

Numerical Simulation of Sloping Aquifer Dynamics

Felix Andrews

October 2003

Contents

1	Introduction	2
2	Review of Horizontal Homogeneous Aquifer Model	2
2.1	Derivation	2
2.2	Explicit Finite Difference Scheme	3
2.3	Implicit Finite Difference Scheme	4
3	Sloping Aquifer Model	4
3.1	Formulation	4
3.2	Explicit Finite Difference Scheme	5
3.3	Implicit Finite Difference Scheme	6
3.3.1	Jacobian Matrix	6
4	Results	7
4.1	Horizontal Homogeneous Aquifer Model	7
4.2	Sloping Aquifer Model	7
4.2.1	Comparison	7
4.2.2	Sloping Dynamics	9
4.2.3	Scaling Effects of h/L	9
4.3	Efficiency	13
5	Conclusions	13
6	References	13
A	Code	13
A.1	Horizontal Implicit	13
A.2	Sloping Implicit	16

1 Introduction

Groundwater modelling is becoming increasingly important in environmental management. Changing land use has caused water tables to rise and led to both riverine and dryland salinity problems. In other areas, pumping fresh groundwater for irrigation is greatly exceeding the natural recharge rate. In many parts of Australia, groundwater is the primary source of water (MDBC, 2003). Additionally, the groundwater contribution to stream flow (baseflow) often exceeds that from surface runoff.

A common approach is to model the steady state of an aquifer with a time-averaged recharge rate. This may be used to assess the impact of pumping sites (or free-flowing bores). For discharge studies dynamic models are needed. These may be either lumped conceptual or distributed physics-based models. In the lumped case, discharge for each spatial lump is represented by a simple function, often a single-valued function of storage (as noted in Sloan, 2000). Such an approach is favoured at large scales because of its identifiability (well-posedness for calibration) and computational efficiency.

Distributed physics-based models are generally based on equations describing flow in the saturated zone. In general it is only possible to solve these numerically. The computational load involved is often large, while the data requirements are almost never satisfied in practice. However they are widely used with some success (eg Lin, 1972; Hornberger et. al., 1970).

The system can be reduced to a single hillslope between the catchment boundary and the discharge site (stream). For simplicity we will consider one space dimension, assuming that the flow lines are coplanar.

Of the many assumptions that underlie this approach, a major one is that the confining bed is horizontal. This study aimed to numerically investigate the dynamics of an aquifer with a sloping confining bed. A further aim was to characterise its discharge function for potential use in a lumped conceptual model.

2 Review of Horizontal Homogeneous Aquifer Model

2.1 Derivation

The generally accepted equations governing the dynamics of an unconfined aquifer are presented briefly here.

Darcy's law is an empirical relationship between the water table height and the flow rate. It is consistent with more general fluid physics but makes some assumptions about the type of flow (Smith 2002, p. 44). We further make the Dupuit assumption which is equivalent to the statement: "equipotential lines are vertical (that is a function of horizontal distance only and independent of height); the hydraulic gradient is equal to the slope of the water table and invariant with depth; and the streamlines are horizontal (flow is essentially in the horizontal plane)." (Smith 2002, p. 46). The equation incorporating all this is:

$$Q = -T(x) \frac{\partial h}{\partial x} \quad (1)$$

Q is flow or discharge;

$T(x)$ is depth-averaged transmissivity;

h is the hydraulic head (height of water table above stream).

The continuity equation is derived from conservation of mass (Smith 2002, p. 53):

$$\frac{\partial Q}{\partial x} - R = -\varepsilon \frac{\partial h}{\partial t} \quad (2)$$

R is recharge;

ε is porosity.

Substituting (1) into (2) gives the Boussinesq equation:

$$\frac{\partial}{\partial x} \left[T(x) \frac{\partial h}{\partial x} \right] + R = \varepsilon \frac{\partial h}{\partial t} \quad (3)$$

We define the boundary conditions for the case of a hillslope where the top is assumed to be the catchment boundary, so we have the Neumann condition:

$$\left. \frac{\partial h}{\partial x} \right|_{x=0} = 0$$

and the head is fixed at the stream with the Dirichlet condition:

$$h_L = 0$$

where L is the length of the hillslope (the end where the aquifer discharges). The initial state can be any function; typically either dry or in steady state.

If we further assume that the aquifer is homogeneous in its transmissivity we can write (3) as:

$$T \frac{\partial^2 h}{\partial x^2} + R = \varepsilon \frac{\partial h}{\partial t} \quad (4)$$

which is just the heat equation (ignoring recharge).

2.2 Explicit Finite Difference Scheme

To solve (4) numerically a basic finite difference approximation can be used. First we discretise the space domain $[0, L]$ into lengths Δx , and the time domain $[0, t_{end}]$ into periods Δt . Using the explicit FTCS (Forward Time, Centred Space) scheme gives

$$h_x^{t+1} = h_x^t + \frac{\Delta t}{\varepsilon} \left(\frac{T}{\Delta x^2} (h_{x+1}^t - 2h_x^t + h_{x-1}^t) + R \right) \quad (5)$$

where subscripts are space indices and superscripts are time indices. The stability of this method can be investigated by substituting a solution of the form $h_x^t = \xi_k^t e^{ikx\Delta x}$ with k a real positive wavenumber. (Note that ξ is to the power of t here.) This is known as von Neumann stability analysis (Savage, 2003). For stability we require $|\xi| \leq 1$ which corresponds to the condition

