2.4))*(pow((SAND_DIAMETER_50/(ifb_n[ifb_i][ifb_j]+STILL_DEPTH+ifb_-
            de[ifb_i][ifb_j])),1.2));
            ifb_q_bedloadu[ifb_i][ifb_j] = ifb_q_bedload[ifb_i][ifb_j]*((ifb_u[ifb_i][ifb_j])/(ifb_vector_velocity));
            ifb_q_bedloadv[ifb_i][ifb_j] = ifb_q_bedload[ifb_i][ifb_j]*((ifb_v[ifb_i][ifb_j])/(ifb_vector_velocity));
        }
        else if(ifb_crit_vel[ifb_i][ifb_j]>ifb_vector_velocity){//if critical velocity exceeded can
calculate transport
            ifb_q_bedload[ifb_i][ifb_j] = 0.0;
            ifb_q_bedloadu[ifb_i][ifb_j] = 0.0;
            ifb_q_bedloadv[ifb_i][ifb_j] = 0.0;
        }
        else{
            mistake();
        }
    }
}
for(ifb_i=0; ifb_i < (ifb_gsx-1); ifb_i++){
    for(ifb_j=0; ifb_j < (ifb_gsy-1); ifb_j++){
        ifb_qx[ifb_i][ifb_j] = ifb_q_bedloadu[ifb_i][ifb_j] - ifb_q_bedloadu[ifb_i][ifb_j];
        ifb_qy[ifb_i][ifb_j] = ifb_q_bedloadv[ifb_i][ifb_j] - ifb_q_bedloadv[ifb_i][ifb_j];
    }
}
//write the best configuration to file
initialvf = fopen( "initialv.txt", "w+");  //Can't open file
initialuf = fopen( "initialu.txt", "w+");
initialnf = fopen( "initialn.txt", "w+");
initialdef = fopen( "initialde.txt", "w+");
initialqxf = fopen( "initialqx.txt", "w+");
initialqyf = fopen( "initialqy.txt", "w+");
if( initialvf == NULL ) { other_mistake(1); //Can't open file
}
else{
    for(ifb_j=(ifb_gsy-1); ifb_j > (-1); ifb_j--){
        fprintf(initialvf, "%e ", ifb_v[ifb_i][ifb_j]);
        fprintf(initialuf, "%e ", ifb_u[ifb_i][ifb_j]);
        fprintf(initialnf, "%e ", ifb_n[ifb_i][ifb_j]);
        fprintf(initialdef, "%e ", ifb_de[ifb_i][ifb_j]);
        fprintf(initialqxf, "%e ", ifb_qx[ifb_i][ifb_j]);
        fprintf(initialqyf, "%e ", ifb_qy[ifb_i][ifb_j]);
    }
    fprintf(initialvf, "\n");
    fprintf(initialuf, "\n");
    fprintf(initialnf, "\n");
    fprintf(initialdef, "\n");
    fprintf(initialqxf, "\n");
    fprintf(initialqyf, "\n");
}
fclose(initialvf);
fclose(initialuf);
fclose(initialnf);
fclose(initialdef);
fclose(initialqxf);
fclose(initialqyf);
} //end write the best configurations to file

/*Deletion of dynamic memory for arrays*/
delete [] ifb_v;
delete [] ifb_u;
```
delete [] ifb_n;
delete [] ifb_de;
delete [] ifb_vol;
delete [] ifb_bits;
delete [] ifb_energy;
delete [] ifb_crit_vel;
delete [] ifb_q_bedload;
delete [] ifb_q_bedloadu;
delete [] ifb_q_bedloadv;
delete [] ifb_pt;
return;
}

/**************************************************************************
**************Save results under different names Function*****************
**************************************************************************

void change_results_name(int no_loops){
    int copy_results;
    char old1[25] = "allmin.txt";
    char old3[25] = "bestde.txt";
    char old4[25] = "bestu.txt";
    char old5[25] = "bestv.txt";
    char old6[25] = "bestn.txt";
    char old7[25] = "sastapop.txt";
    char new1a[25] = "allmina.txt";
    char new3a[25] = "bestdea.txt";
    char new4a[25] = "bestua.txt";
    char new5a[25] = "bestva.txt";
    char new6a[25] = "bestna.txt";
    char new7a[25] = "sastapop.txt";
    char new1b[25] = "allminb.txt";
    char new3b[25] = "bestdeb.txt";
    char new4b[25] = "bestub.txt";
    char new5b[25] = "bestvb.txt";
    char new6b[25] = "bestnb.txt";
    char new7b[25] = "sastapopb.txt";
    char new1c[25] = "allminc.txt";
    char new3c[25] = "bestdec.txt";
    char new4c[25] = "bestuc.txt";
    char new5c[25] = "bestvc.txt";
    char new6c[25] = "bestnc.txt";
    char new7c[25] = "sastapopc.txt";
    char new1d[25] = "allmind.txt";
    char new3d[25] = "bestded.txt";
    char new4d[25] = "bestud.txt";
    char new5d[25] = "bestvd.txt";
    char new6d[25] = "bestnd.txt";
    char new7d[25] = "sastapopd.txt";
    char new1e[25] = "allmine.txt";
    char new3e[25] = "bestdee.txt";
    char new4e[25] = "bestue.txt";
    char new5e[25] = "bestve.txt";
    char new6e[25] = "bestne.txt";
    char new7e[25] = "sastapope.txt";
    char new1f[25] = "allminf.txt";
    char new3f[25] = "bestdef.txt";
    char new4f[25] = "bestuf.txt";
    char new5f[25] = "bestvf.txt";
    char new6f[25] = "bestnf.txt";
    char new7f[25] = "sastapopf.txt";
    char new1g[25] = "allming.txt";
    char new3g[25] = "bestdeg.txt";
    char new4g[25] = "bestug.txt";
    char new5g[25] = "bestvg.txt";
    char new6g[25] = "bestng.txt";
    char new7g[25] = "sastapopg.txt";
    char new1h[25] = "allminh.txt";
    char new3h[25] = "bestdeh.txt";
    char new4h[25] = "bestuh.txt";
    char new5h[25] = "bestvh.txt";
    char new6h[25] = "bestth.txt";
    char new7h[25] = "sastapoph.txt";

    if (no_loops == 1){
        copy_results = rename( old1, new1a );
        if( copy_results != 0 )
            printf("Could not rename '%s'
", old1);
    }
    else
        printf("File '%s' renamed to '%s'
", old1, new1a);
}

copy_results = rename( old3, new3a );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old3 );
