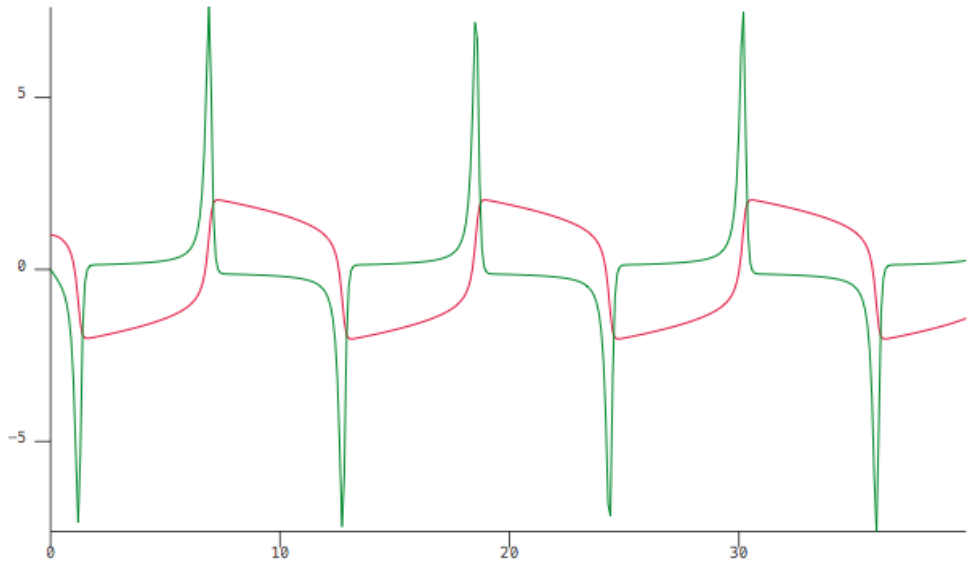
A Crash Course of Scientific Programming

Keiji Imoto

18/25-Oct-2017

revision 04-Oct-2017
revision 12-Mar-2017



Contents

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 /Fe`*program_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

[1] Of 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 your 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 Python[6].
   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

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

- 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

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

- `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

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

- `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

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

- 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

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

```
# makefile for rng
cxx     = clang++
cflags  = -O3
ldflags =
exec    = rng
objs    = UniRand.o main.o

all: $(exec)

$(exec): $(objs)
        $(cxx) $(ldflags) $(objs) -o $@

%.o: %.cpp
        $(cxx) -c $(cflags) $<

clean:
        -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 porgrams.

- `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)

```
// main.cpp of plotgraph
#include <iostream>
#include <fstream>
#include "UniRand.h"
int main(int argc, char** argv){
    double dt = 0.1;
    double ttime = 10.0;
    int npts = int(ttime/dt) + 1;
    UniRand urng;
    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
    for(int i=0;i<npts;i++){
        double a = urng.rand();
        fd << dt*i << " " << a << std::endl;
    }
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt ");
#else
    system("../../tools/gc ax.txt ");
#endif
    return 0;
}
```

- 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.

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

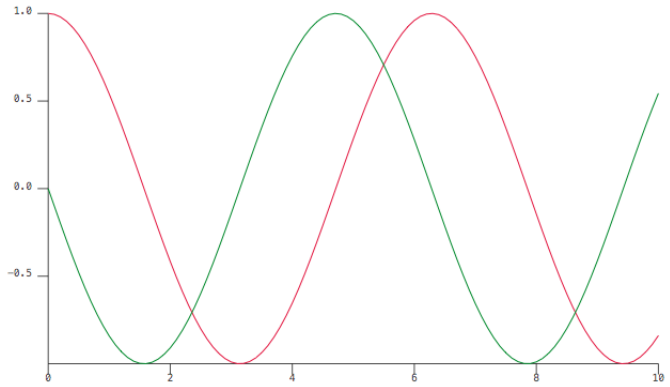
If y and z are components of an array $y[\,]$, $y[0]$ and $y[1]$, respectively, the equations are written,

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

```
//  main.cpp of ode (ex07)
#include <iostream>
#include <fstream>

#include "RungeKutta4.h"

void rhs(double tt, vector<double> y, vector<double>& dy){
    dy[0] =  y[1];
    dy[1] = -y[0];
}

int main(int argc, char** argv){
    int ndim = 2;                   // number of equations
    double ttime = 10.0;            // total time
    double dt = 0.1;                // step size
    int npts = (int)(ttime/dt)+1; // number of points
    vector<double> y={1.0, 0.0};  // initial values

    RungeKutta4 rk(ndim, dt);

    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
```

```
    for(int i=0;i<npts;i++){
        double t = dt*i;
        fd << t << " " << y[0] << " " << y[1] << std::endl;
        rk.advanceStep(rhs, t, y);
    }
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt ");
#else
    system("../../tools/gc ax.txt ");
#endif
    return 0;
}
```

- 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.

```
//  main.cpp of ode (ex07a)
#include <iostream>
#include <fstream>
#include "Eqs.h"

int main(int argc, char** argv){
    int ndim = 2;       // number of equations
    double ttime = 10.0;       // total time
    double dt = 0.1;    // step size
    int npts = (int)(ttime/dt)+1;
    vector<double>  y={1.0, 0.0};

    Eqs ce;
    ce.initRK4(ndim, dt);

    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
    for(int i=0;i<npts;i++){
        double t = dt*i;
        fd << t << " " << y[0] << " " << y[1] << std::endl;
        ce.advanceStep(t, y);
    }
```

```
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt ");
#else
    system("../../tools/gc ax.txt ");
#endif
    return 0;
}
```

```
// Eqs.h
#ifndef __included_eqs_h__
#define __included_eqs_h__

#include <vector>
#include "RungeKutta4.h"

using std::vector;

class Eqs : public RungeKutta4
{
public:
    void rhs(double tt, vector<double> y, vector<double>& dy);
private:
};
#endif
```

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

```
# makefile for ode (ex07f)
cxx     = clang++
cflags  = -std=c++0x -O3
ldflags =
exec    = ode
objs    = RungeKutta4.o Eqs.o main.o

all: $(exec)
$(exec): $(objs)
        $(cxx) $(ldflags) $(objs) -o $@
%.o: %.cpp
        $(cxx) -c $(cflags) $<
clean:
        -rm $(exec) $(objs)
```

Class `Eqs` is defined as a subclass of `RungeKutta4`.

7.3 ex07b: Passing command line parameters

```
//  main.cpp of ode (ex07b)
#include <iostream>
#include <fstream>
#include <string>
#include "Eqs.h"

int main(int argc, char** argv){
    int ndim = 2;                   // number of equations
    double ttime = 100.0;           // total time
    double dt = 0.1;                // step size
    int npts = (int)(ttime/dt)+1;
    vector<double>  y={1.0, 0.0};
    double a = 1.0;                 // default parameter
    double b = 0.1;                 // default parameter
    if(argc >= 3){
```

```
        a = std::stof(std::string(argv[1]));
        b = std::stof(std::string(argv[2]));
        if(argc == 5){
            y[0] = std::stof(std::string(argv[3]));
            y[1] = std::stof(std::string(argv[4]));
        }
    }
    Eqs ce;
    ce.setParams(a,b);
    ce.initRK4(ndim, dt);

    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
    for(int i=0;i<npts;i++){
        double t = dt*i;
        fd << t << " " << y[0] << " " << y[1] << std::endl;
        ce.advanceStep(t, y);
    }
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt ");
#else
    system("../../tools/gc ax.txt ");
#endif
    return 0;
}
```

```
// Eqs.h
#ifndef __included_eqs_h__
#define __included_eqs_h__

#include <vector>
#include "RungeKutta4.h"

using std::vector;

class Eqs : public RungeKutta4 {
public:
    void setParams(double a, double b);
    void rhs(double tt, vector<double> y, vector<double>& dy);
private:
    double f;
    double d;
};
#endif
```

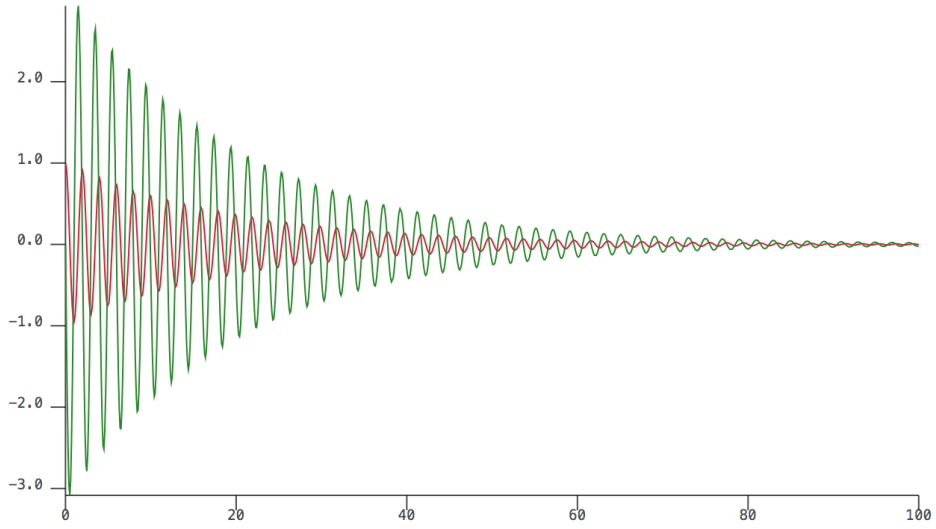
```
// Eqs.cpp
#include "Eqs.h"
void Eqs::setParams(double a, double b){
    f = a;
    d = b;
}
void Eqs::rhs(double tt, vector<double> y, vector<double>& dy){
    dy[0] =  y[1];
    dy[1] = -f*y[0] - d*y[1];
}
```

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.



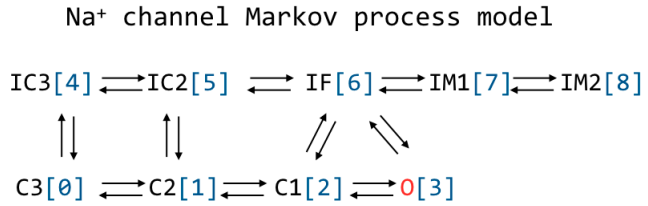
7.4 Exercise: Van der Pol equation

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

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

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.



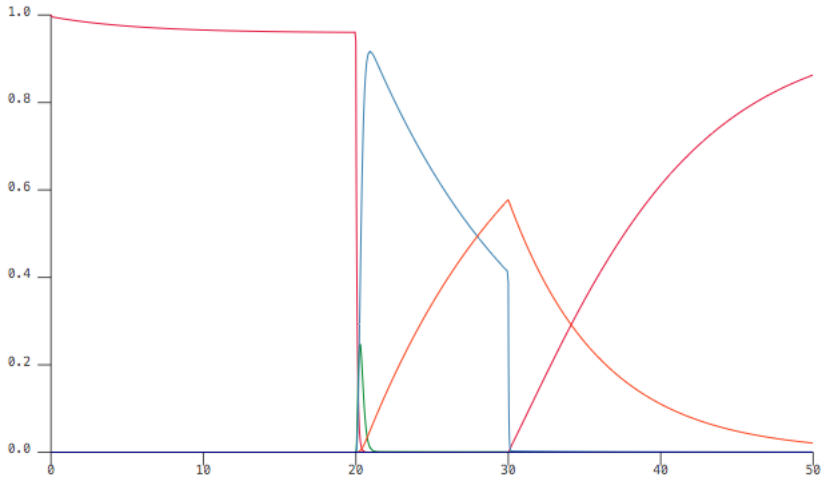

