MATH3104 - Mathematical Biology
Primer - Finite Differences Method
Dietmar Oelz

February 19, 2024

Here we focus on straightforward ways to treat numerically reaction-drift-diffusion equations in 1D. Typically we are looking for solutions $u = u(t, x)$ where $t \geq 0$ is time and $x \in [a, b] \subset \mathbb{R}$.

In the two assignments for this part of MATH3104 we'll restrict to periodic boundary conditions, i.e. $\rho(t, a) = \rho(t, b)$ and $\partial_x \rho(t, a) = \partial_x \rho(t, b)$, therefore all the differential operators and equations we will discuss in this primer are implicitly meant to be coupled to periodic b.c. .

1 Discrete approximation of functions

Numerically, i.e. on the computer, we cannot represent functions on intervals, but only vectors. To approximate a function $u = u(x)$ by a vector $\bar{u} = (u_1, u_2, ..., u_N)$ we may use an equi-distant grid with grid-size $\Delta x = (b - a)/(N - 1)$ and N grid points $x_1 = a$, ..., $x_i = a + i\Delta x$,...,$x_N = b$ such that $u_i \approx u(x_i)$.

For time-dependent problems we write the solution $u(t, x)$ as $\bar{u}^n = (u_1^n, ..., u_N^n)$ where $n = 0, 1, 2...$ is an index of time such that $u_i^n \approx u(n\Delta t, i\Delta x)$. Here $\Delta t > 0$ is the size of the time-steps.

2 Finite differences

The most straightforward way to discretise derivatives is $\partial_x u(x_i) \approx \frac{u_{i+1} - u_i}{\Delta x}$ (forward difference operator), $\partial_x u(x_i) \approx \frac{u_i - u_{i-1}}{\Delta x}$ (backward difference operator) or $\partial_x u(x_i) \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x}$ (central difference operator)

For second derivatives the easiest approach is to combine a forward and a backward difference:

$$\partial_{xx} u(x_i) \approx \frac{\frac{u_{i+1} - u_i}{\Delta x} - \frac{u_i - u_{i-1}}{\Delta x}}{\Delta x} = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}$$

3 Time-stepping for ODEs

To obtain an approximate numerical solution for the ODE

$$\begin{cases} \dfrac{d}{dx} u(x) = f(x, u(x)) \ , \\ u(x_0) = u_0 \ , \end{cases}$$

for $x > x_0$ you might fix the step size Δx, introduce the discretized points $x_i = x_0 + i\,\Delta x$, and formulate the

1. **explict Euler scheme**
$$\frac{u_{i+1} - u_i}{\Delta x} = f(x_i, u_i) \ ,$$

where the right hand side of the equation is evaluated at the grid point i where the solution is known. In this case the update rule is $u_{i+1} = u_i + \Delta x f(x_i, u_i)$. Unfortunately the explicit Euler scheme is only stable if Δx is not too large, which imposes a constraint on your choice of the step size Δx.

2. An alternative method evaluates the right hand side of the equation at the new grid point(the one to be computed in this iterative scheme). This is called **implicit Euler scheme** and boils down to

$$\frac{u_{i+1} - u_i}{\Delta x} = f(x_{i+1}, u_{i+1}) \, .$$

In general you therefore have to solve a non-linear equation to compute the update $u_i \to u_{i+1}$. The implicit scheme is always stable. Therefore, particularly if f is simple enough and you can solve explicitly for u_{i+1}, this approach is beneficial. E.g. if $f(x, u) = -x\,u$ then you need to solve

$$\frac{u_{i+1} - u_i}{\Delta x} = -x_{i+1}\, u_{i+1}$$

for u_{i+1}, i.e. $u_{i+1} = u_i/(1 + \Delta x\, x_{i+1})$ is the update rule in this case. A numerical code (here in Julia) to solve

$$\begin{cases} \dfrac{d}{dx}u(x) = -x\,u(x) \, , \\ u(4) = 5 \, , \end{cases}$$

for $x > 4$ is here:

```
using Plots
const dx=0.01

function main(x0=4, u0=5)
  u=u0
  x=x0
  usav=Float64[]
  xsav=Float64[]
  for i=1:100
    u=u/(1+dx*x)
    push!(usav, u)
    push!(xsav, x)
    x+=dx
  end
  return xsav, usav
end

xvec, uvec=main()
plot(xvec, uvec)
```

4 Discrete diffusion operator in 1D

The linear diffusion operator $u \mapsto D\partial_{xx}$ assigns the 2nd derivative (times D) to a given function. Analogously the discrete diffusion operator is a linear map - a matrix - applied to a vector of

grid values $\bar{u} = (u_1, ..., u_N)$, namely

$$\begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \mapsto \frac{1}{(\Delta x)^2} \begin{pmatrix} u_2 - 2u_1 + u_N \\ u_3 - 2u_2 + u_1 \\ \vdots \\ u_N - 2u_{N-1} + u_{N-2} \\ u_1 - 2u_N + u_{N-1} \end{pmatrix}.$$

In matrix notation we obtain $\bar{u} = (u_1, ..., u_N) \mapsto D\,M\bar{u}$, where

$$M = \frac{1}{(\Delta x)^2} \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 & -2 \end{pmatrix}.$$

Note that the periodic boundary condition implies that x_N is taken as the "left" neighbour of x_1, which itself is taken as the "right" neighbour of x_N.

