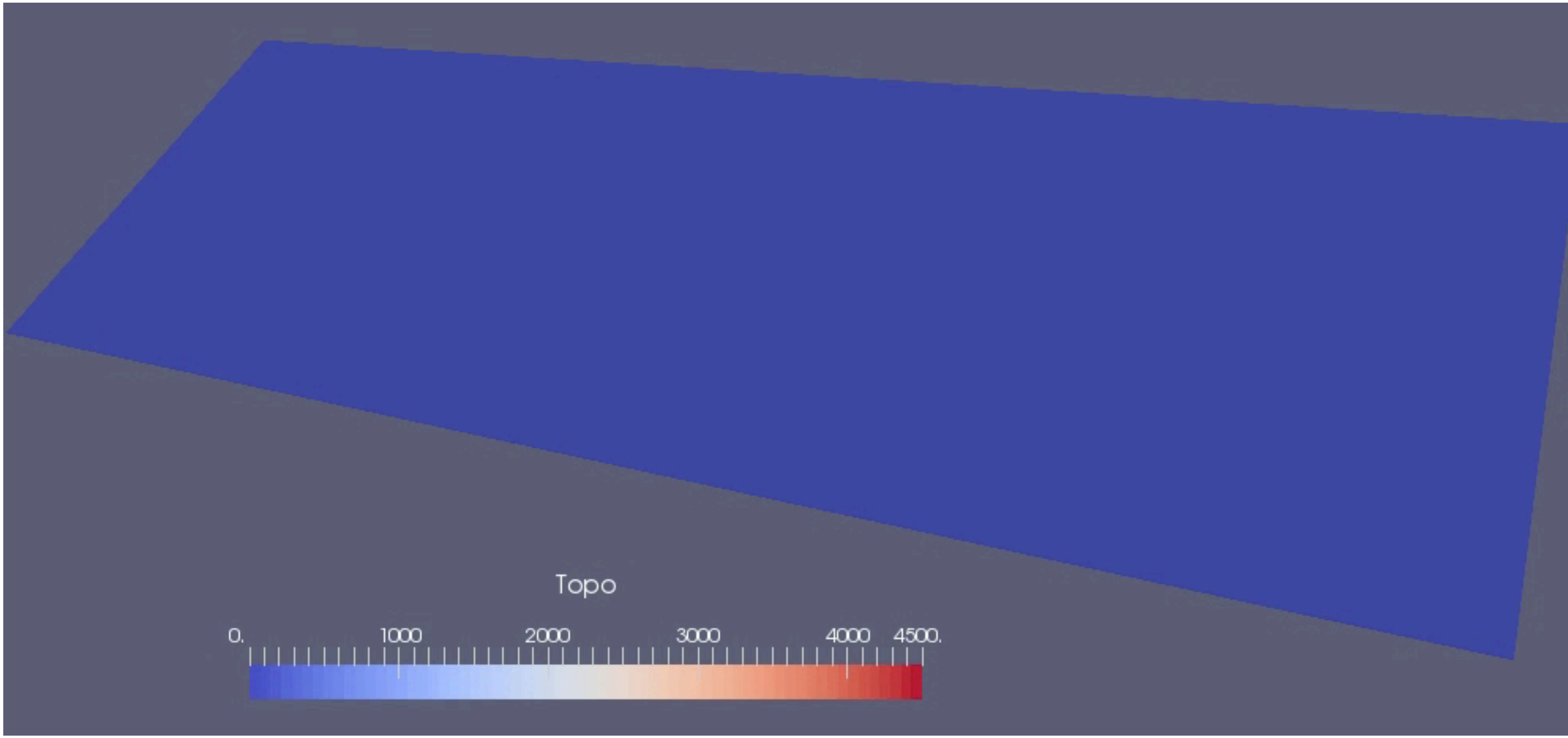
Basic Numerical Modelling Techniques in the Earth Sciences with Application to Landscape Evolution

Jean BRAUN
Université Grenoble Alpes
University of California, Berkeley (since January 2016)
Potsdam University and GFZ (rom September 2016)

A short course on numerical modelling



… using landscape evolution as an example

Content

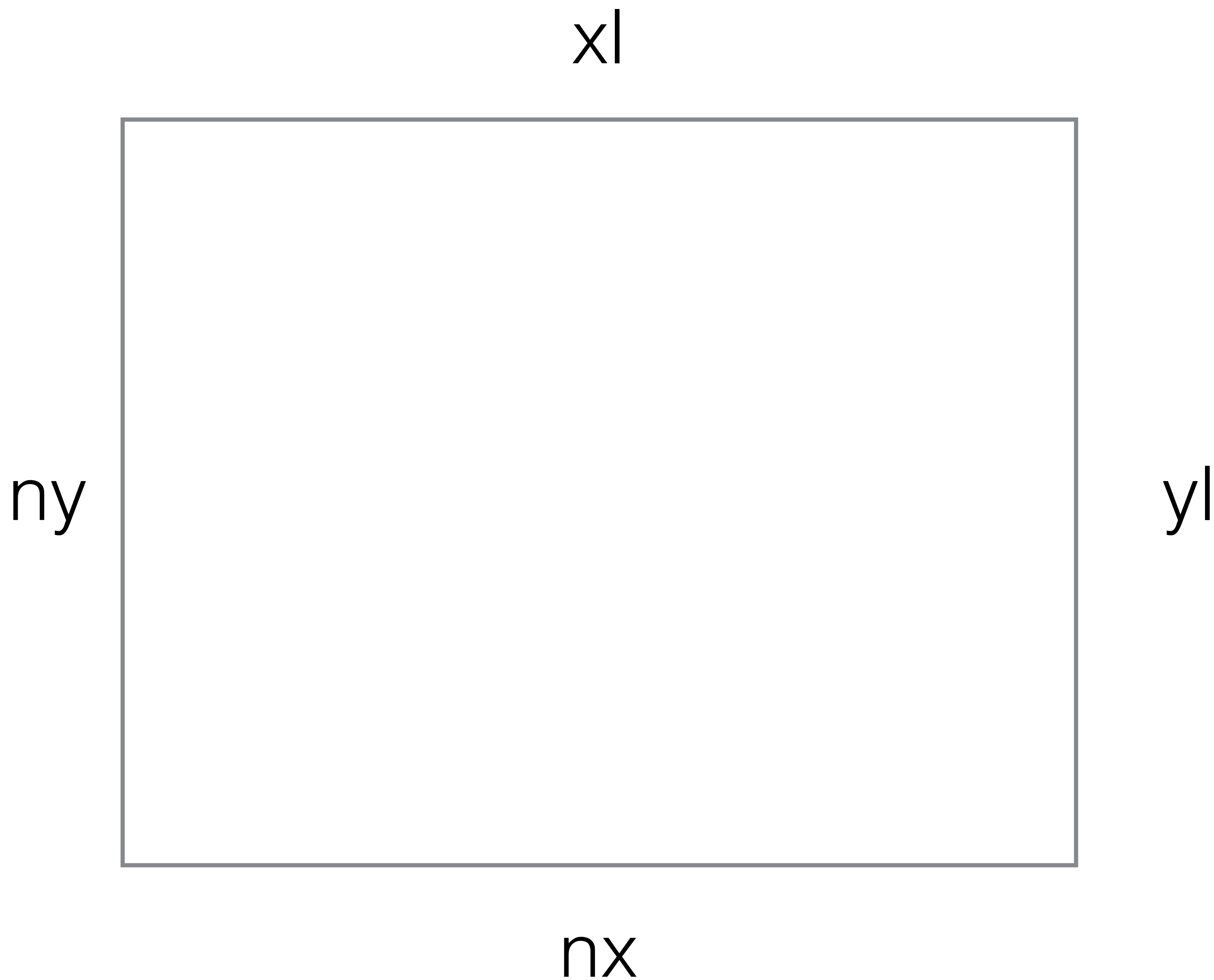
1. Theory
   1. Stream power law
   2. Hillslope processes
   3. Boundary and initial conditions
   4. Spatial and temporal discretisation
   5. Taylor series
   6. Finite difference
   7. Implicit vs explicit time integration
   8. Ordering to compute area and solve equations
2. Constructing a LEM
3. Accuracy and stability
4. Exercices

