

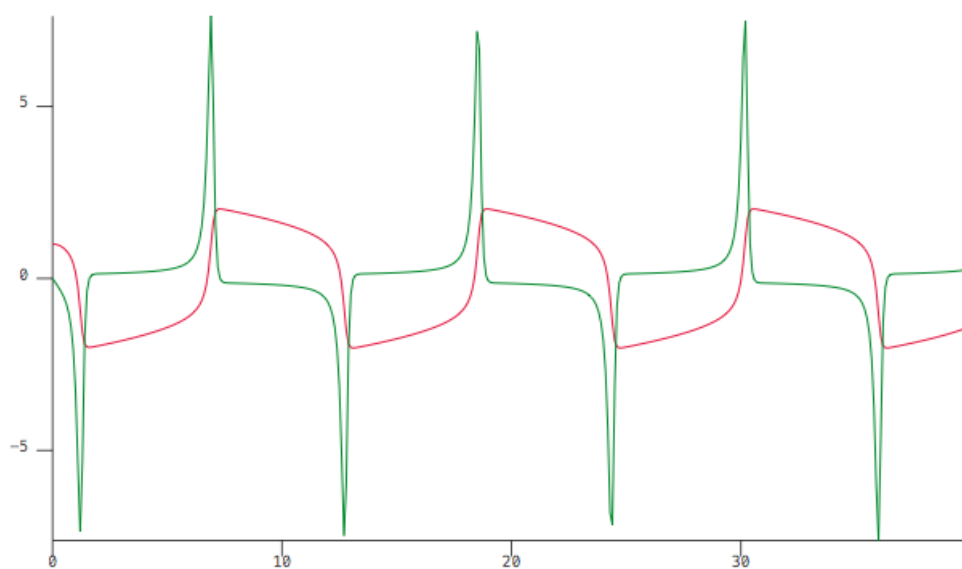
A Crash Course of Scientific Programming

Keiji Imoto

18/25-Oct-2017

revision 04-Oct-2017

revision 12-Mar-2017



Contents

I	Aim and usage of this note	3
1	Aim	3
2	Usage	3
II	Preparation	5
3	For Mac users	5
3.1	Installing tools	5
3.2	Optional tools	5
4	For Windows users	5
4.1	Installing tools	5
4.2	Optional tools	6
5	Installing examples	6
III	Fundamental programmes	7
1	ex01: A simple program: <i>hello</i>	7
2	ex02: for loop: <i>loop</i>	7
3	ex03: if-else : <i>ifelse</i>	8
4	ex04: while loop: <i>wloop</i>	8
5	ex05: Random numbers: <i>rng</i>	9
6	ex06: Plot graph: <i>plotgraph</i>	10
IV	More Practical Programmes	11
7	Solving simple ODEs: <i>ode</i>	11
7.1	ex07: A simple case (ex07)	11
7.2	ex07a: Use a class	12
7.3	ex07b: Passing command line parameters	13
7.4	Exercise: Van der Pol equation	15
8	ex08: Markov process model of the voltage-gated Na ⁺ channel: <i>channel</i>	16
9	ex09: Spiking neuron: <i>cellpot</i>	19
9.1	Burst firing	19
9.2	Exercise: A pharmacological experiment	22
10	ex10: Transmitter release: <i>release</i>	23
	Appendix	27

A	Uniform distribution random number generator	27
B	Normal distribution random number generator	27
C	Runge-Kutta method (1)	28
D	Runge-Kutta method (2)	28
E	Matlab	30
	E.1 ex01	30
	E.2 ex02	30
	E.3 ex03	30
	E.4 ex04	30
	E.5 ex05	30
	E.6 ex06	30
	E.7 ex07	31
	E.8 ex08	31
	E.9 ex09	32
	E.10 ex10	33
F	Python	36
	F.1 ex01	36
	F.2 ex02	36
	F.3 ex03	36
	F.4 ex04	36
	F.5 ex05	36
	F.6 ex06	36
	F.7 ex07	37
	F.8 ex08	37
	F.9 ex09	38
	F.10 ex10	40
G	Java	43
	G.1 ex01	43
	G.2 ex02	43
	G.3 ex03	43
	G.4 ex04	43
	G.5 ex05	43
	G.6 ex06	44
	G.7 ex07a	44
	G.8 ex08	46
	G.9 ex09	47
	G.10 ex10	50
H	Fortran	53
	H.1 ex01	53
	H.2 ex02	53
	H.3 ex03	53
	H.4 ex04	53
	H.5 ex05	53
	H.6 ex06	57
	H.7 ex07	58
	H.8 ex08	59
	H.9 ex09	60

H.10 ex10	63
I How to use C++ examples in Visual Studio IDE (Windows)	66

Cover figure: Van der Pol equation (p. 15)

Part I

Aim and usage of this note

1 Aim

This note describes simple programs (Sections 1~6) and similar but more practical programs (Sections 7~10). The main subjects are ordinary differential equations (ODEs) (Sections 7~9) and Monte Carlo simulation (Section 10), both of which are essential techniques for computer simulation.

Although it is not easy to master programming skills in a short time, this note may help you realize that you are able to do practical computer simulations, just by making a small step forward from the beginner's state.

2 Usage

In each section, source codes are ready to build/run the program from the Terminal.app (Mac) or the Visual Studio 2017 Developer Command Prompt (Windows).

¹

- For C++ programs: Mac users
 - You enter the directory, and input commands,
 - `$ make program_name` (without `'.cpp'`)
for `ex01 ~ ex04`.
 - `$ make`
where `makefile` is available (`ex05 ~ ex10`).
 - The program is executed by,
`$./program_name`
 - To build the program freshly, delete previous object files, etc, by
`$ make clean`
- For C++ programs: Windows users
 - You enter the directory, and input commands,
 - `$ cl /EHsc /Feprogram_name.exe program_name.cpp`
for `ex01 ~ ex04`.
 - `$ nmake /Fwin.mk`
where `makefile` (named `win.mk`) is available (`ex05 ~ ex10`).
 - The program is executed by,
`$ program_name`
 - To build the program freshly, delete previous object files, etc, by
`$ nmake clean`
- For Python programs

¹Of course, you can use one of IDEs (Integrated Development Environments; Xcode for Mac, Visual Studio for Windows). IDEs are quite helpful in programming. Additional procedures are required to build and run programs. If you seriously consider programming, you should set up an IDE. Please note that IDE requires a relatively high-spec hardware with a larger memory (4 ~ 8 Gbytes) for comfortable operation.

- You enter the directory, and input commands,
\$ `python program_name.py`

- For Matlab programs,

- Programs can be executed from the Matlab Command Window.

Go through the directories, and enjoy building/running the program. If you are interested, you can change parameters, or you can read the source code more carefully.

Part II

Preparation

3 For Mac users

3.1 Installing tools

1. Install C++ programming tools. Open the Terminal.app, and input,
`$ xcode-select --install`
Follow the instructions to install command line tools, which include `c++` (compiler), `ld` (linker) and `make`.
If you are serious about C++ programming, you should install the whole Xcode, although it occupies a huge disk space (~5 Gbytes).
2. Install a text editor suitable for programming.
A recommended text editor is **Microsoft Visual Studio Code**.²

3.2 Optional tools

The following items are optional for trying other programming languages.

1. Python and its libraries (`numpy`, `scipy`, `matplotlib`) are installed as default. If they are not installed yet,
`$ easy-install pip`
`$ pip install numpy`
`$ pip install scipy`
`$ pip install matplotlib`

Alternatively, you may install Anaconda Python.³ Choose version 3 unless you have a strong reason to use version 2.

2. Java SE Development Kit (JDK)
To use Java compiler, install JDK, which is available at Oracle web site.
3. Fortran compiler
Install gcc (with or without C/C++ compiler).⁴

4 For Windows users

4.1 Installing tools

1. Install **Visual Studio IDE 2017 Community** or **Build Tools for Visual Studio 2017**.⁵
The latter is in the selection of **Other Tools and Frameworks**.
2. Install a text editor suitable for programming.
A recommended text editor is **Microsoft Visual Studio Code**.⁴
Download from

²<https://code.visualstudio.com/download>

³<https://www.anaconda.com/download/#macos>

⁴<http://hpc.sourceforge.net>

⁵<https://www.visualstudio.com/downloads/>

4.2 Optional tools

1. Anaconda Python⁶.

Choose version 3 unless you have a strong reason to use version 2. This Python distribution contains `numpy`, `scipy`, and `matplotlib`.

5 Installing examples

For both of MacOS and Windows.

1. Download and unzip `TM2017.zip` somewhere, for example, in 'Desktop'

⁶<https://www.anaconda.com/download/#windows>

Part III

Fundamental programmes

1 ex01: A simple program: *hello*

★ #include
★ main
★ cout

```
1 // hello.cpp
2 #include <iostream>
3 int main(int argc, char** argv){
4     std::cout << "Hello, World!" << std::endl;
5     return 0;
6 }
```

- A simple program. It is useful to test the system for building the program.
- There is one 'main' function, and the program starts there.
- `std::cout` and `<<` are useful functions to output strings and numbers (binary data are converted to strings) to the terminal.
- Open the terminal, and go to the directory where the source file is.
if the folder of the teaching materials is in 'Desktop',
\$ cd
\$ cd Desktop/TM2017/CPP/ex01
- To build a program,
\$ clang++ -o hello hello.cpp
The option `-o` specifies the output file. If not specified, the output file is `a.out`.
- Alternatively, simply,
\$ make hello
- To run (execute) the program,
\$./hello

2 ex02: for loop: *loop*

★ for

```
1 // loop.cpp
2 #include <iostream>
3 int main(int argc, char** argv){
4     for(int i=0; i<10; i++){
5         std::cout << "i = " << i << std::endl;
6     }
7     return 0;
8 }
```

- `for` is one of the iteration (loop) statements. Other loop statements include `while` (see `ex04`) and `do~while`.

- The notation ++ means increment, and i++ is the same as i=i+1 or i+=1.
- Other C/C++-specific notations include --, --, *=, and /=.
- To go to the ex02 directory,
\$ cd ../ex02
'..' means the parent directory.
- The procedures of building and execution are the same as in ex01.

3 ex03: if-else : *ifelse*

★ if, else

```

1 // ifelse.cpp
2 #include <iostream>
3 int main(int argc, char** argv){
4     for(int i=0;i<10;i++){
5         if(i%2==0){
6             std::cout << "even: " << i << std::endl;
7         }else{
8             std::cout << "odd: " << i << std::endl;
9         }
10    }
11    return 0;
12 }
```

- if, if ~ else, if ~ if else ~ else are commonly used for flow control.
- if statements can be nested.
- == is a comparison operator to check the equality. Other comparison operators are != (not equal), >, <, >=, and <=.
- The notation % is the modulo operator. It gives the remainder of a division of two integer values.

4 ex04: while loop: *wloop*

★ while

★ break

```

1 // while.cpp
2 #include <iostream>
3 int main(int argc, char** argv){
4     int i=1;
5     while(true){
6         if(i%17==0 && (i+2)%11==0 && (i+4)%23==0){
7             std::cout << "i = " << i << std::endl;
8             break;
9         }
10        i++;
11    }
12    return 0;
13 }
```

- Logical expressions can be combined using logical operators && (AND) and || (OR).
- ! is a NOT operator.
- while(true){ ... } is an infinite loop. break jumps out of the loop.

clearpage

5 ex05: Random numbers: *rng*

- ★ multiple files
- ★ make
- ★ class
- ★ Mersenne-Twistter method

```
1 // main.cpp of rng
2 #include <iostream>
3 #include "UniRand.h"
4 int main(int argc, char** argv){
5     UniRand urng;
6     for(int i=0;i<10;i++){
7         std::cout << i << ": " << urng.rand() << std::endl;
8     }
9     return 0;
10 }
```

```
1 # makefile for rng
2 cxx      = clang++
3 cflags   = -O3
4 ldflags  =
5 exec     = rng
6 objs     = UniRand.o main.o
7
8 all: $(exec)
9
10 $(exec): $(objs)
11         $(cxx) $(ldflags) $(objs) -o $@
12
13 %.o: %.cpp
14         $(cxx) -c $(cflags) $<
15
16 clean:
17         -rm $(exec) $(objs)
```

- Usually, groups of functions are defined in separate files.
- There are two kinds of files, a source code file (**.cpp**) and a header file (**.h**), in case of C/C++ language. The header file contains information required for compiling related files.
- Each source file is compiled separately to yield an object file.
- Object files are linked to make an executable program file.
- UniRand is a wrapper class for uniform distribution random number generator, which is a built-in function of recent C/C++ compilers.
- To build a program,
\$ clang++ -c main.cpp
\$ clang++ -c UniRand.cpp
\$ clang++ main.o UniRand.o -o rng
(Commands and options are different for Windows.)
- Because this is not an efficient way when the number of files are large, the utility program **make** is often used.

- `make` checks the dependency among the files, and compile the files that require re-compiling. `make` also handles linking of object files and other procedures. Dependency, other relations and actions are described in the `makefile`. If its name is 'makefile', just
\$ `make` (\$ `nmake` for Windows)

If the file name is something else,

\$ `make -f filename` (\$ `nmake /F filename` for Windows)

- `makefile` is usually stereotyped. You have to change few words to use a `makefile` for other programs.
- `UniRand.cpp` (Appendix A) uses the Mersenne Twister method. It was developed by Drs. Makoto Matsumoto and Takuji Nishimura of Hiroshima University, Japan. It is not only small and very efficient but also is theoretically proven for its randomness.
- A class instance is made from a class (template) when it is declared.
`UniRand urng;`
- Normal distribution random number generator is provided in Appendix B.

6 ex06: Plot graph: *plotgraph*

★ file input/output (i/o)

```

1 // main.cpp of plotgraph
2 #include <iostream>
3 #include <fstream>
4 #include "UniRand.h"
5 int main(int argc, char** argv){
6     double dt = 0.1;
7     double ttime = 10.0;
8     int npts = int(ttime/dt) + 1;
9     UniRand urng;
10    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
11    for(int i=0; i<npts; i++){
12        double a = urng.rand();
13        fd << dt*i << " " << a << std::endl;
14    }
15    fd.close();
16    #if defined(_WIN32)
17        system("../tools\\gc ax.txt ");
18    #else
19        system("../tools/gc ax.txt ");
20    #endif
21    return 0;
22 }
```

- Text output to files is similar to terminal output.
- `std::fstream` is the common way to open a file.
- The file must be closed explicitly.
- `system` handles command line inputs.
- `gc` is a in-house utility program to draw a graph.

Part IV

More Practical Programmes

7 Solving simple ODEs: *ode*

7.1 ex07: A simple case (ex07)

★ functions, subroutines

★ vector

Solving differential equations is one of the important fields of computational science. Many kinds of phenomenon are described using differential equations, but usually they cannot be solved analytically. Then, the equations are computed numerically (numerical integration) to obtain results.

As a simple example, we solve the simplest second order ordinary differential equation (of course, this equation can be solved analytically),

$$\frac{d^2y}{dt^2} = -y$$

This second order ODE is described by a pair of first order equations, if we define a new variable z , which we equate to dy/dt .

$$\begin{aligned}\frac{dy}{dt} &= z \\ \frac{d^2y}{dt^2} &= \frac{dz}{dt} = -y\end{aligned}$$

If y and z are components of an array $y[]$, $y[0]$ and $y[1]$, respectively, the equations are written,

$$\begin{aligned}\frac{dy[0]}{dt} &= y[1] \\ \frac{dy[1]}{dt} &= -y[0]\end{aligned}$$

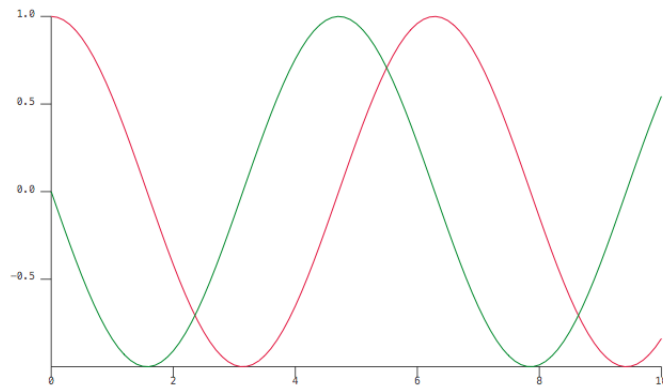
```
1 // main.cpp of ode (ex07)
2 #include <iostream>
3 #include <fstream>
4
5 #include "RungeKutta4.h"
6
7 void rhs(double tt, vector<double> y, vector<double>& dy){
8     dy[0] = y[1];
9     dy[1] = -y[0];
10 }
11
12 int main(int argc, char** argv){
13     int ndim = 2; // number of equations
14     double ttime = 10.0; // total time
15     double dt = 0.1; // step size
16     int npts = (int)(ttime/dt)+1; // number of points
17     vector<double> y={1.0, 0.0}; // initial values
18
19     RungeKutta4 rk(ndim, dt);
20
21     std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
```

```

22     for(int i=0;i<npts;i++){
23         double t = dt*i;
24         fd << t << " " << y[0] << " " << y[1] << std::endl;
25         rk.advanceStep(rhs, t, y);
26     }
27     fd.close();
28 #if defined(_WIN32)
29     system("../tools\\gc ax.txt ");
30 #else
31     system("../tools/gc ax.txt ");
32 #endif
33     return 0;
34 }

```

- The differential equations are defined in `main.cpp` as a function `rhs`.
- For numerical integration, the fourth-order Runge-Kutta method is used. This method is portable, and is very efficient and accurate, although it may fail for 'stiff' problems.
- `vector` is a class to handle arrays (including multi-dimensional arrays) more elegantly. A merit to use `vector` class is that it can be dynamically allocated and does not require deallocation procedure.



