講義参考資料

総合研究大学院大学 生命科学研究科
井本 敬二

version 08-Feb-2006

第 I 部
開発環境の準備

1 はじめに

昔と異なり、現在は計算環境としていろいろなツールがある。電気生理のデータの整理・解析には、Igor Pro、Mathematica、Matlab、Maple、Origin といったソフトが使用可能である。実際これらのソフトのあるものでは、常微分方程式を解析的に解いたり、数値計算を行ったりすることが可能である。従って小規模なシミュレーション計算を行う場合に、C/C++ といったプログラム言語を用いてプログラムを書くという必要はない。しかしシミュレーションの規模が大きくなってくると、計算速度が重要な要素となり、計算速度を上げるためには、C/C++ などのプログラム言語でプログラムを開発することが必要となってくる。

実際に C/C++ でプログラムを書く必要があるのは、多くのコンポーネント（細胞など）を想定する場合や、モンテカルロ法などの乱数を用いた計算の場合などが考えられる。

2 開発環境のテスト

最も有名な C 言語のプログラム。

```
//------------------------
// program rk01

#include <stdio.h>
int main(){
    printf("Hello, World!\n");
    return 0;
}
//------------------------
```

[説明]

はじめの行の、#include <stdio.h>は、コンパイルするときに、stdio.h という名前のファイルを含めるという意味。（h ファイルは、ヘッダーファイルと呼ばれる。）stdio は standard input/output の意味で、標準入出力のこと。stdio.h には、コンソール・キーボードからの入出力、ファイルの入出力などの基本的なことが含まれている。

プログラムには、main と呼ばれるところが 1 つある。プログラムはここから始まる。

printf() はコンソールへの出力。” ”で囲まれた部分は、文字列を示す。\n は改行を示す文字。（日本語環境では、\ は円マーク¥で表示される。）関数は、たとえ引数がなくても () が必要。他の言語では、引数がない場合には () が不要なものもある。見方を変えれば、() が付いていれば、それが関数であるということ。配列に ( ) ではなく、[ ] を用いる。

プログラムを書く場合、statement の最後には';' が必要。改行、タブは必要ではない。但し、見易くしておく事は、保守上重要である。

なお、ここで書かれている例題は、大型コンピュータなどの他のシステムとの互換性を重視し、新しい言語仕様を使うことは出来るだけ避けている。

3 簡単なプログラム

```
//------------------------
// program rk02

#include <stdio.h>
#include <math.h>
int main(){
    int i;
    double x, y;
    for(i=0;i<10;i++){
        x = 0.1 * i;
        y = sin(x);
        printf("%d: x = %lf, y = %lf\n", i, x, y);
    }
    return 0;
}
//------------------------
```

[説明]

三角関数が用いられるため、#include <math.h>が加えられている。

プログラムにおいて、$A = B$; は、B の値を A に代入することを意味する。等しいという表現には==、等しくないという表現には!=が用いられる。

int、double は変数のタイプ。基本的なタイプとしては、他に、float、char などがある。通常、int は 32 ビットの整数、double は 64 ビットの実数、float は 32 ビットの実数。計算速度の面からは、double と float には差がありません。メモリーのスペースは 2 倍を要す。

プログラムの流れを制御する方法にループ、分岐などがある。ループには、for、while が、分岐には、if、if-else、switch などが用いられる。for(初期条件; 続ける条件; ループが一回終わった後の処理) という形で、i++ は、i = i+1 の意味。

printf() のなかの %d、%lf は、それぞれ int、double を文字列に変換することを意味している。数値と文字列は異なることに注意。

4　テキストファイルへの書き出し

```
//------------------------
// program rk03
#include <stdio.h>
#include <math.h>
int main(){
    FILE* fd;
    int i;
    double x, y;

    fd=fopen("rkdat.txt","wt");
    for(i=0;i<10;i++){
        x = 0.1 * i;
        y = sin(x);
        printf("%d: x = %lf, y = %lf\n", i, x, y);
        fprintf(fd, "%d: x = %lf, y = %lf\n", i, x, y);
    }
    fclose(fd);
    return 0;
}
//------------------------
```

[説明]

ファイル関係のことは、なかなか難しい。最初は雛形の真似をするとよい。FILE* fd とあるのは、FILE へのポインタとして fd を定義するという意味。ファイルをテキストモード開けて、ファイルに書き込み、ファイルを閉じている。

fprintf() は、ファイルへの書き出しであるが、実は printf() は、fprintf() で書き出し先を stdout としたものである。stdout は、stdio.h で定義されている。

5　バイナリーファイルへの書き出し

```
//------------------------
// program rk04
#include <stdio.h>
#include <math.h>
int main(){
    FILE* fd;
    int i;
```

```
    double x, y;
    double a[2];

    fd=fopen("rkdat.dat","wb");
    for(i=0;i<100;i++){
        x = 0.1 * i;
        y = sin(x);
        a[0]=x;
        a[1]=y;
        printf("%d: x = %lf, y = %lf\n", i, x, y);
        fwrite(a, sizeof(double),2,fd);
    }
    fclose(fd);
    return 0;
}
//------------------------
```

[説明]

double a[2] は、double の配列を宣言している。配列要素は 0 より始まり、要素の個数は 2 である。従って a[2] は存在しないことに注意。

ファイルをバイナリーモードで開いている。この場合の書き込みには fwrite() を用いる。

sizeof() はタイプのサイズを返すマクロで、8 と書かずこのように書くことにより、システム間での互換性が保たれる（ソースコードレベルでの）。

6 グラフを書くアプリを走らせる

```
//------------------------
// program rk05
#include <stdio.h>
#include <math.h>
#include <process.h>
int main(){
    FILE* fd;
    int i;
    double x, y;
    double a[2];

    fd=fopen("rkdat.dat","wb");
    for(i=0;i<100;i++){
        x = 0.1 * i;
```

```
        y = sin(x);
        a[0]=x;
        a[1]=y;
        printf("%d: x = %lf, y = %lf\n", i, x, y);
        fwrite(a, sizeof(double),2,fd);
    }
    fclose(fd);
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "1", NULL);
    return 0;
}
//------------------------
```

[説明]

_spawnl() という関数を用いて、プログラム中から他のプログラムを走らせている。この関数は、process.h で定義されている。この部分は、Windows に特有の表現。Unix/Linux では、system(”gc rkdata.dat 1”) といった表現をするのが普通。以前の Microsoft C++ では、system() という関数があったと思うが、今はなくなっている。

gc.exe は、自家製の簡易グラフソフト。データファイル名とトレースの本数を引数として入れる。間違えるとハングアップする。

7 複数のトレースを描く

//————————— // program rk06

```
#include <stdio.h>
#include <math.h>
#include <process.h>
int main(){
    FILE* fd;
    int i;
    int n = 1000;
    double x;
    double a[4];

    fd=fopen("rkdat.dat","wb");
    for(i=0;i<n;i++){
        x = 0.1 * i;
        a[0]=x;
        a[1]=5*sin(x/7);
        a[2]=2*cos(3*x);
        a[3]=a[1]+a[2];
```

```
        fwrite(a, sizeof(double),4,fd);
    }
    fclose(fd);
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "3", NULL);
    return 0;
}
//------------------------
```

参考 — Visual C++ 2005 Express Edition の使い方 —

Visual C++ 2005 Express Edition は、制限があるものの free でダウンロードし、プログラムを作成することが出来る開発環境。上位の開発環境である VisualStudio 2005 には、大別して 3 つのプログラム開発環境がある。

1. Win32: 他のコンピュータシステムとの互換性が最も保たれている環境。ほぼ ANSI C++。
2. MFC (Microsoft Foundation Class): 従来、ウィンドウ関係のプログラムに用いられてきた環境。比較的わずかのコーディングで、かなりのプログラムを書くことができるのが特徴。ただし使えるようになるまでには、かなりのトレーニングが必要。他のコンピュータシステムとの互換性はない。
3. CLR (Common Language Runtime): .net framework と呼ばれていた環境。Visual Basic、C# (C-sharp)、J#といった他の言語と共通のライブラリを使用するようになっている。ネット環境とデータベースの使用を考慮して作られたプログラム環境。C++ に関しては大幅に言語仕様が拡張されており、ポインタの使用を避けハンドルを使うようになっているらしい。グラフィック環境あり。他のコンピュータシステムとの互換性なし。

従って、単独のコンピュータで走らせる数値計算のプログラムを書く場合、Win32 コンソールという環境が一番他のコンピュータシステムと互換性がある環境と言うことになる。この様な開発環境としては、Windows で他のコンパイラを使用する（たとえば Intel C++ コンパイラ)、Linux を使う、計算サーバの UNIX システムを使うといったことが考えられるが、Microsoft の開発環境で一旦書いて、その後、移植する方が便利であろうと思う。

Visual C++ Express Edition では、MFC の部分がサポートされていない。またリソースを編集する機能も無い様です。Win32 プログラムの開発に関しては、制限はなさそうです。

さて Visual C++ 2005 の使い方。

1. まずは Visual C++ 2005 Express Edition を起動。大きいプログラムなのか、起動に比較的時間がかかる。[ファイル] ⇒ [新規作成]⇒ [プロジェクト] で、新しいプロジェクトを作成するディアログが開く。
2. プロジェクトの種類では、”Win32”を選び、テンプレートでは、”Win32 コンソール　アプリケーション”（これしかないはず）を選択。適当なプロジェクト名を入れ、適当な場所を指定する。ソリューションのディレクトリを作成のチェックは入れないほうがよい（チェックすると、もう一段階フォルダができる)。そして”OK”をクリック。
3. Win32 アプリケーション ウィザードのディアログが開く。初めのページでは選択事項なし。”次へ”をクリック。”コンソール アプリケーション”と”空のプロジェクト”をチェックして、”完了”をクリック。ソリューション エクスプローラという名前のウインドウに、プロジェクト名のついたフォルダ・ファイル

