数値計算の理論と実際

第10回 常微分方程式の解法3

1　3体問題

重力による質点の運動は、2点までなら解析的に解くことが可能であるが、3点以上では一般解が存在しないことが証明されている。したがって天体や人工衛星の軌道を計算するために、数値計算が非常に大きな役割を果たしてきた。今回は質点が3つの場合、いわゆる**3体問題**について実習を行う。

時刻をtとすると運動方程式は

$$\frac{d^2\boldsymbol{r}_i}{dt^2} = \sum_{j\neq i}^{N} Gm_j \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^3}. \tag{1}$$

ここでNは質点数、mは質点の質量、$\boldsymbol{r}$はi番目の質点の空間座標ベクトルである。添字i, jは粒子番号を表す。

これまでに実装した1体問題では、空間2次元の場合、1質点につき4つの連立微分方程式を数値的に解いた。3体問題の場合は12で、空間3次元の場合は18の連立微分方程式を解くことになる。

質点が増えた場合もエネルギー保存が成立する。ただし保存するエネルギーは全質点の総エネルギーである。

以下空間3次元の場合のサンプルコードである。

```
.
略
.
static const int NMAX = 128;

//ルンゲクッタ用の変数
static double xtmp[NMAX][3];
static double k[NMAX][3], k1[NMAX][3], k2[NMAX][3], k3[NMAX][3], k4[NMAX][3];
static double l[NMAX][3], l1[NMAX][3], l2[NMAX][3], l3[NMAX][3], l4[NMAX][3];

void gravity( double (*x)[3], double *m, double (*a)[3],
              double *p, const int n){

  for( int i=0; i<n; i++){
    各質点にはたらく重力を計算する
  }

}
```

```
double energy( double (*v)[3], double *m, double *p, const int n){

  総エネルギーを計算する

}

void runge4( double (*x)[3], double (*v)[3], double *m, double (*a)[3],
             double *p, const double dt, const int n){

  4次ルンゲクッタ

}

int main(){

  double x[NMAX][3];
  double v[NMAX][3];
  double a[NMAX][3];
  double m[NMAX];
  double p[NMAX];
  int n;
  double dt, tend;

  cerr << "input n, dt, tend, m, x, v" << endl;
  cin >> n >> dt >> tend;
  cerr << n << "\t" << dt << "\t" << tend << endl;
  for( int i=0; i<n; i++){
    cin >> m[i];
    for( int j=0; j<3; j++)  cin >> x[i][j];
    for( int j=0; j<3; j++)  cin >> v[i][j];
  }

  gravity( x, m, a, p, n);
  double e0 = energy( v, m, p, n);

  double tnow = 0.0;

  cout << tnow << "\t"
       << x[0][0] << "\t" << x[0][1] << "\t" << x[0][2] << "\t"
       << x[1][0] << "\t" << x[1][1] << "\t" << x[1][2] << "\t"
       << x[2][0] << "\t" << x[2][1] << "\t" << x[2][2]  << endl;

  while( tnow < tend){

    runge4( x, v, m, a, p, dt, n);
    double e = energy( v, m, p, n);
    tnow += dt;

    cout << tnow << "\t"
         << x[0][0] << "\t" << x[0][1] << "\t" << x[0][2] << "\t"
         << x[1][0] << "\t" << x[1][1] << "\t" << x[1][2] << "\t"
         << x[2][0] << "\t" << x[2][1] << "\t" << x[2][2]  << "\t"
         << e << "\t" << (e-e0)/e0 << endl;
  }
```

```
}
```

2 gnuplot による簡易アニメーション

gnuplot ではあるデータをグラフ化するだけでなく、簡単なアニメーションを表示させることも可能である。データブロックという概念を利用すると簡単である。

test.dat というファイル名で保存されたデータ、

```
0.0 1.0
1.0 2.0
```

に対して、

```
gnuplot> p "test.dat" u 1:2
```

とすると 2 点がプロットされる。一方、

```
0.0 1.0


1.0 2.0
```

のように間に 2 行を空ける。このデータに対して、

```
gnuplot> p "test.dat" index 0 u 1:2
gnuplot> p "test.dat" index 1 u 1:2
```

とするとはじめのコマンドでは [0.0,1.0] のみが、2 番めのコマンドでは [1.0,2.0] のみがプロットされる。

gnuplot では 2 行空いたデータはそれぞれ別のグループのデータ (データブロック) とみなす。データブロックはファイルの上から 0,1,2... というインデックスが与えられる。コマンド中の"index" オプションでプロットするブロックを指定することができる。

そして次のファイルを用意し、anim.pl というファイル名で保存したとする。

```
if(i==0)  set size square
if(i==0)  set xrange[-2:2]
if(i==0)  set yrange[-2:2]
print i
p "output" u 2:3 index i notitle ps 5 pt 65
i=i+1
pause 0.01
if(i<100000) reread
```

詳しい説明は省略するが、$i = 0$ の時にグラフの設定を行い、0.01 秒毎に i をインクリメントし、"output" ファイルの i 番めのデータブロックをプロットしている。output ファイル中で各時刻の座標を 2 行置きに書いておけば、i がインクリメントされるたびにプロットするデータブロックが変更され、アニメーションされるという仕組みである。

anim.pl は gnuplot 上で次のようにロードする。

```
gnuplot> i=0
gnuplot> load "anim.pl"
```