```cpp
// main.cpp of channel
#include <iostream>
#include <fstream>
#include <cmath>
#include "Markov.h"

int main(){
    const int ndim = NUM_OF_NA_STATES;
    const double dt = 0.01;
    const double totalTime = 50.0;
    const int npts = (int)(totalTime/dt)+1;

    std::vector<double> y(ndim);

    Markov ce;
    ce.initRK4(ndim, dt);
    ce.setInitialValues(y);

    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
    for(int i=0; i<npts; i++){
        double t = dt*i;
        fd << t << " " << y[0] << " " << y[3]<< " " << y[6]<< " "
                   << y[7] << " " << y[8]<<std::endl;
        ce.advanceStep(t, y);
    }
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt &");
#else
    system("../../tools/gc ax.txt &");
#endif
    return 0;
}
```

```cpp
// markov.h
#ifndef __included__markov_h__
#define __included__markov_h__

#include <vector>
#include "RungeKutta4.h"

#define NUM_OF_NA_STATES  9
#define NA_OPEN_STATE 3

using std::vector;

class Markov : public RungeKutta4 {
public:
    void setInitialValues(vector<double>& y);
```

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

```
// markov.cpp
#include <cmath>
#include "Markov.h"

void Markov::setInitialValues(vector<double>& y){
    y[0] = 1.0;
    for(int i = 1 ; i < NUM_OF_NA_STATES ; i++){
        y[i] = 0.0;
    }
    vold = -1000.0;
};
void Markov::rhs(double t, vector<double> y, vector<double>& dy){
    double v = t<20.0 || t>= 30.0 ? -100.0 : 0.0;
    if(v != vold){
        a11 = 3.802/(0.1027*exp(-v/17.0) + 0.20*exp(-v/150));
        a12 = 3.802/(0.1027*exp(-v/15.0) + 0.23*exp(-v/150));
        a13 = 3.802/(0.1027*exp(-v/12.0) + 0.25*exp(-v/150));

        b11 = 0.1917*exp(-v/20.3);
        b12 = 0.20*exp(-(v-5)/20.3);
        b13 = 0.22*exp(-(v-10)/20.3);

        a3 = (3.7933e-7)*exp(-v/7.7);
        b3 = 0.0084 + 0.00002*v;

        a2 = 9.178*exp(v/29.68);
        b2 = (a13*a2*a3)/(b13*b3);

        a4 = a2/100;
        b4 = a3;

        a5 = a2/(9.5e4);
        b5 = a3/50;
        vold = v;
    }
    dy[0] = -(a11 + b3)*y[0] + b11*y[1] + a3*y[4];
    dy[1] = -(b11 + b3 + a12)*y[1] + a11*y[0] + b12*y[2] + a3*y[5];
    dy[2] = -(b12 + b3 + a13)*y[2] + a12*y[1] + b13*y[3] + a3*y[6];
    dy[3] = -(b13 + a2)*y[3] + a13*y[2] + b2*y[6];
    dy[4] = -(a11 + a3)*y[4] + b11*y[5] + b3*y[0];
    dy[5] = -(b11 + a3 + a12)*y[5] + a11*y[4] + b12*y[6] + b3*y[1];
    dy[6] = -(b12 + a3 + b2 + a4)*y[6] + a12*y[5] + b3*y[2] + a2*y[3] + b4*y[7];
    dy[7] = -(b4 + a5)*y[7] + a4*y[6] + b5*y[8];
    dy[8] = -b5*y[8] + a5*y[7];
}
```

- 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

```cpp
// main.cpp of cellpot
#include <iostream>
#include <fstream>
#include "Cell.h"

int main(){
    const int m = 6;                // number of equations
    const double dt = 0.05;         // step size
    const double totalTime = 600.0; // simulation time
    const int npts = (int)(totalTime/dt)+1;
    vector<double> y(m);
    vector<double> vol(npts);

    double vinit = -63.27;  // initial voltate
    double Iinj  = -2000.0; // current injection(pA)
    double Ton   = 50.0;    // on-time
    double Toff  = 300.0;   // off-time

    Cell ce;
    ce.initRK4(m, dt);
    ce.setInitialValues(vinit, y);
    ce.setPulse(Ton, Toff, Iinj);

    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
    for(int i=0; i<npts; i++){
        double t = dt*i;
        vol[i] = y[0];
        fd << t << " " << y[0] << std::endl;
        ce.advanceStep(t, y);
    }
    // count the number of spikes
    int sn=0;
    double th = -20.0;
    for(int i=0;i<npts-1;i++){
        if(vol[i+1]>=th && vol[i] < th) sn++;
    }
    std::cout << "number of spikes = " << sn << std::endl;
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt &");
#else
    system("../../tools/gc ax.txt &");
#endif
    return 0;
}
```

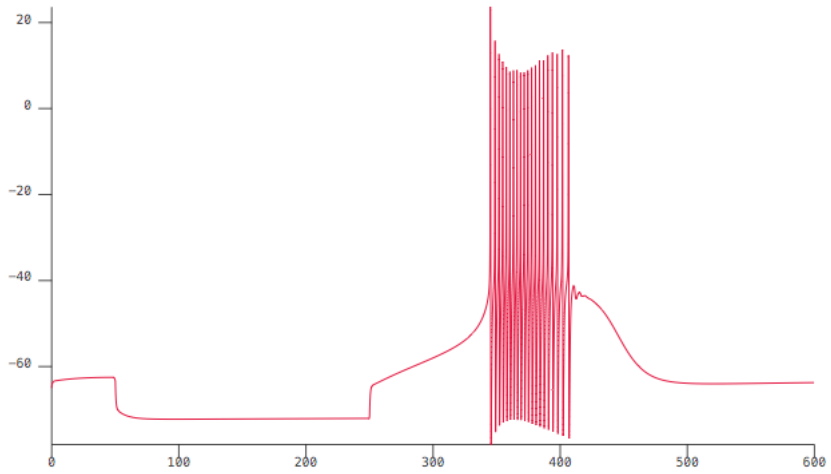
```cpp
// cell.h
#ifndef __ode__Cell__
#define __ode__Cell__

#include <vector>
#include "RungeKutta4.h"

using std::vector;
class Cell : public RungeKutta4
{
public:
    Cell(){};
    ~Cell(){};
    void setInitialValues(double v_init, vector<double>& y);
    void setPulse(double ton, double toff, double current);
    void rhs(double t, vector<double> y, vector<double>& yt);
private:
    static constexpr double Dm = 60.0;  // (micrometer) diameter of the cell
    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;    // (squre_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;
```

```
}
void Cell::setPulse(double ton, double toff, double current){
    Ton  = ton;
    Toff = toff;
    Iinj = current;
}
void Cell::get_Na_m(double v, double* a, double* b){
    double v2 = v-vtraub;
    *a = 0.32*(13.0-v2)/(exp((13.0-v2)/4.0)-1.0);
    *b = 0.28*(v2-40.0)/(exp((v2-40.0)/5.0)-1.0);
}
void Cell::get_Na_h(double v, double* a, double* b){
    double v2 = v-vtraub;
    *a = 0.128 * exp((17.0-v2)/18.0);
    *b = 4.0/(1.0 + exp((40.0-v2)/5.0));
}
void Cell::get_K_n(double v, double* a, double* b){
    double v2 = v - vtraub;
    *a = 0.032*(15.0-v2)/(exp((15.0-v2)/5.0)-1.0);
    *b = 0.5*exp((10.0-v2)/40.0);
}
void Cell::get_TCa_m(double v, double* tau, double* inf){
    double vshift=0.0;
    double vs = v + vshift;
    *tau = (3.0+1.0/(exp((vs+25.0)/10.0)+exp(-(vs+100.0)/15.0)));
    *inf = 1.0/(1.0+exp(-(vs+50.0)/7.4));
}
void Cell::get_TCa_h(double v, double* tau, double* inf){
    double vshift=0.0;
    double vs = v + vshift;
    *tau = (85.0+1.0/(exp((vs+46.0)/4.0)+exp(-(vs+405.0)/50.0))) ;
    *inf = 1.0/(1.0+exp((vs+78.0)/5.0));
}

void Cell::rhs(double tt, std::vector<double> y, std::vector<double>& dy){
    double INa, IK, ITCa;
    double a, b, tau, inf;
    double shift = 0.0;
    double v = y[0];
    double c = (tt<Ton || tt>Toff)? 0.0 : Iinj;

    INa = GNa * pow(y[1],3) * y[2]  * (v-ENa); // INa = GNa * m^3 * h * (v-Ena)
    IK = GK * pow(y[3],4) * (v-EK);            // IK = GK * n^4 (v-EK)
    ITCa = GTCa * pow(y[4],2) * y[5] * (v - ECa);
    dy[0] = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa);
    // Na channel
    get_Na_m(v, &a, &b);
    dy[1] = a - (a+b)*y[1];
    get_Na_h(v, &a, &b);
    dy[2] = a - (a+b)*y[2];
    // K channel (delayed rectifier)
    get_K_n(v, &a, &b);
    dy[3] = a - (a+b)*y[3];
    // T-type Ca channel
    get_TCa_m(v, &tau, &inf);
    dy[4] = (inf - y[4]) / tau;
    get_TCa_h(v, &tau, &inf);
    dy[5] = (inf - y[5]) / tau;
}
```

- 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).

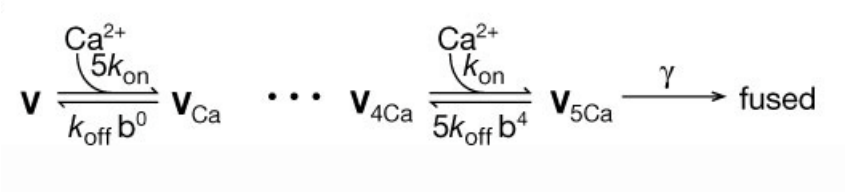


9.2 Exercise: A pharmacological experiment

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

10 ex10: Transmitter release: *release*

Neurotransimitter 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.

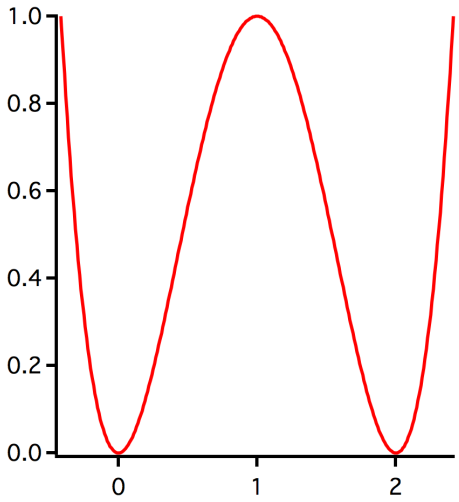


The following parameters were used: k on, 9×10^7 M^{-1} s^{-1};
koff, 9,500 s^{-1}; gamma, 6,000 s^{-1}; b = 0.25.

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

```
double kon   =  9.0e-2; // μM^-1 ms^-1
double koff  =  9.5;    // ms^-1
double beta  =  0.25;   // cooperativity
double gamma = 6.0;     // 6.0d0  ! ms^-1
double epsilon = 0.3;   // 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 facter d is determined so that the half-width of the transient is equal to hw.