のツリーが現れる。まだファイルは作成されていない。

4. ソリューション エクスプローラのソースファイルの部分を右クリックし、[追加] → [新しい項目]。新しい項目の追加のディアログが現れる。ソースファイルなので、C++ ファイル (cpp) を選択。適当な名前を入れる。main() のファイルは、プロジェクト名と同じにしておくとわかりやすい。そして [追加] をクリック。
5. ソースコードを書いて、ビルド。ヘッダーファイルを変更した場合には、リビルドを行ったほうが無難。
6. ”デバッグ開始”、または”デバッグなしで開始”

第 II 部
常微分方程式の数値解

8　常微分方程式の数値解

様々な現象を記載する場合、微分方程式を用いて表すことが多い。微分方程式を解析的に解くことが好ましいが、実際的には解析的の解くことが出来る場合は少なく、そのような場合、数値計算によらなくてはならない。また、いろいろな条件が時間的に変化していくので、その意味でも数値計算で解を求める技法は重要である。微分方程式のなかでも独立変数が一つのものを常微分方程式という。

関数 y が、

$$\frac{dy(x)}{dx} = f(x, y) \tag{1}$$

で書き表される場合、first-order differential equation という。連立の場合は、

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, y_2, \ldots, y_N), \qquad i = 1, \ldots N \tag{2}$$

N-coupled first-order differential equation という。注意すべきことは、左辺は微分の形であるが、右辺は現在の値であり、右辺は計算可能な部分である。右辺を Right-handed side と言うらしく、rhs という略称が出てくる場合がある。

最も単純な計算方法は、Euler method と呼ばれるものであり、

$$\frac{dy(x)}{dx} = f(x, y) \tag{3}$$

$$y(x + h) = y(x) + hf(x, y) \tag{4}$$

で表される。この方法は誤差が蓄積していきやすいという欠点を持つ。誤差を少なくするようにいろいろな工夫がされている。代表的なものが Runge-Kutta method と呼ばれるものである

9　Second-order Runte-Kutta method

midpoint method とも呼ばれる。

$$\frac{dy(x)}{dx} = f(x, y) \tag{5}$$

とした場合、

$$k_1 = hf(x, y) \tag{6}$$

$$k_2 = hf(x + \frac{1}{2}h, y + \frac{1}{2}k_1) \tag{7}$$

$$y(x + h) = y(x) + k_2 + O(h^3) \tag{8}$$

10 Fourth-order Runte-Kutta method

$$k_1 = hf(x, y) \tag{9}$$
$$k_2 = hf(x + \frac{1}{2}h, y + \frac{1}{2}k_1) \tag{10}$$
$$k_3 = hf(x + \frac{1}{2}h, y + \frac{1}{2}k_2) \tag{11}$$
$$k_4 = hf(x + h, y + k_3) \tag{12}$$
$$y(x + h) = y(x) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5) \tag{13}$$

連立の場合は、

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, y_2, \ldots, y_N) \tag{14}$$
$$k_{1,i} = hf_i(x, y_1, y_2, \ldots, y_N) \tag{15}$$
$$k_{2,i} = hf_i(x + \frac{1}{2}h, y_1 + \frac{1}{2}k_{1,1}, y_2 + \frac{1}{2}k_{1,2}, \ldots, y_N + \frac{1}{2}k_{1,N}) \tag{16}$$
$$k_{3,i} = hf_i(x + \frac{1}{2}h, y_1 + \frac{1}{2}k_{2,1}, y_2 + \frac{1}{2}k_{2,2}, \ldots, y_N + \frac{1}{2}k_{2,N}) \tag{17}$$
$$k_{4,i} = hf_i(x + h, y_1 + k_{3,1}, y_2 + k_{3,2}, \ldots, y_N + k_{3,N}) \tag{18}$$
$$y_i(x + h) = y_i(x) + \frac{k_{1,i}}{6} + \frac{k_{2,i}}{3} + \frac{k_{3,i}}{3} + \frac{k_{4,i}}{6} + O(h^5) \tag{19}$$

11 Runge-Kutta method の例 1

12 Runge-Kutta method の例 2

2 次の常微分方程式

$$\frac{d^2y(x)}{dx^2} = -y(x) \tag{20}$$
$$y(0) = 0 \tag{21}$$
$$y'(0) = 1 \tag{22}$$

を解く。このような 2 次の常微分方程式を解く定石的な手法として、次数を下げて連立一次にする方法がよく知られている。

$$y' = z \tag{23}$$

と置くと、

$$y' = z \tag{24}$$
$$z' = -y \tag{25}$$

という連立一次常微分方程式として表すことができる。

プログラム用に表記に配列を用い、$y \Rightarrow y[0]$、$z \Rightarrow y[1]$ とする。

$$y[0]' = y[1] \tag{26}$$
$$y[1]' = -y[0] \tag{27}$$

```
//------------------------
// program rk08
//
#include <stdio.h>
#include <process.h>
int main(){
    FILE* fd;
    int i;
    int j;
    double x;
    double h;
    double y[2];
    double yn[2];
    double k1[2];
    double k2[2];
    double k3[2];
    double k4[2];
    double a[3];
//初期条件
    h=0.02;
    x=0;
    y[0] = 0;
    y[1] = 1;

    fd=fopen("rkdat.dat","wb");
    for(j=0;j<1000;j++){
        x = h * j;
        a[0] = x;
        a[1] = y[0];
        a[2] = y[1];

        fwrite(a, sizeof(double),3,fd);

        k1[0] = y[1];
        k1[1] = -y[0];
```

```
        k2[0] = y[1] + h*k1[1]/2;
        k2[1] = -(y[0] + h*k1[0]/2);

        k3[0] = y[1] + h*k2[1]/2;
        k3[1] = -(y[0] + h*k2[0]/2);

        k4[0] = y[1] + h*k3[1];
        k4[1] = -(y[0] + h*k3[0]);

        y[0] += h*(k1[0] + 2*k2[0] + 2*k3[0] + k4[0])/6;
        y[1] += h*(k1[1] + 2*k2[1] + 2*k3[1] + k4[1])/6;
    }
    fclose(fd);
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "2", NULL);
    return 0;
}
//------------------------
```

13 関数を使って整理

```
//------------------------
// program rk09
//
#include <stdio.h>
#include <process.h>
void rhs(double x, double *yt, double *k)
{
    k[0] = yt[1];
    k[1] = -yt[0];
}

int main(){
    FILE* fd;
    int i, j;
    double x, h;
    double y[2], yn[2], k1[2], k2[2], k3[2], k4[2];
    double a[3];
```

```
    h=0.02;
    x=0;
    y[0] = 0;
    y[1] = 1;

    fd=fopen("rkdat.dat","wb");
    for(j=0;j<1000;j++){
        x = h * j;
        a[0] = x;
        a[1] = y[0];
        a[2] = y[1];
        fwrite(a, sizeof(double),3,fd);

        rhs(x, y, k1);
        for(i=0;i<2;i++) yn[i] = y[i] + h*k1[i]/2;
        rhs(x+h/2,yn, k2);
        for(i=0;i<2;i++) yn[i] = y[i] + h*k2[i]/2;
        rhs(x+h/2,yn, k3);
        for(i=0;i<2;i++) yn[i] = y[i] + h*k3[i];
        rhs(x+h, yn, k4);
        for(i=0;i<2;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
    }
    fclose(fd);
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "2", NULL);
    return 0;
}
//-----------------------------------------------------
```

[説明] 関数 void rhs(double x, double *yt, double *k) は、void rhs(double x, double yt[], double k[]) という書き方もある。どちらも配列の先頭のアドレス（ポインタ）を渡している。C/C++ のプログラムでは、配列の大きさはチェックされない。配列の大きさ以上のところをアクセスすると予期不能の状態となり、プログラムがフリーズする可能性が高い。

void は関数が値を返さないことを示している。

14 動的な配列

```
//------------------------
// program rk10
//
#include <stdio.h>
```

```
#include <process.h>
void rhs(double x, double yt[], double k[])
{
    k[0] = yt[1];
    k[1] = -yt[0];
}
void rk4(int n, double x, double h, double* y, double* yn, double* k1, double* k2,
         double* k3, double* k4)
{
    int i;
    rhs(x, y, k1);
    for(i=0;i<n;i++) yn[i] = y[i] + h*k1[i]/2;
    rhs(x+h/2,yn, k2);
    for(i=0;i<n;i++) yn[i] = y[i] + h*k2[i]/2;
    rhs(x+h/2,yn, k3);
    for(i=0;i<n;i++) yn[i] = y[i] + h*k3[i];
    rhs(x+h, yn, k4);
    for(i=0;i<n;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
}
int main(){
    FILE* fd;
    int i, j;
    int n = 2;
    double x, h;
    double y[2];
    double a[3];
    double *yn, *k1, *k2, *k3, *k4;

    yn = new double[n];
    k1 = new double[n];
    k2 = new double[n];
    k3 = new double[n];
    k4 = new double[n];

    h=0.02;
    x=0;
    y[0] = 0;
    y[1] = 1;
    fd=fopen("rkdat.dat","wb");
    for(j=0;j<1000;j++){
```

```
        x = h * j;
        a[0] = x;
        a[1] = y[0];
        a[2] = y[1];
        fwrite(a, sizeof(double),3,fd);
        rk4(n, x, h, y, yn, k1, k2, k3, k4);
    }
    fclose(fd);
    delete [] yn;
    delete [] k1;
    delete [] k2;
    delete [] k3;
    delete [] k4;
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "2", NULL);
    return 0;
}
//-----------------------------------------------------
```

[説明] 配列の大きさがどれだけいるかは、前もってわからない場合が多い。その場合には、プログラムの途中で配列をつくる。プログラムが終わるときには配列を廃棄する必要がある。new — delete という方法は、C++ の仕様である。C 言語の場合は、malloc() と free() を用いる。

