APPENDIX 1

Partial Differential Equations from Conservation Principles: The Anisotropic Diffusion Equation

In some of the preceding chapters, we discussed the origin of partial differential equations (PDEs) by simplifying general PDE systems. For example, we arrived at the damped wave equation for an electric field as a special case of the Maxwell equations for electromagnetic (EM) fields. Also, we obtained Burgers' equation as a special case of the Euler and Navier Stokes equations of fluid mechanics.

While this approach of starting with a general PDE system is always an important first step to consider in developing a PDE model for a physical application, it also has limitations in the sense that the general system may not encompass all of the physical phenomena we wish to include in a mathematical model. For example, the Navier Stokes equations, as we used them, did not include an energy balance so that thermal effects, for example, a temperature field, could not be included in an analysis of a nonisothermal system. Of course, an energy balance could be (and has been) included with the Navier Stokes equations, but this requires some additional analysis. Also, we might be interested in a physical situation that is not reflected in a general PDE system, and we therefore have to derive the relevant PDEs starting from first principles, usually conservation principles in the case of physical applications.

In this appendix we consider the derivation of the *equations for anisotropic diffusion*, that is, a PDE system for which the diffusivity is a *nine-component tensor*. The final result is a generalization of the usual diffusion equation that includes directional effects that are significant in certain physical systems. The primary intent is to illustrate a method for the derivation of PDEs to include physical effects and phenomena that might not appear in previously available general PDE systems. The analysis leads to PDEs in Cartesian, cylindrical, and spherical coordinates, but the use of other orthogonal coordinate systems follows in the same way as illustrated here. Examples of PDEs in cylindrical and spherical coordinates are given in Chapters 13 and 14.

We start with a mass balance on an incremental cube with sides of length Δx, Δy, Δz.

$$\begin{aligned}
\Delta x\Delta y\Delta z\frac{\partial c}{\partial t} = &-\Delta y\Delta z D_{xx}\frac{\partial c}{\partial x}|_x - \left(-\Delta y\Delta z D_{xx}\frac{\partial c}{\partial x}|_{x+\Delta x}\right)\\
&-\Delta y\Delta z D_{xy}\frac{\partial c}{\partial y}|_x - \left(-\Delta y\Delta z D_{xy}\frac{\partial c}{\partial y}|_{x+\Delta x}\right)\\
&-\Delta y\Delta z D_{xz}\frac{\partial c}{\partial z}|_x - \left(-\Delta y\Delta z D_{xz}\frac{\partial c}{\partial z}|_{x+\Delta x}\right)\\
&-\Delta x\Delta z D_{yx}\frac{\partial c}{\partial x}|_y - \left(-\Delta x\Delta z D_{yx}\frac{\partial c}{\partial x}|_{y+\Delta y}\right)\\
&-\Delta x\Delta z D_{yy}\frac{\partial c}{\partial y}|_y - \left(-\Delta x\Delta z D_{yy}\frac{\partial c}{\partial y}|_{y+\Delta y}\right)\\
&-\Delta x\Delta z D_{yz}\frac{\partial c}{\partial z}|_y - \left(-\Delta x\Delta z D_{yz}\frac{\partial c}{\partial z}|_{y+\Delta y}\right)\\
&-\Delta x\Delta y D_{zx}\frac{\partial c}{\partial x}|_z - \left(-\Delta x\Delta y D_{zx}\frac{\partial c}{\partial x}|_{z+\Delta z}\right)\\
&-\Delta x\Delta y D_{zy}\frac{\partial c}{\partial y}|_z - \left(-\Delta x\Delta y D_{zy}\frac{\partial c}{\partial y}|_{z+\Delta z}\right)\\
&-\Delta x\Delta y D_{zz}\frac{\partial c}{\partial z}|_z - \left(-\Delta x\Delta y D_{zz}\frac{\partial c}{\partial z}|_{z+\Delta z}\right) \qquad \text{(A.1.1)}
\end{aligned}$$

with the interpretation of a component of the diffusivity tensor, D_{ij}, as the diffusive flux in direction i due to a concentration gradient in direction j. Diffusion based on the nine-component diffusivity tensor is termed *anisotropic* because of the directionality imparted through the components with $i \neq j$. For example, D_{xy} produces a diffusion flux in the x direction due to a concentration gradient in the y direction.

A straightforward rearrangement of Eq. (A.1.1) gives

$$\begin{aligned}
\frac{\partial c}{\partial t} = &\frac{D_{xx}\dfrac{\partial c}{\partial x}|_{x+\Delta x} - D_{xx}\dfrac{\partial c}{\partial x}|_x}{\Delta x}\\
&+\frac{D_{xy}\dfrac{\partial c}{\partial y}|_{x+\Delta x} - D_{xy}\dfrac{\partial c}{\partial y}|_x}{\Delta x}\\
&+\frac{D_{xz}\dfrac{\partial c}{\partial z}|_{x+\Delta x} - D_{xz}\dfrac{\partial c}{\partial z}|_x}{\Delta x}\\
&+\frac{D_{yx}\dfrac{\partial c}{\partial x}|_{y+\Delta y} - D_{yx}\dfrac{\partial c}{\partial x}|_y}{\Delta y}
\end{aligned}$$

$$
\begin{aligned}
&+ \frac{D_{yy}\frac{\partial c}{\partial y}|_{y+\Delta y} - D_{yy}\frac{\partial c}{\partial y}|_{y}}{\Delta y} \\
&+ \frac{D_{yz}\frac{\partial c}{\partial z}|_{y+\Delta y} - D_{yz}\frac{\partial c}{\partial z}|_{y}}{\Delta y} \\
&+ \frac{D_{zx}\frac{\partial c}{\partial x}|_{z+\Delta z} - D_{zx}\frac{\partial c}{\partial x}|_{z}}{\Delta z} \\
&+ \frac{D_{zy}\frac{\partial c}{\partial y}|_{z+\Delta z} - D_{zy}\frac{\partial c}{\partial y}|_{z}}{\Delta z} \\
&+ \frac{D_{zz}\frac{\partial c}{\partial z}|_{z+\Delta z} - D_{zz}\frac{\partial c}{\partial z}|_{z}}{\Delta z}
\end{aligned} \tag{A.1.2}
$$

In the limit $\Delta x, \Delta y, \Delta z \longrightarrow 0$, Eq. (A.1.2) becomes

$$
\begin{aligned}
\frac{\partial c}{\partial t} = &\frac{\partial\left(D_{xx}\frac{\partial c}{\partial x}\right)}{\partial x} + \frac{\partial\left(D_{xy}\frac{\partial c}{\partial y}\right)}{\partial x} + \frac{\partial\left(D_{xz}\frac{\partial c}{\partial z}\right)}{\partial x} \\
&+ \frac{\partial\left(D_{yx}\frac{\partial c}{\partial x}\right)}{\partial y} + \frac{\partial\left(D_{yy}\frac{\partial c}{\partial y}\right)}{\partial y} + \frac{\partial\left(D_{yz}\frac{\partial c}{\partial z}\right)}{\partial y} \\
&+ \frac{\partial\left(D_{zx}\frac{\partial c}{\partial x}\right)}{\partial z} + \frac{\partial\left(D_{zy}\frac{\partial c}{\partial y}\right)}{\partial z} + \frac{\partial\left(D_{zz}\frac{\partial c}{\partial z}\right)}{\partial z}
\end{aligned} \tag{A.1.3}
$$

Equation (A.1.3) can be expressed in vector–tensor notation (vectors and tensor expressed in boldface) as

$$
\frac{\partial c}{\partial t} = \nabla \cdot (\mathbf{D} \cdot \nabla c) \tag{A.1.4}
$$

where

$$
\nabla c = \mathbf{i}\frac{\partial c}{\partial x} + \mathbf{j}\frac{\partial c}{\partial y} + \mathbf{k}\frac{\partial c}{\partial z} \tag{A.1.5}
$$

and $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are unit vectors in Cartesian coordinates.

Here we introduce the idea of *dimensions of tensors*, and the *net dimensions of operations on tensors* ([1], p. 808). To start, refer to Table A.1.1 for some terminology.

Table A.1.1. Basic terminology for tensors of various orders

Term	Corresponding tensor order
Scalar	Zero order
Vector	First order
Tensor	Second and higher order

The reason the terminology in Table A.1.1 is useful is that it suggests dimensions that can be used to check expressions involving tensors such as the RHS of Eq. (A.1.4); this idea is explained next.

Various tensor operations, which we call *multiplications*, have associated orders summarized in the Table A.1.2, where we have used the symbol Σ to represent the *sum of the orders of the quantities being multiplied*. For example, using the definitions given in Table A.1.1,

- For the *product* of scalar s with vector $\mathbf{v}$ or the tensor $\mathbf{D}$ of order 2, written as $s\mathbf{v}$ or $s\mathbf{D}$, the orders are given by $\Sigma = 0 + 1 = 1$ or $\Sigma = 0 + 2 = 2$, respectively, and the result is a *vector* or a *tensor*.
- For the *cross product* of two vectors $\mathbf{v}$ and $\mathbf{w}$, written as $\mathbf{v} \times \mathbf{w}$, the order is given by $\Sigma - 1 = 1 + 1 - 1 = 1$, and the result is a *vector*.
- For the *dot product* of two vectors $\mathbf{v}$ and $\mathbf{w}$, written as $\mathbf{v} \cdot \mathbf{w}$, the order is given by $\Sigma - 2 = 1 + 1 - 2 = 0$, and the result is a *scalar*.
- For the *product* of two vectors $\mathbf{v}$ and $\mathbf{w}$ or two tensors $\mathbf{D}$ and $\mathbf{E}$ of order 2 (also known as the *dyadic product*), written as $\mathbf{vw}$ or $\mathbf{DE}$ without a product sign separating the variables, the order is given by $\Sigma = 1 + 1 = 2$ or $\Sigma = 2 + 2 = 4$, respectively, and the result is a *tensor* of order 2 or 4.
- For the *double-dot product* of two tensors $\mathbf{D}$ and $\mathbf{E}$, written as $\mathbf{D} : \mathbf{E}$, the order is given by $\Sigma - 4 = 2 + 2 - 4 = 0$, and the result is a *scalar*.
- For the *dot product* of a tensor $\mathbf{D}$ of order 2 with the vector $\mathbf{v}$, written as $\mathbf{D} \cdot \mathbf{v}$, the order is given by $\Sigma - 2 = 2 + 1 - 2 = 1$, and the result is a *vector*.

We now illustrate the use of Table A.1.2 applied to the LHS of Eq. (A.1.5). ∇c has two components: ∇, a vector with dimension 1, and c, a scalar with dimension 0. Also, it involves a dyadic multiplication, so from Table A.1.2, the net dimension is $\Sigma = 1 + 0 = 1$, a vector.

Table A.1.2. Orders for tensor multiplications

Name	Multiplication sign	Order of result
Product or dyadic product	None	Σ
Vector product or cross product	$\times$	$\Sigma - 1$
Scalar product or dot product	$\cdot$	$\Sigma - 2$
Double-dot product	:	$\Sigma - 4$

Continuing on with the RHS of Eq. (A.1.4),

$$\begin{aligned}
\mathbf{D}\cdot\nabla c &= (\mathbf{ii}D_{xx}+\mathbf{ij}D_{xy}+\mathbf{ik}D_{xz}\\
&\quad+\mathbf{ji}D_{yx}+\mathbf{jj}D_{yy}+\mathbf{jk}D_{yz}\\
&\quad+\mathbf{ki}D_{zx}+\mathbf{kj}D_{zy}+\mathbf{kk}D_{zz})\cdot\left(\mathbf{i}\frac{\partial c}{\partial x}+\mathbf{j}\frac{\partial c}{\partial y}+\mathbf{k}\frac{\partial c}{\partial z}\right)\\
&= \mathbf{i}(\mathbf{i}\cdot\mathbf{i})D_{xx}\frac{\partial c}{\partial x}+\mathbf{j}(\mathbf{i}\cdot\mathbf{i})D_{yx}\frac{\partial c}{\partial x}+\mathbf{k}(\mathbf{i}\cdot\mathbf{i})D_{zx}\frac{\partial c}{\partial x}\\
&\quad+\mathbf{i}(\mathbf{j}\cdot\mathbf{j})D_{xy}\frac{\partial c}{\partial y}+\mathbf{j}(\mathbf{j}\cdot\mathbf{j})D_{yy}\frac{\partial c}{\partial y}+\mathbf{k}(\mathbf{j}\cdot\mathbf{j})D_{zy}\frac{\partial c}{\partial y}\\
&\quad+\mathbf{i}(\mathbf{k}\cdot\mathbf{k})D_{xz}\frac{\partial c}{\partial z}+\mathbf{j}(\mathbf{k}\cdot\mathbf{k})D_{yz}\frac{\partial c}{\partial z}+\mathbf{k}(\mathbf{k}\cdot\mathbf{k})D_{zz}\frac{\partial c}{\partial z}\\
&= \mathbf{i}\left(D_{xx}\frac{\partial c}{\partial x}+D_{xy}\frac{\partial c}{\partial y}+D_{xz}\frac{\partial c}{\partial z}\right)\\
&\quad+\mathbf{j}\left(D_{yx}\frac{\partial c}{\partial x}+D_{yy}\frac{\partial c}{\partial y}+D_{yz}\frac{\partial c}{\partial z}\right)\\
&\quad+\mathbf{k}\left(D_{zx}\frac{\partial c}{\partial x}+D_{zy}\frac{\partial c}{\partial y}+D_{zz}\frac{\partial c}{\partial z}\right) \qquad \text{(A.1.6)}
\end{aligned}$$

with dimensions $\mathbf{D} \Leftrightarrow 2$ (a second-order tensor), $\nabla c \Leftrightarrow 1$ (a vector), and therefore net dimensions of (from the dot product of Table A.1.2) $\Sigma - 2 = 2 + 1 - 2 = 1$ (a vector). We have made use of the property of the *dot product* between orthogonal unit vectors (e.g., **a** and **b**):

θ	$\mathbf{a}\cdot\mathbf{b} = \lvert a\rvert\lvert b\rvert\cos\theta$
0	1
$\pi/2$	0

Then,

$$\begin{aligned}
\nabla\cdot(\mathbf{D}\cdot\nabla c) &= \left(\mathbf{i}\frac{\partial}{\partial x}+\mathbf{j}\frac{\partial}{\partial y}+\mathbf{j}\frac{\partial}{\partial z}\right)\\
&\quad\cdot\left[\mathbf{i}\left(D_{xx}\frac{\partial c}{\partial x}+D_{xy}\frac{\partial c}{\partial y}+D_{xz}\frac{\partial c}{\partial z}\right)\right.\\
&\quad+\mathbf{j}\left(D_{yx}\frac{\partial c}{\partial x}+D_{yy}\frac{\partial c}{\partial y}+D_{yz}\frac{\partial c}{\partial z}\right)\\
&\quad\left.+\mathbf{k}\left(D_{zx}\frac{\partial c}{\partial x}+D_{zy}\frac{\partial c}{\partial y}+D_{zz}\frac{\partial c}{\partial z}\right)\right]
\end{aligned}$$

$$= \frac{\partial}{\partial x}\left(D_{xx}\frac{\partial c}{\partial x} + D_{xy}\frac{\partial c}{\partial y} + D_{xz}\frac{\partial c}{\partial z}\right)$$
$$+ \frac{\partial}{\partial y}\left(D_{yx}\frac{\partial c}{\partial x} + D_{yy}\frac{\partial c}{\partial y} + D_{yz}\frac{\partial c}{\partial z}\right)$$
$$+ \frac{\partial}{\partial z}\left(D_{zx}\frac{\partial c}{\partial x} + D_{zy}\frac{\partial c}{\partial y} + D_{zz}\frac{\partial c}{\partial z}\right) \tag{A.1.7}$$

which is the RHS of Eq. (A.1.3) with dimensions $1 + 1 - 2 = 0$ (a scalar as required by Eq. (A.1.3) since the LHS is a scalar $\partial c/\partial t$, and c and t are scalars).

If $\mathbf{D}$ is a constant tensor, Eq. (A.1.3) becomes

$$\frac{\partial c}{\partial t} = D_{xx}\frac{\partial^2 c}{\partial x^2} + D_{xy}\frac{\partial^2 c}{\partial x\,\partial y} + D_{xz}\frac{\partial^2 c}{\partial x\,\partial z}$$
$$+ D_{yx}\frac{\partial^2 c}{\partial y\,\partial x} + D_{yy}\frac{\partial^2 c}{\partial y^2} + D_{yz}\frac{\partial^2 c}{\partial y\,\partial z}$$
$$+ D_{zx}\frac{\partial^2 c}{\partial z\,\partial x} + D_{zy}\frac{\partial^2 c}{\partial z\,\partial y} + D_{zz}\frac{\partial^2 c}{\partial z^2} \tag{A.1.8}$$

Equation (A.1.8) can be written as

$$\frac{\partial c}{\partial t} = \mathbf{D}^T : \nabla\nabla c \tag{A.1.9}$$

where

$$\nabla\nabla c = \left(\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}\right)\left(\mathbf{i}\frac{\partial c}{\partial x} + \mathbf{j}\frac{\partial c}{\partial y} + \mathbf{k}\frac{\partial c}{\partial z}\right)$$
$$= \mathbf{ii}\frac{\partial^2 c}{\partial x^2} + \mathbf{ij}\frac{\partial^2 c}{\partial x\,\partial y} + \mathbf{ik}\frac{\partial^2 c}{\partial x\,\partial z}$$
$$+ \mathbf{ji}\frac{\partial^2 c}{\partial y\,\partial x} + \mathbf{jj}\frac{\partial^2 c}{\partial y^2} + \mathbf{jk}\frac{\partial^2 c}{\partial y\,\partial z}$$
$$+ \mathbf{ki}\frac{\partial^2 c}{\partial z\,\partial x} + \mathbf{kj}\frac{\partial^2 c}{\partial z\,\partial y} + \mathbf{kk}\frac{\partial^2 c}{\partial z^2}$$

with dimensions $1 + 1 - 0 = 2$ (a tensor).

$$\mathbf{D}^T : \nabla\nabla c = (\mathbf{ii}D_{xx} + \mathbf{ij}D_{yx} + \mathbf{ik}D_{zx}$$
$$+ \mathbf{ji}D_{xy} + \mathbf{jj}D_{yy} + \mathbf{jk}D_{zy}$$
$$+ \mathbf{ki}D_{xz} + \mathbf{kj}D_{yz} + \mathbf{kk}D_{zz})$$
$$: \left(\mathbf{ii}\frac{\partial^2 c}{\partial x^2} + \mathbf{ij}\frac{\partial^2 c}{\partial x\,\partial y} + \mathbf{ik}\frac{\partial^2 c}{\partial x\,\partial z}\right.$$
$$+ \mathbf{ji}\frac{\partial^2 c}{\partial y\,\partial x} + \mathbf{jj}\frac{\partial^2 c}{\partial y^2} + \mathbf{jk}\frac{\partial^2 c}{\partial y\,\partial z}$$
$$\left. + \mathbf{ki}\frac{\partial^2 c}{\partial z\,\partial x} + \mathbf{kj}\frac{\partial^2 c}{\partial z\,\partial y} + \mathbf{kk}\frac{\partial^2 c}{\partial z^2}\right)$$

$$= \mathbf{ii} : \mathbf{ii} D_{xx} \frac{\partial^2 c}{\partial x^2} + \mathbf{ij} : \mathbf{ji} D_{yx} \frac{\partial^2 c}{\partial y\, \partial x} + \mathbf{ik} : \mathbf{ki} D_{zx} \frac{\partial^2 c}{\partial z\, \partial x}$$
$$+ \mathbf{ji} : \mathbf{ij} D_{xy} \frac{\partial^2 c}{\partial x\, \partial y} + \mathbf{jj} : \mathbf{jj} D_{yy} \frac{\partial^2 c}{\partial y^2} + \mathbf{jk} : \mathbf{kj} D_{zy} \frac{\partial^2 c}{\partial z\, \partial y}$$
$$+ \mathbf{ki} : \mathbf{ik} D_{xz} \frac{\partial^2 c}{\partial x\, \partial z} + \mathbf{kj} : \mathbf{jk} D_{yz} \frac{\partial^2 c}{\partial y\, \partial z} + \mathbf{kk} : \mathbf{kk} D_{zz} \frac{\partial^2 c}{\partial z^2}$$

with dimensions $2 + 2 - 4 = 0$ (a scalar).

Thus,

$$\frac{\partial c}{\partial t} = D_{xx} \frac{\partial^2 c}{\partial x^2} + D_{yx} \frac{\partial^2 c}{\partial y\, \partial x} + D_{zx} \frac{\partial^2 c}{\partial z\, \partial x}$$
$$+ D_{xy} \frac{\partial^2 c}{\partial x\, \partial y} + D_{yy} \frac{\partial^2 c}{\partial y^2} + D_{zy} \frac{\partial^2 c}{\partial z\, \partial y}$$
$$+ D_{xz} \frac{\partial^2 c}{\partial x\, \partial z} + D_{yz} \frac{\partial^2 c}{\partial y\, \partial z} + D_{zz} \frac{\partial^2 c}{\partial z^2} \tag{A.1.10}$$

which is Eq. (A.1.8).