} else{
    printf( "File '%s' renamed to '%s'", old3, new3a );
}
copy_results = rename( old4, new4a );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old4 );
} else{
    printf( "File '%s' renamed to '%s'", old4, new4a );
}
copy_results = rename( old5, new5a );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old5 );
} else{
    printf( "File '%s' renamed to '%s'", old5, new5a );
}
copy_results = rename( old6, new6a );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old6 );
} else{
    printf( "File '%s' renamed to '%s'", old6, new6a );
}
copy_results = rename( old7, new7a );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old7 );
} else{
    printf( "File '%s' renamed to '%s'", old7, new7a );
}
}
else if (no_loops == 2){
copy_results = rename( old1, new1b );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old1 );
} else{
    printf( "File '%s' renamed to '%s'", old1, new1b );
}
else if (no_loops == 3){
copy_results = rename( old1, new1c );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old1 );
} else{
    printf( "File '%s' renamed to '%s'", old1, new1c );
}
}
copy_results = rename( old3, new3c );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old3 );
} else{
    printf( "File '%s' renamed to '%s'", old3, new3c );
}
copy_results = rename( old4, new4c );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old4 );
} else{
    printf( "File '%s' renamed to '%s'", old4, new4c );
}
copy_results = rename( old5, new5c );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old5 );
} else{
    printf( "File '%s' renamed to '%s'", old5, new5c );
}
copy_results = rename( old6, new6c );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old6 );
} else{
    printf( "File '%s' renamed to '%s'", old6, new6c );
}
copy_results = rename( old7, new7c );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old7 );
} else{
    printf( "File '%s' renamed to '%s'", old7, new7c );
}
}
else if (no_loops == 4){
copy_results = rename( old1, new1d );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old1 );
} else{
    printf( "File '%s' renamed to '%s'", old1, new1d );
}
copy_results = rename( old3, new3d );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old3 );
} else{
    printf( "File '%s' renamed to '%s'", old3, new3d );
}
copy_results = rename( old4, new4d );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old4 );
} else{
    printf( "File '%s' renamed to '%s'", old4, new4d );
}
copy_results = rename( old5, new5d );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old5 );
} else{
    printf( "File '%s' renamed to '%s'", old5, new5d );
}
copy_results = rename( old6, new6d );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old6 );
} else{
    printf( "File '%s' renamed to '%s'", old6, new6d );
}
copy_results = rename( old7, new7d );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old7 );
} else{
    printf( "File '%s' renamed to '%s'", old7, new7d );
}
}
else if (no_loops == 5){
copy_results = rename( old1, new1e );
if( copy_results != 0 ){
    printf( "Could not rename '%s'", old1 );
} else{
    printf( "File '%s' renamed to '%s'", old1, new1e );
}
else{
    printf( "File 's' renamed to 's\n", old1, new1e );
}
copy_results = rename( old3, new3e );
if( copy_results != 0 ){  
    printf( "Could not rename 's\n", old3 );
} else{
    printf( "File 's' renamed to 's\n", old3, new3e );
}
copy_results = rename( old4, new4e );
if( copy_results != 0 ){  
    printf( "Could not rename 's\n", old4 );
} else{
    printf( "File 's' renamed to 's\n", old4, new4e );
}
copy_results = rename( old5, new5e );
if( copy_results != 0 ){  
    printf( "Could not rename 's\n", old5 );
} else{
    printf( "File 's' renamed to 's\n", old5, new5e );
}
}
else if (no_loops == 6){
    copy_results = rename( old1, new1f );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old1 );
    } else{
        printf( "File 's' renamed to 's\n", old1, new1f );
    }
    copy_results = rename( old3, new3f );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old3 );
    } else{
        printf( "File 's' renamed to 's\n", old3, new3f );
    }
    copy_results = rename( old4, new4f );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old4 );
    } else{
        printf( "File 's' renamed to 's\n", old4, new4f );
    }
    copy_results = rename( old5, new5f );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old5 );
    } else{
        printf( "File 's' renamed to 's\n", old5, new5f );
    }
    copy_results = rename( old6, new6f );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old6 );
    } else{
        printf( "File 's' renamed to 's\n", old6, new6f );
    }
    copy_results = rename( old7, new7f );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old7 );
    } else{
        printf( "File 's' renamed to 's\n", old7, new7f );
    }
} else if (no_loops == 7){
    copy_results = rename( old1, new1g );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old1 );
