シミュレーション概論ノート第８講

12　偏微分方程式

12.1　全般的注意

独立変数を２個以上含む関数の偏微分を含むような方程式系を偏微分方程式という。ところが偏微分方程式を統一的に論じることは難しく、ある特定の型の方程式に対して個別に議論する場合が多い。逆に言えばそれだけ偏微分方程式にはバリエーションとバラエティがあるということである。

本講義でもある程度対称を限定して話をする。ここでは対象とする偏微分方程式を専ら線形のものに限定し、更に放物型、双曲型、楕円型の代表例である拡散方程式、波動方程式、ラプラス方程式を扱う。また簡単のため差分方程式に基づく解法のみを紹介する。

12.2　拡散方程式の陽解法

拡散方程式は熱伝導や溶液上での溶質の拡散などに普遍的に現れる

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{1}$$

を初期条件 $u(x,0) = \phi(x)$、境界条件 $u(0,t) = u(1,t) = 0$ for $t \geq 0$ で考えてみる。Δx, Δt をそれぞれ空間、時間の差分間隔として $N\Delta x = 1$ であるとする。$x_j = j\Delta x$, $t_n = n\Delta t$ として時間に関して前進差分、空間に関して中心差分を考えてみよう。このとき $u(x_j, t_n)$ の差分近似をした解を U_j^nとおくと

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{(\Delta x)^2} \tag{2}$$

となることは直ちにわかる。或は上式は$\alpha = \Delta t/(\Delta x)^2$を用いて

$$U_j^{n+1} = \alpha(U_{j+1}^n + U_{j-1}^n) + (1 - 2\alpha)U_j^n \tag{3}$$

と書き直すことができる。これは常微分方程式のオイラー法と同様に離散列 U_j^nから U_j^{n+1}へのマップと考えられるので数値的に解く（漸化式から時間発展を求める）ことは容易である。

アルゴリズムは極めて単純である。解を求める時間の上限をTとして適当な自然数 Mに対して $\Delta t = T/M$とすると、アルゴリズムは

- N, M, Tの設定。
$$\Delta x = 1/N, \quad \Delta t = T/M, \quad \alpha = \Delta t/(\Delta x)^2$$
- 初期条件：$j = 0, 1, \cdots, N$の順に
$$U_j = \phi(j\Delta x)$$
を繰り返す。

境界条件からダミー変数 UU_jのうち両端は

$$UU_0 = 0, UU_N = 0$$

を常に満たす。

- 時間発展： $n = 0, 1, \cdots, M-1$ の順に
  - 空間の格子点について $j = 1, 2, \cdots, N-1$ の順に

$$UU_j = \alpha(U_{j+1} + U_{j-1}) + (1 - 2\alpha)U_j$$

    を繰り返す。
  - $j = 0, 1, \cdots, N$の順に

$$U_j = UU_j$$

    を繰り返す

  を繰り返す。

ここで配列 U_j,UU_jは時間の index を持たないことを注意して欲しい。時間発展の際に一旦ダミーの配列 UU_jにデータを保管してから元に戻すという作業を繰り返すことでメモリを節約している。一般に時間発展の回数は極めて大きく、時間に対して配列を切っていたらすぐ メモリーが足りなくなる。そのため上記の考え方は標準的となっている。

上記の陽解法に対するサンプルプログラムは以下の通り

```
*   拡散方程式のシミュレーション (explicit scheme, 単精度)

         parameter (nmax=50)
         real*4 A,dx,dt
         real*4 U(0:nmax),UU(0:nmax)

        N=2000
        dt=1.0/10000.0
        dx=1.0/real(nmax)
        a=dt/dx**2

        DO 5 I=0,nmax
           U(I)=F(real(I)/real(nmax))
           UU(I)=0.0
5       CONTINUE
*
         DO 100 J=0,N
*   結果出力
        IF(MOD(J,400).EQ.0) THEN
c           PRINT *, 'T=',J-1
            write(6,1000) (dx*real(i),U(I),I=0,nmax)
            write(6,200)
 200        format()
```

```
       ENDIF
*  メインループ
          DO 20 I=1,nmax-1
             UU(I)=A*U(I+1)+(1.0d0-2.0d0*A)*U(I)+A*U(I-1)
20        CONTINUE
          DO 10 I=0,nmax
             U(I)=Uu(I)
10        CONTINUE

100    CONTINUE
1000   FORMAT(5x,2f10.4)
       END

       Real FUNCTION F(X)
         F=2.0*X*(1.0-X)
       RETURN
       END
```