Building my own Landscape Evolution Model (using the FastScape algorithm)

Rules of the game

- We will each write our version of the FastScape algorithm in a few (7) steps.
- Let’s organise teams
- It is highly recommended that, at the end of each step, we check/verify the results of the computation either by printing the newly evaluated variable/array or plotting it
- This will be done by using our favorite programming language and I will give you the solution in Fortran

0. Let’s define the problem



- xl = 100 x 10^3 m
- yl = 100 x 10^3 m
- nx = 101
- ny = 101
- dt = 1000 yr
- nstep = 1000

We want to solve the stream power equation:

$$\frac{\partial h}{\partial t} = U - KA^m S^n$$

where *h* is topographic height, *t* is time, *U* is uplift rate, *K* is a constant (=), *S* is slope (= dh/dl), *A* is drainage area.

We first assume that *U* is a constant, uniform in space and time. We will later consider the case where *U* varies in space and time.

n will be first assumed to be unity but we will also adapt the algorithm to any value for *n*. *m* will be assumed, as commonly done, to be such that *m/n* ~ 0.4

The problem will also require boundary conditions; 3 cases will be envisaged:

C1: h = 0 (and U = 0) along all 4 boundaries (fixed base level)

C2: h = 0 at y = 0 and y = yl and cyclic boundary conditions between x = and x = xl

C3: h = 0 at y = 0 and reflective boundary conditions at y = yl and cyclic boundary conditions between x = and x = xl

We will also need an initial condition because we have to deal with an evolution equation. We will assume $h = 0$ but we will need to perturb this by adding a 1 m high random noise to seed the network computation.

First we will define a node numbering scheme starting at the bottom left corner ($x = 0$ and $y = 0$) any proceeding along the x axis first then along the y axis.

(ny-1)*nx+1	(ny-1)*nx+2	…	ny*nx
…	…	…	…
2*nx+1	2*nx+2	…	3*nx
nx+1	nx+2	…	2*nx
1	2	…	nx

We can produce a list going from 1 to nx*ny = nn in two ways:

```
do ij=1,nn
…
enddo
```

or

```
do j=1,ny
   do i=1,nx
   ij=i+(j-1)*nx
   …
   enddo
enddo
```

1. Variable initialization

```
xl=100.e3
yl=100.e3
nx=101
ny=101
nn=nx*ny
dx=xl/(nx-1)
dy=yl/(ny-1)
dt=1000.
nstep=1000
call random_number (h)
   do ij=1,nn
   h(ij)=0.+h(ij)
   enddo
```

```
k=1.e-4
n=1
m=0.4
u=2.e-3
```

2. Building receiver array

For each *ij* (=1,…,*nn*), let's first compute the « steepest neighbor node » which we will also call the « receiver node » (*rec(ij)*) as well as the horizontal distance between node *ij* and its receiver node, *length(ij)*:

In this first version of the code we will assume that the 4 boundaries are fixed at base level

```
do ij=1,nn
rec(ij)=ij
length(ij)=0.
enddo

do j=2,ny-1
do i=2,nx-1
ij=i+(j-1)*nx
smax=0.
    do jj=-1,1
    do ii=-1,1
    iii=i+ii
    jjj=j+jj
    ijk=iii+(jjj-1)*nx
        if (ijk.ne.ij) then
        l=sqrt((dx*ii)**2+(dy*jj)**2)
        slope=(h(ij)-h(ijk))/l
            if (slope.gt.smax) then
            smax=slope
            rec(ij)=ijk
            length(ij)=l
            endif
        endif
    enddo
    enddo
enddo
enddo
```

3. Building donor array

From the receiver list, we now need to build the « reversed » list of the donor nodes. For each node *ij*, we need to compute how many neighboring nodes have *ij* as a receiver node and the list of these neighbors. This will mean computing 2 lists, one of length *nn*, called *ndon*, which is the number of donors for each node, and a double indexed array, *donors(8,nn)*, containing up to eight nodes per node (there are 8 neighbors to each node). Don't forget to initialize *ndon* to 0…

```
do ij=1,nn
ndon(ij)=0
enddo

do ij=1,nn
   if (rec(ij).ne.ij) then
   ijk=rec(ij)
   ndon(ijk)=ndon(ijk)+1
   donor(ndon(ijk),ijk)=ij
   endif
enddo
```

4. Building the stack

We now have all the information to build the stack information which will be stored in the array *stack(ij)*, which correspond to the optimum order to solve the equation and its inverse the optimum way to compute the drainage area.