7.2 ex07a: Use a class

It is more convenient to describe ODEs as a class member function.

```

1 // main.cpp of ode (ex07a)
2 #include <iostream>
3 #include <fstream>
4 #include "Eqs.h"
5
6 int main(int argc, char** argv){
7     int ndim = 2;          // number of equations
8     double ttime = 10.0;   // total time
9     double dt = 0.1;       // step size
10    int npts = (int)(ttime/dt)+1;
11    vector<double> y={1.0, 0.0};
12
13    Eqs ce;
14    ce.initRK4(ndim, dt);
15
16    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
17    for(int i=0;i<npts;i++){
18        double t = dt*i;
19        fd << t << " " << y[0] << " " << y[1] << std::endl;
20        ce.advanceStep(t, y);
21    }

```

```

22     fd.close();
23 #if defined(_WIN32)
24     system("../..\\tools\\gc ax.txt ");
25 #else
26     system("../tools/gc ax.txt ");
27 #endif
28     return 0;
29 }

```

```

1 // Eqs.h
2 #ifndef __included_eqs_h__
3 #define __included_eqs_h__
4
5 #include <vector>
6 #include "RungeKutta4.h"
7
8 using std::vector;
9
10 class Eqs : public RungeKutta4
11 {
12 public:
13     void rhs(double tt, vector<double> y, vector<double>& dy);
14 private:
15 };
16 #endif

```

```

1 // Eqs.cpp
2 #include "Eqs.h"
3 void Eqs::rhs(double tt, vector<double> y, vector<double>& dy){
4     dy[0] = y[1];
5     dy[1] = -y[0];
6 }

```

```

1 # makefile for ode (ex07f)
2 cxx      = clang++
3 cflags   = -std=c++0x -O3
4 ldflags  =
5 exec     = ode
6 objs     = RungeKutta4.o Eqs.o main.o
7
8 all: $(exec)
9 $(exec): $(objs)
10         $(cxx) $(ldflags) $(objs) -o $@
11 %.o: %.cpp
12         $(cxx) -c $(cflags) $<
13 clean:
14         -rm $(exec) $(objs)

```

Class Eqs is defined as a subclass of RungeKutta4.

7.3 ex07b: Passing command line parameters

```

1 // main.cpp of ode (ex07b)
2 #include <iostream>
3 #include <fstream>
4 #include <string>
5 #include "Eqs.h"
6
7 int main(int argc, char** argv){
8     int ndim = 2;           // number of equations
9     double ttime = 100.0;   // total time
10    double dt = 0.1;         // step size
11    int npts = (int)(ttime/dt)+1;
12    vector<double> y={1.0, 0.0};
13    double a = 1.0;          // default parameter
14    double b = 0.1;          // default parameter
15    if(argc >= 3){

```

```

16     a = std::stof(std::string(argv[1]));
17     b = std::stof(std::string(argv[2]));
18     if(argc == 5){
19         y[0] = std::stof(std::string(argv[3]));
20         y[1] = std::stof(std::string(argv[4]));
21     }
22 }
23 Eqs ce;
24 ce.setParams(a,b);
25 ce.initRK4(ndim, dt);
26
27 std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
28 for(int i=0;i<npts;i++){
29     double t = dt*i;
30     fd << t << " " << y[0] << " " << y[1] << std::endl;
31     ce.advanceStep(t, y);
32 }
33 fd.close();
34 #if defined(_WIN32)
35 system("../tools\\gc ax.txt ");
36 #else
37 system("../tools/gc ax.txt ");
38 #endif
39 return 0;
40 }

```

```

1 // Eqs.h
2 #ifndef __included_eqs_h__
3 #define __included_eqs_h__
4
5 #include <vector>
6 #include "RungeKutta4.h"
7
8 using std::vector;
9
10 class Eqs : public RungeKutta4 {
11 public:
12     void setParams(double a, double b);
13     void rhs(double tt, vector<double> y, vector<double>& dy);
14 private:
15     double f;
16     double d;
17 };
18 #endif

```

```

1 // Eqs.cpp
2 #include "Eqs.h"
3 void Eqs::setParams(double a, double b){
4     f = a;
5     d = b;
6 }
7 void Eqs::rhs(double tt, vector<double> y, vector<double>& dy){
8     dy[0] = y[1];
9     dy[1] = -f*y[0] - d*y[1];
10 }

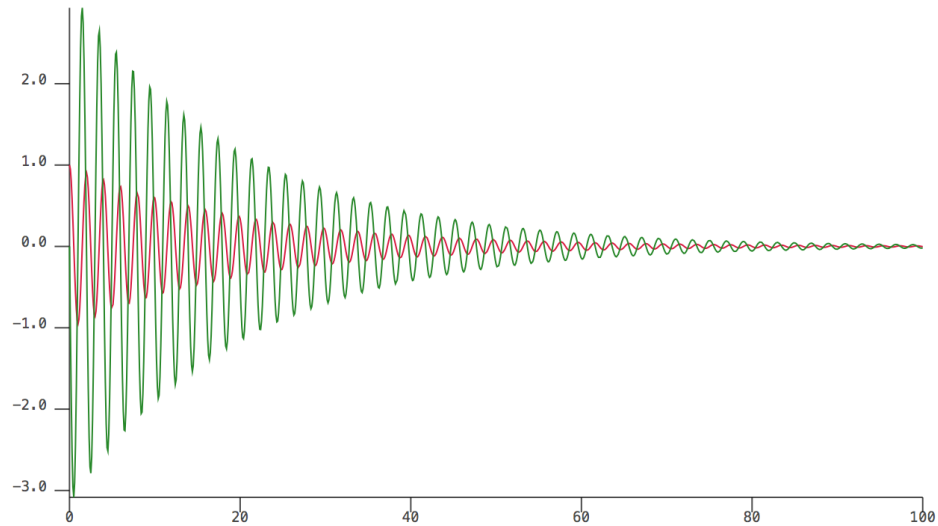
```

The equation is changed. There are two constant variables f and d .

$$\frac{d^2y}{dt^2} = -f \cdot y - d \cdot \frac{dy}{dt}$$

- It is not so convenient to compile and build the program every time when the parameters are changed.
- Parameters can be passed to the program from the command line.
- `argc` is the number of arguments, and `argv` is an array of character array. `argv[0]` is the path of the program.

- Because the input in the command line is a series of characters, it has to be converted into a string (`string`), then into a double number (`stof`).
- In case of an integer, use `stoi` function.
- The initial values of `y` are also specified from the command line.



7.4 Exercise: Van der Pol equation

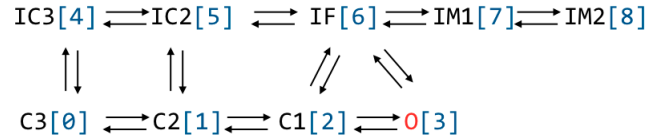
Calculate the Van der Pol oscillator. Try $\mu = 5$.

$$\frac{d^2 y}{dt^2} - \mu(1 - y^2) \frac{dy}{dy} + y = 0$$

8 ex08: Markov process model of the voltage-gated Na⁺ channel: *channel*

Gating behavior of the voltage-gated Na⁺ channel is well described by Markov process model. In this case, nine states are assumed, and the transition rates between states are voltage dependent. If the channel is voltage-clamped, the transition rates are constant.

Na⁺ channel Markov process model



```

1 // main.cpp of channel
2 #include <iostream>
3 #include <fstream>
4 #include <cmath>
5 #include "Markov.h"
6
7 int main(){
8     const int ndim = NUM_OF_NA_STATES;
9     const double dt = 0.01;
10    const double totalTime = 50.0;
11    const int npts = (int)(totalTime/dt)+1;
12
13    std::vector<double> y(ndim);
14
15    Markov ce;
16    ce.initRK4(ndim, dt);
17    ce.setInitialValues(y);
18
19    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
20    for(int i=0; i<npts; i++){
21        double t = dt*i;
22        fd << t << " " << y[0] << " " << y[3] << " " << y[6] << " "
23           << y[7] << " " << y[8] << std::endl;
24        ce.advanceStep(t, y);
25    }
26    fd.close();
27    #if defined(_WIN32)
28    system("../..\\tools\\gc ax.txt &");
29    #else
30    system("../tools/gc ax.txt &");
31    #endif
32    return 0;
33 }

```

```

1 // markov.h
2 #ifndef __included__markov_h__
3 #define __included__markov_h__
4
5 #include <vector>
6 #include "RungeKutta4.h"
7
8 #define NUM_OF_NA_STATES 9
9 #define NA_OPEN_STATE 3
10
11 using std::vector;
12
13 class Markov : public RungeKutta4 {
14 public:
15     void setInitialValues(vector<double>& y);

```

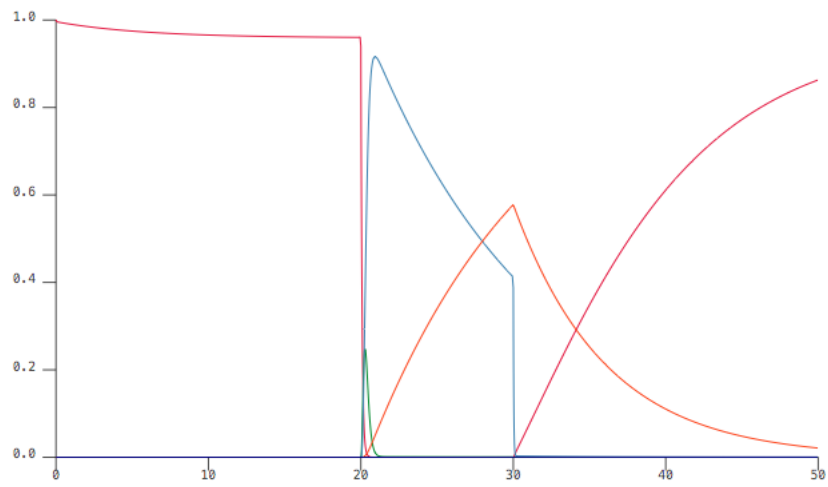
```

16     void rhs(double t, vector<double> y, vector<double>& dy);
17 private:
18     double a11, a12, a13, a2, a3, a4, a5;
19     double b11, b12, b13, b2, b3, b4, b5;
20     double vold;
21 };
22 #endif

1 // markov.cpp
2 #include <cmath>
3 #include "Markov.h"
4
5 void Markov::setInitialValues(vector<double>& y){
6     y[0] = 1.0;
7     for(int i = 1 ; i < NUM_OF_NA_STATES ; i++){
8         y[i] = 0.0;
9     }
10    vold = -1000.0;
11 };
12 void Markov::rhs(double t, vector<double> y, vector<double>& dy){
13     double v = t<20.0 || t>= 30.0 ? -100.0 : 0.0;
14     if(v != vold){
15         a11 = 3.802/(0.1027*exp(-v/17.0) + 0.20*exp(-v/150));
16         a12 = 3.802/(0.1027*exp(-v/15.0) + 0.23*exp(-v/150));
17         a13 = 3.802/(0.1027*exp(-v/12.0) + 0.25*exp(-v/150));
18
19         b11 = 0.1917*exp(-v/20.3);
20         b12 = 0.20*exp(-(v-5)/20.3);
21         b13 = 0.22*exp(-(v-10)/20.3);
22
23         a3 = (3.7933e-7)*exp(-v/7.7);
24         b3 = 0.0084 + 0.00002*v;
25
26         a2 = 9.178*exp(v/29.68);
27         b2 = (a13*a2*a3)/(b13*b3);
28
29         a4 = a2/100;
30         b4 = a3;
31
32         a5 = a2/(9.5e4);
33         b5 = a3/50;
34         vold = v;
35     }
36     dy[0] = -(a11 + b3)*y[0] + b11*y[1] + a3*y[4];
37     dy[1] = -(b11 + b3 + a12)*y[1] + a11*y[0] + b12*y[2] + a3*y[5];
38     dy[2] = -(b12 + b3 + a13)*y[2] + a12*y[1] + b13*y[3] + a3*y[6];
39     dy[3] = -(b13 + a2)*y[3] + a13*y[2] + b2*y[6];
40     dy[4] = -(a11 + a3)*y[4] + b11*y[5] + b3*y[0];
41     dy[5] = -(b11 + a3 + a12)*y[5] + a11*y[4] + b12*y[6] + b3*y[1];
42     dy[6] = -(b12 + a3 + b2 + a4)*y[6] + a12*y[5] + b3*y[2] + a2*y[3] + b4*y[7];
43     dy[7] = -(b4 + a5)*y[7] + a4*y[6] + b5*y[8];
44     dy[8] = -b5*y[8] + a5*y[7];
45 }

```

- This program solves ODEs, and has the same program structure as *channel* in ex07b.
- RungeKutta4.h and RungeKutta4.cpp are the same as in ex07b.
- rhs function is described in separate files (.h and .cpp files).



Time courses of C3 (red), O (green), IF (blue), and IM1 (orange).
IM2 is barely visible (almost zero).