関数化することにより、main() の内容がすっきりしたことがわかる。

15 structure

上の例では、rk4() への関数が数多く厄介。これらの変数を一まとめにして扱う。

```
//------------------------
// program rk11
//
#include <stdio.h>
#include <process.h>
struct RKData{
    int n;
    double x;
    double h;
    double* y;
    double* yn;
    double* k1;
    double* k2;
    double* k3;
```

```
    double* k4;
};

void rhs(double x, double yt[], double k[]){
    k[0] = yt[1];
    k[1] = -yt[0];
}
void rk4(RKData* a){
    int i;
    rhs(a->x, a->y, a->k1);
    for(i=0;i<a->n;i++) a->yn[i] = a->y[i] + a->h*a->k1[i]/2;
    rhs(a->x+a->h/2,a->yn, a->k2);
    for(i=0;i<a->n;i++) a->yn[i] = a->y[i] + a->h*a->k2[i]/2;
    rhs(a->x+a->h/2,a->yn, a->k3);
    for(i=0;i<a->n;i++) a->yn[i] = a->y[i] + a->h*a->k3[i];
    rhs(a->x+a->h, a->yn, a->k4);
    for(i=0;i<a->n;i++) a->y[i] += a->h*(a->k1[i] + 2*a->k2[i] + 2*a->k3[i] +  a->k4[i])/6;
}
int main(){
    FILE* fd;
    RKData rkdat;
    int i,j;
    double y[2];
    double a[3];
    double *yn, *k1, *k2, *k3, *k4;
    rkdat.n = 2;
    rkdat.h = 0.02;
    rkdat.y = y;
    rkdat.yn = new double[rkdat.n];
    rkdat.k1 = new double[rkdat.n];
    rkdat.k2 = new double[rkdat.n];
    rkdat.k3 = new double[rkdat.n];
    rkdat.k4 = new double[rkdat.n];
    rkdat.x=0;
    y[0] = 0;
    y[1] = 1;
    fd=fopen("rkdat.dat","wb");
    for(j=0;j<1000;j++){
        rkdat.x = rkdat.h * j;
        a[0] = rkdat.x;
```

```
        a[1] = y[0];
        a[2] = y[1];
        fwrite(a, sizeof(double),3,fd);
        rk4(&rkdat);
    }
    fclose(fd);
    delete [] rkdat.yn;
    delete [] rkdat.k1;
    delete [] rkdat.k2;
    delete [] rkdat.k3;
    delete [] rkdat.k4;
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "2", NULL);
    return 0;
}
//------------------------------------
```

16 クラス化 1

struct はデータの集まりであり、それ自体が何かをするということは出来ない。言わば名詞の集まりといったところである。class は、struct に動詞の機能を持たせたものと言っても良いと思う。class は、いわゆる OOP (Object-Oriented Program) の基本となる考え方である。テクニカルに理解しにくいが、一旦マスターすると、非常に便利な手法である。これまでに C++ を OOP 的に用いた経験がなければ、以下のコードを理解することは出来ないと思うが、そっくり移して動かしてみれば、実用上どの部分が大切かわかってくると思う。

5 つのファイルに分割した。

```
//------------------------
//     RKData.h
//
class CRKData{
public:
    CRKData();
    ~CRKData();
private:
    int n;
    double x;
    double h;
    double* y;
    double* yn;
    double* k1;
```

```
    double* k2;
    double* k3;
    double* k4;
public:
    void initState(int ndim, double dx, double* yd);
    void rk4(double xd);
    virtual void rhs(double x, double* yt, double* k){};
};

//------------------------
//     RKData.cpp
//
#include "RKData.h"

CRKData::CRKData()
{
}

CRKData::~CRKData()
{
    delete [] yn;
    delete [] k1;
    delete [] k2;
    delete [] k3;
    delete [] k4;
}
void CRKData::initState(int ndim, double dx, double* yd)
{
    n = ndim;
    h = dx;
    y  =  yd;
    yn =  new double[n];
    k1 =  new double[n];
    k2 =  new double[n];
    k3 =  new double[n];
    k4 =  new double[n];
}

void CRKData::rk4(double xd)
{
```

```
    int i;
    x = xd;
    rhs(x, y, k1);

    for(i=0;i<n;i++) yn[i] = y[i] + h*k1[i]/2;
    rhs(x+h/2,yn, k2);

    for(i=0;i<n;i++) yn[i] = y[i] + h*k2[i]/2;
    rhs(x+h/2,yn, k3);

    for(i=0;i<n;i++) yn[i] = y[i] + h*k3[i];
    rhs(x+h, yn, k4);

    for(i=0;i<n;i++) y[i] += h*(k1[i] + 2*k2[i] + 2*k3[i] + k4[i])/6;
}

//------------------------
//     solveODE.h
//
class CRKData;

class CsolveODE : public CRKData
{
public:
    CsolveODE();
    ~CsolveODE();
public:
    void rhs(double x, double* y, double* dy);
};

//------------------------
//     solveODE.cpp
//
#include "RKData.h"
#include "solveODE.h"
CsolveODE::CsolveODE()
{
}
CsolveODE::~CsolveODE()
{
```

```
}
void CsolveODE::rhs(double x, double* y, double* dy)
{
    dy[0] = y[1];
    dy[1] = -y[0];
}

//------------------------
//     Program rk12.cpp
//
#include <stdio.h>
#include <process.h>
#include "RKData.h"
#include "solveODE.h"
int main(){
    FILE* fd;
    int i, n;
    double x, h;
    double y[2];
    double a[3];

    CsolveODE de;

    n=2;
    h=0.02;
    de.initState(n,h,y);
    y[0] = 0;
    y[1] = 1;

    fd=fopen("rkdat.dat","wb");

    for(i=0;i<1000;i++){
        x = h*i;
        a[0] = x;
        a[1] = y[0];
        a[2] = y[1];
        fwrite(a, sizeof(double),3,fd);
        de.rk4(x);
    }
    fclose(fd);
```

```
    _spawnl(_P_NOWAIT, "gc", "gc", "rkdat.dat", "2", NULL);
    return 0;
}

//---------------------------
```

[説明]
異なる微分方程式を扱うには、solveODE の rhs() のところを変えるだけよい。class CRKData を変更する必要は無い。main() を含むプログラムがかなり単純になった。

17 クラス化 2

メインの部分のデータの出力、表示の部分をクラス化した。SolveODE.(h,cpp)、RKData.(h,cpp) は変わり無し。

```
//-----------------------------------------
//       DataRec.h
//

class CDataRecorder
{
public:
    CDataRecorder();
    ~CDataRecorder();
    bool Start(int n, double*y);
    bool Start(int n, double*y, char* fname);
    void StoreData(double x);
    void Stop();
    void Replay();
private:
    FILE* fd;
    int nt;
    int m;
    double* py;
    double* a;
    char filename[128];
private:
    bool Start0();
};

//-----------------------------------------
```

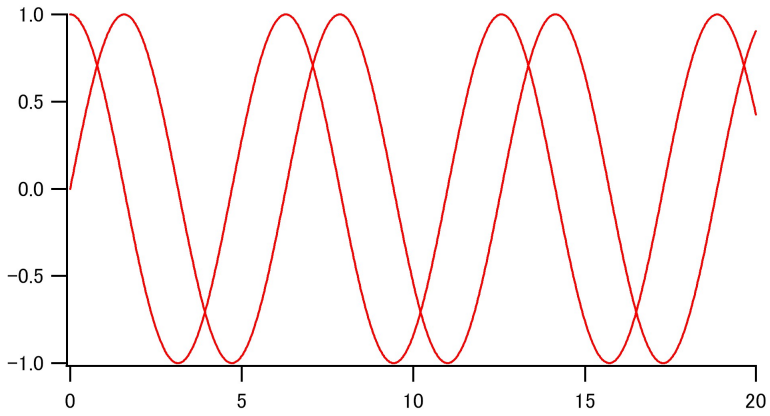
```
//       DataRec.cpp
//
#include <stdio.h>
#include <process.h>
#include <string.h>
#include "DataRec.h"

CDataRecorder::CDataRecorder(){}
CDataRecorder::~CDataRecorder(){}
bool CDataRecorder::Start0()
{
    m = nt + 1;
    a = new double[m];
    fd=fopen(filename,"wb");
    if(fd==NULL) return false;
    return true;
}
bool CDataRecorder::Start(int n, double *y)
{
    nt = n;
    py = y;
    strcpy(filename, "rkdat.dat");
    return Start0();
}
bool CDataRecorder::Start(int n, double*y, char* fname)
{
    nt = n;
    py = y;
    strcpy(filename,fname);
    return Start0();
}
void CDataRecorder::StoreData(double x){
    double* p;
    a[0] = x;
    p = py;
    for(int i=0;i<nt;i++){
        a[i+1] = *p++;
    }
    fwrite(a, sizeof(double),m,fd);
}
```

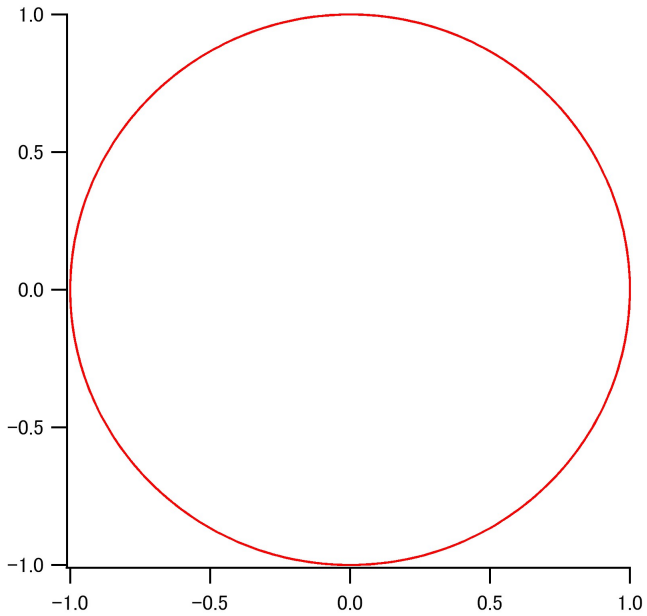
```
void CDataRecorder::Stop()
{
    fclose(fd);
    delete [] a;
}
void CDataRecorder::Replay()
{
    char s[32];
    sprintf(s, "%d", nt);
    _spawnl(_P_NOWAIT, "gc", "gc", filename, s, NULL);
}

//-----------------------------------------
//       program rk13
//
#include <stdio.h>
#include "RKData.h"
#include "solveODE.h"
#include "DataRec.h"
int main()
{
// initial conditions
    double y[] = {0, 1.0};
    double h = 0.02;
    int iteration = 1000;
//-------------------------------
    int n = sizeof(y)/sizeof(double);
    int i;
    double x;

    CsolveODE de;
    CDataRecorder rec;

    de.initState(n,h,y);
    rec.Start(n,y);

    for(i=0;i<iteration;i++){
        x = h*i;
        rec.StoreData(x);
        de.rk4(x);
```

```
    }
    rec.Stop();
    rec.Replay();
    return 0;
}
```