We will start by the nodes on the boundaries or nodes that are local minima (those that are their own receiver), and proceed through the list of donors, recursively. For this we will need to define a recursive routine/function (i.e. a routine that can call itself).

```
nstack=0
   do ij=1,nn
      if (rec(ij).eq.ij) then
      nstack=nstack+1
      stack(nstack)=ij
      call find_stack(ij,donor,ndon,nn,stack,nstack)
      endif
   enddo
```

```
recursive subroutine find_stack
   > (ij,donor,ndon,nn,stack,nstack)

integer donor(8,nn),ndon(nn)

   do k=1,ndon(ij)
   ijk=donor(k,ij)
   nstack=nstack+1
   stack(nstack)=ijk
   call find_stack (ijk,donor,ndon,nn,stack,nstack)
   enddo

return
end
```

5. Compute drainage area

Having built the stack, we can now compute the drainage area by summing elemental areas down the network, i.e. in the reverse stack order.

```
do ij=1,nn
a(ij)=(precipitation*dt)*dx*dy
enddo

do ij=nn,1,-1
ijk=stack(ij)
  if (rec(ijk).ne.ijk) then
  a(rec(ijk))=a(rec(ijk))+a(ijk)
  endif
enddo
```

6. Solve equation

We can now solve the equation. But first we need to compute uplift, by simply adding $U*dt$ step to h

```
do j=2,ny-1
do i=2,nx-1
ij=i+(j-1)*nx
h(ij)=h(ij)+u*dt
enddo
enddo
```

Now we use an implicit finite element scheme to solve the evolution equation:

$$\frac{\partial h}{\partial t} = U - KA^m S$$

$$h_i^{t+} = h_i^t - K \; A_i^m \; \frac{h_i^{t+} - h_r^{t+}}{l_i} \; \Delta t$$

$$h_i^{t+} = \frac{h_i^t + F h_r^{t+}}{1 + F}$$

$$\text{where } F = \frac{KA^m \Delta t}{l_i}$$

```
do ij=1,nn
ijk=stack(ij)
ijr=rec(ijk)
  if(ijr.ne.ijk) then
  l=length(ijk)
  f=k*dt*a(ijk)**m/l
  h(ijk)=(h(ijk)+f*h(ijr))/(1.+f)
  endif
enddo
```

7. Time stepping

To finalize the algorithm, we need to encapsulate the entier procedure (excluding the initialization phase) into a loop over time.

We might also wish to include an output/plot option every *nfreq* step.

```
INITIALIZATION
  do istep=1,nstep
  …
    if ((istep/nfreq)*nfreq.eq.istep) then
    OUTPUT
    endif
  enddo
```

8. Other boundary conditions

To implement the two other boundary conditions we will need to change 2 things in the code, first in the computation of the receiver list and secondly when we impose the uplift rate.

C2:

```
do j=2,ny-1
do i=1,nx
...
```

at (*) add the following lines:

```
if (iii.lt.1) iii=iii+nx
if (iii.gt.nx) iii=iii-nx
...
enddo
enddo
```

C3:

```
do j=2,ny
do i=1,nx
...
```

in addition, at (**) add the following line:

```
if (jjj.gt.ny) jjj=ny
...
enddo
enddo
```

In case C2:

```
do j=2,ny-1
do i=1,nx
ij=i+(j-1)*nx
h(ij)=h(ij)+u*dt
enddo
enddo
```

In case C3:

```
do j=2,ny
do i=1,nx
ij=i+(j-1)*nx
h(ij)=h(ij)+u*dt
enddo
enddo
```

9. n≠1

The case n≠1 is slightly more difficult to handle. The implicit algorithm becomes:

$$h_i^{t+} = h_i^t - F(h_i^{t+} - h_r^{t+})^n$$

$$\text{where } F = \frac{K \;\; A_i^m \;\; \Delta t}{l^n}$$

which is a non linear equation in h_i^{t+}. To solve it we will use a simple iterative technique (Newton-Raphson) to find the root of a non linear function, i.e., x such that $f(x)=0$.

Use n = 2, m = 0.8 and K = 2.e-6

1. Pose $x^{O} = x^{init}$

2. $x^{N} = x^{O} - \dfrac{f(x^{O})}{\frac{\partial f}{\partial x}(x^{O})}$

3. $x^{O} = x^{N}$

4. Goto 2 until: $|x^{O} - x^{N}| < \epsilon$

Here $f(h_i^{t+}) = h_i^{t+} - h_i^{t} + F(h_i^{t+} - h_r^{t+})^{n}$

Such that the iterative step 2 becomes:

$$h_i^{t+,N} = h_i^{t+,O} - \frac{h_i^{t+,O} + h_i^{t} + F(h_i^{t+,O} - h_r^{t+})^{n}}{1 + Fn(h_i^{t+,O} - h_r^{t+})^{n-1}}$$

```
tol=1.e-3
  do ij=1,nn
  ijk=stack(ij)
  ijr=rec(ijk)
    if (ijr.ne.ijk) then
    l=length(ijk)
    fact=k*dt*a(ijk)**m/l**n
    h0=h(ijk)
    hp=h0
111 h(ijk)=h(ijk)-(h(ijk)-h0+
  > fact*(h(ijk)-h(ijr))**n)/
  > (1.+fact*n*(h(ijk)-h(ijr))**(n-1))
    diff=h(ijk)-hp
    hp=h(ijk)
    if (abs(diff).gt.tol) goto 111
    endif
  enddo
```

Have fun…

Try to experiment with variable U both in space and time…

Some important notes on the precision and stability of a numerical integration scheme

Example from the implicit vs explicit time integration schemes applied to the linear diffusion equation

Transport | Mass conservation | Diffusion

$$Q_s = -K\frac{\partial h}{\partial x} \longrightarrow \frac{\partial h}{\partial t} = -\frac{\partial Q_s}{\partial x} \longrightarrow \frac{\partial h}{\partial t} = K\frac{\partial^2 h}{\partial x^2}$$

Explicit finite difference

$$\frac{h_i^{t+\Delta t} - h_i^t}{\Delta t} \approx K\frac{h_{i+1}^t - 2h_i^t + h_{i-1}^t}{\Delta x^2}$$

1 unknown per equation

$$h_i^{t+\Delta t} \approx h_i^t + \frac{K\Delta t}{\Delta x^2}\left(h_{i+1}^t - 2h_i^t + h_{i-1}^t\right)$$