C もこれに対応して

```
************************************************************
  シミュレーション概論

  simulation of deffusion equation (explicit scheme, double)

  コンパイルは、
  gcc tadifexp.c -lm

************************************************************/
#include <stdio.h>

#define NMAX 50
#define N    2000

double f(double);

main()
{
  int i, j;
  double a, dx, dt, tmax;
  double U[NMAX+1], UU[NMAX+1];
```

```
  dt = 1.0 / 10000.0;
  dx = 1.0 / (double)NMAX;
  a = dt / dx / dx;

  for(i = 0; i <= NMAX; i++){
    U[i] = f((double)i / (double)NMAX);
    UU[i] = 0.0;
  }

  for(j = 0; j <= N; j++){

    if((j % 400) == 0){
      for(i = 0; i <= NMAX; i++){
printf("\t%10.4f\t%10.4f\n",dx * (double)i, U[i]);
      }
      printf("\n");
    }

    for(i = 1; i <= NMAX - 1; i++){
      UU[i] = a * U[i+1] + (1.0 - 2.0 * a) * U[i] + a * U[i-1];
    }

    for(i = 1; i <= NMAX; i++) {
      U[i] = UU[i];
    }
  }
}

double f(double x)
{
  return 2.0 * x * (1.0 - x);
}
```

通常拡散方程式は Fourier 級数を使って解くが、差分方程式もほぼ同様の解法を用いることができる。即ち

$$U_j^n = f(n)\exp(ikj\Delta x) \tag{4}$$

とおいて (3) に代入すると

$$f(n+1) = (1 - 4\alpha \sin^2(\frac{k\Delta x}{2}))f(n) \tag{5}$$

という漸化式を得る。ここで $f(0) = 1$ とおくと

$$U_j^n = (1 - 4\alpha \sin^2(\frac{k\Delta x}{2}))^n \exp(ikj\Delta x) \tag{6}$$

となる。従って任意の kに対して

$$|1 - 4\alpha \sin^2(\frac{k\Delta x}{2})| \leq 1 \tag{7}$$

がスキームの安定性の必要条件となる。従って

$$\alpha \leq 1/2 \tag{8}$$

が必要である。

問１ $\phi(x) = 2x(1-x)$ とおき次の場合の拡散方程式を解け。

(1) $\Delta x = 1/6$, $\Delta t = 1/100$ の場合の $n = 0, 4, 8, \cdots, 20$ での差分解を図示せよ。

(2) $\Delta x = 1/10, \Delta t = 1/100$ の場合の差分解。$n = 0, 1, 2, \cdots, 7$ での値を図示せよ。

(3) $\Delta x = 1/10, \Delta t = 1/500$ での差分解。 $n = 0, 20, 40, \cdots, 100$ での値を図示せよ。

ここで説明した方法は陽解法と呼ばれる。陽解法には時間発展に Runge-Kutta 法を用いたものも含まれる。

問２： 問１に対して Runge-Kutta 法のプログラムを書け。その場合の安定性の条件を求めてみよ。

12.3 拡散方程式の陰解法

陽解法はアルゴリズムは簡単で便利であるが、安定性の条件 (8) は実際の数値計算を行なう上でかなり厳しい。実際空間の刻みを 1/10 にすると時間刻みは 1/100 になってしまって計算効率が甚だしく悪い。勿論、時間発展に工夫をこらし Runge-Kutta 法などを採用すればその負担はかなり軽減されるものではあるがそれほど画期的な改善は見込めない。むしろ安定性条件を必ず満たすアルゴリズムの開発をすれば問題点の多くは解決できると期待される。