[説明]

main() の部分の見通しが随分よくなった。違う微分方程式の計算をする場合に変更しなくてはならないのは、main() の初期条件の部分、SolveODE.cpp の微分方程式の定義の部分だけである。





参考 — Unix/Linux の場合 —

何らかの理由で、Unix/Linux でプログラムを走らせたい場合があるかもしれない。変更が必要なところは、DataRec.cpp のグラフを描く Replay() のところだけである。参考までに、Linux で gnuplot を用いてグラフを書く場合は次のようにすればよい。

```
// DataRec.cpp (Linux version) Replay() の部分のみ
//
void CDataRecorder::Replay(){
    FILE* fdt;
    int i;
// create a gnuplot command file
    fdt = fopen("$cmd.txt","wt");
    fprintf(fdt, "plot ");
    for(i=2;i<m;i++){
        fprintf(fdt,"'$data.txt' using 1:%d with lines, ", i);
    }
    fprintf(fdt,"'$data.txt' using 1:%d with lines\n", m);
    fclose(fdt);
// create a gnuplot data file
    a = new double[m];
    fd=fopen(filename,"rb");
    fdt = fopen("$data.txt", "wt");
    while(1){
        if(fread(a, sizeof(double), m, fd)==0) break;
        for(i=0;i<nt;i++){
            fprintf(fdt,"%g\t",a[i]);
        }
        fprintf(fdt,"%g\n",a[nt]);
    }
    fclose(fd);
    fclose(fdt);
    delete [] a;
// run gnuplot
    system("gnuplot -persist \\$cmd.txt");
}
```

ついでに makefile も載せておく。

```
CC = g++
TARGET = rk
CFLAGS = -c -O3
LFLAGS = -lm
```

```
OBJS = rk.o DataRec.o RKData.o solveODE.o

$(TARGET):        $(OBJS)
        $(CC) -o $(LFLAGS) $(OBJS) -o $(TARGET)
.cpp.o:
        $(CC) $(CFLAGS) $<
```

第 III 部
いろいろな常微分方程式

18　van der Pol 方程式

$$\frac{d^2x}{dt^2} + (x^2 - 5)\frac{dx}{dt} + 9x = 0 \tag{28}$$

は、van der Pol equation と呼ばれ、心臓のリズムの初めての数学的モデルである。初期条件 $t = 0$ の時、$x(0) = 1$、$dx/dt = 0$ として、数値解を求めてみよう。

定法として、$dx/dt = y$ とおきます。そうすると、上記の式は、

$$\frac{dy}{dt} + (x^2 - 5)y + 9x = 0 \tag{29}$$

となる。すなわち、連立の常微分方程式

$$\frac{dx}{dt} = y \tag{30}$$

$$\frac{dy}{dt} = -(x^2 - 5)y - 9x \tag{31}$$

が得られる。プログラムでは、変数を配列で表しているので、$x \Rightarrow y[0]$、$y \Rightarrow y[1]$ とする。また $dt \Rightarrow dx$ とする。

$$\frac{dy[0]}{dx} = y[1] \tag{32}$$

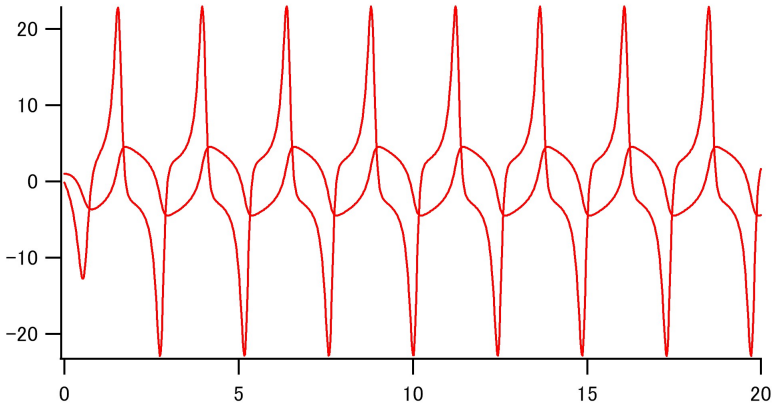
$$\frac{dy[1]}{dx} = -(y[0]^2 - 5)y[1] - 9y[0] \tag{33}$$

この式を、SolveODE.cpp に入れる。

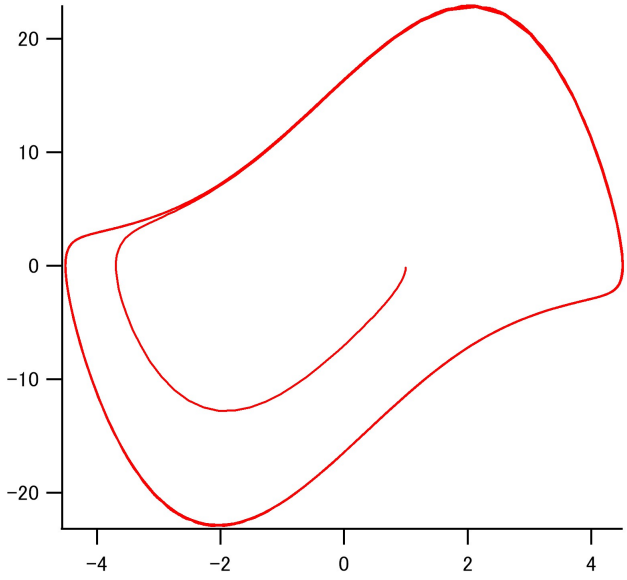
```
//
//  SolveODE.cpp
//
#include "solveODE.h"
CsolveODE::CsolveODE(){}
CsolveODE::~CsolveODE(){}
void CsolveODE::rhs(double x, double* y, double* dy){
    dy[0] = y[1];
    dy[1] = -(y[0]*y[0]-5)*y[1] - 9*y[0];
}
```

また、main() の初期条件を変更する。

```
    y[]={1.0, 0.0};
```



また、y と y' を XY 平面でプロットすると、このようになる。



19 FitzHugh-Nagumo equation

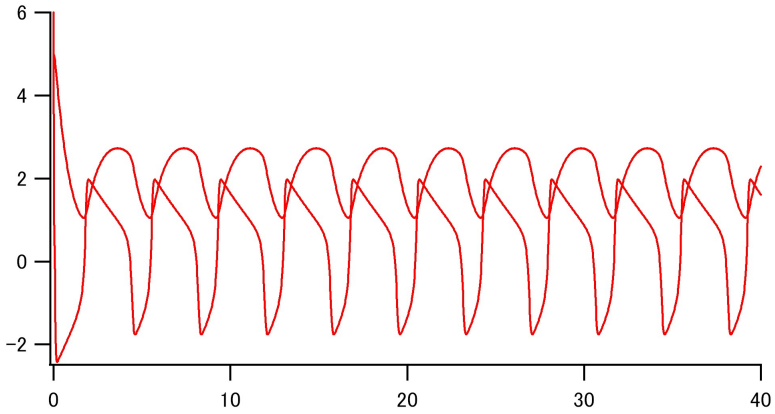
V は電位、R は回復力であり、主に K^+ 電流を反映する。

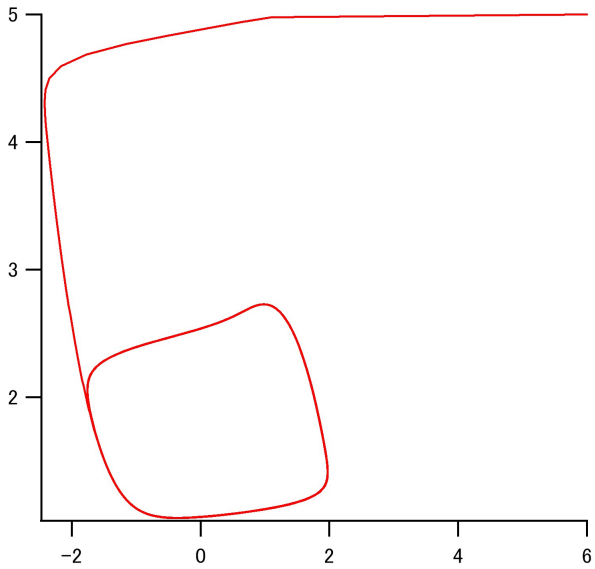
$$\frac{dV}{dt} = 10 * (V - (V^3/3) - R + input) \tag{34}$$

$$\frac{dR}{dt} = 0.8 * (-R + 1.25 * V + 1.5) \tag{35}$$

```
//    V  -> y[0]
//    R  -> y[1]
#include <math.h>
#include "solveODE.h"
```

```
CsolveODE::CsolveODE(){}
CsolveODE::~CsolveODE(){}
void CsolveODE::rhs(double x, double* y, double* dy){
double Input = 2.0;
dy[0] = 10 * (y[0] - pow(y[0],3.0)/3 - y[1]+ Input);
dy[1] = 0.8 * (-y[1] + 1.25*y[0] + 1.5);
}
```