    } else{
        printf( "File 's' renamed to 's\n", old1, new1g );
    }
    copy_results = rename( old3, new3g );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old3 );
    } else{
        printf( "File 's' renamed to 's\n", old3, new3g );
    }
    copy_results = rename( old4, new4g );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old4 );
    } else{
        printf( "File 's' renamed to 's\n", old4, new4g );
    }
    copy_results = rename( old5, new5g );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old5 );
    } else{
        printf( "File 's' renamed to 's\n", old5, new5g );
    }
    copy_results = rename( old6, new6g );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old6 );
    } else{
        printf( "File 's' renamed to 's\n", old6, new6g );
    }
    copy_results = rename( old7, new7g );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old7 );
    } else{
        printf( "File 's' renamed to 's\n", old7, new7g );
    }
} else if (no_loops == 8){
    copy_results = rename( old1, new1h );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old1 );
    } else{
        printf( "File 's' renamed to 's\n", old1, new1h );
    }
    copy_results = rename( old3, new3h );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old3 );
    } else{
        printf( "File 's' renamed to 's\n", old3, new3h );
    }
    copy_results = rename( old4, new4h );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old4 );
    } else{
        printf( "File 's' renamed to 's\n", old4, new4h );
    }
    copy_results = rename( old5, new5h );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old5 );
    } else{
        printf( "File 's' renamed to 's\n", old5, new5h );
    }
    copy_results = rename( old6, new6h );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old6 );
    } else{
        printf( "File 's' renamed to 's\n", old6, new6h );
    }
    copy_results = rename( old7, new7h );
    if( copy_results != 0 ){  
        printf( "Could not rename 's\n", old7 );
    } else{
        printf( "File 's' renamed to 's\n", old7, new7h );
    }
}
printf( "Could not rename 'ks'\n", old1 );
} }
else{
    printf( "File 'ks' renamed to 'ks'\n", old1, new1g );
    copy_results = rename( old3, new3g );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old3 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old3, new3g );
    copy_results = rename( old4, new4g );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old4 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old4, new4g );
    copy_results = rename( old5, new5g );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old5 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old5, new5g );
    copy_results = rename( old6, new6g );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old6 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old6, new6g );
    copy_results = rename( old7, new7g );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old7 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old7, new7g );
} }
else if (no_loops == 8){
    copy_results = rename( old1, new1h );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old1 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old1, new1h );
    copy_results = rename( old3, new3h );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old3 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old3, new3h );
    copy_results = rename( old4, new4h );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old4 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old4, new4h );
    copy_results = rename( old5, new5h );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old5 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old5, new5h );
    copy_results = rename( old6, new6h );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old6 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old6, new6h );
    copy_results = rename( old7, new7h );
    if( copy_results != 0 ){
        printf( "Could not rename 'ks'\n", old7 );
    } }
else{
    printf( "File 'ks' renamed to 'ks'\n", old7, new7h );
} }
else{
/* This random number generator comes from a version published by George Marsaglia and Arif Zaman, Florida State University and modified by the Dept. of Computer Science at Fachhochschule Wiesbaden, Germany. It is used as the Rand() function available in C/C++ is only pseudo random, and this version has been found to be the best random number generator known. It has a period of $2^{144}$, so should be adequate for the GA population generation and probability of mutation and crossover. */

void RandomInitialise(int ij, int kl) {
    double s, t;
    int ii, i, j, k, l, jj, m;
    /* Handle the seed range errors                        */
    /* First random number seed must be between 0 and 31328*  */
    if (ij < 0 || ij > 31328 || kl < 0 || kl > 30081)
    {
        ij = 1802;
        kl = 9373;
    }