We can now apply the same analysis in cylindrical coordinates by starting with a mass balance on an incremental volume $(r\Delta\theta)(\Delta r)(\Delta z)$ ([1], p. 840):

$$r\Delta\theta\Delta r\Delta z \frac{\partial c}{\partial t} = -r\Delta\theta\Delta z D_{rr} \frac{\partial c}{\partial r}|_r - \left(-r\Delta\theta\Delta z D_{rr} \frac{\partial c}{\partial r}|_{r+\Delta r}\right)$$
$$- r\Delta\theta\Delta z D_{r\theta} \frac{\partial c}{r\,\partial\theta}|_r - \left(-r\Delta\theta\Delta z D_{r\theta} \frac{\partial c}{r\,\partial\theta}|_{r+\Delta r}\right)$$
$$- r\Delta\theta\Delta z D_{rz} \frac{\partial c}{\partial z}|_r - \left(-r\Delta\theta\Delta z D_{rz} \frac{\partial c}{\partial z}|_{r+\Delta r}\right)$$
$$- \Delta r\Delta z D_{\theta r} \frac{\partial c}{\partial r}|_\theta - \left(-\Delta r\Delta z D_{\theta r} \frac{\partial c}{\partial r}|_{\theta+\Delta\theta}\right)$$
$$- \Delta r\Delta z D_{\theta\theta} \frac{\partial c}{r\,\partial\theta}|_\theta - \left(-\Delta r\Delta z D_{\theta\theta} \frac{\partial c}{r\,\partial\theta}|_{\theta+\Delta\theta}\right)$$
$$- \Delta r\Delta z D_{\theta z} \frac{\partial c}{\partial z}|_\theta - \left(-\Delta r\Delta z D_{\theta z} \frac{\partial c}{\partial z}|_{\theta+\Delta\theta}\right)$$
$$- r\Delta\theta\Delta r D_{zr} \frac{\partial c}{\partial r}|_z - \left(-r\Delta\theta\Delta r D_{zr} \frac{\partial c}{\partial r}|_{z+\Delta z}\right)$$
$$- r\Delta\theta\Delta r D_{z\theta} \frac{\partial c}{r\,\partial\theta}|_z - \left(-r\Delta\theta\Delta r D_{z\theta} \frac{\partial c}{r\,\partial\theta}|_{z+\Delta z}\right)$$
$$- r\Delta\theta\Delta r D_{zz} \frac{\partial c}{\partial z}|_z - \left(-r\Delta\theta\Delta r D_{zz} \frac{\partial c}{\partial z}|_{z+\Delta z}\right) \tag{A.1.11}$$

Rearrangement of Eq. (A.1.11) gives

$$\begin{aligned}
\frac{\partial c}{\partial t} = & \frac{r\Delta\theta\Delta z D_{rr}\frac{\partial c}{\partial r}|_{r+\Delta r} - r\Delta\theta\Delta z D_{rr}\frac{\partial c}{\partial r}|_{r}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{r\Delta\theta\Delta z D_{r\theta}\frac{\partial c}{r\,\partial\theta}|_{r+\Delta r} - r\Delta\theta\Delta z D_{r\theta}\frac{\partial c}{r\,\partial\theta}|_{r}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{r\Delta\theta\Delta z D_{rz}\frac{\partial c}{\partial z}|_{r+\Delta r} - r\Delta\theta\Delta z D_{rz}\frac{\partial c}{\partial z}|_{r}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{\Delta r\Delta z D_{\theta r}\frac{\partial c}{\partial r}|_{\theta+\Delta\theta} - \Delta r\Delta z D_{\theta r}\frac{\partial c}{\partial r}|_{\theta}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{\Delta r\Delta z D_{\theta\theta}\frac{\partial c}{r\,\partial\theta}|_{\theta+\Delta\theta} - \Delta r\Delta z D_{\theta\theta}\frac{\partial c}{r\,\partial\theta}|_{\theta}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{\Delta r\Delta z D_{\theta z}\frac{\partial c}{\partial z}|_{\theta+\Delta\theta} - \Delta r\Delta z D_{\theta z}\frac{\partial c}{\partial z}|_{\theta}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{r\Delta\theta\Delta r D_{zr}\frac{\partial c}{\partial r}|_{z+\Delta z} - r\Delta\theta\Delta r D_{zr}\frac{\partial c}{\partial r}|_{z}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{r\Delta\theta\Delta r D_{z\theta}\frac{\partial c}{r\,\partial\theta}|_{z+\Delta z} - r\Delta\theta\Delta r D_{z\theta}\frac{\partial c}{r\,\partial\theta}|_{z}}{r\Delta\theta\Delta r\Delta z} \\
& + \frac{r\Delta\theta\Delta r D_{zz}\frac{\partial c}{\partial z}|_{z+\Delta z} - r\Delta\theta\Delta r D_{zz}\frac{\partial c}{\partial z}|_{z}}{r\Delta\theta\Delta r\Delta z}
\end{aligned} \tag{A.1.12}$$

In the limit $\Delta r, \Delta\theta, \Delta z \to 0$, Eq. (A.1.12) becomes

$$\begin{aligned}
\frac{\partial c}{\partial t} = & \frac{\partial\left(rD_{rr}\frac{\partial c}{\partial r}\right)}{r\,\partial r} + \frac{\partial\left(rD_{r\theta}\frac{\partial c}{r\,\partial\theta}\right)}{r\,\partial r} + \frac{\partial\left(rD_{rz}\frac{\partial c}{\partial z}\right)}{r\,\partial r} \\
& + \frac{\partial\left(D_{\theta r}\frac{\partial c}{\partial r}\right)}{r\,\partial\theta} + \frac{\partial\left(D_{\theta\theta}\frac{\partial c}{r\,\partial\theta}\right)}{r\,\partial\theta} + \frac{\partial\left(D_{\theta z}\frac{\partial c}{\partial z}\right)}{r\,\partial\theta} \\
& + \frac{\partial\left(D_{zr}\frac{\partial c}{\partial r}\right)}{\partial z} + \frac{\partial\left(D_{z\theta}\frac{\partial c}{r\,\partial\theta}\right)}{\partial z} + \frac{\partial\left(D_{zz}\frac{\partial c}{\partial z}\right)}{\partial z}
\end{aligned} \tag{A.1.13}$$

For constant **D**, Eq. (A.1.13) becomes

$$\begin{aligned}\frac{\partial c}{\partial t} &= D_{rr}\left(\frac{\partial^2 c}{\partial r^2}+\frac{1}{r}\frac{\partial c}{\partial r}\right)+\frac{D_{r\theta}}{r}\frac{\partial^2 c}{\partial r\,\partial\theta}+D_{rz}\left(\frac{\partial^2 c}{\partial r\,\partial z}+\frac{1}{r}\frac{\partial c}{\partial z}\right)\\ &+\frac{D_{\theta r}}{r}\frac{\partial^2 c}{\partial\theta\,\partial r}+\frac{D_{\theta\theta}}{r^2}\frac{\partial^2 c}{\partial\theta^2}+\frac{D_{\theta z}}{r}\frac{\partial^2 c}{\partial\theta\,\partial z}\\ &+D_{zr}\frac{\partial^2 c}{\partial z\,\partial r}+\frac{D_{z\theta}}{r}\frac{\partial^2 c}{\partial z\,\partial\theta}+D_{zz}\frac{\partial^2 c}{\partial z^2}\end{aligned}\tag{A.1.14}$$

As a check on Eq. (A.1.14), we can consider the special case for which the main diagonal elements of the diffusivity tensor are equal ($D_{rr}=D_{\theta\theta}=D_{zz}=D$) and the off-diagonal diffusivities are zero. Equation (A.1.14) then becomes

$$\frac{\partial c}{\partial t}=D\left(\frac{\partial^2 c}{\partial r^2}+\frac{1}{r}\frac{\partial c}{\partial r}+\frac{1}{r^2}\frac{\partial^2 c}{\partial\theta^2}+\frac{\partial^2 c}{\partial z^2}\right)$$

which is the well-known form of the diffusion equation ([2], p. 3)

$$\frac{\partial c}{\partial t}=D\nabla^2 c$$

where ∇^2 is the Laplacian operator.

We now consider if Eq. (A.1.13) can be expressed in the vector–tensor notation of Eq. (A.1.4) with $\mathbf{i_r}$, $\mathbf{j}_\theta$, and $\mathbf{k_z}$ the unit vectors in cylindrical coordinates.

$$\nabla c=\mathbf{i}_r\frac{\partial c}{\partial r}+\mathbf{j}_\theta\frac{1}{r}\frac{\partial c}{\partial\theta}+\mathbf{k}_z\frac{\partial c}{\partial z}\tag{A.1.15}$$

$$\begin{aligned}\mathbf{D}\cdot\nabla c &= (\mathbf{i}_r\mathbf{i}_r D_{rr}+\mathbf{i}_r\mathbf{j}_\theta D_{r\theta}+\mathbf{i}_r\mathbf{k}_z D_{rz}\\ &+\mathbf{j}_\theta\mathbf{i}_r D_{\theta r}+\mathbf{j}_\theta\mathbf{j}_\theta D_{\theta\theta}+\mathbf{j}_\theta\mathbf{k}_z D_{\theta z}\\ &+\mathbf{k}_z\mathbf{i}_r D_{zr}+\mathbf{k}_z\mathbf{j}_\theta D_{z\theta}+\mathbf{k}_z\mathbf{k}_z D_{zz})\\ &\cdot\mathbf{i}_r\frac{\partial c}{\partial r}+\mathbf{j}_\theta\frac{1}{r}\frac{\partial c}{\partial\theta}+\mathbf{k}_z\frac{\partial c}{\partial z}\\ &=\mathbf{i}_r(\mathbf{i}_r\cdot\mathbf{i}_r)D_{rr}\frac{\partial c}{\partial r}+\mathbf{j}_\theta(\mathbf{i}_r\cdot\mathbf{i}_r)D_{\theta r}\frac{\partial c}{\partial r}+\mathbf{k}_z(\mathbf{i}_r\cdot\mathbf{i}_r)D_{zr}\frac{\partial c}{\partial r}\\ &+\mathbf{i}_r(\mathbf{j}_\theta\cdot\mathbf{j}_\theta)D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+\mathbf{j}_\theta(\mathbf{j}_\theta\cdot\mathbf{j}_\theta)D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+\mathbf{k}_z(\mathbf{j}_\theta\cdot\mathbf{j}_\theta)D_{z\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}\\ &+\mathbf{i}_r(\mathbf{k}_z\cdot\mathbf{k}_z)D_{rz}\frac{\partial c}{\partial z}+\mathbf{j}_\theta(\mathbf{k}_z\cdot\mathbf{k}_z)D_{\theta z}\frac{\partial c}{\partial z}+\mathbf{k}_z(\mathbf{k}_z\cdot\mathbf{k}_z)D_{zz}\frac{\partial c}{\partial z}\\ &=\mathbf{i}_r\left(D_{rr}\frac{\partial c}{\partial r}+D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+D_{rz}\frac{\partial c}{\partial z}\right)\\ &+\mathbf{j}_\theta\left(D_{\theta r}\frac{\partial c}{\partial r}+D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+D_{\theta z}\frac{\partial c}{\partial z}\right)\\ &+\mathbf{k}_z\left(D_{zr}\frac{\partial c}{\partial r}+D_{z\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+D_{zz}\frac{\partial c}{\partial z}\right)\end{aligned}$$

Finally, the RHS of Eq. (A.1.4) is ([2], p. 2)

$$\begin{aligned}\nabla\cdot(\mathbf{D}\cdot\nabla c) &= \left(\mathbf{i}_r\frac{1}{r}\frac{\partial}{\partial r}(r)+\mathbf{j}_\theta\frac{1}{r}\frac{\partial}{\partial\theta}+\mathbf{k}_z\frac{\partial}{\partial z}\right)\\ &\quad\cdot\left[\mathbf{i}_r\left(D_{rr}\frac{\partial c}{\partial r}+D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+D_{rz}\frac{\partial c}{\partial z}\right)\right.\\ &\quad+\mathbf{j}_\theta\left(D_{\theta r}\frac{\partial c}{\partial r}+D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+D_{\theta z}\frac{\partial c}{\partial z}\right)\\ &\quad\left.+\mathbf{k}_z\left(D_{zr}\frac{\partial c}{\partial r}+D_{z\theta}\frac{1}{r}\frac{\partial c}{\partial y}+D_{zz}\frac{\partial c}{\partial z}\right)\right]\\ &=\frac{1}{r}\frac{\partial}{\partial r}\left(rD_{rr}\frac{\partial c}{\partial r}+rD_{r\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+rD_{rz}\frac{\partial c}{\partial z}\right)\\ &\quad+\frac{1}{r}\frac{\partial}{\partial\theta}\left(D_{\theta r}\frac{\partial c}{\partial r}+D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial\theta}+D_{\theta z}\frac{\partial c}{\partial z}\right)\\ &\quad+\frac{\partial}{\partial z}\left(D_{zr}\frac{\partial c}{\partial r}+D_{z\theta}\frac{1}{r}\frac{\partial c}{\partial y}+D_{zz}\frac{\partial c}{\partial z}\right)\end{aligned}$$

which is the RHS of Eq. (A.1.13). In other words, Eq. (A.1.4) is correct for Cartesian coordinates *and* cylindrical coordinates.

For constant $\mathbf{D}$ (Eq. (A.1.14)), we now determine if Eq. (A.1.9) applies. ∇c is given by Eq. (A.1.15). Then

$$\begin{aligned}\nabla\nabla c &= \left(\mathbf{i}_r\frac{1}{r}\frac{\partial}{\partial r}(r)+\mathbf{j}_\theta\frac{1}{r}\frac{\partial}{\partial\theta}+\mathbf{k}_z\frac{\partial}{\partial z}\right)\left(\mathbf{i}_r\frac{\partial c}{\partial r}+\mathbf{j}_\theta\frac{1}{r}\frac{\partial c}{\partial\theta}+\mathbf{k}_z\frac{\partial c}{\partial z}\right)\\ &=\mathbf{i}_r\mathbf{i}_r\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial r}\right)+\mathbf{i}_r\mathbf{j}_\theta\frac{1}{r}\frac{\partial^2 c}{\partial r\,\partial\theta}+\mathbf{i}_r\mathbf{k}_z\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial z}\right)\\ &\quad+\mathbf{j}_\theta\mathbf{i}_r\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial r}+\mathbf{j}_\theta\mathbf{j}_\theta\frac{1}{r^2}\frac{\partial^2 c}{\partial\theta^2}+\mathbf{j}_\theta\mathbf{k}_z\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial z}\\ &\quad+\mathbf{k}_z\mathbf{i}_r\frac{\partial^2 c}{\partial z\,\partial r}+\mathbf{k}_z\mathbf{j}_\theta\frac{1}{r}\frac{\partial^2 c}{\partial z\,\partial\theta}+\mathbf{k}_z\mathbf{k}_z\frac{\partial^2 c}{\partial z^2}\end{aligned}$$

The RHS of Eq. (A.1.9) is then

$$\begin{aligned}\mathbf{D}^T:\nabla\nabla c &= (\mathbf{i}_r\mathbf{i}_rD_{rr}+\mathbf{i}_r\mathbf{j}_\theta D_{\theta r}+\mathbf{i}_r\mathbf{k}_zD_{zr}\\ &\quad+\mathbf{j}_\theta\mathbf{i}_rD_{r\theta}+\mathbf{j}_\theta\mathbf{j}_\theta D_{\theta\theta}+\mathbf{j}_\theta\mathbf{k}_zD_{z\theta}\\ &\quad+\mathbf{k}_z\mathbf{i}_rD_{rz}+\mathbf{k}_z\mathbf{j}_\theta D_{\theta z}+\mathbf{k}_z\mathbf{k}_zD_{zz})\\ &\quad:\left[\mathbf{i}_r\mathbf{i}_r\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial r}\right)+\mathbf{i}_r\mathbf{j}_\theta\frac{1}{r}\frac{\partial^2 c}{\partial r\,\partial\theta}+\mathbf{i}_r\mathbf{k}_z\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial z}\right)\right.\\ &\quad+\mathbf{j}_\theta\mathbf{i}_r\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial r}+\mathbf{j}_\theta\mathbf{j}_\theta\frac{1}{r^2}\frac{\partial^2 c}{\partial\theta^2}+\mathbf{j}_\theta\mathbf{k}_z\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial z}\\ &\quad\left.+\mathbf{k}_z\mathbf{i}_r\frac{\partial^2 c}{\partial z\,\partial r}+\mathbf{k}_z\mathbf{j}_\theta\frac{1}{r}\frac{\partial^2 c}{\partial z\,\partial\theta}+\mathbf{k}_z\mathbf{k}_z\frac{\partial^2 c}{\partial z^2}\right]\end{aligned}$$

$$
\begin{aligned}
&= \mathbf{i}_r\mathbf{i}_r : \mathbf{i}_r\mathbf{i}_r D_{rr}\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial r}\right) + \mathbf{i}_r\mathbf{j}_\theta : \mathbf{j}_\theta\mathbf{i}_r D_{\theta r}\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial r} + \mathbf{i}_r\mathbf{k}_z : \mathbf{k}_z\mathbf{i}_r D_{zr}\frac{\partial^2 c}{\partial z\,\partial r} \\
&\quad + \mathbf{j}_\theta\mathbf{i}_r : \mathbf{i}_r\mathbf{j}_\theta D_{r\theta}\frac{1}{r}\frac{\partial^2 c}{\partial r\,\partial\theta} + \mathbf{j}_\theta\mathbf{j}_\theta : \mathbf{j}_\theta\mathbf{j}_\theta D_{\theta\theta}\frac{1}{r^2}\frac{\partial^2 c}{\partial\theta^2} + \mathbf{j}_\theta\mathbf{k}_z : \mathbf{k}_z\,\mathbf{j}_\theta D_{z\theta}\frac{1}{r}\frac{\partial^2 c}{\partial z\,\partial\theta} \\
&\quad + \mathbf{k}_z\mathbf{i}_r : \mathbf{i}_r\mathbf{k}_z D_{rz}\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial z}\right) + \mathbf{k}_z\mathbf{j}_\theta : \mathbf{j}_\theta\mathbf{k}_z D_{\theta z}\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial z} + \mathbf{k}_z\mathbf{k}_z : \mathbf{k}_z\mathbf{k}_z D_{zz}\frac{\partial^2 c}{\partial z^2} \\
&= D_{rr}\left(\frac{\partial^2 c}{\partial r^2} + \frac{1}{r}\frac{\partial c}{\partial r}\right) + D_{\theta r}\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial r} + D_{zr}\frac{\partial^2 c}{\partial z\,\partial r} \\
&\quad + D_{r\theta}\frac{1}{r}\frac{\partial^2 c}{\partial r\,\partial\theta} + D_{\theta\theta}\frac{1}{r^2}\frac{\partial^2 c}{\partial\theta^2} + D_{z\theta}\frac{1}{r}\frac{\partial^2 c}{\partial z\,\partial\theta} \\
&\quad + D_{rz}\left(\frac{\partial^2 c}{\partial r\,\partial z} + \frac{1}{r}\frac{\partial c}{\partial z}\right) + D_{\theta z}\frac{1}{r}\frac{\partial^2 c}{\partial\theta\,\partial z} + D_{zz}\frac{\partial^2 c}{\partial z^2}
\end{aligned}
$$

which is the RHS of Eq. (A.1.14). Thus, Eq. (A.1.9) applies to Cartesian *and* cylindrical coordinates.