9 ex09: Spiking neuron: *cellpot*

9.1 Burst firing

```
1 // main.cpp of cellpot
2 #include <iostream>
3 #include <fstream>
4 #include "Cell.h"
5
6 int main(){
7     const int m = 6; // number of equations
8     const double dt = 0.05; // step size
9     const double totalTime = 600.0; // simulation time
10    const int npts = (int)(totalTime/dt)+1;
11    vector<double> y(m);
12    vector<double> vol(npts);
13
14    double vinit = -63.27; // initial voltate
15    double Iinj = -2000.0; // current injection(pA)
16    double Ton = 50.0; // on-time
17    double Toff = 300.0; // off-time
18
19    Cell ce;
20    ce.initRK4(m, dt);
21    ce.setInitialValues(vinit, y);
22    ce.setPulse(Ton, Toff, Iinj);
23
24    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
25    for(int i=0; i<npts; i++){
26        double t = dt*i;
27        vol[i] = y[0];
28        fd << t << " " << y[0] << std::endl;
29        ce.advanceStep(t, y);
30    }
31    // count the number of spikes
32    int sn=0;
33    double th = -20.0;
34    for(int i=0; i<npts-1; i++){
35        if(vol[i+1]>=th && vol[i] < th) sn++;
36    }
37    std::cout << "number of spikes = " << sn << std::endl;
38    fd.close();
39    #if defined(_WIN32)
40        system("../tools\\gc ax.txt &");
41    #else
42        system("../tools/gc ax.txt &");
43    #endif
44    return 0;
45 }
```

```
1 // cell.h
2 #ifndef __ode__Cell__
3 #define __ode__Cell__
4
5 #include <vector>
6 #include "RungeKutta4.h"
7
8 using std::vector;
9 class Cell : public RungeKutta4
10 {
11 public:
12     Cell(){};
13     ~Cell(){};
14     void setInitialValues(double v_init, vector<double>& y);
15     void setPulse(double ton, double toff, double current);
16     void rhs(double t, vector<double> y, vector<double>& yt);
17 private:
18     static constexpr double Dm = 60.0; // (micrometer) diameter of the cell
19     static constexpr double Lm = 70.0; // (micrometer) length of the cell
```

```

20
21     static constexpr double Eleak = -65.0;    // (mV)
22     static constexpr double ENa = 50.0;      // (mV)
23     static constexpr double EK = -100.0;     // (mV)
24     static constexpr double ECa = 50.0;      // (mV)
25
26     static constexpr double dGleak = 0.002; // (mS/cm^2)
27     static constexpr double dGNa  = 0.05;  //0.53;   // (mS/cm^2)
28     static constexpr double dGK   = 0.15;  // 0.13;   // (mS/cm^2)
29     static constexpr double dGTCa = 0.03;   // (mS/cm^2)
30
31     static constexpr double vtraub = -55.0;  // -55.0 mV
32     static constexpr double fac = 10;       // conversion factor (mS/cm^2) -> (pS/micrometer^2)
33
34     int ndim;
35     // double v,vinit;
36     double Iinj;    // 2000.0 (pA)
37     double Ton;     // 20 ms
38     double Toff;    // 40 ms
39
40     double Am;      // surface area
41     double Cm;
42     double InvCm;   // 1/Cm;
43
44     double Gleak;
45     double GNa;
46     double GK;
47     double GTCa;
48
49 private:
50     void get_Na_m(double v, double* a, double* b);
51     void get_Na_h(double v, double* a, double* b);
52     void get_K_n(double v, double* a, double* b);
53     void get_TCa_m(double v, double* tau, double* inf);
54     void get_TCa_h(double v, double* tau, double* inf);
55 };
56
57 #endif /* defined(__ode__Cell__) */

```

```

1 // cell.cpp
2 #include <iostream>
3 #include <cmath>
4 #include "Cell.h"
5
6 void Cell::setInitialValues(double v_init, std::vector<double>& y){
7     Am = 3.14 * Dm * Lm;    // (square_micrometer) surface area
8     Cm = 0.01 * Am;
9     InvCm = 1.0/Cm;
10
11     Gleak = Am * dGleak * fac;
12     GNa  = Am * dGNa * fac;
13     GK   = Am * dGK * fac;
14     GTCa = Am * dGTCa * fac;
15
16     double v = y[0] = v_init;
17     double a, b, tau, inf;
18     get_Na_m(v, &a, &b);
19     y[1] = a/(a+b);
20
21     get_Na_h(v, &a, &b);
22     y[2] = a/(a+b);
23
24     get_K_n(v, &a, &b);
25     y[3] = a/(a+b);
26
27     get_TCa_m(v, &tau, &inf);
28     y[4] = inf;
29
30     get_TCa_h(v, &tau, &inf);
31     y[5] = inf;

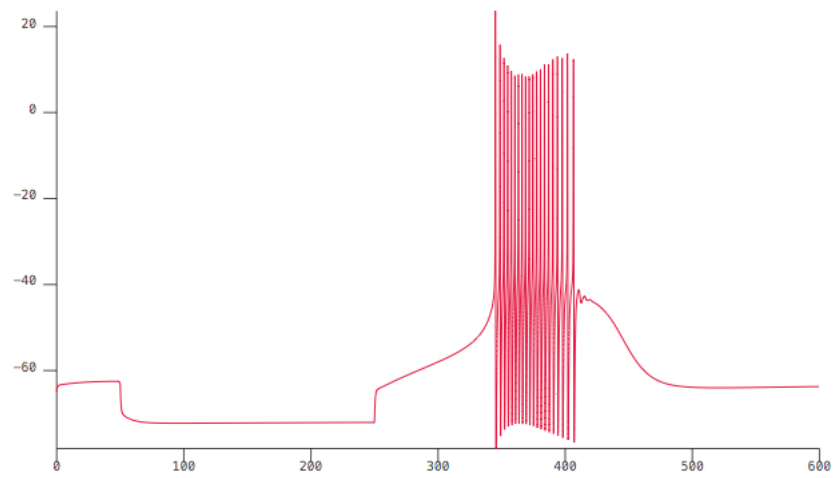
```

```

32 }
33 void Cell::setPulse(double ton, double toff, double current){
34     Ton = ton;
35     Toff = toff;
36     Iinj = current;
37 }
38 void Cell::get_Na_m(double v, double* a, double* b){
39     double v2 = v-vtraub;
40     *a = 0.32*(13.0-v2)/(exp((13.0-v2)/4.0)-1.0);
41     *b = 0.28*(v2-40.0)/(exp((v2-40.0)/5.0)-1.0);
42 }
43 void Cell::get_Na_h(double v, double* a, double* b){
44     double v2 = v-vtraub;
45     *a = 0.128 * exp((17.0-v2)/18.0);
46     *b = 4.0/(1.0 + exp((40.0-v2)/5.0));
47 }
48 void Cell::get_K_n(double v, double* a, double* b){
49     double v2 = v - vtraub;
50     *a = 0.032*(15.0-v2)/(exp((15.0-v2)/5.0)-1.0);
51     *b = 0.5*exp((10.0-v2)/40.0);
52 }
53 void Cell::get_TCa_m(double v, double* tau, double* inf){
54     double vshift=0.0;
55     double vs = v + vshift;
56     *tau = (3.0+1.0/(exp((vs+25.0)/10.0)+exp(-(vs+100.0)/15.0)));
57     *inf = 1.0/(1.0+exp(-(vs+50.0)/7.4));
58 }
59 void Cell::get_TCa_h(double v, double* tau, double* inf){
60     double vshift=0.0;
61     double vs = v + vshift;
62     *tau = (85.0+1.0/(exp((vs+46.0)/4.0)+exp(-(vs+405.0)/50.0))) ;
63     *inf = 1.0/(1.0+exp((vs+78.0)/5.0));
64 }
65
66 void Cell::rhs(double tt, std::vector<double> y, std::vector<double>& dy){
67     double INa, IK, ITCa;
68     double a, b, tau, inf;
69     double shift = 0.0;
70     double v = y[0];
71     double c = (tt<Ton || tt>Toff)? 0.0 : Iinj;
72
73     INa = GNa * pow(y[1],3) * y[2] * (v-ENa); // INa = GNa * m^3 * h * (v-Ena)
74     IK = GK * pow(y[3],4) * (v-EK); // IK = GK * n^4 (v-EK)
75     ITCa = GTCa * pow(y[4],2) * y[5] * (v - ECa);
76     dy[0] = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa);
77     // Na channel
78     get_Na_m(v, &a, &b);
79     dy[1] = a - (a+b)*y[1];
80     get_Na_h(v, &a, &b);
81     dy[2] = a - (a+b)*y[2];
82     // K channel (delayed rectifier)
83     get_K_n(v, &a, &b);
84     dy[3] = a - (a+b)*y[3];
85     // T-type Ca channel
86     get_TCa_m(v, &tau, &inf);
87     dy[4] = (inf - y[4]) / tau;
88     get_TCa_h(v, &tau, &inf);
89     dy[5] = (inf - y[5]) / tau;
90 }

```

- This program may appear complicated, but structurally it is almost the same as **ex07a** and **ex08**.
- This is a simple neuron model, which includes voltage-gated sodium channel, voltage-gated potassium channel, voltage-gated calcium channel (T type), and leak channel. Behaviour of voltage-gated channels and cell potential are described by ODEs.
- Termination of negative current injection evokes burst firing (rebound bursting).

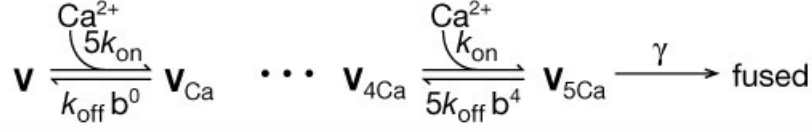


9.2 Exercise: A pharmacological experiment

Block Na^+ channel, and see the response to current injection.

10 ex10: Transmitter release: *release*

Neurotransmitter release is a stochastic process, and the process is dependent on Ca^{2+} concentration. Several models have been proposed. Here, we use the model of Ralf Schneggenburger & Erwin Neher ⁷, and try to simulate the release according to the model.

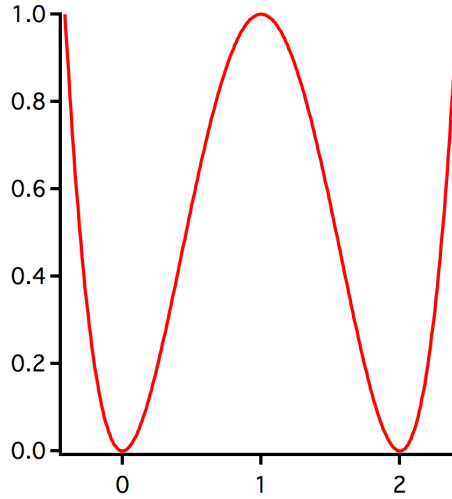


The following parameters were used: k_{on} , $9 \times 10^7 \text{ M}^{-1} \text{ s}^{-1}$;
 k_{off} , $9,500 \text{ s}^{-1}$; γ , $6,000 \text{ s}^{-1}$; $b = 0.25$.

Since the units which can be used effortlessly are ms (millisecond) and μM (micro molar), the parameters described above are coded,

```
1 double kon = 9.0e-2; //  $\mu\text{M}^{-1} \text{ ms}^{-1}$ 
2 double koff = 9.5; //  $\text{ms}^{-1}$ 
3 double beta = 0.25; // cooperativity
4 double gamma = 6.0; // 6.0d0 !  $\text{ms}^{-1}$ 
5 double epsilon = 0.3; //  $\text{ms}^{-1}$ , recovery rate from released stated
```

There is no experimental data of the temporal change of Ca^{2+} , a bell-shaped function is assumed. From a technical point of view, the onset and offset of calcium spike are assumed to be smooth and not sudden. A function $y = t^2(t-2)^2$ is used. This equation can be written as $y = ((t-1)^2 - 1)^2$



A scaling factor d is determined so that the half-width of the transient is equal to hw .

$$\begin{aligned} y(t) &= \text{peak} \times ((d \cdot t - 1)^2 - 1)^2 \\ d &= 2 \times (1 - (0.5)^{1/2})^{1/2} / hw \\ 0 &\leq t \leq 2.0/d \end{aligned}$$

The peak Ca^{2+} concentration, *peak*, is $20 \mu\text{M}$. The half width, *hw*, is 1 ms.

⁷Intracellular calcium dependence of transmitter release rates at a fast central synapse. Nature 406: 889-893, 2000.

There are 7 states. At each state, values of a random number determines stay, or forward or backward transitions.

```

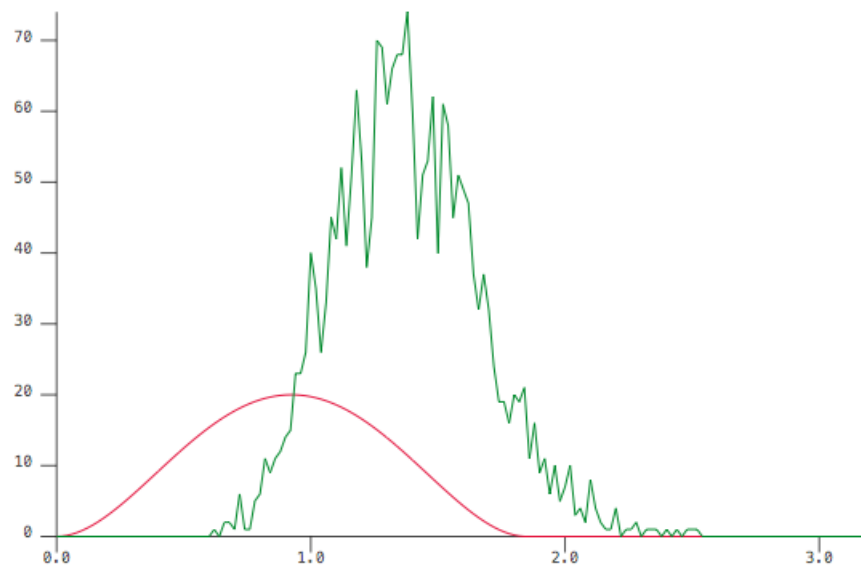
1 // release.cpp
2 // Stochastic neurotransmitter release
3 // Ralf Schneggenburger & Erwin Neher
4 // Nature 406, 889-893, 2000.
5 #include <iostream>
6 #include <fstream>
7 #include <vector>
8 #include <cmath>
9 #include "UniRand.h"
10 int main(int argc, const char * argv[]){
11     const double ttime = 4.0;
12     const double dt=0.02;
13     const double peak = 20.0;
14     const double hw = 1.0;
15     const int iter = 100000;
16
17     const double kon = 9.0e-2; // microM-1 ms-1
18     const double koff = 9.5; // ms-1
19     const double beta = 0.25; // cooperativity
20     const double gamma = 6.0; // 6.0d0 ! ms-1
21     const double epsilon = 0.3; // ms-1
22     const int initialState = 1;
23
24     std::cout << iter << " iterations" << std::endl;
25
26     int npts = (int)(ttime/dt);
27     int counter = 0;
28     std::vector<double> ca(npts, 0.0);
29     std::vector<int> sum(npts,0);
30
31     UniRand urng; // Uniform random number generator
32
33     // ca transient
34     double d = 2.0*sqrt((1.0 - sqrt(0.5)))/hw;
35     for(int i=0;i<(int)(2.0/(dt*d));i++){
36         ca[i] = peak * pow(pow(d*dt*i - 1.0, 2.0)-1.0, 2.0);
37     }
38
39     double konh = dt * kon;
40     double koffh = dt * koff;
41     double gammah = dt * gamma;
42     double epsilonh = dt*epsilon;
43     double beta2 = beta*beta;
44     double beta3 = beta*beta2;
45     double beta4 = beta*beta3;
46
47     for(int k=0;k<iter;k++){ // iteration loop
48         int s = initialState; //initial state
49         for(int i=0;i<npts;i++){
50             double th;
51             double r = urng.rand(); // get a random number [0, 1]
52             double alpha = konh * ca[i]; // get Ca level
53             int snext = s;
54             switch (s){
55                 case (1):
56                     th = 5.0 * alpha;
57                     if(r<th) snext = 2;
58                     break;
59                 case (2):
60                     th = 4.0 * alpha;
61                     if(r<th) {
62                         snext = 3;
63                     }else{
64                         th = th + koffh;
65                         if(r<th) snext = 1;
66                     }
67                     break;
68                 case (3):

```

```

69         th = 3.0 * alpha;
70         if(r<th) {
71             snext = 4;
72         }else{
73             th = th + koffh * beta;
74             if(r<th) snext = 2;
75         }
76         break;
77     case (4):
78         th = 2.0 * alpha;
79         if(r<th){
80             snext = 5;
81         }else{
82             th = th + koffh * beta2;
83             if(r<th) snext = 3;
84         }
85         break;
86     case (5):
87         th = alpha;
88         if(r<th){
89             snext = 6;
90         }else{
91             th = th + koffh * beta3;
92             if(r<th) snext = 4 ;
93         }
94         break;
95     case (6):
96         th = gammah;
97         if(r<th){
98             snext = 7;
99         }else{
100             th = th + koffh * beta4;
101             if(r<th) snext = 5;
102         }
103         break;
104     case (7):
105         snext = 0; //release
106         sum[i] += 1; // release
107         counter++;
108         break;
109     case (0): //recovery from release
110         th = epsilonh;
111         if(r<th) snext = 1;
112         break;
113     default:
114         ;
115     } // end of switch
116     s = snext;
117 } //end of i-loop
118 } // end of k-loop
119
120 std::cout << "release probability = " << double(counter)/(double)iter << std::endl;;
121 std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
122 for(int i=0;i<npts;i++){
123     fd << dt*i << " " << ca[i] << " " << (double)sum[i] << std::endl;
124 }
125 fd.close();
126 #if defined(_WIN32)
127     system("../..\\tools\\gc ax.txt &");
128 #else
129     system("../tools/gc ax.txt &");
130 #endif
131     return 0;
132 }