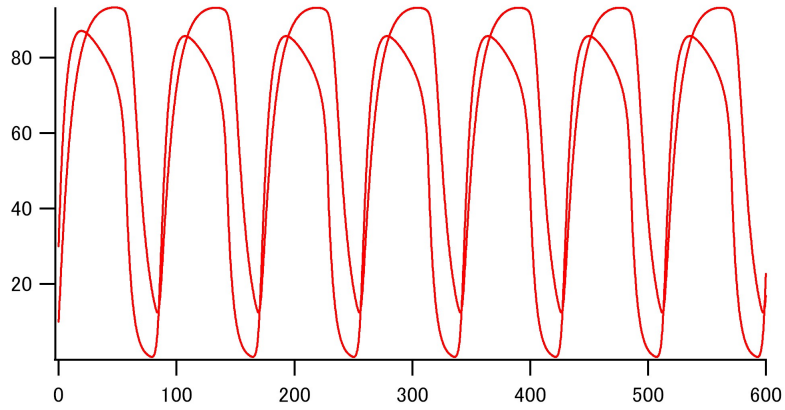
20 Wilson-Cowan network oscillator

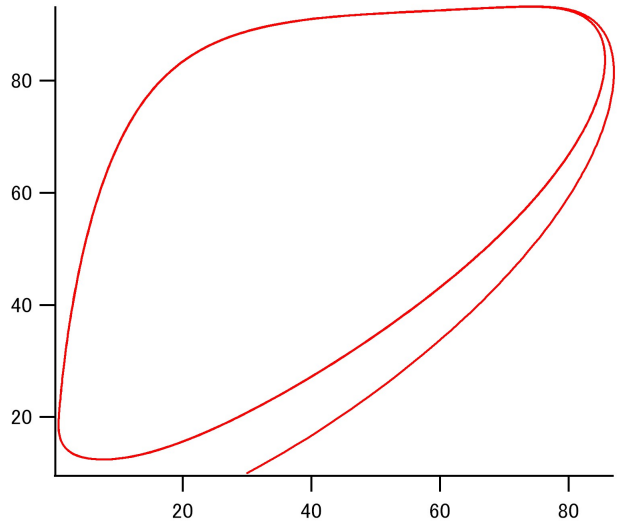
$$\frac{dE}{dt} = \frac{1}{5}(-E + S(1.6E - I + K)) \tag{36}$$

$$\frac{dI}{dt} = \frac{1}{10}(-I + S(1.5E)) \tag{37}$$

$$S(P) = \begin{cases} \frac{MP^N}{\sigma^N + P^N} & forP \geq 0 \\ 0 & forP < 0 \end{cases} \tag{38}$$

$N = 2$、$M = 100$、$\sigma = 30$、$K = 20$。初期条件は、$0 \geq E \geq 100$、$0 \geq I \geq 100$ でいろいろ試してみよ。





21 Rinzel & Lee のモデル

$$\frac{dV}{dt} = -\{17.81 + 47.58V + 33.8V^2\}(V - 0.48) - 26R(V + 0.95) \tag{39}$$

$$-1.93X(1 - 0.5C)(V - 1.4) - 3.25C(V + 0.95) \tag{40}$$

$$\frac{dR}{dt} = \frac{1}{5.6}(-R + 1.29V + 0.79 + 3.3(V + 0.38)^2) \tag{41}$$

$$\frac{dX}{dt} = \frac{1}{30}(-X + 7.33(V + 0.86)(V + 0.84)) \tag{42}$$

$$\frac{dC}{dt} = \frac{1}{100}(-C + 3X) \tag{43}$$

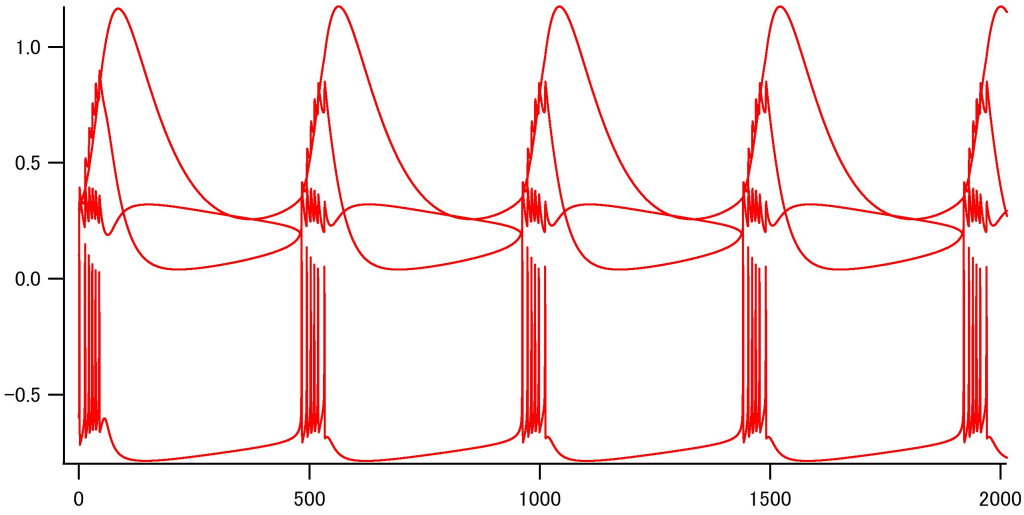
X は Ca^{2+} コンダクタンス、C は細胞内 Ca^{2+} 濃度を意味している。

rk.cpp で y[] の初期値を設定。

```
double y[] = {-0.6, 0.1, 0.1, 0.3};
```

また、solveODE.cpp を変える。

```
//
//  Rinzel & Lee model (1987)
//
//    V  -> y[0]
//    R  -> y[1]
//    X  -> y[2]
//    C  -> y[3]

#include <math.h>
#include "solveODE.h"

CsolveODE::CsolveODE(){}
CsolveODE::~CsolveODE(){}

void CsolveODE::rhs(double x, double* y, double* dy){
    double V = y[0];
    dy[0] = -(17.81+47.58*V + 33.8*V*V)*(V-0.48)-26*y[1]*(V+0.95)
            -1.93*y[2]*(1-0.5*y[3])*(V-1.4)-3.25*y[3]*(V+0.95);
    dy[1] = (-y[1] + 1.29*V + 0.79 + 3.3* (V+0.38)*(V+0.38))/5.6;
    dy[2] = (-y[2]+7.33*(V+0.86)*(V+0.84))/30.0;
    dy[3] = 0.01*(-y[3]+3*y[2]);
}
```



22 Izhikevich のモデル

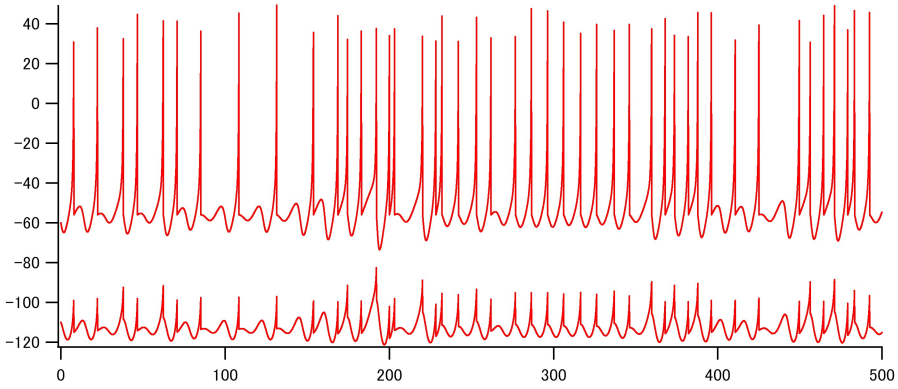
Izhikevich の論文（Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks, 15:1063-1070, 2003）に記載されているモデル。計算量が少ないわりに神経細胞にいろいろな性質を持たせることが出来る。スパイクが発生した後、再分極相の計算を省くことにより計算量を少なくしている。

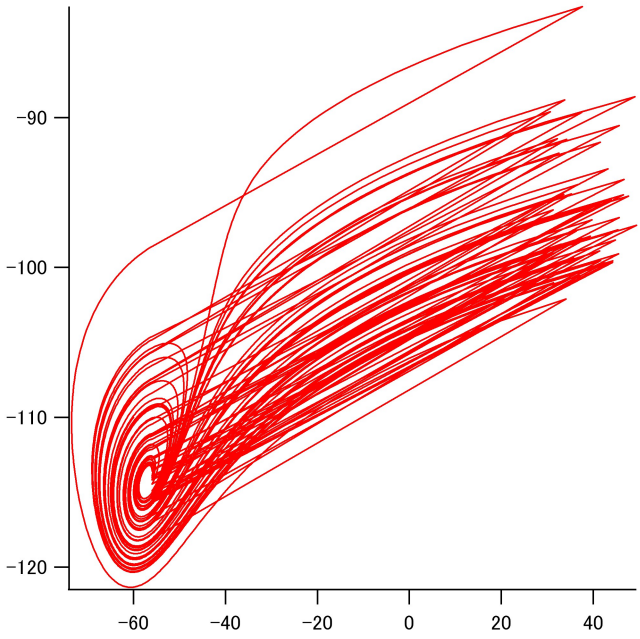
RKData.h で y を private から protected に変更し、CRKData から派生したクラスである CsolveODE からもアクセス可能にした。また CsolveODE で rk4() を overload している。

```
//
// solveODE.h
//
#include "RKData.h"
class CsolveODE : public CRKData
{
public:
    CsolveODE();
    ~CsolveODE();

public:
    double a;
    double b;
    double c;
    double d;
    double I;
```

```
public:
    void rhs(double x, double* yt, double* k);
    void rk4(double x);
};


//
// solveODE.cpp
//

//  Izhikevich model
//     IEEE Transactions on Neural Netowrks, 15:1063-1070, 2003
//    v voltage
//    u recovery variable (mainly reflecting K+ outward current)
//
//    dv/dt = 0.04*v^2 + 5*v + 140 -u + I
//    du/dt = a*(b*v-u)
//    if(v>= 30){v = c; u = u+d;}
//
//    (a, b, c, d)= (0.2, 2, -56, -16)
//    I = -99
//
//    v  -> y[0]
//    u  -> y[1]
#include "solveODE.h"
CsolveODE::CsolveODE()
{
    a = 0.2;
    b = 2;
    c = -56;
    d = -16;
    I = -99;
}
CsolveODE::~CsolveODE(){}
void CsolveODE::rhs(double x, double* y, double* dy)
{
    double v = y[0];
    double u = y[1];
    dy[0]  = 0.04*v*v + 5*v + 140 -u + I;
```

```
    dy[1]   = a*(b*v-u);
}

void CsolveODE::rk4(double xd)
{
    if(y[0]>=30){
        y[0] = c;
        y[1] += d;
    }else{
        CRKData::rk4(xd);
    }
}
```





