計算物理学 第三単元レポート

61908697 佐々木良輔

4-A.(e), (f)

プログラムについて

プログラムは基本的に教材として配布された wave.f90 を用いている．ただし標準入力によるデータ入力は使い勝手が悪かったため, 定数はソースコードにベタ書きへ変更した変更した. ソースコードはソースコード A.2 に示す．また厳密解を出力するプログラムの PAD 図は図 1 のようになる．そのソースコードはソースコード A.3 である．

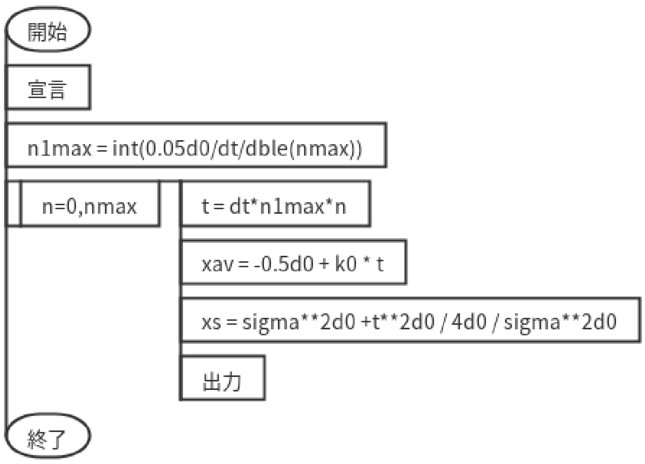

図 1　厳密解の出力プログラムの PAD 図

結果

表 1 に計算条件を示す. ここで dt は数値解の安定性条件が最も厳しくなる条件 3 で安定となるように定めた．図 2, 図 3 に各条件での $\langle x \rangle$, $\langle (x - \langle x \rangle)^2 \rangle$ の数値解及び厳密解を示す．また図 4, 図 5 に $t = 0.05$ での Δx^2 対 $\langle x \rangle$, $\langle (x - \langle x \rangle)^2 \rangle$ の数値解及び厳密解のグラフを示す.