Explicit-implicit finite difference

$$\frac{h_i^{t+\Delta t} - h_i^t}{\Delta t} \approx \frac{h_{i+1}^t - 2h_i^{t+\Delta t} + h_{i-1}^t}{\Delta x^2}$$

$$h_i^{t+\Delta t} = \frac{h_i^t + F(h_{i+1}^t + h_{i-1}^t)}{1 + 2F} \text{ where } F = \frac{K\Delta t}{\Delta x^2}$$

Transport — Mass conservation — Diffusion

$$Q_s = -K\frac{\partial h}{\partial x} \longrightarrow \frac{\partial h}{\partial t} = -\frac{\partial Q_s}{\partial x} \longrightarrow \frac{\partial h}{\partial t} = K\frac{\partial^2 h}{\partial x^2}$$

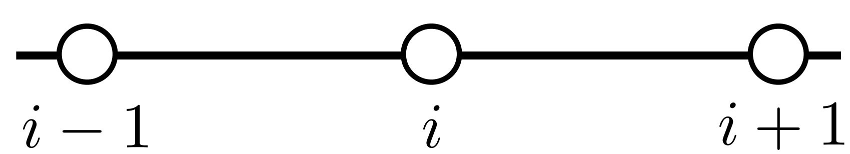
Implicit finite difference

$$\frac{h_i^{t+\Delta t} - h_i^t}{\Delta t} \approx K\,\frac{h_{i+1}^{t+\Delta t} - 2h_i^{t+\Delta t} + h_{i-1}^{t+\Delta t}}{\Delta x^2}$$

3 unknowns per equation

$$-\left(\frac{K\Delta t}{\Delta x^2}\right)h_{i+1}^{t+\Delta t} + \left(1 + 2\frac{K\Delta t}{\Delta x^2}\right)h_i^{t+\Delta t} - \left(\frac{K\Delta t}{\Delta x^2}\right)h_{i-1}^{t+\Delta t} = h_i^t$$

Each node has 2 neighbours



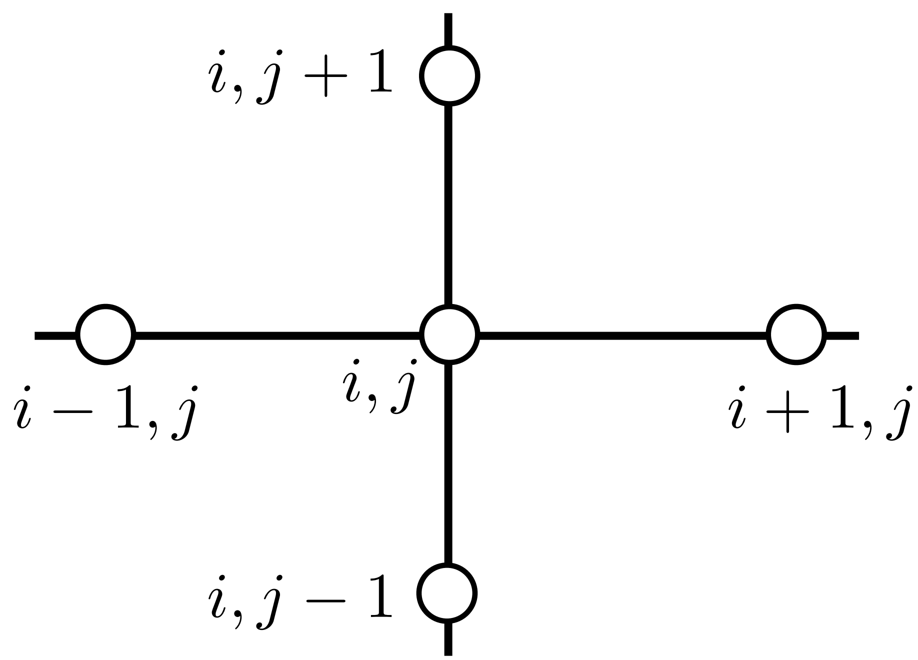
Tri-diagonal system of equations: can be solved in *O(n)* operations

In TWO dimensions

5 unknowns per equation

$$-(\frac{K\Delta t}{\Delta x^2})h_{i+1,j}^{t+\Delta t} - (\frac{K\Delta t}{\Delta y^2})h_{i,j+1}^{t+\Delta t} + (1 + 2\frac{K\Delta t}{\Delta x^2} + 2\frac{K\Delta t}{\Delta y^2})h_{i,j}^{t+\Delta t} - (\frac{K\Delta t}{\Delta x^2})h_{i-1,j}^{t+\Delta t} - (\frac{K\Delta t}{\Delta y^2})h_{i,j-1}^{t+\Delta t} = h_{i,j}^{t}$$

Each node has 5 neighbours



Sparse matrix - Cannot be solved in *O(n)* operations

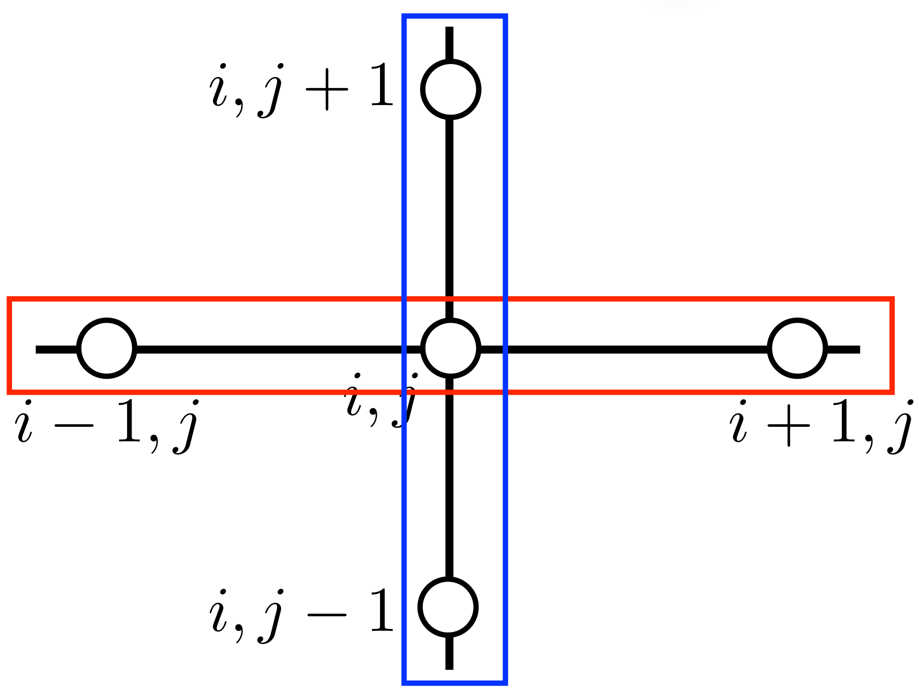
ADI - Alternating Direction Implicit Method