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

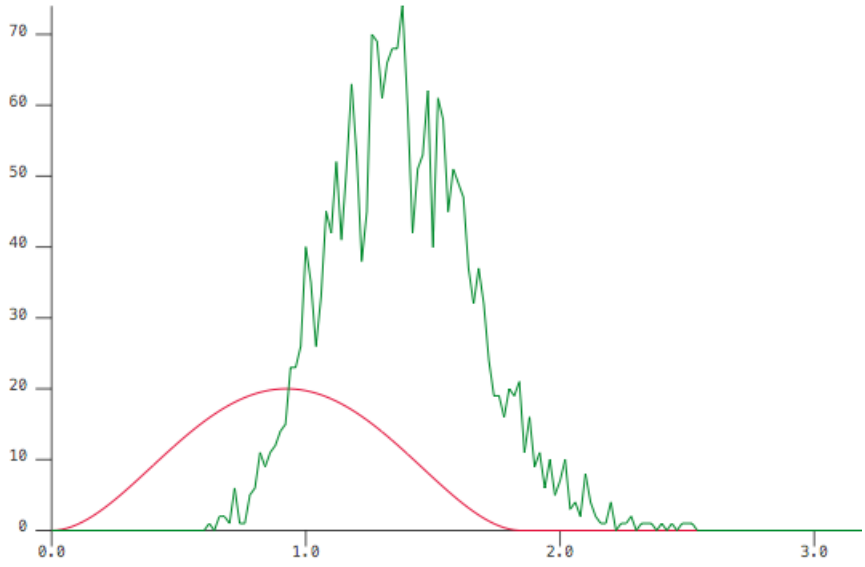
The peak Ca^{2+} concentration, *peak*, is 20 μM. The half width, *hw*, is 1 ms.

[7] Intracellular 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.

```
// release.cpp
// Stochastic neurotransmitter release
// Ralf Schneggenburger & Erwin Neher
// Nature 406, 889-893, 2000.
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include "UniRand.h"
int main(int argc, const char * argv[]){
    const double ttime = 4.0;
    const double dt=0.02;
    const double peak = 20.0;
    const double hw = 1.0;
    const int iter = 100000;

    const double kon   =  9.0e-2; // microM^{-1} ms^{-1}
    const double koff  =  9.5;    // ms^{-1}
    const double beta  =  0.25;   // cooperativity
    const double gamma = 6.0;     // 6.0d0  ! ms^{-1}
    const double epsilon = 0.3;   // ms^{-1}
    const int initialState = 1;

    std::cout << iter << " iterations" << std::endl;

    int npts = (int)(ttime/dt);
    int counter = 0;
    std::vector<double> ca(npts, 0.0);
    std::vector<int> sum(npts,0);

    UniRand urng; // Uniform random number generator

    // ca transient
    double d = 2.0*sqrt((1.0 - sqrt(0.5)))/hw;
    for(int i=0;i<(int)(2.0/(dt*d));i++){
        ca[i] = peak * pow(pow(d*dt*i - 1.0, 2.0)-1.0, 2.0);
    }

    double konh   = dt * kon;
    double koffh  = dt * koff;
    double gammah = dt * gamma;
    double epsilonh = dt*epsilon;
    double beta2 = beta*beta;
    double beta3 = beta*beta2;
    double beta4 = beta*beta3;

    for(int k=0;k<iter;k++){    // iteration loop
        int s  = initialState;  //initial state
        for(int i=0;i<npts;i++){
            double th;
            double r = urng.rand();                    // get a random number [0, 1]
            double alpha = konh * ca[i];  // get Ca level
            int snext = s;
            switch (s){
                case (1):
                    th = 5.0 * alpha;
                    if(r<th) snext = 2;
                    break;
                case (2):
                    th = 4.0 * alpha;
                    if(r<th) {
                        snext = 3;
                    }else{
                        th = th + koffh;
                        if(r<th) snext = 1;
                    }
                    break;
                case (3):
```

```
                    th = 3.0 * alpha;
                    if(r<th) {
                        snext = 4;
                    }else{
                        th = th + koffh * beta;
                        if(r<th) snext = 2;
                    }
                    break;
                case (4):
                    th = 2.0 * alpha;
                    if(r<th){
                        snext = 5;
                    }else{
                        th = th + koffh * beta2;
                        if(r<th) snext = 3;
                    }
                    break;
                case (5):
                    th = alpha;
                    if(r<th){
                        snext = 6;
                    }else{
                        th = th + koffh * beta3;
                        if(r<th) snext = 4 ;
                    }
                    break;
                case (6):
                    th = gammah;
                    if(r<th){
                        snext = 7;
                    }else{
                        th = th + koffh * beta4;
                        if(r<th) snext = 5;
                    }
                    break;
                case (7):
                    snext = 0;     //release
                    sum[i] += 1;  // release
                    counter++;
                    break;
                case (0):          //recovery from release
                    th = epsilonh;
                    if(r<th) snext = 1;
                    break;
                default:
                    ;
            } // end of switch
            s = snext;
        } //end of i-loop
    } // end of k-loop

    std::cout << "release probability = " << double(counter)/(double)iter << std::endl;;
    std::fstream fd("ax.txt", std::ios::out | std::ios::trunc);
    for(int i=0;i<npts;i++){
        fd << dt*i << " " << ca[i] << " " << (double)sum[i] << std::endl;
    }
    fd.close();
#if defined(_WIN32)
    system("..\\..\\tools\\gc ax.txt  &");
#else
    system("../../tools/gc ax.txt  &");
#endif
    return 0;
}
```


70
60
50
40
30
20
10
0
0.0
1.0
2.0
3.0

A Uniform distribution random number generator

```cpp
// UniRand.h
#ifndef INCLUDED_UNIRAND_H_
#define INCLUDED_UNIRAND_H_
#include <random>
class UniRand
{
public:
    UniRand();       // constructor
    UniRand(double a, double b); // constructor
    ~UniRand(){};    // destructor
    double rand();
private:
    std::mt19937 engine; // Mersenne Twister
    std::uniform_real_distribution<double> distribution;
};
#endif
```

```cpp
// UniRand.cpp
#include "UniRand.h"
UniRand::UniRand():distribution(0.0, 1.0){
    std::random_device seed;
    engine.seed(seed());
}
UniRand::UniRand(double a, double b):distribution(a, b){
    std::random_device seed;
    engine.seed(seed());
}
double UniRand::rand(){
    return distribution(engine);
}
```

B Normal distribution random number generator

```cpp
// NormalRand.h
#ifndef INCLUDED_NormalRandRAND_H_
#define INCLUDED_NormalRandRAND_H_
#include <random>
class NormalRand
{
public:
    NormalRand();       // constructor
    NormalRand(double mu, double sigma); // constructor
    ~NormalRand(){};    // destructor
    double rand();
private:
    std::mt19937 engine; // Mersenne Twister
    std::normal_distribution<double> distribution;
};
#endif
```

```cpp
// NormalRand.cpp
#include "NormalRand.h"
NormalRand::NormalRand():distribution(0.0, 1.0){
    std::random_device seed;
    engine.seed(seed());
}
NormalRand::NormalRand(double mu, double sigma):distribution(mu, sigma){
    std::random_device seed;
    engine.seed(seed());
}
double NormalRand::rand(){
    return distribution(engine);
}
```

C Runge-Kutta method (1)

```
//  RungeKutta4.h

#ifndef __include_rungekutta4_h__
#define __include_rungekutta4_h__


#include <vector>
using std::vector;

typedef void (*RHSFunc)(double tt, vector<double> y, vector<double>& dy);


class RungeKutta4 {
public:
    RungeKutta4(int n, double dt);
    void advanceStep(RHSFunc rhs, double t, vector<double>& y);
private:
    int n;
    double h;
    vector<double> yn;
    vector<double> k1;
    vector<double> k2;
    vector<double> k3;
    vector<double> k4;
};
#endif
```

```
//  RungeKutta4.cpp
#include "RungeKutta4.h"
RungeKutta4::RungeKutta4(int ndim, double dt){
    n = ndim;
    h = dt;
    yn =  vector<double>(n);
    k1 =  vector<double>(n);
    k2 =  vector<double>(n);
    k3 =  vector<double>(n);
    k4 =  vector<double>(n);
}
void RungeKutta4::advanceStep(RHSFunc rhs, double t, vector<double>& y){
    rhs(t, y, k1);
    for(int i=0;i<n;i++) yn[i] = y[i] + h*k1[i]/2;
    rhs(t+h/2, yn, k2);
    for(int i=0;i<n;i++) yn[i] = y[i] + h*k2[i]/2;
    rhs(t+h/2, yn, k3);
    for(int i=0;i<n;i++) yn[i] = y[i] + h*k3[i];
    rhs(t+h, yn, k4);
    for(int i=0;i<n;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
}
```

