

REFERENCES

1. Aydin, A. & B. Karasozen. 2007. "Symplectic and multisymplectic methods for coupled nonlinear Schrödinger equation with periodic solutions." *Computer physics communications*. 177, pp. 566 – 583.
2. Beiser, A. 2003. *Concept of Modern Physics*; Six edition, New York: Mc Graw Hill.
3. Brown, L.M. & H. Rechenberg, 2005. "P.A.M Dirac (1930) and J. Von Neumann (1932), Books on Quantum mechanics." *Landmark writing in western mathematics*, pp. 1640 – 1940.
4. Canosa, J. & Oliveira, R. G.D 1970. "A new method for the solution of Schrödinger equation". *Journal of Computational Physics* (5) pp. 188 – 207.
5. Cutlip, M.B & M.Shacham 2001. "The Numerical Method of Lines for Partial Differential Equations. PP 1-6.
6. Zettili, N. 2007. *Quantum Mechanics, Concept and Application*. England: Wiley.
7. -Chang, S. -L.,C. -S. Chien, & B. -W. Jeng, "Computing wave function of nonlinear Schrödinger equation; A time – dependent approach." *Journal of computational physics*. 226, pp. 104 – 130.
8. Chang, I-S, & S-Y, Kang. 2008. "Fredholm integral equation method for the integro – differential Schrödinger equation." *Computer & Mathematics with applications* (56). pp. 2676 – 2685.
9. de la Hoz, F. & F. Vadillo, 2008. "An exponential time differencing method for the non linear Schrödinger equation". *Computer Physics communication*, 179, pp.449 – 456.
10. Demirkaya, G., Arino, F., Selcuk, N. & Ayrançi, I. 2005. "Comparison between performance of Monte Carlo method and method of lines solution of discrete ordinate method." *Journal of quantitative spectroscopy and Radiative transfer* 93, pp. 115 – 124.
11. Fibich, G., N. Gavish, & X-P, Wang. 2005. "New singular solution of nonlinear Schrödinger equation." *Physica D* 211. pp. 193 – 220.
12. Foudas, C. 2007. "Relativistic quantum Mechanics". Lecture 5, 01 February 2007. PP1-11.
13. French, A.P & Taylor, E. F. 1978. *An Introduction to Quantum Physics*. USA: CRC Press.
14. Hamdan, N., A. Altorra, & H.A Salman, 2007. "Modified Dirac equation with classical Zitterbewegung." *Proc. Pakistan Acad. Sci.* 44(4). Pp. 263 – 272.
15. Hamdi, S., W.E Schiesser & G.w. Griffith, 17 October 2009." Method of Lines: Part I: Basic Concepts. <http://www.scholarpedia.org/article/method_of_lines>.
16. Han, H., J. Jin, & X. Wu. 2005. "A finite difference method for the one – dimensional Time – dependent Schrödinger equation on unbounded domain." *Computer and Mathematics with application*. 50, pp. 1345 – 1362.

17. Hotson, D.L. 2002. "Dirac's Equation and Sea of Negative Energy". *Infinite Energy* ISSUE (43). PP.1-20.
18. Iitaka, T., N. Carjan, & D. Strottman, 1995. "Stability of Symmetric Multistep methods for time – dependent Schrödinger equation." *Computer Physics communications*. 90, pp. 251 – 259.
19. Inui, N. 2007. "Numerical errors resulting from finite difference approximation in computations of a one – dimensional Schrödinger equation with Trotter formula." *Computer physics communication* 177, pp. 781 – 786.
20. Jime'nez, S. M.I. L lorente, M.A. Mancho, M.V. Pere'z – Garcia' & L. Va'zquez. 2003. "A numerical scheme for the simulation of blow – up of the nonlinear Schrödinger equation." *Applied mathematics and computation* 134, pp. 271 – 291.
21. Kalogiratou, Z., T. Monovasilis & T.E Simos. 2009. "Computation of the eigen values of Schrödinger equation by exponentially – fitted Runge – Kutta – Nystrom methods." *Computer Physics communication*. 180, pp. 167-176.
22. Koch, O., W. Kreuzer, & A. Scrinzi, 2006. "Approximation of the time – dependent electronic Schrödinger equation by MCTDHF." *Applied mathematics and computation*. 173, pp. 960 – 976.
23. Kosloff, D. & Kosloff, R. 1982. "A Fourier Method Solution of Time Dependent Schrödinger Equation as a Tool in Molecular dynamics". *Journal of Computational Physics* (52): PP. 35-53.
24. Lambert, N. 2001 "Numerical Solutions of Schrödinger Equation TB2".pp 1-24.
25. Lee, H.Y. 1994. "Fully discrete method for the Nonlinear Schrödinger equation." *Computer mathematics and application* Vol. 28, No. 6, pp. 9 – 24.
26. Lee, H.S., C.J. Matthews, R.D. Braddock, G. C Sander, & F. Gandola. 2003. "A MATLAB method of lines template for transport equations". *Environmental modeling & software* 19, pp.603 – 614.
27. M., V. Ledoux, M. Van Daele, G. Vanden Berghe, & N. Carjan. 2008. "Finite difference approach for the two – dimensional Schrödinger equation with application to scission – neutron emissions." *Computer Physics communication*. 179, pp. 466 – 478.
28. Mazzone, A.M & V. Morandi, 2006. "Solution of time – dependent multi – particle Schrödinger equation using Monte Carlo and numerical integration." *Computational material sciences*. 38, pp. 231 – 239.
29. Moebs, G. 1998. "A multi level method for the resolution of a Stochastic weakly damped nonlinear Schrödinger equation". *Applied numerical mathematics* (26). Pp. 353 – 375.
30. Muslu, G.M. & H.A. Erbay, 2005. "Higher – order split – step Fourier scheme's for the generalized nonlinear Schrödinger equation. " *Mathematics and computers in simulation*. 67, pp.581 – 595.
31. n.a. 04 November 2009. "Method of Lines". *Wikipedia, the free encyclopedia*. <http://enc.wikipedia.org/wiki/method_of_lines>.
32. n.a. 07 November 2009. "History of Quantum Mechanics" *Wikipedia, free encyclopedia* < http://en.wikipedia.org/wiki/History_of_quantum_mechanics>.

33. Nakanishi, H. & M. Suguwara. 2000. "Numerical solution of Schrödinger equation by a magnetic algorithm." *Chemical Physics letters*. 327, pp. 429 – 438.
34. Nash, P.L. 2008. "A new Fourth order Fourier – Bessel split – steps method for the extended nonlinear Schrödinger equation." *Journal of computational physics*. 227, pp. 2073 – 2082.
35. Nizar Hamdan, A. Altorra & H.A Salman. 2007. "Modified Dirac equation with classical Zitterbewegung". *Proc. Pakistan, Acad. Sci.* 44(4). 263-272.
36. Phillips, A.C, D.J Sandiford & F. Mandl (ed.). 2003. Introduction to Quantum Mechanics. The Manchester Physics Series, England: Wiley.
37. Pregla, R. 2008. Analysis of Electromagnetic field and waves; The method of lines. England: Wiley.
38. Swanson, D.G. 2006. Quantum Mechanics, Foundations and Applications, New York: Taylor & Francis.
39. Ramos, J.I 2002. "Linearly implicit method for nonlinear Schrödinger equation in non homogeneous media." *Applied mathematics and computation* 133, pp. 1 – 28.
40. Ramos, J. I. & F.R. Villatoro. 1994. "The nonlinear Schrödinger equation in the Finite line." *Mathematical computer modeling*. Vol. 20; No. 3, pp. 31 – 59.
41. Ramos, j. I. F.R. Villatoro. 1994. "A Quantum mechanics analogy for the nonlinear Schrödinger equation in the finite line." *Computer mathematics application*. Vol. 28, No. 4, pp. 3 – 17.
42. Rainville, E.D, P.E Bedient & R.E Bedient 1997. Elementary Differential equation. (Eight Edition) England: Simon & Schuster Rizea,
43. Sadiku, M.N.O & C.N Obiozor. 2000. "A Simple Introduction to the Method of Lines." *International Journal of Electrical Engineering education* (37/3): PP. 283-296.
44. Samrout, Y.M. 2007. "New second- and Fourth- order accurate numerical schemes for the cubic Schrödinger equation". *International Journal of Computer Mathematics* vol. 84. No.11, pp. 1625 – 1651.
45. Susan, M.L 2004. Mathematics for Physicists. USA: (eight editions). England: Simon & Schuster.
46. Simus, T.E & G.V. Mitsou, 1994. "An expert system for the Numerical solution of one – dimensional Schrödinger equation". *Mathematics & Computers in simulation* (36) pp. 195-202.
47. Schwabl, F. 2007. Quantum Mechanics. Fourth edition, USA: Springer.
48. Schiesser, W.E & G.W Griffith, 2009. "A Compendium of Partial Differential Equation Model: Method of Lines analysis with Mat lab. England: Cambridge Universal Press.
49. Schiesser, W.E. 1991. Numerical method of lines; Integration of partial differential equation. California: Academic press limited.
50. Shakeri, F. & M. Dehghan, 2008. "The method of lines for one – dimensional wave equation subject to an integral conservation condition". *Computer and mathematics with applications* 56, pp. 2175 – 2188.

51. Shao, S., W. Cai. & H. Tang 2006. "Accurate calculation of Green's function of Schrödinger equation in a block layered potential." *Journal of Computational Physics* (219) pp. 733-748.
52. Stern, M.S. 1976. "Comparison of Numerical solutions of partial – wave Schrödinger differential and integral equations." *Journal of computational physics*. 25, pp. 56 – 70.
53. Tang, C.L, 2005. *Fundamental of Quantum Mechanics for solid state electronics and optics*. New York: Cambridge university press.
54. Utsumi, A. Takayuki, K. James & Y. Mitsuru. 2006. ". Solution of 1D Couple and Non linear Schrödinger equation by the CIP-BS Method". *Communication in Computational Physics* vol. 1. No.2 , pp. 261-275
55. Van de vyver, H. 2005. "Modified explit Runge – Kutta methods for the numerical solution of Schrödinger equation." *Applied mathematics and computation* 171, pp. 1025 -1036.
56. Wang, T., T. Nie., L. Zhang, & F. Chen, 2008. "Numerical simulation of Nonlinear coupled Schrödinger system: A linear uncoupled finite difference scheme." *Mathematics and computer simulation*. 79, pp. 607-621.
57. Wouwer, A.V., P. Saucez, & W.E. Schiesser, 2009. *Adaptive method of line*; New York: Tailor & Francis.
58. Zisowsky, A. & M. Ehrhardt. 2008. "Discrete artificial boundary condition for nonlinear Schrödinger equations." *Mathematical and computer modeling*. 47, pp.1264 – 1283.
59. Zheng, C. 2007. "A perfectly matched layer approach to the nonlinear Schrödinger wave equations." *Journal of computational physics* 227, pp. 537 – 556.