$$\Delta t \leq \frac{\varepsilon}{2T} \Delta x^2 \quad (6)$$

2.3 Implicit Finite Difference Scheme

The stability condition just noted for the explicit scheme (6) is quite severe. For example if we discretise the aquifer into 100 points it is necessary for $\Delta t \leq 0.00005$ (with unit length and unit parameters). That is at least 20,000 updates for each time unit. The alternative is to use an implicit method. We can replace the second derivative with the implicit Crank-Nicolson approximation, giving an unconditional stable scheme. It is also second-order accurate in both space and time (Heath 2002, p. 459):

$$\varepsilon \frac{h_x^{t+1} - h_x^t}{\Delta t} = T \frac{h_{x+1}^{t+1} - 2h_x^{t+1} + h_{x-1}^{t+1} + h_{x+1}^t - 2h_x^t + h_{x-1}^t}{2\Delta x^2} + R$$

This is a system of linear equations which we can formulate as a matrix problem. If we let $\phi = \frac{T\Delta t}{2\Delta x^2\varepsilon}$ then we can write

$$\begin{bmatrix} 1 + \phi & -\phi & & & & & \\ -\phi & 1 + 2\phi & -\phi & & & & \\ & -\phi & 1 + 2\phi & \ddots & & & \\ & & & \ddots & \ddots & & \\ & & & & -\phi & 1 + 2\phi & \\ & & & & -\phi & 1 + 2\phi & \end{bmatrix} \begin{bmatrix} h_1^{t+1} \\ h_2^{t+1} \\ h_3^{t+1} \\ \vdots \\ h_{L-1}^{t+1} \end{bmatrix} = \begin{bmatrix} h_1^t + \phi(h_2^t - 2h_1^t + h_0^t) + \frac{\Delta t}{\varepsilon}R \\ \vdots \\ \end{bmatrix} \quad (7)$$

This is a tridiagonal system which is easy to solve. A simple linear-time LU-factorisation algorithm is given in Heath (2002, p. 88). We can then solve for \mathbf{h}^{t+1} as usual by back-substitution. Of course we use a sparse representation of the matrix.

If we were working in 2 space dimensions the system would typically be *block tridiagonal*. We could use a sparse direct method such as a banded solver (in this case a bandwidth of 3), reordered Cholesky factorisation, or alternatively, one of the many iterative algorithms.

With this scheme we are free to use more reasonable time steps, the only consideration being accuracy.

3 Sloping Aquifer Model

3.1 Formulation

The model described above can be generalised to the case of a sloping confining bed, as described in Smith (2002, p. 76). Furthermore we will discard the Dupuit assumption for a homogeneous aquifer. The Boussinesq equation for a sloping aquifer then becomes

$$\varepsilon \frac{\partial h}{\partial t} + \frac{kgS}{v} \frac{\partial h}{\partial x} = \frac{kgC}{v} \frac{\partial}{\partial x} \left(h \frac{\partial h}{\partial x} \right) + R \quad (8)$$

- k is permeability;
- g is gravity field strength;
- v is kinematic viscosity;
- S is $\sin \theta$;

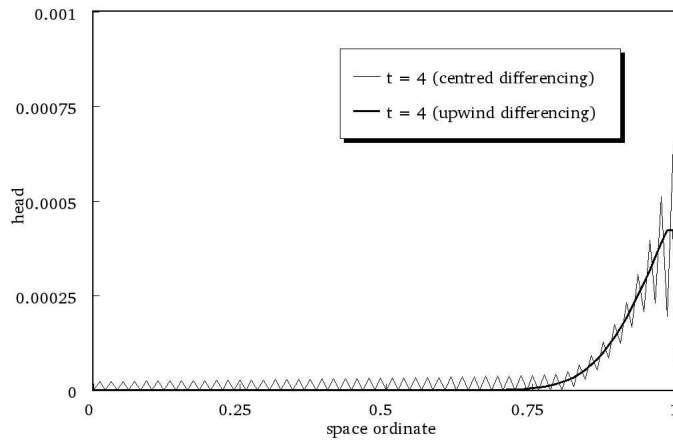


Figure 1: Instability arising from centred space approximations, and solution using upwind differencing. Here the aquifer started with a uniform head, and has discharged at $x = L$ for 4 time units.

C is $\cos \theta$;

θ is the angle of declination of the sloping bed below horizontal;

h is now measured perpendicular to the sloping bed.

So we now have a non-linear parabolic partial differential equation to solve.

The Neumann boundary condition at the top of the catchment must be modified for this model. If we used a zero derivative as before, that would be equivalent to water flowing in from the top according to the slope. We would like the boundary to be mass-neutral, which means the derivative of head perpendicular to gravity must be 0. If we only consider the adjacent point, some simple trigonometry gives us

$$h_0 = h_1 - \Delta x \tan \theta \quad (9)$$

Apart from the nonlinearity in h , (8) is a (dynamic) convection diffusion equation (Hackbusch, 1992, p. 247). It has been shown that centred finite difference approximations in this case can develop an oscillation for certain parameter ranges. The standard solution is to use (first order) upwind differencing for these terms, sampled from the direction of the oncoming front (Hackbusch, 1992, p. 250; Heath, 2002, p. 461). In fact these oscillations were observed in the initial implementation, and resolved by sampling the upslope side of each point: see Figure (1).