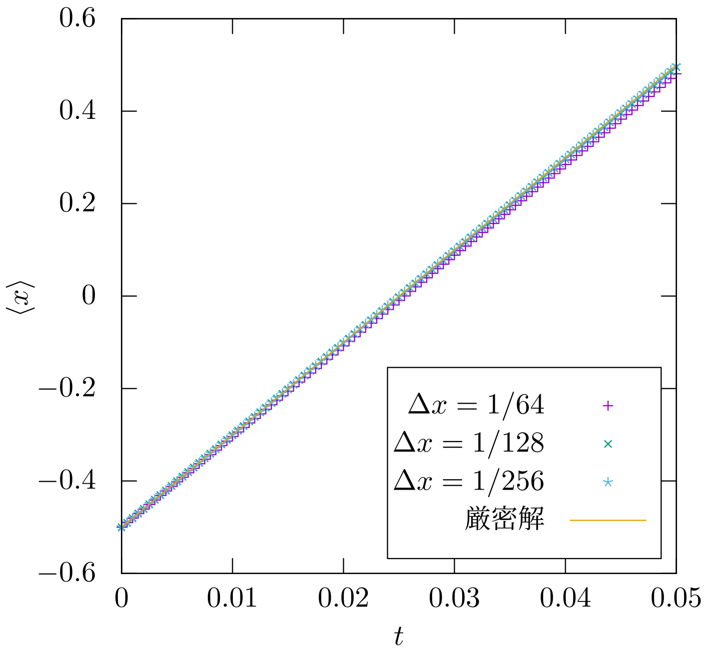

図 2　$\langle x \rangle$ の数値解と厳密解

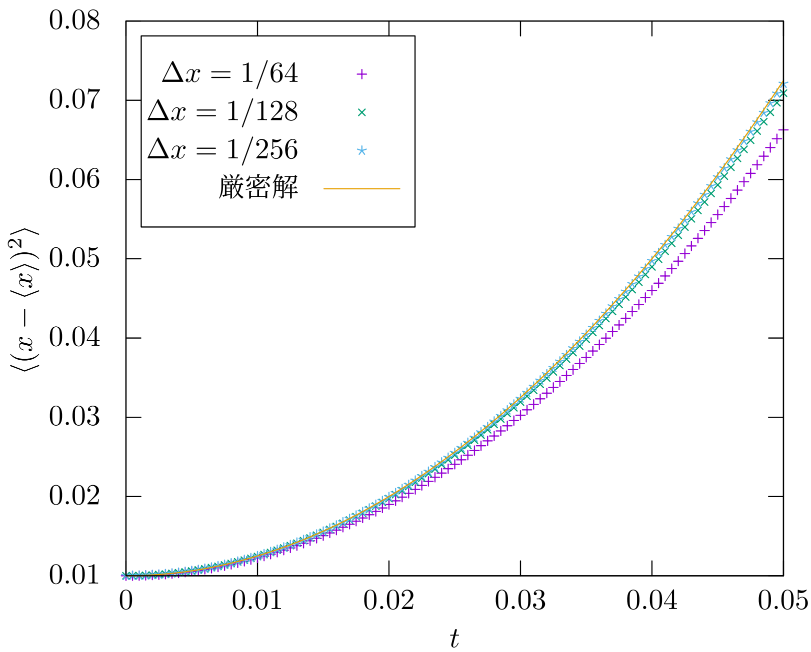

図 3　$\langle (x - \langle x \rangle)^2 \rangle$ の数値解と厳密解

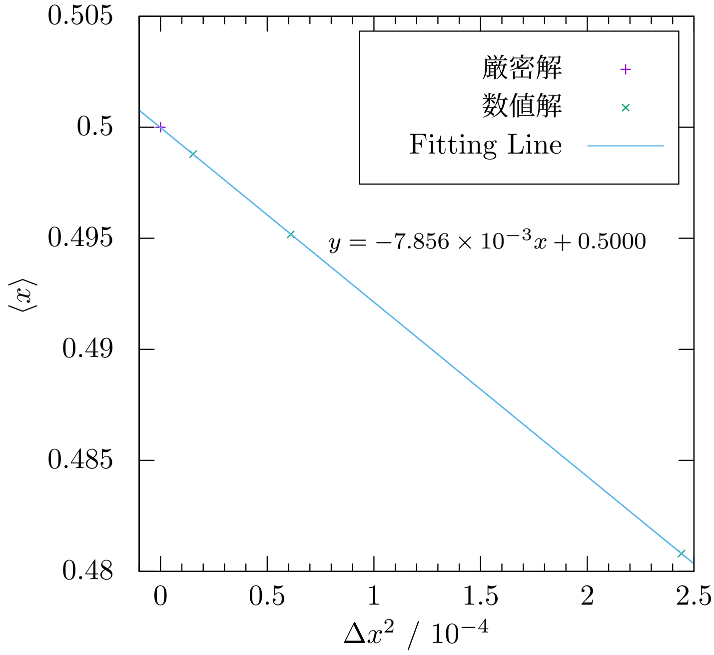

図 4　Δx^2 対 $\langle x \rangle$

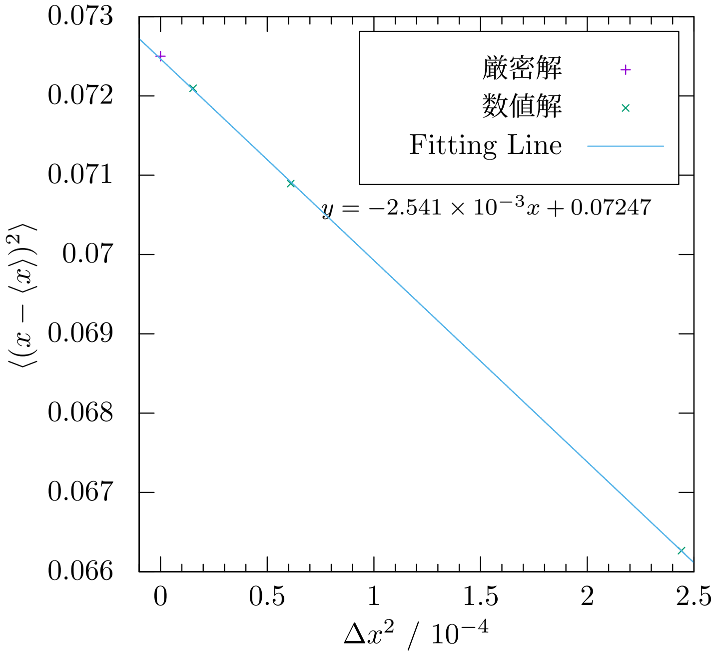

図 5　Δx^2 対 $\langle (x - \langle x \rangle)^2 \rangle$

表 1　計算条件

条件	Δx	dt	σ	k_0	L	x_0
1	1/64	1.0×10^{-5}	0.1	20	4	-0.5
2	1/128	〃	〃	〃	〃	〃
3	1/256	〃	〃	〃	〃	〃

考察

Gauss 波束の期待値, 分散の厳密解は

$$\langle x \rangle = x_0 + k_0 t \tag{1}$$

$$\langle (x - \langle x \rangle)^2 \rangle = \sigma^2 \left(1 + \frac{t^2}{4\sigma^4} \right) \tag{2}$$

であった. 図 2, 図 3 から期待値と分散がそれぞれ線形, 二次曲線的な振る舞いをしていることがわかる. また Δx が小さくなるほど数値解が厳密解に近づいていることもわかる.