APPENDICES

Appendix A:

Below is the program used for the output of Figure 3.3 a main program extracted from differential ODE routines of the compendium of partial differential equations chapter 2 by Schiesser & Griffith (2009) and then modified to suit our request.

We labeled it as *Listing 3.0.1*

```
% Clear previous files
clear all
clc
%
% Parameters shared with the ODE routine
global ncall ndss
%
% Initial condition
n=31;
for i=1:n
    u0(i)=sin((pi/2.0)*(i-1)/(n-1));
end
%
% Independent variable for ODE integration
t0=0.0;
tf=5.0;
tout=linspace(t0,tf,n);
nout=n;
ncall=0;
%
% ODE integration (the nonstiff integrator ode45, requires
% ca 4,000 to 13,000 calls to pde_1.m, pde_2.m, pde_3.m;
% thus, the use of ode15s below)
mf=3;
reltol=1.0e-04; abstol=1.0e-04;
options=odeset('RelTol',reltol,'AbsTol',abstol);
if(mf==1) % explicit FDs
    [t,u]=ode15s(@pde_1,tout,u0,options); end
if(mf==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_2,tout,u0,options); end
if(mf==3) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
```

```

[t,u]=ode15s(@pde_3,tout,u0,options); end
%
% Store numerical and analytical solutions, errors at  $x = 1/2$ 
n2=(n-1)/2.0+1;
sine=sin(pi/2.0*0.5);
for i=1:nout
    u_plot(i)=u(i,n2);
    u_anal(i)=exp(-pi^2/4.0*t(i))*sine;
    err_plot(i)=u_plot(i)-u_anal(i);
end
%
% Display selected output
fprintf('\n mf = %2d  abstol = %8.1e  reltol = %8.1e\n',...
        mf,abstol,reltol);
fprintf('\n  t    u(0.5,t)  u_anal(0.5,t)  err u(0.5,t)\n');
for i=1:5:nout
    fprintf('%6.3f%15.6f%15.6f%15.7f\n',...
            t(i),u_plot(i),u_anal(i),err_plot(i));
end
fprintf('\n ncall = %4d\n',ncall);
%
% Plot numerical solution and errors at  $x = 1/2$ 
figure(1);
subplot(1,2,1)
plot(t,u_plot); axis tight
title('u(0.5,t) vs t'); xlabel('t'); ylabel('u(0.5,t)')
subplot(1,2,2)
plot(t,err_plot); axis tight
title('Err u(0.5,t) vs t'); xlabel('t'); ylabel('Err u(0.5,t)')
print -deps -r300 pde.eps; print -dps -r300 pde.ps; print -dpng -r300 pde.png
%
% Plot numerical solution in 3D perspective
figure(2);
colormap('Gray');
C=ones(n);
g=linspace(0,1,n); % For distance x
h1 = waterfall(t,g,u',C);
axis('tight');
grid off
xlabel('t, time')
ylabel('x, distance')
zlabel('u(x,t)')
s1 = sprintf('Diffusion Equation - MOL Solution');

```



```

sTmp = sprintf('u(x,0) = sin(\ \pi x/2)');
s2 = sprintf('Initial condition: %s', sTmp);
title([s1, s2], 'fontsize', 12);
v = [0.8616 -0.5076 0.0000 -0.1770
     0.3712 0.6301 0.6820 -0.8417
     0.3462 0.5876 -0.7313 8.5590
     0 0 0 1.0000];
view(v);
rotate3d on;

```

Three cases were programmed corresponding to $mf=1,2,3$ for which ODE routines, `pde_1`, `pde_2` and `pde_3` are called. The variable `ndss` refers to a library of differential routines for use in the MOL solution of PDEs. We listed below this `dss` (they are 42, 44, 48 or 50) they are quoted from the same reference mention above.

```

% File: dss042.m
%
function [uxx]=dss042(xl,xu,n,u,ux,nl,nu)
%
% Function dss042 computes a second-order approximation of a
% second-order derivative, with or without the normal derivative
% at the boundary.
%
% Argument list
%
% xl    Left value of the spatial independent variable (input)
%
% xu    Right value of the spatial independent variable (input)
%
% n     Number of spatial grid points, including the end
%       points (input)
%
% u     One-dimensional array of the dependent variable to be
%       differentiated (input)
%
% ux    One-dimensional array of the first derivative of u.
%       The end values of ux, ux(1) and ux(n), are used in
%       Neumann boundary conditions at  $x = xl$  and  $x = xu$ ,
%       depending on the arguments nl and nu (see the de-
%       scription of nl and nu below)

```

```

%
% uxx One-dimensional array of the second derivative of u
% (output)
%
% nl Integer index for the type of boundary condition at
% x = xl (input). The allowable values are
%
% 1 - Dirichlet boundary condition at x = xl
% (ux(1) is not used)
%
% 2 - Neumann boundary condition at x = xl
% (ux(1) is used)
%
% nu Integer index for the type of boundary condition at
% x = xu (input). the allowable values are
%
% 1 - Dirichlet boundary condition at x = xu
% (ux(n) is not used)
%
% 2 - Neumann boundary condition at x = xu
% (ux(n) is used)
%
% The following derivation is for a set of second-order four-point
% approximations for a second derivative that can be used at the
% boundaries of a spatial domain. These approximations have the
% features
%
% (1) Only interior and boundary points are used (i.e., no
% fictitious points are used)
%
% (2) The normal derivative at the boundary is included as part
% of the approximation for the second derivative
%
% (3) Approximations for the boundary conditions are not used.
%
% The derivation is by Professor Gilbert A. Stengle, Department of
% Mathematics, Lehigh University, Bethlehem, pa 18015, and was done
% on December 7, 1985.
%
% For an approximation at the left boundary, involving the points
% i, i+1, i+2 and i+3, consider the following Taylor series expan-
% sions
%

```


$$u(i+1) = u(i) + \frac{ux(i)(dx)}{1} + \frac{uxx(i)(dx)^2}{2} + \frac{uxxx(i)(dx)^3}{6} + \dots$$

$$u(i+2) = u(i) + \frac{ux(i)(2dx)}{1} + \frac{uxx(i)(2dx)^2}{2} + \frac{uxxx(i)(2dx)^3}{6} + \dots$$

If we now form the following linear combination, involving constants a , b , c and d to be determined, and use the preceding two Taylor series,

$$a*u(i) + b*ux(i) + c*u(i+1) + d*u(i+2)$$

we have

$$\begin{aligned} & a*u(i) + b*ux(i) + c*u(i+1) + d*u(i+2) = \\ & (a + b + c + d)*u(i) + \\ & (b + dx*c + 2*dx*d)*ux(i) + \\ & (c*(dx^2)/2 + d*((2*dx)^2)/2)*uxx(i) + \\ & (c*(dx^3)/6 + d*((2*dx)^3)/6)*uxxx(i) + O(dx^4) \end{aligned}$$

The third derivative, $uxxx(i)$, can be dropped by taking

$$c = -8*d$$

The second derivative, $uxx(i)$, can be retained by taking

$$(dx^2)(c/2 + 2*d) = 1$$

which, when combined with the preceding result gives

$$d = -1/(2*(dx^2))$$

$$c = 4/(dx^2)$$

The first derivative, $ux(i)$, can be dropped by taking

$$\% \quad b + dx*c + 2*dx*d = 0$$

%

% or

%

$$\% \quad b = -dx*c - 2*dx*d = -4/dx - 2*dx*(-1/(2*(dx**2))) = -3/dx$$

%

% Finally, $u(i)$, can be dropped by taking

%

$$\% \quad a = -c - d = 8*d - d = -7*d = -7/(2*(dx**2))$$

%

% If we now solve for the derivative of interest, $u_{xx}(i)$,

%

$$\% \quad u_{xx}(i) = -7/(2(dx**2))*u(i) - 3/dx*ux(i)$$

%

$$\% \quad + 8/(dx**2)*u(i+1) - 1/(2*(dx**2))*u(i+2) + O(dx**2)$$

%

$$\% \quad = (1/(2*(dx**2)))*(-u(i+2) + 8*u(i+1) - 7*u(i) - 6*dx*ux(i))$$

%

$$\% \quad + O(dx**2)$$

%

% which is the four-point, second-order approximation for the second derivative, $u_{xx}(i)$, including the first derivative, $u_x(i)$.

%

% Four checks of this approximation can easily be made for $u(i) =$

% 1, $u(i) = x$, $u(i) = x**2$ and $u(i) = x**3$

%

$$\% \quad u_{xx}(i) = (1/(2*(dx**2)))*(-1 + 8*1 - 7*1 - 6*dx*0) = 0$$

%

$$\% \quad u_{xx}(i) = (1/(2*(dx**2)))*(-(x + 2*dx) + 8*(x + dx)$$

%

$$\% \quad - 7*x - 6*dx*1) = 0$$

%

$$\% \quad u_{xx}(i) = (1/(2*(dx**2)))*(-(x + 2*dx)**2 + 8*(x + dx)**2$$

%

$$\% \quad - 7*(x**2) - 6*dx*(2*x))$$

%

$$\% \quad = (-x**2 - 4*x*dx - 4*dx**2$$

%

$$\% \quad + 8*x**2 + 16*x*dx + 8*dx**2$$

%

$$\% \quad - 7*x**2 - 12*x*dx)/(2*(dx**2)) = 2$$

%

$$\% \quad u_{xx}(i) = (1/(2*(dx**2)))*(-(x + 2*dx)**3 + 8*(x + dx)**3$$


```

%
%      - 7*(x**3) - 6*dx*(3*x**2))
%
%      = (1/(2*(dx**2)))*(- x**3 - 6*dx*x**2 - 12*x*dx**2
%
%      - 8*dx**3 + 8*x**3 + 24*dx*x**2 + 24*x*dx**2 + 8*dx**3
%
%      - 7*x**3 - 18*dx*x**2)
%
%      = (1/(2*(dx**2)))*(12*x*dx**2) = 6*x
%
% The preceding approximation for uxx(i) can be applied at the
% left boundary value of x by taking i = 1. An approximation at
% the right boundary is obtained by taking dx = -dx and reversing
% the subscripts in the preceding approximation, with i = n
%
%      uxx(i)
%
%      = (1/(2*(dx**2)))*(-u(i-2) + 8*u(i-1) - 7*u(i) + 6*dx*ux(i))
%
%      + O(dx**2)
%
% To obtain approximations of the second derivative which do not
% involve the first derivative, we take as the linear combination
%
%      a*u(i) + b*u(i+1) + c*u(i+2) + d*u(i+3)
%
% we have
%
%      a*u(i) + b*u(i+1) + c*u(i+2) + d*u(i+3) =
%
%      (a + b + c + d)*u(i) +
%
%      (dx*b + 2*dx*c + 4*dx*d)*ux(i) +
%
%      (b*(dx**2)/2 + c*((2*dx)**2)/2 + d*((3*dx)**2)/2)*uxx(i) +
%
%      (b*(dx**3)/6 + c*((2*dx)**3)/6 + d*((3*dx)**3)/6)*uxx(i) +
%
%      O(dx**4)
%
% The third derivative, uxxx(i), can be dropped by taking
%