```



A Uniform distribution random number generator

```
1 // UniRand.h
2 #ifndef INCLUDED_UNIRAND_H_
3 #define INCLUDED_UNIRAND_H_
4 #include <random>
5 class UniRand
6 {
7 public:
8     UniRand(); // constructor
9     UniRand(double a, double b); // constructor
10    ~UniRand(){}; // destructor
11    double rand();
12 private:
13     std::mt19937 engine; // Mersenne Twister
14     std::uniform_real_distribution<double> distribution;
15 };
16 #endif

1 // UniRand.cpp
2 #include "UniRand.h"
3 UniRand::UniRand():distribution(0.0, 1.0){
4     std::random_device seed;
5     engine.seed(seed());
6 }
7 UniRand::UniRand(double a, double b):distribution(a, b){
8     std::random_device seed;
9     engine.seed(seed());
10 }
11 double UniRand::rand(){
12     return distribution(engine);
13 }
```

B Normal distribution random number generator

```
1 // NormalRand.h
2 #ifndef INCLUDED_NormalRandRAND_H_
3 #define INCLUDED_NormalRandRAND_H_
4 #include <random>
5 class NormalRand
6 {
7 public:
8     NormalRand(); // constructor
9     NormalRand(double mu, double sigma); // constructor
10    ~NormalRand(){}; // destructor
11    double rand();
12 private:
13     std::mt19937 engine; // Mersenne Twister
14     std::normal_distribution<double> distribution;
15 };
16 #endif

1 // NormalRand.cpp
2 #include "NormalRand.h"
3 NormalRand::NormalRand():distribution(0.0, 1.0){
4     std::random_device seed;
5     engine.seed(seed());
6 }
7 NormalRand::NormalRand(double mu, double sigma):distribution(mu, sigma){
8     std::random_device seed;
9     engine.seed(seed());
10 }
11 double NormalRand::rand(){
12     return distribution(engine);
13 }
```

C Runge-Kutta method (1)

```
1 // RungeKutta4.h
2
3 #ifndef __include_rungekutta4_h__
4 #define __include_rungekutta4_h__
5
6
7 #include <vector>
8 using std::vector;
9
10 typedef void (*RHSFunc)(double tt, vector<double> y, vector<double>& dy);
11
12
13 class RungeKutta4 {
14 public:
15     RungeKutta4(int n, double dt);
16     void advanceStep(RHSFunc rhs, double t, vector<double>& y);
17 private:
18     int n;
19     double h;
20     vector<double> yn;
21     vector<double> k1;
22     vector<double> k2;
23     vector<double> k3;
24     vector<double> k4;
25 };
26 #endif
```

```
1 // RungeKutta4.cpp
2 #include "RungeKutta4.h"
3 RungeKutta4::RungeKutta4(int ndim, double dt){
4     n = ndim;
5     h = dt;
6     yn = vector<double>(n);
7     k1 = vector<double>(n);
8     k2 = vector<double>(n);
9     k3 = vector<double>(n);
10    k4 = vector<double>(n);
11 }
12 void RungeKutta4::advanceStep(RHSFunc rhs, double t, vector<double>& y){
13     rhs(t, y, k1);
14     for(int i=0;i<n;i++) yn[i] = y[i] + h*k1[i]/2;
15     rhs(t+h/2, yn, k2);
16     for(int i=0;i<n;i++) yn[i] = y[i] + h*k2[i]/2;
17     rhs(t+h/2, yn, k3);
18     for(int i=0;i<n;i++) yn[i] = y[i] + h*k3[i];
19     rhs(t+h, yn, k4);
20     for(int i=0;i<n;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
21 }
```

D Runge-Kutta method (2)

```
1 // RungeKutta4.
2 #ifndef __include_rungekutta4_h__
3 #define __include_rungekutta4_h__
4
5 #include <vector>
6
7 using std::vector;
8
9 class RungeKutta4{
10 public:
11     void initRK4(int n, double dt);
12     void advanceStep(double t, vector<double>& y);
13     virtual void rhs(double tt, vector<double> y, vector<double>& dy){};
14 }
```

```

14 private:
15     int n;
16     double h;
17     vector<double> yn;
18     vector<double> k1;
19     vector<double> k2;
20     vector<double> k3;
21     vector<double> k4;
22 };
23 #endif

```

```

1 // RungeKutta4.cpp
2 #include "RungeKutta4.h"
3
4 void RungeKutta4::initRK4(int ndim, double dt){
5     n = ndim;
6     h = dt;
7     yn = vector<double>(n);
8     k1 = vector<double>(n);
9     k2 = vector<double>(n);
10    k3 = vector<double>(n);
11    k4 = vector<double>(n);
12 }
13
14 void RungeKutta4::advanceStep(double t, vector<double>& y){
15     rhs(t, y, k1);
16     for(int j=0; j<n; j++) yn[j] = y[j] + h*k1[j]/2.0;
17     rhs(t+h/2.0, yn, k2);
18     for(int j=0; j<n; j++) yn[j] = y[j] + h*k2[j]/2.0;
19     rhs(t+h/2.0, yn, k3);
20     for(int j=0; j<n; j++) yn[j] = y[j] + h*k3[j];
21     rhs(t+h, yn, k4);
22     for(int j=0; j<n; j++) y[j] += h*(k1[j] + 2*k2[j] + 2*k3[j] + k4[j])/6.0;
23 }

```

E Matlab

E.1 ex01

```
1 % hello.m
2 fprintf('Hello, World!\n')
```

E.2 ex02

```
1 % loop.m
2 for i = 0:9
3     fprintf('i = %d\n', i)
4 end
```

E.3 ex03

```
1 % ifelse.m
2 for i = 0:9
3     if mod(i,2)==0
4         fprintf('%d even\n', i)
5     else
6         fprintf('%d odd\n', i)
7     end
8 end
```

E.4 ex04

```
1 % wloop.m
2 i=1;
3 while true
4     if mod(i,17)==0 && mod(i+2,11)==0 && mod(i+4, 23)==0
5         fprintf('i = %d\n', i)
6         break
7     end
8     i=i+1;
9 end
```

E.5 ex05

```
1 % rng.m
2 r = rand(1, 10);
3 for i=1:10
4     fprintf('%d %f\n', i, r(i))
5 end
```

E.6 ex06

```
1 % plotgraph.m
2 dt = 0.1;
3 ttime = 10.0;
4 npts = int32(ttime/dt) + 1;
5 a = rand(1, npts);
6 x = linspace(0.0, ttime, npts);
7 plot(x,a)
```

E.7 ex07

```
1 % ode.m
2 function ode
3     ttime = 10.0;
4     dt = 0.1;
5     npts = int32(ttime/dt) + 1;
6     tt = linspace(0.0, ttime, npts);
7     [t,y] = ode45(@rhs, tt, [1.0 0.0]);
8     plot(t, y(:,1), t, y(:,2))
9 end
10
11 function dydt = rhs(t, y)
12     dydt = [y(2); -y(1)];
13 end
```

E.8 ex08

```
1 % channel.m
2 function channel()
3     dt = 0.01;
4     ttime = 50.0;
5     npts = int32(ttime/dt) + 1;
6     tt = linspace(0, ttime, npts);
7     y0 = zeros(1,9);
8     y0(1) = 1.0;
9
10     vx = -1000.0;
11     [a11, a12, a13, b11, b12, b13] = deal(0.0);
12     [a2, b2, a3, b3, a4, b4, a5, b5] = deal(0.0);
13
14     [t,y] = ode45(@(tt, yy)RHS(tt, yy), tt, y0);
15     plot(t,y(:,1), t,y(:,4), t,y(:,7), t,y(:,8),t,y(:,9))
16
17     function dy = RHS(t, y)
18         if t<20.0 || t>= 30.0
19             v = -100.0;
20         else
21             v = 0.0;
22         end
23         if v ~= vx
24             a11 = 3.802/(0.1027*exp(-v/17.0) + 0.20*exp(-v/150));
25             a12 = 3.802/(0.1027*exp(-v/15.0) + 0.23*exp(-v/150));
26             a13 = 3.802/(0.1027*exp(-v/12.0) + 0.25*exp(-v/150));
27
28             b11 = 0.1917*exp(-v/20.3);
29             b12 = 0.20*exp(-(v-5)/20.3);
30             b13 = 0.22*exp(-(v-10)/20.3);
31
32             a3 = (3.7933e-7)*exp(-v/7.7);
33             b3 = 0.0084 + 0.00002*v;
34
35             a2 = 9.178*exp(v/29.68);
36             b2 = (a13*a2*a3)/(b13*b3);
37
38             a4 = a2/100;
39             b4 = a3;
40
41             a5 = a2/(9.5e4);
42             b5 = a3/50;
43             vx = v;
44         end
45
46         dy=[-(a11 + b3)*y(1) + b11*y(2) + a3*y(5);
47             -(b11 + b3 + a12)*y(2) + a11*y(1) + b12*y(3) + a3*y(6);
48             -(b12 + b3 + a13)*y(3) + a12*y(2) + b13*y(4) + a3*y(7);
49             -(b13 + a2)*y(4) + a13*y(3) + b2*y(7);
50             -(a11 + a3)*y(5) + b11*y(6) + b3*y(1);
```

```

51         -(b11 + a3 + a12)*y(6) + a11*y(5) + b12*y(7) + b3*y(2);
52         -(b12 + a3 + b2 + a4)*y(7) + a12*y(6) + b3*y(3) + a2*y(4) + b4*y(8);
53         -(b4 + a5)*y(8) + a4*y(7) + b5*y(9);
54         -b5*y(9) + a5*y(8)];
55     end % end of function RHS
56 end % of main

```

E.9 ex09

```

1 % cellpot.m
2 function cellpot
3     tic
4     m = 6;
5     dt = 0.05;
6     totalTime = 600.0;
7     npts = int32(totalTime/dt)+1;
8
9     Dm = 60.0;      % (micrometer) diameter of the cell
10    Lm = 70.0;      % (micrometer) length of the cell
11
12    dGleak = 0.002; % (mS/cm^2)
13    dGNa = 0.05;   % 0.53; % (mS/cm^2)
14    dGK = 0.15;    % 0.13; % (mS/cm^2)
15    dGTCa = 0.03;  % (mS/cm^2)
16    fac = 10;      % conversion factor (mS/cm^2) -> (pS/micrometer^2)
17
18    Eleak = -65.0; % (mV)
19    ENa = 50.0;    % (mV)
20    EK = -100.0;   % (mV)
21    ECa = 50.0;    % (mV)
22    vtraub = -55.0; % (mV)
23
24    Am = 3.14 * Dm * Lm; % (square_micrometer) surface area
25    Gleak = Am * dGleak * fac;
26    Cmm = 0.01 * Am;
27    InvCm = 1/Cmm;
28    GNa = Am * dGNa * fac;
29    GK = Am * dGK * fac;
30    GTCa = Am * dGTCa * fac;
31
32    Iinj = -2000; %2000.0; % (pA)
33    Ton = 50;
34    Toff = 300; % 40;
35    vinit = -63.27;
36
37    y = zeros(1,m);
38    y = setInitialValue(y);
39
40    t = linspace(0.0, totalTime, npts);
41    [t,y] = ode45(@(tt, yy)RHS_eqs(tt, yy), t, y);
42
43    th = -20.0;
44    sn = 0;
45    for i=1:size(y,1)-1
46        if y(i+1,1)>=th && y(i,1)<th
47            sn = sn +1;
48        end
49    end
50    fprintf('number of spikes = %d\n', sn)
51    toc
52    plot(t, y(:,1))
53
54    function [a,b] = get_Na_m(v)
55        v2 = v - vtraub;
56        a = 0.32 * (13.0-v2) / ( exp((13.0-v2)/4.0) - 1.0);
57        b = 0.28 * (v2-40.0) / ( exp((v2-40.0)/5.0) - 1.0);
58    end
59
60    function [a, b] = get_Na_h(v)

```

```

61     v2 = v - vtraub;
62     a = 0.128 * exp((17.0-v2)/18.0);
63     b = 4.0/(1.0 + exp((40.0-v2)/5.0));
64 end
65
66 function [a,b] = get_K_n(v)
67     v2 = v - vtraub;
68     a = 0.032 * (15.0-v2)/( exp((15.0-v2)/5.0) - 1.0);
69     b = 0.5 * exp((10.0-v2)/40);
70 end
71
72 function [tau, inf] = get_TCa_m(v)
73     vshift=0;
74     vs = v + vshift;
75     tau = (3.0 + 1.0 / ( exp((vs+25.0)/10) + exp(-(vs+100.0)/15.0) ) );
76     inf = 1.0 / ( 1.0 + exp(-(vs+50.0)/7.4) );
77 end
78
79 function [tau, inf] = get_TCa_h(v)
80     vshift=0;
81     vs = v + vshift;
82     tau = ( 85.0 + 1.0 / ( exp((vs+46.0)/4.0) + exp(-(vs+405.0)/50.0) ) );
83     inf = 1.0 / ( 1.0 + exp((vs+78.0)/5.0) );
84 end
85
86 function y = setInitialValue(y)
87     y(1) = vinit;
88     v = y(1);
89     [a,b] = get_Na_m(v);    y(2) = a/(a+b);
90     [a,b] = get_Na_h(v);    y(3) = a/(a+b);
91     [a,b] = get_K_n(v);     y(4) = a/(a+b);
92     [~,inf] = get_TCa_m(v); y(5) = inf;
93     [~,inf] = get_TCa_h(v); y(6) = inf;
94 end
95
96
97 function dy = RHS_eqs(t, y)
98     if t<Ton || t>=Toff
99         c = 0.0;
100     else
101         c = Iinj;
102     end
103     v = y(1);
104     INa = GNa*y(2)^3*y(3)*(v-ENa); % INa = GNa * m^3 * h * (v-ENa)
105     IK = GK * y(4)^4 * (v-EK); % IK = GK * n^4 (v-EK)
106     ITCa = GTCa * y(5)^2 * y(6) * (v - ECa);
107
108     [a1,b1] = get_Na_m(v);
109     [a2,b2] = get_Na_h(v);
110     [a3,b3] = get_K_n(v);
111     [tau1,inf1] = get_TCa_m(v);
112     [tau2,inf2] = get_TCa_h(v);
113
114     dy = [ InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa );
115           a1 - (a1+b1)*y(2);
116           a2 - (a2+b2)*y(3);
117           a3 - (a3+b3)*y(4);
118           (inf1 - y(5)) / tau1;
119           (inf2 - y(6)) / tau2];
120 end
121 end

```

E.10 ex10

```

1 % release.m
2 % Stochastic neurotransmitter release: ex12.m
3 % Ralf Schneggenburger & Erwin Neher
4 % Nature 406, 889-893, 2000}
5

```

```

6 function main()
7     tic
8
9     ttime = 4.0;    % total time
10    dt = 0.02;    % time step
11    peak = 20.0;    % peak calcium concentration
12    hw = 1.0;    % half width of calcium transient
13    iter = 100000; % number of trials
14
15    npts = int32(ttime/dt)+1;
16    tt = linspace(0.0, ttime, npts);
17    sum = zeros(1, npts, 'int32');
18    ca = zeros(1, npts, 'double');
19
20    sprintf('%d iterations\n', iter)
21
22    % ca transient
23    d = 2.0*sqrt((1.0 - sqrt(0.5)))/hw;
24    k = int32(2.0/(dt*d));
25    ca(1:k) = peak * (((d*tt(1:k) - 1.0).^2.0)-1.0).^2 ;
26
27    % kinetic constants
28    kon = 9.0e-2; % microM-1 ms-1
29    koff = 9.5; % ms-1
30    beta = 0.25; % cooperativity
31    gamma = 6.0; % 6.0d0 ! ms-1
32    epsilon = 0.3; % ms-1
33    initialState = 1;
34
35    konh = dt * kon;
36    koffh = dt * koff;
37    gammah = dt * gamma;
38    epsilonh = dt*epsilon;
39    beta2 = beta*beta;
40    beta3 = beta*beta2;
41    beta4 = beta*beta3;
42
43    counter = 0;
44    for k = 1:iter
45        s = initialState; %initial state
46        for i = 1:npts
47            r = rand; % get a random number [0, 1]
48            alpha = konh * ca(i); % get Ca level
49            snext = s;
50            switch s
51                case 1
52                    th = 5.0 * alpha;
53                    if r<th
54                        snext = 2;
55                    end
56                case 2
57                    th = 4.0 * alpha;
58                    if r < th
59                        snext = 3;
60                    else
61                        th = th + koffh;
62                        if r<th
63                            snext = 1;
64                        end
65                    end
66                case 3
67                    th = 3.0 * alpha;
68                    if r < th
69                        snext = 4;
70                    else
71                        th = th + koffh * beta;
72                        if r < th
73                            snext = 2;
74                        end
75                    end
76                case 4