3.2 Explicit Finite Difference Scheme

We can follow the same procedure as in Section (2.2) to obtain an explicit difference equation for \mathbf{h}^{t+1} . However, the von Neumann stability analysis in this case is nasty, and there is no obvious way to simplify it. It is likely that the conditions would be at least as strict as in the previous case. For reliability and efficiency we again move to an implicit scheme.

Algorithm 1 Newton's method in N dimensions (Heath, 2002, p. 239). \mathbf{J}_f is the Jacobian: see section (3.3.1).

μ_0 = initial guess

for $k = 0, 1, 2, \dots$

 solve $\mathbf{J}_f(\mu_k)\mathbf{s}_k = -f(\mu_k)$ for \mathbf{s}_k

$\mu_{k+1} = \mu_k + \mathbf{s}_k$

end

3.3 Implicit Finite Difference Scheme

If we use an implicit finite difference approximation to the derivatives in (8) we arrive at a system of non-linear equations. One attractive option for solving these is a generalisation of Newton's method. As an iterative one dimensional root-finder it is simple (using a first-order Taylor series expansion) and in most cases converges quadratically. All that is required is the derivative of the function, and an initial guess that is *close enough* — a condition that depends on the function and is hard to define in general.

The analogous method for a system of non-linear equations given in Algorithm (1).

Bearing in mind the remarks about centred differencing above, and noting that $\frac{\partial}{\partial x} \left(h \frac{\partial h}{\partial x} \right) = \left(\frac{\partial h}{\partial x} \right)^2 + h \frac{\partial^2 h}{\partial x^2}$, our function can be formulated as follows. It has been arranged as a root-finding problem.

$$f(h_x^{t+1}) = 0 = \frac{\Delta t}{\epsilon} R - h_x^{t+1} + h_x^t + \frac{kg\Delta t}{v\epsilon} \left[-S \frac{h_x^{t+1} - h_{x-1}^{t+1} + h_x^t - h_{x-1}^t}{2\Delta x} \right] + \frac{kg\Delta t}{v\epsilon} C \left(\left[\frac{h_x^{t+1} - h_{x-1}^{t+1} + h_x^t - h_{x-1}^t}{2\Delta x} \right]^2 + h_x^t \left[\frac{h_x^{t+1} - 2h_x^{t+1} + h_{x-1}^{t+1} + h_x^t - 2h_x^t + h_{x-1}^t}{2\Delta x^2} \right] \right)$$

3.3.1 Jacobian Matrix

The Jacobian \mathbf{J}_f in Algorithm (1) is a matrix of partial derivatives of each element of the function vector with respect to each discrete space element: $\{\mathbf{J}_f(\mu)\}_{ij} = \frac{\partial f_i(\mu)}{\partial \mu_j}$. The Δx^2 denominator in the second derivative term matches that in the squared derivative, so we can simplify substantially. The diagonal elements then become:

$$\frac{\partial f_x(\mu)}{\partial \mu_x} = -1 + \frac{kg\Delta t}{v\epsilon} \left[-S \frac{1}{2\Delta x} + C \frac{\mu_x - \mu_{x-1} + h_x^t - h_{x-1}^t}{2\Delta x^2} \right] \quad (10)$$

Similarly the superdiagonal elements, which are derivatives with respect to the adjacent upslope points, are

$$\frac{\partial f_x(\mu)}{\partial \mu_{x-1}} = \frac{kg\Delta t}{v\epsilon} \left[-S \frac{-1}{2\Delta x} + C \frac{-\mu_x + \mu_{x-1} + h_x^t - h_{x-1}^t}{2\Delta x^2} \right] \quad (11)$$

and the subdiagonal elements, with respect to the downslope points, are

$$\frac{\partial f_x(\mu)}{\partial \mu_{x+1}} = \frac{kg\Delta t}{v\epsilon} \left[C \frac{h_x^t}{2\Delta x^2} \right] \quad (12)$$

All other elements of the matrix are 0, since the function only includes adjacent points. So the linear system in Algorithm (1) is again a tridiagonal one and the remarks above apply equally here.

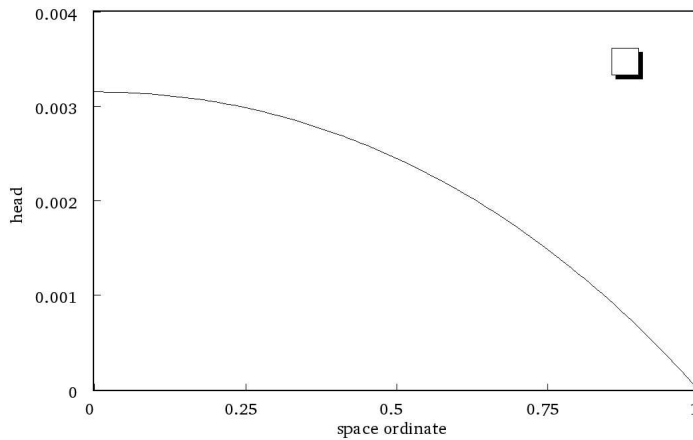


Figure 2: Steady state aquifer for horizontal homogeneous model.

The choice of initialisation is obvious in this case: since we are solving each time step as a perturbation of the previous one, the initial guess for Newton's method is just $\mu_0 = \mathbf{h}^f$. This imposes a constraint on our time step to ensure that the method converges, but we have a similar constraint anyway to ensure temporal accuracy.