Finally, we apply the same analysis in spherical coordinates by starting with a mass balance on an incremental volume $(\Delta r)(r\Delta\theta)(r\sin\theta\Delta\phi)$ ([1], p. 840); θ is the angle of r with respect to the z axis and ϕ is the angle of r projected onto the x–y plane.

$$
\begin{aligned}
&(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,\frac{\partial c}{\partial t} \\
&\quad = -(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{rr}\frac{\partial c}{\partial r}|_r - \left(-(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{rr}\frac{\partial c}{\partial r}|_{r+\Delta r}\right) \\
&\qquad - (r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{r\theta}\frac{\partial c}{r\,\partial\theta}|_r - \left(-(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{r\theta}\frac{\partial c}{r\,\partial\theta}|_{r+\Delta r}\right) \\
&\qquad - (r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{r\phi}\frac{\partial c}{r\sin\theta\,\partial\phi}|_r - \left(-(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{r\phi}\frac{\partial c}{r\sin\theta\,\partial\phi}|_{r+\Delta r}\right) \\
&\qquad - (\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta r}\frac{\partial c}{\partial r}|_\theta - \left(-(\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta r}\frac{\partial c}{\partial r}|_{\theta+\Delta\theta}\right) \\
&\qquad - (\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\theta}\frac{\partial c}{r\,\partial\theta}|_\theta - \left(-(\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\theta}\frac{\partial c}{r\,\partial\theta}|_{\theta+\Delta\theta}\right) \\
&\qquad - (\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\phi}\frac{\partial c}{r\sin\theta\,\partial\phi}|_\theta - \left(-(\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\phi}\frac{\partial c}{r\sin\theta\,\partial\phi}|_{\theta+\Delta\theta}\right) \\
&\qquad - (\Delta r)\,(r\Delta\theta)\,D_{\phi r}\frac{\partial c}{\partial r}|_\phi - \left(-(\Delta r)\,(r\Delta\theta)\,D_{\phi r}\frac{\partial c}{\partial r}|_{\phi+\Delta\phi}\right) \\
&\qquad - (\Delta r)\,(r\Delta\theta)\,D_{\phi\theta}\frac{\partial c}{r\,\partial\theta}|_\phi - \left(-(\Delta r)\,(r\Delta\theta)\,D_{\phi\theta}\frac{\partial c}{r\,\partial\theta}|_{\phi+\Delta\phi}\right) \\
&\qquad - (\Delta r)\,(r\Delta\theta)\,D_{\phi\phi}\frac{\partial c}{r\sin\theta\,\partial\phi}|_\phi - \left(-(\Delta r)\,(r\Delta\theta)\,D_{\phi\phi}\frac{\partial c}{r\sin\theta\partial\phi}|_{\phi+\Delta\phi}\right) \qquad \text{(A.1.16)}
\end{aligned}
$$

Rearrangement of Eq. (A.1.16) gives

$$\frac{\partial c}{\partial t} = \frac{(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{rr}\dfrac{\partial c}{\partial r}|_{r+\Delta r} - (r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{rr}\dfrac{\partial c}{\partial r}|_{r}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{r\theta}\dfrac{\partial c}{r\,\partial\theta}|_{r+\Delta r} - (r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{r\theta}\dfrac{\partial c}{r\,\partial\theta}|_{r}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{rz}\dfrac{\partial c}{r\sin\theta\,\partial\phi}|_{r+\Delta r} - (r\Delta\theta)\,(r\sin\theta\Delta\phi)\,D_{rz}\dfrac{\partial c}{r\sin\theta\,\partial\phi}|_{r}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta r}\dfrac{\partial c}{\partial r}|_{\theta+\Delta\theta} - (\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta r}\dfrac{\partial c}{\partial r}|_{\theta}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\theta}\dfrac{\partial c}{r\,\partial\theta}|_{\theta+\Delta\theta} - (\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\theta}\dfrac{\partial c}{r\,\partial\theta}|_{\theta}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\phi}\dfrac{\partial c}{r\sin\theta\,\partial\phi}|_{\theta+\Delta\theta} - (\Delta r)\,(r\sin\theta\Delta\phi)\,D_{\theta\phi}\dfrac{\partial c}{r\sin\theta\,\partial\phi}|_{\theta}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(\Delta r)\,(r\Delta\theta)\,D_{\phi r}\dfrac{\partial c}{\partial r}|_{\phi+\Delta\phi} - (\Delta r)\,(r\Delta\theta)\,D_{\phi r}\dfrac{\partial c}{\partial r}|_{\phi}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(\Delta r)\,(r\Delta\theta)\,D_{\phi\theta}\dfrac{\partial c}{r\,\partial\theta}|_{\phi+\Delta\phi} - (\Delta r)\,(r\Delta\theta)\,D_{\phi\theta}\dfrac{\partial c}{r\,\partial\theta}|_{\phi}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$
$$+ \frac{(\Delta r)\,(r\Delta\theta)\,D_{\phi\phi}\dfrac{\partial c}{r\sin\theta\,\partial\phi}|_{\phi+\Delta\phi} - (\Delta r)\,(r\Delta\theta)\,D_{\phi\phi}\dfrac{\partial c}{r\sin\theta\,\partial\phi}|_{\phi}}{(\Delta r)\,(r\Delta\theta)\,(r\sin\theta\Delta\phi)}$$

or after some simplification,

$$\frac{\partial c}{\partial t} = \frac{1}{r^2}\frac{r^2 D_{rr}\dfrac{\partial c}{\partial r}|_{r+\Delta r} - r^2 D_{rr}\dfrac{\partial c}{\partial r}|_{r}}{\Delta r}$$
$$+ \frac{1}{r^2}\frac{rD_{r\theta}\dfrac{\partial c}{\partial\theta}|_{r+\Delta r} - rD_{r\theta}\dfrac{\partial c}{\partial\theta}|_{r}}{\Delta r}$$
$$+ \frac{1}{r^2\,\sin\theta}\frac{rD_{r\phi}\dfrac{\partial c}{\partial\phi}|_{r+\Delta r} - rD_{r\phi}\dfrac{\partial c}{\partial\phi}|_{r}}{\Delta r}$$
$$+ \frac{1}{r\sin\theta}\frac{(\sin\theta)\,D_{\theta r}\dfrac{\partial c}{\partial r}|_{\theta+\Delta\theta} - (\sin\theta)\,D_{\theta r}\dfrac{\partial c}{\partial r}|_{\theta}}{\Delta\theta}$$

$$+\frac{1}{r\sin\theta}\frac{\sin\theta D_{\theta\theta}\dfrac{\partial c}{r\,\partial\theta}|_{\theta+\Delta\theta}-\sin\theta D_{\theta\theta}\dfrac{\partial c}{r\,\partial\theta}|_{\theta}}{\Delta\theta}$$
$$+\frac{1}{r^2\sin\theta}\frac{D_{\theta\phi}\dfrac{\partial c}{\partial\phi}|_{\theta+\Delta\theta}-D_{\theta\phi}\dfrac{\partial c}{\partial\phi}|_{\theta}}{\Delta\theta}$$
$$+\frac{1}{r\sin\theta}\frac{D_{\phi r}\dfrac{\partial c}{\partial r}|_{\phi+\Delta\phi}-D_{\phi r}\dfrac{\partial c}{\partial r}|_{\phi}}{\Delta\phi}$$
$$+\frac{1}{r^2\sin\theta}\frac{D_{\phi\theta}\dfrac{\partial c}{\partial\theta}|_{\phi+\Delta\phi}-D_{\phi\theta}\dfrac{\partial c}{\partial\theta}|_{\phi}}{\Delta\phi}$$
$$+\frac{1}{r^2\sin^2\theta}\frac{D_{\phi\phi}\dfrac{\partial c}{\partial\phi}|_{\phi+\Delta\phi}-D_{\phi\phi}\dfrac{\partial c}{\partial\phi}|_{\phi}}{\Delta\phi} \tag{A.1.17}$$

In the limit $\Delta r, \Delta\theta, \Delta\phi \longrightarrow 0$, Eq. (A.1.17) becomes

$$\frac{\partial c}{\partial t}=\frac{1}{r^2}\frac{\partial\left(r^2D_{rr}\dfrac{\partial c}{\partial r}\right)}{\partial r}+\frac{1}{r^2}\frac{\partial\left(rD_{r\theta}\dfrac{\partial c}{\partial\theta}\right)}{\partial r}+\frac{1}{r^2\sin\theta}\frac{\partial\left(rD_{r\phi}\dfrac{\partial c}{\partial\phi}\right)}{\partial r}$$
$$+\frac{1}{r\sin\theta}\frac{\partial\left(\sin\theta D_{\theta r}\dfrac{\partial c}{\partial r}\right)}{\partial\theta}+\frac{1}{r^2\sin\theta}\frac{\partial\left(\sin\theta D_{\theta\theta}\dfrac{\partial c}{\partial\theta}\right)}{\partial\theta}+\frac{1}{r^2\sin\theta}\frac{\partial\left(D_{\theta\phi}\dfrac{\partial c}{\partial\phi}\right)}{\partial\theta}$$
$$+\frac{1}{r\sin\theta}\frac{\partial\left(D_{\phi r}\dfrac{\partial c}{\partial r}\right)}{\partial\phi}+\frac{1}{r^2\sin\theta}\frac{\partial\left(D_{\phi\theta}\dfrac{\partial c}{\partial\theta}\right)}{\partial\phi}+\frac{1}{r^2\sin^2\theta}\frac{\partial\left(D_{\phi\phi}\dfrac{\partial c}{\partial\phi}\right)}{\partial\phi} \tag{A.1.18}$$

For constant $\mathbf{D}$, Eq. (A.1.18) becomes

$$\frac{\partial c}{\partial t}=D_{rr}\left(\frac{\partial^2c}{\partial r^2}+\frac{2}{r}\frac{\partial c}{\partial r}\right)+\frac{D_{r\theta}}{r}\left(\frac{\partial^2c}{\partial r\,\partial\theta}+\frac{1}{r}\frac{\partial c}{\partial\theta}\right)+\frac{D_{r\phi}}{r\sin\theta}\left(\frac{\partial^2c}{\partial r\,\partial\phi}+\frac{1}{r}\frac{\partial c}{\partial\phi}\right)$$
$$+\frac{D_{\theta r}}{r}\left(\frac{\partial^2c}{\partial\theta\,\partial r}+\frac{\cos\theta}{\sin\theta}\frac{\partial c}{\partial r}\right)+\frac{D_{\theta\theta}}{r^2}\left(\frac{\partial^2c}{\partial\theta^2}+\frac{\cos\theta}{\sin\theta}\frac{\partial c}{\partial\theta}\right)+\frac{D_{\theta\phi}}{r^2\sin\theta}\frac{\partial^2c}{\partial\theta\,\partial\phi}$$
$$+\frac{D_{\phi r}}{r\sin\theta}\frac{\partial^2c}{\partial\phi\,\partial r}+\frac{D_{\phi\theta}}{r^2\sin\theta}\frac{\partial^2c}{\partial\phi\,\partial\theta}+\frac{D_{\phi\phi}}{r^2\sin^2\theta}\frac{\partial^2c}{\partial\phi^2} \tag{A.1.19}$$

As a check on Eq. (A.1.19), we can consider the special case for which the main diagonal elements of the diffusivity tensor are equal ($D_{rr}=D_{\theta\theta}=D_{\phi\phi}=D$) and the off-diagonal diffusivities are zero. Equation (A.1.19) then becomes

$$\frac{\partial c}{\partial t}=D_{rr}\left(\frac{\partial^2c}{\partial r^2}+\frac{2}{r}\frac{\partial c}{\partial r}\right)+\frac{D_{\theta\theta}}{r^2}\left(\frac{\partial^2c}{\partial\theta^2}+\frac{\cos\theta}{\sin\theta}\frac{\partial c}{\partial\theta}\right)+\frac{D_{\phi\phi}}{r^2\sin^2\theta}\frac{\partial^2c}{\partial\phi^2}$$

which is ([2], p. 3)

$$\frac{\partial c}{\partial t} = D\nabla^2 c$$

in spherical coordinates.

We now consider if Eq. (A.1.18) can be expressed in the vector–tensor notation of Eq. (A.1.4).

$$\nabla c = \mathbf{i}_r \frac{\partial c}{\partial r} + \mathbf{j}_\theta \frac{1}{r}\frac{\partial c}{\partial \theta} + \mathbf{k}_\phi \frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi} \tag{A.1.20}$$

$$\begin{aligned}
\mathbf{D}\cdot\nabla c = {} & (\mathbf{i}_r\mathbf{i}_r D_{rr} + \mathbf{i}_r\mathbf{j}_\theta D_{r\theta} + \mathbf{i}_r\mathbf{k}_\phi D_{r\phi} \\
& + \mathbf{j}_\theta\mathbf{i}_r D_{\theta r} + \mathbf{j}_\theta\mathbf{j}_\theta D_{\theta\theta} + \mathbf{j}_\theta\mathbf{k}_\phi D_{\theta\phi} \\
& + \mathbf{k}_\phi\mathbf{i}_r D_{\phi r} + \mathbf{k}_\phi\mathbf{j}_\theta D_{\phi\theta} + \mathbf{k}_\phi\mathbf{k}_\phi D_{\phi\phi}) \\
& \cdot \left(\mathbf{i}_r \frac{\partial c}{\partial r} + \mathbf{j}_\theta \frac{1}{r}\frac{\partial c}{\partial \theta} + \mathbf{k}_\phi \frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right) \\
= {} & \mathbf{i}_r(\mathbf{i}_r\cdot\mathbf{i}_r)D_{rr}\frac{\partial c}{\partial r} + \mathbf{j}_\theta(\mathbf{i}_r\cdot\mathbf{i}_r)D_{\theta r}\frac{\partial c}{\partial r} + \mathbf{k}_\phi(\mathbf{i}_r\cdot\mathbf{i}_r)D_{\phi r}\frac{\partial c}{\partial r} \\
& + \mathbf{i}_r(\mathbf{j}_\theta\cdot\mathbf{j}_\theta)D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + \mathbf{j}_\theta(\mathbf{j}_\theta\cdot\mathbf{j}_\theta)D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + \mathbf{k}_\phi(\mathbf{j}_\theta\cdot\mathbf{j}_\theta)D_{\phi\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} \\
& + \mathbf{i}_r(\mathbf{k}_\phi\cdot\mathbf{k}_\phi)D_{r\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi} + \mathbf{j}_\theta(\mathbf{k}_\phi\cdot\mathbf{k}_\phi)D_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi} \\
& + \mathbf{k}_\phi(\mathbf{k}_\phi\cdot\mathbf{k}_\phi)D_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi} \\
= {} & \mathbf{i}_r\left(D_{rr}\frac{\partial c}{\partial r} + D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + D_{r\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right) \\
& + \mathbf{j}_\theta\left(D_{\theta r}\frac{\partial c}{\partial r} + D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + D_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right) \\
& + \mathbf{k}_\phi\left(D_{\phi r}\frac{\partial c}{\partial r} + D_{\phi\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + D_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right)
\end{aligned}$$

Finally, the RHS of Eq. (A.1.4) is ([2], p. 2)

$$\begin{aligned}
\nabla\cdot(\mathbf{D}\cdot\nabla c) = {} & \left(\mathbf{i}_r\frac{1}{r^2}\frac{\partial}{\partial r}(r^2) + \mathbf{j}_\theta\frac{1}{r\sin\theta}\frac{\partial}{\partial \theta}(\sin\theta) + \mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial}{\partial \phi}\right) \\
& \cdot\left[\mathbf{i}_r\left(D_{rr}\frac{\partial c}{\partial r} + D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + D_{r\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right)\right. \\
& + \mathbf{j}_\theta\left(D_{\theta r}\frac{\partial c}{\partial r} + D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + D_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right) \\
& \left. + \mathbf{k}_\phi\left(D_{\phi r}\frac{\partial c}{\partial r} + D_{\phi\theta}\frac{1}{r}\frac{\partial c}{\partial \theta} + D_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right)\right]
\end{aligned}$$

$$= \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 D_{rr}\frac{\partial c}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 D_{r\theta}\frac{1}{r}\frac{\partial c}{\partial \theta}\right) + \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 D_{r\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right)$$
$$+ \frac{1}{r\sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta\, D_{\theta r}\frac{\partial c}{\partial r}\right) + \frac{1}{r\sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta\, D_{\theta\theta}\frac{1}{r}\frac{\partial c}{\partial \theta}\right)$$
$$+ \frac{1}{r\sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta\, D_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right) + \frac{1}{r\sin\theta}\frac{\partial}{\partial \phi}\left(D_{\phi r}\frac{\partial c}{\partial r}\right)$$
$$+ \frac{1}{r\sin\theta}\frac{\partial}{\partial \phi}\left(D_{\phi\theta}\frac{1}{r}\frac{\partial c}{\partial \theta}\right) + \frac{1}{r\sin\theta}\frac{\partial}{\partial \phi}\left(D_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial \phi}\right)\Bigg) \qquad \text{(A.1.21)}$$

which is the RHS of Eq. (A.1.18). In other words, Eq. (A.1.4) is correct for Cartesian, cylindrical, *and* spherical coordinates. The final result is therefore Eq. (A.1.18).

For **D** constant (Eq. (A.1.19)), we now determine if Eq. (A.1.9) applies. ∇c is given by Eq. (A.1.20). Then

$$\nabla\nabla c = \left(\mathbf{i}_r\frac{1}{r^2}\frac{\partial}{\partial r}(r^2) + \mathbf{j}_\theta\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta) + \mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\right)\left(\mathbf{i}_r\frac{\partial c}{\partial r} + \mathbf{j}_\theta\frac{1}{r}\frac{\partial c}{\partial\theta} + \mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)$$
$$= \mathbf{i}_r\mathbf{i}_r\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial c}{\partial r}\right) + \mathbf{i}_r\mathbf{j}_\theta\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r}\frac{\partial c}{\partial\theta}\right) + \mathbf{i}_r\mathbf{k}_\phi\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)$$
$$+ \mathbf{j}_\theta\mathbf{i}_r\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial c}{\partial r}\right) + \mathbf{j}_\theta\mathbf{j}_\theta\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r}\frac{\partial c}{\partial\theta}\right)$$
$$+ \mathbf{j}_\theta\mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)$$
$$+ \mathbf{k}_\phi\mathbf{i}_r\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{\partial c}{\partial r} + \mathbf{k}_\phi\mathbf{j}_\theta\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r}\frac{\partial c}{\partial\theta} + \mathbf{k}_\phi\mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}$$

The RHS of Eq. (A.1.9) is then

$$\mathbf{D}^T : \nabla\nabla c = (\mathbf{i}_r\mathbf{i}_r D_{rr} + \mathbf{i}_r\mathbf{j}_\theta D_{\theta r} + \mathbf{i}_r\mathbf{k}_\phi D_{\phi r}$$
$$+ \mathbf{j}_\theta\mathbf{i}_r D_{r\theta} + \mathbf{j}_\theta\mathbf{j}_\theta D_{\theta\theta} + \mathbf{j}_\theta\mathbf{k}_\phi D_{\phi\theta}$$
$$+ \mathbf{k}_\phi\mathbf{i}_r D_{r\phi} + \mathbf{k}_\phi\mathbf{j}_\theta D_{\theta\phi} + \mathbf{k}_\phi\mathbf{k}_\phi D_{\phi\phi})$$
$$: \Bigg[\mathbf{i}_r\mathbf{i}_r\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial c}{\partial r}\right) + \mathbf{i}_r\mathbf{j}_\theta\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r}\frac{\partial c}{\partial\theta}\right) + \mathbf{i}_r\mathbf{k}_\phi\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)$$
$$+ \mathbf{j}_\theta\mathbf{i}_r\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial c}{\partial r}\right) + \mathbf{j}_\theta\mathbf{j}_\theta\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r}\frac{\partial c}{\partial\theta}\right)$$
$$+ \mathbf{j}_\theta\mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)$$
$$+ \mathbf{k}_\phi\mathbf{i}_r\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{\partial c}{\partial r} + \mathbf{k}_\phi\mathbf{j}_\theta\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r}\frac{\partial c}{\partial\theta} + \mathbf{k}_\phi\mathbf{k}_\phi\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\Bigg]$$

$$
\begin{aligned}
&= \mathbf{i}_r\mathbf{i}_r : \mathbf{i}_r\mathbf{i}_r D_{rr}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial c}{\partial r}\right) + \mathbf{i}_r\mathbf{j}_\theta : \mathbf{j}_\theta\mathbf{i}_r D_{\theta r}\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial c}{\partial r}\right)\\
&\quad + \mathbf{i}_r\mathbf{k}_\phi : \mathbf{k}_\phi\mathbf{i}_r D_{\phi r}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{\partial c}{\partial r}\\
&\quad + \mathbf{j}_\theta\mathbf{i}_r : \mathbf{i}_r\mathbf{j}_\theta D_{r\theta}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r}\frac{\partial c}{\partial\theta}\right) + \mathbf{j}_\theta\mathbf{j}_\theta : \mathbf{j}_\theta\mathbf{j}_\theta D_{\theta\theta}\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r}\frac{\partial c}{\partial\theta}\right)\\
&\quad + \mathbf{j}_\theta\mathbf{k}_\phi : \mathbf{k}_\phi\mathbf{j}_\theta D_{\phi\theta}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r}\frac{\partial c}{\partial\theta}\\
&\quad + \mathbf{k}_\phi\mathbf{i}_r : \mathbf{i}_r\mathbf{k}_\phi D_{r\phi}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)\\
&\quad + \mathbf{k}_\phi\mathbf{j}_\theta : \mathbf{j}_\theta\mathbf{k}_\phi D_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)\\
&\quad + \mathbf{k}_\phi\mathbf{k}_\phi : \mathbf{k}_\phi\mathbf{k}_\phi D_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\\
&= D_{rr}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial c}{\partial r}\right) + D_{\theta r}\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial c}{\partial r}\right) + D_{\phi r}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{\partial c}{\partial r}\\
&\quad + D_{r\theta}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r}\frac{\partial c}{\partial\theta}\right) + D_{\theta\theta}\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r}\frac{\partial c}{\partial\theta}\right) + D_{\phi\theta}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r}\frac{\partial c}{\partial\theta}\\
&\quad + D_{r\phi}\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right) + D_{\theta\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\right)\\
&\quad + D_{\phi\phi}\frac{1}{r\sin\theta}\frac{\partial}{\partial\phi}\frac{1}{r\sin\theta}\frac{\partial c}{\partial\phi}\\
&= D_{rr}\left(\frac{\partial^2 c}{\partial r^2} + \frac{2}{r}\frac{\partial c}{\partial r}\right) + \frac{D_{\theta r}}{r}\left(\frac{\partial^2 c}{\partial\theta\,\partial r} + \frac{\cos\theta}{\sin\theta}\frac{\partial c}{\partial r}\right) + \frac{D_{\phi r}}{r\sin\theta}\frac{\partial^2 c}{\partial\phi\,\partial r}\\
&\quad + \frac{D_{r\theta}}{r}\left(\frac{\partial^2 c}{\partial r\,\partial\theta} + \frac{1}{r}\frac{\partial c}{\partial\theta}\right) + \frac{D_{\theta\theta}}{r^2}\left(\frac{\partial^2 c}{\partial\theta^2} + \frac{\cos\theta}{\sin\theta}\frac{\partial c}{\partial\theta}\right) + \frac{D_{\phi\theta}}{r^2\sin\theta}\frac{\partial^2 c}{\partial\phi\,\partial\theta}\\
&\quad + \frac{D_{r\phi}}{r\sin\theta}\left(\frac{\partial^2 c}{\partial r\,\partial\phi} + \frac{1}{r}\frac{\partial c}{\partial\phi}\right) + \frac{D_{\theta\phi}}{r^2\sin\theta}\frac{\partial^2 c}{\partial\theta\,\partial\phi} + \frac{D_{\phi\phi}}{r^2\sin^2\theta}\frac{\partial^2 c}{\partial\phi^2}
\end{aligned}
$$

which is the RHS of Eq. (A.1.19). Thus, Eq. (A.1.9) applies to Cartesian, cylindrical, *and* spherical coordinates.