第 IV 部
Hodgkin-Huxley model

23 m と h

m-gate が open である確率を m、closed である確率を $1-m$ とする。closed $\Rightarrow$ open の rate constant を α、open $\Rightarrow$ closed の rate constant を β とする。,

$$\frac{dm}{dt} = \alpha(1-m) - \beta m \tag{44}$$

十分長い時間が経過した後、dm/dt の値は 0 に近づいていくであろう。この時の m の値を m_∞ とすると、

$$m_\infty = \frac{\alpha}{\alpha+\beta} \tag{45}$$

式 (44) は、解くことが出来て、

$$m = A\exp\left(-\frac{t}{\frac{1}{\alpha+\beta}}\right) + \frac{\alpha}{\alpha+\beta} \tag{46}$$

A は、初期値により定まる値。

$$\tau = \frac{t}{\frac{1}{\alpha+\beta}} \tag{47}$$

とおくと、

$$m = A\exp\left(-\frac{t}{\tau}\right) + m_\infty \tag{48}$$

と書くことが出来る。この式の両辺を微分すると、

$$\frac{dm}{dt} = -\frac{A}{\tau}\exp\left(-\frac{t}{\tau}\right) \tag{49}$$

上二つの式を合わせると、

$$\frac{dm}{dt} = \frac{m-m_\infty}{\tau} \tag{50}$$

数値計算を行う場合、式（44）で計算すればよいわけであるが、式（50）もよく用いられる式である。

24 HH model (1)

先ずは、original の HH model。これまでと同じように、solveODE.cpp に常微分方程式を書き加える。いろいろな変数は、CsolveODE のコンストラクタのところで数値を設定している。

```
#include <math.h>
#include "solveODE.h"

CsolveODE::CsolveODE()
{
    Dm = 60.0;  // micrometer, diameter of the cell
```

```
    Lm = 70.0;  // micrometer, length of the cell
    Am = 3.14 * Dm * Lm; // squre_micrometer, surface area
    Cm = 0.01 * Am;
    InvCm = 1/Cm;
    fac = 10;  // conversion factor (mS/cm^2) -> (pS/micrometer^2)
// reversal potentials (resting potential = 0)
    Erest = -65;
    Eleak = 10.6;   // -54.3;//  -65.0;   // mV
    ENa   = 115;    // 50.0;      // mV
    EK    = -12;    //  -77.5;//   100.0;     // mV
// channel density
    dGleak = 0.0003;  //0.002;
    dNa = 0.120;   //   0.05;      // mS/cm^2
    dK  = 0.036;  //  0.15;
    Gleak = Am * dGleak * fac;
    GNa   = Am * dNa * fac;
    GK    = Am * dK * fac;
// injected current
    Iinj = 1000;     // pA
    Ton  = 100;       // ms
    Toff = 300;      // ms
}
CsolveODE::~CsolveODE(){}

void CsolveODE::rhs(double x, double* y, double* dy)
{
    double v = y[0]-Erest;
    double a,b;
    double     INa, IK;
    double c = (x<Ton || x>Toff)? 0 : Iinj;
// potential
    INa = GNa * pow(y[1],3.0) * y[2]  * (v-ENa);
    IK  = GK  * pow(y[3],4.0) * (v-EK);
    dy[0] = InvCm * (c - Gleak * (v-Eleak) - INa - IK );
// m-gate
    a = 0.1*(-v+25)/(exp((-v+25)/10)-1);
    b = 4 * exp(-v/18);
    dy[1]= a - (a+b)*y[1];
//h-gate
    a = 0.07 * exp(-v/20);
```

```
    b = 1 / (exp((-v+30)/10) + 1);
    dy[2] = a - (a+b)*y[2];
// n-gate
    a = 0.01*(-v+10)/(exp((-v+10)/10)-1);
    b = 0.125*exp(-v/80);
    dy[3]= a - (a+b)*y[3];
}
```

また y[] の初期値を、main() で

```
    double y[] = {-65, 0.05, 0.6, 0.3};
```

と設定。

25 HH model (2)

次に、Na^+ チャネル、K^+ チャネル、低閾値型 Ca^{2+} チャネルを持つ cell で、活動電位を起こさせてみよう。プログラムは、基本的にはこれまでのものと変わらないが大幅に行数が増加したので、区別するために solveODE.(h,cpp) を solveHH.(h,cpp) と変えた。

```
//
//   solveHH.h
//
#include "RKData.h"
class CsolveHH : public CRKData{
public:
    CsolveHH();
    ~CsolveHH();
public:
    void rhs(double x, double* yt, double* k);
    void initValues();
private:
    double Cm;     //   20.0 pF
    double Dm;     //   60.0 micrometer, diameter of the cell
    double Lm;     //   70.0 micrometer, length of the cell

    double Iinj;   // 2000.0 (pA)
    double Ton;    //   20 ms
    double Toff;   //   40 ms

    double Eleak;  //  -65.0 mV
    double ENa;    //   50.0 mV
```

```
    double EK;     // -100.0 mV
    double ECa;    //   50.0 mV

    double dGleak; //    0.002 mS/cm^2
    double dGNa;   //    0.53 mS/cm^2
    double dGK;    //    0.13 mS/cm^2
    double dGTCa;  //    0.03 mS/cm^2

    double vtraub; //  -55.0 mV

    double Am;     //  surface area
    double InvCm;  //  1/Cm;

    double Gleak;
    double GNa;
    double GK;
    double GTCa;
    double fac;    // conversion factor (mS/cm^2) -> (pS/micrometer^2)

private:
    void get_Na_m(double* tau, double* inf);
    void get_Na_h(double* tau, double* inf);
    void get_K_n(double* tau, double* inf);
    void get_TCa_m(double* tau, double* inf);
    void get_TCa_h(double* tau, double* inf);
};
//----------------------------------------

//
//   solveHH.h
// HH:   Traub & Miles 1991 "Neuronal Networks of the Hippocampus", Cambridge Univ Press.
// ITCa: Destexhe et al (1996) J Neurosci16:169-1856.
//
//    y[0]  v : voltage
//    y[1]  m : activation parameter for Na channel
//    y[2]  h : inactivation parameter for Na channel
//    y[3]  n : activation parameter for K channel
//    y[4]  m : activation parameter for TCa channel
//    y[5]  h : inactivation parameter for TCa channel
```

```
#include <math.h>
#include "solveHH.h"
CsolveHH::CsolveHH(){}
CsolveHH::~CsolveHH(){}

void CsolveHH::initValues(){{
    double tau, inf;
    double v;
    Cm  = 20.0;   // (pF)
    Dm = 60.0;  // (micrometer) diameter of the cell
    Lm = 70.0;  // (micrometer) length of the cell
    Iinj = -2000; //2000.0;  //(pA)
    Ton = 20;
    Toff = 220;// 40;
    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)
    Am = 3.14 * Dm * Lm;    // (squre_micrometer) surface area
    Cm = 0.01 * Am;
    InvCm = 1/Cm;
    fac = 10;  // conversion factor (mS/cm^2) -> (pS/micrometer^2)
    Gleak = Am * dGleak * fac;
    GNa   = Am * dGNa * fac;
    GK    = Am * dGK * fac;
    GTCa  = Am * dGTCa * fac;

    y[0] = Eleak;
    get_Na_m(&tau, &inf);
    y[1] = inf;
    get_Na_h(&tau, &inf);
    y[2] = inf;
    get_K_n(&tau, &inf);
    y[3] = inf;
    get_TCa_m(&tau, &inf);
```

```
    y[4] = inf;
    get_TCa_h(&tau, &inf);
    y[5] = inf;
}
void CsolveHH::get_Na_m(double* tau, double* inf){
    double a, b;
    double v2 = y[0]-vtraub;
    a = 0.32 * (13-v2) / ( exp((13-v2)/4) - 1);
    b = 0.28 * (v2-40) / ( exp((v2-40)/5) - 1);
    *tau = 1 / (a + b);
    *inf = a / (a + b);
}
void CsolveHH::get_Na_h(double* tau, double* inf){
    double a, b;
    double v2 = y[0]-vtraub;
    a = 0.128 * exp((17-v2)/18);
    b = 4 / ( 1 + exp((40-v2)/5) );
    *tau = 1 / (a + b);
    *inf = a / (a + b);
}
void CsolveHH::get_K_n(double* tau, double* inf){
    double a, b;
    double v2 = y[0] - vtraub;
    a = 0.032 * (15-v2) / ( exp((15-v2)/5) - 1);
    b = 0.5 * exp((10-v2)/40);
    *tau = 1 / (a + b);
    *inf = a / (a + b);
}
void CsolveHH::get_TCa_m(double* tau, double* inf){
    double vshift=0;
    double vs = y[0] + vshift;
    *tau = ( 3 + 1.0 / ( exp((vs+25)/10) + exp(-(vs+100)/15) ) );
    *inf = 1.0 / ( 1 + exp(-(vs+50)/7.4) );
}
void CsolveHH::get_TCa_h(double* tau, double* inf){
    double vshift=0;
    double vs = y[0] + vshift;
    *tau = ( 85 + 1.0 / ( exp((vs+46)/4) + exp(-(vs+405)/50) ) ) ;
    *inf = 1.0 / ( 1 + exp((vs+78)/5.0) );
}
```

```
void CsolveHH::rhs(double x, double* y, double* dy){
    double INa, IK, ITCa;
    double tau, inf;
    double shift = 0;
    double v = y[0];
    double c = (x<Ton || x>Toff)? 0 : Iinj;
// voltage
    INa = GNa * pow(y[1],3.0) * y[2]  * (v-ENa);   // INa = GNa * m^3 * h * (v-Ena)
    IK = GK * pow(y[3],4.0) * (v-EK);             // IK = GK * n^4 (v-EK)
    ITCa = GTCa * pow(y[4],2.0) * y[5] * (v - ECa);
    dy[0] = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa );
// Na channel
    get_Na_m(&tau, &inf);
    dy[1] = (inf - y[1]) / tau;
    get_Na_h(&tau, &inf);
    dy[2] = (inf - y[2]) / tau;
// K channel (delayed rectifier)
    get_K_n(&tau, &inf);
    dy[3] = (inf - y[3]) / tau;
// T-type Ca channel
    get_TCa_m(&tau, &inf);
    dy[4] = (inf - y[4]) / tau;
    get_TCa_h(&tau, &inf);
    dy[5] = (inf - y[5]) / tau;
}

//-------------------------------------------------------
//    rk16.cpp
//
#include "solveHH.h"
#include "DataRec.h"
int main(){
// initial conditions
    double y[6];
    double h = 0.01;
    int iteration = (int)(500.0/h);
//-------------------------------
    int n = sizeof(y)/sizeof(double);
    int i;
```

```
    double x;
    CsolveHH hh;
    CDataRecorder rec;
    hh.initState(n,h,y);
    hh.initValues();
    rec.Start(n,y);
    for(i=0;i<iteration;i++){
        x = h*i;
        rec.StoreData(x);
        hh.rk4(x);
    }
    rec.Stop();
    rec.Replay();
    return 0;
}
```