4 Results

4.1 Horizontal Homogeneous Aquifer Model

The explicit method was observed to fail (falling negative) consistently with the FTCS condition. With a reasonable spatial resolution the implicit method was much faster. As we would hope, the two methods agreed while stable. With a constant recharge of 0.001 per unit space and time, the steady state aquifer was that given in Figure (2) — the classic aquifer profile.

The discharge response from an initial state of a flat head was the decay form in Figure (3). This represents a uniform rainfall (and infiltration) event into a dry aquifer. The result obtained matches the form given in Hornberger et. al. (1970, p. 604).

We can also look at the discharge response from an initial steady state, by using constant recharge until the discharge stabilises, then stopping recharge. The result given in Figure (4) roughly matches that in Sloan (2000, p. 232).

4.2 Sloping Aquifer Model

4.2.1 Comparison

Previous results from this model were not found in the literature. However we can at least compare the horizontal case to those just described of the homogeneous model (section 4.1). The steady state aquifers are compared in Figure (5). The difference is due to the simpler linear model having a constant transmissivity, which was here estimated as k times the unit recharge (in reality $T = hk$, as in the non-linear model). Similar comments apply to the differences between dry state (Figure 6) and steady

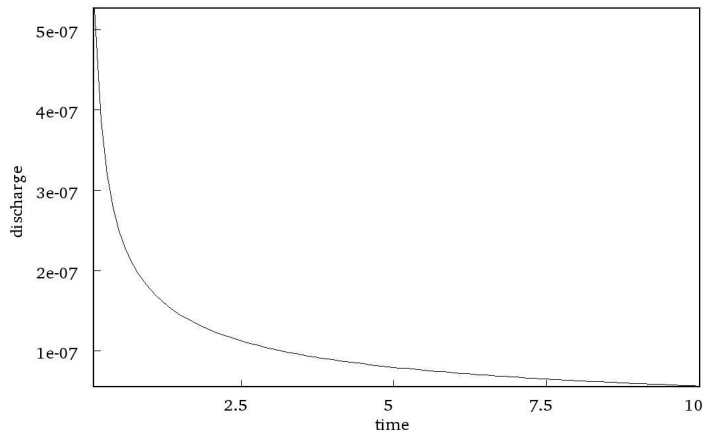


Figure 3: Discharge response from initial uniform water table (horizontal homogeneous model).

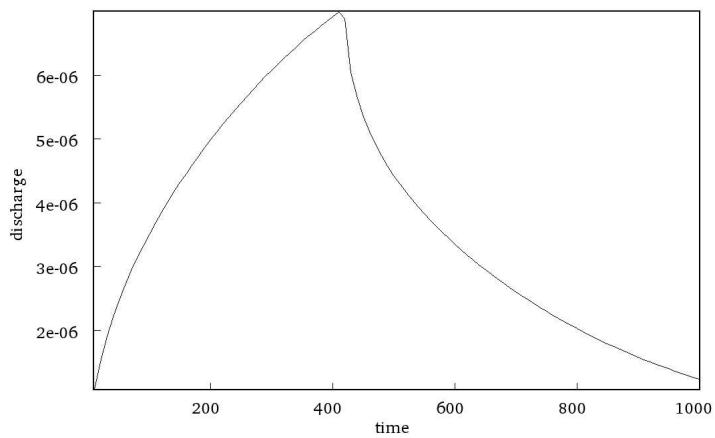


Figure 4: Discharge response from steady state (horizontal homogeneous model).

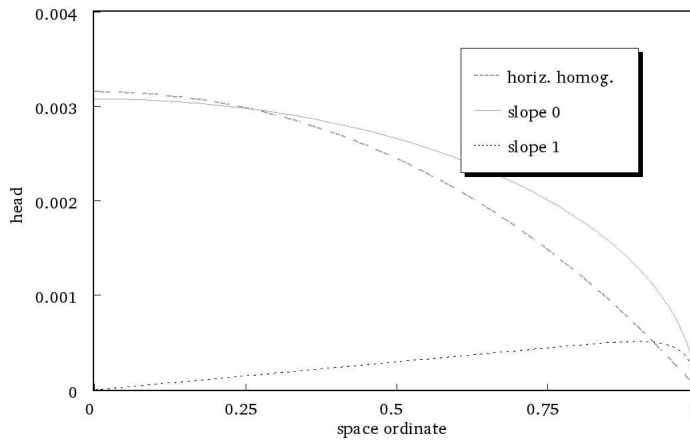


Figure 5: Steady state aquifers for the two horizontal models compared.

state discharge responses (Figure 7), although it is hard to compare these directly due to parameter scale differences.

4.2.2 Sloping Dynamics

We now turn to the effect of increasing the gradient of the confining bed. Figure (8) shows steady state aquifers for several slopes. Note that the 1-degree slope data is the same as on Figure (5), which shows the horizontal case. Apparently any slight slope reverses the shape of the aquifer such that the head is highest near the stream. On this scale the aquifers appear linear, with gradient inversely proportional to the slope of the confining bed.

The discharge responses from dry state are quite simple in form: see Figure (9). It seems that the water flows off the hillslope in a largely uniform manner, giving constant discharge proportional to the slope. The response tails off smoothly for smaller angles.

Even simpler in form are the steady state discharge responses: see Figure (10). They are apparently almost linear, similar to the dry state discharge tails.

4.2.3 Scaling Effects of h/L

The ratio of hydraulic head to the aquifer length becomes important for the sloping model. In the horizontal case it had no effect on the form of output, but here we are dealing with angles so it changes the forces involved. In the results above a ratio of at most 1:1000 was maintained, because a real hillslope might be several kilometres while the water table is usually a few metres.