また図 4 と図 5 を見ると数値解の誤差が Δx^2 に比例している様子がわかる. また数値解と Δx^2 の関係はそれぞれ最小二乗法により

$$y = -7.856 \times 10^3 x + 0.5000 \tag{3}$$

$$y = -2.541 \times 10^3 x + 0.07247 \tag{4}$$

となったので $\Delta x \to 0$ での期待値と分散は

$$\langle x \rangle = 0.5000 \tag{5}$$

$$\langle (x - \langle x \rangle)^2 \rangle = 0.07247 \tag{6}$$

となる. 一方で厳密解から得られる期待値と分散は

$$\langle x \rangle = -0.5 + 20 \times 0.05 = 0.5 \tag{7}$$

$$\langle (x - \langle x \rangle)^2 \rangle = 0.1^2 \left(1 + \frac{0.05^2}{4 \times 0.1^4} \right) = 7.25 \times 10^{-2} \tag{8}$$

となり, それぞれ相対誤差は 0.000 %, 4.138×10^{-2} % となり, 良く一致していることがわかる.

4-A. 追加問題

プログラムについて

基本的には前問のプログラムを流用しているが, 出力部をソースコード 1 のように変更し各離散点における $|\psi(x)|^2$, $\mathrm{Re}(\psi(x))$, $\mathrm{Im}(\psi(x))$ を出力するようにした．また出力も $t=0$ と $t=0.05$ でのみ行うように変更した.

ソースコード 1　出力部の改変

```
do j=1, jmax+1
  write(*,'(4e18.8e3)') x(j), r(j), real(cp(j)), aimag(cp(j))
end do
```

結果

図 6, 図 7 に $t=0$ と $t=0.05$ におけるグラフを示す．計算条件は表 1 の条件 3 と同等である. それぞれ高さ方向に実軸, 奥行き方向に虚軸を取っている.