```

```

77         th = 2.0 * alpha;
78         if r < th
79             snext = 5;
80         else
81             th = th + koffh * beta2;
82             if r < th
83                 snext = 3;
84             end
85         end
86     case 5
87         th = alpha;
88         if r < th
89             snext = 6;
90         else
91             th = th + koffh * beta3;
92             if r < th
93                 snext = 4;
94             end
95         end
96     case 6
97         th = gammah;
98         if r < th
99             snext = 7;
100        else
101            th = th + koffh * beta4;
102            if r < th
103                snext = 5;
104            end
105        end
106    case 7          % release
107        snext = 0;
108        sum(i) = sum(i) + 1;
109        counter = counter + 1;
110    case 0          %recovery from release
111        th = epsilonh;
112        if r < th
113            snext = 1;
114        end
115    end % end of switch
116    s = snext;
117 end % end of i-loop
118 end % end of k-loop
119 sprintf('release probability = %f\n', counter/iter)
120 toc
121 plot(tt,sum, tt, ca)
122 end

```

F Python

F.1 ex01

```
1 # hello.py
2 print('Hello, World!')
```

F.2 ex02

```
1 # loop.py
2 for i in range(10):
3     print('i= ', i)
```

F.3 ex03

```
1 # ifelse.py
2 for i in range(10):
3     if i%2 == 0:
4         print('even', i)
5     else:
6         print('odd ', i)
```

F.4 ex04

```
1 # while.py
2 i=1
3 while True:
4     if i%17==0 and (i+2)%11==0 and (i+4)%23==0:
5         print("i = ", i)
6         break
7     i=i+1
```

F.5 ex05

```
1 # rng.py
2 import numpy as np
3 rng = np.random.RandomState()
4 a = rng.uniform(0.0, 1.0, 10)
5 for i in range(10):
6     print(i, a[i])
```

F.6 ex06

```
1 # plotgraph.py
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5
6 dt = 0.1
7 ttime = 10.0
8 npts = int(10.0/0.1) + 1
9 rng = np.random.RandomState()
10 a=rng.uniform(0.0, 1.0, npts)
11 x = np.linspace(0.0, ttime, npts)
12
13 plt.figure()
14 plt.plot(x,a)
```

```

15 plt.show()
16 """
17 f = open('a.txt','w')
18 for i in range(101):
19     f.write('%f %f\n'%(x[i], a[i]))
20 f.close()
21 os.system('gc a.txt &')
22 """

```

F.7 ex07

```

1 # ode.py
2 import numpy as np
3 from scipy.integrate import odeint
4 import matplotlib.pyplot as plt
5
6 def rhs(y,t): # return derivatives of the array y
7     return np.array([y[1], -y[0]])
8
9 def ode():
10     ndim = 2
11     ttime = 10.0
12     dt = 0.1
13     npts = int(ttime/dt)+1
14     y0 = np.zeros(ndim)
15     y0[0] = 1.0
16
17     t = np.linspace(0.0, ttime, npts)
18     y = odeint(rhs, y0, t)
19
20     plt.figure()
21     plt.plot(t, y[:,0], t, y[:,1])
22     plt.xlabel('t')
23     plt.ylabel('y')
24     plt.show()
25
26 if __name__ == '__main__':
27     ode()

```

F.8 ex08

```

1 # channel.py
2 import numpy as np
3 import math
4 from scipy.integrate import odeint
5 import matplotlib.pyplot as plt
6
7 def channel():
8     ndim = 9
9     dt = 0.01
10     totalTime = 50.0;
11     npts = (int)(totalTime/dt)+1;
12     vold = -1000.0
13     a2 = a3 = a4 = a5 = a11 = a12 = a13 = 0.0
14     b2 = b3 = b4 = b5 = b11 = b12 = b13 = 0.0
15     def rhs(y, t):
16         if t<20.0 or t>= 30.0:
17             v = -100.0
18         else:
19             v = 0.0
20         if v != vold:
21             a11 = 3.802/(0.1027*math.exp(-v/17.0) + 0.20*math.exp(-v/150.0))
22             a12 = 3.802/(0.1027*math.exp(-v/15.0) + 0.23*math.exp(-v/150.0))
23             a13 = 3.802/(0.1027*math.exp(-v/12.0) + 0.25*math.exp(-v/150.0))
24
25             b11 = 0.1917*math.exp(-v/20.3)

```

```

26     b12 = 0.20*math.exp(-(v-5)/20.3)
27     b13 = 0.22*math.exp(-(v-10)/20.3)
28
29     a3 = (3.7933e-7)*math.exp(-v/7.7)
30     b3 = 0.0084 + 0.00002*v
31
32     a2 = 9.178*math.exp(v/29.68)
33     b2 = (a13*a2*a3)/(b13*b3)
34
35     a4 = a2/100.0
36     b4 = a3
37
38     a5 = a2/(9.5e4)
39     b5 = a3/50.0
40     v = vold
41     return np.array(
42         [-(a11 + b3)*y[0] + b11*y[1] + a3*y[4],
43          -(b11 + b3 + a12)*y[1] + a11*y[0] + b12*y[2] + a3*y[5],
44          -(b12 + b3 + a13)*y[2] + a12*y[1] + b13*y[3] + a3*y[6],
45          -(b13 + a2)*y[3] + a13*y[2] + b2*y[6],
46          -(a11 + a3)*y[4] + b11*y[5] + b3*y[0],
47          -(b11 + a3 + a12)*y[5] + a11*y[4] + b12*y[6] + b3*y[1],
48          -(b12 + a3 + b2 + a4)*y[6] + a12*y[5] + b3*y[2] + a2*y[3] + b4*y[7],
49          -(b4 + a5)*y[7] + a4*y[6] + b5*y[8],
50          -b5*y[8] + a5*y[7] ])
51
52     y0 = np.zeros(ndim)
53     y0[0] = 1.0
54     t = np.linspace(0.0, totalTime, npts)
55     y = odeint(rhs, y0, t)
56
57     plt.figure()
58     plt.plot(t, y[:,0], t, y[:,3], t, y[:,6], t, y[:,7], t, y[:,8])
59     plt.xlabel('time (ms)')
60     plt.ylabel('probability')
61     plt.show()
62
63 if __name__ == '__main__':
64     channel()

```

F.9 ex09

```

1 # cellpot.py
2 import os
3 import numpy as np
4 import scipy.integrate
5 import math
6 import matplotlib.pyplot as plt
7
8 def cellpot():
9     ndim = 6
10    ttime = 600.0
11    dt = 0.05
12    npts = int(ttime/dt)+1
13    istate = 0
14
15    Dm = 60.0      # (micrometer) diameter of the cell
16    Lm = 70.0      # (micrometer) length of the cell
17
18    Eleak = -65.0 # (mV)
19    ENa = 50.0 # (mV)
20    EK = -100.0 # (mV)
21    ECa = 50.0 # (mV)
22
23    dGleak = 0.002 # (mS/cm^2)
24    dGNa = 0.05 # 0.53; // (mS/cm^2)
25    dGK = 0.15 # 0.13; // (mS/cm^2)
26    dGTCa = 0.03 # (mS/cm^2)
27

```

```

28 vtraub = -55.0 # (mV)
29
30 Iinj = -2000.0 # 2000.0; //(pA)
31 Ton = 50.0
32 Toff = 300.0 # 40;
33 vinit = -63.27
34
35 fac = 10 # conversion factor (mS/cm^2) -> (pS/micrometer^2)
36
37 Am = 3.14 * Dm * Lm # (square_micrometer) surface area
38 Cm = 0.01 * Am
39 InvCm = 1/Cm
40
41 Gleak = Am * dGleak * fac
42 GNa = Am * dGNa * fac
43 GK = Am * dGK * fac
44 GTCa = Am * dGTCa * fac
45
46 def get_Na_m(v):
47     v2 = v-vtraub
48     a = 0.32 * (13-v2) / ( math.exp((13-v2)/4) - 1)
49     b = 0.28 * (v2-40) / ( math.exp((v2-40)/5) - 1)
50     return [a,b]
51
52 def get_Na_h(v):
53     v2 = v-vtraub
54     a = 0.128 * math.exp((17-v2)/18)
55     b = 4 / ( 1 + math.exp((40-v2)/5) )
56     return [a,b]
57
58 def get_K_n(v):
59     v2 = v - vtraub
60     a = 0.032 * (15-v2) / ( math.exp((15-v2)/5) - 1)
61     b = 0.5 * math.exp((10-v2)/40)
62     return [a,b]
63
64 def get_TCa_m(v):
65     vshift=0
66     vs = v + vshift
67     tau = ( 3 + 1.0 / ( math.exp((vs+25)/10) + math.exp(-(vs+100)/15)))
68     inf = 1.0 / ( 1 + math.exp(-(vs+50)/7.4) )
69     return [tau,inf]
70
71 def get_TCa_h(v):
72     vshift=0
73     vs = v + vshift
74     tau = (85.0+1.0/(math.exp((vs+46.0)/4.0) + math.exp(-(vs+405.0)/50.0) ) )
75     inf = 1.0 / ( 1.0 + math.exp((vs+78.0)/5.0) )
76     return [tau, inf]
77
78 def setInitialValues(y):
79     y[0] = vinit
80     v = y[0]
81     [a,b] = get_Na_m(v)
82     y[1] = a/(a+b)
83
84     [a,b] = get_Na_h(v)
85     y[2] = a/(a+b)
86
87     [a,b] = get_K_n(v)
88     y[3] = a/(a+b)
89
90     [tau,inf] = get_TCa_m(v)
91     y[4] = inf
92
93     [tau,inf] = get_TCa_h(v)
94     y[5] = inf
95
96
97 def rhs(y,t): # return derivatives of the array y
98     if t >= Ton and t < Toff :

```

```

99         c = Iinj
100     else:
101         c = 0.0
102
103     v = y[0];
104     INa = GNa * y[1]**3 * y[2] * (v-ENa)
105     IK = GK * y[3]**4 * (v-EK)
106     ITCa = GTCa * pow(y[4],2.0) * y[5] * (v - ECa)
107
108     [a1,b1]=get_Na_m(v)
109     [a2,b2] = get_Na_h(v)
110     [a3,b3] = get_K_n(v)
111     [tau1,inf1] = get_TCa_m(v)
112     [tau2,inf2] = get_TCa_h(v)
113
114     return np.array(
115         [ InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa ),
116           a1 - (a1+b1)*y[1],
117           a2 - (a2+b2)*y[2],
118           a3 - (a3+b3)*y[3],
119           (inf1 - y[4]) / tau1,
120           (inf2 - y[5]) / tau2])
121
122 # main
123 t = np.linspace(0.0, ttime, npts)
124 y0 = np.zeros(ndim)
125 setInitialValues(y0)
126 y = scipy.integrate.odeint(rhs, y0, t)
127
128 th = -20.0
129 sn = 0
130 for i in range(npts-1):
131     if y[i+1,0] >= th and y[i,0] < th:
132         sn = sn+1
133 print('number of spikes = ', sn)
134
135 plt.figure()
136 plt.plot(t, y[:,0])
137 plt.xlabel('time (ms)')
138 plt.ylabel('voltage (mV)')
139 plt.show()
140
141 if __name__=='__main__':
142     cellpot()

```

F.10 ex10

```

1  """
2  release.py
3  Stochastic neurotransmitter release
4  Ralf Schneggenburger & Erwin Neher
5  Nature 406, 889-893, 2000
6  """
7  import numpy as np
8  import random
9  import math
10 import matplotlib.pyplot as plt
11
12 def release():
13     ttime = 4.0
14     dt = 0.02
15     peak = 20.0
16     hw = 1.0
17     iter = 100000
18
19     npts = (int)(ttime/dt)+1
20     tt = np.linspace(0.0, ttime, npts)
21     sum = np.zeros(npts, int)
22     ca = np.zeros(npts, float)

```

```

23
24 print('{0:d} iterations'.format(iter))
25
26 # ca transient
27 d = 2.0*math.sqrt((1.0 - math.sqrt(0.5)))/hw
28 k = (int)(2.0/(dt*d))
29 ca[0:k] = peak * pow(pow(d*tt[0:k] - 1.0, 2.0)-1.0, 2.0)
30
31 kon  = 9.0e-2 # microM-1 ms-1
32 koff = 9.5    # ms-1
33 beta = 0.25   # cooperativity
34 gamma = 6.0  # 6.0d0 ! ms-1
35 epsilon = 0.3 # ms-1
36 initialState = 1
37
38 konh  = dt * kon
39 koffh = dt * koff
40 gammah = dt * gamma
41 epsilonh = dt*epsilon
42 beta2 = beta*beta
43 beta3 = beta*beta2
44 beta4 = beta*beta3
45
46 counter = 0
47 random.seed()
48
49 for k in range(iter):
50     s = initialState #initial state
51     for i in range(npts):
52         r = random.random() # get a random number [0, 1]
53         alpha = konh * ca[i] # get Ca level
54         snext = s
55         if s == 1:
56             th = 5.0 * alpha
57             if r < th:
58                 snext = 2
59         elif s == 2:
60             th = 4.0 * alpha
61             if r < th:
62                 snext = 3
63             else:
64                 th = th + koffh
65                 if r < th:
66                     snext = 1
67         elif s == 3:
68             th = 3.0 * alpha
69             if r < th:
70                 snext = 4
71             else:
72                 th = th + koffh * beta
73                 if r < th:
74                     snext = 2
75         elif s == 4:
76             th = 2.0 * alpha
77             if r < th:
78                 snext = 5
79             else:
80                 th = th + koffh * beta2
81                 if r < th:
82                     snext = 3
83         elif s == 5:
84             th = alpha
85             if r < th:
86                 snext = 6
87             else:
88                 th = th + koffh * beta3
89                 if r < th:
90                     snext = 4
91         elif s == 6:
92             th = gammah
93             if r < th:

```

```

94         snext = 7
95     else:
96         th = th + koffh * beta4
97         if r < th:
98             snext = 5
99     elif s == 7:
100         snext = 0      #release
101         sum[i] += 1    # release
102         counter += 1
103     elif s == 0:      #recovery from release
104         th = epsilonh
105         if r < th:
106             snext = 1
107
108     s = snext;
109
110     print('release probability = {0:.4f}'.format(float(counter)/iter))
111
112 #     plt.figure()
113 #     plt.plot(tt,sum, tt, ca)
114 #     plt.show()
115
116 if __name__ == '__main__':
117     release()

```

G Java

G.1 ex01

```
1 // hello.java
2 public class hello {
3     public static void main(String [] args){
4         System.out.println("Hello, World!");
5     }
6 }
```

G.2 ex02

```
1 // loop.java
2 public class loop {
3     public static void main(String [] args){
4         for(int i=0;i<10;i++){
5             System.out.println("i = " + i);
6         }
7     }
8 }
```

G.3 ex03

```
1 // ifelse.java
2 public class ifelse {
3     public static void main(String [] args){
4
5         for(int i=0;i<10;i++){
6             if(i%2==0){
7                 System.out.println("even: " + i);
8             }else{
9                 System.out.println("odd: " + i);
10            }
11        }
12    }
13 }
```

G.4 ex04

```
1 // while.java
2 public class wloop {
3     public static void main(String [] args){
4         int i=1;
5         while(true){
6             if(i%17==0 && (i+2)%11==0 && (i+4)%23==0){
7                 System.out.println("i = " + i);
8                 break;
9             }
10            i++;
11        }
12    }
13 }
```

G.5 ex05

```
1 // rng.java
2 import java.util.Random;
3 public class rng{
```

```

4     public static void main(String [] args){
5         Random ran = new Random();
6         for(int i=0;i<10;i++){
7             System.out.println(i + ": " + ran.nextDouble());
8         }
9     }
10 }

