Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.
- You should include tests of your algorithms. This could be represented by unit tests and/or tests of mathematical aspects of the algorithm.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Use your github address to hand in your projects.
- Make a folder for each project. For each project you should have three folders: one for the code files, one for the report and finally a folder with specific benchmark calculations. The latter can be in the form of output from your code for a selected set of runs and input parameters.

Finally, we encourage you to work two and two together. Optimal working groups consist of 2-3 students. You can then hand in a common report.

Project 4, Diffusion of neurotransmitters in the synaptic cleft, April 29

The dominant way of transporting signals between neurons (nerve cells) in the brain is by means of diffusion of particular signal molecules called *neurotransmitters* across the synaptic cleft separating the cell membranes of the two cells. A drawing of a synapse is given in Fig. 1.

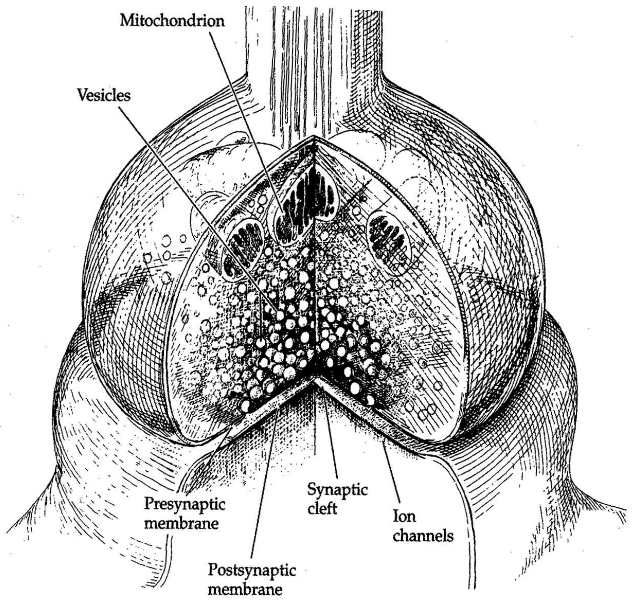


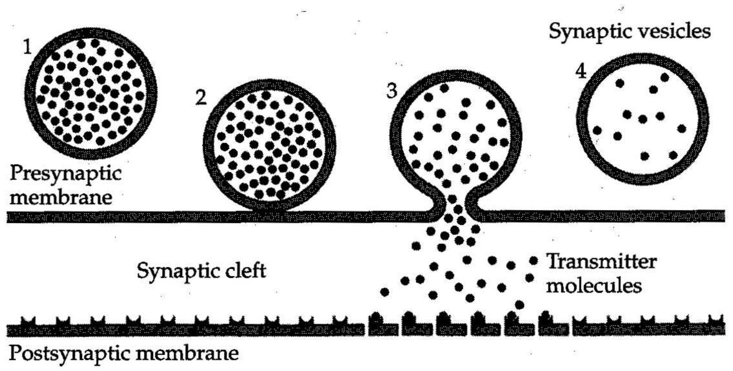
Figure 1: Drawing of a synapse. The axon terminal is the knoblike structure and the spine of the receiving neuron is the bottom one. The synaptic cleft is the small space between the presynaptic (axon) and postsynaptic (dendritic spine) membrane. (From Thompson: "The Brain", Worth Publ., 2000)

Following the arrival of an action potential in the axon terminal a process is initiated in which (i) vesicles inside the axon terminal (filled with neurotransmitter molecules) merge with the presynaptic (axon) membrane and (ii) release neurotransmitters into the synaptic cleft. These neurotransmitters diffuse across the synaptic cleft to receptors on the postsynaptic side which "receives" the signal. A schematic illustration of this process is shown in Fig. 2(left). Since the transport process in the synaptic cleft is governed by diffusion, we can describe it mathematically by

$$\frac{\partial u}{\partial t} = D\nabla^2 u, \tag{1}$$

where u is the concentration of the particular neurotransmitter, and D is the diffusion coefficient of the neurotransmitter in this particular environment (solvent in synaptic cleft).