```

$$\% \quad b + 8*c + 27*d = 0$$

%

% The second derivative, $u_{xx}(i)$, can be retained by taking

%

$$\% \quad (dx^{**2})*(b/2 + 2*c + (9/2)*d) = 1$$

%

% The first derivative can be dropped by taking

%

$$\% \quad b + 2*c + 3*d = 0$$

%

% Solution of the preceding equations for c and d by elimination of

% b gives

%

$$\% \quad 6*c + 24*d = 0$$

%

$$\% \quad 4*c + 18*d = -2/(dx^{**2})$$

%

% Then, eliminating c, gives

%

$$\% \quad (18 - 16)*d = -2/(dx^{**2})$$

%

% or

%

$$\% \quad d = -1/(dx^{**2})$$

%

$$\% \quad c = (24/6)/(dx^{**2}) = 4/(dx^{**2})$$

%

$$\% \quad b = -8/(dx^{**2}) + 3/(dx^{**2}) = -5/(dx^{**2})$$

%

% $u(i)$ can be dropped by taking

%

$$\% \quad a + b + c + d = 0$$

%

% or

%

$$\% \quad a = (5 - 4 + 1)/(dx^{**2}) = 2/(dx^{**2})$$

%

% If we now solve for the derivative of interest, $u_{xx}(i)$,

%

$$\% \quad u_{xx}(i) = (1/dx^{**2})*(2*u(i) - 5*u(i+1) + 4*u(i+2) - 1*u(i+3))$$

%

$$\% \quad + O(dx^{**2})$$

%

% Which is the four-point, second-order approximation for the second
 % derivative, $u_{xx}(i)$, without the first derivative, $u_x(i)$.

%

% Four checks of this approximation can easily be made for $u(i) =$

% 1, $u(i) = x$, $u(i) = x^{**2}$ and $u(i) = x^{**3}$

%

% $u_{xx}(i) = (1/dx^{**2})*(2 - 5 + 4 - 1) = 0$

%

% $u_{xx}(i) = (1/dx^{**2})*(2*x - 5*(x + dx) + 4*(x + 2*dx)$

%

% $- 1*(x + 3*dx)) = 0$

%

% $u_{xx}(i) = (1/dx^{**2})*(2*x^{**2} - 5*(x + dx)^{**2} + 4*(x + 2*dx)^{**2}$

%

% $- 1*(x + 3*dx)^{**2}) = 2$

%

% $u_{xx}(i) = (1/dx^{**2})*(2*x^{**3} - 5*(x + dx)^{**3} + 4*(x + 2*dx)^{**3}$

%

% $- 1*(x + 3*dx)^{**3})$

%

% $= (1/dx^{**2})*(2*x^{**3} - 5*x^{**3} - 15*x*dx^{**2}$

%

% $- 15*dx*x^{**2} - 5*dx^{**3} + 4*x^{**3} + 24*dx*x^{**2}$

%

% $+ 48*x*dx^{**2} + 32*dx^{**3} - x^{**3} - 9*dx*x^{**2}$

%

% $- 27*x*dx^{**2} - 27dx^{**3})$

%

% $= (1/dx^{**2})*(6*x*dx^{**2}) = 6*x$

%

% The preceding approximation for $u_{xx}(i)$ can be applied at the
 % left boundary value of x by taking $i = 1$. An approximation at
 % the right boundary is obtained by taking $dx = -dx$ and reversing
 % the subscripts in the preceding approximation, with $i = n$

%

% $u_{xx}(i) = (1/dx^{**2})*(2*u(i) - 5*u(i-1) + 4*u(i-2) - 1*u(i-3))$

%

% $+ O(dx^{**2})$

%

% Grid spacing

$dx = (x_u - x_l) / (n - 1);$

%

% u_{xx} at the left boundary, without u_x

```

if nl==1
    uxx(1)=(( 2.)*u( 1)...
        +(-5.)*u( 2)...
        +( 4.)*u( 3)...
        +(-1.)*u( 4))/(dx^2);
%
% uxx at the left boundary, including ux
elseif nl==2
    uxx(1)=((-7.)*u( 1)...
        +( 8.)*u( 2)...
        +(-1.)*u( 3))/(2.*dx^2)...
        +(-6.)*ux( 1)/(2.*dx);
    end
%
% uxx at the right boundary, without ux
if nu==1
    uxx(n)=(( 2.)*u(n )...
        +(-5.)*u(n-1)...
        +( 4.)*u(n-2)...
        +(-1.)*u(n-3))/(dx^2);
%
% uxx at the right boundary, including ux
elseif nu==2
    uxx(n)=((-7.)*u(n )...
        +( 8.)*u(n-1)...
        +(-1.)*u(n-2))/(2.*dx^2)...
        +( 6.)*ux(n)/(2.*dx);
    end
%
% uxx at the interior grid points
for i=2:n-1
    uxx(i)=(u(i+1)-2.*u(i)+u(i-1))/dx^2;
end

```

```

% File: ds044.m

```

```

%

```

```

function [uxx]=dss044(xl,xu,n,u,ux,nl,nu)

```

```

%

```

```

% Function dss044 computes a fourth-order approximation of a
% second-order derivative, with or without the normal derivative

```



```

% at the boundary.
%
% Argument list
%
% xl    Left value of the spatial independent variable (input)
%
% xu    Right value of the spatial independent variable (input)
%
% n     Number of spatial grid points, including the end
%       points (input)
%
% u     One-dimensional array of the dependent variable to be
%       differentiated (input)
%
% ux    One-dimensional array of the first derivative of u.
%       The end values of ux, ux(1) and ux(n), are used in
%       Neumann boundary conditions at  $x = x_l$  and  $x = x_u$ ,
%       depending on the arguments nl and nu (see the de-
%       scription of nl and nu below)
%
% uxx   One-dimensional array of the second derivative of u
%       (output)
%
% nl    Integer index for the type of boundary condition at
%        $x = x_l$  (input). The allowable values are
%
%       1 - Dirichlet boundary condition at  $x = x_l$ 
%           (ux(1) is not used)
%
%       2 - Neumann boundary condition at  $x = x_l$ 
%           (ux(1) is used)
%
% nu    Integer index for the type of boundary condition at
%        $x = x_u$  (input). The allowable values are
%
%       1 - Dirichlet boundary condition at  $x = x_u$ 
%           (ux(n) is not used)
%
%       2 - Neumann boundary condition at  $x = x_u$ 
%           (ux(n) is used)
%
% The following derivation was completed by W. E. Schiesser, Depts
% of CHE and Math, Lehigh University, Bethlehem, PA 18015, USA, on

```

```

% December 15, 1986. Additional details are given in function
% dss042.
%
% *****
%
% (1) uxx at the interior points 3, 4,..., n-2
%
% To develop a set of fourth-order correct differentiation formulas
% for the second derivative uxx, we consider first the interior
% grid points at which a symmetric formula can be used.
%
% If we consider a formula of the form
%
%  $a*u(i-2) + b*u(i-1) + e*u(i) + c*u(i+1) + d*u(i+2)$ 
%
% Taylor series expansions of  $u(i-2)$ ,  $u(i-1)$ ,  $u(i+1)$  and  $u(i+2)$ 
% can be substituted into this formula. We then consider the
% linear algebraic equations relating  $a$ ,  $b$ ,  $c$  and  $d$  which will
% retain certain terms, i.e.,  $u_{xx}$ , and drop others, e.g.,  $u_{xxx}$ ,
%  $u_{xxxx}$  and  $u_{xxxxx}$ .
%
% Thus, for grid points 3, 4,..., n-2
%
% To retain  $u_{xx}$ 
%
%  $4*a + b + c + 4*d = 2$  (1)
%
% To drop  $u_{xxx}$ 
%
%  $-8*a - b + c + 8*d = 0$  (2)
%
% To drop  $u_{xxxx}$ 
%
%  $16*a + b + c + 16*d = 0$  (3)
%
% To drop  $u_{xxxxx}$ 
%
%  $-32*a - b + c + 32*d = 0$  (4)
%
% Equations (1) to (4) can be solved for  $a$ ,  $b$ ,  $c$  and  $d$ . If equation
% (1) is added to equation (2)
%
%  $-4*a + 2*c + 12*d = 2$  (5)

```


%

% If equation (1) is subtracted from equation (3)

%

$$\% \quad 12^*a + 12^*d = -2 \quad (6)$$

%

% If equation (1) is added to equation (4)

%

$$\% \quad -28*a + 2*c + 36*d = 2 \quad (7)$$

%

% Equations (5) to (7) can be solved for a, c and d. If equation

% (5) is subtracted from equation (7), and the result combined

% with equation (6)

%

$$12^*a + 12^*d = -2 \quad (6)$$

%

$$-24^*a + 24^*d = 0 \quad (8)$$

%

% Equations (6) and (8) can be solved for a and d . From (8), a

% = d. From equation (6), $a = -1/12$ and $d = -1/12$. Then, from

% equation (5), $c = 4/3$, and from equation (1), $b = 4/3$.