As a few concluding points:

1. The preceding analysis (mass balance on an incremental volume) can be applied in any *orthogonal coordinate system* ([3], pp. 26, 115).
2. The analysis can be extended to include convective terms, and thereby arrive at tensor forms of *convective–diffusive* (*hyperbolic–parabolic*) PDE systems.
3. In principle, the method of lines (MOL) can be applied to these various PDE systems. In practice, experience has indicated that, in particular, when mixed

partial derivatives are included, as in most of the preceding PDEs, numerical difficulties can develop.

4. Mixed partials can be calculated within the MOL by using stagewise differentiation. For example, to compute the mixed partial $\partial^2 u/\partial x\,\partial y$, we can differentiate u first with respect to x and then with respect to y.
5. Because numerical difficulties can occur with each new problem, generally some trial and error is required to arrive at a numerical solution with acceptable accuracy and computational effort. In other words, the MOL is not necessarily a straightforward, mechanical procedure, and success in using it is generally dependent, to some degree, on the experience and creativity of the analyst. Our hope is that the preceding examples can serve as a guide to the solution of new problems.

REFERENCES

[1] Bird, R. B., W. E. Stewart, and E. N. Lightfoot (2002), *Transport Phenomena*, 2nd ed., Wiley, New York

[2] vande Wouwer, A., P. Saucez, and W. E. Schiesser (Eds.) (2001), *Adaptive Method of Lines*, CRC Press, Boca Raton, FL

[3] Morse, P. and H. Feshbach (1953), *Methods of Theoretical Physics*, McGraw-Hill, New York

APPENDIX 2

Order Conditions for Finite-Difference Approximations

We have used the DSS (Differentiation in Space Subroutines[1]) to compute finite-difference (FD) approximations of spatial (boundary-value) derivatives in partial differential equations (PDEs). Also, we have mentioned the order of these approximations, for example, second-order approximations in `dss002`. In this appendix, we explain briefly what is meant by the order of an FD.

If we consider the derivative of a polynomial of order p, for example, of second order with $p = 2$,

$$p_2(x) = a_0 + a_1 x + a_2 x^2 \tag{A.2.1}$$

then the pth derivative is a constant ($2a_2$) and the $(p + 1)$st derivative is zero.

Then a *pth-order FD differentiates a pth-order polynomial exactly*. Thus, if we apply `dss002` (with three-point, second-order FDs) to a second-order polynomial, the numerical derivatives should be exact. But the numerical derivatives of third- and higher-order polynomials will not be exact.

To illustrate these ideas, consider the test program given in Listing A.2.1.

```
%
  clear all
  clc
%
% Varying order of differentiator
  for ndss=2:2:10
%
% Varying order of test polynomial
  for norder=1:12
```

[1] These routines were first written in Fortran, and thus the name "subroutines." They were subsequently translated into Matlab, for which the corresponding name is "function." Also, we use just "routines" in referring to them.

```
%
% Grid in x
  xl=0.0;
  xu=1.0;
  n=11;
  dx=(xu-xl)/(n-1);
%
% Test polynomial
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u(i)=x(i)^norder;
  end
%
% ux
  if ndss==2   ux=dss002(xl,xu,n,u); end
  if ndss==4   ux=dss004(xl,xu,n,u); end
  if ndss==6   ux=dss006(xl,xu,n,u); end
  if ndss==8   ux=dss008(xl,xu,n,u); end
  if ndss==10  ux=dss010(xl,xu,n,u); end
%
% Comparison of numerical and analytical ux
  if(norder-ndss)>-1 & (norder-ndss)< 3
  fprintf('\n\n ndss = %2d   order = %2d\n\n',ndss,norder);
  for i=1:n
    uxa(i)=norder*x(i)^(norder-1);
    err(i)=ux(i)-uxa(i);
    fprintf(' i = %3d  ux(i) = %8.5f  uxa(i) = %8.5f
            err(i) = %8.5f\n',i,ux(i),uxa(i),err(i));
  end
  end
%
% Next order
  end
%
% Next differentiator
  end
```

Listing A.2.1. Program `ux_poly_1` for the numerical differentiation of polynomials

We can note the following points about this program:

1. The pair of nested `for` loops varies the order of the FD approximation (outer loop), and for each FD order, the order of the test polynomial is varied (inner loop).

```
%
  clear all
  clc
%
% Varying order of differentiator
  for ndss=2:2:10
%
% Varying order of test polynomial
  for norder=1:12
```

2. A grid in x is defined and the polynomial of order `norder` is evaluated as `u(i)`.

```
%
% Grid in x
  xl=0.0;
  xu=1.0;
  n=11;
  dx=(xu-xl)/(n-1);
%
% Test polynomial
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u(i)=x(i)^norder;
  end
```

3. The numerical first derivative `ux` of `u` is then computed for FD orders two (`dss002`) to ten (`dss010`).

```
%
% ux
  if ndss==2   ux=dss002(xl,xu,n,u); end
  if ndss==4   ux=dss004(xl,xu,n,u); end
  if ndss==6   ux=dss006(xl,xu,n,u); end
  if ndss==8   ux=dss008(xl,xu,n,u); end
  if ndss==10  ux=dss010(xl,xu,n,u); end
```

4. The exact derivative (from analytical differentiation) is put in array `uxa(i)` and the difference between the exact and numerical derivatives is computed as `err(i)`.

```
%
% Comparison of numerical and analytical ux
  if(norder-ndss)>-1 & (norder-ndss)< 3
  fprintf('\n\n ndss = %2d   order = %2d\n\n',ndss,norder);
  for i=1:n
    uxa(i)=norder*x(i)^(norder-1);
    err(i)=ux(i)-uxa(i);
    fprintf(' i = %3d  ux(i) = %8.5f  uxa(i) = %8.5f
              err(i) = %8.5f\n',i,ux(i),uxa(i),err(i));
  end
  end
```

The first `if` statement limits the comparison of the analytical and exact derivatives (in the output) to the cases `norder = ndss` to `norder = ndss +2` (just to display the most interesting conditions when the order of the polynomial equals or exceeds by 2 the order of the differentiator).

A selected portion of the output from this test program is given in Table A.2.1. We can note the following details about the output given in Table A.2.1:

1. For `ndss = norder = 2`, the numerical (FD) derivative is exact (to machine accuracy, which is about 15 figures for Matlab running on a 32-bit machine).

```
ndss =  2   order =  2

 i =  1  ux(i) = 0.00000  uxa(i) = 0.00000  err(i) =  0.00000
 i =  2  ux(i) = 0.20000  uxa(i) = 0.20000  err(i) =  0.00000
 i =  3  ux(i) = 0.40000  uxa(i) = 0.40000  err(i) =  0.00000
 i =  4  ux(i) = 0.60000  uxa(i) = 0.60000  err(i) =  0.00000
 i =  5  ux(i) = 0.80000  uxa(i) = 0.80000  err(i) = -0.00000
 i =  6  ux(i) = 1.00000  uxa(i) = 1.00000  err(i) =  0.00000
 i =  7  ux(i) = 1.20000  uxa(i) = 1.20000  err(i) =  0.00000
 i =  8  ux(i) = 1.40000  uxa(i) = 1.40000  err(i) =  0.00000
 i =  9  ux(i) = 1.60000  uxa(i) = 1.60000  err(i) = -0.00000
 i = 10  ux(i) = 1.80000  uxa(i) = 1.80000  err(i) = -0.00000
 i = 11  ux(i) = 2.00000  uxa(i) = 2.00000  err(i) = -0.00000
```

2. For `ndss = 2, norder = 3`, the numerical (FD) derivatives are not exact. In fact, the difference between the exact and numerical derivatives is a constant that is given by the *principal error term* (with $p = 2$).

$$\texttt{err} = c\frac{d^{p+1}f(x)}{dx^{p+1}} = c\frac{d^3f(x)}{dx^3} \tag{A.2.2a}$$

where c is a constant.

Table A.2.1. Partial output from `ux_poly_1` of Listing A.2.1

```
ndss =  2   order =  2

i =   1  ux(i) =  0.00000  uxa(i) =  0.00000  err(i) =  0.00000
i =   2  ux(i) =  0.20000  uxa(i) =  0.20000  err(i) =  0.00000
i =   3  ux(i) =  0.40000  uxa(i) =  0.40000  err(i) =  0.00000
i =   4  ux(i) =  0.60000  uxa(i) =  0.60000  err(i) =  0.00000
i =   5  ux(i) =  0.80000  uxa(i) =  0.80000  err(i) = -0.00000
i =   6  ux(i) =  1.00000  uxa(i) =  1.00000  err(i) =  0.00000
i =   7  ux(i) =  1.20000  uxa(i) =  1.20000  err(i) =  0.00000
i =   8  ux(i) =  1.40000  uxa(i) =  1.40000  err(i) =  0.00000
i =   9  ux(i) =  1.60000  uxa(i) =  1.60000  err(i) = -0.00000
i =  10  ux(i) =  1.80000  uxa(i) =  1.80000  err(i) = -0.00000
i =  11  ux(i) =  2.00000  uxa(i) =  2.00000  err(i) = -0.00000

ndss =  2   order =  3

i =   1  ux(i) = -0.02000  uxa(i) =  0.00000  err(i) = -0.02000
i =   2  ux(i) =  0.04000  uxa(i) =  0.03000  err(i) =  0.01000
i =   3  ux(i) =  0.13000  uxa(i) =  0.12000  err(i) =  0.01000
i =   4  ux(i) =  0.28000  uxa(i) =  0.27000  err(i) =  0.01000
i =   5  ux(i) =  0.49000  uxa(i) =  0.48000  err(i) =  0.01000
i =   6  ux(i) =  0.76000  uxa(i) =  0.75000  err(i) =  0.01000
i =   7  ux(i) =  1.09000  uxa(i) =  1.08000  err(i) =  0.01000
i =   8  ux(i) =  1.48000  uxa(i) =  1.47000  err(i) =  0.01000
i =   9  ux(i) =  1.93000  uxa(i) =  1.92000  err(i) =  0.01000
i =  10  ux(i) =  2.44000  uxa(i) =  2.43000  err(i) =  0.01000
i =  11  ux(i) =  2.98000  uxa(i) =  3.00000  err(i) = -0.02000

ndss =  2   order =  4

i =   1  ux(i) = -0.00600  uxa(i) =  0.00000  err(i) = -0.00600
i =   2  ux(i) =  0.00800  uxa(i) =  0.00400  err(i) =  0.00400
i =   3  ux(i) =  0.04000  uxa(i) =  0.03200  err(i) =  0.00800
i =   4  ux(i) =  0.12000  uxa(i) =  0.10800  err(i) =  0.01200
i =   5  ux(i) =  0.27200  uxa(i) =  0.25600  err(i) =  0.01600
i =   6  ux(i) =  0.52000  uxa(i) =  0.50000  err(i) =  0.02000
i =   7  ux(i) =  0.88800  uxa(i) =  0.86400  err(i) =  0.02400
i =   8  ux(i) =  1.40000  uxa(i) =  1.37200  err(i) =  0.02800
i =   9  ux(i) =  2.08000  uxa(i) =  2.04800  err(i) =  0.03200
i =  10  ux(i) =  2.95200  uxa(i) =  2.91600  err(i) =  0.03600
i =  11  ux(i) =  3.92600  uxa(i) =  4.00000  err(i) = -0.07400

           .                                        .
           .                                        .
           .                                        .
```

```
                Output for ndss = 4 to 8 removed

            .                                           .
            .                                           .
            .                                           .
ndss = 10   order = 10

i =   1  ux(i) = -0.00000  uxa(i) =  0.00000  err(i) = -0.00000
i =   2  ux(i) =  0.00000  uxa(i) =  0.00000  err(i) =  0.00000
i =   3  ux(i) =  0.00001  uxa(i) =  0.00001  err(i) =  0.00000
i =   4  ux(i) =  0.00020  uxa(i) =  0.00020  err(i) =  0.00000
i =   5  ux(i) =  0.00262  uxa(i) =  0.00262  err(i) = -0.00000
i =   6  ux(i) =  0.01953  uxa(i) =  0.01953  err(i) =  0.00000
i =   7  ux(i) =  0.10078  uxa(i) =  0.10078  err(i) = -0.00000
i =   8  ux(i) =  0.40354  uxa(i) =  0.40354  err(i) = -0.00000
i =   9  ux(i) =  1.34218  uxa(i) =  1.34218  err(i) =  0.00000
i =  10  ux(i) =  3.87420  uxa(i) =  3.87420  err(i) = -0.00000
i =  11  ux(i) = 10.00000  uxa(i) = 10.00000  err(i) =  0.00000

ndss = 10   order = 11

i =   1  ux(i) = -0.00036  uxa(i) =  0.00000  err(i) = -0.00036
i =   2  ux(i) =  0.00004  uxa(i) =  0.00000  err(i) =  0.00004
i =   3  ux(i) = -0.00001  uxa(i) =  0.00000  err(i) = -0.00001
i =   4  ux(i) =  0.00007  uxa(i) =  0.00006  err(i) =  0.00000
i =   5  ux(i) =  0.00115  uxa(i) =  0.00115  err(i) = -0.00000
i =   6  ux(i) =  0.01074  uxa(i) =  0.01074  err(i) =  0.00000
i =   7  ux(i) =  0.06651  uxa(i) =  0.06651  err(i) = -0.00000
i =   8  ux(i) =  0.31073  uxa(i) =  0.31072  err(i) =  0.00000
i =   9  ux(i) =  1.18111  uxa(i) =  1.18112  err(i) = -0.00001
i =  10  ux(i) =  3.83550  uxa(i) =  3.83546  err(i) =  0.00004
i =  11  ux(i) = 10.99964  uxa(i) = 11.00000  err(i) = -0.00036

ndss = 10   order = 12

i =   1  ux(i) = -0.00200  uxa(i) =  0.00000  err(i) = -0.00200
i =   2  ux(i) =  0.00020  uxa(i) =  0.00000  err(i) =  0.00020
i =   3  ux(i) = -0.00005  uxa(i) =  0.00000  err(i) = -0.00005
i =   4  ux(i) =  0.00004  uxa(i) =  0.00002  err(i) =  0.00002
i =   5  ux(i) =  0.00049  uxa(i) =  0.00050  err(i) = -0.00001
i =   6  ux(i) =  0.00587  uxa(i) =  0.00586  err(i) =  0.00001
i =   7  ux(i) =  0.04353  uxa(i) =  0.04354  err(i) = -0.00001
i =   8  ux(i) =  0.23730  uxa(i) =  0.23728  err(i) =  0.00002
i =   9  ux(i) =  1.03074  uxa(i) =  1.03079  err(i) = -0.00005
i =  10  ux(i) =  3.76596  uxa(i) =  3.76573  err(i) =  0.00023
i =  11  ux(i) = 11.99764  uxa(i) = 12.00000  err(i) = -0.00236
```

An alternate form of the principal error term of Eq. (A.2.2a) is

$$\texttt{err} = c\Delta x^p \tag{A.2.2b}$$

where Δx is the grid spacing in x (Note that the constant c is different in each case).

```
ndss =  2   order =  3

 i =  1  ux(i) = -0.02000 uxa(i) = 0.00000  err(i) = -0.02000
 i =  2  ux(i) =  0.04000 uxa(i) = 0.03000  err(i) =  0.01000
 i =  3  ux(i) =  0.13000 uxa(i) = 0.12000  err(i) =  0.01000
 i =  4  ux(i) =  0.28000 uxa(i) = 0.27000  err(i) =  0.01000
 i =  5  ux(i) =  0.49000 uxa(i) = 0.48000  err(i) =  0.01000
 i =  6  ux(i) =  0.76000 uxa(i) = 0.75000  err(i) =  0.01000
 i =  7  ux(i) =  1.09000 uxa(i) = 1.08000  err(i) =  0.01000
 i =  8  ux(i) =  1.48000 uxa(i) = 1.47000  err(i) =  0.01000
 i =  9  ux(i) =  1.93000 uxa(i) = 1.92000  err(i) =  0.01000
 i = 10  ux(i) =  2.44000 uxa(i) = 2.43000  err(i) =  0.01000
 i = 11  ux(i) =  2.98000 uxa(i) = 3.00000  err(i) = -0.02000
```

Thus, for a third-order polynomial, $d^3f(x)/dx^3$ is constant. Note, however, that c in Eqs. (A.2.2a) and (A.2.2b) at the boundaries ($x = xl = \texttt{x(1)}$, $xu = \texttt{x(11)}$) or `-0.02000` is twice the value at the interior points, `0.01000`. This is usually the case. That is, although the order of the FD is the same throughout x, the *multiplying constant c in Eqs. (A.2.2a) and (A.2.2b) is larger at the boundaries than at the interior points*.

3. For `ndss = 2, norder = 4`, the numerical (FD) derivative is not exact and increases with x as expected from Eq. (A.2.2a) (since for the fourth-order polynomial, the derivative $d^3f(x)/dx^3$ increases with x).

```
ndss =  2   order =  4

 i =  1  ux(i) = -0.00600 uxa(i) = 0.00000  err(i) = -0.00600
 i =  2  ux(i) =  0.00800 uxa(i) = 0.00400  err(i) =  0.00400
 i =  3  ux(i) =  0.04000 uxa(i) = 0.03200  err(i) =  0.00800
 i =  4  ux(i) =  0.12000 uxa(i) = 0.10800  err(i) =  0.01200
 i =  5  ux(i) =  0.27200 uxa(i) = 0.25600  err(i) =  0.01600
 i =  6  ux(i) =  0.52000 uxa(i) = 0.50000  err(i) =  0.02000
 i =  7  ux(i) =  0.88800 uxa(i) = 0.86400  err(i) =  0.02400
 i =  8  ux(i) =  1.40000 uxa(i) = 1.37200  err(i) =  0.02800
 i =  9  ux(i) =  2.08000 uxa(i) = 2.04800  err(i) =  0.03200
 i = 10  ux(i) =  2.95200 uxa(i) = 2.91600  err(i) =  0.03600
 i = 11  ux(i) =  3.92600 uxa(i) = 4.00000  err(i) = -0.07400
```

4. The same general results are observed for a tenth-order FD (`ndss = 10`) applied to tenth- to twelfth-order polynomials. For the tenth-order polynomial, the numerical derivatives are exact.

```
ndss = 10   order = 10

 i =  1  ux(i) = -0.00000 uxa(i) =  0.00000 err(i) = -0.00000
 i =  2  ux(i) =  0.00000 uxa(i) =  0.00000 err(i) =  0.00000
 i =  3  ux(i) =  0.00001 uxa(i) =  0.00001 err(i) =  0.00000
 i =  4  ux(i) =  0.00020 uxa(i) =  0.00020 err(i) =  0.00000
 i =  5  ux(i) =  0.00262 uxa(i) =  0.00262 err(i) = -0.00000
 i =  6  ux(i) =  0.01953 uxa(i) =  0.01953 err(i) =  0.00000
 i =  7  ux(i) =  0.10078 uxa(i) =  0.10078 err(i) = -0.00000
 i =  8  ux(i) =  0.40354 uxa(i) =  0.40354 err(i) = -0.00000
 i =  9  ux(i) =  1.34218 uxa(i) =  1.34218 err(i) =  0.00000
 i = 10  ux(i) =  3.87420 uxa(i) =  3.87420 err(i) = -0.00000
 i = 11  ux(i) = 10.00000 uxa(i) = 10.00000 err(i) =  0.00000
```

5. For `ndss = 10, norder = 11`, the numerical (FD) derivatives are not exact. The difference between the exact and numerical derivatives is a constant that is given by the principal error term (with $p = 10$).

```
ndss = 10   order = 11

 i =  1  ux(i) = -0.00036 uxa(i) =  0.00000 err(i) = -0.00036
 i =  2  ux(i) =  0.00004 uxa(i) =  0.00000 err(i) =  0.00004
 i =  3  ux(i) = -0.00001 uxa(i) =  0.00000 err(i) = -0.00001
 i =  4  ux(i) =  0.00007 uxa(i) =  0.00006 err(i) =  0.00000
 i =  5  ux(i) =  0.00115 uxa(i) =  0.00115 err(i) = -0.00000
 i =  6  ux(i) =  0.01074 uxa(i) =  0.01074 err(i) =  0.00000
 i =  7  ux(i) =  0.06651 uxa(i) =  0.06651 err(i) = -0.00000
 i =  8  ux(i) =  0.31073 uxa(i) =  0.31072 err(i) =  0.00000
 i =  9  ux(i) =  1.18111 uxa(i) =  1.18112 err(i) = -0.00001
 i = 10  ux(i) =  3.83550 uxa(i) =  3.83546 err(i) =  0.00004
 i = 11  ux(i) = 10.99964 uxa(i) = 11.00000 err(i) = -0.00036
```

   As before, the numerical derivatives are less accurate near the boundaries in x (because of the variation in c in Eqs. (A.2.2a) and (A.2.2b) with x, but the FD approximations are tenth order throughout x).
6. For `ndss = 10, norder = 12`, the errors are greater than those for the eleventh-order polynomial.

```
ndss = 10   order = 12

 i =  1  ux(i) = -0.00200 uxa(i) =  0.00000 err(i) = -0.00200
 i =  2  ux(i) =  0.00020 uxa(i) =  0.00000 err(i) =  0.00020
 i =  3  ux(i) = -0.00005 uxa(i) =  0.00000 err(i) = -0.00005
 i =  4  ux(i) =  0.00004 uxa(i) =  0.00002 err(i) =  0.00002
 i =  5  ux(i) =  0.00049 uxa(i) =  0.00050 err(i) = -0.00001
 i =  6  ux(i) =  0.00587 uxa(i) =  0.00586 err(i) =  0.00001
 i =  7  ux(i) =  0.04353 uxa(i) =  0.04354 err(i) = -0.00001
 i =  8  ux(i) =  0.23730 uxa(i) =  0.23728 err(i) =  0.00002
 i =  9  ux(i) =  1.03074 uxa(i) =  1.03079 err(i) = -0.00005
 i = 10  ux(i) =  3.76596 uxa(i) =  3.76573 err(i) =  0.00023
 i = 11  ux(i) = 11.99764 uxa(i) = 12.00000 err(i) = -0.00236
```

The preceding general conclusions (mainly, a pth-order FD differentiates a pth-order polynomial exactly) do not necessarily apply to nonpolynomial functions. For example, if we consider an exponential function, e^{ax}, the FD numerical derivatives will in general not be exact. One way to anticipate this result is to consider the exponential function as an infinite-order polynomial expressed as its Taylor series expansion. In fact, the principal error term of Eq. (A.2.2a) is the first term of a Taylor series expansion beyond the point of truncation of the series that gives the FD approximation; thus, this error term is called the *truncation error* of the FD.