5 Linear Elliptic equation (Poisson, etc)

To solve a stationary (not time-dependent) equation such as $0 = D\Delta u - \alpha u + f$ solve

$$0 = DM\bar{u} - \alpha\bar{u} + \bar{f}$$

i.e. $\bar{u} = (D\,M - \alpha\mathbb{1})^{-1}(-\bar{f})$, where $\bar{f}$ is a vector corresponding to the discretised version of the right hand side function f and $\mathbb{1}$ is the identity matrix.

6 Diffusion equation

If we add dependence on time then the canonical 2nd order PDE is the diffusion equation

$$\partial_t \rho = D\partial_{xx}\rho$$

with initial datum $\rho(t = 0, .) = \rho_I(.)$ and diffusion coefficient $D > 0$.

1. Start to iterate in time with the vector $\bar{u}^0$ which is a discretised version of ρ_I. For $\rho_I = \delta(x - c)$ (Dirac-δ, i.e. one particle located at $x = c$), take

   $$\bar{u}_i^0 = \begin{cases} \frac{1}{\Delta x} & i = i_0 \\ 0 & \text{otherwise} \end{cases}$$

   where i_0 is the index which corresponds to the position of the δ-peak given by c. Note that this guarantees that integration of $\bar{u}^0$ using a straightforward quadrature formulate such as $\sum_i \bar{u}_i^0 \Delta x$ (in Matlab: `sum(ubar0)*deltax`) returns 1 just as for the Dirac delta.

2. To compute $\bar{u}^1$ given $\bar{u}^0$, and then $\bar{u}^2$ given $\bar{u}^1$, etc. (time-stepping) use schemes such as

   (a) Explicit Euler: $\frac{\bar{u}^n - \bar{u}^{n-1}}{\Delta t} = D\,M\bar{u}^{n-1}$, hence $\bar{u}^n = \bar{u}^{n-1} + \Delta t D\,M\bar{u}^{n-1}$ Attention: make sure to satisfy the diffusive CFL-condition, $D\Delta t/\Delta x^2 \leq 1/2$, otherwise the scheme is unstable.

   Note that if you solve the diffusion equation starting from a delta-Distribution with this explicit scheme and the optimal value of Δt according to the diffusive CFL-condition, you may obtain zeros in every second grid point. In this case, to avoid that, just set Δt to a slightly smaller value.

(b) Implicit euler $\frac{\bar{u}^n - \bar{u}^{n-1}}{\Delta t} = D\,M\bar{u}^n$, hence $\bar{u}^n = (\mathbb{1} - \Delta t DM)^{-1}\,\bar{u}^{n-1}$. Note that implicit schemes are typically more stable than explicit ones, less restrictive when it comes to step sizes, but harder to implement.

(c) or a midpoint rule such as $\frac{\bar{u}^n - \bar{u}^{n-1}}{\Delta t} = D\,\frac{1}{2}M(\bar{u}^n + \bar{u}^{n-1})$, etc.

Use may use the following scripts as a skeleton for your numerical codes. It approximates the solution of

$$\begin{cases} \partial_t \rho = D\Delta\partial_{xx}\rho\ , \\ \rho(0) = \rho(10)\ , \\ \rho(t=0,x) = 100 \times \delta(x-5)\ . \end{cases}$$

Julia

```
using Plots
using LinearAlgebra

const a, b = 0.0, 10.0
const Dcoef=0.1
const N=100
const dt, dx=0.1, (b-a)/N

const xval=a:dx:(b-dx)
const M= diff(diff([[zeros(1, N-1) 1.0];
  I(N); [1.0 zeros(1, N - 1)]], dims = 1),
  dims = 1) / dx^2

function main()
  rho=zeros(N); rho[div(N,2)]=100.0 / dx
  t=0.0
  while t<20.0
    t=t+dt
    rhoold=rho
    rho=(Id-Dcoef*dt*M)\rhoold
    plot(xval, rho, ylim=(0.0, 100.0),
      show=true)
    #sleep(0.02)
  end
end

main()
```