こうした考え方に基づいた解法が陰解法と呼ばれるものである。いろいろな陰解法があるが最も簡単なものは (3) の右辺の時刻 n を $n+1$ におきかえれてみよう。整理すると

$$-\alpha(U_{j-1}^{n+1} + U_{j+1}^{n+1}) + (1+2\alpha)U_j^{n+1} = U_j^n \tag{9}$$

となる。これは漸化式を解くためには連立一次方程式を解く必要があり、時刻 n から $n+1$ へのマップがすぐ求まらないので一見複雑である。しかし既に連立１次方程式のプログラムは持っている上に、上式は対角に $1+2\alpha$, その脇に$-\alpha$が並ぶ３重対角行列で表現できる簡単な連立方程式である。従って計算効率は同じΔt に対して陽公式より数倍遅くなる程度ですむ。

一方で陰公式の安定性を調べると以下の様に無条件安定であることがわかる。このことは陽解法の差分解 (6) を逆に考えて解が

$$U_j^n = (1 + 4\alpha \sin^2(\frac{k\Delta x}{2}))^{-n} \exp(ikj\Delta x) \tag{10}$$

となることは容易にわかる。ここで

$$1 + 4\alpha \sin^2(\frac{k\Delta x}{2}) \geq 1 \tag{11}$$

なので $(1 + 4\alpha \sin^2(\frac{k\Delta x}{2}))^{-n} \leq 1$ である。従ってαの値によらずに安定である。

アルゴリズムとしては陽解法の３番目の時間発展 $UU_j = \alpha(U_{j+1} + U_{j-1} + (1-2\alpha)U_j$を連立一次方程式を解く様に変えればよい。このとき Pivot 操作は勿論不要であるし、３重対角に特化したプログラムを使えばそれだけ加速が可能である。