$h = 0.02$ では、途中で発散を起こしてしまう。Na^+ チャネルと K^+ チャネルについては、$dy = (inf - y)/tau$ ではなく、$dy = (1 - y)\alpha - y\beta$ の形にすると、より安定であり、$h = 0.08$ 程度まで発散せずに計算することが出来る。数値計算で鬼門は割り算である。Na^+ チャネルが活動電位を発生する場合、電位の値は急激に変化するが、その時に割り算形式の計算方法では、発散を誘発してしまうようである。

26 HH model (3)

更にいろいろなチャネルを加えることを想定して、それぞれのチャネルもクラス化してみた。

```
//  Component.h
//
class CCell;
class CEnv;
class CComponent{
public:
    CComponent();
protected:
    ~CComponent();
public:
    virtual void initState(double* y){};
    virtual void rhs(double x, double* y, double* dy){};
public:
    int neqs;
    int bp;
    double density;
```

```
    double G;          // max conductance
    CCell* pCell;
    CEnv* pEnv;
};

//---------Na channel---------
class CNaChannel : public CComponent{
public:
    CNaChannel();
    ~CNaChannel();
private:
    double vtraub;
public:
    void initState(double* y);
    void rhs(double x, double*y, double* dy);
};
//---------K channel---------
class CKChannel : public CComponent{
public:
    CKChannel();
    ~CKChannel();
private:
    double vtraub;
public:
    void initState(double* y);
    void rhs(double x, double*y, double* dy);
};
//---------TCa channel---------
class CTCaChannel : public CComponent{
public:
    CTCaChannel();
    ~CTCaChannel();
private:
public:
    void initState(double* y);
    void rhs(double x, double*y, double* dy);
};
//
//  Component.cpp
//
```

```
#include <math.h>
#include "Component.h"
CComponent::CComponent(){};
CComponent::~CComponent(){};
//================================================
CNaChannel::CNaChannel()
{
    neqs = 2;
    vtraub = -55.0;
}
CNaChannel::~CNaChannel(){}
void CNaChannel::initState(double* y)
{
    double a,b;
    double v2 = y[0]-vtraub;

    a = 0.32 * (13-v2) / ( exp((13-v2)/4) - 1);
    b = 0.28 * (v2-40) / ( exp((v2-40)/5) - 1);
    y[bp+0]=a/(a+b);

    a = 0.128 * exp((17-v2)/18);
    b = 4 / ( 1 + exp((40-v2)/5) );
    y[bp+1]=a/(a+b);
}

void CNaChannel::rhs(double x, double*y, double* dy)
{
    double v2 = y[0]-vtraub;
    double a,b;
// m-gate
    a = 0.32 * (13-v2) / ( exp((13-v2)/4) - 1);
    b = 0.28 * (v2-40) / ( exp((v2-40)/5) - 1);
    dy[bp+0]= a - (a+b)*y[bp+0];
//h-gate
    a = 0.128 * exp((17-v2)/18);
    b = 4 / ( 1 + exp((40-v2)/5) );
    dy[bp+1] = a - (a+b)*y[bp+1];
}

//================================================
```

```
CKChannel::CKChannel()
{
    neqs = 1;
    vtraub = -55.0;
}
CKChannel::~CKChannel(){}

void CKChannel::initState(double* y)
{
    double v2 = y[0]-vtraub;
    double a,b;
// n-gate
    a = 0.032 * (15-v2) / ( exp((15-v2)/5) - 1);
    b = 0.5 * exp((10-v2)/40);
    y[bp+0]= a/(a+b);
}

void CKChannel::rhs(double x, double*y, double* dy)
{
    double v2 = y[0]-vtraub;
    double a,b;
// n-gate
    a = 0.032 * (15-v2) / ( exp((15-v2)/5) - 1);
    b = 0.5 * exp((10-v2)/40);
    dy[bp+0]= a - (a+b)*y[bp+0];
}

//================================================

CTCaChannel::CTCaChannel()
{
    neqs = 2;
}

CTCaChannel::~CTCaChannel(){}

void CTCaChannel::initState(double* y)
{
    double vshift=0;
```

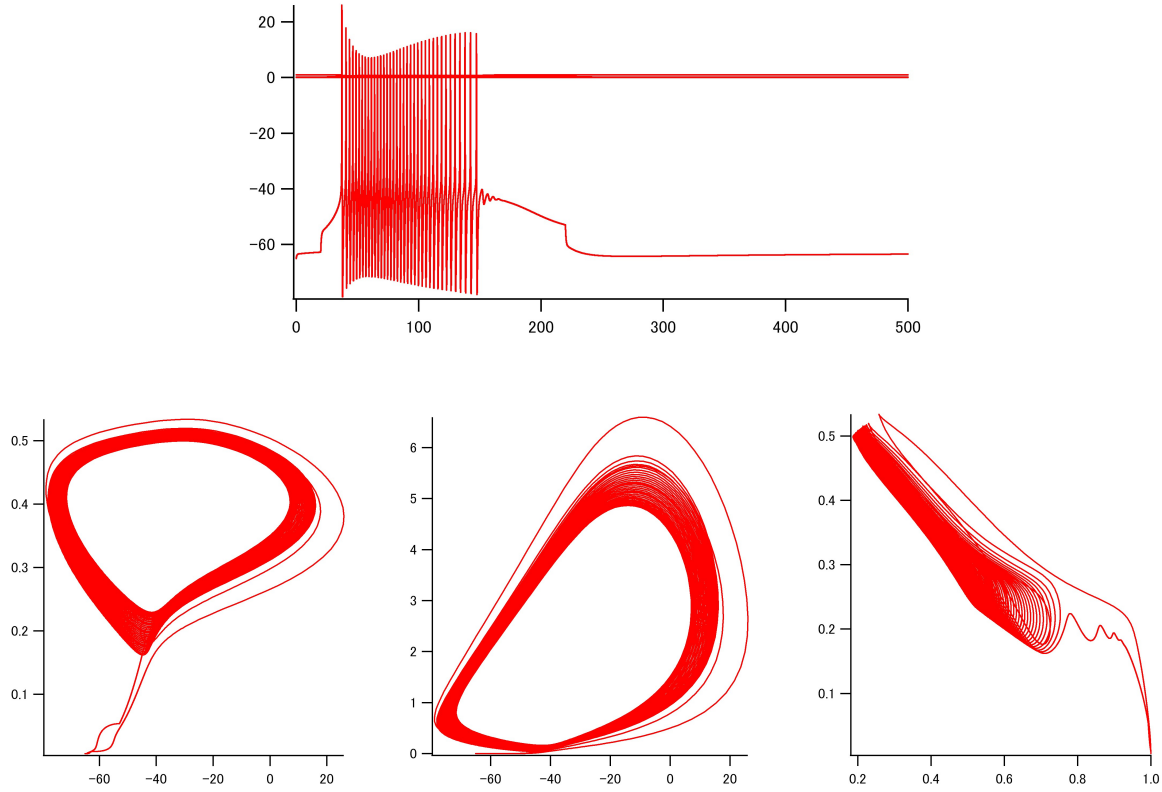
```
    double vs = y[0] + vshift;

    y[bp+0] = 1.0 / ( 1 + exp(-(vs+50)/7.4) );
    y[bp+1] = 1.0 / ( 1 + exp((vs+78)/5.0) );
}

void CTCaChannel::rhs(double x, double*y, double* dy)
{
    double vshift=0;
    double vs = y[0] + vshift;
    double tau, inf;

    tau = ( 3 + 1.0 / ( exp((vs+25)/10) + exp(-(vs+100)/15) ) );
    inf = 1.0 / ( 1 + exp(-(vs+50)/7.4) );
    dy[bp+0] = (inf - y[bp+0]) / tau;

    tau = ( 85 + 1.0 / ( exp((vs+46)/4) + exp(-(vs+405)/50) ) ) ;
    inf = 1.0 / ( 1 + exp((vs+78)/5.0) );
    dy[bp+1] = (inf - y[bp+1]) / tau;
}


////////////////////////////////////////
//  Cell.h
//
class CRKData;
class CComponent;
class CCell : public CRKData
{
public:
    CCell();
//protected:
    ~CCell();
private:
    int neqs;

    double Cm; // 20.0 pF
    double Dm; // 60.0 micrometer :: diameter of the cell
    double Lm; // 70.0 micrometer :: length of the cell
```

```
    double Eleak; //  -65.0 mV
    double ENa;   //   50.0 mV
    double EK;    // -100.0 mV
    double ECa;   //   50.0 mV

    double dGleak; //  0.002 mS/cm^2
    double Gleak;

    double Am;  //  = 3.14 * Dm * Lm;    // (squre_micrometer) surface area
    double InvCm; // = 1/Cm;
    double fac;  // = 10;  // conversion factor (mS/cm^2) -> (pS/micrometer^2)

    CNaChannel*  pNaChannel;
    CKChannel*   pKChannel;
    CTCaChannel* pTCaChannel;
    double dNa;
    double dK;
    double dTCa;

    double Iinj;   // 2000.0 (pA)
    double Ton;    //   20 ms
    double Toff;   //   40 ms

private:
    int addComponent(CComponent* pComp);
public:
    int initCell(double h);
    void rhs(double x, double*y, double* dy);
};
//
//  Cell.cpp
//
#include <math.h>
#include "RKData.h"
#include "Component.h"
#include "Cell.h"

CCell::CCell()
{
    neqs = 1;    // one equation for voltage;
```

```
// cell shape
    Dm = 60.0;  // micrometer, diameter of the cell
    Lm = 70.0;  // micrometer, length of the cell
    Am = 3.14 * Dm * Lm; // squre_micrometer, surface area
    Cm = 0.01 * Am;
    InvCm = 1/Cm;
    fac = 10;  // conversion factor (mS/cm^2) -> (pS/micrometer^2)
// reversal potentials
    Eleak = -65.0;   // mV
    ENa = 50.0;      // mV
    EK = -100.0;     // mV
    ECa = 50.0;      // mV
// channel density
    dNa = 0.05;      // mS/cm^2
    dK  = 0.15;
    dTCa = 0.03;
    dGleak = 0.002;
    Gleak = Am * dGleak * fac;
// injected current
    Iinj = 2000;     // pA
    Ton  = 20;       // ms
    Toff = 220;      // ms
}

CCell::~CCell()
{
    delete pNaChannel;
    delete pKChannel;
    delete pTCaChannel;
};

int CCell::initCell(double h)
{
    pNaChannel  = new CNaChannel();
    pKChannel   = new CKChannel();
    pTCaChannel = new CTCaChannel();

    pNaChannel->G = dNa * Am * fac;
    pKChannel->G = dK * Am * fac;
    pTCaChannel->G = dTCa * Am * fac;
```

```
    neqs = addComponent(pNaChannel);
    neqs = addComponent(pKChannel);
    neqs = addComponent(pTCaChannel);

    initState(neqs,h); // allocate the working spaces

    y[0]=Eleak;
    pNaChannel->initState(y);
    pKChannel->initState(y);
    pTCaChannel->initState(y);

    return neqs;
}

int CCell::addComponent(CComponent* pComp)
{
    pComp->bp = neqs;
    neqs += pComp->neqs;
    return neqs;
}

void CCell::rhs(double x, double*y, double* dy)
{
    double     INa, IK, ITCa;
    double v = y[0];
    double c = (x<Ton || x>Toff)? 0 : Iinj;
    int bx;

    bx = pNaChannel->bp;
    INa = pNaChannel->G * pow(y[bx],3.0) * y[bx+1]  * (v-ENa);

    bx = pKChannel->bp;
    IK = pKChannel->G * pow(y[bx+0],4.0) * (v-EK);

    bx = pTCaChannel->bp;
    ITCa = pTCaChannel->G * pow(y[bx+0],2.0) * y[bx+1] * (v - ECa);

    dy[0] = InvCm * (c - Gleak * (v-Eleak) - INa - IK - ITCa );
```

```
    pNaChannel->rhs(x,y,dy);
    pKChannel->rhs(x,y,dy);
    pTCaChannel->rhs(x,y,dy);
}

////////////////////////////////////////////////////////////////
//          rk17.cpp
//
#include "RKData.h"
#include "DataRec.h"
#include "Component.h"
#include "Cell.h"

int main()
{
    int n;
    int i;
    int iteration;
    double x;
    double h = 0.01;
    iteration = (int)(500.0/h);
    CDataRecorder rec;
    CCell* pCell =  new CCell();
    n = pCell->initCell(h);
    rec.Start(n,pCell->y);
    for(i=0;i<iteration;i++){
        x = h*i;
        rec.StoreData(x);
        pCell->rk4(x);
    }
    rec.Stop();
    delete pCell;

    rec.Replay();
    return 0;
}
```