If we assume (i) that the neurotransmitter is released roughly equally on the "presynaptic" side of the synaptic cleft, and (ii) that the synaptic cleft is roughly equally wide across the



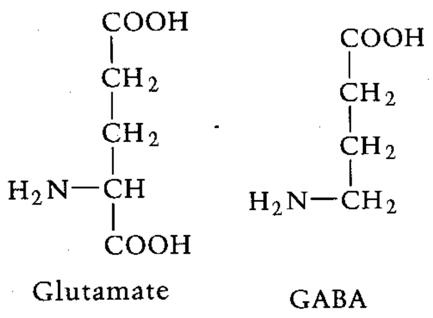


Figure 2: Left: Schematic drawing of the process of vesicle release from the axon terminal and release of transmitter molecules into the synaptic cleft. (From Thompson: "The Brain", Worth Publ., 2000). Right: Molecular structure of the two important neurotransmitters *glutamate* and *GABA*.

whole synaptic terminal, we can, given the large area of the synaptic cleft compared to its width, assume that the neurotransmitter concentration only varies in the direction across the synaptic cleft (from presynaptic to postsynaptic side). We choose this direction to be the x-direction (see Fig. 3). In this case $u(\mathbf{r}) = u(x)$, the diffusion equation reduces to

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}. \tag{2}$$

Immediately after the release of a neurotransmitter into the synaptic cleft ($t = 0$) the concentration profile in the x-direction is given by

$$u(x, t = 0) = N\delta(x), \tag{3}$$

where N is the number of particle released into the synaptic cleft per area of membrane.

To get an idea over the time-dependence of the neurotransmitter concentration at the postsynaptic side ($x = d$), we can look at the solution of a "free" random walk (i.e., no obstacles or particle absorbers in either direction). The solution of Eq. (2) with the initial condition in Eq. (3) is given by (see Nelson: *Biological Physics*, p. 143 or Lectures notes chapter 12.3)

$$u(x, t) = \frac{N}{\sqrt{4\pi Dt}}e^{-x^2/4Dt} \quad . \tag{4}$$

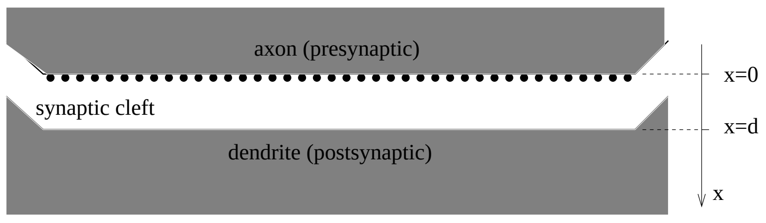


Figure 3: Schematic drawing of the synaptic cleft in our model. The black dots represent neurotransmitter molecules, and the situation shown corresponds to the situation immediately after neurotransmitter release into the synaptic cleft.

The concentration at the postsynaptic side $u(d,t)$ approaches 0 in the limit $t \to 0$ and $t \to \infty$.

The above assumption regarding the neurotransmitter molecules undergoing a "free" random walk, is obviously a simplification. In the true diffusion process in the synaptic cleft the neurotransmitter molecules will, for example, occasionally bump into the presynaptic membrane they came from. Also at the postsynaptic side the neurotransmitters are absorbed by receptors located on the postsynaptic cell membrane and are thus (temporally) removed from the solution.

To approach this situation in our mathematical model we can impose the following boundary and initial conditions with $x \in [0,d]$

$$u(x=0,t>0)=u_0,\quad u(x=d,\text{all } t)=0,\quad u(0<x<d,t<0)=0\ . \tag{5}$$

Hereafter we set $d=1$. This corresponds to that (i) for $t<0$ there are no neurotransmitters in the synaptic cleft, (ii) for $t>0$ the concentration of neurotransmitters at the presynaptic boundary of the synaptic cleft ($x=0$) is kept *fixed*ă at $u=u_0=1$ in our case, and (iii) that the postsynaptic receptors immediately absorb nearby neurotransmitters so that $u=0$ on the postsynaptic side of the cleft ($x=d=1$).