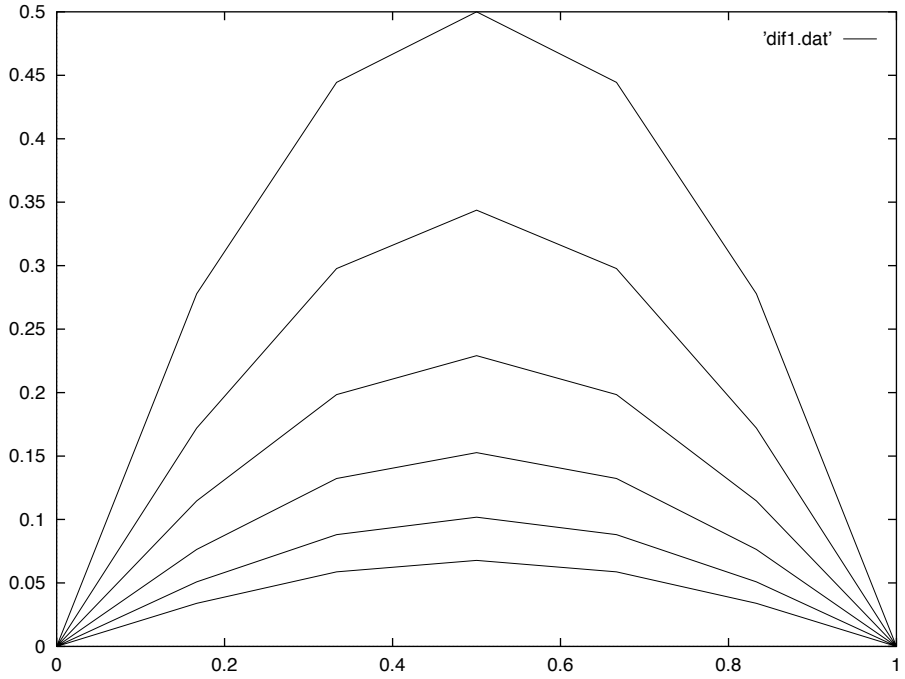

図 1: 問１ (1) の数値解。縦軸は拡散場の値、横軸は空間。

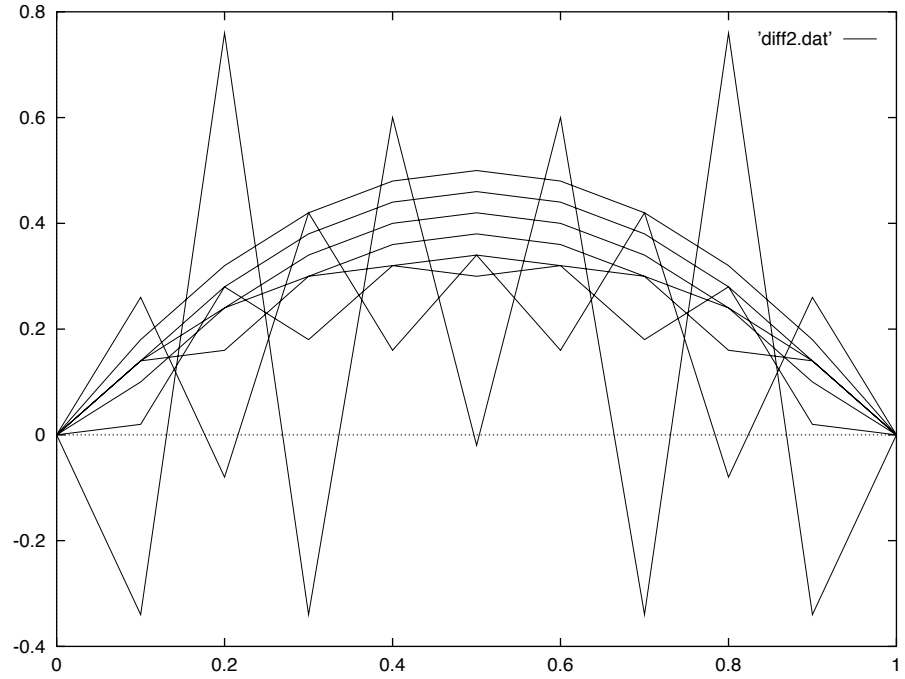

図 2: 問１ (2) の数値解

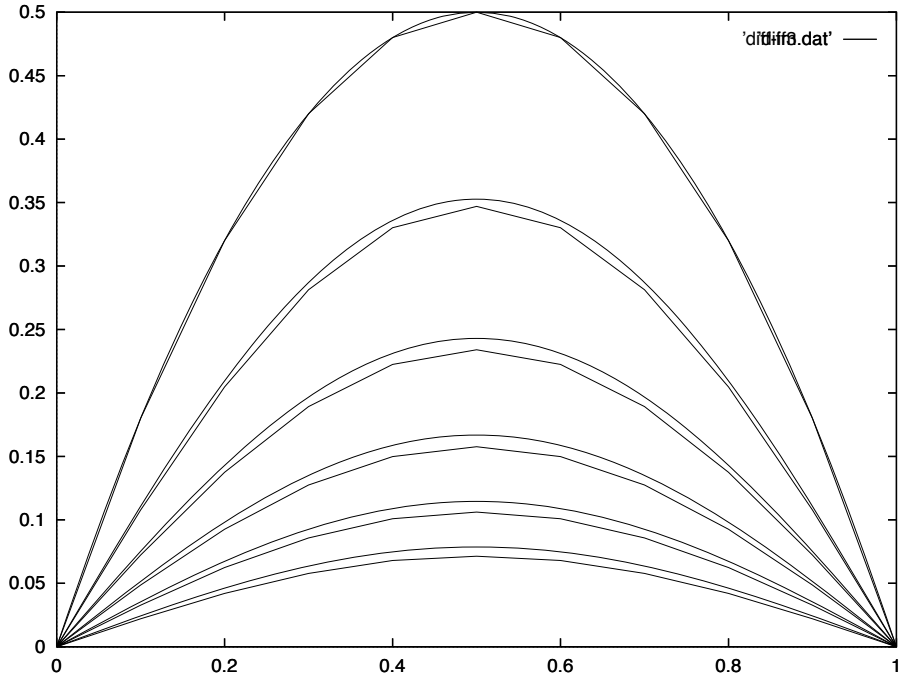

図 3: 問 1 (3) の数値解

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C  A program of the diffsuion equation in terms of implicit scheme   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

*     拡散方程式の数値シミュレーション
*     Implicit scheme
*
      PARAMETER(N=49)
      REAL A(1:N,1:N),AL,L(1:N,1:N),Y(1:N),DX,DT
      REAL B(1:N),BB(1:N),U(1:N,1:N),UXT(0:N+1)
      DATA DX/0.02/,DT/0.01/
C      PRINT *, '拡散方程式シュミレーション dx=0.02,dt=0.01'
*     初期値の入力

      UXT(0)=0.0
      UXT(N+1)=0.0

      DO 10 I=1,N
         UXT(I)=F(REAL(I)/(N+1))
         B(I)=UXT(I)
10    CONTINUE

      write(6,1000) (dx*real(i),UXT(I),I=0,n+1)
```

```
*
      AL=DT/(DX**2)
*    A（N、N）を決める
      DO 1 J=1,N
          DO 2 K=1,N
             A(J,K)=0.0
2     CONTINUE
1     CONTINUE
*
      DO 3 J=1,N
        A(J,J)=1.0+2.0*AL
3     CONTINUE
*
      DO 4 J=1,N-1
        A(J,J+1)=-1.0*AL
4     CONTINUE
*
      DO 5 J=2,N
        A(J,J-1)=-1.0*AL
5     CONTINUE
*
      CALL BUNKAI(A,L,U,N)
*     時間ループ
      DO 100 J=1,20

      DO 20 I=1,N