%

% The final differentiation formula is then obtained as

%

```
% (-1/12)*u(i-2) + (4/3)*u(i-1) +
%
% (4/3)*u(i+1) + (-1/12)*u(i+2)
%
% (-1/12 + 4/3 - 1/12 + 4/3)*u(i) + uxx(i)*(dx**2) + O(dx**6)
```

%

% or

0/

```

% uxx(i) = (1/(12*dx**2))*(-1*u(i-2) + 16*u(i-1)
%
% - 30*u(i) + 16*u(i+1) - 1*u(i+2))      (9)
%
% + O(dx**4)

```

C

Note that the ux term drops out, i.e., the basic equation is

C

$$\% \quad -2*a - b + c + 2*d =$$

C

$$-2^*(-1/12) - (4/3) + (4/3) + 2^*(-1/12) = 0$$

C

% Equation (9) was obtained by dropping all terms in the underlying
 % Taylor series up to and including the fifth derivative, uxxxxx.
 % Thus, equation (9) is exact for polynomials up to and including
 % fifth order. This can be checked by substituting the functions
 % 1, x, x**2, x**3, x**4 and x**5 in equation (9) and computing the
 % corresponding derivatives for comparison with the known second
 % derivatives. This is done for 1 merely by summing the weighting
 % coefficients in equation (9), which should sum to zero, i.e.,
 % -1 + 16 - 30 + 16 -1 = 0.
 %
 % For the remaining functions, the algebra is rather involved, but
 % these functions can be checked numerically, i.e., numerical values
 % of x**2, x**3, x**4 and x**5 can be substituted in equation (9)
 % and the computed derivatives can be compared with the know numeri-
 % cal second derivatives. This is not a proof of correctness of
 % equation (9), but would likely detect any errors in equation (9).
 %
 % *****
 %
 % (2) uxx at the interior points i = 2 and n-1
 %
 % For grid point 2, we consider a formula of the form
 %
 % $a*u(i-1) + f*u(i) + b*u(i+1) + c*u(i+2) + d*u(i+3) + e*u(i+4)$
 %
 % Taylor series expansions of u(i-1), u(i+1), u(i+2), u(i+3) and
 % u(i+4) when substituted into this formula give linear algebraic
 % equations relating a, b, c, d and e.
 %
 % To drop ux
 %
 % $-a + b + 2*c + 3*d + 4*e = 0$ (10)
 %
 % To retain uxx
 %
 % $a + b + 4*c + 9*d + 16*e = 2$ (11)
 %
 % To drop uxxx
 %
 % $-a + b + 8*c + 27*d + 64*e = 0$ (12)
 %
 % To drop uxxxx
 %

$$a + b + 16*c + 81*d + 256*e = 0 \quad (13)$$

%

To drop uxxxxx

%

$$-a + b + 32*c + 243*d + 1024*e = 0 \quad (14)$$

%

Equations (11), (12), (13) and (14) can be solved for a, b, c, d and e. If equation (10) is added to equation (11)

%

$$2*b + 6*c + 12*d + 20*e = 2 \quad (15)$$

%

If equation (10) is subtracted from equation (12)

%

$$6*c + 24*d + 60*e = 0 \quad (16)$$

%

If equation (10) is added to equation (13)

%

$$2*b + 18*c + 84*d + 260*e = 0 \quad (17)$$

%

If equation (10) is subtracted from equation (14)

%

$$30*c + 240*d + 1020*e = 0 \quad (18)$$

%

Equations (15), (16), (17) and (18) can be solved for b, c, d and e.

%

$$6*c + 24*d + 60*e = 0 \quad (16)$$

%

If equation (15) is subtracted from equation (17)

%

$$12*c + 72*d + 240*e = -2 \quad (19)$$

%

$$30*c + 240*d + 1020*e = 0 \quad (18)$$

%

Equations (16), (18) and (19) can be solved for c, d and e. If two times equation (16) is subtracted from equation (19),

%

$$24*d + 120*e = -2 \quad (20)$$

%

If five times equation (16) is subtracted from equation (18),

%

$$120*d + 720*e = 0 \quad (21)$$

%

% Equations (20) and (21) can be solved for d and e. From (21),
 % $e = (-1/6)*d$. Substitution in equation (20) gives $d = -1/2$.
 % thus, $e = 1/12$. From equation (16), $c = 7/6$. From equation
 % (15), $b = -1/3$. From equation (10), $a = 5/6$.

%
 % The final differentiation formula is then obtained as

%
 % $(5/6)*u(i-1) + (-1/3)*u(i+1) + (7/6)*u(i+2) + (-1/2)*u(i+3)$
 %
 % $+ (1/12)*u(i+4) = (5/6 - 1/3 + 7/6 - 1/2 + 1/12)*u(i)$
 %
 % $+ u_{xx}(dx^{**2}) + O(dx^{**6})$

%
 % or

%
 % $u_{xx}(i) = (1/12*dx^{**2})*(10*u(i-1) - 15*u(i) - 4*u(i+1))$
 % (22)
 % $+ 14*u(i+2) - 6*u(i+3) + 1*u(i+4)) + O(dx^{**4})$

%
 % Equation (22) will be applied at $i = 2$ and $n-1$. thus

%
 % $u_{xx}(2) = (1/12*dx^{**2})*(10*u(1) - 15*u(2) - 4*u(3))$
 % (23)
 % $+ 14*u(4) - 6*u(5) + 1*u(6)) + O(dx^{**4})$

%
 % $u_{xx}(n-1) = (1/12*dx^{**2})*(10*u(n) - 15*u(n-1) - 4*u(n-2))$
 % (24)
 % $+ 14*u(n-3) - 6*u(n-4) + 1*u(n-5)) + O(dx^{**4})$

%
 % *****

%
 % (3) u_{xx} at the boundary points 1 and n

%
 % Finally, for grid point 1, an approximation with a Neumann bound-
 % ary condition of the form

%
 % $a*u(i+1) + b*u(i+2) + c*u(i+3) + d*u(i+4) + e*ux(i) + f*u(i)$

%
 % Will be used. the corresponding algebraic equations are

%
 % To drop u_x

%
 % $a + 2*b + 3*c + 4*d + e = 0$ (25)

%

% To retain uxx

%

$$\% \quad a + 4*b + 9*c + 16*d = 2 \quad (26)$$

%

% To drop uxxx

%

$$\% \quad a + 8*b + 27*c + 64*d = 0 \quad (27)$$

%

% To drop uxxxx

%

$$\% \quad a + 16*b + 81*c + 256*d = 0 \quad (28)$$

%

% To drop uxxxxx

%

$$\% \quad a + 32*b + 243*c + 1024*d = 0 \quad (29)$$

%

% Equations (25) to (29) can be solved for a, b, c, d and e. If

%

% Equation (26) is subtracted from equations (27), (28) and (29),

%

$$4*b + 18*c + 48*d = -2 \quad (30)$$

%

$$12*b + 72*c + 240*d = -2 \quad (31)$$

%

$$28*b + 234*c + 1008*d = -2 \quad (32)$$

%

% Equations (30), (31) and (32) can be solved for b, c and d

%

$$18^*c + 96^*d = 4 \quad (33)$$

%

$$\% \quad 108^*c + 672^*d = 12 \quad (34)$$

%

% Equations (3) and (34) can be solved for c and d , $c = 8/9$, $d =$

%

0/

% From equation (30), $b = -3$. From equation (26), $a = 8$. From
 % equation (25), $e = -25/6$.

%

% The final differentiation formula is then obtained as

%

```
% 8*u(i+1) - 3*u(i+2) + (8/9)*u(i+3) - (1/8)*u(i+4)
```

0

% - (25/6)*ux(i)*dx

%

% = (8 - 3 + (8/9) - (1/8))*u(i) + uxx*(dx**2) + O(dx**6)

%

% or

%

% uxx(i) = (1/12*dx**2)*((-415/6)*u(i) + 96*u(i+1) - 36*u(i+2)

%

(35)

% + (32/3)*u(i+3) - (3/2)*u(i+4) - 50*ux(i)*dx) + O(dx**4)

%

% Equation (35) will be applied at i = 1 and i = n

%

% uxx(1) = (1/12*dx**2)*((-415/6)*u(1) + 96*u(2) - 36*u(3)

%

(36)

% + (32/3)*u(4) - (3/2)*u(5) - 50*ux(1)*dx) + O(dx**4)

%

% uxx(n) = (1/12*dx**2)*((-415/6)*u(n) + 96*u(n-1) - 36*u(n-2)

%

(37)

% + (32/3)*u(n-3) - (3/2)*u(n-4) + 50*ux(n)*dx) + O(dx**4)

%

% Alternatively, for grid point 1, an approximation with a Dirichlet
% boundary condition of the form

%

% a*u(i+1) + b*u(i+2) + c*u(i+3) + d*u(i+4) + e*u(i+5) + f*u(i)

%

% can be used. The corresponding algebraic equations are

%

% To drop ux

%

% a + 2*b + 3*c + 4*d + 5*e = 0 (38)

%

% To retain uxx

%

% a + 4*b + 9*c + 16*d + 25*e = 2 (39)

%

% To drop uxxx

%

% a + 8*b + 27*c + 64*d + 125*e = 0 (40)

%

% To drop uxxxx

%

% a + 16*b + 81*c + 256*d + 625*e = 0 (41)

%

% To drop uxxxxx

%

$$\% \quad a + 32*b + 243*c + 1024*d + 3125*e = 0 \quad (42)$$

%

% Equations (38), (39), (40), (41) and (42) can be solved for a,
% b, c, d and e.

%

$$\% \quad 2*b + 6*c + 12*d + 20*e = 2 \quad (43)$$

%

$$\% \quad 6*b + 24*c + 60*d + 120*e = 0 \quad (44)$$

%

$$\% \quad 14*b + 78*c + 252*d + 620*e = 0 \quad (45)$$

%

$$\% \quad 30*b + 240*c + 1020*d + 3120*e = 0 \quad (46)$$

%

% Equations (43), (44), (45) and (46) can be solved for b, c, d
% and e

%

$$\% \quad 6*c + 24*d + 60*e = -6 \quad (47)$$

%

$$\% \quad 36*c + 168*d + 480*e = -14 \quad (48)$$

%

$$\% \quad 150*c + 840*d + 2820*e = -30 \quad (49)$$

%

% Equations (47), (48) and (49) can be solved for c, d and e

%

$$\% \quad 24*d + 120*e = 22 \quad (50)$$

%

$$\% \quad 240*d + 1320*e = 120 \quad (51)$$

%

% From equations (50) and (51), $d = 61/12$, $e = -5/6$. From equation
% (47), $c = -13$. From equation (43), $b = 107/6$. From equation (38),
% $a = -77/6$.