MATLAB

```
a=0; b=10;
Dcoef=0.1;
N=100;

deltax=(b-a)/N;
deltat=0.1;
xval=linspace(a, b-deltax, N);

M=diff(diff([[zeros(1, N-1), 1]; eye(N); ...
  [1, zeros(1, N-1)]]))/deltax^2;
h=figure;

rho=zeros(N, 1); rho(N/2)=100/deltax;
t=0;
while t<100
  t=t+deltat;
  rhoold=rho;
  rho=(eye(N)-Dcoef*deltat*M)\rhoold;

  figure(h);
  plot(xval, rho);
  %yl=ylim;
  ylim([0, 100]);
  drawnow
end
```

7 Scalar conservation laws (1D)

Lax-Friedrichs method

$$\partial_t u + \partial_x(f(u)) = 0\ .$$

The simplest scheme (1st order) is the Lax-Friedrichs method, i.e.

$$\frac{\bar{u}_i^n - \frac{1}{2}\left(\bar{u}_{i+1}^{n-1} + \bar{u}_{i-1}^{n-1}\right)}{\Delta t} + \frac{f(u_{i+1}^{n-1}) - f(u_{i-1}^{n-1})}{2\Delta x} = 0\ ,$$

which implies that

$$\bar{u}_i^n = \frac{1}{2}\left(\bar{u}_{i+1}^{n-1} + \bar{u}_{i-1}^{n-1}\right) - \frac{\Delta t}{2\Delta x}\left(f(u_{i+1}^{n-1}) - f(u_{i-1}^{n-1})\right)\ . \tag{1}$$

Attention: The following CFL-condition,

$$\max_{i=1..N}\left|\frac{f(\bar{u}_i^{n-1})}{\bar{u}_i^{n-1}}\right|\frac{\Delta t}{\Delta x} \leq 1/2\ , \tag{2}$$

guarantees the stability of the scheme. Also bear in mind that this scheme generates significant numerical diffusion, unless the left hand side in (2) is close to its upper bound and unless the grid is sufficiently fine (Δt, Δx small). Therefore it might be a good idea to adapt the step-size Δt in every time-step.

Note that as before the periodic boundary condition is implemented identifying x_{N+1} with x_1 and x_0 with x_N. **To obtain the shifted vectors** $(\bar{u}_{i\pm1})$**, you may want to use MATLAB-*circshift* which implements the periodic boundary "automatically".**

```
circshift([1.0; 2.0; 4.0], 1)=[4.0; 1.0; 2.0]
circshift([1.0; 2.0; 4.0], -1)=[2.0; 4.0; 1.0]
```

and so $\bar{u}_{i\pm1} = \text{circshift}(\bar{u}_i, \mp 1)$.

E.g. if the flux is given by $f(u) = u\partial_x S$, then, starting with a discrete version of S, use circshift to obtain a finite difference approximation of $\partial_x S$, here written as $(\Delta S)_i$, and then set $f(u_i^{n-1}) = u_i(\Delta S)_i$.

Upwind-method in 1D

An even simpler method which works whenever the flux has a sign – either positive or negative – and which is more efficient (less numerical diffusion) than the Lax-Friedrichs method is the Upwind-scheme for the advection equation:

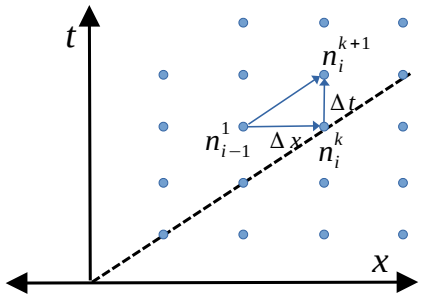
$$\frac{n_i^{k+1} - n_i^k}{\Delta t} + v\frac{n_i^k - n_{i-1}^k}{\Delta x} = 0\ .$$

Here we assume that $v > 0$, therefore to upwind we take the left difference quotient! This implies that

$$n_i^{k+1} = n_i^k\left(1 - v\frac{\Delta t}{\Delta x}\right) + v\frac{\Delta t}{\Delta x}n_{i-1}^k$$

i.e. if the grid is adapted to the velocity: $\Delta x/\Delta t = v$, then this boils down to

$$n_i^{k+1} = n_{i-1}^k \quad \text{"one step left and back in time!"}$$



Note that if the sign of v is negative, we have to take the right-handed difference quotient for the spatial derivative which corresponds to how "information" is flowing in this case: from right to left.

$$\frac{n_i^{k+1} - n_i^k}{\Delta t} + v\frac{n_i^k - n_{i+1}^k}{\Delta x} = 0\ .$$

8 Putting things together: reaction-drift-diffusion

Imagine you want to combine diffusion and something else such as reaction and/or drift,

$$\partial_t u + \partial_x(f(u)) = D\partial_{xx}u + g(u)\ .$$

1. One approach is to compute an explicit time stepping of the entire equation,

$$\bar{u}_i^n = \frac{1}{2}\left(\bar{u}_{i+1}^{n-1} + \bar{u}_{i-1}^{n-1}\right) - \frac{\Delta t}{2\Delta x}\left(f(u_{i+1}^{n-1}) - f(u_{i-1}^{n-1})\right) + (\Delta t D\, M\bar{u}^{n-1})_i + \Delta t\, g(u_i^{n-1}) \,.$$

   Depending on whether drift or diffusion dominates, you will need to satisfy either the diffusive or the hyperbolic CFL conditions (and be aware of the magnitude of numerical diffusion).