左 n vs $v_\circ$ **中** $n^4 * (v - E_K)$ vs $v_\circ$ **右** n vs $h_\circ$

27 Markov 過程のプログラム

Clancy & Rudy のモデルをプログラム化したもので、心筋の Na^+ チャネルである。実際に上記の神経細胞モデルに入れると、活動電位が 1 回発生しますが、それで終わってしまう。不活性化からの回復過程が遅いためと思われる。神経細胞の Na^+ チャネルのモデルは公表されているものはまだないようである。

```
//
//  MarkovNaChannel.h
//
#define MARKOV_NA_C3   0
#define MARKOV_NA_C2   1
#define MARKOV_NA_C1   2
#define MARKOV_NA_O    3
#define MARKOV_NA_IC3  4
#define MARKOV_NA_IC2  5
#define MARKOV_NA_IF   6
#define MARKOV_NA_IM1  7
```

```
class CComponent;
class CMarkovNaChannel : public CComponent
{
public:
    CMarkovNaChannel();
    ~CMarkovNaChannel();
public:
    double Vsh_a13, Vsl_a13;
    double Fac_a3, Fac_a5, Fac_b5;
private:
    double a11, a12, a13, a2, a3, a4, a5;
    double b11, b12, b13, b2, b3, b4, b5;
    double pC3, pC2, pC1;
    double pO;
    double pIC3, pIC2, pIF;
    double pIM1, pIM2;
public:
    void initState(double* y);
    void rhs(double x, double*y, double* dy);
};

//
//  MarkovNaChannel.cpp
//

//
//   Clancy CE & Rudy Y (2002) Circulation 105:1208-1213.
//
//   Na channel Markov process model
//   (9 states)
//
//               IC3  - IC2   -  IF - IM1 - IM2
//                 |    |      /  |
//                C3  - C2  - C1 - O
//
//        0 -- C3
//        1 -- C2
//        2 -- C1
//        3 -- O   (open)
//        4 -- IC3
```

```
//        5 -- IC2
//        6 -- IF
//        7 -- IM1
//        8 -- IM2
//

#include <math.h>
#include "Component.h"
#include "MarkovNaChannel.h"

CMarkovNaChannel::CMarkovNaChannel()
{
    neqs = 8;
}

CMarkovNaChannel::~CMarkovNaChannel(){}

void CMarkovNaChannel::initState(double* y)
{
    Vsh_a13 = 0;
    Vsl_a13 = 12.0;
    Fac_a3 = 1.0;
    Fac_a5 = 1.0;
    Fac_b5 = 50.0;

    pC3 = 0.52;
    pC2 = 0.02;
    pC1 = 0.0;
    pO  = 0.0;
    pIC3= 0.1;
    pIC2= 0.1;
    pIF = 0.01;
    pIM1= 0.05;

    y[bp+MARKOV_NA_C3]=   pC3;
    y[bp+MARKOV_NA_C2]=   pC2;
    y[bp+MARKOV_NA_C1]=   pC1;
    y[bp+MARKOV_NA_O]=    pO;
    y[bp+MARKOV_NA_IC3]=  pIC3;
    y[bp+MARKOV_NA_IC2]=  pIC2;
```

```
    y[bp+MARKOV_NA_IF]=   pIF;
    y[bp+MARKOV_NA_IM1]=  pIM1;
}

void CMarkovNaChannel::rhs(double t, double* y, double* dy)
{
    double v = y[0];

    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-Vsh_a13)/Vsl_a13) + 0.25*exp(-v/150));
    // shift = 0, slope = 12.0

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

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

    a3 = Fac_a3*(3.7933e-7)*exp(-v/7.7);
    b3 = 0.0084 + 0.00002*v;

    a4 = a2/100;
    b4 = a3/Fac_a3;

    a5 = Fac_a5 * a2/(9.5e4);
    b5 = a3/Fac_b5;       //  Fac_b5 = 50 for wild-type

    pC3  = y[bp+MARKOV_NA_C3];
    pC2  = y[bp+MARKOV_NA_C2];
    pC1  = y[bp+MARKOV_NA_C1];
    pO   = y[bp+MARKOV_NA_O];
    pIC3 = y[bp+MARKOV_NA_IC3];
    pIC2 = y[bp+MARKOV_NA_IC2];
    pIF  = y[bp+MARKOV_NA_IF];
    pIM1 = y[bp+MARKOV_NA_IM1];

    pIM2 = 1 - pC3 - pC2 - pC1 - pO - pIC3 - pIC2 - pIF - pIM1;
```

```
    dy[bp+MARKOV_NA_C3] = -(a11 + b3)*pC3 + b11*pC2 + a3*pIC3;
    dy[bp+MARKOV_NA_C2] = -(b11 + b3 + a12)*pC2 + a11*pC3 + b12*pC1 + a3*pIC2;
    dy[bp+MARKOV_NA_C1] = -(b12 + b3 + a13)*pC1 + a12*pC2 + b13*pO  + a3*pIF;
    dy[bp+MARKOV_NA_O]  = -(b13 + a2)*pO + a13*pC1 + b2*pIF;
    dy[bp+MARKOV_NA_IC3] = -(a11 + a3)*pIC3 + b11*pIC2 + b3*pC3;
    dy[bp+MARKOV_NA_IC2] = -(b11 + a3 + a12)*pIC2 + a11*pIC3 + b12*pIF + b3*pC2;
    dy[bp+MARKOV_NA_IF] = -(b12 + a3 + b2 + a4)*pIF + a12*pIC2 + b3*pC1 + a2*pO + b4*pIM1;
    dy[bp+MARKOV_NA_IM1] = -(b4 + a5)*pIM1 + a4*pIF + b5*pIM2;

}
```

28 偏微分方程式

独立変数が 2 個以上ある場合の微分方程式は、偏微分方程式 Partial differential equation といわれる。身近な例としては、cable equation がある。

$$\tau \frac{\partial V}{\partial t} = D^2 \frac{\partial^2 V}{\partial x^2} - V \tag{51}$$

この cable equation は、dendrites の電位の変化などを計算するために用いられる。

偏微分方程式には、これまでに述べた Runge-Kutta 法を用いることは出来ない。数値計算には、有限要素法 Finite element method などが用いられる。拡散方程式の計算には、Crank-Nicolson 法が多く用いられている。

未完です。

予定

・複数のコンパートメントを動かし、相互のカップリングを計算する。

REFERENCES

1. Johnston D & Wu SM (1995) “Foundations of cellular neurophysiology” MIT Press, Cambridge, MA.
2. Press WH, et al. (1994) “Numerical recipes in C” 2nd ed. Cambridge University Press, New York.
3. Wilson HR (1999) “Spikes, decisions, and actions” Oxford University Press, New York.
4. 宮川博義, 井上雅司 (2003) “ニューロンの生物物理” 丸善, 東京.