%

% The final differentiation formula is then obtained as

%

$$\% \quad (-77/6)*u(i+1) + (107/6)*u(i+2) - 13*u(i+3) + (61/12)*u(i+4)$$

%

$$\% \quad - (5/6)*u(i+5) = (-77/6 + 107/6 - 13 + 61/12 - 5/6)*u(i) +$$

%

$$\% \quad u_{xx}(i)*(dx^{**2}) + O(dx^{**6})$$

%

% or

```
%
% uxx(i) = (1/12*dx**2)*(45*u(i) - 154*u(i+1) + 214*u(i+2)
%                                     (52)
%      - 156*u(i+3) + 61*u(i+4) - 10*u(i+5)) + O(dx**4)
```

```
% Equation (52) will be applied at i = 1 and i = n
```

```
%
% uxx(1) = (1/12*dx**2)*(45*u(1) - 154*u(2) + 214*u(3)
%                                     (53)
%      - 156*u(4) + 61*u(5) - 10*u(6)) + O(dx**4)
```

```
%
% uxx(n) = (1/12*dx**2)*(45*u(n) - 154*u(n-1) + 214*u(n-2)
%                                     (54)
%      - 156*u(n-3) + 61*u(n-4) - 10*u(n-5)) + O(dx**4)
```

```
% *****
```

```
% Grid spacing
```

```
dx=(xu-xl)/(n-1);
```

```
%
% 1/(12*dx**2) for subsequent use
r12dxs=1./(12.0*dx^2);
```

```
% uxx at the left boundary
```

```
% Without ux (equation (53))
```

```
if nl==1
```

```
uxx(1)=r12dxs*...
```

```
( 45.0*u(1)...
-154.0*u(2)...
+214.0*u(3)...
-156.0*u(4)...
+61.0*u(5)...
-10.0*u(6));
```

```
%
```

```
% With ux (equation (36))
```

```
elseif nl==2
```

```
uxx(1)=r12dxs*...
```

```
(-415.0/6.0*u(1)...
+96.0*u(2)...
-36.0*u(3)...
+32.0/3.0*u(4)...
-3.0/2.0*u(5)...
```



```

-50.0*ux(1)*dx);
end
%
% uxx at the right boundary
%
% Without ux (equation (54))
if nu==1
    uxx(n)=r12dxs*...
        ( 45.0*u(n )...
          -154.0*u(n-1)...
          +214.0*u(n-2)...
          -156.0*u(n-3)...
          +61.0*u(n-4)...
          -10.0*u(n-5));
%
% With ux (equation (37))
elseif nu==2
    uxx(n)=r12dxs*...
        (-415.0/6.0*u(n )...
          +96.0*u(n-1)...
          -36.0*u(n-2)...
          +32.0/3.0*u(n-3)...
          -3.0/2.0*u(n-4)...
          +50.0*ux(n )*dx);
end
%
% uxx at the interior grid points
%
% i = 2 (equation (23))
    uxx(2)=r12dxs*...
        ( 10.0*u(1)...
          -15.0*u(2)...
          -4.0*u(3)...
          +14.0*u(4)...
          -6.0*u(5)...
          +1.0*u(6));
%
% i = n-1 (equation (24))
    uxx(n-1)=r12dxs*...
        ( 10.0*u(n )...
          -15.0*u(n-1)...
          -4.0*u(n-2)...
          +14.0*u(n-3)...

```

```

        -6.0*u(n-4)...
        +1.0*u(n-5));
%
% i = 3, 4,..., n-2 (equation (9))
for i=3:n-2
    uxx(i)=r12dxs*...
        ( -1.0*u(i-2)...
          +16.0*u(i-1)...
          -30.0*u(i) ...
          +16.0*u(i+1)...
          -1.0*u(i+2));
End

```

```
% File: dss046.m
```

```
%
```

```
function [uxx]=dss046(xl,xu,n,u,ux,nl,nu)
```

```
%
```

```
% Function dss046 computes a sixth-order approximation of a
% second-order derivative, with or without the normal derivative
% at the boundary.
```

```
%
```

```
% Argument list
```

```
%
```

```
% xl Left value of the spatial independent variable (input)
```

```
%
```

```
% xu Right value of the spatial independent variable (input)
```

```
%
```

```
% n Number of spatial grid points, including the end
% points (input)
```

```
%
```

```
% u One-dimensional array of the dependent variable to be
% differentiated (input)
```

```
%
```

```
% ux One-dimensional array of the first derivative of u.
```

```
% The end values of ux, ux(1) and ux(n), are used in
```

```
% Neumann boundary conditions at  $x = xl$  and  $x = xu$ ,
% depending on the arguments nl and nu (see the de-
% scription of nl and nu below)
```

```
%
```

```
% uxx One-dimensional array of the second derivative of u
% (output)
```



```

%
% nl Integer index for the type of boundary condition at
% x = xl (input). The allowable values are
%
% 1 - Dirichlet boundary condition at x = xl
% (ux(1) is not used)
%
% 2 - Neumann boundary condition at x = xl
% (ux(1) is used)
%
% nu Integer index for the type of boundary condition at
% x = xu (input). The allowable values are
%
% 1 - Dirichlet boundary condition at x = xu
% (ux(n) is not used)
%
% 2 - Neumann boundary condition at x = xu
% (ux(n) is used)
%
% The following sixth-order, eight-point approximations for a
% second derivative can be used at the boundaries of a spatial
% domain. These approximations have the features
%
% (1) Only interior and boundary points are used (i.e., no
% fictitious points are used)
%
% (2) The normal derivative at the boundary is included as part
% of the approximation for the second derivative
%
% (3) Approximations for the boundary conditions are not used.
%
% Weighting coefficients for finite difference approximations, computed
% by an algorithm of B. Fornberg(1), are used in the following approxi-
% mations. The combination of these coefficients to give the second
% derivative approximations at the boundaries in x, including the first
% derivative, was suggested by Professor Fornberg.
%
% (1) Fornberg, B., Fast Generation of Weights in Finite Difference
% Formulas, In Recent Developments in Numerical Methods and
% Software for ODEs/DAEs/PDEs, G. Byrne and W. E. Schiesser (eds),
% World Scientific, River Edge, NJ, 1992
%
% Grid spacing

```

```

dx=(xu-xl)/(n-1);
rdxs=1.0/dx^2;
%
% uxx at the left boundary
%
% Without ux
if nl==1
uxx(1)=rdxs*...
( 5.211111111111110*u(1)...
-22.300000000000000*u(2)...
+43.950000000000000*u(3)...
-52.722222222222200*u(4)...
+41.000000000000000*u(5)...
-20.100000000000000*u(6)...
+5.661111111111110*u(7)...
-0.700000000000000*u(8));
%
% With ux
elseif nl==2
uxx(1)=rdxs*...
( -7.493888888888860*u(1)...
+12.000000000000000*u(2)...
-7.499999999999940*u(3)...
+4.4444444444444570*u(4)...
-1.874999999999960*u(5)...
+0.479999999999979*u(6)...
-0.055555555555568*u(7)...
-4.900000000000000*ux(1)*dx);
end
%
% uxx at the right boundary
%
% Without ux
if nu==1
uxx(n)=rdxs*...
( 5.211111111111110*u(n )...
-22.300000000000000*u(n-1)...
+43.950000000000000*u(n-2)...
-52.722222222222200*u(n-3)...
+41.000000000000000*u(n-4)...
-20.100000000000000*u(n-5)...
+5.661111111111110*u(n-6)...
-0.700000000000000*u(n-7));

```



```

%
% With ux
elseif nu==2
uxx(n)=rdxs*...
    (-7.493888888888860*u(n )...
    +12.000000000000000*u(n-1)...
    -7.499999999999940*u(n-2)...
    +4.4444444444444570*u(n-3)...
    -1.874999999999960*u(n-4)...
    +0.479999999999979*u(n-5)...
    -0.055555555555568*u(n-6)...
    +4.900000000000000*ux(n)*dx);
end
%
% uxx at the interior grid points
%
% i = 2
uxx(2)=rdxs*...
    ( 0.700000000000000*u(1)...
    -0.388888888888889*u(2)...
    -2.700000000000000*u(3)...
    +4.750000000000000*u(4)...
    -3.722222222222220*u(5)...
    +1.800000000000000*u(6)...
    -0.500000000000000*u(7)...
    +0.061111111111111*u(8));
%
% i = 3
uxx(3)=rdxs*...
    (-0.061111111111111*u(1)...
    +1.188888888888890*u(2)...
    -2.100000000000000*u(3)...
    +0.722222222222223*u(4)...
    +0.472222222222222*u(5)...
    -0.300000000000000*u(6)...
    +0.088888888888889*u(7)...
    -0.011111111111111*u(8));
%
% i = n-1
uxx(n-1)=rdxs*...
    ( 0.700000000000000*u(n)...
    -0.388888888888889*u(n-1)...
    -2.700000000000000*u(n-2)...

```

```

+4.7500000000000000*u(n-3)...
-3.7222222222222220*u(n-4)...
+1.8000000000000000*u(n-5)...
-0.5000000000000000*u(n-6)...
+0.0611111111111111*u(n-7));

```

```
%
```

```

% i = n-2
uxx(n-2)=rdxs*...
( -0.0611111111111111*u(n )...
+1.1888888888888890*u(n-1)...
-2.1000000000000000*u(n-2)...
+0.7222222222222223*u(n-3)...
+0.4722222222222222*u(n-4)...
-0.3000000000000000*u(n-5)...
+0.0888888888888889*u(n-6)...
-0.0111111111111111*u(n-7));

```

```
%
```

```

% i = 4, 5, ..., n-3
for i=4:n-3
uxx(i)=rdxs*...
( 0.0111111111111111*u(i-3)...
-0.1500000000000000*u(i-2)...
+1.5000000000000000*u(i-1)...
-2.7222222222222220*u(i )...
+1.5000000000000000*u(i+1)...
-0.1500000000000000*u(i+2)...
+0.0111111111111111*u(i+3));

```

```
End
```

```
% File: dss048.m
```

```
%
```

```
function [uxx]=dss048(xl,xu,n,u,ux,nl,nu)
```

```
%
```

```
% Function dss048 computes a eighth-order approximation of a
% second-order derivative, with or without the normal derivative
% at the boundary.
```

```
%
```

```
% Argument list
```

```
%
```

```
% xl Left value of the spatial independent variable (input)
```

```
%
```



```

% xu    Right value of the spatial independent variable (input)
%
% n     Number of spatial grid points, including the end
%       points (input)
%
% u     One-dimensional array of the dependent variable to be
%       differentiated (input)
%
% ux    One-dimensional array of the first derivative of u.
%       The end values of ux, ux(1) and ux(n), are used in
%       Neumann boundary conditions at  $x = x_l$  and  $x = x_u$ ,
%       depending on the arguments nl and nu (see the de-
%       scription of nl and nu below)
%
% uxx   One-dimensional array of the second derivative of u
%       (output)
%
% nl    Integer index for the type of boundary condition at
%        $x = x_l$  (input). the allowable values are
%
%       1 - Dirichlet boundary condition at  $x = x_l$ 
%           (ux(1) is not used)
%
%       2 - Neumann boundary condition at  $x = x_l$ 
%           (ux(1) is used)
%
% nu    Integer index for the type of boundary condition at
%        $x = x_u$  (input). The allowable values are
%
%       1 - Dirichlet boundary condition at  $x = x_u$ 
%           (ux(n) is not used)
%
%       2 - Neumann boundary condition at  $x = x_u$ 
%           (ux(n) is used)
%
% The following eighth-order, ten-point approximations for a
% second derivative can be used at the boundaries of a spatial
% domain. These approximations have the features
%
% (1) Only interior and boundary points are used (i.e., no
%     fictitious points are used)
%
% (2) The normal derivative at the boundary is included as part

```

% of the approximation for the second derivative

%

% (3) Approximations for the boundary conditions are not used.