```

G.6 ex06

```

1 // plotgraph.java
2 import java.util.Random;
3 import java.io.FileWriter;
4 import java.io.IOException;
5
6 public class plotgraph{
7
8     public static void main(String [] args){
9         double dt = 0.1;
10        double ttime = 10.0;
11        int npts = (int)(ttime/dt) + 1;
12        Random ran = new Random();
13
14        try{
15            FileWriter fw = new FileWriter("ax.txt");
16            for(int i=0;i<npts;i++){
17                double a = ran.nextDouble();
18                fw.write(dt*i + " " + a + "\n");
19            }
20            fw.close();
21        }catch(IOException e){
22            System.out.println(e);
23        }
24
25
26        String cmd;
27        String OS = System.getProperty("os.name", "generic").toLowerCase();
28        if (OS.indexOf("win") >= 0) {
29            cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
30        }else{ // mac
31            cmd = "../..\\tools//gc ax.txt";
32        }
33
34        try{
35            Process p = Runtime.getRuntime().exec(cmd);
36            p.waitFor();
37        } catch (Exception e) {}
38    }
39 }

```

G.7 ex07a

```

1 // ode.java (ex07)
2 import java.io.FileWriter;
3 import java.io.IOException;
4
5 public class ode{
6
7     public static void main(String [] args){
8
9         int n = 2; // number of equations
10        double h = 0.1; // step size
11        double ttime = 10.0; // total time
12        int npts = (int)(ttime/h)+1; // number of points
13        double [] y = {1.0, 0.0}; // initial values
14

```

```

15     Eqs eq = new Eqs();
16     eq.initRK4(n, h);
17
18     try{
19         FileWriter fw = new FileWriter("ax.txt");
20         for(int i=0;i<npts;i++){
21             double t = h*i;
22             fw.write(t + " " + y[0] + " " + y[1] + "\n");
23             eq.advanceStep(t, y);
24         }
25         fw.close();
26     }catch(IOException e){
27         System.out.println(e);
28     }
29
30     String cmd;
31     String OS = System.getProperty("os.name", "generic").toLowerCase();
32     if (OS.indexOf("win") >= 0) {
33         cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
34     }else{ // mac
35         cmd = "../..//tools//gc ax.txt";
36     }
37
38     try{
39         Process p = Runtime.getRuntime().exec(cmd);
40         p.waitFor();
41     } catch (Exception e) {}
42 }
43 }

```

```

1
2 public class Eqs extends RungeKutta4 {
3
4     public void rhs(double tt, double [] y, double [] dy){
5         dy[0] = y[1];
6         dy[1] = -y[0];
7     }
8 }

```

```

1
2 public class RungeKutta4{
3     int n;
4     double h;
5     double [] yn;
6     double [] k1;
7     double [] k2;
8     double [] k3;
9     double [] k4;
10
11     public void initRK4(int ndim, double dt){
12         n = ndim;
13         h = dt;
14         yn = new double[n];
15         k1 = new double[n];
16         k2 = new double[n];
17         k3 = new double[n];
18         k4 = new double[n];
19     }
20
21     public void advanceStep(double t, double [] y){
22         rhs(t, y, k1);
23         for(int i=0;i<n;i++) yn[i] = y[i] + h*k1[i]/2;
24         rhs(t+h/2, yn, k2);
25         for(int i=0;i<n;i++) yn[i] = y[i] + h*k2[i]/2;
26         rhs(t+h/2, yn, k3);
27         for(int i=0;i<n;i++) yn[i] = y[i] + h*k3[i];
28         rhs(t+h, yn, k4);
29         for(int i=0;i<n;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
30     }
31     public void rhs(double tt, double [] y, double [] dy){}

```

```
32 }
```

G.8 ex08

```
1 // channel.java (ex08)
2 import java.io.FileWriter;
3 import java.io.IOException;
4
5 public class channel{
6
7     public static void main(String [] args){
8         int n = 9; // number of equations
9         double h = 0.01; // step size
10        double totalTime = 50.0; // total time
11        int npts = (int)(totalTime /h)+1; // number of points
12        double [] y = {1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}; // initial values
13
14        Eqs eq = new Eqs();
15        eq.initRK4(n, h);
16
17        try{
18            FileWriter fw = new FileWriter("ax.txt");
19            for(int i=0;i<npts;i++){
20                double t = h*i;
21                fw.write(t+" "+ y[0] + " "+ y[3]+ " "+ y[6]+ " "+ y[7]+ " "+ y[8] + "\n");
22                eq.advanceStep(t, y);
23            }
24            fw.close();
25        }catch(IOException e){
26            System.out.println(e);
27        }
28
29
30
31        String cmd;
32        String OS = System.getProperty("os.name", "generic").toLowerCase();
33        if (OS.indexOf("win") >= 0) {
34            cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
35        }else{ // mac
36            cmd = "../..//tools//gc ax.txt";
37        }
38
39        try{
40            Process p = Runtime.getRuntime().exec(cmd);
41            p.waitFor();
42        } catch (Exception e) {}
43
44
45
46    }
47 }
48 }
```

```
1 public class Eqs extends RungeKutta4 {
2
3     double a11, a12, a13, a2, a3, a4, a5;
4     double b11, b12, b13, b2, b3, b4, b5;
5     double vold;
6
7     public Eqs(){
8         vold = -1000.0;
9     }
10
11    public void rhs(double t, double [] y, double [] dy){
12        double v = t<20.0 || t>= 30.0 ? -100.0 : 0.0;
13        if(v != vold){
14            a11 = 3.802/(0.1027*Math.exp(-v/17.0) + 0.20*Math.exp(-v/150));
15            a12 = 3.802/(0.1027*Math.exp(-v/15.0) + 0.23*Math.exp(-v/150));
```

```

16         a13 = 3.802/(0.1027*Math.exp(-v/12.0) + 0.25*Math.exp(-v/150));
17
18         b11 = 0.1917*Math.exp(-v/20.3);
19         b12 = 0.20*Math.exp(-(v-5)/20.3);
20         b13 = 0.22*Math.exp(-(v-10)/20.3);
21
22         a3 = (3.7933e-7)*Math.exp(-v/7.7);
23         b3 = 0.0084 + 0.00002*v;
24
25         a2 = 9.178*Math.exp(v/29.68);
26         b2 = (a13*a2*a3)/(b13*b3);
27
28         a4 = a2/100;
29         b4 = a3;
30
31         a5 = a2/(9.5e4);
32         b5 = a3/50;
33         vold = v;
34     }
35     dy[0] = -(a11 + b3)*y[0] + b11*y[1] + a3*y[4];
36     dy[1] = -(b11 + b3 + a12)*y[1] + a11*y[0] + b12*y[2] + a3*y[5];
37     dy[2] = -(b12 + b3 + a13)*y[2] + a12*y[1] + b13*y[3] + a3*y[6];
38     dy[3] = -(b13 + a2)*y[3] + a13*y[2] + b2*y[6];
39     dy[4] = -(a11 + a3)*y[4] + b11*y[5] + b3*y[0];
40     dy[5] = -(b11 + a3 + a12)*y[5] + a11*y[4] + b12*y[6] + b3*y[1];
41     dy[6] = -(b12 + a3 + b2 + a4)*y[6] + a12*y[5] + b3*y[2] + a2*y[3] + b4*y[7];
42     dy[7] = -(b4 + a5)*y[7] + a4*y[6] + b5*y[8];
43     dy[8] = -b5*y[8] + a5*y[7];
44 }
45 }

```

G.9 ex09

```

1 // Cellpot java
2 import java.io.FileWriter;
3 import java.io.IOException;
4
5 public class Cellpot{
6
7     public static void main(String [] args){
8         int n = 6; // number of equations
9         double h = 0.05; // step size
10        double totalTime = 600.0; // total time
11        int npts = (int)(totalTime /h)+1; // number of points
12        double vinit = -63.27; // initial voltate
13        double Iinj = -2000.0; // current injection(pA)
14        double Ton = 50.0; // on-time
15        double Toff = 300.0; // off-time
16
17        double [] y = new double[n];
18        double [] vol = new double[npts];
19
20        Cell cell = new Cell();
21        cell.initRK4(n, h);
22        cell.setInitialValues(vinit, y);
23        cell.setPulse(Ton, Toff, Iinj);
24
25        try{
26            FileWriter fw = new FileWriter("ax.txt");
27            for(int i=0;i<npts;i++){
28                double t = h*i;
29                vol[i] = y[0];
30                fw.write(t + " " + y[0] + "\n");
31                cell.advanceStep(t, y);
32            }
33            fw.close();
34        }catch(IOException e){
35            System.out.println(e);
36        }
37    }
38 }

```

```

37
38
39     String cmd;
40     String OS = System.getProperty("os.name", "generic").toLowerCase();
41     if (OS.indexOf("win") >= 0) {
42         cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
43     } else { // mac
44         cmd = "../..\\tools//gc ax.txt";
45     }
46
47     try{
48         Process p = Runtime.getRuntime().exec(cmd);
49         p.waitFor();
50     } catch (Exception e) {}
51
52 }
53 }

```

```

1
2
3 public class Cell extends RungeKutta4 {
4
5     final double Dm = 60.0; // (micrometer) diameter of the cell
6     final double Lm = 70.0; // (micrometer) length of the cell
7
8     final double Eleak = -65.0; // (mV)
9     final double ENa = 50.0; // (mV)
10    final double EK = -100.0; // (mV)
11    final double ECa = 50.0; // (mV)
12
13    final double dGleak = 0.002; // (mS/cm^2)
14    final double dGNa = 0.05; // 0.53; // (mS/cm^2)
15    final double dGK = 0.15; // 0.13; // (mS/cm^2)
16    final double dGTCa = 0.03; // (mS/cm^2)
17
18    final double vtraub = -55.0; // -55.0 mV
19    final double fac = 10; // conversion factor (mS/cm^2) -> (pS/micrometer^2)
20
21    int ndim;
22    // double v, vinit;
23    double Iinj; // 2000.0 (pA)
24    double Ton; // 20 ms
25    double Toff; // 40 ms
26
27    double Am; // surface area
28    double Cm;
29    double InvCm; // 1/Cm;
30
31    double Gleak;
32    double GNa;
33    double GK;
34    double GTCa;
35    double v_init;
36
37    // public Cell::setInitialValues(double v_init, vector<double>& y){
38    public void setInitialValues(double vinit, double [] y){
39        Am = 3.14 * Dm * Lm; // (square_micrometer) surface area
40        Cm = 0.01 * Am;
41        InvCm = 1.0/Cm;
42
43        Gleak = Am * dGleak * fac;
44        GNa = Am * dGNa * fac;
45        GK = Am * dGK * fac;
46        GTCa = Am * dGTCa * fac;
47
48        double v = y[0] = v_init = vinit;
49        double [] aa = new double[2];
50
51        get_Na_m(v, aa);
52        y[1] = aa[0]/(aa[0]+aa[1]);

```

```

53     get_Na_h(v,aa);
54     y[2] = aa[0]/(aa[0]+aa[1]);
55
56     get_K_n(v,aa);
57     y[3] = aa[0]/(aa[0]+aa[1]);
58
59
60     get_TCa_m(v,aa);
61     y[4] = aa[1];
62
63     get_TCa_h(v,aa);
64     y[5] = aa[1];
65 }
66
67
68 public void setPulse(double ton, double toff, double current){
69     Ton = ton;
70     Toff = toff;
71     Iinj = current;
72 }
73
74 private void get_Na_m(double v, double [] aa){
75     double v2 = v-vtraub;
76     double a = 0.32*(13.0-v2)/(Math.exp((13.0-v2)/4.0)-1.0);
77     double b = 0.28*(v2-40.0)/(Math.exp((v2-40.0)/5.0)-1.0);
78     aa[0] = a;
79     aa[1] = b;
80 }
81
82 private void get_Na_h(double v, double [] aa){
83     double v2 = v-vtraub;
84     double a = 0.128 * Math.exp((17.0-v2)/18.0);
85     double b = 4.0/(1.0 + Math.exp((40.0-v2)/5.0));
86     aa[0] = a;
87     aa[1] = b;
88 }
89
90 private void get_K_n(double v, double [] aa){
91     double v2 = v - vtraub;
92     double a = 0.032*(15.0-v2)/(Math.exp((15.0-v2)/5.0)-1.0);
93     double b = 0.5*Math.exp((10.0-v2)/40.0);
94     aa[0] = a;
95     aa[1] = b;
96 }
97
98 private void get_TCa_m(double v, double [] aa){
99     double vshift=0.0;
100     double vs = v + vshift;
101     double tau = (3.0+1.0/(Math.exp((vs+25.0)/10.0)+Math.exp(-(vs+100.0)/15.0)));
102     double inf = 1.0/(1.0+Math.exp(-(vs+50.0)/7.4));
103     aa[0] = tau;
104     aa[1] = inf;
105 }
106
107 private void get_TCa_h(double v, double [] aa){
108     double vshift=0.0;
109     double vs = v + vshift;
110     double tau = (85.0+1.0/(Math.exp((vs+46.0)/4.0)+Math.exp(-(vs+405.0)/50.0))) ;
111     double inf = 1.0/(1.0+Math.exp((vs+78.0)/5.0));
112     aa[0] = tau;
113     aa[1] = inf;
114 }
115
116 public void rhs(double tt, double [] y, double [] dy){
117     double INa, IK, ITCa;
118     double a, b, tau, inf;
119     double shift = 0.0;
120     double v = y[0];
121     double c = (tt<Ton || tt>Toff)? 0.0 : Iinj;
122
123     double [] aa = new double[2];

```

```

124
125     INa = GNa * Math.pow(y[1],3) * y[2] * (v-ENa); // INa = GNa * m^3 * h * (v-Ena)
126     IK = GK * Math.pow(y[3],4) * (v-EK); // IK = GK * n^4 (v-EK)
127     ITCa = GTCa * Math.pow(y[4],2) * y[5] * (v - ECa);
128     dy[0] = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa);
129     // Na channel
130     get_Na_m(v, aa);
131     dy[1] = aa[0] - (aa[0]+aa[1])*y[1];
132
133     get_Na_h(v, aa);
134     dy[2] = aa[0] - (aa[0]+aa[1])*y[2];
135     // K channel (delayed rectifier)
136     get_K_n(v, aa);
137     dy[3] = aa[0] - (aa[0]+aa[1])*y[3];
138     // T-type Ca channel
139     get_TCa_m(v, aa);
140     dy[4] = (aa[1] - y[4]) / aa[0];
141     get_TCa_h(v, aa);
142     dy[5] = (aa[1] - y[5]) / aa[0];
143 }
144 }

```

G.10 ex10

```

1 // release.java
2 // Stochastic neurotransmitter release
3 // Ralf Schneggenburger & Erwin Neher
4 // Nature 406, 889-893, 2000.
5
6 import java.util.Random;
7 import java.io.FileWriter;
8 import java.io.IOException;
9
10 public class Release{
11
12     public static void main(String [] args){
13         double ttime = 4.0;
14         double dt=0.02;
15         double peak = 20.0;
16         double hw = 1.0;
17         int iter = 100000;
18
19         double kon = 9.0e-2; // microM^{-1} ms^{-1}
20         double koff = 9.5; // ms^{-1}
21         double beta = 0.25; // cooperativity
22         double gamma = 6.0; // 6.0d0 ! ms^{-1}
23         double epsilon = 0.3; // ms^{-1}
24         int initialState = 1;
25
26         System.out.println(iter + " iterations");
27
28         int npts = (int)(ttime/dt);
29         int counter = 0;
30         double [] ca = new double[npts];
31         int [] sum = new int[npts];
32
33         java.util.Arrays.fill(ca,0.0);
34         java.util.Arrays.fill(sum,0);
35
36         Random ran = new Random();
37
38         // ca transient
39         double d = 2.0*Math.sqrt((1.0 - Math.sqrt(0.5)))/hw;
40         for(int i=0;i<(int)(2.0/(dt*d));i++){
41             ca[i] = peak * Math.pow(Math.pow(d*dt*i - 1.0, 2.0)-1.0, 2.0);
42         }
43
44         double konh = dt * kon;
45         double koffh = dt * koff;