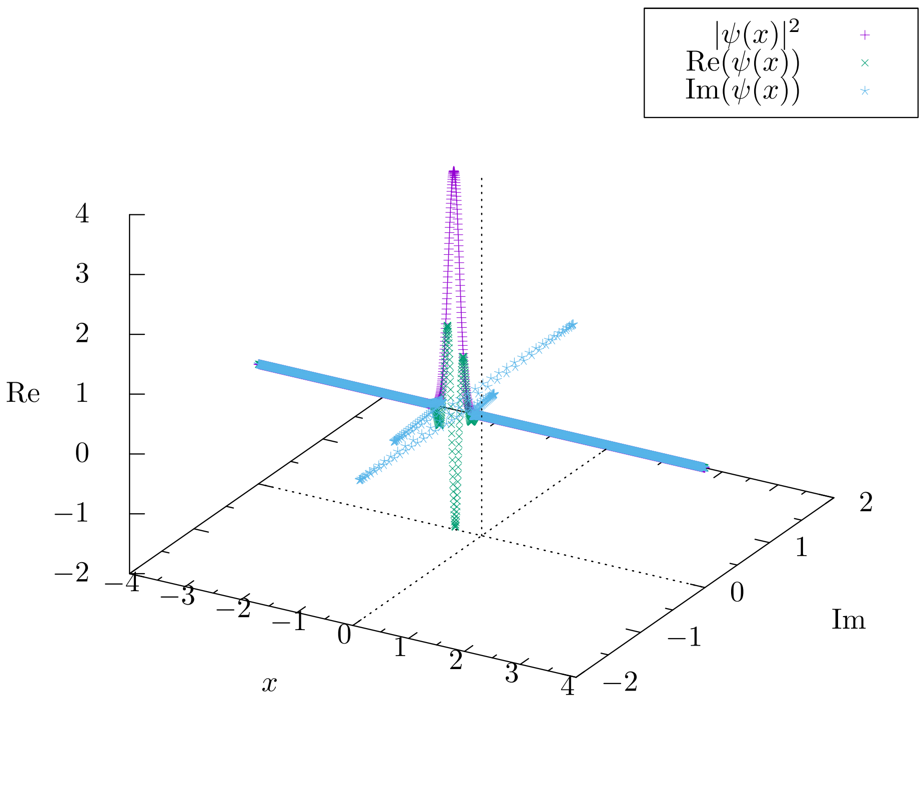

図 6　$t=0$ での波形

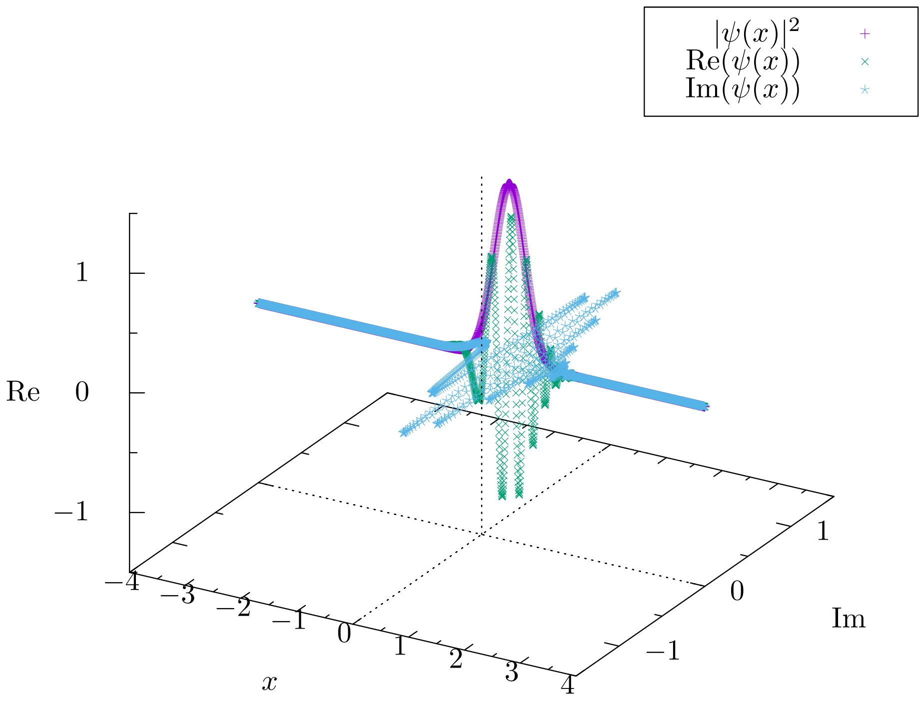

図 7　$t = 0.05$ での波形

考察

図 6, 図 7 から時間発展に伴って波束が進行しながら, 広がり鈍くなっていることがわかる．これは (1), (2) 式において期待値が時間とともに進行し, 分散が広がることと整合する．

このことは Gauss 波束を Fourier 変換することで様々な波数 k を持った平面波に展開できることから説明できる．平面波の位相速度は

$$v_k = \frac{\hbar k}{m} \tag{9}$$

であり, 波数ごとに異なった位相速度を持つことがわかる. これにより時間発展と共に波数の大きい成分と小さい成分が空間的に分離することによって Gauss 波束の広がりが大きくなる.

4-B.

プログラムについて

ソースコードをソースコード A.4 に示す．またプログラムの PAD 図は図 9 に示す．ポテンシャルを含む離散 Schrödinger 方程式は以下のようになる．

$$i\frac{\partial\psi_j}{\partial t} = -\frac{1}{2\Delta x^2}(\psi_{j+1} - 2\psi_j + \psi_{j-1}) + V\psi_j \tag{10}$$

ここでは配布された wave.x を元にポテンシャルの導入に加えて抽象化を行い可読性を向上した．ポテンシャルは図 8 のような形状のポテンシャルである．これを PAD 図のポテンシャル生成部で離散化し，配列 $V(j)$ に格納している．また k_0, σ, Δx などの定数はソースコード A.5 に示した module を用いて入力している．これによって透過率や反射率の計算を定数のすべての組み合わせについて総当り的に計算できるようになっている．

Runge-Kutta 法は第 2 回の授業で実装した Runge-Kutta 関数を複素数型に拡張した関数を用いている．Runge-Kutta 関数のソースコードはソースコード A.6 に示す．