%

% Weighting coefficients for finite difference approximations, computed

% by an algorithm of B. Fornberg (1,2), are used in the following

% approximations. The combination of these coefficients to give the

% second derivative approximations at the boundaries in x, including

% the first derivative, was suggested by Professor Fornberg.

%

% (1) Fornberg, B., Fast Generation of Weights in Finite Difference

% Formulas, In Recent Developments In Numerical Methods And

% Software for ODEs/DAEs/PDEs, G. Byrne and W. E. Schiesser (eds),

% World Scientific, River Edge, NJ, 1992

%

% (2) Fornberg, B., Calculation of Weights In Finite Difference

% Formulas, SIAM Review, Vol. 40, No. 3, Pp 685-691, September,

% 1999

%

% Grid spacing

dx=(xu-xl)/(n-1);

rdxs=1.0/dx^2;

%

% uxx at the left boundary

%

% Without ux

if nl==1

uxx(1)=rdxs*...

(6.463293650793652*u(1)...

-32.921428571428578*u(2)...

+83.842857142857156*u(3)...

-139.75555555555555*u(4)...

+162.47499999999999*u(5)...

-132.50000000000000*u(6)...

+74.54444444444444*u(7)...

-27.628571428571430*u(8)...

+6.083928571428572*u(9)...

-0.603968253968254*u(10));

%

% With ux

elseif nl==2

uxx(1)=rdxs*...

(-8.914169501133784*u(1)...


```

+16.0000000000000000*u(2)...
-14.0000000000000000*u(3)...
+12.4444444444444444*u(4)...
-8.7500000000000000*u(5)...
+4.4800000000000000*u(6)...
-1.5555555555555555*u(7)...
+0.326530612244898*u(8)...
-0.0312500000000000*u(9)...
-5.435714285714286*ux(1)*dx);
end

%
% uxx at the right boundary
%
% Without ux
if nu==1
uxx(n)=rdxs*...
( -0.603968253968254*u(n-9)...
+6.083928571428572*u(n-8)...
-27.628571428571430*u(n-7)...
+74.544444444444444*u(n-6)...
-132.500000000000000*u(n-5)...
+162.474999999999999*u(n-4)...
-139.755555555555555*u(n-3)...
+83.842857142857156*u(n-2)...
-32.921428571428578*u(n-1)...
+6.463293650793652*u(n ));
%
% With ux
elseif nu==2
uxx(n)=rdxs*...
(-0.0312500000000000*u(n-8)...
+0.326530612244905*u(n-7)...
-1.555555555555555*u(n-6)...
+4.4800000000000000*u(n-5)...
-8.7500000000000000*u(n-4)...
+12.444444444444444*u(n-3)...
-14.0000000000000000*u(n-2)...
+16.0000000000000000*u(n-1)...
-8.914169501133790*u(n )...
+5.435714285714287*ux(n )*dx);
end

%
% uxx at the interior grid points

```

%

i = 2

uxx(2)=rdxs*...

(0.603968253968254*u(1)...
 +0.423611111111111*u(2)...
 -5.742857142857143*u(3)...
 +11.366666666666665*u(4)...
 -12.922222222222221*u(5)...
 +10.274999999999999*u(6)...
 -5.666666666666665*u(7)...
 +2.068253968253968*u(8)...
 -0.450000000000000*u(9)...
 +0.044246031746032*u(10));

%

i = 3

uxx(3)=rdxs*...

(-0.044246031746032*u(1)...
 +1.046428571428571*u(2)...
 -1.567460317460318*u(3)...
 -0.433333333333333*u(4)...
 +2.075000000000000*u(5)...
 -1.772222222222222*u(6)...
 +0.983333333333333*u(7)...
 -0.357142857142857*u(8)...
 +0.077182539682540*u(9)...
 -0.007539682539688*u(10));

%

i = 4

uxx(4)=rdxs*...

(0.007539682539683*u(1)...
 -0.119642857142857*u(2)...
 +1.385714285714286*u(3)...
 -2.472222222222222*u(4)...
 +1.150000000000000*u(5)...
 +0.175000000000000*u(6)...
 -0.188888888888889*u(7)...
 +0.078571428571429*u(8)...
 -0.017857142857143*u(9)...
 +0.001785714285714*u(10));

%

i = n-3

uxx(n-3)=rdxs*...

(0.001785714285714*u(n-9)...


```

-0.017857142857143*u(n-8)...
+0.078571428571429*u(n-7)...
-0.188888888888889*u(n-6)...
+0.175000000000000*u(n-5)...
+1.150000000000000*u(n-4)...
-2.472222222222224*u(n-3)...
+1.385714285714286*u(n-2)...
-0.119642857142857*u(n-1)...
+0.007539682539683*u(n ));

```

```
%
```

```
% i = n-2
```

```
uxx(n-2)=rdxs*...
```

```

(-0.007539682539683*u(n-9)...
+0.077182539682540*u(n-8)...
-0.357142857142857*u(n-7)...
+0.983333333333333*u(n-6)...
-1.772222222222222*u(n-5)...
+2.075000000000001*u(n-4)...
-0.433333333333335*u(n-3)...
-1.567460317460318*u(n-2)...
+1.046428571428572*u(n-1)...
-0.044246031746032*u(n ));

```

```
%
```

```
% i = n-1
```

```
uxx(n-1)=rdxs*...
```

```

(0.044246031746032*u(n-9)...
-0.450000000000000*u(n-8)...
+2.068253968253968*u(n-7)...
-5.666666666666667*u(n-6)...
+10.275000000000000*u(n-5)...
-12.922222222222222*u(n-4)...
+11.366666666666667*u(n-3)...
-5.742857142857142*u(n-2)...
+0.423611111111111*u(n-1)...
+0.603968253968254*u(n ));

```

```
%
```

```
% i = 5, 6,..., n-4
```

```
for i=5:n-4
```

```
uxx(i)=rdxs*...
```

```

(-0.001785714285714*u(i-4)...
+0.025396825396825*u(i-3)...
-0.200000000000000*u(i-2)...
+1.600000000000000*u(i-1)...

```

```

-2.84722222222221*u(i )...
+1.600000000000000*u(i+1)...
-0.200000000000000*u(i+2)...
+0.025396825396825*u(i+3)...
-0.001785714285714*u(i+4));

```

End

```

% File: dss050.m

```

```

%
```

```

function [uxx]=dss050(xl,xu,n,u,ux,nl,nu)

```

```

%
```

```

% Function dss050 computes a tenth-order approximation of a
% second-order derivative, with or without the normal derivative
% at the boundary.

```

```

%
```

```

% Argument list

```

```

%
```

```

% xl    Left value of the spatial independent variable (input)

```

```

%
```

```

% xu    Right value of the spatial independent variable (input)

```

```

%
```

```

% n     Number of spatial grid points, including the end
%       points (input)

```

```

%
```

```

% u     One-dimensional array of the dependent variable to be
%       differentiated (input)

```

```

%
```

```

% ux    One-dimensional array of the first derivative of u.
%       The end values of ux, ux(1) and ux(n), are used in
%       Neumann boundary conditions at  $x = xl$  and  $x = xu$ ,
%       depending on the arguments nl and nu (see the de-
%       scription of nl and nu below)

```

```

%
```

```

% uxx   One-dimensional array of the second derivative of u
%       (output)

```

```

%
```

```

% nl    Integer index for the type of boundary condition at
%        $x = xl$  (input). The allowable values are

```

```

%
```

```

%       1 - Dirichlet boundary condition at  $x = xl$ 
%       (ux(1) is not used)

```

```

%
```



```

%
%      2 - Neumann boundary condition at  $x = x_l$ 
%      (ux(1) is used)
%
%      nu      Integer index for the type of boundary condition at
%       $x = x_u$  (input). The allowable values are
%
%      1 - Dirichlet boundary condition at  $x = x_u$ 
%      (ux(n) is not used)
%
%      2 - Neumann boundary condition at  $x = x_u$ 
%      (ux(n) is used)
%
% The following tenth-order, twelve-point approximations for a
% second derivative can be used at the boundaries of a spatial
% domain. These approximations have the features
%
% (1) Only interior and boundary points are used (i.e., no
%      fictitious points are used)
%
% (2) The normal derivative at the boundary is included as part
%      of the approximation for the second derivative
%
% (3) Approximations for the boundary conditions are not used.
%
% Weighting coefficients for finite difference approximations, computed
% by an algorithm of B. Fornberg (1,2), are used in the following
% approximations. The combination of these coefficients to give the
% second derivative approximations at the boundaries in  $x$ , including
% the first derivative, was suggested by Professor Fornberg.
%
% (1) Fornberg, B., Fast Generation of Weights in Finite Difference
%      Formulas, In Recent Developments In Numerical Methods And
%      Software For ODEs/DAEs/PDEs, G. Byrne and W. E. Schiesser (eds),
%      World Scientific, River Edge, NJ, 1992
%
% (2) Fornberg, B., Calculation of Weights in Finite Difference
%      Formulas, SIAM Review, Vol. 40, No. 3, Pp 685-691, September,
%      1999
%
% Grid spacing
%  $dx=(x_u-x_l)/(n-1);$ 
%

```

```

% 1/dx**2 for subsequent use
rdxs=1.0/dx^2;
%
% uxx at the left boundary
%
% Without ux
if nl==1
uxx(1)=rdxs*...
( 7.561626984126982*u(1)...
-44.437301587301590*u(2)...
+138.593253968253980*u(3)...
-295.519841269841320*u(4)...
+457.029761904761930*u(5)...
-521.113333333333460*u(6)...
+439.394444444444330*u(7)...
-271.261904761904700*u(8)...
+119.413690476190470*u(9)...
-35.551587301587304*u(10)...
+6.423730158730158*u(11)...
-0.532539682539683*u(12));
%
% With ux
elseif nl==2
uxx(1)=rdxs*...
(-10.128622763920379*u(1)...
+20.000000000000000*u(2)...
-22.499999999999972*u(3)...
+26.666666666666657*u(4)...
-26.249999999999943*u(5)...
+20.159999999999968*u(6)...
-11.666666666666686*u(7)...
+4.897959183673493*u(8)...
-1.406250000000000*u(9)...
+0.246913580246911*u(10)...
-0.0200000000000001*u(11)...
-5.857936507936508*ux(1)*dx);
end
%
% uxx at the right boundary
%
% Without ux
if nu==1
uxx(n)=rdxs*...