```

```

46     double gammah = dt * gamma;
47     double epsilonh = dt*epsilon;
48     double beta2 = beta*beta;
49     double beta3 = beta*beta2;
50     double beta4 = beta*beta3;
51
52     for(int k=0;k<iter;k++){ // iteration loop
53         int s = initialState; //initial state
54         for(int i=0;i<npts;i++){
55             double th;
56             double r = ran.nextDouble(); // get a random number [0, 1]
57             double alpha = konh * ca[i]; // get Ca level
58             int snext = s;
59             switch (s){
60                 case (1):
61                     th = 5.0 * alpha;
62                     if(r<th) snext = 2;
63                     break;
64                 case (2):
65                     th = 4.0 * alpha;
66                     if(r<th) {
67                         snext = 3;
68                     }else{
69                         th = th + koffh;
70                         if(r<th) snext = 1;
71                     }
72                     break;
73                 case (3):
74                     th = 3.0 * alpha;
75                     if(r<th) {
76                         snext = 4;
77                     }else{
78                         th = th + koffh * beta;
79                         if(r<th) snext = 2;
80                     }
81                     break;
82                 case (4):
83                     th = 2.0 * alpha;
84                     if(r<th){
85                         snext = 5;
86                     }else{
87                         th = th + koffh * beta2;
88                         if(r<th) snext = 3;
89                     }
90                     break;
91                 case (5):
92                     th = alpha;
93                     if(r<th){
94                         snext = 6;
95                     }else{
96                         th = th + koffh * beta3;
97                         if(r<th) snext = 4 ;
98                     }
99                     break;
100                case (6):
101                    th = gammah;
102                    if(r<th){
103                        snext = 7;
104                    }else{
105                        th = th + koffh * beta4;
106                        if(r<th) snext = 5;
107                    }
108                    break;
109                case (7):
110                    snext = 0; //release
111                    sum[i] += 1; // release
112                    counter++;
113                    break;
114                case (0): //recovery from release
115                    th = epsilonh;
116                    if(r<th) snext = 1;

```

```

117         break;
118         default:
119             ;
120     } // end of switch
121     s = snext;
122 } //end of i-loop
123 } // end of k-loop
124
125 System.out.println("release probability = " + (double)counter/(double)iter);
126
127 try{
128     FileWriter fw = new FileWriter("ax.txt");
129     for(int i=0;i<npts;i++){
130         fw.write(dt*i+" "+ ca[i] +" "+ (double)sum[i]+"\\n");
131     }
132     fw.close();
133 }catch(IOException e){
134     System.out.println(e);
135 }
136
137
138
139 String cmd;
140 String OS = System.getProperty("os.name", "generic").toLowerCase();
141 if (OS.indexOf("win") >= 0) {
142     cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
143 }else{ // mac
144     cmd = "../..//tools//gc ax.txt";
145 }
146
147 try{
148     Process p = Runtime.getRuntime().exec(cmd);
149     p.waitFor();
150 } catch (Exception e) {}
151 }
152 }

```

H Fortran

H.1 ex01

```
1 program hello
2   write(*,*) 'Hello, World!'
3 end program hello
```

H.2 ex02

```
1 program loop
2   implicit none
3   integer :: i
4
5   do i = 0, 9
6     write (*, fmt='("i = ", i3)') i
7   end do
8 end program loop
```

H.3 ex03

```
1 ! ifelse.f90
2 program ifelse
3   implicit none
4   integer :: i
5
6   do i = 0, 9
7     if( mod(i,2) == 0) then
8       write (*, fmt='(i2, " even")') i
9     else
10      write (*, fmt='(i2, " odd")') i
11    end if
12  end do
13
14 end program ifelse
```

H.4 ex04

```
1 ! wloop.f90
2 program wloop
3   implicit none
4   integer :: i
5
6   i = 1
7   do while (.true.)
8     if(mod(i,17)==0 .and. mod(i+2,11)==0 .and. mod(i+4, 23)==0)then
9       write(*, fmt='("i = ", I6)') i
10      exit
11    end if
12    i = i+1
13  end do
14 end program wloop
```

H.5 ex05

```
1 program rng
2   use mtmod
3   implicit none
4
```

```

5     integer, parameter :: npts = 10
6     integer :: j, k, c
7
8     call system_clock(count=c)
9     call sgrnd(c)
10
11    do j=0,npts-1
12        write (*, fmt='(I2, " ", F9.6)') j, grnd()
13    enddo
14
15 end program rng

```

```

1  !-----
2  ! A C-program for MT19937: Real number version
3  !   genrand() generates one pseudorandom real number (double)
4  ! which is uniformly distributed on [0,1]-interval, for each
5  ! call. sgenrand(seed) set initial values to the working area
6  ! of 624 words. Before genrand(), sgenrand(seed) must be
7  ! called once. (seed is any 32-bit integer except for 0).
8  ! Integer generator is obtained by modifying two lines.
9  !   Coded by Takuji Nishimura, considering the suggestions by
10 !   Topher Cooper and Marc Rieffel in July-Aug. 1997.
11 !
12 ! This library is free software; you can redistribute it and/or
13 ! modify it under the terms of the GNU Library General Public
14 ! License as published by the Free Software Foundation; either
15 ! version 2 of the License, or (at your option) any later
16 ! version.
17 ! This library is distributed in the hope that it will be useful,
18 ! but WITHOUT ANY WARRANTY; without even the implied warranty of
19 ! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
20 ! See the GNU Library General Public License for more details.
21 ! You should have received a copy of the GNU Library General
22 ! Public License along with this library; if not, write to the
23 ! Free Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
24 ! 02111-1307 USA
25 !
26 ! Copyright (C) 1997 Makoto Matsumoto and Takuji Nishimura.
27 ! When you use this, send an email to: matumoto@math.keio.ac.jp
28 ! with an appropriate reference to your work.
29 !
30 !*****
31 ! Fortran translation by Hiroshi Takano. Jan. 13, 1999.
32 !
33 !   genrand()      -> double precision function grnd()
34 !   sgenrand(seed) -> subroutine sgrnd(seed)
35 !                   integer seed
36 !
37 ! This program uses the following non-standard intrinsics.
38 !   ishft(i,n): If n>0, shifts bits in i by n positions to left.
39 !               If n<0, shifts bits in i by n positions to right.
40 !   iand (i,j): Performs logical AND on corresponding bits of i and j.
41 !   ior  (i,j): Performs inclusive OR on corresponding bits of i and j.
42 !   ieor (i,j): Performs exclusive OR on corresponding bits of i and j.
43 !
44 !*****
45 ! Fortran version rewritten as an F90 module and mt state saving and getting
46 ! subroutines added by Richard Woloshyn. (rwww@triumf.ca). June 30, 1999
47
48 module mtmod
49 ! Default seed
50     integer, parameter :: defaultsd = 4357
51 ! Period parameters
52     integer, parameter :: N = 624, N1 = N + 1
53
54 ! the array for the state vector
55     integer, save, dimension(0:N-1) :: mt
56     integer, save                      :: mti = N1
57
58 ! Overload procedures for saving and getting mt state

```

```

59     interface mtsave
60         module procedure mtsavef
61         module procedure mtsaveu
62     end interface
63     interface mtget
64         module procedure mtgetf
65         module procedure mtgetu
66     end interface
67
68     contains
69
70     !Initialization subroutine
71     subroutine sgrnd(seed)
72         implicit none
73     !
74     !     setting initial seeds to mt[N] using
75     !     the generator Line 25 of Table 1 in
76     !     [KNUTH 1981, The Art of Computer Programming
77     !     Vol. 2 (2nd Ed.), pp102]
78     !
79     integer, intent(in) :: seed
80
81     mt(0) = iand(seed,-1)
82     do mti=1,N-1
83         mt(mti) = iand(69069 * mt(mti-1),-1)
84     enddo
85     !
86     return
87     end subroutine sgrnd
88
89     !Random number generator
90     real(8) function grnd()
91         implicit integer(a-z)
92
93     ! Period parameters
94     integer, parameter :: M = 397, MATA = -1727483681
95     !                                     constant vector a
96     integer, parameter :: LMASK = 2147483647
97     !                                     least significant r bits
98     integer, parameter :: UMASK = -LMASK - 1
99     !                                     most significant w-r bits
100    ! Tempering parameters
101    integer, parameter :: TMASKB= -1658038656, TMASKC= -272236544
102
103    dimension mag01(0:1)
104    data mag01/0, MATA/
105    save mag01
106    !                                     mag01(x) = x * MATA for x=0,1
107
108    TSHFTU(y)=ishft(y,-11)
109    TSHFTS(y)=ishft(y,7)
110    TSHFTT(y)=ishft(y,15)
111    TSHFTL(y)=ishft(y,-18)
112
113    if(mti.ge.N) then
114    !                                     generate N words at one time
115        if(mti.eq.N+1) then
116    !                                     if sgrnd() has not been called,
117            call sgrnd( defaultsd )
118    !                                     a default initial seed is used
119        endif
120
121        do kk=0,N-M-1
122            y=ior(iand(mt(kk),UMASK),iand(mt(kk+1),LMASK))
123            mt(kk)=ieor(ieor(mt(kk+M),ishft(y,-1)),mag01(iand(y,1)))
124        enddo
125        do kk=N-M,N-2
126            y=ior(iand(mt(kk),UMASK),iand(mt(kk+1),LMASK))
127            mt(kk)=ieor(ieor(mt(kk+(M-N)),ishft(y,-1)),mag01(iand(y,1)))
128        enddo
129        y=ior(iand(mt(N-1),UMASK),iand(mt(0),LMASK))

```

```

130     mt(N-1)=ieor(ieor(mt(M-1),ishft(y,-1)),mag01(iand(y,1)))
131     mti = 0
132     endif
133
134     y=mt(mti)
135     mti = mti + 1
136     y=ieor(y,TSHFTU(y))
137     y=ieor(y,iand(TSHFTS(y),TMASKB))
138     y=ieor(y,iand(TSHFTT(y),TMASKC))
139     y=ieor(y,TSHFTL(y))
140
141     if(y .lt. 0) then
142         grnd=(dble(y)+2.0d0**32)/(2.0d0**32-1.0d0)
143     else
144         grnd=dble(y)/(2.0d0**32-1.0d0)
145     endif
146
147     return
148 end function grnd
149
150 !State saving subroutines.
151 ! Usage:  call mtsave( file_name, format_character )
152 !         or  call mtsave( unit_number, format_character )
153 ! where   format_character = 'u' or 'U' will save in unformatted form, otherwise
154 !         state information will be written in formatted form.
155 subroutine mtsavef( fname, forma )
156
157 !NOTE: This subroutine APPENDS to the end of the file "fname".
158
159     character(*), intent(in) :: fname
160     character, intent(in)    :: forma
161
162     select case (forma)
163     case('u','U')
164         open(unit=10,file=trim(fname),status='UNKNOWN',form='UNFORMATTED', &
165             position='APPEND')
166         write(10)mti
167         write(10)mt
168
169     case default
170         open(unit=10,file=trim(fname),status='UNKNOWN',form='FORMATTED', &
171             position='APPEND')
172         write(10,*)mti
173         write(10,*)mt
174
175     end select
176     close(10)
177
178     return
179 end subroutine mtsavef
180
181 subroutine mtsaveu( unum, forma )
182
183     integer, intent(in)    :: unum
184     character, intent(in)  :: forma
185
186     select case (forma)
187     case('u','U')
188         write(unum)mti
189         write(unum)mt
190
191     case default
192         write(unum,*)mti
193         write(unum,*)mt
194
195     end select
196
197     return
198 end subroutine mtsaveu
199
200 !State getting subroutines.

```

```

201 ! Usage:  call mtget( file_name, format_character )
202 !       or  call mtget( unit_number, format_character )
203 ! where   format_character = 'u' or 'U' will read in unformatted form, otherwise
204 !       state information will be read in formatted form.
205 subroutine mtgetf( fname, forma )
206
207     character(*), intent(in) :: fname
208     character,   intent(in)   :: forma
209
210     select case (forma)
211     case('u','U')
212         open(unit=10,file=trim(fname),status='OLD',form='UNFORMATTED')
213         read(10)mti
214         read(10)mt
215
216     case default
217         open(unit=10,file=trim(fname),status='OLD',form='FORMATTED')
218         read(10,*)mti
219         read(10,*)mt
220
221     end select
222     close(10)
223
224     return
225 end subroutine mtgetf
226
227 subroutine mtgetu( unum, forma )
228
229     integer, intent(in) :: unum
230     character, intent(in) :: forma
231
232     select case (forma)
233     case('u','U')
234         read(unum)mti
235         read(unum)mt
236
237     case default
238         read(unum,*)mti
239         read(unum,*)mt
240
241     end select
242
243     return
244 end subroutine mtgetu
245
246 end module mtmod

```

```

1 fc      = gfortran
2 cflags  = -O3
3 lflags  =
4 exec    = rng
5 objs    = mtfort90.o rng.o
6
7 all: $(exec)
8 $(exec): $(objs)
9         $(fc) $(lflags) $(objs) -o $@
10 %.o: %.f90
11         $(fc) -c $(cflags) $<
12 clean:
13         -rm $(exec) $(objs) *.mod

```

H.6 ex06

mtfort90.f90 is the same as in ex05.

```

1 program rng
2     use mtmod
3     implicit none

```

```

4
5     integer, parameter :: npts = 100
6     integer :: j, k, c
7
8     call system_clock(count=c)
9     call sgrnd(c)
10
11    open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
12    do j=0,npts-1
13        write (unit=11, fmt='(2F15.7)') dble(j), grnd()
14    enddo
15    close(unit=11)
16
17    call system("../tools/gc ax.txt ");
18 end program rng

```

```

1 fc      = gfortran
2 cflags   = -O3
3 lflags   =
4 exec     = plotgraph
5 objs     = mtfort90.o plotgraph.o
6
7 all: $(exec)
8 $(exec): $(objs)
9         $(fc) $(lflags) $(objs) -o $$
10 %.o: %.f90
11         $(fc) -c $(cflags) $<
12 clean:
13         -rm $(exec) $(objs) *.mod

```

H.7 ex07

```

1 ! ode.f90
2
3 program ode
4     implicit none
5     integer, parameter :: ndim = 2 ! number of equations
6     double precision, parameter :: ttime = 10.0d0 ! total time
7     double precision, parameter :: h = 0.1d0 ! step size
8
9     double precision, dimension(ndim) :: y = (/1.0d0, 0.0d0/)
10
11     integer :: npts, i
12     double precision :: t
13
14     npts = int(ttime/h)+1;
15
16     open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
17     do i = 1, npts
18         t = h*i
19         write(unit=11, fmt='(3E15.7)') t, y
20         call advanceStep(t, y)
21     end do
22     close(unit=11)
23
24     call system("../tools/gc ax.txt ");
25
26 contains
27
28     subroutine advanceStep(t, y)
29         double precision, intent(in) :: t
30         double precision, dimension(:), intent(inout) :: y
31         double precision, dimension(ndim) :: yn, k1, k2, k3, k4
32
33         call rhs(t, y, k1)
34         yn = y + h * k1/2.0d0
35
36         call rhs(t+h/2.0d0, yn, k2)

```

```

37     yn = y + h*k2/2.0d0
38
39     call rhs(t+h/2.0d0, yn, k3)
40     yn = y + h*k3
41
42     call rhs(t+h, yn, k4);
43     y = y + h*(k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/6.0d0
44 end subroutine advanceStep
45
46
47 subroutine rhs (tt, y, dy)
48     double precision :: tt
49     double precision, dimension(:), intent(in):: y
50     double precision, dimension(:), intent(out):: dy
51     dy(1) = y(2)
52     dy(2) = -y(1)
53 end subroutine rhs
54
55 end program ode

```

H.8 ex08

```

1  ! channel.f90
2
3  program channel
4      implicit none
5
6      integer, parameter :: ndim = 9 ! number of equations
7      double precision, parameter :: ttime = 50.0d0 ! total time
8      double precision, parameter :: h = 0.01d0 ! step size
9      integer, parameter :: npts = floor(ttime/h)+1;
10     double precision :: vold = -1000.0d0
11
12     double precision :: a11, a12, a13, b11, b12, b13
13     double precision :: a2, b2, a3, b3, a4, b4, a5, b5
14     double precision :: v
15
16     double precision :: t
17     double precision, dimension(ndim) :: y
18     integer :: i
19
20     y(1) = 1.0d0
21     y(2:ndim) = 0.0d0
22
23     open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
24     do i = 1, npts
25         t = h*i
26         write(unit=11, fmt='(6E15.7)') t, y(1), y(4), y(7:9)
27         call advanceStep(t, y)
28     end do
29     close(unit=11)
30
31     call system("../tools/gc ax.txt ");
32
33 !   call execute_command_line("../tools\\gc ax.txt ");
34
35 contains
36
37     subroutine advanceStep(tt, y)
38         double precision, intent(in) :: tt
39         double precision, dimension(:), intent(inout) :: y
40         double precision, dimension(ndim) :: yn, k1, k2, k3, k4
41
42         call rhs(tt, y, k1)
43         yn = y + h * k1/2.0d0
44
45         call rhs(tt+h/2.0d0, yn, k2)
46         yn = y + h*k2/2.0d0
47