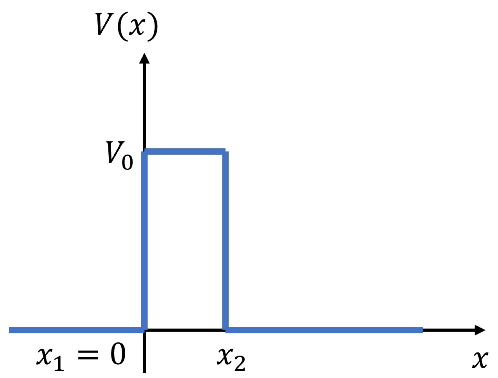

図 8　ポテンシャルの形状

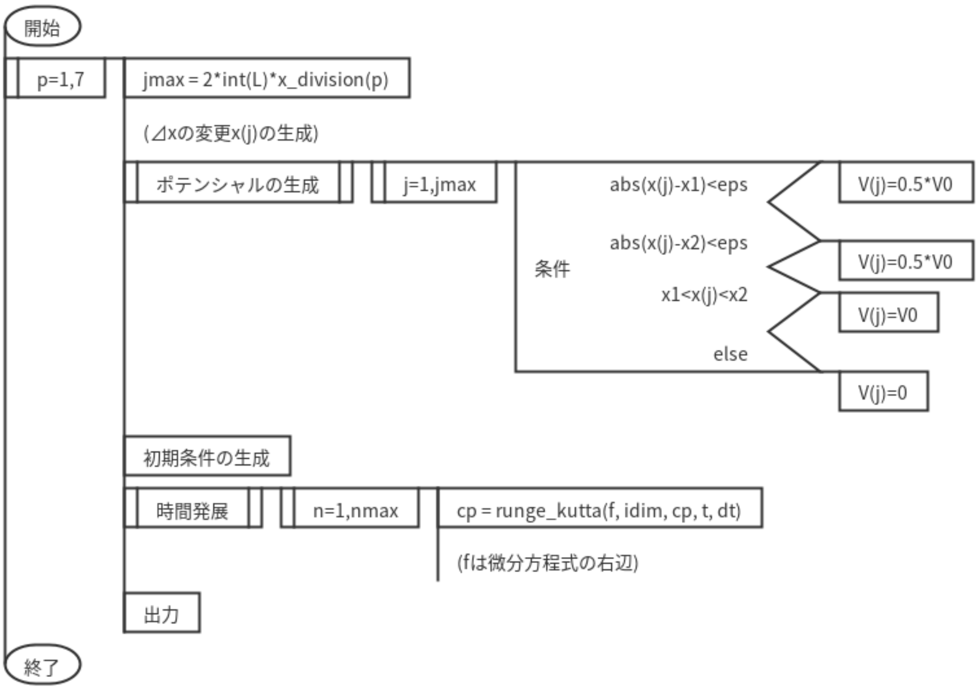

図 9　4-B の PAD 図

結果

表 2 に計算条件を示す．ここで dt は 4-A. と同様に数値解の安定性条件を満たしている．表 3 に各条件での P_{left}(反射率), P_{center}, P_{right}(透過率) 及び全確率 $P = P_{\mathrm{left}} + P_{\mathrm{center}} + P_{\mathrm{right}}$ を示す．また図 10, 図 11 に Δx^2 対 P_{left}, P_{right} のグラフを示す．

表 2　計算条件

条件	Δx	dt	σ	k_0	L	x_0	x_1	x_2	V_0
1	1/64	1.0×10^{-5}	0.6	30	5	-2.5	0	1	800
2	1/96	〃	〃	〃	〃	〃	〃	〃	〃
3	1/128	〃	〃	〃	〃	〃	〃	〃	〃
4	1/160	〃	〃	〃	〃	〃	〃	〃	〃
5	1/192	〃	〃	〃	〃	〃	〃	〃	〃
6	1/224	〃	〃	〃	〃	〃	〃	〃	〃
7	1/256	〃	〃	〃	〃	〃	〃	〃	〃

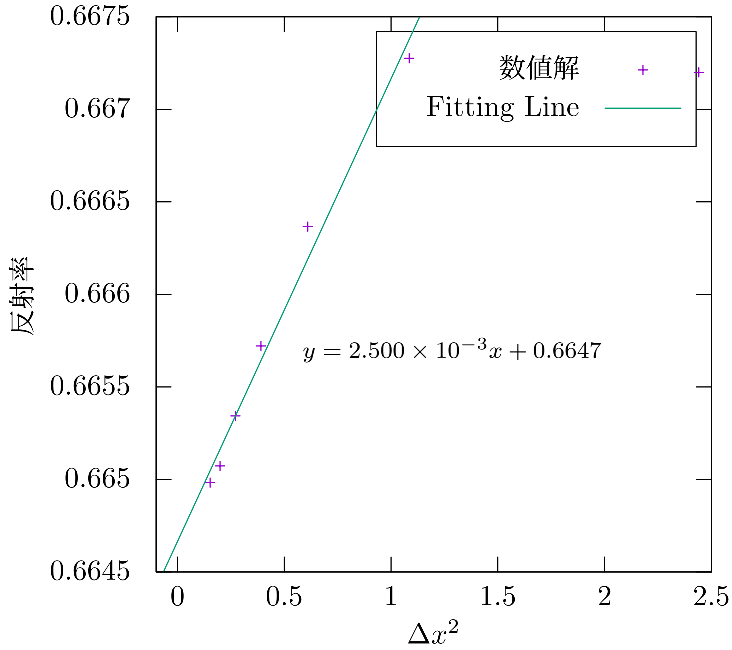

図 10　Δx^2 対反射率

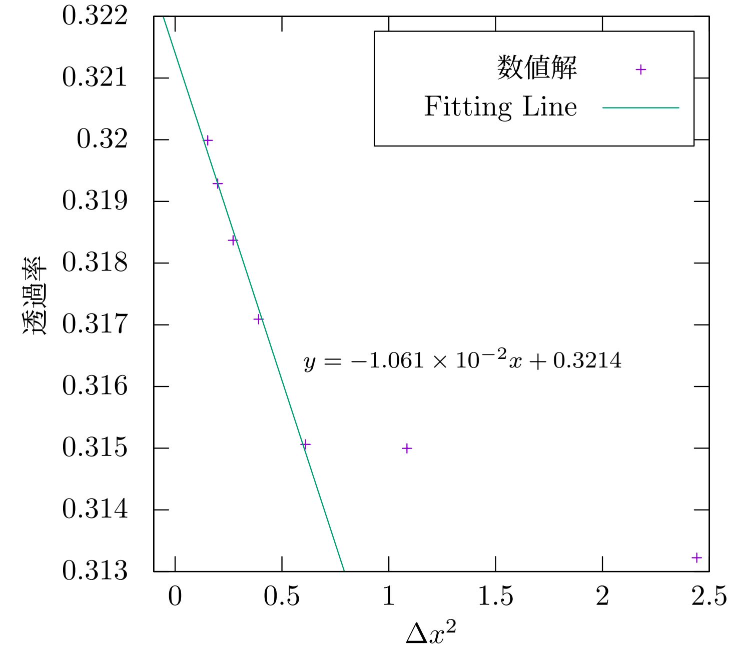

図 11　Δx^2 対透過率

表 3　計算結果

Δx	P_{left}	P_{center}	P_{right}	P
1/64	0.6671	0.01957	0.3132	1.000
1/96	0.6672	0.01772	0.3149	1.000
1/128	0.6663	0.01856	0.3150	1.000
1/160	0.6657	0.01715	0.3170	1.000
1/192	0.6653	0.01621	0.3183	1.000
1/224	0.6650	0.01553	0.3192	1.000
1/256	0.6649	0.01489	0.3199	1.000