    i = (ij / 177) % 177 + 2;
    j = (ij % 177) + 2;
    k = (kl / 169) % 178 + 1;
    l = (kl % 169);
    for (ii=0; ii<97; ii++)
    {
        s = 0.0;
        t = 0.5;
        for (jj=0; jj<24; jj++)
        {
            m = (((i * j) % 179) * k) % 179;
            i = j;
            j = k;
            k = m;
            l = (53 * l + 1) % 169;
            if (((l * m % 64)) >= 32)
                s += t;
            t *= 0.5;
        }
        u[ii] = s;
    }
    c = 362436.0 / 16777216.0;
    cd = 7654321.0 / 16777216.0;
    cm = 16777213.0 / 16777216.0;
    i97 = 97;
    j97 = 33;
    test = TRUE;
}

/* This is the random number generator proposed by George Marsaglia*/

double RandomUniform(void) {
    double uni;
    /* Make sure the initialisation routine has been called */
    if (!test)
        RandomInitialise(1802, time(NULL));
    RandomInitialise(1802, 9373);
    uni = u[i97-1] - u[j97-1];
    if (uni <= 0.0)
        uni++;
    u[i97-1] = uni;
}
```c
i97--;  
if (i97 == 0)  
i97 = 97;  

j97--;  
if (j97 == 0)  
j97 = 97;

if (i97 == 0)  
i97 = 97;
c -= cd;
if (c < 0.0)  
c += cm;
uni -= c;
if (uni < 0.0)  
uni++;  

return(uni);  
}

/*Algorithm from: Transactions on Mathematical Software,*  
*distributed pseudo-random number when called            */
double RandomGaussian(double mean,double stddev){  
double q,u,v,x,y;  

/*Generate P = (u,v) uniform in rect. enclosing acceptance region     *  
*Make sure that any random numbers <= 0 are rejected, since          *  
*gaussian() requires uniforms > 0, but RandomUniform() delivers >= 0.*/  
do {  
u = RandomUniform();  
v = RandomUniform();  
if (u <= 0.0 || v <= 0.0) {  
u = 1.0;  
v = 1.0;  
}  
v = 1.7156 * (v - 0.5);  
/*Evaluate the quadratic form*/  
x = u - 0.449871;  
y = fabs(v) + 0.386595;  
q = x * x + y * (0.19600 * y - 0.25472 * x);  
/*Accept P if inside inner ellipse*/  
if (q < 0.27597)  
break;  
/*Reject P if outside outer ellipse, or outside acceptance region*/} while ((q > 0.27846) || (v * v > -4.0 * log(u) * u * u));  
/*Return ratio of P's coordinates as the normal deviate*/  
return (mean + stddev * v / u);  
}

/*Return random integer within a range, lower -> upper INCLUSIVE*/
int RandomInt(int lower,int upper){  
   // printf("%d", (int)(RandomUniform() * (upper - lower + 1)) + lower);  
   return((int)(RandomUniform() * (upper - lower + 1)) + lower);  
}

/*Return random float within a range, lower -> upper*/
double RandomDouble(double lower,double upper){  
   // printf("%f", (upper - lower) * RandomUniform() + lower);  
   return((upper - lower) * RandomUniform() + lower);  
}

/*End of Program*/
```
Appendix D – Laboratory Lagoon Experiment Results

Five experiments were performed in a unidirectional flume containing a 1.8m sand filled section in its centre, with narrow fixed bed entrance and exit channels at opposite ends of the sand filled section. A diagram of the experimental set-up is shown in Figure D.1. The formation of equilibrium morphologies due to flow over an initially flat sand bed were analysed. The complete morphologies in each experiment were measured using a laser scanner once the system had obtained a stable, quasi-equilibrium state. An initial scan was also made to allow before and after comparisons to be made. Four of the experiments were run for six or seven hours, with one experiment run for an extended period of time. The extended experiment is discussed in Chapter Four, Section 4.5.1. The data obtained and morphologies observed for the shorter experiments are contained within this Appendix.

Two of the short experiments were run with flows just greater than critical velocity and shallow water depths, while the other two had a greater velocity and deeper water. (See Table D.1 for the exact flow and depth conditions of each experiment.)

The following sections detail the experimental results obtained.

---

**Figure D.1** Flume set-up for lagoon laboratory experiments.

**Table D.1** Flow conditions used for each of the five experiments.

<table>
<thead>
<tr>
<th>Exp. No.</th>
<th>Exit Flow Depth (mm)</th>
<th>Flow Rate (Ls⁻¹)</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>190</td>
<td>24.5</td>
</tr>
<tr>
<td>two</td>
<td>41</td>
<td>3.5</td>
</tr>
<tr>
<td>three</td>
<td>34</td>
<td>4.32</td>
</tr>
<tr>
<td>four</td>
<td>98</td>
<td>9.84</td>
</tr>
<tr>
<td>five</td>
<td>52</td>
<td>4.79</td>
</tr>
</tbody>
</table>
**Experiment One Results**

The centre section of the erodible bed area at equilibrium in Experiment One can be seen in Figure D.2. This is the resultant morphology after six hours of flow. To the right of the measured area was a large section of erosion, down to the base of the flume. This was beyond the capability of the scanner to measure as the difference in depth was outside its boundaries. The bottom left section of the measured area was also too deep to measure and extended to the base of the flume.