These conclusions are illustrated with the test program given in Listing A.2.2, which is a minor variation of `ux_poly_1` in Listing A.2.1.

```
%
  clear all
  clc
%
% Varying order of differentiator
  for ndss=2:2:10
%
% Grid in x
  xl=0.0;
  xu=1.0;
  n=11;
  dx=(xu-xl)/(n-1);
%
% Test exponential
  a=1.0;
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u(i)=exp(a*x(i));
  end
```

```
%
% ux
  if ndss==2   ux=dss002(xl,xu,n,u); end
  if ndss==4   ux=dss004(xl,xu,n,u); end
  if ndss==6   ux=dss006(xl,xu,n,u); end
  if ndss==8   ux=dss008(xl,xu,n,u); end
  if ndss==10  ux=dss010(xl,xu,n,u); end
%
% Comparison of numerical and analytical ux
  fprintf('\n\n ndss = %2d   a = %4.2f\n\n',ndss,a);
  for i=1:n
    uxa(i)=a*exp(x(i));
    err(i)=ux(i)-uxa(i);
    fprintf(' i = %3d  ux(i) = %8.5f  uxa(i) = %8.5f
              err(i) = %8.5f\n',i,ux(i),uxa(i),err(i));
  end
%
% Next differentiator
  end
```

Listing A.2.2. Program `ux_exp_1` for the numerical differentiation of an exponential function

The only essential difference is the use of `exp(a*x(i))` and its analytical derivative `a*exp(a*x(i))`. A portion of the output from this program is given in Table A.2.2.

We can note the following points about the output given in Table A.2.2:

1. The second-order FDs give a first derivative that is accurate to less than three figures.

```
ndss =  2   a = 1.00

i =  1  ux(i) = 0.99640  uxa(i) = 1.00000  err(i) = -0.00360
i =  2  ux(i) = 1.10701  uxa(i) = 1.10517  err(i) =  0.00184
i =  3  ux(i) = 1.22344  uxa(i) = 1.22140  err(i) =  0.00204
i =  4  ux(i) = 1.35211  uxa(i) = 1.34986  err(i) =  0.00225
i =  5  ux(i) = 1.49431  uxa(i) = 1.49182  err(i) =  0.00249
i =  6  ux(i) = 1.65147  uxa(i) = 1.64872  err(i) =  0.00275
i =  7  ux(i) = 1.82516  uxa(i) = 1.82212  err(i) =  0.00304
i =  8  ux(i) = 2.01711  uxa(i) = 2.01375  err(i) =  0.00336
i =  9  ux(i) = 2.22925  uxa(i) = 2.22554  err(i) =  0.00371
i = 10  ux(i) = 2.46370  uxa(i) = 2.45960  err(i) =  0.00410
i = 11  ux(i) = 2.70987  uxa(i) = 2.71828  err(i) = -0.00841
```

Table A.2.2. Partial output from `ux_exp_1` of Listing A.2.2

```
ndss =  2   a = 1.00

i =   1  ux(i) =  0.99640  uxa(i) =  1.00000  err(i) = -0.00360
i =   2  ux(i) =  1.10701  uxa(i) =  1.10517  err(i) =  0.00184
i =   3  ux(i) =  1.22344  uxa(i) =  1.22140  err(i) =  0.00204
i =   4  ux(i) =  1.35211  uxa(i) =  1.34986  err(i) =  0.00225
i =   5  ux(i) =  1.49431  uxa(i) =  1.49182  err(i) =  0.00249
i =   6  ux(i) =  1.65147  uxa(i) =  1.64872  err(i) =  0.00275
i =   7  ux(i) =  1.82516  uxa(i) =  1.82212  err(i) =  0.00304
i =   8  ux(i) =  2.01711  uxa(i) =  2.01375  err(i) =  0.00336
i =   9  ux(i) =  2.22925  uxa(i) =  2.22554  err(i) =  0.00371
i =  10  ux(i) =  2.46370  uxa(i) =  2.45960  err(i) =  0.00410
i =  11  ux(i) =  2.70987  uxa(i) =  2.71828  err(i) = -0.00841

ndss =  4   a = 1.00

i =   1  ux(i) =  0.99998  uxa(i) =  1.00000  err(i) = -0.00002
i =   2  ux(i) =  1.10518  uxa(i) =  1.10517  err(i) =  0.00001
i =   3  ux(i) =  1.22140  uxa(i) =  1.22140  err(i) = -0.00000
i =   4  ux(i) =  1.34985  uxa(i) =  1.34986  err(i) = -0.00000
i =   5  ux(i) =  1.49182  uxa(i) =  1.49182  err(i) = -0.00000
i =   6  ux(i) =  1.64872  uxa(i) =  1.64872  err(i) = -0.00001
i =   7  ux(i) =  1.82211  uxa(i) =  1.82212  err(i) = -0.00001
i =   8  ux(i) =  2.01375  uxa(i) =  2.01375  err(i) = -0.00001
i =   9  ux(i) =  2.22553  uxa(i) =  2.22554  err(i) = -0.00001
i =  10  ux(i) =  2.45961  uxa(i) =  2.45960  err(i) =  0.00001
i =  11  ux(i) =  2.71824  uxa(i) =  2.71828  err(i) = -0.00005

           .                                          .
           .                                          .
           .                                          .

                Output for ndss = 6 and 8 removed

           .                                          .
           .                                          .
           .                                          .

ndss = 10   a = 1.00

i =   1  ux(i) =  1.00000  uxa(i) =  1.00000  err(i) = -0.00000
i =   2  ux(i) =  1.10517  uxa(i) =  1.10517  err(i) =  0.00000
i =   3  ux(i) =  1.22140  uxa(i) =  1.22140  err(i) = -0.00000
i =   4  ux(i) =  1.34986  uxa(i) =  1.34986  err(i) =  0.00000
i =   5  ux(i) =  1.49182  uxa(i) =  1.49182  err(i) = -0.00000
i =   6  ux(i) =  1.64872  uxa(i) =  1.64872  err(i) =  0.00000
i =   7  ux(i) =  1.82212  uxa(i) =  1.82212  err(i) = -0.00000
i =   8  ux(i) =  2.01375  uxa(i) =  2.01375  err(i) =  0.00000
i =   9  ux(i) =  2.22554  uxa(i) =  2.22554  err(i) = -0.00000
i =  10  ux(i) =  2.45960  uxa(i) =  2.45960  err(i) =  0.00000
i =  11  ux(i) =  2.71828  uxa(i) =  2.71828  err(i) = -0.00000
```

2. The fourth-order FDs give a substantial improvement in the accuracy of the derivative.

```
ndss =  4   a = 1.00

i =  1  ux(i) = 0.99998  uxa(i) = 1.00000  err(i) = -0.00002
i =  2  ux(i) = 1.10518  uxa(i) = 1.10517  err(i) =  0.00001
i =  3  ux(i) = 1.22140  uxa(i) = 1.22140  err(i) = -0.00000
i =  4  ux(i) = 1.34985  uxa(i) = 1.34986  err(i) = -0.00000
i =  5  ux(i) = 1.49182  uxa(i) = 1.49182  err(i) = -0.00000
i =  6  ux(i) = 1.64872  uxa(i) = 1.64872  err(i) = -0.00001
i =  7  ux(i) = 1.82211  uxa(i) = 1.82212  err(i) = -0.00001
i =  8  ux(i) = 2.01375  uxa(i) = 2.01375  err(i) = -0.00001
i =  9  ux(i) = 2.22553  uxa(i) = 2.22554  err(i) = -0.00001
i = 10  ux(i) = 2.45961  uxa(i) = 2.45960  err(i) =  0.00001
i = 11  ux(i) = 2.71824  uxa(i) = 2.71828  err(i) = -0.00005
```

3. The tenth-order FDs appear to give an exact first derivative. However, the output is deceptive and this conclusion is not correct.

```
ndss = 10   a = 1.00

i =  1  ux(i) = 1.00000  uxa(i) = 1.00000  err(i) = -0.00000
i =  2  ux(i) = 1.10517  uxa(i) = 1.10517  err(i) =  0.00000
i =  3  ux(i) = 1.22140  uxa(i) = 1.22140  err(i) = -0.00000
i =  4  ux(i) = 1.34986  uxa(i) = 1.34986  err(i) =  0.00000
i =  5  ux(i) = 1.49182  uxa(i) = 1.49182  err(i) = -0.00000
i =  6  ux(i) = 1.64872  uxa(i) = 1.64872  err(i) =  0.00000
i =  7  ux(i) = 1.82212  uxa(i) = 1.82212  err(i) = -0.00000
i =  8  ux(i) = 2.01375  uxa(i) = 2.01375  err(i) =  0.00000
i =  9  ux(i) = 2.22554  uxa(i) = 2.22554  err(i) = -0.00000
i = 10  ux(i) = 2.45960  uxa(i) = 2.45960  err(i) =  0.00000
i = 11  ux(i) = 2.71828  uxa(i) = 2.71828  err(i) = -0.00000
```

Rather, the errors in the numerical derivative are not displayed by the `%8.5f` format. This can easily be demonstrated by increasing `a` (i.e., $a > 1.0$) so that the exponential has greater curvature and therefore the numerical differentiation gives larger errors.

Since exponential functions are frequently solutions to ODE/PDE systems, the preceding discussion implies that in general we cannot expect the numerical solutions to be exact. In fact, numerical solutions computed with approximations such as FDs will generally not be exact since the

approximations will have a *nonzero truncation error* (as expressed by the principal error term, such as in Eq. (A.2.2a) for polynomials, which defines the apparent order of the approximation).

This completes the discussion of numerical first derivatives. We next consider second derivatives, again with polynomial and exponential test functions. A program for the second-order derivative of a polynomial with Dirichlet boundary conditions (BCs) computed by FDs of orders 2–10 is given in Listing A.2.3.

```
%
  clear all
  clc
%
% Varying order of differentiator
  for ndss=42:2:50
%
% Varying order of test polynomial
  for norder=1:12
%
% Grid in x
  xl=0.0;
  xu=1.0;
  n=12;
  dx=(xu-xl)/(n-1);
%
% Test polynomial
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u(i)=x(i)^norder;
  end
%
% uxx, Dirichlet BC
  nl=1;
  nu=1;
  ux=zeros(1,n);
  if ndss==42  uxx=dss042(xl,xu,n,u,ux,nl,nu); end
  if ndss==44  uxx=dss044(xl,xu,n,u,ux,nl,nu); end
  if ndss==46  uxx=dss046(xl,xu,n,u,ux,nl,nu); end
  if ndss==48  uxx=dss048(xl,xu,n,u,ux,nl,nu); end
  if ndss==50  uxx=dss050(xl,xu,n,u,ux,nl,nu); end
%
% Comparison of numerical and analytical uxx
  if(norder-(ndss-40))>-1 & (norder-(ndss-40))< 3
  fprintf('\n\n ndss = %2d   nl = 1   nu = 1
         order = %2d\n\n',ndss,norder);
  for i=1:n
```

```
    uxxa(i)=norder*(norder-1)*x(i)^(norder-2);
    err(i)=uxx(i)-uxxa(i);
    fprintf(' i = %3d  uxx(i) = %8.5f  uxxa(i) = %8.5f
             err(i) = %8.5f\n',i,uxx(i),uxxa(i),err(i));
  end
  end
%
% Next order
  end
%
% Next differentiator
  end
```

Listing A.2.3. Program `uxx_poly_1` for the numerical second derivative of polynomials

We can note the following points about this program:

1. As with the two preceding programs, two `for` loops cycle through FD routines for second derivatives, `ndss = 42` to `50` corresponding to `dss042` to `dss050`, for polynomials of order 1–12. In this case, 12 grid points are used because the tenth-order FDs of `dss050` use 12 grid points.

```
%
  clear all
  clc
%
% Varying order of differentiator
  for ndss=42:2:50
%
% Varying order of test polynomial
  for norder=1:12
%
% Grid in x
  xl=0.0;
  xu=1.0;
  n=12;
  dx=(xu-xl)/(n-1);
```

2. The polynomial test function in u is differentiated by the five FD routines for Dirichlet BCs (`nl = nu = 1`). The call to `zeros` is used to assign values to the first derivative `ux` that is an input argument to the FD routines. In the case of Dirichlet BCs, this first derivative is not actually used by the FD routines, but Matlab requires that all input arguments for a function must have at least one assigned value.

```
%
% Test polynomial
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u(i)=x(i)^norder;
  end
%
% uxx, Dirichlet BC
  nl=1;
  nu=1;
  ux=zeros(1,n);
  if ndss==42  uxx=dss042(xl,xu,n,u,ux,nl,nu); end
  if ndss==44  uxx=dss044(xl,xu,n,u,ux,nl,nu); end
  if ndss==46  uxx=dss046(xl,xu,n,u,ux,nl,nu); end
  if ndss==48  uxx=dss048(xl,xu,n,u,ux,nl,nu); end
  if ndss==50  uxx=dss050(xl,xu,n,u,ux,nl,nu); end
```

3. The difference between the analytical and numerical derivatives is computed and the numerical output is displayed.

```
%
% Comparison of numerical and analytical uxx
  if(norder-(ndss-40))>-1 & (norder-(ndss-40))< 3
  fprintf('\n\n ndss = %2d   nl = 1   nu = 1
          order = %2d\n\n',ndss,norder);
  for i=1:n
    uxxa(i)=norder*(norder-1)*x(i)^(norder-2);
    err(i)=uxx(i)-uxxa(i);
    fprintf(' i = %3d  uxx(i) = %8.5f  uxxa(i) = %8.5f
            err(i) = %8.5f\n',i,uxx(i),uxxa(i),err(i));
  end
  end
```

The output from this test program has the same essential properties as in the case of the first derivative: (a) the pth-order FD is exact for pth-order polynomials, (b) the error in the $(p+1)$th-order numerical derivative is constant (except for the multiplying constant as in Eqs. (A.2.2a) and (A.2.2b) that increases near the boundaries in x), and (c) the error in the $(p+2)$th numerical derivative is not constant. Therefore, because of the similarity with the output in Table A.2.1, we will not list this output here.

A program for the numerical second derivative with Neumann BCs can be constructed directly from the program of Listing A.2.3. The only required difference is the programming of the BCs.

```
%
% uxx, Neumann BC
  nl=2;
  nu=2;
  ux(1)=norder*x(1)^(norder-1);
  ux(n)=norder*x(n)^(norder-1);
  if ndss==42  uxx=dss042(xl,xu,n,u,ux,nl,nu); end
  if ndss==44  uxx=dss044(xl,xu,n,u,ux,nl,nu); end
  if ndss==46  uxx=dss046(xl,xu,n,u,ux,nl,nu); end
  if ndss==48  uxx=dss048(xl,xu,n,u,ux,nl,nu); end
  if ndss==50  uxx=dss050(xl,xu,n,u,ux,nl,nu); end
```

Here we have used the analytical derivatives of the polynomial at the boundaries, `ux(1), ux(n)`. Again the output has the same essential properties as before and is therefore not listed here.

Test programs for the numerical second derivative of e^{ax} for Dirichlet and Neumann BCs can easily be constructed from the preceding programs (they are not discussed here to save some space). The general conclusions for the output are the same as before: (a) the FDs are not exact, (b) the error decreases with increasing order of the FDs, and (c) the error increases with increasing curvature of e^{ax} (from larger a).

In summary, our intention in presenting this appendix is to demonstrate the order conditions for the FDs in the DSS routines through application of the routines to some test problems. Basically, we have considered *p-refinement* (for the improvement of numerical derivative accuracy by increasing the order p of the FD approximations). In general, we could also improve the accuracy of the numerical derivatives by increasing the number of grid points. This approach is suggested by Eq. (A.2.2b); if the number of grid points is increased, the grid spacing Δx is decreased (and thus, the truncation error is decreased). Since the grid spacing is frequently given the symbol h in the numerical analysis literature, this procedure of changing the grid spacing is termed *h-refinement*.

Finally, the accuracy of the numerical derivatives is dependent on the properties of the function that is differentiated. Polynomials can be differentiated exactly by FDs. Nonpolynomial functions generally cannot (and FDs are not recommended for discontinuous functions).

APPENDIX 3

Analytical Solution of Nonlinear, Traveling Wave Partial Differential Equations

Partial differential equations (PDEs) that are nonlinear and have traveling wave solutions (to be explained) are of wide interest for their mathematical properties and their application to a broad spectrum of physical systems. We have used the traveling wave analytical solution of the Korteweg-de Vries (KdV) equation [1] in Chapter 7. Here we consider the analytical solution of a class of nonlinear, traveling wave PDEs whose analytical solutions can provide useful test problems for numerical PDE methods.

The starting point for the analysis is the following nonlinear, third-order PDE, which is a minor extension of the KdV equation with a coefficient a multiplying the *nonlinear convective* term, uu_x, and a coefficient b multiplying the *linear, dispersive* term, u_{xxx} (here we use subscript notation for partial derivatives, e.g., $\partial u/\partial t = u_t$).

$$u_t + auu_x + bu_{xxx} = 0 \tag{A.3.1}$$

The analytical method that follows is an extension of the method given by Knobel [2]. We look for a *traveling wave solution* of the form $u(x, t) = f(z), z = x - ct$, where $c > 0$, and $f(z), f'(z)$, and $f''(z)$ tend to 0 as $z \to \pm\infty$ ($'$ denotes differentiation with respect to z).

Substituting $u(x, t) = f(x - ct)$ into Eq. (A.3.1) gives a third-order nonlinear ordinary differential equation (ODE) for $f(z)$:

$$-cf' + aff' + bf''' = 0 \tag{A.3.2}$$

In arriving at Eq. (A.3.2), we have used

$$\frac{\partial z}{\partial t} = \frac{\partial(x - ct)}{\partial t} = -c$$

$$\frac{\partial z}{\partial x} = \frac{\partial(x - ct)}{\partial t} = 1$$

$$u_t = \frac{\partial f(x - ct)}{\partial t} = \frac{df(z)}{dz}\frac{\partial z}{\partial t} = -cf'$$

$$u_x = \frac{\partial f(x-ct)}{\partial x} = \frac{df(z)}{dz}\frac{\partial z}{\partial x} = f'$$

$$u_{xx} = \frac{\partial^2 f(x-ct)}{\partial x^2} = \frac{\partial\left[\dfrac{\partial f(x-ct)}{\partial x}\right]}{\partial x} = \frac{df'}{dz}\frac{\partial z}{\partial x} = f''$$

$$u_{xxx} = \frac{\partial^3 f(x-ct)}{\partial x^3} = \frac{\partial\left[\dfrac{\partial^2 f(x-ct)}{\partial x^2}\right]}{\partial x} = \frac{df''}{dz}\frac{\partial z}{\partial x} = f'''$$

Equation (A.3.2) can be integrated once to give

$$-cf + \frac{1}{2}af^2 + bf'' = c_1 \tag{A.3.3}$$

where c_1 is a constant of integration. From the assumption that $f(z)$ and $f'' \to 0$ as $z \to \infty$, the value of c_1 is zero.