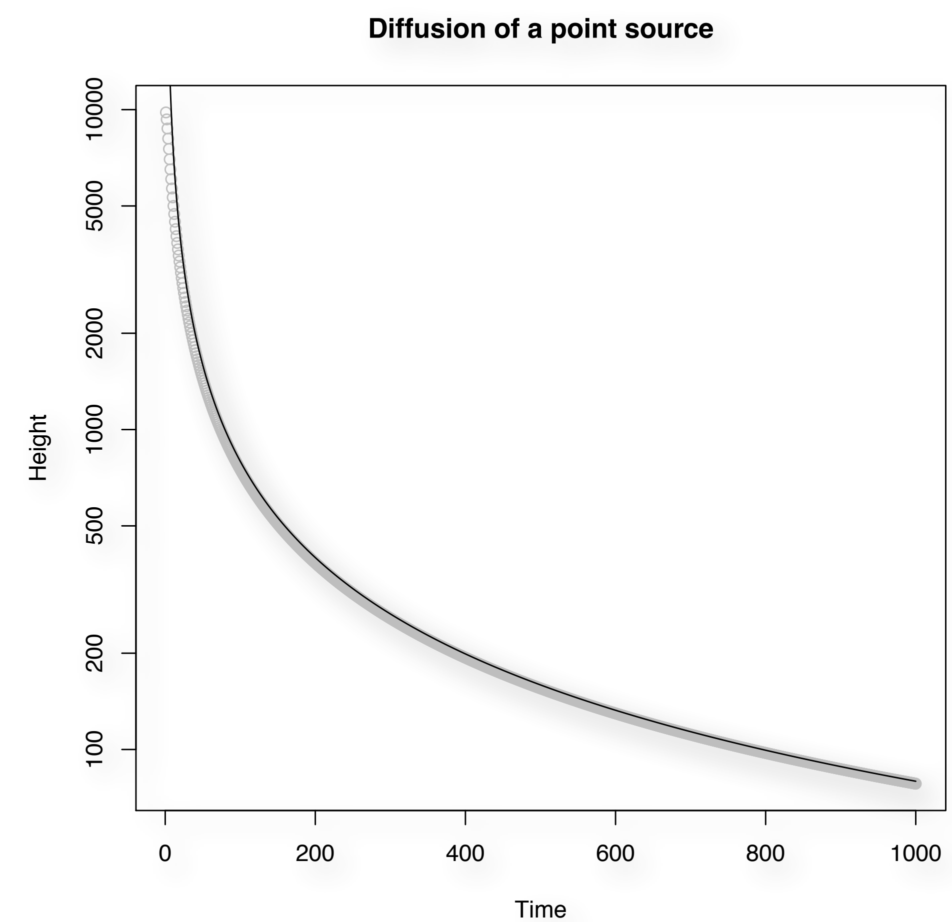
2 equations with 3 unknowns per node

$$-(\frac{K\Delta t}{2\Delta x^2})h_{i+1,j}^{t+\Delta t/2} + (1 + 2\frac{K\Delta t}{2\Delta x^2})h_{i,j}^{t+\Delta t/2} - (\frac{K\Delta t}{2\Delta x^2})h_{i-1,j}^{t+\Delta t/2} = h_{i,j}^{t} - (\frac{K\Delta t}{\Delta y^2})(h_{i,j+1}^{t} - 2h_{i,j}^{t} + h_{i,j-1}^{t})$$

$$-(\frac{K\Delta t}{2\Delta y^2})h_{i,j+1}^{t+\Delta t} + (1 + 2\frac{K\Delta t}{2\Delta y^2})h_{i,j}^{t+\Delta t} - (\frac{K\Delta t}{2\Delta y^2})h_{i,j-1}^{t+\Delta t} = h_{i,j}^{t+\Delta t/2} - (\frac{K\Delta t}{\Delta x^2})(h_{i+1,j}^{t+\Delta t/2} - 2h_{i,j}^{t+\Delta t/2} + h_{i-1,j}^{t+\Delta t/2})$$

Each node has 5 neighbours





Can be solved in *O(n)* operations

Explicit time integration scheme

$$\Delta t < 10^{-3}\tau_{\Delta x} = 10^{-3}\frac{\Delta x^2}{K_D}$$

Stability: you can't transfer information over one grid cell faster than diffusion allows it (Courant condition)

Note: for accuracy you need a small grid cell but this imposes small time steps…

$$\Delta t < 10^{-3}\tau = 10^{-3}\frac{L^2}{K_D}$$

Accuracy: time step has to be a very small fraction of the characteristic time scale (diffusive)