2. Fully implicit scheme such as $\frac{\bar{u}^n - \bar{u}^{n-1}}{\Delta t} = D\, M\bar{u}^n + g(\bar{u}^n, x, t)$ which needs to be solved either exactly using Newton iteration (if f is non-linear) or approximately (only the first or a few Newton iterations).

3. You might also want to convert some $\bar{u}^{n-1}$ into $\bar{u}^n$ to obtain (semi-)implicit schemes with better stability. You might try (here for the reaction-diffusion equation $\partial_t u = \Delta u + u(1-u)$),

$$\frac{\bar{u}^n - \bar{u}^{n-1}}{\Delta t} = M\bar{u}^n + \bar{u}^{n-1}(1 - \bar{u}^n)$$

   or

$$\frac{\bar{u}^n - \bar{u}^{n-1}}{\Delta t} = M\bar{u}^n + \bar{u}^n(1 - \bar{u}^{n-1}) \,,$$

   making sure that the resulting equation is linear in $\bar{u}^n$.

4. **Another approach - the one I recommend for its simplicity** - is to use a splitting method where - within one time-step - you compute e.g.

   (a) one time-step of a scheme for the diffusion equation (see section 6) such as $\bar{u}^n = (\mathbb{1} - \Delta t D M)^{-1}\, \bar{u}^{n-1}$ (fully implicit).

   (b) then rename ($\bar{u}^{n-1} = \bar{u}^n$!) and compute one time-step of an ODE-scheme for the reaction part of the equation such as

$$\bar{u}_i^n = \bar{u}_i^{n-1} + \Delta t\, \bar{u}_i^{n-1}(1 - \bar{u}_i^n) \,,$$

   i.e. $\bar{u}_i^n = \left(\bar{u}_i^{n-1}(1+\Delta t)\right) / \left(1 + \Delta t\, \bar{u}_i^{n-1}\right)$.

   (c) then rename ($\bar{u}^{n-1} = \bar{u}^n$!) and compute one time-step for the drift part (assignment 8) such as Lax-Friedrichs Method (1).

5. Ex. The following Julia code using the upwind method and the splitting method (and a simple quadrature formula) to approximate the solution $n : \mathbb{R}_+ \mapsto \mathbb{R}_+$ of the structured model

$$\begin{cases} \partial_t n + \partial_s n + \mu(s) n = 0 \,, \\ \rho(t, s=0) = \displaystyle\int_0^\infty \mu(\tilde{s}) n(t, \tilde{s})\, d\tilde{s} \,, \\ \rho(t=0, s) = n_I(s) \,. \end{cases}$$

```
using Plots
using Printf
const smax=10.0
const N=100
const sval=range(0.0, smax, length=N)
const dt=sval[2]-sval[1] #sec
mu(s)=s #off-rate
nI(s)=max(10-s,0)  #initial condition

function main()
```

```
  plot(show=true)
  n=nI.(sval)
  t=0.0
  for i=1:100
    t+=dt
    for j=1:N
      n[j]=n[j]/(1+dt*(mu(sval[j])))
    end
    n0=sum(n .* mu.(sval))*dt
    nold=n
    n=[n0; nold[1:end-1]]
    plot(sval, n, show=true, title=@sprintf("t=%f", t))
    println(sum(n)*dt)
    sleep(0.2)
  end
  return S, n
end
S, n=main()
plot(sval, n)
```

9 Systems of equations

Again the splitting scheme is the easiest way to deal with systems of equations. For the parabolic-elliptic Keller-Segel system, in every time step you may want to

1. obtain the vector $(S_i^{n-\frac{1}{2}})_i$ solving the elliptic equation for the chemoattractant given the right hand side $\bar{u}^{n-1}$ (see section 5),
2. and then compute $\bar{u}^n$ as time-update of the drift-diffusion equation given the chemoattractant concentration $(S_i^{n-\frac{1}{2}})_i$ (see section 8).

Likewise, for the Turing system in every time-step you may want to

1. find $(u_i^{n-\frac{1}{2}})$ computing one Euler step with the reaction term $f(u_i^{n-1}, v_i^{n-1})$ and then compute (u_i^n) by solving diffusion equation.
2. and then apply the same procedure $(v_i^{n-1})_i$ to obtain $(v_i^n)_i$.