![Figure D.2 Contour plot of the equilibrium bed morphology of Experiment One. The flow direction was from left to right.](image)

**Experiment Two Results**

The flow used in Experiment Two was a lot smaller than in Experiment One. Due to the use of a lesser flow the erosion pattern was also milder. Scour did not extend to the base of the flume and a ripple pattern emerged down the side of the flume where the majority of the flow pattern was. Initially, the sand in the flume was levelled with a standard deviation of ±3.7mm. The morphology associated with the weaker water flow after six hours is shown in Figure D.3.
Figure D.3  Contour plot of the equilibrium bed morphology of Experiment Two. The flow direction was from left to right.

Figure D.4 and Figure D.5 show pictures of the box morphology after six hours.

Figure D.4  Morphology of Experiment Two after six hours, flow was from the top right corner to the bottom left corner. Notice the ripple patterns marking the flow path.
Figure D.5 Morphology of Experiment Two after six hours, flow was from the bottom right corner to the top left corner. Notice the ripple patterns marking the flow path.

Experiment Three Results

The flow used in Experiment Three was similar in intensity to that used in Experiment Two. The initial flat bed had a standard deviation of ±5.3mm. Again, a ripple pattern formed on one side of the flume, with a channel between the flume wall and the ripple deposition pattern.

The final morphology after seven hours can be seen in Figure D.6. A smaller, deeper channel formed along the side of the flume, in line with the flow entrance, which then curved around near the end wall to meet the flow exit. To one side of this channel, away from the wall side, there was a bank of ripples, which was moving slowly towards the flow exit. This ripple bank moved some of the sediment eroded from the bed in the vicinity of the flow entrance channel. The bed in the shadow of the shoulder enclosing the movable bed area remained unchanged, with the flow in this area associated with a velocity less than the critical velocity required to initiate sediment transport.
Experiment Four Results

The flow used in Experiment Four was more than double the flow in Experiments Two and Three, but less than the flow in Experiment One. The area of erosion near the flow entrance and exit extended to the base of the flume, while a large deposition mound grew to one side of the eroded channel, almost piercing the surface of the water at its highest point. The initial flat bed had a standard deviation of ±4.0mm.

After six hours, the flow was stopped and the final morphology was measured. This can be seen in Figure D.7. In this plot, the average initial bed level was taken as the baseline. As can be seen on the plot of the end bed morphology, a deep area of erosion occurred near to the flow entrance and along the adjacent wall. An erosion channel was formed which the bulk of the flow passed through. Once the flow hit the back wall, it bounced back off, causing a large amount of erosion, and was redirected towards the exit, where it continued to erode as it moved out of the boxed in area. After the initial spurt of energy as the water entered the box, some energy was lost, causing some sand to be deposited downstream of the initial erosion hole near the entrance of the box. This mound was formed with steep walls nearing the critical angle of repose, in areas where the velocity was approaching critical. As more sand was piled onto the mound, avalanching occurred to reduce the angle of the sidewalls.
Appendix D Morphology in Coastal and River Environments

Figure D.7  Contour plot of the equilibrium bed morphology of Experiment Four. The flow direction was from left to right.

Conclusions

The experiments showed that consistently, erosion channels formed in-line with the flow entrance channel along one side of the open sandy lagoon area. Erosion also occurred in the vicinity of the flow exit channel and the wall adjacent to it. Deposition occurred in the form of a rippled mound to one side of the erosion channel. As the flow increased, the erosion and deposition in the lagoon area also increased. Sloff et al. (2004) found similar results to the laboratory study discussed in this Appendix. They undertook physical modelling of a wide reservoir in a unidirectional flume. They utilised a different set-up, but found that an erosion channel tended to form on the opposite side of the flume to the flow exit. They also found that the channel consistently formed on this side.
Three experiments were performed in a sand-filled two-dimensional wave basin containing a detached breakwater parallel to a beach. The formation of equilibrium morphologies before and after the placement of the breakwater were analysed in detail. The complete morphologies in each experiment were measured using a laser scanner once the system had obtained a stable, quasi-equilibrium state. During the formation of equilibrium morphologies, above water scans were also measured intermittently. Four wave probes were used throughout the experiments to measure wave heights and periods. These probes were calibrated a number of times during the experiment by raising and lowering the water level in the basin by known amounts. The data obtained and morphologies observed are contained within this Appendix. The experimental results were obtained to aid in the formulation of an optimisation-based numerical model of equilibrium morphologies associated with breakwaters, as is discussed in Chapter Five.