If we increase the ratio of h/L to around 1:10 we get a generally more non-linear response. The steady state aquifers are much more curved, even attaining a maximum head midway for some angles: see Figure (11). The discharge also has more pronounced curvature.

Other parameters alter only the time or magnitude scales involved.

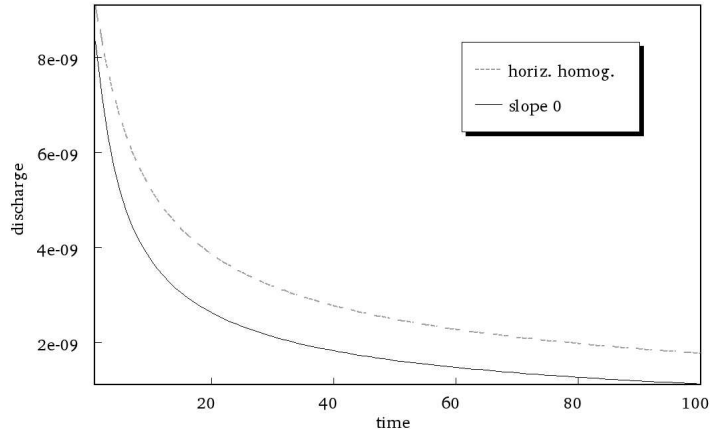


Figure 6: Discharge responses from dry state for the two horizontal models compared.

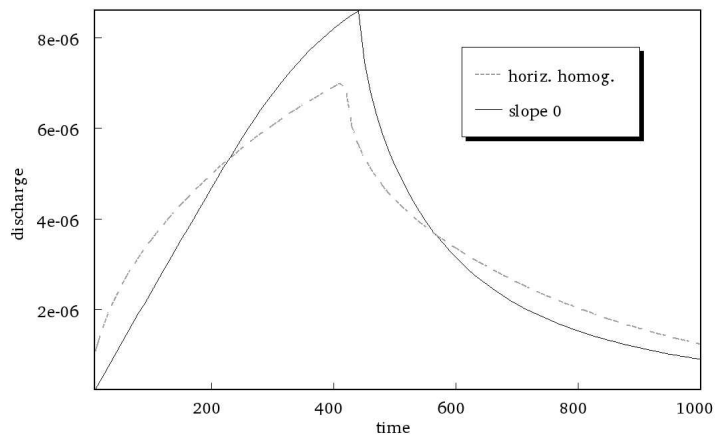


Figure 7: Discharge responses from steady state for the two horizontal models compared.

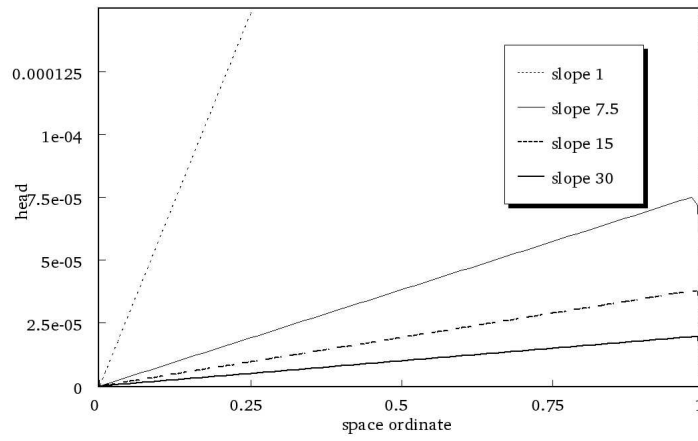


Figure 8: Steady state aquifers found for various slopes of the confining bed (in degrees).

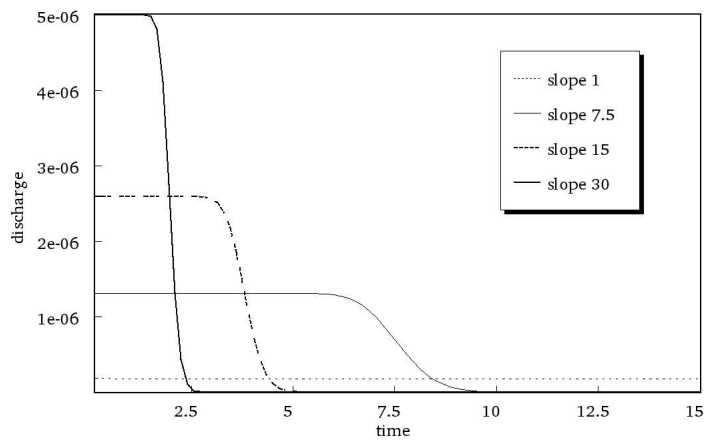


Figure 9: Dry state discharge responses for various slopes.

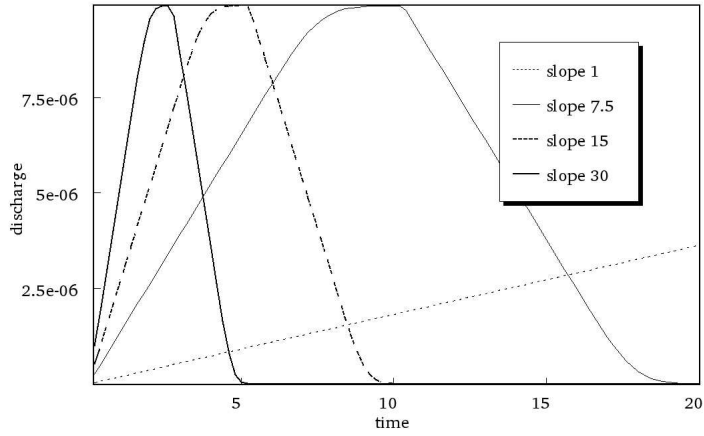


Figure 10: Steady state discharge responses for various slopes.