D Runge-Kutta method (2)

```
//  RungeKutta4.
#ifndef __include_rungekutta4_h__
#define __include_rungekutta4_h__

#include <vector>

using std::vector;

class RungeKutta4{
public:
    void initRK4(int n, double dt);
    void advanceStep(double t, vector<double>& y);
    virtual void rhs(double tt, vector<double> y, vector<double>& dy){};
```

```
private:
    int n;
    double h;
    vector<double> yn;
    vector<double> k1;
    vector<double> k2;
    vector<double> k3;
    vector<double> k4;
};
#endif
```

```
//  RungeKutta4.cpp
#include "RungeKutta4.h"

void RungeKutta4::initRK4(int ndim, double dt){
    n = ndim;
    h = dt;
    yn =  vector<double>(n);
    k1 =  vector<double>(n);
    k2 =  vector<double>(n);
    k3 =  vector<double>(n);
    k4 =  vector<double>(n);
}

void RungeKutta4::advanceStep(double t, vector<double>& y){
    rhs(t, y, k1);
    for(int j=0; j<n; j++) yn[j] = y[j] + h*k1[j]/2.0;
    rhs(t+h/2.0, yn, k2);
    for(int j=0; j<n; j++) yn[j] = y[j] + h*k2[j]/2.0;
    rhs(t+h/2.0, yn, k3);
    for(int j=0; j<n; j++) yn[j] = y[j] + h*k3[j];
    rhs(t+h, yn, k4);
    for(int j=0; j<n; j++) y[j] += h*(k1[j] + 2*k2[j] + 2*k3[j] + k4[j])/6.0;
}
```

E Matlab

E.1 ex01

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

E.2 ex02

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

E.3 ex03

```
% ifelse.m
for i = 0:9
    if mod(i,2)==0
        fprintf('%d even\n', i)
    else
        fprintf('%d odd\n', i)
    end
end
```

E.4 ex04

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

E.5 ex05

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

E.6 ex06

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

E.7 ex07

```
%  ode.m
function ode
    ttime = 10.0;
    dt = 0.1;
    npts = int32(ttime/dt) + 1;
    tt = linspace(0.0, ttime, npts);
    [t,y] = ode45(@rhs, tt, [1.0 0.0]);
    plot(t, y(:,1), t, y(:,2))
end

function dydt = rhs(t, y)
    dydt = [y(2); -y(1)];
end
```

E.8 ex08

```
% channel.m
function channel()
    dt = 0.01;
    ttime = 50.0;
    npts = int32(ttime/dt) + 1;
    tt = linspace(0, ttime, npts);
    y0 = zeros(1,9);
    y0(1) = 1.0;

    vx  = -1000.0;
    [a11, a12, a13, b11, b12, b13] = deal(0.0);
    [a2, b2, a3, b3, a4, b4, a5, b5] = deal(0.0);

    [t,y] = ode45(@(tt, yy)RHS(tt, yy), tt, y0);
    plot(t,y(:,1), t,y(:,4), t,y(:,7), t,y(:,8),t,y(:,9))

    function dy = RHS(t, y)
        if t<20.0 || t>= 30.0
            v = -100.0;
        else
            v = 0.0;
        end
        if v ~= vx
            a11 = 3.802/(0.1027*exp(-v/17.0) + 0.20*exp(-v/150));
            a12 = 3.802/(0.1027*exp(-v/15.0) + 0.23*exp(-v/150));
            a13 = 3.802/(0.1027*exp(-v/12.0) + 0.25*exp(-v/150));

            b11 = 0.1917*exp(-v/20.3);
            b12 = 0.20*exp(-(v-5)/20.3);
            b13 = 0.22*exp(-(v-10)/20.3);

            a3 = (3.7933e-7)*exp(-v/7.7);
            b3 = 0.0084 + 0.00002*v;

            a2 = 9.178*exp(v/29.68);
            b2 = (a13*a2*a3)/(b13*b3);

            a4 = a2/100;
            b4 = a3;

            a5 = a2/(9.5e4);
            b5 = a3/50;
            vx = v;
        end

        dy=[-(a11 + b3)*y(1) + b11*y(2) + a3*y(5);
            -(b11 + b3 + a12)*y(2) + a11*y(1) + b12*y(3) + a3*y(6);
            -(b12 + b3 + a13)*y(3) + a12*y(2) + b13*y(4) + a3*y(7);
            -(b13 + a2)*y(4) + a13*y(3) + b2*y(7);
            -(a11 + a3)*y(5) + b11*y(6) + b3*y(1);
```

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

E.9 ex09

```
% cellpot.m
function cellpot
    tic
    m = 6;
    dt = 0.05;
    totalTime = 600.0;
    npts = int32(totalTime/dt)+1;

    Dm = 60.0;      % (micrometer) diameter of the cell
    Lm = 70.0;      % (micrometer) length of the cell

    dGleak = 0.002; % (mS/cm^2)
    dGNa   = 0.05;  % 0.53;   % (mS/cm^2)
    dGK    = 0.15;  % 0.13;  % (mS/cm^2)
    dGTCa  = 0.03;  % (mS/cm^2)
    fac = 10;       % conversion factor (mS/cm^2) -> (pS/micrometer^2)

    Eleak = -65.0;  % (mV)
    ENa   = 50.0;   % (mV)
    EK    = -100.0; % (mV)
    ECa   = 50.0;   % (mV)
    vtraub = -55.0; % (mV)

    Am    = 3.14 * Dm * Lm;    % (squre_micrometer) surface area
    Gleak = Am * dGleak * fac;
    Cmm   =  0.01 * Am;
    InvCm = 1/Cmm;
    GNa   = Am * dGNa * fac;
    GK    = Am * dGK * fac;
    GTCa  = Am * dGTCa * fac;

    Iinj = -2000; %2000.0;  %(pA)
    Ton  = 50;
    Toff = 300;   % 40;
    vinit = -63.27;

    y = zeros(1,m);
    y = setInitialValue(y);

    t = linspace(0.0, totalTime, npts);
    [t,y] = ode45(@(tt, yy)RHS_eqs(tt, yy), t, y);

    th = -20.0;
    sn = 0;
    for i=1:size(y,1)-1
        if y(i+1,1)>=th && y(i,1)<th
            sn = sn +1;
        end
    end
    fprintf('number of spikes = %d\n', sn)
    toc
    plot(t, y(:,1))

    function [a,b] = get_Na_m(v)
        v2 = v - vtraub;
        a = 0.32 * (13.0-v2) / ( exp((13.0-v2)/4.0) - 1.0);
        b = 0.28 * (v2-40.0) / ( exp((v2-40.0)/5.0) - 1.0);
    end

    function [a, b] = get_Na_h(v)
```

```
        v2 = v - vtraub;
        a = 0.128 * exp((17.0-v2)/18.0);
        b = 4.0/(1.0 + exp((40.0-v2)/5.0));
    end

    function [a,b] = get_K_n(v)
        v2 = v - vtraub;
        a = 0.032 * (15.0-v2)/( exp((15.0-v2)/5.0) - 1.0);
        b = 0.5 * exp((10.0-v2)/40);
    end

    function [tau, inf] = get_TCa_m(v)
        vshift=0;
        vs = v + vshift;
        tau = (3.0 + 1.0 / ( exp((vs+25.0)/10) + exp(-(vs+100.0)/15.0) ) );
        inf = 1.0 / ( 1.0 + exp(-(vs+50.0)/7.4) );
    end

    function [tau, inf] = get_TCa_h(v)
        vshift=0;
        vs = v + vshift;
        tau = ( 85.0 + 1.0 / ( exp((vs+46.0)/4.0) + exp(-(vs+405.0)/50.0) ) ) ;
        inf = 1.0 / ( 1.0 + exp((vs+78.0)/5.0) );
    end

    function y = setInitialValue(y)
        y(1) = vinit;
        v = y(1);
        [a,b] = get_Na_m(v);    y(2) = a/(a+b);
        [a,b] = get_Na_h(v);    y(3) = a/(a+b);
        [a,b]= get_K_n(v);      y(4) = a/(a+b);
        [~,inf] = get_TCa_m(v); y(5) = inf;
        [~,inf] = get_TCa_h(v); y(6) = inf;
    end


    function dy = RHS_eqs(t, y)
        if t<Ton || t>=Toff
            c = 0.0;
        else
            c = Iinj;
        end
        v = y(1);
        INa = GNa*y(2)^3*y(3)*(v-ENa);  % INa = GNa * m^3 * h * (v-Ena)
        IK  = GK * y(4)^4 * (v-EK);     % IK = GK * n^4 (v-EK)
        ITCa = GTCa * y(5)^2 * y(6) * (v - ECa);

        [a1,b1] = get_Na_m(v);
        [a2,b2] = get_Na_h(v);
        [a3,b3] = get_K_n(v);
        [tau1,inf1] = get_TCa_m(v);
        [tau2,inf2] = get_TCa_h(v);

        dy = [ InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa );
            a1 - (a1+b1)*y(2);
            a2 - (a2+b2)*y(3);
            a3 - (a3+b3)*y(4);
            (inf1 - y(5)) / tau1;
            (inf2 - y(6)) / tau2];
    end
end
```

E.10 ex10

```
% release.m
% Stochastic neurotransmitter release: ex12.m
% Ralf Schneggenburger & Erwin Neher
% Nature 406, 889-893, 2000}

```

```
function main()
    tic

    ttime = 4.0;     % total time
    dt    = 0.02;    % time step
    peak  = 20.0;    % peak calcium concentration
    hw    = 1.0;     % half width of calcium transient
    iter  = 100000;  % number of trials

    npts = int32(ttime/dt)+1;
    tt   = linspace(0.0, ttime, npts);
    sum = zeros(1, npts, 'int32');
    ca   = zeros(1, npts, 'double');

    sprintf('%d iterations\n', iter)

    % ca transient
    d = 2.0*sqrt((1.0 - sqrt(0.5)))/hw;
    k = int32(2.0/(dt*d));
    ca(1:k) = peak * (((d*tt(1:k) - 1.0).^2.0)-1.0).^2 ;

    % kinetic constants
    kon    = 9.0e-2; % microM^{-1} ms^{-1}
    koff   = 9.5;    % ms^{-1}
    beta   = 0.25;   % cooperativity
    gamma  = 6.0;    % 6.0d0  ! ms^{-1}
    epsilon= 0.3;    % ms^{-1}
    initialState = 1;

    konh   = dt * kon;
    koffh  = dt * koff;
    gammah = dt * gamma;
    epsilonh = dt*epsilon;
    beta2  = beta*beta;
    beta3  = beta*beta2;
    beta4  = beta*beta3;

    counter = 0;
    for k = 1:iter
        s  = initialState;  %initial state
        for i = 1:npts
            r = rand;              % get a random number [0, 1]
            alpha = konh * ca(i);  % get Ca level
            snext = s;
            switch s
                case 1
                    th = 5.0 * alpha;
                    if r<th
                        snext = 2;
                    end
                case 2
                    th = 4.0 * alpha;
                    if r < th
                        snext = 3;
                    else
                        th = th + koffh;
                        if r<th
                            snext = 1;
                        end
                    end
                case 3
                    th = 3.0 * alpha;
                    if r < th
                        snext = 4;
                    else
                        th = th + koffh * beta;
                        if r < th
                            snext = 2;
                        end
                    end
                case 4
```

```
                    th = 2.0 * alpha;
                    if r < th
                        snext = 5;
                    else
                        th = th + koffh * beta2;
                        if r < th
                            snext = 3;
                        end
                    end
                case  5
                    th = alpha;
                    if r < th
                        snext = 6;
                    else
                        th = th + koffh * beta3;
                        if r < th
                            snext = 4;
                        end
                    end
                case 6
                    th = gammah;
                    if r < th
                        snext = 7;
                    else
                        th = th + koffh * beta4;
                        if r < th
                            snext = 5;
                        end
                    end
                case  7      % release
                    snext = 0;
                    sum(i) = sum(i) + 1;
                    counter = counter + 1;
                case  0      %recovery from release
                    th = epsilonh;
                    if r < th
                        snext = 1;
                    end
            end % end of switch
            s = snext;
        end  % end of i-loop
    end % end of k-loop
    sprintf('release probability = %f\n', counter/iter)
    toc
    plot(tt,sum, tt, ca)
end
```