**Experimental Method**

**Set-up Details**

A number of laboratory experiments were undertaken in a wave basin approximately 12m long, 7m wide and 600mm deep. In part of the basin a 4.4m wide sandy beach was established. The beach was composed of fine sand with a $d_{50}$ of 240µm. Bricks and sandbags were used to trap the sand in this designated area and prevent it from travelling to other parts of the basin. Initially the beach was roughly shaped to include a relatively flat section in the deepwater parts of the basin, and a gradually sloping beach in the breaker and swash areas of the basin. Waves were then run to allow the beach to form an equilibrium slope in agreement with the designated wave conditions. The wave paddle used to generate the waves was at an angle of approximately four degrees. When an equilibrium beach had formed, a detached breakwater (585mm long) was placed offshore of the breaker zone, parallel to the shoreline and similar waves were run creating a new equilibrium morphology in relation to the breakwater. A diagram of the experimental set-up can be seen in Figure E.1.
Appendix E  Morphology in Coastal and River Environments

Figure E.1  General breakwater laboratory set-up. (The wave direction was at a slight angle of approximately four degrees.)

Wave Height and Period Measurements

Initially the wave basin was slowly filled with water. Periodically filling was stopped and depth measurements recorded. At the same time, measurements of the voltage moving through the wave probes was recorded over a one minute interval, to obtain an average value for each probe. These measurements were then used to calibrate the wave probes. A calibration graph was plotted relating voltage to water depth. An example of a calibration chart, recorded during Experiment Three can be seen in Figure E.2.

\[ y = 11.713x + 77.739 \quad R^2 = 0.9998 \]
\[ y = 9.9432x + 75.506 \quad R^2 = 1 \]
\[ y = 10.503x + 77.357 \quad R^2 = 0.9999 \]
\[ y = 11.035x + 77.632 \quad R^2 = 0.9999 \]

Figure E.2  Calibration chart for initial beach formation in Experiment Three.

When the basin was filled to the desired water depth, and waves were formed in the basin, voltages were recorded. These were then converted to changes in
water depth over each time increment. The zero up crossing method was then used to calculate wave heights and periods for the time series recorded by each of the probes in use (Goda, 2000). An example of data used to calculate average wave heights and periods is plotted in Figure E.3. Similar waves were used throughout the experiments, except during the storm event in experiment three, where waves with twice the height were used.

Wave heights and periods were measured periodically throughout the experiments, at the beginning and end of each wave attack event on the shoreline. A wave attack event can be defined in this instance as a period of time where waves continuously attacked the shoreface. The wave paddle was stopped periodically in order to use the scanner to record beach morphologies above the water face as an instantaneous picture without sand and water movement. This momentary halt of wave action defined the end of one wave attack event and the start of the following one.

*Observation of Movement Towards an Equilibrium Morphology*

Once an initial equilibrium beach had formed with respect to the waves used in the basin, a detached breakwater was placed parallel to the shoreline. Waves were then rerun in the basin and scans of the resultant dynamic morphology were made using the scanner at designated time intervals. Once the morphology appeared to be stable visually, with the distance to the SWL from the basin edge to the point of the salient no longer changing, an above SWL scan was performed. This scan was compared to a scan after the same waves had been run in the
basin for another 30 to 60 minutes. The two scans were compared and if there was minimum difference between them, the system was said to have reached an equilibrium morphology. The initial and final distances to the SWL from the basin wall for each of the three experiments are given in Table E.1. The initial distance from the breakwater to the SWL for each of the experiments is given in Table E.2.

Table E.1 Initial and final distances to the SWL for each of the experiments.

<table>
<thead>
<tr>
<th></th>
<th>Distance to SWL without breakwater</th>
<th>Distance to SWL with breakwater (end)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prac 1</td>
<td>510</td>
<td>570</td>
</tr>
<tr>
<td>Prac 2</td>
<td>465</td>
<td>650</td>
</tr>
<tr>
<td>Prac 3</td>
<td>630</td>
<td>800</td>
</tr>
</tbody>
</table>

Table E.2 Initial distances from the breakwater to the SWL for each of the experiments.

<table>
<thead>
<tr>
<th></th>
<th>Distance from Breakwater to SWL (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prac 1</td>
<td>750</td>
</tr>
<tr>
<td>Prac 2</td>
<td>590</td>
</tr>
<tr>
<td>Prac 3</td>
<td>425</td>
</tr>
</tbody>
</table>

Typically an equilibrium morphology was reached after approximately 10 to 20 hours of wave action with the breakwater in position.

Morphological Measurements

The morphology was measured using a laser scanner (designed by Joe Davis) with a specified grid spacing of 2mm by 10mm. The scanner transmitted the data directly to a controlling computer. Output was in the form of voltage from the laser, which was then converted to distance using a calibration curve. The scans were performed with the laser unsubmerged in water, so a dry sand calibration value of 19.284 was used as the slope in the conversion equation.