The full solution of the diffusion equation with boundary/initial conditions in Eq. (5) can be found in a closed form. We will use this solution to test our numerical calculations.

We are thus looking at a one-dimensional problem

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, t>0, x\in[0,d]$$

or

$$u_{xx}=u_t,$$

with initial conditions, i.e., the conditions at $t=0$,

$$u(x,0)=0 \qquad 0<x<d$$

with $d=1$ the length of the x-region of interest. The boundary conditions are

$$u(0,t)=1 \qquad t>0,$$

and

$$u(d,t)=0 \qquad t>0.$$

In this project we want to study the numerical stability of three methods for partial differential equations (PDEs). These methods are

1. The explicit forward Euler algorithm with discretized versions of time given by a forward formula and a centered difference in space resulting in

$$u_t \approx \frac{u(x,t+\Delta t)-u(x,t)}{\Delta t} = \frac{u(x_i,t_j+\Delta t)-u(x_i,t_j)}{\Delta t}$$

and

$$u_{xx} \approx \frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{\Delta x^2},$$

or

$$u_{xx} \approx \frac{u(x_i+\Delta x,t_j)-2u(x_i,t_j)+u(x_i-\Delta x,t_j)}{\Delta x^2}.$$

2. The implicit Backward Euler with

$$u_t \approx \frac{u(x,t) - u(x,t-\Delta t)}{\Delta t} = \frac{u(x_i,t_j) - u(x_i,t_j-\Delta t)}{\Delta t}$$

and

$$u_{xx} \approx \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2},$$

or

$$u_{xx} \approx \frac{u(x_i+\Delta x,t_j) - 2u(x_i,t_j) + u(x_i-\Delta x,t_j)}{\Delta x^2},$$

3. Finally we use the implicit Crank-Nicolson scheme with a time-centered scheme at $(x, t+\Delta t/2)$

$$u_t \approx \frac{u(x,t+\Delta t) - u(x,t)}{\Delta t} = \frac{u(x_i,t_j+\Delta t) - u(x_i,t_j)}{\Delta t}.$$

The corresponding spatial second-order derivative reads

$$u_{xx} \approx \frac{1}{2}\left(\frac{u(x_i+\Delta x,t_j) - 2u(x_i,t_j) + u(x_i-\Delta x,t_j)}{\Delta x^2} + \right.$$

$$\left.\frac{u(x_i+\Delta x,t_j+\Delta t) - 2u(x_i,t_j+\Delta t) + u(x_i-\Delta x,t_j+\Delta t)}{\Delta x^2}\right).$$

Note well that we are using a time-centered scheme wih $t + \Delta t/2$ as center.

a) Find the closed form solution to this problem. You will need this in order to study the numerical accuracy of your results. To find the closed-form solution, we will need the stationary solution (steady-state solution). The solution to the steady-state problem is on the form $u(x) = Ax + b$. The solution for the steady-state case u_s that obeys the above boundary conditions is

$$u_s(x) = 1 - x.$$

You can use this solution to define a new function $v(x) = u(x) - u_s(x)$ with boundary conditions $v(0) = v(d) = 0$. The latter is easier to solve both numerically and on a closed form.

b) Write down the algorithms for these three methods and the equations you need to implement. For the implicit schemes show that the equations lead to a tridiagonal matrix system for the new values.

c) Find the truncation errors of these three schemes and investigate their stability properties.