      BB(I)=B(I)
20    CONTINUE

      CALL LULAW(L,U,BB,B,Y,N)

      IF(MOD(J,4).EQ.0) THEN
        do 12 i=1,n
             uxt(i)=b(i)
 12     continue
           write(6,1000) (dx*real(i),UXT(I),I=0,n+1)
           write(6,200)
      ENDIF
 100   CONTINUE

1000  FORMAT(5x,2f10.4)
 200  format(/)
      END
```

```
*
      SUBROUTINE BUNKAI(A,L,U,N)
*   LU 分解 A(N,N) をうけて L、U をかえす。
      REAL A(1:N,1:N),L(1:N,1:N),U(1:N,1:N),M
*    L,U の初期化
      DO 7 I=1,N
        DO 6 J=1,N
           L(I,J)=0.0
           U(I,J)=0.0
6       CONTINUE
        L(I,I)=1.0
7     CONTINUE
*
      DO 30 I=1,N-1
        DO 20 J=I+1,N
           M=A(J,I)/A(I,I)
           A(J,I)=M
           DO 10 K=I+1,N
              A(J,K)=A(J,K)-M*A(I,K)
10         CONTINUE
20      CONTINUE
30    CONTINUE
      DO 40 I=2,N
      DO 40 J=1,I-1
          L(I,J)=A(I,J)
40    CONTINUE
      DO 45 I=1,N
      DO 45 J=I,N
          U(I,J)=A(I,J)
45    CONTINUE
      END
*
      SUBROUTINE LULAW(L,U,B,X,Y,N)
*    LUX=B を解いて X(N) をかえす
*    LY=B を解く
      REAL L(N,N),U(N,N),B(N),X(N)
      REAL Y(N)
      Y(1)=B(1)
      DO 60 I=2,N
         S=0.0
         DO 50 J=1,I-1
            S=S+L(I,J)*Y(J)
50       CONTINUE
```