Firstly, measurements were taken at the end of each wave attack event considering the morphology of the sand above the SWL only. Measurements were taken by first positioning the scanner at a distance from the basin edge where the salient first emerged from the water. The scanner was then programmed to take measurements at 2mm intervals in a longitudinal run for the 1m of the frame length. Once a longitudinal transect was completed, the scanner repositioned itself, 10mm in the transverse direction and another
A longitudinal transect was completed. Areas submerged by water recorded a large voltage value that was ignored in the calculation of the actual morphology.

When scans from two consecutive wave attack events were similar in morphology, water was drained from the basin and a full dry scan was obtained over the area between the breakwater and the limit of the scanner frame, approximately one metre by one metre in size. A full dry scan was also performed over the equilibrium beach before the placement of the breakwater. The movement of sediment over the entire morphological event was then analysed using a comparison of the before and after breakwater deployment scans.

Experiment Three – Observations of Re-Equilibrium After a Storm Event

In Experiment Three the experiment was carried out as per normal for the first three hours. A storm was then simulated with larger waves attacking the beach for approximately half an hour. The system was then allowed to re-equilibrate to the original wave type, which was reapplied for 18.5 hours. Morphological scans were undertaken periodically as per the first two experiments.

Experimental Results

Experiment One Results

The initial equilibrium beach from Experiment One can be seen in Figure E.4. Notice that the SWL (120mm) is almost parallel to the edge of the basin (at the top of the Figure).

After ten minutes of wave action with the detached breakwater in place offshore, the shoreline had been disrupted. This can be seen in Figure E.5 (a).

After 30 minutes, the salient lobe protruding towards the breakwater became more defined. This can be seen in Figure E.5 (b).

The salient formation was better defined after one hour of wave action, as can be seen in Figure E.5 (c).
After six hours of wave action, not only was the salient deposition more defined, but erosion on either side of the salient appeared more pronounced, as is shown in Figure E.6 (b), where darker shading represents morphology too far below the SWL for the scanner to measure. This trend of more pronounced salient shaping
continued for a number of hours as can be seen in Figure E.6 (c) and (d), the morphologies after ten and twelve hours respectively.

Figure E.6 Contour plot of the bed morphology of Experiment One after (a) five hours, (b) six hours, (c) ten hours and (d) twelve hours of wave action on a breakwater protected beach. The wave direction was from bottom to top.

The morphology after 16 hours differed little from the morphology observed after 12 hours of wave action on the breakwater. The system was then left for one more hour and the morphologies after 16 and 17 hours of wave action compared. The difference in elevation between the two morphologies can be seen in Figure E.8 (a). As this Figure portrays, the difference in morphology was minimal. The
line through the centre of the plot was caused by differences in measurements through the water surface near to the SWL.

A plot of the difference in elevation between the initial and final morphologies pinpoints the areas where erosion and deposition have occurred. This is shown in Figure E.8 (b), where deposition of the salient, opposite the breakwater is clearly marked, with erosion concentrated to either side. Erosion was more pronounced on the side that was more directly attacked by waves due to the slight angle of the wave direction.

![Figure E.8 Difference in bed morphology of Experiment One between (a) 16 hour and 17 hour and (b) initial and 17 hour morphologies. The wave direction was from bottom to top.](image)

The complete morphology at equilibrium is shown in Figure E.9. Ripple markings show that the flow bent around the breakwater. The wave attack direction was also pronounced, with a greater amount of erosion on the right side of the Figure. The salient is pointed directly at the breakwater. It also contained two smaller lobes on either side of the main salient, in line with the ends of the breakwater. These were observed when a breakwater system was modelled using an optimisation-based method in Chapter Five.

The difference in elevation between the initial equilibrium beach and the beach at equilibrium after placement of a breakwater can be seen in Figure E.10. This plot highlights areas where deposition and erosion have occurred both above and below the waterline. As can be observed, there was little erosion in the scanned area. Sand was moved from offshore, around the ends of the breakwater and deposited in its lee. There the deposited sand formed of moving ripple backs,
where the breakwater shielded the sand from high-energy wave attack. Above the water line, in the swash zone, the breakwater also sheltered the beach from high-energy attack. Sand was deposited in the swash zone in the lee of the breakwater. It formed a salient protrusion towards the breakwater, the visible zone of deposition above the SWL. The sand used to form this salient originated from either side, as can be seen by the small erosion zones in the top left and right corners of the scanned area.

![Figure E.9](image1.png)

*Figure E.9  Contour plot of the final equilibrium bed morphology of Experiment One after 17 hours of wave action towards the breakwater. The 120mm contour line represents the SWL. The wave direction was from bottom to top.*

![Figure E.10](image2.png)

*Figure E.10  Difference in complete bed morphology of Experiment One between initial and 17 hour morphologies. The wave direction was from bottom to top.*
Experiment Two Results

In Experiment Two, a similar setup to Experiment One was utilised. The beach was flattened from the previous experiment and the waves run for 5.5 hours, until an equilibrium beach formed. The equilibrium was obtained in a shorter time than Experiment One, possibly due to a sorting of sand already occurring. There may also have been a slight slope remaining after the beach had been racked at the end of the first experiment. It can be seen from Figure E.11 (a) that the initial beach at the water line was a slightly inverse salient, level with the breakwater placement in the previous experiment, as the sediment in this area may have been slightly in excess after levelling.