Multiplying Eq. (A.3.3) by f'

$$-cff' + \frac{1}{2}af^2f' + bf'f'' = 0 \tag{A.3.4}$$

and integrating Eq. (A.3.4) results in the first-order equation

$$-\frac{1}{2}cf^2 + \frac{1}{6}af^3 + \frac{1}{2}b(f')^2 = c_2 \tag{A.3.5}$$

Since $f(z), f'(z) \to 0$ as $z \to \infty$, the constant of integration c_2 is zero. Solving for $(f')^2$ gives

$$3b(f')^2 = (3c - af)f^2 \tag{A.3.6}$$

From here we will require $0 < f(z) < 3c/a$ in order to have a positive RHS in Eq. (A.3.6). Taking the positive square root of Eq. (A.3.6) gives

$$\frac{\sqrt{3b}}{\sqrt{3c - af}\,f}f' = 1 \tag{A.3.7}$$

To integrate the LHS of Eq. (A.3.7), we make the rationalizing substitution $g^2 = 3c - af$; substituting $f = 3c/a - g^2/a$ and $f' = -(2/a)gg'$ gives

$$\frac{\sqrt{3b}}{g\,(3c/a - g^2/a)}\left[-(2/a)gg'\right] = 1$$

or

$$\frac{2\sqrt{3b}}{3c - g^2}g' = -1 \tag{A.3.8}$$

By the method of partial fractions, we have

$$\frac{1}{3c - g^2} = \frac{A}{\sqrt{3c} - g} + \frac{B}{\sqrt{3c} + g} = \frac{A(\sqrt{3c} + g) + B(\sqrt{3c} - g)}{3c - g^2}$$

and therefore (equating numerators)

$$A\left(\sqrt{3c}+g\right)+B\left(\sqrt{3c}-g\right)=1$$

or

$$A=B=\frac{1}{2\sqrt{3c}}$$

Then, from Eq. (A.3.8),

$$\frac{2\sqrt{3b}}{3c-g^2}g'=\frac{\sqrt{b/c}}{\sqrt{3c}-g}g'+\frac{\sqrt{b/c}}{\sqrt{3c}+g}g'=-1 \tag{A.3.9}$$

Integration of both sides of Eq. (A.3.9) with respect to z gives

$$\ln(\sqrt{3c}+g)-\ln(\sqrt{3c}-g)=-\sqrt{c/b}\,z+d$$

or

$$\ln\left(\frac{\sqrt{3c}+g}{\sqrt{3c}-g}\right)=-\sqrt{c/b}\,z+d \tag{A.3.10}$$

with a constant of integration d. Solving Eq. (A.3.10) for g, we have

$$\frac{\sqrt{3c}+g}{\sqrt{3c}-g}=e^{-\sqrt{c/b}z+d}$$

$$\sqrt{3c}+g=\left(\sqrt{3c}-g\right)e^{-\sqrt{c/b}z+d}$$

$$g(1+e^{-\sqrt{c/b}z+d})=\sqrt{3c}(e^{-\sqrt{c/b}z+d}-1)$$

$$g(z)=\sqrt{3c}\frac{\exp(-\sqrt{c/b}z+d)-1}{\exp(-\sqrt{c/b}z+d)+1} \tag{A.3.11}$$

Since

$$\tanh(x)=\frac{\sinh(x)}{\cosh(x)}=\frac{\dfrac{e^x-e^{-x}}{2}}{\dfrac{e^x+e^{-x}}{2}}=\frac{e^0-e^{-2x}}{e^0+e^{-2x}}=-\frac{e^{-2x}-1}{e^{-2x}+1}$$

Eq. (A.3.11) can be written as

$$g(z)=\sqrt{3c}\frac{\exp(-\sqrt{c/b}z+d)-1}{\exp(-\sqrt{c/b}z+d)+1}=-\sqrt{3c}\,\tanh\left[\frac{1}{2}(\sqrt{c/b}z-d)\right] \tag{A.3.12}$$

Then from $f=3c/a-g^2/a$, we obtain

$$af=3c-\left\{-\sqrt{3c}\,\tanh\left[\frac{1}{2}(\sqrt{c/b}z-d)\right]\right\}^2$$

$$af=3c-3c\,\tanh^2\left[\frac{1}{2}(\sqrt{c/b}z-d)\right]$$

$$af = 3c\frac{\cosh^2\left[\frac{1}{2}(\sqrt{c/b}z - d)\right] - \sinh^2\left[\frac{1}{2}(\sqrt{c/b}z - d)\right]}{\cosh^2\left[\frac{1}{2}(\sqrt{c/b}z - d)\right]} \tag{A.3.13}$$

Since

$$\cosh^2(x) - \sinh^2(x) = \left(\frac{e^x + e^{-x}}{2}\right)^2 - \left(\frac{e^x - e^{-x}}{2}\right)^2$$

$$= \frac{e^{2x} + 2e^x e^{-x} + e^{-2x} - e^{2x} + 2e^x e^{-x} - e^{-2x}}{4} = 1$$

Eq. (A.3.13) reduces to

$$f(z) = \frac{3c}{a}\,\mathrm{sech}^2\left[\frac{1}{2}(\sqrt{c/b}z - d)\right] \tag{A.3.14}$$

Since the arbitrary constant d is simply a shift of the sech, it does not have a substantial effect on the solution and we therefore take $d = 0$.

$$u(x,t) = f(z) = \frac{3c}{a}\,\mathrm{sech}^2\left[\frac{1}{2}\sqrt{\frac{c}{b}}z\right] = \frac{3c}{a}\,\mathrm{sech}^2\left[\frac{1}{2}\sqrt{\frac{c}{b}}(x - ct)\right] \tag{A.3.15}$$

We can get an idea of what this traveling wave looks like as follows. The traveling wave solution of the KdV equation (see also Chapter 7) is

$$u(x,t) = f(z) = 3c\,\mathrm{sech}^2\left[\frac{\sqrt{c}}{2}(x - ct)\right] \tag{A.3.16}$$

A comparison of Eqs. (A.3.15) and (A.3.16) indicates the following.

1. The amplitude of the sech soliton of Eq. (A.3.16) is changed from $3c$ to $3c/a$ in Eq. (A.3.15) (thus, the soliton of Eq. (A.3.15) can be scaled vertically).
2. The argument of the soliton of Eq. (A.3.16) is changed from $(\sqrt{c}/2)(x - ct)$ to $(1/2)\sqrt{c/b}(x - ct)$ in Eq. (A.3.15) (thus, the soliton of Eq. (A.3.15) can be made "sharper" using $b < 1$).

This method of solution for Eq. (A.3.1) can be readily extended to other PDEs. For example, we can consider PDEs with mixed partials; the analytical solutions (derived next) provide tests of numerical solutions for this important class of PDEs (with mixed partials).

To start,

$$u_t + auu_x - bu_{xxt} = 0 \tag{A.3.17}$$

is the so-called *equal-width* equation. Again, we look for a traveling wave solution $u(x,t) = f(x - ct)$. Following the previous approach, we arrive at the ODE

$$-cf' + aff' + df''' = 0 \tag{A.3.18}$$

with $d = bc$. Note that this equation is essentially the same as Eq. (A.3.2), so the solution follows immediately from Eq. (A.3.15):

$$u(x,t) = f(z) = \frac{3c}{a}\,\mathrm{sech}^2\left[\frac{\sqrt{1/b}}{2}(x-ct)\right] \tag{A.3.19}$$

Equation (A.3.19) indicates that the *width* of the soliton is independent of c (thus, the name "equal-width" equation). Also, the *Benjamin-Bona-Mahony* equation ([3], p. 583) is a special case of Eq. (A.3.17) for $a = -1$.

For the equation

$$u_t + auu_x + bu_{xtt} = 0 \tag{A.3.20}$$

the solution follows immediately from Eq. (A.3.15)

$$u(x,t) = f(z) = \frac{3c}{a}\mathrm{sech}^2\left[\frac{\sqrt{1/(bc)}}{2}(x-ct)\right] \tag{A.3.21}$$

so that the *width* now increases with increasing c.

For the equation

$$u_t + auu_x - bu_{ttt} = 0 \tag{A.3.22}$$

the solution follows immediately from Eq. (A.3.15)

$$u(x,t) = f(z) = \frac{3c}{a}\mathrm{sech}^2\left[\frac{\sqrt{1/(bc^2)}}{2}(x-ct)\right] \tag{A.3.23}$$

so that again the *width* increases with increasing c. Equations (A.3.19), (A.3.21), and (A.3.23) could be used to test numerical solutions of Eqs. (A.3.17), (A.3.20), and (A.3.22), respectively.

The following variation of Eq. (A.3.17) also has a traveling wave solution ([3], pp. 583–584):

$$u_t + au^k u_x - bu_{xxt} = 0 \tag{A.3.24}$$

We could also consider equations in which x and t are interchanged, for example,

$$u_x + auu_t + bu_{ttt} = 0 \tag{A.3.25}$$

Development of traveling wave solutions based on minor variations of Eq. (A.3.2) and its solution, Eq. (A.3.15), would be possible. Many other PDEs with traveling wave solutions are discussed in [3].

In summary, this discussion of a method for deriving analytical, traveling wave, PDE solutions is intended to illustrate a rather general approach in terms of some specific example PDEs. The essential features of this method are

1. A traveling wave solution of the form $u(x,t) = f(x-ct)$ is assumed that is then substituted in the PDE.
2. A (generally nonlinear) ODE for $f(z)$ results from this substitution that is then integrated with respect to the composite independent variable $z = x - ct$.

3. In the process of integrating the ODE, the assumption that $f(z)$ and its derivatives tend to zero as $|z| \to \infty$ is employed.
4. Basically, this assumption requires that the initial condition for the PDE, $u(x, t = 0) = f(x)$, is nonzero only over a finite interval in x such that the solution and its derivatives remain at zero for large $|z|$. Recall, also, that we require $0 < f(z) < 3c/a$ from Eq. (A.3.6).
5. The generality of the method is limited largely by the integration of the ODE in z. This integration is performed analytically, possibly with the assistance of a symbolic (computer) algebra system.

REFERENCES

[1] Korteweg, D. J. and F. de Vries (1895), On the Change of Form of Long Waves Advancing in a Rectangular Canal, and on a New Type of Long Stationary Waves, *Phil. Mag.*, **39**: 422–443.

[2] Knobel, R. (2000), Korteweg–de Vries Equation, In: *An Introduction to the Mathematical Theory of Waves*, American Mathematical Society, Institute for Advanced Study, Providence, RI Chapter 5, pp. 31–35

[3] Polyanin, A. D. and V. F. Zaitsev (2004), *Handbook of Nonlinear Partial Differential Equations*, Chapman & Hall/CRC, Boca Raton, FL

APPENDIX 4

Implementation of Time-Varying Boundary Conditions

We consider here several methods for the implementation of time-varying boundary conditions (BCs). To illustrate the basic problem, consider the one-dimensional (1D) diffusion equation

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{A.4.1a}$$

Equation (A.4.1a) is first order in t and second order in x. It therefore requires one initial condition (IC) and two boundary conditions (BCs), which are taken as

$$u(x, t = 0) = 0 \tag{A.4.1b}$$

$$u(x = 0, t) = f(t), \tag{A.4.1c}$$

$$\frac{\partial u(x = 1, t)}{\partial x} = 0 \tag{A.4.1d}$$

Equation (A.4.1c) is the time-varying Dirichlet BC; $f(t)$ is a function to be specified.

In considering this problem, it would seem that the implementation of BC (A.4.1c) should be straightforward. For example, in the ordinary differential equation (ODE) routine, since t is an input argument, a statement to implement Eq. (A.4.1c) might be

```
u(1)=f(t);
```

However, most of the Matlab ODE integrators, for example, `ode15s`, do not accept the setting of the dependent variables (such as `u(1)`). Rather, if the dependent variables are defined as in Eq. (A.4.1c), they must be programmed either through a corresponding ODE or by explicit programming of the dependent variables outside of the ODE routine.

These alternatives are illustrated with the Matlab routines given in Listing A.4.1, in which $f(t) = 1 - \cos(a\pi t)$ in Eq. (A.4.1c). First, a main program for Eqs. (A.4.1a)–(A.4.1d) follows:

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with the ODE routine
  global a D n xl xu dx ncall ndss
%
% Parameters
  a=1.0;
  D=0.3;
  xl=0.0;
  xu=1.0;
%
% Cases with the time varying Dirichlet boundary condition
% (TVDBC) in or outside the spatial grid
  mf=5;
%
% mf = 1,2,3; TVDBC not part of the spatial grid
%
%   mf = 1 - explicit programming of the FDs with boundary
%            value calculated for the ODE at x = dx in pde_1.m
%
%   mf = 2 - ux routines dss002 to dss010 with boundary value
%            calculated for MOL ODEs in pde_2.m
%
%   mf = 3 - uxx routines dss042 to dss050 with boundary value
%            calculated for MOL ODEs in pde_3.m
%
%   For mf = 1,2,3, the boundary value must be calculated
%   after the return from the ODE integrator if it is to be
%   displayed and/or plotted.
    if(mf<=3)
      n=20;
      dx=(xu-xl)/n;
    end
%
% mf = 4,5,6,7; TVDBC part of the spatial grid
%
%   mf = 4 - explicit programming of the FDs with boundary
%            value calculated at boundary in pde_4.m; this
%            boundary value is not returned by the ODE
```

```
%            integrator and it must therefore be restored
%            after the return from the integrator if it is
%            to be displayed and/or plotted
%
%   mf = 5 - explicit programming of the FDs with boundary
%            value calculated and derivative of boundary value
%            also calculated (for ODE at boundary) in pde_5.m;
%            the ODE integrator returns the correct boundary
%            value
%
%   mf = 6 - ux routines dss002 to dss010 with boundary value
%            calculated for MOL ODEs in pde_6.m; the ODE
%            integrator returns the correct boundary value
%
%   mf = 7 - uxx routines dss042 to dss050 with boundary value
%            calculated for MOL ODEs in pde_7.m; the ODE
%            integrator returns the correct boundary value
    if(mf>3)
      n=21;
      dx=(xu-xl)/(n-1);
    end
%
% Spatial grid
  x=[xl:dx:xu]';
%
% Initial condition
  u0=zeros(n,1);
%
% Independent variable for ODE integration
  t0=0.0;
  tf=2.5;
  tout=[0.0:0.25:tf]';
  nout=11;
  ncall=0;
%
% ODE integration
  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
  if(mf==4) % explicit FDs
    [t,u]=ode15s(@pde_4,tout,u0,options); end
  if(mf==5) % explicit FDs
```

```
    [t,u]=ode15s(@pde_5,tout,u0,options); end
  if(mf==6) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_6,tout,u0,options); end
  if(mf==7) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
    [t,u]=ode15s(@pde_7,tout,u0,options); end
%
% Display selected output
  if(mf==1)
    n2=n/2;
    fprintf('\n             mf = %2d   n = %3d',mf,n);
  elseif(mf==2|mf==3)
    n2=n/2;
    fprintf('\n      mf = %2d   n = %3d   ndss = %2d', ...
            mf,n,ndss);
  elseif(mf==4|mf==5)
    n2=(n-1)/2+1;
    fprintf('\n             mf = %2d   n = %3d',mf,n);
  elseif(mf==6|mf==7)
    n2=(n-1)/2+1;
    fprintf('\n       mf = %2d    n = %3d   ndss = %2d', ...
            mf,n,ndss);
  end
  fprintf('\n   abstol = %8.1e    reltol = %8.1e', ...
          abstol,reltol);
  if(mf<=3)
    fprintf('\n\n    t   u(x=dx,t)  u(x=0.5,t)   u(x=1,t)\n');
  else
    fprintf('\n\n    t    u(x=0,t)  u(x=0.5,t)   u(x=1,t)\n');
  end
  for it=1:nout
    if(mf==4)u(it,1)=1-cos(a*pi*t(it));end
    fprintf('%6.2f%12.6f%12.6f%12.6f\n',t(it),u(it,1), ...
            u(it,n2),u(it,n));
  end
  fprintf('\n ncall = %4d\n',ncall);
%
% Plot numerical solution as u(x,t) vs x with t as parameter
  if(mf<=3)
    uplot(:,2:n+1)=u(:,1:n);
%
%   Include boundary value in plot
    for it=1:nout
      uplot(it,1)=1-cos(a*pi*t(it));
    end
  elseif(mf>3)
    uplot=u;
%
```

```
%   Restore boundary value for plot (mf = 4)
    if(mf==4)
      for it=1:nout
        uplot(it,1)=u(it,1);
      end
    end
  end
  figure(1);
  plot(x,uplot,'-k'); axis tight
  title('u(x,t) vs x, t = 0,0.25,...,2.5');
  xlabel('x');
  ylabel('u(x,t)')
  figure(2);
  surf(x,tout,uplot);
  shading interp; axis tight
  colormap jet
  title('u(x,t) vs x, t = 0,0.25,...,2.5');
  xlabel('x');
  ylabel('t');
  zlabel('u(x,t)');
  rotate3d on
% print -deps pde.eps; print -dps pde.ps
```

Listing A.4.1. Main program `pde_1_main` for the solution of Eqs. (A.4.1a)–(A.4.1d)

We can note the following points about this program:

1. A *global* area is set up to share parameters between this main program and the ODE routines. The parameters pertaining to BCs (A.4.1c) and (A.4.1d) are then defined.

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with the ODE routine
  global a D n xl xu dx ncall ndss
%
% Parameters
  a=1.0;
  D=0.3;
  xl=0.0;
  xu=1.0;
```

2. As the comments explain, for the first three cases (`mf<=3`), BC (A.4.1c) is implemented at a grid point for $x = 0$ that is *algebraic* (does not involve an ODE), as will be explained when ODE routines `pde_1, pde_2, pde_3` are discussed.

```
%
% Cases with the time varying Dirichlet boundary condition
% (TVDBC) in or outside the spatial grid
  mf=1;
%
% mf = 1,2,3; TVDBC not part of the spatial grid
%
%   mf = 1 - explicit programming of the FDs with boundary
%            value calculated for the ODE at x = dx in pde_1.m
%
%   mf = 2 - ux routines dss002 to dss010 with boundary value
%            calculated for MOL ODEs in pde_2.m
%
%   mf = 3 - uxx routines dss042 to dss050 with boundary value
%            calculated for MOL ODEs in pde_3.m
%
%   For mf = 1,2,3, the boundary value must be calculated
%   after the return from the ODE integrator if it is to be
%   displayed and/or plotted.
    if(mf<=3)
      n=20;
      dx=(xu-xl)/n;
    end
```

Note that 20 grid points in x are used for these three cases.

3. For the next four cases (`mf = 4` to `mf = 7`), BC (A.4.1c) is implemented at a grid point for $x = 0$ that is *differential* (involves an ODE), as will be explained when ODE routines `pde_4` to `pde_7` are discussed. Note that 21 grid points are now used.

```
%
% mf = 4,5,6,7; TVDBC part of the spatial grid
%
%   mf = 4 - explicit programming of the FDs with boundary
%            value calculated at boundary in pde_4.m; this
%            boundary value is not returned by the ODE
%            integrator and it must therefore be restored
%            after the return from the integrator if it is to
%            be displayed and/or plotted
```

```
%
%   mf = 5 - explicit programming of the FDs with boundary
%            value calculated and derivative of boundary value
%            also calculated (for ODE at boundary) in pde_5.m;
%            the ODE integrator returns the correct boundary
%            value
%
%   mf = 6 - ux routines dss002 to dss010 with boundary value
%            calculated for MOL ODEs in pde_6.m; the ODE
%            integrator returns the correct boundary value
%
%   mf = 7 - uxx routines dss042 to dss050 with boundary value
%            calculated for MOL ODEs in pde_7.m; the ODE
%            integrator returns the correct boundary value
    if(mf>3)
      n=21;
      dx=(xu-xl)/(n-1);
    end
```

The comments largely explain how BC (A.4.1c) is handled within each of the seven ODE routines.

4. A spatial grid in x and IC (A.4.1b) are specified.

```
%
% Spatial grid
  x=[xl:dx:xu]';
%
% Initial condition
  u0=zeros(n,1);
```

5. A timescale $0 \leq t \leq 5$ is set with outputs displayed every 0.5 for a total of 11 output points.

```
%
% Independent variable for ODE integration
  t0=0.0;
  tf=2.5;
  tout=[0.0:0.25:tf]';
  nout=11;
  ncall=0;
%
% ODE integration
  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
if(mf==4) % explicit FDs
  [t,u]=ode15s(@pde_4,tout,u0,options); end
if(mf==5) % explicit FDs
  [t,u]=ode15s(@pde_5,tout,u0,options); end
if(mf==6) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
  [t,u]=ode15s(@pde_6,tout,u0,options); end
if(mf==7) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
  [t,u]=ode15s(@pde_7,tout,u0,options); end
```

Integration of the ODEs for the seven cases is by `ode15s`.

6. Output is displayed for each of the seven cases at $x = 0.5$ according to what is available and pertains to the case. Finally, the number of calls to the ODE routine, `ncall`, is displayed.

```
%
% Display selected output
  if(mf==1)
    n2=n/2;
    fprintf('\n              mf = %2d   n = %3d',mf,n);
  elseif(mf==2|mf==3)
    n2=n/2;
    fprintf('\n       mf = %2d   n = %3d   ndss = %2d', ...
            mf,n,ndss);
  elseif(mf==4|mf==5)
    n2=(n-1)/2+1;
    fprintf('\n              mf = %2d   n = %3d',mf,n);
  elseif(mf==6|mf==7)
    n2=(n-1)/2+1;
    fprintf('\n       mf = %2d   n = %3d   ndss = %2d', ...
            mf,n,ndss);
  end
  fprintf('\n   abstol = %8.1e   reltol = %8.1e', ...
          abstol,reltol);
  if(mf<=3)
    fprintf('\n\n    t   u(x=dx,t)  u(x=0.5,t)   u(x=1,t)\n');
  else
    fprintf('\n\n    t    u(x=0,t)  u(x=0.5,t)   u(x=1,t)\n');
  end
```

```
  for it=1:nout
    if(mf==4)u(it,1)=1-cos(a*pi*t(it));end
    fprintf('%6.2f%12.6f%12.6f%12.6f\n',t(it),u(it,1), ...
            u(it,n2),u(it,n));
  end
  fprintf('\n ncall = %4d\n',ncall);
```

7. Plotting of the solution is included according to the particular case.

```
%
% Plot numerical solution as u(x,t) vs x with t as parameter
  if(mf<=3)
    uplot(:,2:n+1)=u(:,1:n);
%
%   Include boundary value in plot
    for it=1:nout
      uplot(it,1)=1-cos(a*pi*t(it));
    end
  elseif(mf>3)
    uplot=u;
%
%   Restore boundary value for plot (mf = 4)
    if(mf==4)
      for it=1:nout
        uplot(it,1)=u(it,1);
      end
    end
  end
  figure(1);
  plot(x,uplot,'-k'); axis tight
  title('u(x,t) vs x, t = 0,0.25,...,2.5');
  xlabel('x');
  ylabel('u(x,t)')
  figure(2);
  surf(x,tout,uplot);
  shading interp; axis tight
  colormap jet
  title('u(x,t) vs x, t = 0,0.25,...,2.5');
  xlabel('x');
  ylabel('t');
  zlabel('u(x,t)');
  rotate3d on
% print -deps pde.eps; print -dps pde.ps
```

Note that the plots include the boundary value at $x = 0$ through the use of the BC function $u(x = 0, t) = f(t) = 1 - \cos(a\pi t)$ according to BC (A.4.1c).