考察

図 10, 図 11 より, 反射率, 透過率は最小二乗法より Δx^2 に対して以下のような線形関係になっている.

$$P_{\mathrm{left}} = 2.500 \times 10^{-3} z + 0.6647 \tag{11}$$

$$P_{\mathrm{right}} = -1.061 \times 10^{2} x + 0.3214 \tag{12}$$

よって $\Delta x \to 0$ においては

$$P_{\mathrm{left}} \to 0.6647 \tag{13}$$

$$P_{\mathrm{right}} \to 0.3214 \tag{14}$$

となることがわかる.

補遺 A　ソースコード

ソースコード A.2　4-A のソースコード

```
program wave
  !============================================================-
  ! sengen
  !============================================================-
  implicit none
  integer,parameter:: idim=4097
  real(kind=8):: a(4), b(4), r(idim), x(idim)
  complex(kind=8):: cp(idim), cs(idim), cps(idim), cdp(idim)
  !
  integer:: iok,jmax,nmax,n1max,j,n,n1,m
  real(kind=8):: rl,sig,rk,x0,dt,dx,w,xj,sum,rnf,xav,xs
  complex(kind=8):: ci
  !============================================================-
  ! nyuuryoku
  !============================================================-
  Nmax = 100
  rl = 4d0
  sig = 0.1d0
  rk = 20d0
  x0 = -0.5d0
  jmax = 2*int(rl)*256
  dt = 1d0*10d0**(-5d0)
  n1max = int(0.05d0/dt/dble(nmax))
  write(*,'(''# Jmax = '',i12)') jmax
  write(*,'(''# Nmax = '',i12)') nmax
  write(*,'(''# N1max = '',i12)') n1max
  write(*,'(''# L = '',e18.8e3)') rl
  write(*,'(''# sig = '',e18.8e3)') sig
  write(*,'(''# k0 = '',e18.8e3)') rk
  write(*,'(''# x0 = '',e18.8e3)') x0
  write(*,'(''# dt = '',e18.8e3)') dt
  !============================================================-
  ! junbi
  !============================================================-
  a(1)=0.0d0
  a(2)=0.5d0
  a(3)=0.5d0
  a(4)=1.0d0
```

```
b(1)=1.0d0/6.0d0
b(2)=1.0d0/3.0d0
b(3)=1.0d0/3.0d0
b(4)=1.0d0/6.0d0
dx=2.0d0*rl/dble(jmax)
w =0.5d0/dx**2
ci=(0.0d0,1.0d0)
do j=1,jmax+1
  x(j)=dx*dble(j-1)-rl
enddo
!========================================================-
! shoki-jooken
!========================================================-
sum=0.0d0
do j=2,jmax
  xj=x(j)
  cp(j)=cdexp(ci*rk*xj-((xj-x0)/(2.0d0*sig))**2)
  sum=sum+cp(j)*dconjg(cp(j))
enddo
cp(1) =(0.d0,0.d0)
cp(jmax+1)=(0.d0,0.d0)
rnf=1.0d0/dsqrt(sum*dx)
do j=2,jmax
  cp(j)=rnf*cp(j)
enddo
!
n=0
write(*,'( ''#''/ ''# time'',12x,'' <x> '',12x, '' <(x-<x>)**2>'' )')
!========================================================-
! shuukei
!========================================================-
! r=:|psi|^2, cp=:psi
do j=2,jmax
  r(j)=cp(j)*dconjg(cp(j))
enddo
! mean x
xav=0.0d0
do j=2,jmax
  xav=xav+x(j)*r(j)*dx
enddo
! variance x
xs=0.0d0
```

```
do j=2,jmax
  xs=xs+((x(j)-xav)**2)*r(j)*dx
enddo
write(*,'(3e18.8e3)') dt*n1max*n,xav,xs
!
cdp(:)=0.0d0
!===========================================================-
do n=1,Nmax
  do n1=1,N1max
    !=================================================-
    ! 1 step sekibun
    !=================================================-
    do j=2,jmax
      cs(j)=(0.0d0,0.0d0)
    enddo
    do m=1,4
      do j=2,jmax
        cps(j)=cp(j)+a(m)*cdp(j)
      enddo
      do j=2,jmax
        cdp(j)=(cps(j+1)-2.0d0*cps(j)+cps(j-1))*w*ci*dt
      enddo
      do j=2,jmax
        cs(j)=cs(j)+b(m)*cdp(j)
      enddo
    enddo
    do j=2,jmax
      cp(j)=cp(j)+cs(j)
    enddo
  enddo
  !===========================================================-
  ! shuukei
  !===========================================================-
  do j=2,jmax
    r(j)=cp(j)*dconjg(cp(j))
  enddo
  xav=0.0d0
  do j=2,jmax
    xav=xav+x(j)*r(j)*dx
  enddo
  xs=0.0d0
  do j=2,jmax
```

```
      xs=xs+((x(j)-xav)**2)*r(j)*dx
    enddo
    write(*,'(3e18.8e3)') dt*n1max*n,xav,xs
  enddo
end program wave
```