```



```
( -0.532539682539683*u(n-11)...
+6.423730158730161*u(n-10)...
-35.551587301587304*u(n-9)...
+119.413690476190480*u(n-8)...
-271.261904761904760*u(n-7)...
+439.394444444444390*u(n-6)...
-521.113333333333340*u(n-5)...
+457.029761904761930*u(n-4)...
-295.519841269841270*u(n-3)...
+138.593253968253980*u(n-2)...
-44.437301587301590*u(n-1)...
+7.561626984126984*u(n ));
```

```
%
```

```
%
```

```
With ux
```

```
elseif nu==2
```

```
uxx(n)=rdxs*...
```

```
( -0.019999999999999999*u(n-10)...
+0.246913580246904*u(n-9)...
-1.40624999999999986*u(n-8)...
+4.897959183673493*u(n-7)...
-11.6666666666666686*u(n-6)...
+20.15999999999999968*u(n-5)...
-26.24999999999999943*u(n-4)...
+26.6666666666666629*u(n-3)...
-22.49999999999999972*u(n-2)...
+20.00000000000000000*u(n-1)...
-10.128622763920383*u(n)...
+5.857936507936508*ux(n )*dy;
```

```
end
```

```
%
```

```
% uxx at the interior grid points
```

```
%
```

```
%
```

```
%
```

```
i = 2
```

```
uxx(2)=rdxs*...
```

```
( 0.532539682539683*u(1)...
+1.171150793650794*u(2)...
-9.289682539682540*u(3)...
+21.434523809523807*u(4)...
-31.912698412698408*u(5)...
+35.25833333333333333*u(6)...
-29.046666666666663*u(7)...
+17.623015873015873*u(8)...
-7.654761904761902*u(9)...
```

+2.254960317460317*u(10)...
 -0.403968253968254*u(11)...
 +0.033253968253968*u(12));

%

% i = 3

uxx(3)=rdxs*...

(-0.033253968253968*u(1)...
 +0.931587301587302*u(2)...
 -1.023611111111112*u(3)...
 -1.973809523809523*u(4)...
 +4.973809523809524*u(5)...
 -5.575555555555556*u(6)...
 +4.531666666666666*u(7)...
 -2.709523809523809*u(8)...
 +1.162301587301588*u(9)...
 -0.338888888888889*u(10)...
 +0.060198412698413*u(11)...
 -0.004920634920635*u(12));

%

% i = 4

uxx(4)=rdxs*...

(0.004920634920635*u(1)...
 -0.092301587301587*u(2)...
 +1.256349206349206*u(3)...
 -2.106150793650793*u(4)...
 +0.461904761904762*u(5)...
 +1.076666666666667*u(6)...
 -1.028888888888889*u(7)...
 +0.634523809523810*u(8)...
 -0.273809523809524*u(9)...
 +0.079761904761905*u(10)...
 -0.014126984126984*u(11)...
 +0.001150793650794*u(12));

%

% i = 5

uxx(5)=rdxs*...

(-0.001150793650794*u(1)...
 +0.018730158730159*u(2)...
 -0.168253968253968*u(3)...
 +1.509523809523809*u(4)...
 -2.675793650793650*u(5)...
 +1.373333333333333*u(6)...
 +0.013333333333334*u(7)...

-0.117460317460318*u(8)...
 +0.064880952380952*u(9)...
 -0.020634920634921*u(10)...
 +0.003809523809524*u(11)...
 -0.000317460317460*u(12));

%

% i = n-4

uxx(n-4)=rdxs*...
 (-0.000317460317460*u(n-11)...
 +0.003809523809524*u(n-10)...
 -0.020634920634921*u(n-9)...
 +0.064880952380952*u(n-8)...
 -0.117460317460317*u(n-7)...
 +0.013333333333333*u(n-6)...
 +1.373333333333333*u(n-5)...
 -2.675793650793652*u(n-4)...
 +1.509523809523810*u(n-3)...
 -0.168253968253968*u(n-2)...
 +0.018730158730159*u(n-1)...
 -0.001150793650794*u(n));

%

% i = n-3

uxx(n-3)=rdxs*...
 (0.001150793650794*u(n-11)...
 -0.014126984126984*u(n-10)...
 +0.079761904761905*u(n-9)...
 -0.273809523809524*u(n-8)...
 +0.634523809523809*u(n-7)...
 -1.028888888888889*u(n-6)...
 +1.076666666666667*u(n-5)...
 +0.461904761904762*u(n-4)...
 -2.106150793650793*u(n-3)...
 +1.256349206349206*u(n-2)...
 -0.092301587301587*u(n-1)...
 +0.004920634920635*u(n));

%

% i = n-2

uxx(n-2)=rdxs*...
 (-0.004920634920635*u(n-11)...
 +0.060198412698413*u(n-10)...
 -0.338888888888889*u(n-9)...
 +1.162301587301588*u(n-8)...
 -2.709523809523809*u(n-7)...

```

+4.5316666666666666*u(n-6)...
-5.5755555555555556*u(n-5)...
+4.973809523809526*u(n-4)...
-1.973809523809526*u(n-3)...
-1.0236111111111110*u(n-2)...
+0.931587301587302*u(n-1)...
-0.033253968253968*u(n ));

```

```
%
```

```
% i = n-1
```

```
uxx(n-1)=rdxs*...
```

```

( 0.033253968253968*u(n-11)...
-0.403968253968254*u(n-10)...
+2.254960317460318*u(n-9)...
-7.654761904761904*u(n-8)...
+17.623015873015873*u(n-7)...
-29.046666666666670*u(n-6)...
+35.258333333333340*u(n-5)...
-31.912698412698411*u(n-4)...
+21.434523809523810*u(n-3)...
-9.289682539682541*u(n-2)...
+1.171150793650794*u(n-1)...
+0.532539682539683*u(n ));

```

```
%
```

```
% i = 6, 7,..., n-5
```

```
for i=6:n-5
```

```
uxx(i)=rdxs*...
```

```

( 0.000317460317460*u(i-5)...
-0.004960317460317*u(i-4)...
+0.039682539682540*u(i-3)...
-0.238095238095238*u(i-2)...
+1.666666666666667*u(i-1)...
-2.927222222222222*u(i)...
+1.666666666666667*u(i+1)...
-0.238095238095238*u(i+2)...
+0.039682539682540*u(i+3)...
-0.004960317460317*u(i+4)...
+0.000317460317460*u(i+5));

```

```
end
```


Appendix B:

Below are the list of the subordinate routine called by main program in *Listing 3.1-3.9* (Schiesser & Griffith, 2009). They are `initial_1`, `ua`, `simp` and `pde_1`. See `initial_1` first as *Listing 3.10*;

```
% Function initial_1 sets the initial condition for the CSE
%
global ncall q x xl xu rt2 n
%
% q in cubic Schrodinger equation (CSE)
q=1.0;
%
% Precompute  $2^{0.5}$ 
rt2=sqrt(2.0);
%
% Grid spacing
dx=(xu-xl)/(n-1);
%
% IC over the spatial grid
for i=1:n
%
% Uniform grid
x(i)=xl+(i-1)*dx;
%
% Initial real, imaginary parts (v(x,0), w(x,0)), absolute value
% of  $u(x,0) = u_0(x) = (2^{0.5})\exp(0.5*i*x)\operatorname{sech}(x)$ ,  $i = (-1)^{0.5}$ 
% over  $xl \leq x \leq xu$ 
%
%  $\operatorname{sech}(x)$ 
sch=2.0/(exp(x(i))+exp(-x(i)));
%
% Real part
v(i)=rt2*cos(0.5*x(i))*sch;
%
% Imaginary part
w(i)=rt2*sin(0.5*x(i))*sch;
end
%
% Two vectors to one vector
for i=1:n
u0(i) =v(i);
u0(i+n)=w(i);
end
```

We can note the following details about `initial_1`:

1. After the routine is defined and some parameters are declared as global, a spatial grid is defined for $-10 \leq x \leq 40$ (recall that number of grid points n , and left and right boundary values of x , x_l and x_u are set in `pde_1_main` of Listing 3.1-3.9 as global parameters)

```
% Function initial_1 sets the initial condition for the CSE
%
global ncall q x xl xu rt2 n
%
% q in cubic Schrodinger equation (CSE)
q=1.0;
%
% Precompute  $2^{0.5}$ 
rt2=sqrt(2.0);
%
% Grid spacing
dx=(xu-xl)/(n-1);
%
% IC over the spatial grid
for i=1:n
%
% Uniform grid
x(i)=xl+(i-1)*dx;
```

2. Finally real and imaginary parts of the IC of Equation (13.5.8) are computed

```
% Initial real, imaginary parts ( $v(x,0)$ ,  $w(x,0)$ ), absolute value
% of  $u(x,0) = u_0(x) = (2^{0.5})\exp(0.5*i*x)\operatorname{sech}(x)$ ,  $i = (-1)^{0.5}$ 
% over  $x_l \leq x \leq x_u$ 
%
%  $\operatorname{sech}(x)$ 
sch=2.0/(exp(x(i))+exp(-x(i)));
%
% Real part
v(i)=rt2*cos(0.5*x(i))*sch;
%
% Imaginary part
w(i)=rt2*sin(0.5*x(i))*sch;
end
%
% Two vectors to one vector
for i=1:n
u0(i) =v(i);
```



```

u0(i+n)=w(i);
end