We now consider the seven ODE routines. The first, `pde_1`, called by `ode15s` with `mf=1`, is given in Listing A.4.2.

```
  function ut=pde_1(t,u)
%
% Problem parameters
  global a D n xl xu dx ncall ndss
%
% PDE
  dx2=dx^2;
  for i=1:n
    if(i==1)
      ubc=1-cos(a*pi*t);
      ut(1)=(u(2)-2.0*u(1)+ubc)/dx2;
    elseif(i==n)
      ut(i)=2.0*(u(i-1)-u(i))/dx2;
    else
      ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
    end
  end
  ut=D*ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Listing A.4.2. ODE routine `pde_1`

We can note the following points about `pde_1`:

1. The function is defined and a global area is included to share parameter values with the main program, `pde_1_main`.
2. The `for` loop steps through the $n = 20$ grid points in the method of lines (MOL) solution. For the first grid point at `x=dx` (`i=1`), the function $f(t)$ in BC (A.4.1c) is computed and then used in the three-point FD approximation of $\partial^2 u/\partial x^2$

$$\frac{\partial^2 u(x = \Delta x, t)}{\partial x^2} \approx \frac{u(x = 2\Delta x, t) - 2u(x = \Delta x, t) + u(x = 0, t)}{\Delta x^2} \tag{A.4.2}$$

to give the derivative $\partial u/\partial t$ according to Eq. (A.4.1a) (with $a = 1, D = 0.3$).

```
%
% PDE
  dx2=dx^2;
  for i=1:n
    if(i==1)
      ubc=1-cos(a*pi*t);
      ut(1)=(u(2)-2.0*u(1)+ubc)/dx2;
```

Again, note that the first ODE is at $x = dx$, but the required boundary value `ubc` is used, which in effect "drives" the solution at the grid points `i=1` to `20`.

3. The ODE at $x = 1$ includes BC (A.4.1d) approximated (from Eq. (A.4.2)) as

$$\frac{\partial^2 u(x=1,t)}{\partial x^2} \approx 2\frac{u(x=1-\Delta x,t)-u(x=1,t)}{\Delta x^2}$$

where BC (A.4.1d) is approximated as

$$\frac{\partial u(x=1,t)}{\partial x} \approx \frac{u(x=1+\Delta x,t)-u(x=1-\Delta x,t)}{2\Delta x} = 0$$

or

$$u(x=1+\Delta x,t) = u(x=1-\Delta x,t)$$

```
    elseif(i==n)
      ut(i)=2.0*(u(i-1)-u(i))/dx2;
```

4. At the interior points (`i=2` to `i=n-1`), $\partial^2 u/\partial x^2$ is approximated by Eq. (A.4.2) and $\partial u/\partial t$ is coded as

```
    else
      ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
    end
  end
```

5. A transpose is required for `ode15s` and the counter for the calls to `pde_1` is incremented.

```
  ut=D*ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Table A.4.1. Output from `pde_1_main` and `pde_1` (`mf=1`)

```
            mf =  1   n =  20
 abstol = 1.0e-004   reltol = 1.0e-004

    t   u(x=dx,t)  u(x=0.5,t)    u(x=1,t)
 0.00    0.000000    0.000000    0.000000
 0.25    0.222812    0.012251    0.000372
 0.50    0.834292    0.127294    0.018234
 0.75    1.499688    0.380408    0.105870
 1.00    1.838224    0.691448    0.290116
 1.25    1.657580    0.930678    0.533912
 1.50    1.068480    1.000662    0.754220
 1.75    0.419962    0.895990    0.871942
 2.00    0.095138    0.707407    0.859815
 2.25    0.287020    0.569835    0.759544
 2.50    0.885440    0.584187    0.658633

ncall =  126
```

The output from `pde_1_main` (Listing A.4.1) and `pde_1` (Listing A.4.2) (for `mf=1`) is given in Table A.4.1. Note that the first column of `u(x,t)` values corresponds to `x=dx` and therefore does not correspond to BC (A.4.1c) $u(x = 0, t) = f(t) = 1 - \cos(a\pi t)$. The values of this BC are (for selected values of t from Table A.4.1 with $a = 1$) as follows:

t	$1 - \cos(a\pi t)$
0	0
0.5	1
1	2
1.5	1
2	0
2.5	1

These values of $u(x = 0, t)$ were added in `pde_1_main` of Listing A.4.1 so that the plot shown in Figure A.4.1 includes these boundary values.

The interpretation of the line plot of Figure A.4.1 is not obvious (note the $u(x = 0, t)$ values) and is facilitated by the 3D plot of Figure A.4.2. In order to gain satisfactory resolution of the 3D plot, additional output values for $t = 0, 0.05, \ldots, 2.5$ were specified with the following code (used in `pde_1_main`):

```
%
% Independent variable for ODE integration
  t0=0.0;
  tf=2.5;
  tout=[0.0:0.05:tf]';
  nout=51;
```

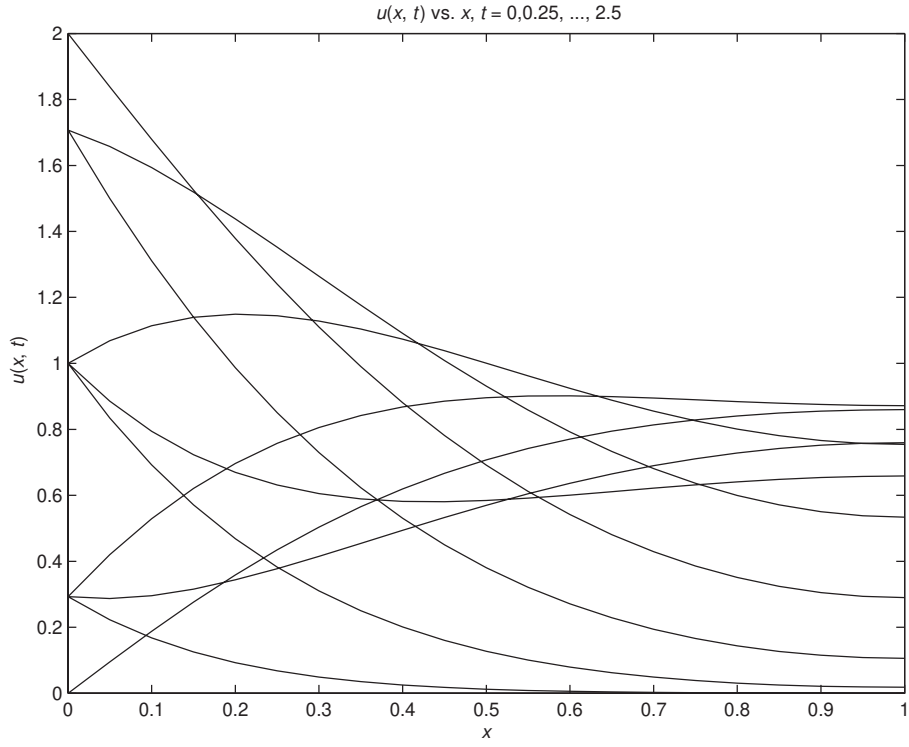

Figure A.4.1. Solution of Eqs. (A.4.1a)–(A.4.1d) from `pde_1_main` and `pde_1` (`mf=1`)

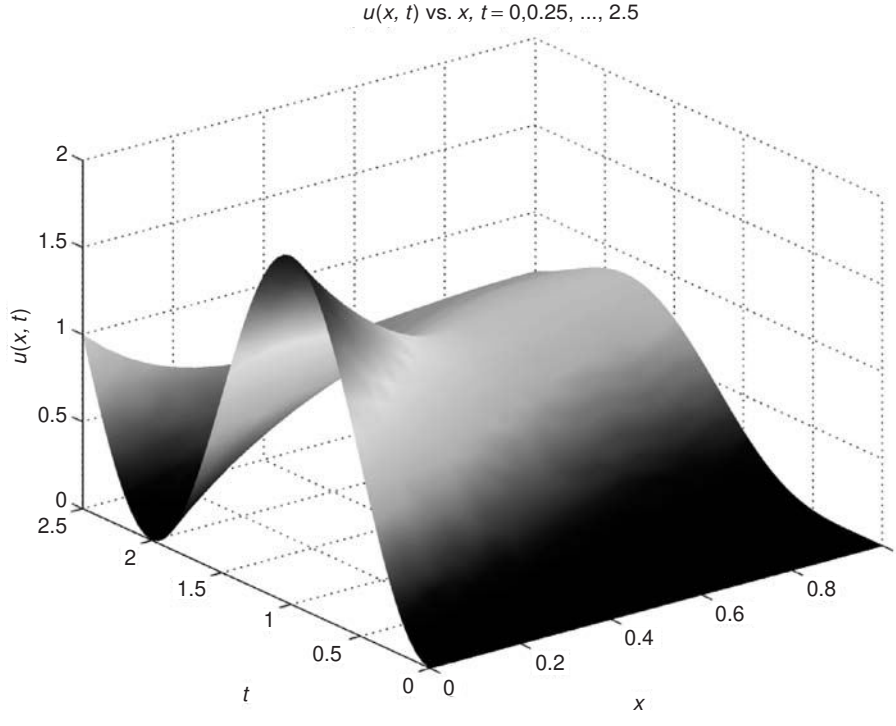

Figure A.4.2. Solution of Eqs. (A.4.1a)–(A.4.1d) from `pde_1_main` and `pde_1` (`mf=1`)

We note from Figure A.4.1

1. BC (A.4.1c) at $x = 0$
2. BC (A.4.1d) (zero slope) at $x = 1$

An analytical solution could easily be derived to test the numerical solution, but we did not include this test to save some space. The next ODE routine, `pde_2`, called by `ode15s` with `mf=2`, is given in Listing A.4.3.

```
  function ut=pde_2(t,u)
%
% Problem parameters
  global a D n xl xu ncall ndss
%
% Calculate ux
  ue(2:n+1)=u;
  ue(1)=1-cos(a*pi*t);
  if    (ndss== 2) uex=dss002(xl,xu,n+1,ue); % second order
  elseif(ndss== 4) uex=dss004(xl,xu,n+1,ue); % fourth order
  elseif(ndss== 6) uex=dss006(xl,xu,n+1,ue); % sixth order
  elseif(ndss== 8) uex=dss008(xl,xu,n+1,ue); % eighth order
  elseif(ndss==10) uex=dss010(xl,xu,n+1,ue); % tenth order
  end
%
% BC at x = 1 (Neumann)
  uex(n+1)=0.0;
%
% Calculate uxx
  if    (ndss== 2) uexx=dss002(xl,xu,n+1,uex); % second order
  elseif(ndss== 4) uexx=dss004(xl,xu,n+1,uex); % fourth order
  elseif(ndss== 6) uexx=dss006(xl,xu,n+1,uex); % sixth order
  elseif(ndss== 8) uexx=dss008(xl,xu,n+1,uex); % eighth order
  elseif(ndss==10) uexx=dss010(xl,xu,n+1,uex); % tenth order
  end
%
% PDE
  uxx=uexx(2:n+1);
  ut=D*uxx';
%
% Increment calls to pde_2
  ncall=ncall+1;
```

Listing A.4.3. ODE routine `pde_2`

We can note the following points about `pde_2`:

1. After the definition of the function and inclusion of a global area, an additional grid point (corresponding to $x = 0$) is added and the BC function of Eq. (A.4.1c) is added. This is done so that the differentiation routine `dss004` (from `ndss=4` set before `ode15s` is called in `pde_1_main` with `mf=2`) can operate on the expanded grid with dependent variable `ue`.

```
% Calculate ux
  ue(2:n+1)=u;
  ue(1)=1-cos(a*pi*t);
  if    (ndss== 2) uex=dss002(xl,xu,n+1,ue); % second order
  elseif(ndss== 4) uex=dss004(xl,xu,n+1,ue); % fourth order
  elseif(ndss== 6) uex=dss006(xl,xu,n+1,ue); % sixth order
  elseif(ndss== 8) uex=dss008(xl,xu,n+1,ue); % eighth order
  elseif(ndss==10) uex=dss010(xl,xu,n+1,ue); % tenth order
  end
```

2. BC (A.4.1d) is used to reset the $x = 1$ value of `uex`. Then the second derivative in x, in array `uexx`, is computed by differentiating the first derivative (*stage-wise differentiation* is used).

```
%
% BC at x = 1 (Neumann)
  uex(n+1)=0.0;
%
% Calculate uxx
  if    (ndss== 2) uexx=dss002(xl,xu,n+1,uex); % second order
  elseif(ndss== 4) uexx=dss004(xl,xu,n+1,uex); % fourth order
  elseif(ndss== 6) uexx=dss006(xl,xu,n+1,uex); % sixth order
  elseif(ndss== 8) uexx=dss008(xl,xu,n+1,uex); % eighth order
  elseif(ndss==10) uexx=dss010(xl,xu,n+1,uex); % tenth order
  end
```

3. Equation (A.4.1a) is then programmed, followed by the transpose (for `ode15s`) and the incrementing of the counter `ncall`. Note that Eq. (A.4.1a) is programmed with $n = 20$ ODEs (not $n = 21$ ODEs).

```
%
% PDE
  uxx=uexx(2:n+1);
  ut=D*uxx';
%
% Increment calls to pde_2
  ncall=ncall+1;
```

The output from pde_1_main and pde_2 is essentially the same as in Table A.4.1 and Figures A.4.1 and A.4.2, so it is not reproduced here.

The next ODE routine, pde_3, called by ode15s with mf=3, is given in Listing A.4.4.

```
  function ut=pde_3(t,u)
%
% Problem parameters
  global a D n xl xu ncall ndss
%
% Calculate ux
  ue(2:n+1)=u;
  ue(1)=1-cos(a*pi*t);
  uex(n+1)=0.0;
  nl=1; % Dirichlet
  nu=2; % Neumann
  if    (ndss==42) uexx=dss042(xl,xu,n+1,ue,uex,nl,nu);
  % second order
  elseif(ndss==44) uexx=dss044(xl,xu,n+1,ue,uex,nl,nu);
  % fourth order
  elseif(ndss==46) uexx=dss046(xl,xu,n+1,ue,uex,nl,nu);
  % sixth order
  elseif(ndss==48) uexx=dss048(xl,xu,n+1,ue,uex,nl,nu);
  % eighth order
  elseif(ndss==50) uexx=dss050(xl,xu,n+1,ue,uex,nl,nu);
  % tenth order
  end
%
% PDE
  uxx=uexx(2:n+1);
  ut=D*uxx';
%
% Increment calls to pde_3
  ncall=ncall+1;
```

Listing A.4.4. ODE routine pde_3

This routine is similar to pde_2 (Listing A.4.3). The only essential difference is the call to dss044 (with mf=3, ndss=44 in pde_1_main rather than two calls to dss004); dss044 computes the second derivative directly from the PDE dependent variable ue. Note that both BC (A.4.1c) and BC (A.4.1d) are included. The output from pde_1_main and pde_3 is essentially the same as in Table A.4.1 and Figures A.4.1 and A.4.2, so it is not reproduced here.

The next ODE routine, pde_4, called by ode15s with mf=4, is given in Listing A.4.5.

```
  function ut=pde_4(t,u)
%
% Problem parameters
  global a D n xl xu dx ncall ndss
%
% PDE
  dx2=dx^2;
  for i=1:n
    if(i==1)
      u(1)=1-cos(a*pi*t);
      ut(1)=0.0;
    elseif(i==n)
      ut(i)=2.0*(u(i-1)-u(i))/dx2;
    else
      ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
    end
  end
  ut=D*ut';
%
% Increment calls to pde_4
  ncall=ncall+1;
```

Listing A.4.5. ODE routine `pde_4`

`pde_4` is similar to `pde_1` in Listing A.4.1. Here are a few additional points:

1. The essential difference between `pde_1` and `pde_4` is that the latter uses an ODE grid point at $x = 0$ (rather than an algebraic grid point as in `pde_1`).
2. Note that the programming of the BC function $f(t)$ in Eq. (A.4.1c) is stored as the dependent variable at the first grid point, `u(1)=1-cos(a*pi*t);`. While this value of `u(1)` is used to "drive" the other $n - 1 = 20$ ODEs, it does not actually set `u(1)` so that it can be returned from `pde_1` (just because that is the way `ode15s` works).
3. In order for `u(1)` to be available for plotting, it is reset in the main program `pde_1_main`. Here is the code in `pde_1_main`.

```
%
%   Restore boundary value for plot (mf = 4)
    if(mf==4)
      for it=1:nout
        uplot(it,1)=u(it,1);
      end
    end
```

4. A similar resetting of `u(1)` is also included for the tabulated numerical output:

```
  for it=1:nout
    if(mf==4)u(it,1)=1-cos(a*pi*t(it));end
    fprintf('%6.2f%12.6f%12.6f%12.6f\n', ...
            t(it),u(it,1),u(it,n2),u(it,n));
  end
```

The output from `pde_1_main` and `pde_4` is essentially the same as in Table A.4.1 and Figures A.4.1 and A.4.2, so it is not reproduced here.

The next ODE routine, `pde_5`, called by `ode15s` with `mf=5`, is given in Listing A.4.6.

```
  function ut=pde_5(t,u)
%
% Problem parameters
  global a D n xl xu dx ncall ndss
%
% PDE
  dx2=dx^2;
  for i=1:n
    if(i==1)
      u(1)=1-cos(a*pi*t);
      ut(1)=a*pi*sin(a*pi*t)/D;
    elseif(i==n)
      ut(i)=2.0*(u(i-1)-u(i))/dx2;
    else
      ut(i)=(u(i+1)-2.0*u(i)+u(i-1))/dx2;
    end
  end
  ut=D*ut';
%
% Increment calls to pde_5
  ncall=ncall+1;
```

Listing A.4.6. ODE routine `pde_5`

`pde_5` is similar to `pde_4` (in Listing A.4.6). The essential difference is that the ODE at $x = 0$ is programmed as the derivative of the function $f(t)$ in BC (A.4.1c).

```
      ut(1)=a*pi*sin(a*pi*t)/D;
```

The division by D is required to maintain the time-varying BC identity $\{du(x = 0, t)\}/dt$ = (ut(1) = $[d\{1 - \cos(a\pi t)\}]/dt = a\pi \sin(a\pi t)$, which is the first element in the vector ut ($= u_t$). This is because, once it is assembled, the whole vector ut is multiplied by D (through ut=D*ut' in the Matlab code) after which ut(1) has the correct value.

This approach illustrates the basic requirement of ode15s that the dependent variables such as u(1) are changed by their corresponding ODEs, and not by algebraic statements such as in pde_4. While this approach may seem completely logical, it has the drawback that the derivative of $f(t)$ is required, and this may not be readily available (depending on the particular $f(t)$ used in an application). The output from pde_1_main and pde_5 is essentially the same as in Table A.4.1 and Figures A.4.1 and A.4.2, so it is not reproduced here.

The next ODE routine, pde_6, called by ode15s with mf=6, is given in Listing A.4.7.

```
  function ut=pde_6(t,u)
%
% Problem parameters
  global a D n xl xu ncall ndss
%
% Calculate ux
  u(1)=1-cos(a*pi*t);
  if    (ndss== 2) ux=dss002(xl,xu,n,u); % second order
  elseif(ndss== 4) ux=dss004(xl,xu,n,u); % fourth order
  elseif(ndss== 6) ux=dss006(xl,xu,n,u); % sixth order
  elseif(ndss== 8) ux=dss008(xl,xu,n,u); % eighth order
  elseif(ndss==10) ux=dss010(xl,xu,n,u); % tenth order
  end
%
% BC at x = 1 (Neumann)
  ux(n)=0.0;
%
% Calculate uxx
  if    (ndss== 2) uxx=dss002(xl,xu,n,ux); % second order
  elseif(ndss== 4) uxx=dss004(xl,xu,n,ux); % fourth order
  elseif(ndss== 6) uxx=dss006(xl,xu,n,ux); % sixth order
  elseif(ndss== 8) uxx=dss008(xl,xu,n,ux); % eighth order
  elseif(ndss==10) uxx=dss010(xl,xu,n,ux); % tenth order
  end
%
% PDE
  ut=D*uxx';
%
% Increment calls to pde_6
  ncall=ncall+1;
```

Listing A.4.7. ODE routine pde_6

pde_6 is similar to pde_2 (in Listing A.4.3). The essential difference is that the grid point at the boundary $x = 0$ is included in the spatial grid (rather than using a separate value outside the grid for $u(x = 0, t)$ as in pde_2).

Again, `u(1)=1-cos(a*pi*t);` sets the value of `u(1)`, so it "drives" the other $n - 1 = 20$ ODEs, but it does not actually set `u(1)` so that it can be returned from pde_1. Rather the coding

```
%
% PDE
  ut=uxx';
```

includes the ODE at $x = 0$. In other words, this approach is both algebraic (from `u(1)=1-cos(a*pi*);`) and differential (from `ut=uxx';`). The output from pde_1_main and pde_6 is essentially the same as in Table A.4.1 and Figures A.4.1 and A.4.2, so it is not reproduced here.