ソースコード A.3　厳密解の計算プログラム

```
program main
  implicit none
  DOUBLE PRECISION, PARAMETER :: dt = 1d0*10d0**(-5d0)
  DOUBLE PRECISION, PARAMETER :: x0 = -0.5d0, k0 = 20d0, sigma=0.1d0
  DOUBLE PRECISION :: xav, xs, t
  INTEGER, PARAMETER :: nmax = 100
  INTEGER :: n1max = int(0.05d0/dt/dble(nmax)), n
  write(*,'( ''#''/ ''# time'',12x,'' <x> '',12x, '' <(x-<x>)**2>'' )')
  do n=0,nmax
    t = dt*n1max*n
    xav = -0.5d0 + k0 * t
    xs = sigma**2d0 +t**2d0 / 4d0 / sigma**2d0
    write(*,'(3e18.8e3)') t,xav,xs
  end do
end program main
```

ソースコード A.4　4-B. のソースコード

```
program main
  use differential
  use consts
  implicit none
  INTEGER, PARAMETER :: idim=4096
  COMPLEX(KIND(0d0)) :: cp(idim)
  COMPLEX(KIND(0d0)), PARAMETER :: ci = (0d0, 1d0)
  INTEGER :: jmax,nmax,j1,j2,p,q,s
  DOUBLE PRECISION :: sigma,k0,dx,w,t
  DOUBLE PRECISION :: r(idim),x(idim),V(idim)=0d0

  do s=2,2
    sigma = sigma_array(s)
    do q=1,1
      k0 = k0_array(q)
      nmax = int(3d0/2d0*L/k0/dt)
      write(*,'(''# sig = '',e18.8e3)') sigma
      write(*,'(''# k0 = '',e18.8e3)') k0
      do p=1,7
        jmax = 2*int(L)*x_division(p)
        dx = 2.0d0*L/dble(jmax)
        w = 0.5d0/dx**2
        call main_roop()
      end do
    end do
  enddo


  contains
  subroutine main_roop()
    implicit none
    INTEGER :: j,n,n1
    DOUBLE PRECISION, PARAMETER :: eps = 2d0**(-52)
    DOUBLE PRECISION :: sum,rnf
    !=========================================================-
    ! generate grid points
    !=========================================================-
    do j=1,jmax+1
      x(j)=dx*dble(j-1)-L
    enddo
    !=========================================================-
```

```
! generate potential array
!============================================================-
do j=2,jmax
  if (abs(x(j)-x1).le.eps) then
    V(j) = V0*0.5d0
  else if (abs(x(j)-x2).le.eps) then
    V(j) = V0*0.5d0
  else if ((x(j).gt.x1).and.(x(j).lt.x2)) then
    V(j) = V0
  else
    V(j) = 0d0
  end if
end do
j = 0
do while (V(j).eq.0d0)
  j = j + 1
end do
j1 = j
j = jmax
do while (V(j).eq.0d0)
  j = j - 1
end do
j2 = j
!============================================================-
! inital condition
!============================================================-
sum = 0d0
n = 0
n1 = 0
do j=2,jmax
  cp(j) = cdexp(ci*k0*x(j)-((x(j)-x0)/(2d0*sigma))**2d0)
  sum = sum+cp(j)*dconjg(cp(j))
enddo
cp(1) = (0d0,0d0)
cp(jmax+1) = (0d0,0d0)
rnf = 1.0d0/dsqrt(sum*dx)
do j=2,jmax
  cp(j)=rnf*cp(j)
enddo
!call output_xav_xs()
!============================================================-
! time evolution
```

```
  !==========================================================-
  do n=1,nmax
    cp = runge_kutta(f, idim, cp, t, dt)
    !call output_xav_xs()
  end do
  do j=2,jmax
    r(j) = cp(j)*dconjg(cp(j))
  enddo
  !call output_wave()
  call output_reflect_ratio()
end subroutine main_roop

subroutine output_reflect_ratio()
  implicit none
  DOUBLE PRECISION :: Pleft, Pcenter, Pright
  INTEGER :: j
  Pleft = 0d0
  Pcenter = 0d0
  Pright = 0d0
  do j=2, j1-1
    Pleft = Pleft + r(j)*dx
  end do
  do j=j1, j2
    Pcenter = Pcenter + r(j)*dx
  end do
  do j=j2+1, jmax
    Pright = Pright + r(j)*dx
  end do
  write(*,'(5e18.8e3)') dx, Pleft, Pcenter, Pright, Pleft+Pcenter+Pright
end subroutine output_reflect_ratio

subroutine output_wave()
  implicit none
  INTEGER :: j
  do j=1, jmax+1
    write(*,'(4e18.8e3)') x(j), r(j), real(cp(j)), aimag(cp(j))
  end do
end subroutine output_wave

subroutine output_xav_xs()
  implicit none
  INTEGER :: j
```

```
    DOUBLE PRECISION :: xav,xs
    xav=0.0d0
    do j=2,jmax
      xav = xav+x(j)*r(j)*dx
    enddo
    xs=0.0d0
    do j=2,jmax
      xs = xs+((x(j)-xav)**2)*r(j)*dx
    enddo
    write(*,'(3e18.8e3)') t,xav,xs
  end subroutine output_xav_xs

  function f(t, x, n, const)
    INTEGER, INTENT(IN) :: n
    DOUBLE PRECISION :: t
    COMPLEX(KIND(0d0)) :: f(n)
    COMPLEX(KIND(0d0)), INTENT(IN) :: x(:)
    COMPLEX(KIND(0d0)), OPTIONAL :: const(:)
    INTEGER :: j
    do j=2,jmax
      f(j)=((x(j+1)-2.0d0*x(j)+x(j-1))*w - V(j)*x(j))*ci
    enddo
    f(1) = (0d0,0d0); f(jmax + 1) = (0d0,0d0)
  end function f
end program main
```