d) Implement the three algorithms in the same code and perform tests of the solution for these three approaches for $\Delta x = 1/10$, $\Delta x = 1/100$ using Δt as dictated by the stability limit of the explicit scheme. Study the solutions at two time points t_1 and t_2 where $u(x, t_1)$ is smooth but still significantly curved and $u(x, t_2)$ is almost linear, close to the stationary state. Remember that for solving the tridiagonal equations you can use your code from project 1.

e) Compare the solutions at t_1 and t_2 with the closed form result for the continuous problem. Which of the schemes would you classify as the best?

f) The above problem can be solved using Monte Carlo methods and random walks. We follow here Farnell and Gibson in Journal of Computational Physics, volume **208**, pages 253-265 (2005). Choose a constant step length $l_0 = \sqrt{2D\Delta t}$ and equal probability for jumping left and right. Set up the algorithm for solving the above diffusion problem and write a code to do it. Compare your results with those from the partial differential equation solution and comment the results.

g) Change the above stepsize by using a Gaussian distribution with mean value 1 and standard deviation 0. The step length of the random walker is now $l_0 = \sqrt{2D\Delta t}\xi$, where ξ is random number chosen from the above Gaussian distribution. Implement this stepsize to the program from f) and compare the results and comment.

We include for the sake of completeness a code which computes random numbers according to a normal distribution. You can alternatively use the **random** class of C++.

```
// function for gaussian random numbers
double gaussian_deviate(long *);

// ran2 for uniform deviates, initialize with negative seed.
double ran2(long *);


// random numbers with gaussian distribution
double gaussian_deviate(long * idum)
{
  static int iset = 0;
  static double gset;
  double fac, rsq, v1, v2;

  if ( idum < 0) iset =0;
  if (iset == 0) {
    do {
      v1 = 2.*ran2(idum) -1.0;
      v2 = 2.*ran2(idum) -1.0;
      rsq = v1*v1+v2*v2;
    } while (rsq >= 1.0 || rsq == 0.);
    fac = sqrt(-2.*log(rsq)/rsq);
    gset = v1*fac;
    iset = 1;
    return v2*fac;
  } else {
    iset =0;
    return gset;
  }
} // end function for gaussian deviates

/*
```

```
** The function
**         ran2()
** is a long periode (> 2 x 10^18) random number generator of
** L'Ecuyer and Bays-Durham shuffle and added safeguards.
** Call with idum a negative integer to initialize; thereafter,
** do not alter idum between sucessive deviates in a
** sequence. RNMX should approximate the largest floating point value
** that is less than 1.
** The function returns a uniform deviate between 0.0 and 1.0
** (exclusive of end-point values).
*/

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

double ran2(long *idum)
{
  int            j;
  long           k;
  static long    idum2 = 123456789;
  static long    iy=0;
  static long    iv[NTAB];
  double         temp;

  if(*idum <= 0) {
    if(-(*idum) < 1) *idum = 1;
    else             *idum = -(*idum);
    idum2 = (*idum);
    for(j = NTAB + 7; j >= 0; j--) {
      k     = (*idum)/IQ1;
      *idum = IA1*(*idum - k*IQ1) - k*IR1;
      if(*idum < 0) *idum +=  IM1;
      if(j < NTAB)  iv[j]  = *idum;
    }
    iy=iv[0];
  }
  k     = (*idum)/IQ1;
  *idum = IA1*(*idum - k*IQ1) - k*IR1;
  if(*idum < 0) *idum += IM1;
  k     = idum2/IQ2;
  idum2 = IA2*(idum2 - k*IQ2) - k*IR2;
  if(idum2 < 0) idum2 += IM2;
  j     = iy/NDIV;
  iy    = iv[j] - idum2;
  iv[j] = *idum;
  if(iy < 1) iy += IMM1;
  if((temp = AM*iy) > RNMX) return RNMX;
  else return temp;
}
#undef IM1
#undef IM2
#undef AM
#undef IMM1
#undef IA1
#undef IA2
#undef IQ1
#undef IQ2
#undef IR1
#undef IR2
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX

// End: function ran2()
```