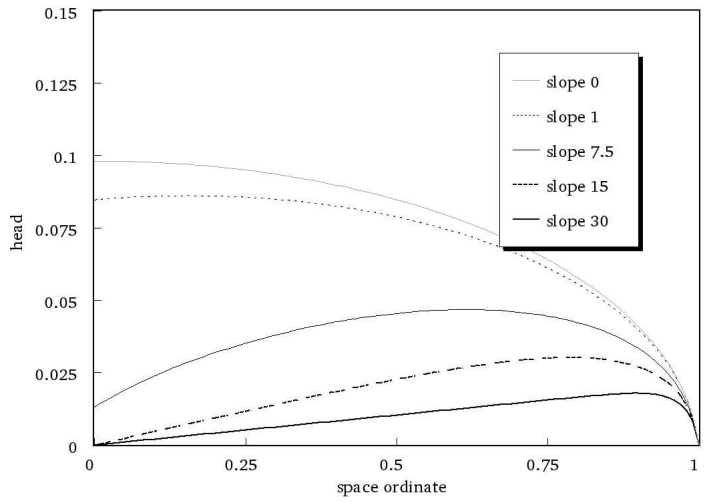


Figure 11: Steady state aquifers for various slopes, with a very large head.

4.3 Efficiency

The sloping implicit model was observed to take roughly 3 times as long to run as the horizontal implicit model, corresponding to the fact that usually 3 Newton steps were taken to reach the (strict) error tolerance.

5 Conclusions

The simulations were successful and gave some reasonable results. Further work should be done including:

- further verifying the numerical results against analytic cases;
- characterising the sloping aquifer dynamics in simple functional forms;
- extending the model to two dimensions to check that similar results occur.

6 References

Hackbusch, W. (1987). ELLIPTIC DIFFERENTIAL EQUATIONS: THEORY AND NUMERICAL TREATMENT. Springer Series in Computational Mathematics – 18. Springer-Verlag.

Heath, M. T. (2002). SCIENTIFIC COMPUTING: AN INTRODUCTORY SURVEY (2nd Edition) McGraw-Hill.

Hornberger, G. M. et. al. (1970). NUMERICAL SOLUTION OF THE BOUSSINESQ EQUATION FOR AQUIFER-STREAM INTERACTION. Water Resources Research, Vol. 6. 2, pp. 601-608.

MDBC (2003). MURRAY-DARLING BASIN COMMISSION EDUCATION CENTRE: GROUNDWATER RESOURCES <http://www.mdbc.gov.au/education/encyclopedia/Groundwater/Groundwater.htm>
<http://www.mdbc.gov.au/education/encyclopedia/Groundwater/Groundwater.htm>

Savage, Craig. (2003). LECTURE NOTES FOR PHYS3038. ANU.

Sloan, W.T. (2000). A PHYSICS-BASED FUNCTION FOR MODELING TRANSIENT GROUNDWATER DISCHARGE AT THE WATERSHED SCALE. Water Resources Research, Vol. 36. 1, pp. 225-241.

Smith, A. (2002). DEVELOPMENT OF A GROUNDWATER MODEL. Honours Thesis, ANU.

A Code

A.1 Horizontal Implicit

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```

#include <time.h>
#include <sys/time.h>
#include <math.h>
int main(int argc, char *argv[])
{
int i_t, i_x;
int N_t, N_x;
double dt, dx, t_end, L;
double transmissivity, porosity, phi;
double *h_t, *h_t_new, *h_t_temp;
double *m, *u_d, *u_d1, *b_t;
int out_sample_mod;
double discharge, discharge_prev;
double recharge;
int pulse;
double aquifer_snap_step = -1;
if (argc < 9) {

    fprintf(stderr, "Usage: %s dt dx t_end L transmissivity porosity out_samples pulse
    return(1);

}
dt = atof(argv[1]);
dx = atof(argv[2]);
t_end = atof(argv[3]);
L = atof(argv[4]);
N_t = t_end / dt + 1;
N_x = L / dx + 1;
transmissivity = atof(argv[5]);
porosity = atof(argv[6]);
out_sample_mod = (int)( N_t / atoi(argv[7]) );
if (out_sample_mod < 1) out_sample_mod = 1;
pulse = atoi(argv[8]);
if (argc > 9) { aquifer_snap_step = (int)( atof(argv[9]) / dt ); }
recharge = 1e-3 * dx;
if (!pulse) recharge *= (dt / porosity);
phi = (dt / porosity) * (transmissivity / (dx*dx));
h_t = (double*) malloc((N_x) * sizeof(double));
h_t_new = (double*) malloc((N_x) * sizeof(double));
m = (double*) malloc((N_x) * sizeof(double));
u_d = (double*) malloc((N_x) * sizeof(double));
u_d1 = (double*) malloc((N_x) * sizeof(double));
b_t = (double*) malloc((N_x) * sizeof(double));
if (!(h_t && h_t_new && m && u_d && u_d1 && b_t)) {

    fprintf(stderr, "Allocation failure; reduce dimensions \n");
    exit(-1);