ソースコード A.5　定数 module

```
module consts
  implicit none
  INTEGER, PARAMETER :: x_division(7) = (/64, 96, 128, 160, 192, 224, 256/)
  DOUBLE PRECISION, PARAMETER :: L = 5d0
  DOUBLE PRECISION, PARAMETER :: sigma_array(3) = (/0.4d0,06d0,0.8d0/)
  DOUBLE PRECISION, PARAMETER :: k0_array(7) = (/30d0,35d0,40d0,45d0,50d0,55d0,60d0/)
  DOUBLE PRECISION, PARAMETER :: x0 = -L/2
  DOUBLE PRECISION, PARAMETER :: dt = 1d0*10d0**(-5d0)
  DOUBLE PRECISION, PARAMETER :: x1 = 0d0
  DOUBLE PRECISION, PARAMETER :: x2 = 1d0
  DOUBLE PRECISION, PARAMETER :: V0 = 800d0

end module consts
```

ソースコード A.6　Runge-Kutta 関数のソースコード

```
module differential
  implicit none

  contains
  !runge_kutta の計算を1 ステップ進める
  !arg: 微分方程式; n: 連立する数; init: 初期値; t_begin: 計算開始; t_end: 計算終了; tau: 刻み幅;
  !const: 定数 (optional); boundary: 境界条件 (optional)
  function runge_kutta(arg, n, init, t_begin, tau, const, boundary)
    implicit none
    interface
      ! 微分方程式
      function arg(t, x, n, const)
        INTEGER, INTENT(IN) :: n
        DOUBLE PRECISION :: arg(n)
        DOUBLE PRECISION, INTENT(IN) :: x(:), t
        DOUBLE PRECISION, OPTIONAL :: const(:)
      end function arg
      ! 境界条件の計算関数
      function boundary(x, n)
        implicit none
        INTEGER, INTENT(IN) :: n
        DOUBLE PRECISION, INTENT(IN) :: x(:)
        DOUBLE PRECISION, OPTIONAL :: boundary(n)
      end function boundary
    end interface
    INTEGER :: i
    ! 段数
    INTEGER, PARAMETER :: order = 4
    INTEGER, INTENT(IN) :: n
    DOUBLE PRECISION, INTENT(in) :: init(:), tau
    DOUBLE PRECISION, INTENT(INOUT) :: t_begin
    DOUBLE PRECISION :: runge_kutta(n), x(n), t, s(n), delta(n)
    DOUBLE PRECISION, OPTIONAL :: const(:)
    DOUBLE PRECISION :: a(order), b(order)
    a(1)=0d0; a(2)=0.5d0; a(3)=0.5d0; a(4)=1d0
    b(1)=1d0/6d0; b(2)=1d0/3d0; b(3)=1d0/3d0; b(4)=1d0/6d0
    x(:) = init(:)
    t = t_begin

    s = 0; delta = 0
```

```
    do i = 1, order
      ! optional の定数が与えられている場合はそれを含む計算を実行
      if (PRESENT(const)) then
        delta(:) = arg(t+a(i)*tau, x(:) + a(i) * delta, n, const)*tau
      else
        delta(:) = arg(t+a(i)*tau, x(:) + a(i) * delta, n)*tau
      end if
      s(:) = s(:) + b(i) * delta(:)
    end do
    x(:) = x(:) + s(:)
    t = t + tau
    t_begin = t
    ! optional の境界条件が与えられている場合はそれを考慮
    if (PRESENT(boundary)) then
      x(:) = boundary(x, n)
    end if
    runge_kutta(:) = x(:)
  end function runge_kutta
end module differential
```