```
      Y(I)=B(I)-S
60    CONTINUE
*
*     UX=Y を解く
      X(N)=Y(N)/U(N,N)
      DO 80 I=N-1,1,-1
         S=0.0
         DO 70 J=I+1,N
         S=S+U(I,J)*X(J)
70    CONTINUE
      X(I)=(Y(I)-S)/U(I,I)
80    CONTINUE
      END
*
      REAL FUNCTION F(X)
         F=2.0*X*(1.0-X)
         RETURN
      END
```

一方 C のプログラムは以下の通り。

```
************************************************************
  シミュレーション概論

  simulation of deffusion equation (implicit scheme, double)

  pivot
  decomp
  solve
  は、ta0702.c の関数をそのまま用いている。

  コンパイルは、
  gcc tadifimp.c -lm

************************************************************/
#include <stdio.h>
#include <math.h>

#define NMAX  50       /* dx = 1 / NMAX */
#define DT    1.0/10000.0
#define N     2000     /* N step 時間を進める */
#define Cycle 400      /* Cycle step に 1 回 data を出力 */
```

```
double f(double);     /* 初期条件 */
void matrix(int, double [][NMAX+1], double);
void pivot(int, double [][NMAX+1], int*, int*, double*, int);
void decomp(int, double [][NMAX+1], int*, int);
void solve(int, double*, double [][NMAX+1], double*, int*);

main()
{
  int i, j;
  double a, dx, dt, tmax;
  double U[NMAX+1], UU[NMAX+1];
  double m[NMAX+1][NMAX+1];
  double piv;
  int ipk, ip[NMAX+1];

  dt = DT;
  dx = 1.0 / (double)NMAX;
  a = dt / dx / dx;

  /* initialization */
  for(i = 0; i <= NMAX; i++){
    U[i] = f((double)i / (double)NMAX);
    UU[i] = 0.0;
  }

  /* LU decomp. (ref. ta0702.c) */
  matrix(NMAX-1, m, a);
  for(i = 1; i <= NMAX-1; i++) ip[i] = i;
  for(i = 1; i <= NMAX-1; i++){
    pivot(NMAX-1, m, ip, &ipk, &piv, i);
    decomp(NMAX-1, m, ip, i);
  }

  for(j = 0; j <= N; j++){

    /* Data out put */
    if((j % Cycle) == 0){
      for(i = 0; i <= NMAX; i++){
printf("\t%10.4f\t%10.4f\n",dx * (double)i, U[i]);
      }
      printf("\n");
    }
```

```
    /* Implicit scheme */
    solve(NMAX-1, UU, m, U, ip);

    for(i = 1; i <= NMAX; i++) {
      U[i] = UU[i];
    }
  }
}

double f(double x)
{
  return 2.0 * x * (1.0 - x);
}

void matrix(int n, double a[][NMAX+1], double alpha)
{
  int i, j;

  for(i = 1; i <= n; i++){
    for(j = 1; j <= n; j++){
      a[i][j] = 0.0;
    }
  }
  for(i = 1; i <= n-1; i++){
    a[i][i+1] = a[i+1][i] = -alpha;
    a[i][i] = 1 + 2 * alpha;
  }
  a[n][n] = 1 + 2 * alpha;
}

void pivot(int n, double a[][NMAX+1], int* ip,
   int* ipk, double* piv, int k)
{
  int i, l;
  double absp, amax;

  l = k;
  amax = a[k][k];
  *ipk = ip[k];
  for(i = k; i <= n; i++){
    absp = fabs(a[ip[i]][k]);
    if(absp > amax){
      amax = absp;
```