```

The analytical solution of equation (13.5.7a) is programmed in ua (see *Listing 3.11*)

```

function uanal=ua(t,x)
%
% Function ua computes the analytical solution to the CSE
%
% The following exact solution is  $|u(x,t)|^2$  which is compared
% with the numerical solution
%
uanal=2.0*(2.0/(exp(x-t)+exp(-(x-t))))^2;

```

routine for calculating the integrals of Equations (13.5.9a)-(13.5.9c) by Simpson's rule is given in *Listing 3.12*:

```

function uint=simp(xl,xu,n,u)
%
% Function simp computes three integral invariants by Simpson's
% rule
%
global q
%
% One vector to two vectors
for i=1:n
    v(i)=u(i);
    w(i)=u(i+n);
end
%
% Three invariants
for int=1:3
    h=(xu-xl)/(n-1);
%
% I1
    if(int==1)
        uabs(1)=u(1)^2+v(1)^2;
        uabs(n)=u(n)^2+v(n)^2;
        uint(1)=uabs(1)-uabs(n);
        for i=3:2:n
            uabs(i-1)=u(i-1)^2+v(i-1)^2;
            uabs(i)=u(i)^2+v(i)^2;
            uint(1)=uint(1)+4.0*uabs(i-1)+2.0*uabs(i);
        end
        uint(1)=h/3.0*uint(1);

```

```

end
%
% I2
if(int==2)
    vx=dss004(xl,xu,n,v);
    wx=dss004(xl,xu,n,w);
    int(1)=v(1)*wx(1)-w(1)*vx(1);
    int(n)=v(n)*wx(n)-w(n)*vx(n);
    uint(2)=int(1)-int(n);
    for i=3:2:n
        int(i-1)=v(i-1)*wx(i-1)-w(i-1)*vx(i-1);
        int(i)=v(i)*wx(i)-w(i)*vx(i);
        uint(2)=uint(2)+4.0*int(i-1)+2.0*int(i);
    end
    uint(2)=h/3.0*uint(2);
end
%
% I3
if(int==3)
    uxabs(1)=vx(1)^2+wx(1)^2;
    uxabs(n)=vx(n)^2+wx(n)^2;
    uint(3)=uxabs(1)-0.5*q*uxabs(1)^2-(uxabs(n)-0.5*q*uxabs(n)^2);
    for i=3:2:n
        uxabs(i-1)=vx(i-1)^2+wx(i-1)^2;
        uxabs(i)=vx(i)^2+wx(i)^2;
        uint(3)=uint(3)+4.0*(uxabs(i-1)-0.5*q*uxabs(i-1)^2)...
            +2.0*(uxabs(i)-0.5*q*uxabs(i)^2);
    end
    uint(3)=2.0*h/3.0*uint(3);
end
end

```

simp has three parts corresponding to the integrals I1, I2, I3 of equations (13.5.9a)-(13.5.9c)

1. The coding for I1 is

```

function uint=simp(xl,xu,n,u)
%
% Function simp computes three integral invariants by Simpson's
% rule
%
global q
%
% One vector to two vectors
for i=1:n

```



```

v(i)=u(i);
w(i)=u(i+n);
end
%
% Three invariants
for int=1:3
    h=(xu-xl)/(n-1);
%
% I1
if(int==1)
    uabs(1)=u(1)^2+v(1)^2;
    uabs(n)=u(n)^2+v(n)^2;
    uint(1)=uabs(1)-uabs(n);
    for i=3:2:n
        uabs(i-1)=u(i-1)^2+v(i-1)^2;
        uabs(i)=u(i)^2+v(i)^2;
        uint(1)=uint(1)+4.0*uabs(i-1)+2.0*uabs(i);
    end
    uint(1)=h/3.0*uint(1);
end

```

After defining the function, vector u is divided into two vectors, v and w , according to $u(x,t) = v(x,t) + i\omega(x,t)$. the integration interval $h=(x_l-x_u)/(n-1)$ is then computed, where $x_l = -10$ and $x_u = 40$ are lower and upper limit of the integral. The for loop for I1 is an implementation of the of the weighted sum for the Simpson's rule applied to the function $|u(x,t)|^2$ (the integration by equation (13.5.9a))

$$\int_{-\infty}^{\infty} u(x,t) dx \approx \frac{h}{3} \left[|u_1|^2 + \sum_{i=2}^{n-2} (4|u_i|^2 + 2|u_{i+1}|^2) + |u_n|^2 \right]$$

2. Similarly, the coding for I2 of Equation (13.5.9b) is

```

% I2
if(int==2)
    vx=dss004(xl,xu,n,v);
    wx=dss004(xl,xu,n,w);
    int(1)=v(1)*wx(1)-w(1)*vx(1);
    int(n)=v(n)*wx(n)-w(n)*vx(n);
    uint(2)=int(1)-int(n);
    for i=3:2:n
        int(i-1)=v(i-1)*wx(i-1)-w(i-1)*vx(i-1);
        int(i)=v(i)*wx(i)-w(i)*vx(i);
        uint(2)=uint(2)+4.0*int(i-1)+2.0*int(i);
    end
end

```

```

end
uint(2)=h/3.0*uint(2);
end

```

The only difference is the use of integrand $v(x,t)\omega_x(x,t) - \omega(x,t)v_x(x,t)$ according to equation (13.5.9b). the derivatives $v_x(x,t)$ and $\omega_x(x,t)$ are computed by call to the differential routine dss004.

3. Finally the coding for I3 of equation (13.5.9c) is

```

% I3
if(int==3)
uxabs(1)=vx(1)^2+wx(1)^2;
uxabs(n)=vx(n)^2+wx(n)^2;
uint(3)=uxabs(1)-0.5*q*uabs(1)^2-(uxabs(n)-0.5*q*uabs(n)^2);
for i=3:2:n
uxabs(i-1)=vx(i-1)^2+wx(i-1)^2;
uxabs(i) =vx(i )^2+wx(i )^2;
uint(3)=uint(3)+4.0*(uxabs(i-1)-0.5*q*uabs(i-1)^2)...
+2.0*(uxabs(i )-0.5*q*uabs(i )^2);
end
uint(3)=2.0*h/3.0*uint(3);
end
end

```

The integrand is $|u_x(x,t)|^2 - 0.5q|u(x,t)|^4$ according to equation (13.5.9c).

The MOL ODE routine, pde_1, called by ode45 or ode15 in pde_1_main of *Listing 3.1-3.9* is given in *Listing 3.13*:

```

function ut=pde_1(t,u)
%
% Function yt computes the t derivative vector of the CSE
%
% global ncall q xl xu n
%
% One vector to two vectors
for i=1:n
v(i)=u(i);
w(i)=u(i+n);
end
%
% PDEs

```



```

%
% v
% xx
%
vx=zeros(n,1);
v(1)=0.0;
v(n)=0.0;
nl=1;
nu=1;
vxx=dss044(xl,xu,n,v,vx,nl,nu);
%
% w
% xx
wx=zeros(n,1);
w(1)=0.0;
w(n)=0.0;
nl=1;
nu=1;
wxx=dss044(xl,xu,n,w,wx,nl,nu);
%
% ODEs at the boundaries
vt(1)=0.0;
wt(1)=0.0;
vt(n)=0.0;
wt(n)=0.0;
%
% ODEs at the interior points
for i=2:n-1
%
%  $v^{*2} + w^{*2}$ 
v2w2(i)=v(i)^2+w(i)^2;
%
% vt
vt(i)=-wxx(i)-q*v2w2(i)*w(i);
%
% wt
wt(i)=vxx(i)+q*v2w2(i)*v(i);
end
%
% Two vectors to one vector
for i=1:n
ut(i) =vt(i);
ut(i+n)=wt(i);
end
ut=ut';
%

```

wxx=dss044(xl,xu,n,w,wx,nl,nu

Note this few details about the calculation of second derivatives

a. The first derivative arrays, vx and ux, which are input arguments to dss044, are not actually used in the calculation of uxx and wxx. However, Matlab requires that at least one element of array that is an input to the function has numerical value. This is accomplished by setting all the elements of v and w to zero through the Matlab zero utility.

b. also BCs of equation (13.5.1) are not required since $-10 \leq x \leq 40$ is equivalent to $-\infty \leq x \leq \infty$. However, the calculation of v_{xx} and w_{xx} by dss044 requires BCs (since this is a general purpose routine that is typically applied to the finite spatial domain that requires BCs), so the Dirichlet BCs $v_{xx}(x=-10,t) = v_{xx}(x=40,t) = 0$ and $w_{xx}(x=-10,t) = w_{xx}(x=40,t) = 0$ are used (and programmed as $v(1)=0.0$, $v(n)=0.0$, $w(1)=0.0$, $w(n)=0.0$). Also these Dirichlet BCs must be identified before calling dss044 by nl=1, nu=1 for the lower ($x=-10$) boundary values of x.

4. Now that the spatial derivatives in equation (13.5.1) have been computed, the MOL programming of equation (13.5.1) can be added.

```
%
% ODEs at the boundaries
vt(1)=0.0;
wt(1)=0.0;
vt(n)=0.0;
wt(n)=0.0;
%
% ODEs at the interior points
for i=2:n-1
%
% v**2 + w**2
v2w2(i)=v(i)^2+w(i)^2;
%
% vt
vt(i)=-wxx(i)-q*v2w2(i)*w(i);
%
% wt
```



```

    wt(i)= vxx(i)+q*v2w2(i)*v(i);
end
%
% Two vectors to one vector
for i=1:n
    ut(i) =vt(i);
    ut(i+n)=wt(i);
end
ut=ut';
%
% Increment calls to pde_1
ncall=ncall+1;

```

Note that the derivatives in t at the boundaries are set to zero (to reflect the Dirichlet BCs). Then the real and imaginary parts of u are programmed for the interior points $2 \leq i \leq n-1$ according to equation (13.5.1). In particular the programming of nonlinear term is accomplished. Then two derivative vectors, vt and w , are returned from `pde_1` as a singular vector, ut , for integration by `ode45` or `ode15s`. The usual transpose of ut is included as well as the updating of the counter for the calls to `pde_1`