}
// set initial conditions
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

```

```

        h_t[i_x] = recharge;
    }
    // set initial Neumann boundary condition
    h_t[0] = h_t[1];
    // set Dirichlet boundary condition
    h_t[N_x] = 0.0;
    h_t_new[N_x] = 0.0;
    // initialise b vector
    for (i_x = (0); i_x <= (N_x); ++i_x) {
        b_t[i_x] = h_t[i_x];
    }
    // generate triangularised matrix
    u_d[1] = (1.0 + 0.5 * phi);
    for (i_x = (0)+1+1; i_x <= N_x; ++i_x) {
        m[i_x] = (-0.5 * phi) / u_d[i_x-1];
        u_d1[i_x-1] = (-0.5 * phi);
        if (i_x == 1) { u_d1[i_x-1] = 1.0; }
        u_d[i_x] = (1.0 + phi) - m[i_x] * u_d1[i_x-1];
        // transform initial b vector
        b_t[i_x] = b_t[i_x] - m[i_x] * b_t[i_x-1];
    }
    // loop over time steps
    for (i_t = 1; i_t <= N_t; ++i_t) {
        // solve next step by back-substitution
        for (i_x = (N_x)-1; i_x >= (0)+1; --i_x) {
            h_t_new[i_x] = b_t[i_x] / u_d[i_x];
            b_t[i_x-1] = b_t[i_x-1] - u_d1[i_x-1] * h_t_new[i_x];
        }
        // calculate discharge
        discharge_prev = discharge;
        discharge = 0.0;
        for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {
            discharge += h_t[i_x] - h_t_new[i_x];
        }
        discharge *= dx / dt;
        // if we have reached steady state, switch off recharge
        if ((!pulse) && (dt * i_t > 1.0) && (fabs(discharge - discharge_prev) < 1e-3 * dt *

        fprintf(stderr, "\n steady state at time: %2.3f \n", dt * i_t);
        pulse = 1;
    }
    // add recharge for steady state case
    if (! pulse) {

        for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

```

```

        h_t_new[i_x] += recharge;
    }
}
// set Neumann boundary condition
h_t_new[0] = h_t_new[1];
// print output
if (aquifer_snap_step == -1) {

    if (i_t % out_sample_mod == 0) { printf("%12.3e %12.3e \n", dt * i_t, discharge)
    }
else if (aquifer_snap_step == i_t) {
    for (i_x = (0); i_x <= (N_x); ++i_x) {

        printf("%12.3e %12.3e \n", dx * i_x, h_t_new[i_x]);
    }
fprintf(stderr, "\n aquifer snapshot time: %2.3f \n", dt * i_t);
return 0;
}
// next h_t iterate (swap to avoid reallocating)
h_t_temp = h_t;
h_t = h_t_new;
h_t_new = h_t_temp;
// calculate new b vector
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

    b_t[i_x] = h_t[i_x] + (0.5 * phi) * (h_t[i_x+1] - 2*h_t[i_x] + h_t[i_x-1]);
}
// apply transformations from triangularisation of matrix
for (i_x = (0)+1; i_x <= N_x; ++i_x) {

    b_t[i_x] = b_t[i_x] - m[i_x] * b_t[i_x-1];
}
}
return 0;
}

```

A.2 Sloping Implicit

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>
#define MAX_NEWTON_ITERATIONS 10
#define PI 3.14159
#define TRUE 1
#define FALSE 0

```

```

int main(int argc, char *argv[])
{
int i_t, i_x;
int N_t, N_x;
double dt, dx, t_end, L;
double porosity, kg_on_v, gamma;
double angle, S, C;
double backflow_threshold;
double *h_t, *h_t_new, *h_t_temp;
double *mu, *s, *f, m;
double *J_diag, *J_sub1, *J_sup1;
double residual;
double dh_dx, d2h_dx2;
int out_sample_mod;
double discharge, discharge_prev;
double recharge;
int pulse;
double aquifer_snap_step = -1;
int i_newt;
int converged;
if (argc < 10) {
fprintf(stderr, "Usage: %s dt dx t_end L angle porosity kg/v out_samples pulse [snap_time]
return(1);
}
dt = atof(argv[1]);
dx = atof(argv[2]);
t_end = atof(argv[3]);
L = atof(argv[4]);
N_t = t_end / dt + 1;
N_x = L / dx + 1;
angle = atof(argv[5]);
porosity = atof(argv[6]);
kg_on_v = atof(argv[7]);
out_sample_mod = (int)( N_t / atoi(argv[8]) );
if (out_sample_mod < 1) out_sample_mod = 1;
pulse = atoi(argv[9]);
if (argc > 10) { aquifer_snap_step = (int)( atof(argv[10]) / dt ); }
recharge = 1e-3 * dx;
if (!pulse) recharge *= (dt / porosity);
discharge = 1e10;
gamma = (dt / porosity) * kg_on_v;
S = sin(angle * PI / 180.0);
C = cos(angle * PI / 180.0);
backflow_threshold = dx * tan(angle * PI / 180.0);
if (C == 0) backflow_threshold = dx;
h_t = (double*) malloc((N_x) * sizeof(double));
h_t_new = (double*) malloc((N_x) * sizeof(double));
mu = (double*) malloc((N_x) * sizeof(double));
s = (double*) malloc((N_x) * sizeof(double));
f = (double*) malloc((N_x) * sizeof(double));