Implicit time integration scheme

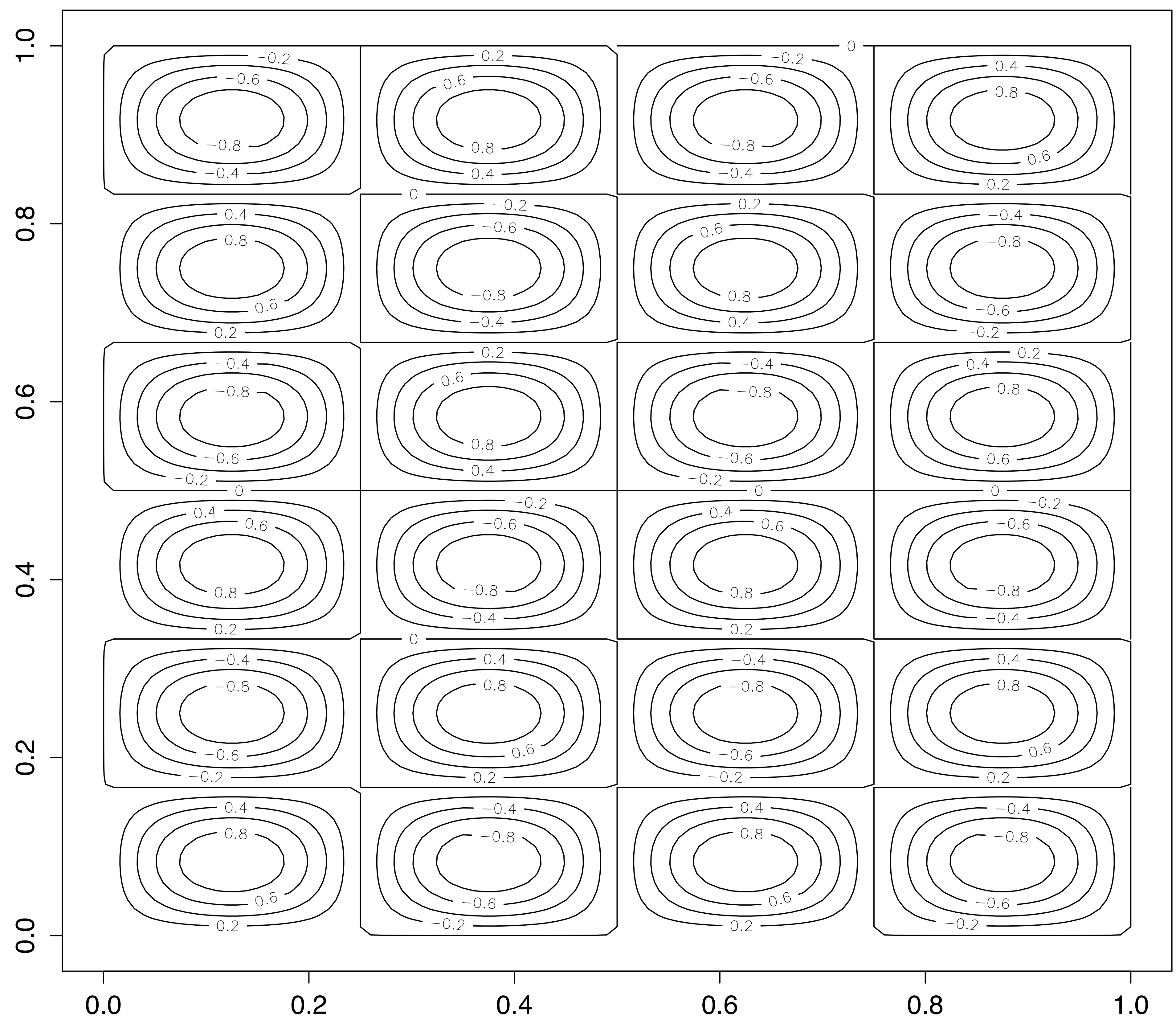
$$\Delta t < 10^{-3}\tau_{\Delta x} = 10^{-3}\frac{\Delta x^2}{K_D}$$

~~Stability: you can't transfer information over one grid cell faster than diffusion allows it (Courant condition)~~

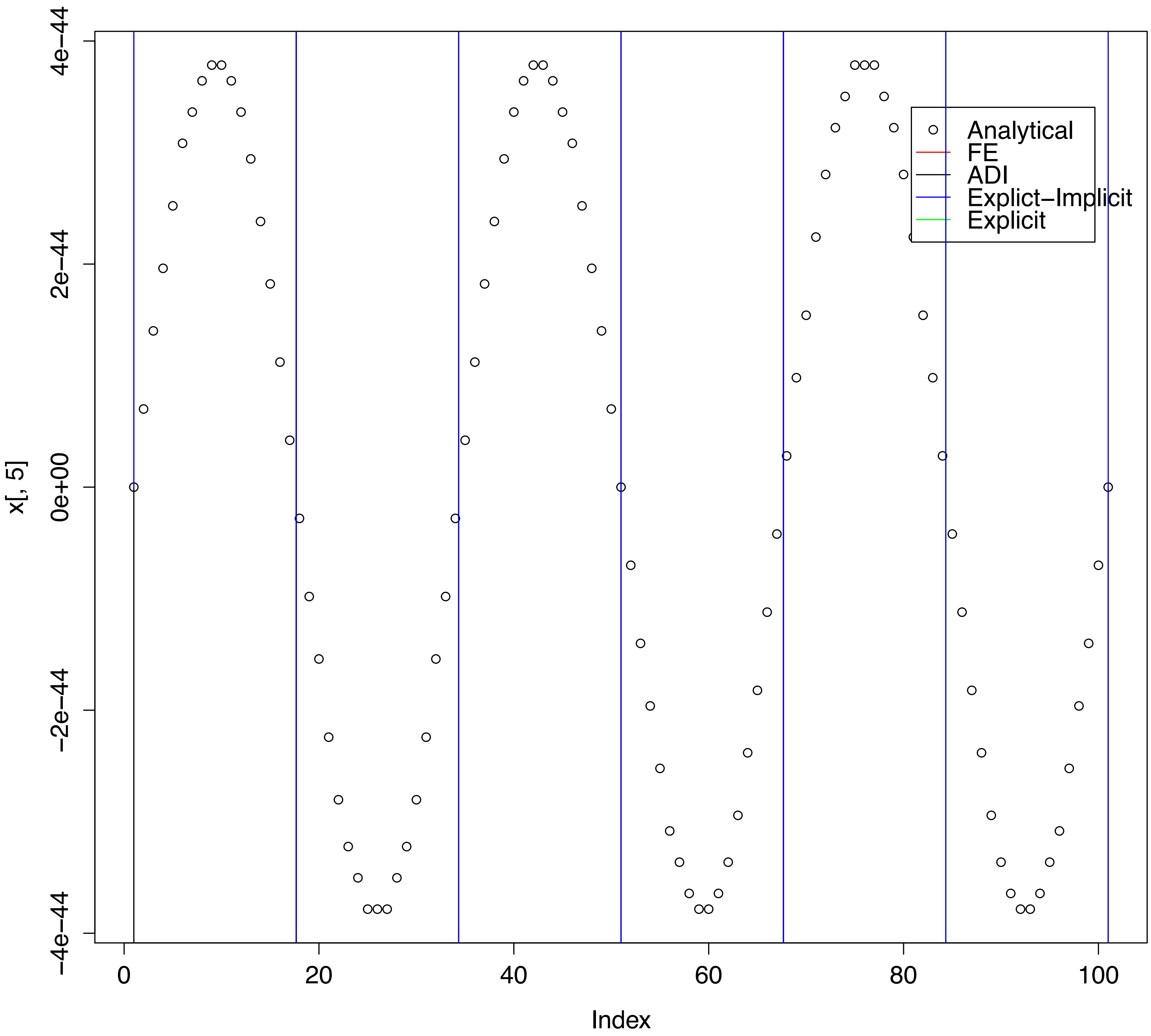
~~Note: for accuracy you need a small grid cell but this imposes small time steps…~~

$$\Delta t < 10^{-\overset{1}{\cancel{3}}}\tau = 10^{-\overset{1}{\cancel{3}}}\frac{L^2}{K_D}$$