After 5.5 hours the beach had reached a new equilibrium with the breakwater placement. A plot of this new equilibrium can be seen in Figure E.11 (b). The sand above the SWL formed a definite salient lope protruding towards the breakwater. The small deposit near the breakwater on the left side of the plot was due to the breakwater being in two parts, with a slim crack where the two joined. This let a little water at full strength directly through the breakwater, which eroded a small amount of sediment in the lee of the breakwater where the break occurred. On the right of the plot, a ripple bank formed, which moved in the direction of the flow, bending around the breakwater.

Figure E.11 Contour plot of the (a) initial and (b) final equilibrium bed morphology of Experiment Two. The dashed line, just above the 60mm contour line represents the SWL. The wave direction was from bottom to top. The placement of the breakwater after this equilibrium beach was formed is alluded to by the black and the white rectangle in the bottom of the plot for (a) and (b) respectively.
A plot of the difference between the beach equilibrium morphologies before and after breakwater insertion can be seen in Figure E.12. This plot clearly indicates areas where deposition and erosion have occurred. There was a large body of deposition above the equilibrium SWL, opposite the breakwater where the salient formed. There were also small pockets of erosion on either side of the breakwater in the deep-water area and a small amount of erosion just below the SWL on the right side of the plot. This was an area where wave attack was stronger due to the wave crest alignments. The system was deemed to have reached an equilibrium morphology as there was little change between the 5hr and 5.5hr scans.

![Figure E.12](image)

**Figure E.12** Difference in complete bed morphology of Experiment Two between initial and 5.5 hour morphologies. The wave direction was from bottom to top. The dashed lines represent the initial and final SWL.

**Experiment Three Results**

A third experiment was undertaken using a similar set-up to the first two experiments but with the inclusion of a storm event. Initially an equilibrium beach was formed, as can be seen in Figure E.13 (a). The contours of the beach area in the breaker zone were relatively straight and parallel to the direction of wave attack.

After 20 hours of wave attack with a breakwater present, the beach reformed itself into a different equilibrium morphology, with a salient protruding from the beach towards the breakwater. This can be seen in Figure E.13 (b). In the deeper water surrounding the breakwater, a ripple pattern emerged, with ripples
bending around the edges of the breakwater, reflecting the pattern of the currents above the sediment.

After ten minutes of wave action, with the breakwater in place, a double salient began to form, with small sand lopes extending towards the ends of the breakwater. The contours of the beach and the SWL began to bend slightly towards the breakwater but remain relatively unchanged. This can be seen in Figure E.14 (a).

After half an hour, there was the definite beginning of a salient, with all contour lines bending towards the breakwater as can be seen in Figure E.14 (b).

After an hour, the point of the salient had shifted slightly to one side of the breakwater centre, in line with the direction of the water attack on the beach, with the left salient growing at a greater rate than the right one and consuming it. This can be seen in Figure E.14 (c).

After two hours, the beach was subjected to an hour of storm waves that varied in strength. The resultant morphology is shown in Figure E.14 (d). The salient became a single rather than double salient, pointed to one side of the breakwater. It had grown in size and was smoother than after one hour.
Next, the original calm waves were used, to bring the beach back to its desired equilibrium morphology with the waves. After 5.5 hours of calm waves, 8.5 hours in total since the placement of the breakwater, the salient had moved position, so that the lobe protruded more towards the centre of the breakwater. This can be seen in Figure E.15 (a).

The waves attacked the beach for another 4.5 hours and further morphological measurements were taken, after 13 hours since breakwater placement. The developed morphology can be seen in Figure E.15 (b). The salient became further pronounced in shape and height above the SWL.

The waves were continued for a further five hours, totalling the time since breakwater placement to 18 hours. The morphology remained much the same, with the salient moving a little closer towards the breakwater (see Figure E.15 (c)).
After 23 hours there was little change in morphology, as can be seen in Figure E.15 (d). The change after 24.5 hours was again only slight, as can be seen in Figure E.15 (e). After 25 hours, the morphology remained similar, as shown in Figure E.15 (f).
Summary and Conclusions

Three experiments were undertaken to measure the quasi-equilibrium morphology that resulted due to the placement of a breakwater on a stretch of beach already at equilibrium, using regular waves. Observations were also made as the beach moved towards equilibrium.

Some general experimental observations are as follows:

★ Initially, the morphology changed quite quickly after the installation of the breakwater, with the observation of a small salient protruding towards the breakwater almost immediately.

★ Once the salient had formed, it grew more slowly, marginally protruding further towards the breakwater.

★ After the simulation of a storm, the beach was able to rearrange itself back to a morphology at equilibrium with the waves applied to it.