F Python

F.1 ex01

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

F.2 ex02

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

F.3 ex03

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

F.4 ex04

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

F.5 ex05

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

F.6 ex06

```
#  plotgraph.py
import numpy as np
import matplotlib.pyplot as plt
import os

dt = 0.1
ttime = 10.0
npts = int(10.0/0.1) + 1
rng = np.random.RandomState()
a=rng.uniform(0.0, 1.0, npts)
x = np.linspace(0.0, ttime, npts)

plt.figure()
plt.plot(x,a)
```

```
plt.show()
"""
f = open('a.txt','w')
for i in range(101):
    f.write('%f %f\n'%(x[i], a[i]))
f.close()
os.system('gc a.txt &')
"""
```

F.7 ex07

```
# ode.py
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def rhs(y,t):  # return derivatives of the array y
        return np.array([y[1], -y[0]])

def ode():
    ndim = 2
    ttime = 10.0
    dt = 0.1
    npts = int(ttime/dt)+1
    y0 = np.zeros(ndim)
    y0[0] = 1.0

    t = np.linspace(0.0, ttime, npts)
    y = odeint(rhs, y0, t)

    plt.figure()
    plt.plot(t, y[:,0], t, y[:,1])
    plt.xlabel('t')
    plt.ylabel('y')
    plt.show()

if __name__ == '__main__':
    ode()
```

F.8 ex08

```
# channel.py
import numpy as np
import math
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def channel():
    ndim = 9
    dt = 0.01
    totalTime = 50.0;
    npts = (int)(totalTime/dt)+1;
    vold = -1000.0
    a2 = a3 = a4 = a5 = a11 = a12 = a13 = 0.0
    b2 = b3 = b4 = b5 = b11 = b12 = b13 = 0.0
    def rhs(y, t):
        if t<20.0 or t>= 30.0:
            v = -100.0
        else:
            v =  0.0
        if v != vold:
            a11 = 3.802/(0.1027*math.exp(-v/17.0) + 0.20*math.exp(-v/150.0))
            a12 = 3.802/(0.1027*math.exp(-v/15.0) + 0.23*math.exp(-v/150.0))
            a13 = 3.802/(0.1027*math.exp(-v/12.0) + 0.25*math.exp(-v/150.0))

            b11 = 0.1917*math.exp(-v/20.3)
```

```
            b12 = 0.20*math.exp(-(v-5)/20.3)
            b13 = 0.22*math.exp(-(v-10)/20.3)

            a3 = (3.7933e-7)*math.exp(-v/7.7)
            b3 = 0.0084 + 0.00002*v

            a2 = 9.178*math.exp(v/29.68)
            b2 = (a13*a2*a3)/(b13*b3)

            a4 = a2/100.0
            b4 = a3

            a5 = a2/(9.5e4)
            b5 = a3/50.0
            v = vold
        return np.array(
            [-(a11 + b3)*y[0] + b11*y[1] + a3*y[4],
             -(b11 + b3 + a12)*y[1] + a11*y[0] + b12*y[2] + a3*y[5],
             -(b12 + b3 + a13)*y[2] + a12*y[1] + b13*y[3] + a3*y[6],
             -(b13 + a2)*y[3] + a13*y[2] + b2*y[6],
             -(a11 + a3)*y[4] + b11*y[5] + b3*y[0],
             -(b11 + a3 + a12)*y[5] + a11*y[4] + b12*y[6] + b3*y[1],
             -(b12 + a3 + b2 + a4)*y[6] + a12*y[5] + b3*y[2] + a2*y[3] + b4*y[7],
             -(b4 + a5)*y[7] + a4*y[6] + b5*y[8],
             -b5*y[8] + a5*y[7] ])

    y0 = np.zeros(ndim)
    y0[0] = 1.0
    t = np.linspace(0.0, totalTime, npts)
    y = odeint(rhs, y0, t)

    plt.figure()
    plt.plot(t, y[:,0], t, y[:,3], t, y[:,6], t, y[:,7],t, y[:,8])
    plt.xlabel('time (ms)')
    plt.ylabel('probability')
    plt.show()

if __name__ == '__main__':
    channel()
```

F.9 ex09

```
# cellpot.py
import os
import numpy as np
import scipy.integrate
import math
import matplotlib.pyplot as plt

def cellpot():
    ndim = 6
    ttime = 600.0
    dt = 0.05
    npts = int(ttime/dt)+1
    istate = 0

    Dm = 60.0     # (micrometer) diameter of the cell
    Lm = 70.0     # (micrometer) length of the cell

    Eleak =  -65.0 # (mV)
    ENa   =   50.0 # (mV)
    EK    = -100.0 # (mV)
    ECa   =   50.0 # (mV)

    dGleak = 0.002 # (mS/cm^2)
    dGNa   = 0.05  # 0.53;   // (mS/cm^2)
    dGK    = 0.15  # 0.13;  // (mS/cm^2)
    dGTCa  = 0.03  # (mS/cm^2)

```

```
    vtraub = -55.0 # (mV)

    Iinj = -2000.0 # 2000.0;  //(pA)
    Ton  = 50.0
    Toff = 300.0   # 40;
    vinit = -63.27

    fac = 10     # conversion factor (mS/cm^2) -> (pS/micrometer^2)

    Am = 3.14 * Dm * Lm # (squre_micrometer) surface area
    Cm = 0.01 * Am
    InvCm = 1/Cm

    Gleak = Am * dGleak * fac
    GNa   = Am * dGNa * fac
    GK    = Am * dGK * fac
    GTCa  = Am * dGTCa * fac

    def get_Na_m(v):
        v2 = v-vtraub
        a = 0.32 * (13-v2) / ( math.exp((13-v2)/4) - 1)
        b = 0.28 * (v2-40) / ( math.exp((v2-40)/5) - 1)
        return [a,b]

    def get_Na_h(v):
        v2 = v-vtraub
        a = 0.128 * math.exp((17-v2)/18)
        b = 4 / ( 1 + math.exp((40-v2)/5) )
        return [a,b]

    def get_K_n(v):
        v2 = v - vtraub
        a = 0.032 * (15-v2) / ( math.exp((15-v2)/5) - 1)
        b = 0.5 * math.exp((10-v2)/40)
        return [a,b]

    def get_TCa_m(v):
        vshift=0
        vs = v + vshift
        tau = ( 3 + 1.0 / ( math.exp((vs+25)/10) +  math.exp(-(vs+100)/15)))
        inf = 1.0 / ( 1 + math.exp(-(vs+50)/7.4) )
        return [tau,inf]

    def get_TCa_h(v):
        vshift=0
        vs = v + vshift
        tau = (85.0+1.0/(math.exp((vs+46.0)/4.0) + math.exp(-(vs+405.0)/50.0) ) )
        inf = 1.0 / ( 1.0 + math.exp((vs+78.0)/5.0) )
        return [tau, inf]

    def setInitialValues(y):
        y[0] = vinit
        v = y[0]
        [a,b] = get_Na_m(v)
        y[1] = a/(a+b)

        [a,b] = get_Na_h(v)
        y[2] = a/(a+b)

        [a,b] = get_K_n(v)
        y[3] = a/(a+b)

        [tau,inf] = get_TCa_m(v)
        y[4] = inf

        [tau,inf] = get_TCa_h(v)
        y[5] = inf


    def rhs(y,t):  # return derivatives of the array y
        if t >= Ton and  t < Toff :
```

```
            c = Iinj
        else:
            c = 0.0

        v = y[0];
        INa = GNa * y[1]**3 * y[2]  * (v-ENa)
        IK  = GK  * y[3]**4 * (v-EK)
        ITCa = GTCa * pow(y[4],2.0) * y[5] * (v - ECa)

        [a1,b1]=get_Na_m(v)
        [a2,b2] = get_Na_h(v)
        [a3,b3] = get_K_n(v)
        [tau1,inf1] = get_TCa_m(v)
        [tau2,inf2] = get_TCa_h(v)

        return np.array(
          [ InvCm * (c - Gleak * (v-Eleak) - INa - IK -  ITCa ),
            a1 - (a1+b1)*y[1],
            a2 - (a2+b2)*y[2],
            a3 - (a3+b3)*y[3],
            (inf1 - y[4]) / tau1,
            (inf2 - y[5]) / tau2])

# main
    t = np.linspace(0.0, ttime, npts)
    y0 = np.zeros(ndim)
    setInitialValues(y0)
    y = scipy.integrate.odeint(rhs, y0, t)

    th = -20.0
    sn = 0
    for i in range(npts-1):
        if y[i+1,0] >= th and y[i,0] < th:
           sn = sn+1
    print('number of spikes = ', sn)

    plt.figure()
    plt.plot(t, y[:,0])
    plt.xlabel('time (ms)')
    plt.ylabel('voltage (mV)')
    plt.show()

if __name__=='__main__':
    cellpot()
```

F.10 ex10

```
"""
release.py
Stochastic neurotransmitter release
Ralf Schneggenburger & Erwin Neher
Nature 406, 889-893, 2000
"""
import numpy as np
import random
import math
import matplotlib.pyplot as plt

def release():
    ttime = 4.0
    dt    = 0.02
    peak  = 20.0
    hw    = 1.0
    iter  = 100000

    npts = (int)(ttime/dt)+1
    tt  = np.linspace(0.0, ttime, npts)
    sum = np.zeros(npts, int)
    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:
```

```
                    snext = 7
                else:
                    th = th + koffh * beta4
                    if r < th:
                        snext = 5
            elif s == 7:
                snext = 0     #release
                sum[i] += 1  # release
                counter += 1
            elif s == 0:          #recovery from release
                th = epsilonh
                if r < th:
                    snext = 1

            s = snext;

    print('release probability = {0:.4f}'.format(float(counter)/iter))

#    plt.figure()
#    plt.plot(tt,sum, tt, ca)
#    plt.show()

if __name__ == '__main__':
    release()
```