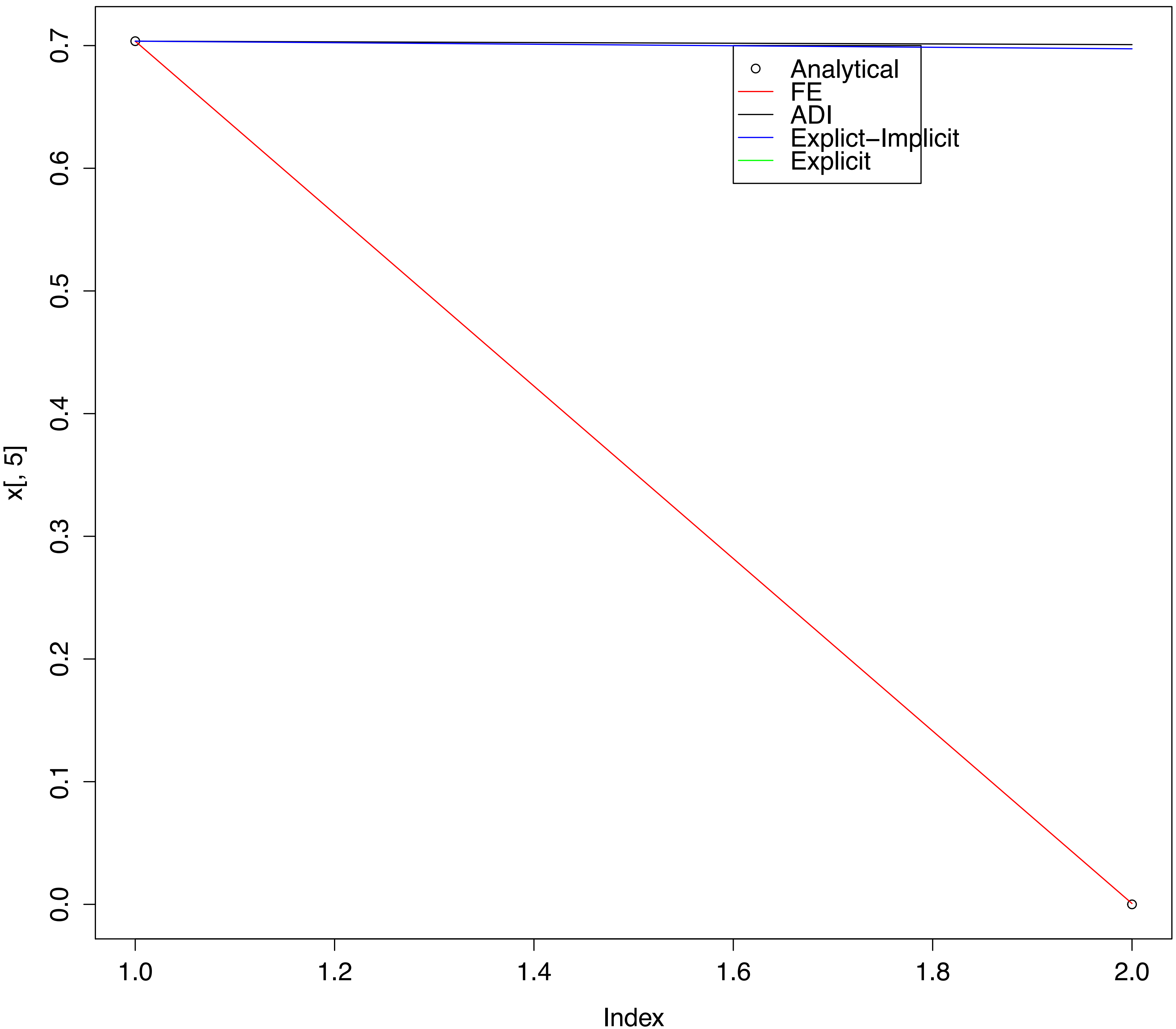
Accuracy: time step has to be a very small fraction of the characteristic time scale (diffusive)