```

```

J_diag = (double*) malloc((N_x) * sizeof(double));
J_sub1 = (double*) malloc((N_x) * sizeof(double));
J_sup1 = (double*) malloc((N_x) * sizeof(double));
if (!(h_t && h_t_new && mu && s && f && J_diag && J_sub1 && J_sup1)) {
    fprintf(stderr, "Allocation failure; reduce dimensions \n");
    exit(-1);
}
// set initial conditions
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {
    h_t[i_x] = recharge;
}
// set initial Neumann boundary condition
h_t[0] = h_t[1] - backflow_threshold;
if (h_t[0] < 0.0) h_t[0] = 0.0;
// set Dirichlet boundary condition
h_t[N_x] = 0.0;
h_t_new[N_x] = 0.0;
// loop over time steps
for (i_t = 1; i_t <= N_t; ++i_t) {
    // initial guess is just previous time step
    for (i_x = (0); i_x <= (N_x); ++i_x) {
        mu[i_x] = h_t[i_x];
    }
    converged = FALSE;
    // Newton iteration
    for (i_newt = 1; i_newt < MAX_NEWTON_ITERATIONS; ++i_newt) {

        // calculate function f(mu)
        for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

            dh_dx = (mu[i_x] - mu[i_x-1] + h_t[i_x] - h_t[i_x-1]) / (2 * dx);
            d2h_dx2 = (mu[i_x+1] - 2*mu[i_x] + mu[i_x-1] + h_t[i_x+1] - 2*h_t[i_x] + h_t[i_x-1]) / (2 * dx * dx);
            f[i_x] = -mu[i_x] + h_t[i_x] + gamma * (-S*(dh_dx) + C*( dh_dx*dh_dx + h_t[i_x]*d2h_dx2));
            // actually we want -f(mu)!
            f[i_x] = -f[i_x];
        }
        // calculate Jacobian J_f(mu)
        for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

            J_diag[i_x] = -1 + gamma * (-S*( 1/(2*dx) ) + C*( (mu[i_x] - mu[i_x-1] - h_t[i_x] - h_t[i_x-1]) / (2 * dx) ));
            if (i_x > (0)+1) {

                J_sub1[i_x] = gamma * C * h_t[i_x] * 1/(2*dx*dx);
            }
            if (i_x < (N_x)-1) {

                J_sup1[i_x] = gamma * (-S*(-1/(2*dx) ) + C*( (-mu[i_x] + mu[i_x-1] + h_t[i_x] + h_t[i_x+1]) / (2 * dx) ));
            }
        }
    }
}

```

```

    }
}
// triangularise system J*s=f (in place)
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

    m = J_subl[i_x] / J_diag[i_x-1];
    J_diag[i_x] = J_diag[i_x] - m * J_supl[i_x-1];
    f[i_x] = f[i_x] - m * f[i_x-1];
}
// solve system for step length (by back-substitution)
for (i_x = (N_x)-1; i_x >= (0)+1; --i_x) {

    s[i_x] = f[i_x] / J_diag[i_x];
    f[i_x-1] = f[i_x-1] - J_supl[i_x-1] * s[i_x];
}
// update: mu = mu + s
residual = 0.0;
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

    mu[i_x] = mu[i_x] + s[i_x];
    residual += fabs(s[i_x]);
}
// if step size is small we have converged
if (residual < dt * dx * 1e-4) {
    converged = TRUE;
    break;
}
// maintain Neumann boundary condition
mu[0] = mu[1] - backflow_threshold;
if (mu[0] < 0.0) mu[0] = 0.0;
}
if (!converged) {
    fprintf(stderr, "\n Newton method failed to converge! \n");
    return 1;
}
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

    if (mu[i_x] < 0.0) mu[i_x] = 0.0; // kludge?
}
h_t_temp = h_t_new;
h_t_new = mu;
mu = h_t_temp;
// calculate discharge
discharge_prev = discharge;
discharge = 0.0;
for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {
    discharge += h_t[i_x] - h_t_new[i_x];
}

```

```

}
discharge *= dx / dt;
// if we have reached steady state, switch off recharge
if ((!pulse) && (dt * i_t > 1.0) && (fabs(discharge - discharge_prev) < 1e-3 * dt *

    fprintf(stderr, "\n steady state at time: %2.3f \n", dt * i_t);
    pulse = 1;
}
// add recharge for steady state case
if (! pulse) {

    for (i_x = (0)+1; i_x <= (N_x)-1; ++i_x) {

        h_t_new[i_x] += recharge;
    }
}
// set Neumann boundary condition
h_t_new[0] = h_t_new[1] - backflow_threshold;
if (h_t_new[0] < 0.0) h_t_new[0] = 0.0;
// print output
if (aquifer_snap_step == -1) {

    if (i_t % out_sample_mod == 0) { printf("%12.3e %12.3e \n", dt * i_t, discharge)
}
else if (aquifer_snap_step == i_t) {
    for (i_x = (0); i_x <= (N_x); ++i_x) {

        printf("%12.3e %12.3e \n", dx * i_x, h_t_new[i_x]);
    }
    fprintf(stderr, "\n aquifer snapshot time: %2.3f \n", dt * i_t);
    return 0;
}
// next h_t iterate (swap to avoid reallocating)
h_t_temp = h_t;
h_t = h_t_new;
h_t_new = h_t_temp;
}
return 0;
}

```