```
      l = i;
    }
  }

  if(l != k){
    ip[k] = ip[l];
    ip[l] = *ipk;
    *ipk = ip[k];
  }
  *piv = a[*ipk][k];
}

void decomp(int n, double a[][NMAX+1], int *ip, int k)
{
  int i, j;
  double w[NMAX+1];

  a[ip[k]][k] = 1.0 / a[ip[k]][k];

  for(i = k + 1; i <= n; i++){
    a[ip[i]][k] = a[ip[i]][k] * a[ip[k]][k];
    for(j = k + 1; j <= n; j++){
      w[j] = a[ip[i]][j] - a[ip[i]][k] * a[ip[k]][j];
    }
    for(j = k + 1; j <= n; j++){
      a[ip[i]][j] = w[j];
    }
  }
}

void solve(int n, double *x,double a[][NMAX+1], double *b, int *ip)
{
  int i, j;
  double t;

  /* forward substitution */

  for(i = 1; i <= n; i++){
    t = b[ip[i]];
    for(j = 1; j <= i - 1; j++){
      t -= a[ip[i]][j] * x[j];
    }
    x[i] = t;
```

```
  }

  /* backward substitution */

  for(i = n; i >= 1; i--){
    t = x[i];
    for(j = i + 1; j <= n; j++){
      t -= a[ip[i]][j] * x[j];
    }
    x[i] = t * a[ip[i]][i];
  }
}
```

問 3 今までと同じ問題を陰解法で$\Delta x = 1/50, \Delta t = 1/100$ で解き、$n = 0, 4, 8, \cdots, 20$ の差分解を図示せよ。また同様の計算をΔx を固定して陽解法で行なってみよ。Δt 及び計算時間を比較せよ。

拡散方程式は空間の 1 点の情報が次の瞬間には全領域に伝わる。この性質を考えると隣合うサイトにしか情報が伝わらない陽解法より全領域に情報の伝わる陰解法の方が元の性質を保持していることは想像できよう。このことが安定性の差となったのである。

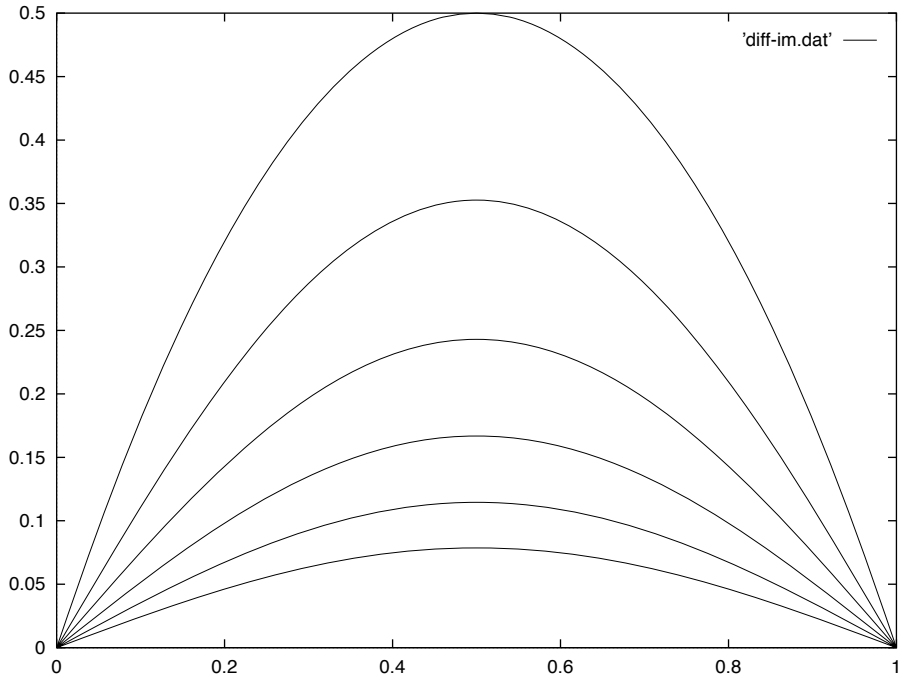

図 4: 問 3 の陰解法に基づく数値解。縦軸は拡散場の値、横軸は空間。