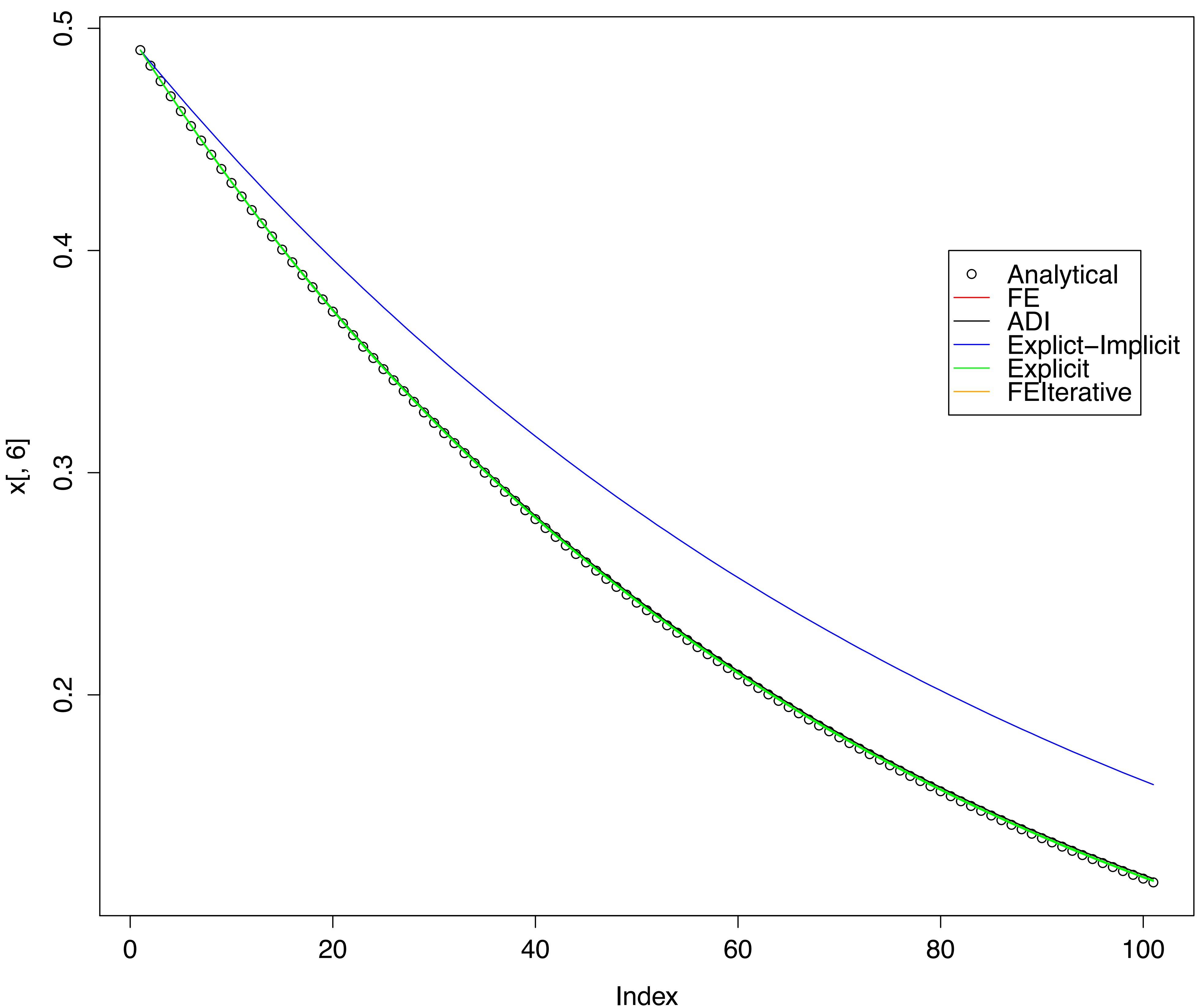
0.0
0.2
0.4
0.6
0.8
1.0



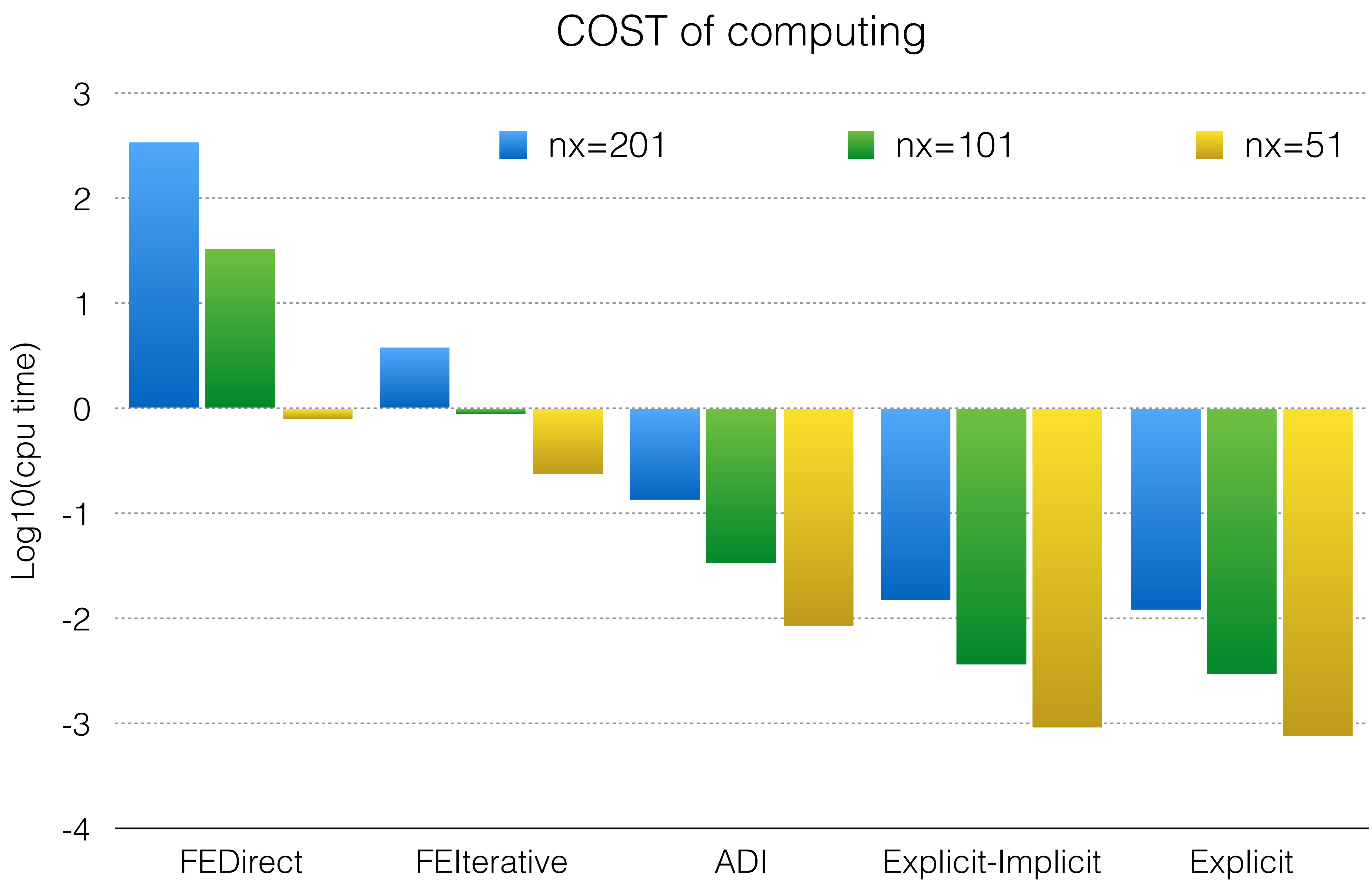
1. 1/10000
2. 1/1000
3. 1/100
4. 1/10
5. 1
6. 10
7. 100



1. 1/10000
2. 1/1000
3. 1/100
4. 1/10
5. 1
6. 10
7. 100
8. 1000



1. nx = 101
2. nx = 201
3. nx = 51


COST of computing
nx=201
nx=101
nx=51
Log10(cpu time)
3
2
1
0
-1
-2
-3
-4
FEDirect
FEIterative
ADI
Explicit-Implicit
Explicit