G Java

G.1 ex01

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

G.2 ex02

```java
//  loop.java
public class loop {
    public static void main(String [] args){
        for(int i=0;i<10;i++){
            System.out.println("i = " +  i);
        }
    }
}
```

G.3 ex03

```java
//  ifelse.java
public class ifelse {
    public static void main(String [] args){

        for(int i=0;i<10;i++){
            if(i%2==0){
                System.out.println("even: " + i);
            }else{
                System.out.println("odd:  " + i);
            }
        }
    }
}
```

G.4 ex04

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

G.5 ex05

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

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

G.6 ex06

```
// plotgraph.java
import java.util.Random;
import java.io.FileWriter;
import java.io.IOException;

public class plotgraph{

    public static void main(String [] args){
        double dt = 0.1;
        double ttime = 10.0;
        int npts = (int)(ttime/dt) + 1;
        Random ran = new Random();

        try{
            FileWriter fw = new FileWriter("ax.txt");
            for(int i=0;i<npts;i++){
                double a = ran.nextDouble();
                fw.write(dt*i + " " + a + "\n");
            }
            fw.close();
        }catch(IOException e){
            System.out.println(e);
        }


        String cmd;
        String OS = System.getProperty("os.name", "generic").toLowerCase();
        if (OS.indexOf("win") >= 0) {
            cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
        }else{ // mac
            cmd = "..//..//tools//gc ax.txt";
        }

        try{
            Process p = Runtime.getRuntime().exec(cmd);
            p.waitFor();
        } catch (Exception e) {}
    }
}
```

G.7 ex07a

```
//  ode.java (ex07)
import java.io.FileWriter;
import java.io.IOException;

public class ode{

    public static void main(String [] args){

        int n = 2;                  // number of equations
        double h = 0.1;               // step size
        double ttime = 10.0;          // total time
        int npts = (int)(ttime/h)+1; // number of points
        double [] y = {1.0, 0.0};  // initial values

```

```
        Eqs eq = new Eqs();
        eq.initRK4(n, h);

        try{
            FileWriter fw = new FileWriter("ax.txt");
            for(int i=0;i<npts;i++){
                double t = h*i;
                fw.write(t + " " + y[0] + " " + y[1] + "\n");
                eq.advanceStep(t, y);
            }
            fw.close();
        }catch(IOException e){
            System.out.println(e);
        }

        String cmd;
        String OS = System.getProperty("os.name", "generic").toLowerCase();
        if (OS.indexOf("win") >= 0) {
            cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
        }else{ // mac
            cmd = "..//..//tools//gc ax.txt";
        }

        try{
            Process p = Runtime.getRuntime().exec(cmd);
            p.waitFor();
        } catch (Exception e) {}
    }
}
```

```

public class Eqs extends RungeKutta4 {

    public  void rhs(double tt, double [] y, double [] dy){
        dy[0] =  y[1];
        dy[1] = -y[0];
    }
}
```

```

public class RungeKutta4{
    int n;
    double h;
    double [] yn;
    double [] k1;
    double [] k2;
    double [] k3;
    double [] k4;

    public void initRK4(int ndim, double dt){
        n = ndim;
        h = dt;
        yn =  new double[n];
        k1 =  new double[n];
        k2 =  new double[n];
        k3 =  new double[n];
        k4 =  new double[n];
    }

    public  void advanceStep(double t, double [] y){
        rhs(t, y, k1);
        for(int i=0;i<n;i++) yn[i] = y[i] + h*k1[i]/2;
        rhs(t+h/2, yn, k2);
        for(int i=0;i<n;i++) yn[i] = y[i] + h*k2[i]/2;
        rhs(t+h/2, yn, k3);
        for(int i=0;i<n;i++) yn[i] = y[i] + h*k3[i];
        rhs(t+h, yn, k4);
        for(int i=0;i<n;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
    }
    public void rhs(double tt, double [] y, double [] dy){}
```

```
}
```

G.8 ex08

```
//  channel.java (ex08)
import java.io.FileWriter;
import java.io.IOException;

public class channel{

    public static void main(String [] args){
        int n = 9;                          // number of equations
        double h = 0.01;                    // step size
        double totalTime  = 50.0;           // total time
        int npts = (int)(totalTime /h)+1; // number of points
        double [] y = {1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};  // initial values

        Eqs eq = new Eqs();
        eq.initRK4(n, h);

        try{
            FileWriter fw = new FileWriter("ax.txt");
            for(int i=0;i<npts;i++){
                double t = h*i;
                fw.write(t+" "+ y[0] +" "+ y[3]+" "+ y[6]+" "+ y[7]+" "+ y[8] + "\n");
                eq.advanceStep(t, y);
            }
            fw.close();
        }catch(IOException e){
            System.out.println(e);
        }



        String cmd;
        String OS = System.getProperty("os.name", "generic").toLowerCase();
        if (OS.indexOf("win") >= 0) {
            cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
        }else{ // mac
            cmd = "..//..//tools//gc ax.txt";
        }

        try{
            Process p = Runtime.getRuntime().exec(cmd);
            p.waitFor();
        } catch (Exception e) {}




    }
}
```

```
public class Eqs extends RungeKutta4 {

    double a11, a12, a13, a2, a3, a4, a5;
    double b11, b12, b13, b2, b3, b4, b5;
    double vold;

    public Eqs(){
        vold = -1000.0;
    }

    public void rhs(double t, double [] y, double [] dy){
        double v = t<20.0 || t>= 30.0 ? -100.0 : 0.0;
        if(v != vold){
            a11 = 3.802/(0.1027*Math.exp(-v/17.0) + 0.20*Math.exp(-v/150));
            a12 = 3.802/(0.1027*Math.exp(-v/15.0) + 0.23*Math.exp(-v/150));
```

```
            a13 = 3.802/(0.1027*Math.exp(-v/12.0) + 0.25*Math.exp(-v/150));

            b11 = 0.1917*Math.exp(-v/20.3);
            b12 = 0.20*Math.exp(-(v-5)/20.3);
            b13 = 0.22*Math.exp(-(v-10)/20.3);

            a3 = (3.7933e-7)*Math.exp(-v/7.7);
            b3 = 0.0084 + 0.00002*v;

            a2 = 9.178*Math.exp(v/29.68);
            b2 = (a13*a2*a3)/(b13*b3);

            a4 = a2/100;
            b4 = a3;

            a5 = a2/(9.5e4);
            b5 = a3/50;
            vold = v;
        }
        dy[0] = -(a11 + b3)*y[0] + b11*y[1] + a3*y[4];
        dy[1] = -(b11 + b3 + a12)*y[1] + a11*y[0] + b12*y[2] + a3*y[5];
        dy[2] = -(b12 + b3 + a13)*y[2] + a12*y[1] + b13*y[3] + a3*y[6];
        dy[3] = -(b13 + a2)*y[3] + a13*y[2] + b2*y[6];
        dy[4] = -(a11 + a3)*y[4] + b11*y[5] + b3*y[0];
        dy[5] = -(b11 + a3 + a12)*y[5] + a11*y[4] + b12*y[6] + b3*y[1];
        dy[6] = -(b12 + a3 + b2 + a4)*y[6] + a12*y[5] + b3*y[2] + a2*y[3] + b4*y[7];
        dy[7] = -(b4 + a5)*y[7] + a4*y[6] + b5*y[8];
        dy[8] = -b5*y[8] + a5*y[7];
    }
}
```

G.9 ex09

```
// Cellpot java
import java.io.FileWriter;
import java.io.IOException;

public class Cellpot{

    public static void main(String [] args){
        int n = 6;                          // number of equations
        double h = 0.05;                    // step size
        double totalTime  = 600.0;           // total time
        int npts = (int)(totalTime /h)+1; // number of points
        double vinit = -63.27;  // initial voltate
        double Iinj  = -2000.0; // current injection(pA)
        double Ton   = 50.0;    // on-time
        double Toff  = 300.0;   // off-time

        double [] y = new double[n];
        double [] vol = new double[npts];

        Cell cell = new Cell();
        cell.initRK4(n, h);
        cell.setInitialValues(vinit, y);
        cell.setPulse(Ton, Toff, Iinj);

        try{
            FileWriter fw = new FileWriter("ax.txt");
            for(int i=0;i<npts;i++){
                double t = h*i;
                vol[i] = y[0];
                fw.write(t + " " + y[0] + "\n");
                cell.advanceStep(t, y);
            }
            fw.close();
        }catch(IOException e){
            System.out.println(e);
        }
```

```
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;    // (squre_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
54          get_Na_h(v,aa);
55          y[2] = aa[0]/(aa[0]+aa[1]);
56
57          get_K_n(v,aa);
58          y[3] = aa[0]/(aa[0]+aa[1]);
59
60
61          get_TCa_m(v,aa);
62          y[4] = aa[1];
63
64          get_TCa_h(v,aa);
65          y[5] = aa[1];
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;
```

```
        double gammah = dt * gamma;
        double epsilonh = dt*epsilon;
        double beta2 = beta*beta;
        double beta3 = beta*beta2;
        double beta4 = beta*beta3;

        for(int k=0;k<iter;k++){    // iteration loop
            int s  = initialState;  //initial state
            for(int i=0;i<npts;i++){
                double th;
                double r = ran.nextDouble();                    // get a random number [0, 1]
                double alpha = konh * ca[i];  // get Ca level
                int snext = s;
                switch (s){
                    case (1):
                        th = 5.0 * alpha;
                        if(r<th) snext = 2;
                        break;
                    case (2):
                        th = 4.0 * alpha;
                        if(r<th) {
                            snext = 3;
                        }else{
                            th = th + koffh;
                            if(r<th) snext = 1;
                        }
                        break;
                    case (3):
                        th = 3.0 * alpha;
                        if(r<th) {
                            snext = 4;
                        }else{
                            th = th + koffh * beta;
                            if(r<th) snext = 2;
                        }
                        break;
                    case (4):
                        th = 2.0 * alpha;
                        if(r<th){
                            snext = 5;
                        }else{
                            th = th + koffh * beta2;
                            if(r<th) snext = 3;
                        }
                        break;
                    case (5):
                        th = alpha;
                        if(r<th){
                            snext = 6;
                        }else{
                            th = th + koffh * beta3;
                            if(r<th) snext = 4 ;
                        }
                        break;
                    case (6):
                        th = gammah;
                        if(r<th){
                            snext = 7;
                        }else{
                            th = th + koffh * beta4;
                            if(r<th) snext = 5;
                        }
                        break;
                    case (7):
                        snext = 0;     //release
                        sum[i] += 1;  // release
                        counter++;
                        break;
                    case (0):         //recovery from release
                        th = epsilonh;
                        if(r<th) snext = 1;
```

```
                        break;
                    default:
                        ;
                } // end of switch
                s = snext;
            } //end of i-loop
        } // end of k-loop

        System.out.println("release probability = " + (double)counter/(double)iter);

        try{
            FileWriter fw = new FileWriter("ax.txt");
            for(int i=0;i<npts;i++){
                fw.write(dt*i+" "+ ca[i] +" "+ (double)sum[i]+"\n");
            }
            fw.close();
        }catch(IOException e){
            System.out.println(e);
        }



        String cmd;
        String OS = System.getProperty("os.name", "generic").toLowerCase();
        if (OS.indexOf("win") >= 0) {
            cmd = "cmd /c ..\\..\\tools\\gc ax.txt";
        }else{ // mac
            cmd = "..//..//tools//gc ax.txt";
        }

        try{
            Process p = Runtime.getRuntime().exec(cmd);
            p.waitFor();
        } catch (Exception e) {}
    }
}
```