```

```

48      call rhs(tt+h/2.0d0, yn, k3)
49      yn = y + h*k3
50
51      call rhs(tt+h, yn, k4);
52      y = y + h*(k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/6.0d0
53  end subroutine advanceStep
54
55  subroutine rhs (tt, y, dy)
56      double precision, intent(in) :: tt
57      double precision, dimension(:), intent(in):: y
58      double precision, dimension(:), intent(out):: dy
59
60
61      if(tt<20.0d0 .or. tt>= 30.0d0)then
62          v = -100.0d0
63      else
64          v = 0.0d0
65      end if
66      if (v .ne. vold ) then
67          a11 = 3.802d0/(0.1027d0*exp(-v/17.0d0) + 0.20d0*exp(-v/150.0d0))
68          a12 = 3.802d0/(0.1027d0*exp(-v/15.0d0) + 0.23d0*exp(-v/150.0d0))
69          a13 = 3.802d0/(0.1027d0*exp(-v/12.0d0) + 0.25d0*exp(-v/150.0d0))
70
71          b11 = 0.1917d0*exp(-v/20.3d0)
72          b12 = 0.20d0*exp(-(v-5.0d0)/20.3d0)
73          b13 = 0.22d0*exp(-(v-10.0d0)/20.3d0)
74
75          a3 = (3.7933d-7)*exp(-v/7.7d0)
76          b3 = 0.0084d0 + 0.00002d0*v
77
78          a2 = 9.178d0*exp(v/29.68d0)
79          b2 = (a13*a2*a3)/(b13*b3)
80
81          a4 = a2/100d0
82          b4 = a3
83
84          a5 = a2/(9.5d4)
85          b5 = a3/50.0d0
86          vold = v
87      end if
88
89      dy(1) = -(a11 + b3)*y(1) + b11*y(2) + a3*y(5)
90      dy(2) = -(b11 + b3 + a12)*y(2) + a11*y(1) + b12*y(3) + a3*y(6)
91      dy(3) = -(b12 + b3 + a13)*y(3) + a12*y(2) + b13*y(4) + a3*y(7)
92      dy(4) = -(b13 + a2)*y(4) + a13*y(3) + b2*y(7)
93      dy(5) = -(a11 + a3)*y(5) + b11*y(6) + b3*y(1)
94      dy(6) = -(b11 + a3 + a12)*y(6) + a11*y(5) + b12*y(7) + b3*y(2)
95      dy(7) = -(b12 + a3 + b2 + a4)*y(7) + a12*y(6) + b3*y(3) + a2*y(4) + b4*y(8)
96      dy(8) = -(b4 + a5)*y(8) + a4*y(7) + b5*y(9)
97      dy(9) = -b5*y(9) + a5*y(8)
98  end subroutine rhs
99
100 end program channel

```

H.9 ex09

```

1  ! cellpot.f90
2
3  program cellpot
4      implicit none
5
6      integer, parameter :: m = 6
7      double precision, parameter :: dt = 0.05d0
8      double precision, parameter :: totalTime = 600.0d0
9
10     double precision :: Iinj = -2000.0d0  !; %2000.0;  %(pA)
11     double precision :: Ton  = 50.0d0
12     double precision :: Toff = 300.0d0
13     double precision :: vinit = -63.27d0

```

```

14 double precision, parameter :: vtraub = -55.0d0 ! (mV)
15
16 double precision, dimension(m) :: y
17
18
19 double precision :: t
20 integer :: i, npts
21
22 npts = floor(totalTime/dt)+1
23 call setInitialValues(y)
24
25 open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
26 do i = 1, npts
27     t = dt*i
28     write(unit=11, fmt='(2E15.7)') t, y(1)
29     call advanceStep(t, y)
30 end do
31 close(unit=11)
32 call system("../tools/gc ax.txt ");
33
34 contains
35
36 subroutine advanceStep(tt, y)
37     double precision, intent(in) :: tt
38     double precision, dimension(:), intent(inout) :: y
39     double precision, dimension(m) :: yn, k1, k2, k3, k4
40
41     call rhs(tt, y, k1)
42     yn = y + dt * k1/2.0d0
43
44     call rhs(tt+dt/2.0d0, yn, k2)
45     yn = y + dt*k2/2.0d0
46
47     call rhs(tt+dt/2.0d0, yn, k3)
48     yn = y + dt*k3
49
50     call rhs(tt+dt, yn, k4);
51     y = y + dt*(k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/6.0d0
52 end subroutine advanceStep
53
54
55 subroutine get_Na_m(v, a, b)
56     double precision, intent(in) :: v
57     double precision, intent(out) :: a, b
58     double precision :: v2
59     v2 = v - vtraub
60     a = 0.32d0 * (13.0d0-v2) / ( exp((13.0d0-v2)/4.0d0) - 1.0d0)
61     b = 0.28d0 * (v2-40.0d0) / ( exp((v2-40.0d0)/5.0d0) - 1.0d0)
62 end subroutine get_Na_m
63
64 subroutine get_Na_h(v, a, b)
65     double precision, intent(in) :: v
66     double precision, intent(out) :: a, b
67     double precision :: v2
68     v2 = v - vtraub
69     a = 0.128d0 * exp((17.0d0-v2)/18.0d0)
70     b = 4.0d0/(1.0d0 + exp((40.0d0-v2)/5.0d0))
71 end subroutine get_Na_h
72
73 subroutine get_K_n(v, a, b)
74     double precision, intent(in) :: v
75     double precision, intent(out) :: a, b
76     double precision :: v2
77     v2 = v - vtraub;
78     a = 0.032d0 * (15.0d0-v2)/( exp((15.0d0-v2)/5.0d0) - 1.0d0)
79     b = 0.5d0 * exp((10.0d0-v2)/40.0d0);
80 end subroutine get_K_n
81
82 subroutine get_TCa_m(v, tau, inf)
83     double precision, intent(in) :: v
84     double precision, intent(out) :: tau, inf

```

```

85     double precision :: vshift=0, vs
86     vs = v + vshift
87     tau = (3.0d0 + 1.0d0 / ( exp((vs+25.0d0)/10.0d0) + exp(-(vs+100.0d0)/15.0d0)))
88     inf = 1.0d0 / ( 1.0d0 + exp(-(vs+50.0d0)/7.4d0))
89 end subroutine get_TCa_m
90
91 subroutine get_TCa_h(v, tau, inf)
92     double precision, intent(in) :: v
93     double precision, intent(out) :: tau, inf
94     double precision :: vshift=0, vs
95
96     vs = v + vshift
97     tau = ( 85.0d0 + 1.0d0 / ( exp((vs+46.0d0)/4.0d0) + exp(-(vs+405.0d0)/50.0d0) ) )
98     inf = 1.0d0 / ( 1.0d0 + exp((vs+78.0d0)/5.0d0) )
99 end subroutine get_TCa_h
100
101 subroutine setInitialValues(y)
102     double precision, intent(inout), dimension(:) :: y
103     double precision :: v, a, b, tau, inf
104     y(1) = vinit
105     v = y(1)
106     call get_Na_m(v, a, b)
107     y(2) = a/(a+b)
108
109     call get_Na_h(v, a, b)
110     y(3) = a/(a+b)
111
112     call get_K_n(v, a, b)
113     y(4) = a/(a+b)
114
115     call get_TCa_m(v, tau, inf)
116     y(5) = inf
117
118     call get_TCa_h(v, tau, inf)
119     y(6) = inf
120 end subroutine setInitialValues
121
122
123 subroutine rhs (tt, y, dy)
124     double precision, parameter :: Dm = 60.0d0      ! (micrometer) diameter of the cell
125     double precision, parameter :: Lm = 70.0d0      ! (micrometer) length of the cell
126
127     double precision, parameter :: dGleak = 0.002d0 ! (mS/cm^2)
128     double precision, parameter :: dGNa  = 0.05d0  ! 0.53; % (mS/cm^2)
129     double precision, parameter :: dGK   = 0.15d0  ! 0.13; % (mS/cm^2)
130     double precision, parameter :: dGTCa = 0.03d0  ! (mS/cm^2)
131     double precision, parameter :: fac = 10.0d0    ! conversion factor (mS/cm^2) -> (pS/micromet
132
133     double precision, parameter :: Am      = 3.14d0 * Dm * Lm ! (squre_micrometer) surface area
134     double precision, parameter :: Gleak   = Am * dGleak * fac
135     double precision, parameter :: Cmm     = 0.01d0 * Am
136     double precision, parameter :: InvCm   = 1.0d0/Cmm
137     double precision, parameter :: GNa     = Am * dGNa * fac
138     double precision, parameter :: GK      = Am * dGK * fac
139     double precision, parameter :: GTCa    = Am * dGTCa * fac
140
141     double precision, parameter :: Eleak   = -65.0d0 ! (mV)
142     double precision, parameter :: ENa     = 50.0d0 ! (mV)
143     double precision, parameter :: EK      = -100.0d0 ! (mV)
144     double precision, parameter :: ECa     = 50.0d0 ! (mV)
145
146     double precision, intent(in) :: tt
147     double precision, dimension(:), intent(in):: y
148     double precision, dimension(:), intent(out):: dy
149     double precision :: v, c
150     double precision :: a, b, tau, inf
151     double precision :: INa, IK, ITCa
152
153     if(tt<Ton .or. t>=Toff)then
154         c = 0.0d0
155     else

```

```

156         c = Iinj
157     end if
158
159     v = y(1)
160
161     INa = GNa * y(2)**3 * y(3) * (v-ENa)
162     IK  = GK * y(4)**4 * (v-EK)
163     ITCa = GTCa * y(5)**2 * y(6) * (v - ECa)
164
165     dy(1) = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa )
166     call get_Na_m(v, a, b)
167     dy(2) = a - (a+b) * y(2)
168
169     call get_Na_h(v, a, b)
170     dy(3) = a - (a+b) * y(3)
171
172     call get_K_n(v, a, b)
173     dy(4) = a - (a+b) * y(4)
174
175     call get_TCa_m(v, tau, inf)
176     dy(5) = (inf - y(5)) / tau
177
178     call get_TCa_h(v, tau, inf)
179     dy(6) = (inf - y(6)) / tau
180 end subroutine rhs
181
182 end program cellpot

```

H.10 ex10

mtfort90.f90 is the same as in ex05.

```

1  ! release.m
2  ! Stochastic neurotransmitter release: ex12.m
3  ! Ralf Schneggenburger & Erwin Neher
4  ! Nature 406, 889-893, 2000}
5
6  program release
7      use mtmod
8      implicit none
9
10     double precision, parameter :: ttime = 4.0d0    ! total time
11     double precision, parameter :: dt    = 0.02d0   ! time step
12     double precision, parameter :: peak  = 20.0d0   ! peak calcium concentration
13     double precision, parameter :: hw    = 1.0d0    ! half width of calcium transient
14     integer, parameter :: iter = 100000             ! number of trials
15
16     ! kinetic constants
17     double precision, parameter :: kon      = 9.0d-2 ! microM-1 ms-1
18     double precision, parameter :: koff     = 9.5d0  ! ms-1
19     double precision, parameter :: beta     = 0.25d0 ! cooperativity
20     double precision, parameter :: gamma    = 6.0d0  ! ms-1
21     double precision, parameter :: epsilon = 0.3d0   ! ms-1
22     integer, parameter :: initialState = 1
23
24     double precision, parameter :: konh     = dt * kon;
25     double precision, parameter :: koffh    = dt * koff;
26     double precision, parameter :: gammah   = dt * gamma;
27     double precision, parameter :: epsilonh = dt*epsilon;
28     double precision, parameter :: beta2    = beta*beta;
29     double precision, parameter :: beta3    = beta*beta2;
30     double precision, parameter :: beta4    = beta*beta3;
31
32     integer, parameter :: npts = floor(ttime/dt)+1
33     integer, dimension(npts) :: sum = 0
34     double precision, dimension(npts) :: ca = 0.0d0
35     integer :: i, k, counter, s, snext, seed
36     double precision :: d, alpha, r, th
37

```

```

38     write(*, fmt='(I10, " iterations")') iter
39
40 ! ca transient
41 d = 2.0d0*sqrt((1.0d0 - sqrt(0.5d0)))/hw
42 k = int(2.0d0/(dt*d))
43 do i=1,k
44     ca(i) = peak * (((d*dt*(i-1) - 1.0d0)**2)-1.0)**2
45 end do
46
47 call system_clock(count=seed)
48 call sgrnd(seed)
49
50 counter = 0
51 do k = 1,iter
52     s = initialState ! initial state
53     do i = 1,npts
54         r = grnd() ! get a random number [0, 1]
55         alpha = konh * ca(i) ! get Ca level
56         snext = s
57
58         select case (s)
59             case (1)
60                 th = 5.0 * alpha
61                 if(r<th)then
62                     snext = 2
63                 end if
64             case (2)
65                 th = 4.0 * alpha
66                 if(r < th)then
67                     snext = 3
68                 else
69                     th = th + koffh
70                     if(r<th)then
71                         snext = 1
72                     end if
73                 end if
74             case (3)
75                 th = 3.0 * alpha
76                 if(r < th)then
77                     snext = 4
78                 else
79                     th = th + koffh * beta
80                     if(r < th)then
81                         snext = 2
82                     end if
83                 end if
84             case (4)
85                 th = 2.0 * alpha
86                 if (r < th)then
87                     snext = 5
88                 else
89                     th = th + koffh * beta2;
90                     if(r < th)then
91                         snext = 3
92                     end if
93                 end if
94             case (5)
95                 th = alpha
96                 if (r < th)then
97                     snext = 6
98                 else
99                     th = th + koffh * beta3
100                    if (r < th)then
101                        snext = 4
102                    end if
103                end if
104             case (6)
105                 th = gammah
106                 if(r < th)then
107                     snext = 7
108             else

```

```

109         th = th + koffh * beta4
110         if(r < th)then
111             snext = 5
112         end if
113     end if
114     case (7 )      ! release
115         snext = 0
116         sum(i) = sum(i) + 1
117         counter = counter + 1
118     case (0)      !recovery from release
119         th = epsilonh
120         if(r < th)then
121             snext = 1
122         end if
123     end select
124     s = snext
125 end do ! end of i-loop
126 end do ! end of k-loop
127
128 write(*, fmt='("release probability = ", F10.5)') dble(counter)/dble(iter)
129
130
131 open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
132 do i = 1, npts
133     write(unit=11, fmt='(3E15.7)') dt*i, ca(i), dble(sum(i))
134 end do
135 close(unit=11)
136 call system("../tools/gc ax.txt ");
137
138 end program release

```

```

1 fc      = gfortran
2 cflags  = -O3
3 lflags  =
4 exec = release
5 objs = mtfort90.o release.o
6
7 all: $(exec)
8 $(exec): $(objs)
9         $(fc) $(lflags) $(objs) -o $@
10 %.o: %.f90
11         $(fc) -c $(cflags) $<
12 clean:
13         -rm $(exec) $(objs) *.mod

```

I How to use C++ examples in Visual Studio IDE (Windows)

The following instruction shows how to run `ex01` using Visual Studio IDE.

- The folder of TM2017 is supposed to be on the **Desktop**.
- Your user name is `xyz`.

1. Start Visual Studio

2. Create a project

(Main Menu) File → New → Project

- Visual C++, Win32 Console Application
- Name: `ex01`
- Location: `C:\Users\xyz\Desktop\TM2017\CPP\`
- Create directory for solution: uncheck
- Add to Source Control: uncheck

[OK]

3. Wizard Window

[Next]

4. Wizard Window : Application Setting

- Console application: check
- Precompiled Header: no check
- Empty project: check

[Finish]

5. Solution Explorer (Left pane)

Mouse right click on **Source Files** → Add → Existing Item ...

Select `.cpp` files (and `.h` files)

[Add]

6. (Main Menu) Build → Build `ex01`

7. (Main Menu) Debug → Start Debugging or Start Without Debugging

- Just click the `ex01.vcsproj` to open the `ex01` project next time.
- The default configuration is **Debug** and **x86** (32-bit), as indicated below the Main Menu. You can select **Release** and **x64** (64-bit).