H Fortran

H.1 ex01

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

H.2 ex02

```
program loop
    implicit none
    integer :: i

    do i = 0, 9
        write (*, fmt='("i = ", i3)') i
    end do
end program loop
```

H.3 ex03

```
! ifelse.f90
program ifelse
    implicit none
    integer :: i

    do i = 0, 9
        if( mod(i,2) == 0) then
            write (*, fmt='(i2, " even")') i
        else
            write (*, fmt='(i2, " odd")') i
        end if
    end do

end program ifelse
```

H.4 ex04

```
! wloop.f90
program wloop
    implicit none
    integer :: i

    i = 1
    do while (.true.)
        if(mod(i,17)==0 .and.  mod(i+2,11)==0 .and.  mod(i+4, 23)==0)then
            write(*, fmt='("i = ", I6)') i
            exit
        end if
        i = i+1
    end do
end program wloop
```

H.5 ex05

```
program rng
    use mtmod
    implicit none

```

```
    integer, parameter :: npts = 10
    integer :: j, k, c

    call system_clock(count=c)
    call sgrnd(c)

    do j=0,npts-1
        write (*, fmt='(I2, ": ", F9.6)') j, grnd()
    enddo

end program rng
```

```
!-----------------------------------------------------------------------
! A C-program for MT19937: Real number version
!   genrand() generates one pseudorandom real number (double)
! which is uniformly distributed on [0,1]-interval, for each
! call. sgenrand(seed) set initial values to the working area
! of 624 words. Before genrand(), sgenrand(seed) must be
! called once. (seed is any 32-bit integer except for 0).
! Integer generator is obtained by modifying two lines.
!   Coded by Takuji Nishimura, considering the suggestions by
! Topher Cooper and Marc Rieffel in July-Aug. 1997.
!
! This library is free software; you can redistribute it and/or
! modify it under the terms of the GNU Library General Public
! License as published by the Free Software Foundation; either
! version 2 of the License, or (at your option) any later
! version.
! This library is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
! See the GNU Library General Public License for more details.
! You should have received a copy of the GNU Library General
! Public License along with this library; if not, write to the
! Free Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
! 02111-1307  USA
!
! Copyright (C) 1997 Makoto Matsumoto and Takuji Nishimura.
! When you use this, send an email to: matumoto@math.keio.ac.jp
! with an appropriate reference to your work.
!
!***********************************************************************
! Fortran translation by Hiroshi Takano.  Jan. 13, 1999.
!
!   genrand()      -> double precision function grnd()
!   sgenrand(seed) -> subroutine sgrnd(seed)
!                     integer seed
!
! This program uses the following non-standard intrinsics.
!   ishft(i,n): If n>0, shifts bits in i by n positions to left.
!               If n<0, shifts bits in i by n positions to right.
!   iand (i,j): Performs logical AND on corresponding bits of i and j.
!   ior  (i,j): Performs inclusive OR on corresponding bits of i and j.
!   ieor (i,j): Performs exclusive OR on corresponding bits of i and j.
!
!***********************************************************************
! Fortran version rewritten as an F90 module and mt state saving and getting
! subroutines added by Richard Woloshyn. (rwww@triumf.ca). June 30, 1999

 module mtmod
! Default seed
    integer, parameter :: defaultsd = 4357
! Period parameters
    integer, parameter :: N = 624, N1 = N + 1

! the array for the state vector
    integer, save, dimension(0:N-1) :: mt
    integer, save                   :: mti = N1

! Overload procedures for saving and getting mt state
```

```
    interface mtsave
      module procedure mtsavef
      module procedure mtsaveu
    end interface
    interface mtget
      module procedure mtgetf
      module procedure mtgetu
    end interface

 contains

!Initialization subroutine
  subroutine sgrnd(seed)
    implicit none
!
!       setting initial seeds to mt[N] using
!       the generator Line 25 of Table 1 in
!       [KNUTH 1981, The Art of Computer Programming
!          Vol. 2 (2nd Ed.), pp102]
!
    integer, intent(in) :: seed

    mt(0) = iand(seed,-1)
    do mti=1,N-1
      mt(mti) = iand(69069 * mt(mti-1),-1)
    enddo
!
    return
  end subroutine sgrnd

!Random number generator
  real(8) function grnd()
    implicit integer(a-z)

! Period parameters
    integer, parameter :: M = 397, MATA  = -1727483681
!                                    constant vector a
    integer, parameter :: LMASK =  2147483647
!                                    least significant r bits
    integer, parameter :: UMASK = -LMASK - 1
!                                    most significant w-r bits
! Tempering parameters
    integer, parameter :: TMASKB= -1658038656, TMASKC= -272236544

    dimension mag01(0:1)
    data mag01/0, MATA/
    save mag01
!                         mag01(x) = x * MATA for x=0,1

    TSHFTU(y)=ishft(y,-11)
    TSHFTS(y)=ishft(y,7)
    TSHFTT(y)=ishft(y,15)
    TSHFTL(y)=ishft(y,-18)

    if(mti.ge.N) then
!                       generate N words at one time
      if(mti.eq.N+1) then
!                             if sgrnd() has not been called,
        call sgrnd( defaultsd )
!                              a default initial seed is used
      endif

      do kk=0,N-M-1
          y=ior(iand(mt(kk),UMASK),iand(mt(kk+1),LMASK))
          mt(kk)=ieor(ieor(mt(kk+M),ishft(y,-1)),mag01(iand(y,1)))
      enddo
      do kk=N-M,N-2
          y=ior(iand(mt(kk),UMASK),iand(mt(kk+1),LMASK))
          mt(kk)=ieor(ieor(mt(kk+(M-N)),ishft(y,-1)),mag01(iand(y,1)))
      enddo
      y=ior(iand(mt(N-1),UMASK),iand(mt(0),LMASK))
```

```
      mt(N-1)=ieor(ieor(mt(M-1),ishft(y,-1)),mag01(iand(y,1)))
      mti = 0
    endif

    y=mt(mti)
    mti = mti + 1
    y=ieor(y,TSHFTU(y))
    y=ieor(y,iand(TSHFTS(y),TMASKB))
    y=ieor(y,iand(TSHFTT(y),TMASKC))
    y=ieor(y,TSHFTL(y))

    if(y .lt. 0) then
      grnd=(dble(y)+2.0d0**32)/(2.0d0**32-1.0d0)
    else
      grnd=dble(y)/(2.0d0**32-1.0d0)
    endif

    return
  end function grnd

!State saving subroutines.
! Usage:  call mtsave( file_name, format_character )
!    or   call mtsave( unit_number, format_character )
! where   format_character = 'u' or 'U' will save in unformatted form, otherwise
!         state information will be written in formatted form.
  subroutine mtsavef( fname, forma )

!NOTE: This subroutine APPENDS to the end of the file "fname".

    character(*), intent(in) :: fname
    character, intent(in)    :: forma

    select case (forma)
      case('u','U')
       open(unit=10,file=trim(fname),status='UNKNOWN',form='UNFORMATTED', &
            position='APPEND')
       write(10)mti
       write(10)mt

      case default
       open(unit=10,file=trim(fname),status='UNKNOWN',form='FORMATTED', &
            position='APPEND')
       write(10,*)mti
       write(10,*)mt

    end select
    close(10)

    return
  end subroutine mtsavef

  subroutine mtsaveu( unum, forma )

    integer, intent(in)    :: unum
    character, intent(in)  :: forma

    select case (forma)
      case('u','U')
       write(unum)mti
       write(unum)mt

      case default
       write(unum,*)mti
       write(unum,*)mt

      end select

    return
  end subroutine mtsaveu

!State getting subroutines.
```

```
! Usage:  call mtget( file_name, format_character )
!    or   call mtget( unit_number, format_character )
! where   format_character = 'u' or 'U' will read in unformatted form, otherwise
!         state information will be read in formatted form.
  subroutine mtgetf( fname, forma )

    character(*), intent(in) :: fname
    character, intent(in)    :: forma

    select case (forma)
      case('u','U')
       open(unit=10,file=trim(fname),status='OLD',form='UNFORMATTED')
       read(10)mti
       read(10)mt

      case default
       open(unit=10,file=trim(fname),status='OLD',form='FORMATTED')
       read(10,*)mti
       read(10,*)mt

    end select
    close(10)

    return
  end subroutine mtgetf

  subroutine mtgetu( unum, forma )

    integer, intent(in)    :: unum
    character, intent(in)  :: forma

    select case (forma)
      case('u','U')
       read(unum)mti
       read(unum)mt

      case default
       read(unum,*)mti
       read(unum,*)mt

      end select

    return
  end subroutine mtgetu

 end module mtmod
```

```
fc     = gfortran
cflags  = -O3
lflags  =
exec = rng
objs = mtfort90.o rng.o

all: $(exec)
$(exec): $(objs)
        $(fc) $(lflags) $(objs) -o $@
%.o: %.f90
        $(fc) -c $(cflags) $<
clean:
        -rm $(exec) $(objs) *.mod
```

H.6 ex06

`mtfort90.f90` is the same as in `ex05`.

```
program rng
    use mtmod
    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)
```

```
        yn = y + h*k2/2.0d0

        call rhs(t+h/2.0d0, yn, k3)
        yn = y + h*k3

        call rhs(t+h, yn, k4);
        y = y + h*(k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/6.0d0
    end subroutine advanceStep


    subroutine rhs (tt, y, dy)
        double precision :: tt
        double precision, dimension(:), intent(in):: y
        double precision, dimension(:), intent(out):: dy
        dy(1) =  y(2)
        dy(2) = -y(1)
    end subroutine rhs

end program ode
```

H.8 ex08

```
! channel.f90

program channel
    implicit none

    integer, parameter :: ndim = 9 ! number of equations
    double precision, parameter :: ttime = 50.0d0 ! total time
    double precision, parameter :: h = 0.01d0     ! step size
    integer, parameter :: npts = floor(ttime/h)+1;
    double precision :: vold  = -1000.0d0

    double precision :: a11, a12, a13, b11, b12, b13
    double precision :: a2, b2, a3, b3, a4, b4, a5, b5
    double precision :: v

    double precision :: t
    double precision, dimension(ndim) :: y
    integer :: i

    y(1) = 1.0d0
    y(2:ndim) = 0.0d0

    open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
    do i = 1, npts
        t = h*i
        write(unit=11, fmt='(6E15.7)') t, y(1), y(4), y(7:9)
        call advanceStep(t, y)
    end do
    close(unit=11)

    call system("../../tools/gc ax.txt ");

!   call execute_command_line("..\\..\\tools\\gc ax.txt ");

contains

    subroutine advanceStep(tt, y)
        double precision, intent(in) :: tt
        double precision, dimension(:), intent(inout) :: y
        double precision, dimension(ndim) :: yn, k1, k2, k3, k4

        call rhs(tt, y, k1)
        yn = y + h * k1/2.0d0

        call rhs(tt+h/2.0d0, yn, k2)
        yn = y + h*k2/2.0d0

```

```
        call rhs(tt+h/2.0d0, yn, k3)
        yn = y + h*k3

        call rhs(tt+h, yn, k4);
        y = y + h*(k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/6.0d0
    end subroutine advanceStep

    subroutine rhs (tt, y, dy)
        double precision, intent(in) :: tt
        double precision, dimension(:), intent(in):: y
        double precision, dimension(:), intent(out):: dy


        if(tt<20.0d0 .or. tt>= 30.0d0)then
            v = -100.0d0
        else
            v = 0.0d0
        end if
        if (v .ne. vold ) then
            a11 = 3.802d0/(0.1027d0*exp(-v/17.0d0) + 0.20d0*exp(-v/150.0d0))
            a12 = 3.802d0/(0.1027d0*exp(-v/15.0d0) + 0.23d0*exp(-v/150.0d0))
            a13 = 3.802d0/(0.1027d0*exp(-v/12.0d0) + 0.25d0*exp(-v/150.0d0))

            b11 = 0.1917d0*exp(-v/20.3d0)
            b12 = 0.20d0*exp(-(v-5.0d0)/20.3d0)
            b13 = 0.22d0*exp(-(v-10.0d0)/20.3d0)

            a3 = (3.7933d-7)*exp(-v/7.7d0)
            b3 = 0.0084d0 + 0.00002d0*v

            a2 = 9.178d0*exp(v/29.68d0)
            b2 = (a13*a2*a3)/(b13*b3)

            a4 = a2/100d0
            b4 = a3

            a5 = a2/(9.5d4)
            b5 = a3/50.0d0
            vold = v
        end if

        dy(1) = -(a11 + b3)*y(1) + b11*y(2) + a3*y(5)
        dy(2) = -(b11 + b3 + a12)*y(2) + a11*y(1) + b12*y(3) + a3*y(6)
        dy(3) = -(b12 + b3 + a13)*y(3) + a12*y(2) + b13*y(4) + a3*y(7)
        dy(4) = -(b13 + a2)*y(4) + a13*y(3) + b2*y(7)
        dy(5) = -(a11 + a3)*y(5) + b11*y(6) + b3*y(1)
        dy(6) = -(b11 + a3 + a12)*y(6) + a11*y(5) + b12*y(7) + b3*y(2)
        dy(7) = -(b12 + a3 + b2 + a4)*y(7) + a12*y(6) + b3*y(3) + a2*y(4) + b4*y(8)
        dy(8) = -(b4 + a5)*y(8) + a4*y(7) + b5*y(9)
        dy(9) = -b5*y(9) + a5*y(8)
    end subroutine rhs

end program channel
```

H.9 ex09

```
! cellpot.f90

program cellpot
    implicit none

    integer, parameter :: m = 6
    double precision, parameter :: dt = 0.05d0
    double precision, parameter :: totalTime = 600.0d0

    double precision :: Iinj = -2000.0d0  !; %2000.0;  %(pA)
    double precision :: Ton  = 50.0d0
    double precision :: Toff = 300.0d0
    double precision :: vinit = -63.27d0
```

```
    double precision, parameter :: vtraub = -55.0d0 ! (mV)

    double precision, dimension(m) :: y


    double precision :: t
    integer :: i, npts

    npts = floor(totalTime/dt)+1
    call setInitialValues(y)

    open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
    do i = 1, npts
        t = dt*i
        write(unit=11, fmt='(2E15.7)') t, y(1)
        call advanceStep(t, y)
    end do
    close(unit=11)
    call system("../../tools/gc ax.txt ");

contains

    subroutine advanceStep(tt, y)
        double precision, intent(in) :: tt
        double precision, dimension(:), intent(inout) :: y
        double precision, dimension(m) :: yn, k1, k2, k3, k4

        call rhs(tt, y, k1)
        yn = y + dt * k1/2.0d0

        call rhs(tt+dt/2.0d0, yn, k2)
        yn = y + dt*k2/2.0d0

        call rhs(tt+dt/2.0d0, yn, k3)
        yn = y + dt*k3

        call rhs(tt+dt, yn, k4);
        y = y + dt*(k1 + 2.0d0*k2 + 2.0d0*k3 + k4)/6.0d0
    end subroutine advanceStep


    subroutine get_Na_m(v, a, b)
        double precision, intent(in) :: v
        double precision, intent(out) :: a, b
        double precision :: v2
        v2 = v - vtraub
        a = 0.32d0 * (13.0d0-v2) / ( exp((13.0d0-v2)/4.0d0) - 1.0d0)
        b = 0.28d0 * (v2-40.0d0) / ( exp((v2-40.0d0)/5.0d0) - 1.0d0)
    end subroutine get_Na_m

    subroutine get_Na_h(v, a, b)
        double precision, intent(in) :: v
        double precision, intent(out) :: a, b
        double precision :: v2
        v2 = v - vtraub
        a = 0.128d0 * exp((17.0d0-v2)/18.0d0)
        b = 4.0d0/(1.0d0 + exp((40.0d0-v2)/5.0d0))
    end subroutine get_Na_h

    subroutine get_K_n(v, a, b)
        double precision, intent(in) :: v
        double precision, intent(out) :: a, b
        double precision :: v2
        v2 = v - vtraub;
        a = 0.032d0 * (15.0d0-v2)/( exp((15.0d0-v2)/5.0d0) - 1.0d0)
        b = 0.5d0 * exp((10.0d0-v2)/40.0d0);
    end subroutine get_K_n

    subroutine get_TCa_m(v, tau, inf)
        double precision, intent(in) :: v
        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
```

```
            c = Iinj
        end if

        v = y(1)

        INa = GNa * y(2)**3 * y(3) * (v-ENa)
        IK  = GK * y(4)**4 * (v-EK)
        ITCa = GTCa * y(5)**2 * y(6) * (v - ECa)

        dy(1) = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa )
        call get_Na_m(v, a, b)
        dy(2) = a - (a+b) * y(2)

        call get_Na_h(v, a, b)
        dy(3) = a - (a+b) * y(3)

        call get_K_n(v, a, b)
        dy(4) = a - (a+b) * y(4)

        call get_TCa_m(v, tau, inf)
        dy(5) = (inf - y(5)) / tau

        call get_TCa_h(v, tau, inf)
        dy(6) = (inf - y(6)) / tau
    end subroutine rhs

end program cellpot
```

H.10 ex10

`mtfort90.f90` is the same as in `ex05`.

```
! release.m
! Stochastic neurotransmitter release: ex12.m
! Ralf Schneggenburger & Erwin Neher
! Nature 406, 889-893, 2000}

program release
    use mtmod
    implicit none

    double precision, parameter :: ttime = 4.0d0   ! total time
    double precision, parameter :: dt    = 0.02d0  ! time step
    double precision, parameter :: peak  = 20.0d0  ! peak calcium concentration
    double precision, parameter :: hw    = 1.0d0   ! half width of calcium transient
    integer, parameter :: iter  = 100000           ! number of trials

! kinetic constants
    double precision, parameter :: kon    = 9.0d-2 ! microM^{-1} ms^{-1}
    double precision, parameter :: koff   = 9.5d0  ! ms^{-1}
    double precision, parameter :: beta   = 0.25d0 ! cooperativity
    double precision, parameter :: gamma  = 6.0d0  ! ms^{-1}
    double precision, parameter :: epsilon= 0.3d0  !  ms^{-1}
    integer, parameter :: initialState = 1

    double precision, parameter :: konh   = dt * kon;
    double precision, parameter :: koffh  = dt * koff;
    double precision, parameter :: gammah = dt * gamma;
    double precision, parameter :: epsilonh = dt*epsilon;
    double precision, parameter :: beta2  = beta*beta;
    double precision, parameter :: beta3  = beta*beta2;
    double precision, parameter :: beta4  = beta*beta3;

    integer, parameter :: npts = floor(ttime/dt)+1
    integer, dimension(npts) :: sum = 0
    double precision, dimension(npts) :: ca = 0.0d0
    integer :: i, k, counter, s, snext, seed
    double precision :: d, alpha, r, th

```

```
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
```

```
                        th = th + koffh * beta4
                        if(r < th)then
                            snext = 5
                        end if
                    end if
                case (7 )      ! release
                    snext = 0
                    sum(i) = sum(i) + 1
                    counter = counter + 1
                case (0)       !recovery from release
                    th = epsilonh
                    if(r < th)then
                        snext = 1
                    end if
            end select
            s = snext
        end  do ! end of i-loop
    end do ! end of k-loop

    write(*, fmt='("release probability = ", F10.5)') dble(counter)/dble(iter)


    open(unit = 11, file='ax.txt', form='formatted', access='stream', status='replace')
    do i = 1, npts
        write(unit=11, fmt='(3E15.7)') dt*i, ca(i), dble(sum(i))
    end do
    close(unit=11)
    call system("../../tools/gc ax.txt ");

end program release
```

```
fc     = gfortran
cflags  = -O3
lflags  =
exec = release
objs = mtfort90.o release.o

all: $(exec)
$(exec): $(objs)
        $(fc) $(lflags) $(objs) -o $@
%.o: %.f90
        $(fc) -c $(cflags) $<
clean:
        -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 Exploler (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).