The final ODE routine, pde_7, called by ode15s with mf=7, is given in Listing A.4.8.

```
  function ut=pde_7(t,u)
%
% Problem parameters
  global a D n xl xu ncall ndss
%
% Calculate ux
  u(1)=1-cos(a*pi*t);
  ux(n)=0.0;
  nl=1; % Dirichlet
  nu=2; % Neumann
  if    (ndss==42) uxx=dss042(xl,xu,n,u,ux,nl,nu);
  % second order
  elseif(ndss==44) uxx=dss044(xl,xu,n,u,ux,nl,nu);
  % fourth order
  elseif(ndss==46) uxx=dss046(xl,xu,n,u,ux,nl,nu);
  % sixth order
  elseif(ndss==48) uxx=dss048(xl,xu,n,u,ux,nl,nu);
  % eighth order
  elseif(ndss==50) uxx=dss050(xl,xu,n,u,ux,nl,nu);
  % tenth order
  end
%
% PDE
  ut=D*uxx';
%
% Increment calls to pde_7
  ncall=ncall+1;
```

Listing A.4.8. ODE routine pde_7

pde_7 is similar to pde_3 (in Listing A.4.4). The essential difference is that the grid point at the boundary $x = 0$ is included in the spatial grid (rather than using a separate value outside the grid for $u(x = 0, t)$ as in pde_3).

Again, `u(1)=1-cos(a*pi*t);` sets the value of `u(1)`, so it "drives" the other $n - 1 = 20$ ODEs, but it does not actually set `u(1)` so that it can be returned from pde_1. Rather the coding

```
%
% PDE
  ut=uxx';
```

includes the ODE at $x = 0$. In other words, this approach is both algebraic (from `u(1)=1-cos(a*pi*t);`) and differential (from `ut=uxx';`). The output from pde_1_main and pde_7 is essentially the same as in Table A.4.1 and Figures A.4.1 and A.4.2, so it is not reproduced here.

In summary, our intent for this discussion of a time-varying Dirichlet BC is to show several (seven) different ways the same BC can be programmed. The output in all cases was essentially the same. We do not have a preferred method and consider all seven to be about equivalent. The computational requirement for this problem was not significantly different (e.g., `ncall = 126` and `ncall = 133` for pde_1 and pde_7, respectively). Perhaps the most important point to retain is that the Matlab ODE integrators do not return dependent variables that are set algebraically in the ODE routine. Rather, they must be set via ODEs if they are to be returned by the integrator.

APPENDIX 5

The Differentiation in Space Subroutines Library

FIRST-DERIVATIVE ROUTINES

The Differentiation in Space Subroutines (DSS) library contains five Matlab routines for the numerical calculation of first derivatives. They are as follows:

`dss002()`	3-point stencil, 2nd-order approximation
`dss004()`	5-point stencil, 4th-order approximation
`dss006()`	7-point stencil, 6th-order approximation
`dss008()`	9-point stencil, 8th-order approximation
`dss010()`	11-point stencil, 10th-order approximation

The call format is the same for each routine, and the call definition of `dss004()` is given next by way of example as

```
function [ux]=dss004(xl,xu,n,u)
```

where function `dss004` computes the first derivative, u_x, of a variable $u(x)$ over the spatial domain $x_l \leq x \leq x_u$ from a classical *five-point, fourth-order finite-difference* approximation. To provide the necessary number of grid points in x, we recommend that the minimum number of grid points used for a particular problem should be equal to the *stencil size plus 10*.

Argument List

Input Variables

`xl` left value of the spatial independent variable, x

`xu` right value of the spatial independent variable, x

`n` number of spatial grid points in the x domain including the boundary points

`u` one-dimensional (1D) array of n values of dependent variable u at the n points for which the derivative is to be computed

Output Variables

`ux` one-dimensional array of the *first derivative* of u at the n grid points

SECOND-DERIVATIVE ROUTINES

The DSS library contains five Matlab routines for the numerical calculation of second derivatives. They are as follows:

`dss042()`	4-point stencil, 2nd-order approximation
`dss044()`	6-point stencil, 4th-order approximation
`dss046()`	8-point stencil, 6th-order approximation
`dss048()`	10-point stencil, 8th-order approximation
`dss050()`	12-point stencil, 10th-order approximation

The call format is the same for each routine, and the call definition of `dss044()` is given next by way of example as

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

where function `dss044` computes the second derivative, u_{xx} of a variable $u(x)$ over the spatial domain $x_l \leq x \leq x_u$ from a classical *six-point, fourth-order finite-difference* approximation. To provide the necessary number of grid points in x, we recommend that the minimum number of grid points used for a particular problem should be equal to the *stencil size plus 10*.

Argument List

Input Variables

`xl` left value of the spatial independent variable, x

`xu` right value of the spatial independent variable, x

`n` number of spatial grid points in the x domain including the boundary points

`u` one-dimensional array of n values of dependent variable u at the n grid points for which the derivative is to be computed

`ux` one-dimensional array with the *first derivative* of u; the end values of u_x, $u_x(x = x_l)$ and $u_x(x = x_u)$, are used in Dirichlet or Neumann

boundary conditions (BCs) at $x = x_l$ and $x = x_u$, depending on the arguments `nl` and `nu`

`nl` integer index for the type of BC at $x = x_l$. The allowable values are

1. Dirichlet BC at $x = x_l$ ($u_x(x = x_l)$ is not used)
2. Neumann BC at $x = x_l$ ($u_x(x = x_l)$ is used)

`nu` integer index for the type of BC at $x = x_u$ (input). The allowable values are

1. Dirichlet BC at $x = x_u$ ($u_x(x = x_u)$ is not used)
2. Neumann BC at $x = x_u$ ($u_x(x = x_u)$ is used)

Output Variables

`uxx` one-dimensional array with the *second derivative* of u at the n grid points

HIGHER-ORDER AND MIXED DERIVATIVES

Higher-order and mixed derivatives can be calculated by calling the appropriate DSS routines in sequence. This is known as *stagewise differentiation*. For instance, a finite-difference approximation to a third derivative can be obtained by three successive calls to the first-derivative library routine `dss004()`. As we are dealing with a third derivative, this would require three BCs that would be determined by the physical nature of the problem.

Note: In general, p boundary conditions (BCs) are required for each pth derivative with respect to a spatial variable and q initial conditions (ICs) are required for each qth derivative with respect to a temporal variable. Thus, for example, the 1D Korteweg–deVries (KdV) equation $\partial u/\partial t = -u(\partial u/\partial x) + \mu(\partial^3 u/\partial x^3)$ would require three spatial BCs and one IC.

This is illustrated by the Matlab code fragment given in Listing A.5.1 for a dependent variable $u(x, t)$ with a Dirichlet BC at the left end of the spatial domain $u(x = x_l, t) = u_{bl}$, a time-varying Neumann BC at the right end of the spatial domain $u_x(x = x_u, t) = f_1(t)$, and a time-varying second derivative BC at the right end of the spatial domain $u_{xx}(x = x_u, t) = f_2(t)$.

```
function [uxxx] = thirdDerivEx1(t,u)
global n ubl xl xu          % Global variables
.
.
u(1) = ubl;                 % Dirichlet BC at left end of
                            % domain
ux   = dss004(xl,xu,n,u);   % First derivative of u
ux(n)= f1(t);               % Neumann BC at right end of
                            % domain
uxx  = dss004(xl,xu,n,ux);  % Second derivative of u
```

```
uxx(n)=f2(t);                % Second derivative BC at right
                             % end of domain
uxxx = dss004(xl,xu,n,uxx); % Third derivative of u
return;
```

Listing A.5.1. Calculating a third derivative by three successive calls to `dss004()`

An equally valid approach would be to call the second-derivative function `dss044()` followed by the first-derivative function `dss004()`, as illustrated in Listing A.5.2.

```
function [uxxx]=thirdDerivEx2(t,u)
global n ubl xl xu           % Global variables
.
.
u(1) = ubl;                  % Dirichlet BC at left end of
                             % domain
ux(n)= f1(t);                % Neumann BC at right end of
                             % domain
nl   = 1;                    % Left BC is Dirichlet (do
                             % nothing)
nu   = 2;                    % Right BC is Neumann
uxx  = dss044(xl,xu,n,u,ux,nl,nu); % Second derivative of u
uxx(n)=f2(t);                % Second derivative BC at right
                             % end of domain
uxxx = dss004(xl,xu,n,uxx); % Third derivative of u
return;
```

Listing A.5.2. Calculating a third derivative by calls to `dss004()` and `dss044()`

In the code fragments given in Listings A.5.1 and A.5.2 both calls are made to fourth-order finite-difference approximations. In formulating the solution to any PDE problem, it is important to select numerical approximations of the same order for all partial derivatives. This will, in general, ensure that truncation errors (for each derivative) vary by the same order of magnitude as the grid spacing is increased or decreased. An example of the use of the DSS library to calculate mixed partial derivatives is given in Chapter 12.

OBTAINING THE DSS LIBRARY

Additional information about the DSS library, including the PDE routines, is available from `http://www.scholarpedia.org/article/method_of_lines`. Enquiries can be directed to wes1@lehigh.edu.

APPENDIX 6

Animating Simulation Results

GENERAL

The results obtained from running the simulations described in this book are generally provided in the form of a matrix. This matrix includes the solutions for each time step and provides the means of generating a graphical sequence of images that can be used to create an animation. An animation creates the effect of motion and often provides the viewer with a clearer insight into the physical process being analyzed than purely static images. In this appendix we demonstrate some Matlab methods and the corresponding code for creating *AVI movies* and *animated GIF files*. For additional information the reader is referred to general Matlab references [1–3].

MATLAB MOVIE

The creation of a movie is quite straightforward using the built-in tools provided with Matlab. The process consists of the following steps, assuming a type `surfl` plot with fixed viewpoint:

- Define problem mathematically.

```
u=f(x,y);                % Create initial 2D reference data
```

- Plot reference frame with desired specification – only basic option shown; for additional options refer to Matlab help.

```
surfl(x,y,u);              % Surface plot with lighting
axis([x1 x2 y1 y2 u1 u2]); % Ensures scaling is fixed
set(gca,'nextplot', ...    % Set where to draw next plot
        'replacechildren');
```

```
colormap cool;              % Select colormap to enhance plot
shading interp;             % Interpolated color shading
view(elevation azimuth)     % Select desired viewpoint
                            % for frames
```

- Define number of frames say, `N`.

```
myMovie=moviein(N);         % Define frame matrix of size N
```

- Perform `N` calculations, plotting the result of each calculation and saving as a movie frame.

```
for i=1:N                   % for loop to generate N frames
  .
  .
  perform calculation(s)    % Calculations as necessary
  .
  .
  surfl(x,y,u);             % New surface plot
  myMovie(:,i)=getframe;    % Add new plot to frame matrix
end
```

- Display movie within Matlab environment.

```
movie(myMovie)              % Play movie
```

Note:

1. If desired, the viewpoint can also be changed during the movie to reflect the changing solution by including an appropriate `view` statement within the `for` loop after the calculations and prior to the `surfl` statement.
2. The Matlab function `movie` also has additional options, for example, to control the number of frames displayed per second; refer to Matlab help for additional information.

BASIC EXAMPLE

The *peaks example* that uses the `surf` function, given in Listing A.6.1 has been taken from the Matlab help. It shows with just a few lines of code how effective this facility can be in providing clarity to a physical or mathematical process.

```
  Z=peaks; surf(Z);
  axis tight
  set(gca,'nextplot','replacechildren')
%
% Record the movie
  for j=1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j)=getframe;
  end
%
% Play the movie twenty times
  movie(F,20)
```

Listing A.6.1. *Peaks example* from Matlab help

AVI MOVIES

The *AVI movie* creation facility within Matlab enables the user to create a standalone *Audio/Video Interleaved* (AVI) file that can be viewed in media players available on most personal computers. The process for the generation of an AVI file is a simple extension to creating a Matlab movie, as described earlier. The peaks example code given in Listing A.6.2 has been modified to include an extra line to convert the frames to an AVI movie and save the result to the file `peaks.avi`.

```
  Z=peaks; surf(Z);
  axis tight
  set(gca,'nextplot','replacechildren');
%
% Record the movie
  for j=1:20
    surf(sin(2*pi*j/20)*Z,Z)
    F(j)=getframe;
  end
%
% Play the movie twenty times
  movie(F,20)
  movie2avi(F,'peaks.avi','fps',4) % Save movie to file
                                   % 'peaks.avi' with the
                                   % Frames per second option
                                   % set to 4
```

Listing A.6.2. Peaks example with conversion to *AVI format* and *save to file*

We will now provide AVI movie examples that can be used in conjunction with the code detailed in the *Burgers equation*, *Schrödinger equation*, and *Korteweg–deVries equation* chapters. These examples use the Matlab `avifile` function that is slightly more complex than `movie2avi`, but is more flexible. The differences will be apparent from the coding and more information is available from the Matlab help facility.

EXAMPLE – BURGERS EQUATION MOVIE

The following code generates an animation movie file in AVI format and utilizes Matlab workspace data populated by a previous simulation run; that is, this code should be run following the Burgers equation simulation code detailed in Chapter 5.

The file name to save the movie to is initially set to `BurgersEqn_01.avi`. However, if this file already exists, then the user is prompted to choose to either supply a new file name or overwrite the existing file.

The user is then presented with a dialog box from which to choose:

1. Two-dimensional (2D) plot
2. Two-dimensional plus analytical solution plots
3. Three-dimensional surface plot

The movie-generating process then proceeds and the final frame is saved as an image at a resolution of 300 dpi to file `BurgersEqnPlot.png`. A detailed description of the code will not be provided as ample comments are included (see Listing A.6.3).

```
%
% Code to generate Burgers equation Movie
% Run after pde_1_main.m
% Generates animation movies
% User should select from choices presented in dialog boxes!
%
% Get screen size
  scrsz=get(0,'ScreenSize');
%
% Define required position for figure
  rect=[0 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2];
%
% Create figure at required position
  h1=figure('Position',rect);
  set(h1,'DoubleBuffer','on');
%
% Define required dimensions of image  to be saved within
% figure
  frameRect=[0 0 scrsz(3)/2 scrsz(4)/2];
%
```

```
% Set up animation parameters
  frameSteps=3;                  % Plot image every frameSteps
  nframes=ceil(mm/frameSteps);   % Number of frames in the
                                 % movie
  Frames=moviein(nframes);       % Initialize the matrix
                                 % 'Frames'
  avi_fName='BurgersEqn_01.avi'; % Movie file name
  if exist(avi_fName) ~= 0
    fprintf('File: %s, already exists!\n\n', avi_fName);
    fileMsg=sprintf('File: %s, already exists - Choose to:', ...
                    avi_fName);
    namChoice=menu(fileMsg,'Overwrite','New Name');
    if namChoice==1
    delete(avi_fName);
  else
    newName=inputdlg('Movie file name:','File Name Request', ...
                     1,{avi_fName});
    if isempty(newName{1})
      delete(avi_fName);
    else
      [pathstr,name,ext,versn]=fileparts(newName{1});
      if strcmp('.avi',ext)==0
        avi_fName=strcat(newName{1},'.avi');
      else
        avi_fName=newName{1};
      end
    end
    if exist(avi_fName)
      delete(avi_fName);
    end
  end
 %
 % avi movie parameters
   fps=1;                      % Frames per second
   aviQuality=100;             % Image quality, 0-100 (100 best)
   aviCompression='Cinepak'; % Image compression
                               % mode - see help
   aviMov=avifile(avi_fName,'compression',aviCompression, ...
          'quality', aviQuality);
 %
 % Movie calculations
   plotType=1;       % Set plot type, 1=2D, 2=3D
   plotType=menu('Choose Plot Type:','2D', ...
                 '2D+Analytical Solution','3D');
   qq=1;             % Set frame skip counter
   frameCount=0;     % Initialise frame counter
   u1=0.1*ones(mm,n);
```

```
    for it=1:mm
      qq=qq-1;
      if qq==0,
        frameCount=frameCount+1;
        if plotType==1 | plotType==2
%
%       2D Plot
          if plotType==1
            h2=plot(x,u(it,:));
          else
            plot(x,u(it,:),'o',x,u_anal(it,:),'-')
            legend('Numerical solution', ...
                   'Analytical solution', ...
                   'location', 'southwest');
          end
        hold on
        xlabel('x')
        ylabel('u(x,t)')
        set(gca,'XLim',[0 1],'YLim',[0 1]);
        title('Burgers equation');
        grid on
      else
%
%     Update u1 for each frame
      if it <= frameSteps     % First frame
        u1(it,:)=u(it,:);
      else                    % Subsequent frames
        u1(it-frameSteps:it,:)=u(it-frameSteps:it,:);
      end
      surfl(x,t,u1,'light');
      shading interp
      title('Burgers equation');
      set(get(gca,'XLabel'),'String','space, x')
      set(get(gca,'YLabel'),'String','time, t')
      set(get(gca,'ZLabel'),'String',...
                            'dependent variable, u(t,x)')
      set(gca,'XLim',[0 1],'YLim',[0 1],'ZLim', [0 1]);
      axis tight
      view(40,29);
      colormap('cool');   % set color map
      end
      Frames=getframe(h1, frameRect); % Create image in Frames
                                      % array
      aviMov=addframe(aviMov,Frames); % Add frame to avi movie
                                      % file
      disp(['    Plot number: ', num2str(frameCount), ...
            ' created']);
```

```
    qq=frameSteps;  % reset qq
  end
  aviMov=close(aviMov); % Close avi movie file
  fprintf('\navi movie created in file: %s \n', avi_fName);
%
% Print image to file
% print -dpng -r300 BurgersEqnPlot
```

Listing A.6.3. Code to be used following a Burgers equation simulation run to produce a variety of 2D and 3D plots

An example of 3D plot is shown in Chapter 5.

EXAMPLE – SCHRÖDINGER EQUATION MOVIE

The following code generates an animation movie file in AVI format and utilizes Matlab workspace data populated by a previous simulation run; that is, this code should be run following the Schrödinger simulation code detailed in Chapter 6.

The file name to save the movie to is initially set to `SchrodingersEqn_01.avi`. However, if this file already exists, then the user is prompted to choose to either supply a new file name or overwrite the existing file.

The user is then presented with a dialog box from which to choose:

1. Two-dimensional plot
2. Two-dimensional plus analytical solution plots
3. Three-dimensional surface plot

The movie-generating process then proceeds and the final frame is saved as an image at a resolution of 300 dpi to file `SchrodingersEqnPlot.png`. A detailed description of the code will not be provided as ample comments are included (see Listing A.6.4).

```
%
% Code to generate Schrodingers equation movie
% Run after pde_1_main.m
% Generates animation movies
% User should select from choices presented in dialog boxes!
%
  avi_fName='SchrodingerEqn_01.avi'; % Movie file name
  if exist(avi_fName) ~= 0
    fprintf('File: %s, already exists!\n\n', avi_fName);
    fileMsg=sprintf('File: %s, already exists - Choose to:', ...
                    avi_fName);
    namChoice=menu(fileMsg,'Overwrite','New Name');
    if namChoice==1
```

```
      delete(avi_fName);
    else
      newName=inputdlg('Movie file name:', ...
                       'File Name Request',1,{avi_fName});
      if isempty(newName{1})
        delete(avi_fName);
      else
        [pathstr,name,ext,versn]=fileparts(newName{1});
        if strcmp('.avi',ext)==0
          avi_fName=strcat(newName{1},'.avi');
        else
          avi_fName=newName{1};
        end
      end
      if exist(avi_fName)
        delete(avi_fName);
      end
    end
  end
%
% Set Plot Type: 1=2D, 2=2D + Analytic, 3=3D
  plotType=menu([{'Cubic Schrodinger Equation Solution.'}, ...
           {'Choose Plot Type:'}],'2D', ...
            '2D+Analytical Solution','3D');
%
% Get screen size
  scrsz=get(0,'ScreenSize');
%
% Define required position for figure
  rect=[0 scrsz(4)/2 scrsz(3)/2 scrsz(4)/2];
%
% Create figure at required position
  h1=figure('Position',rect);
  set(h1,'DoubleBuffer','on');
%
% Define required dimensions of image  to be saved
% within figure
  frameRect=[0 0 scrsz(3)/2 scrsz(4)/2];
%
% avi movie parameters
  fps=2;                     % Frames per second
  aviQuality=100;            % Image quality, 0-100 (100 best)
  aviCompression='Cinepak'; % Image compression mode - see
                             % help
  aviMov=avifile(avi_fName,'compression',aviCompression, ...
         'quality', aviQuality);
%
```

```
% Movie calculations
% Plot image every frameStep
  if plotType==1 | plotType==2
    frameStep=10;
  else
    frameStep=10;
  end
  qq=frameStep;               % Set frame skip counter
%
% Set up animation parameters
  [mm nn]=size(v2w2);
  nframes=ceil(mm/frameStep); % Number of frames in the movie
  Frames=moviein(nframes);    % Initialize the matrix 'Frames'
  frameCount=0;               % Initialise frame counter
  u1=zeros(mm,n);
  for it=1:mm
    if qq==0
      frameCount=frameCount+1;
      if plotType==1 | plotType==2
%
%       2D Plot
        if plotType==1
          h2=plot(x,v2w2(it,:));
          title('CSE - numerical solution.');
        else
          h2=plot(x,v2w2(it,:),'o',x,v2w2_anal(it,:),'-');
          title('CSE: o - numerical; solid - analytical.');
        end
        xlabel('x')
        ylabel('u(x,t)')
        set(gca,'XLim',[-10 40],'YLim', [0 2]);
        grid on
      else
%
%     Update u1 for each frame
      if it <= frameStep      % First frame
        u1(it,:)=v2w2(it,:);
      else                    % Subsequent frames
        u1(it-frameStep:it,:)=v2w2(it-frameStep:it,:);
      end
      h2=surfl(x,t,u1, 'light');
      shading interp
      title('Cubic Schrodinger Equation');
      set(get(gca,'XLabel'),'String','space, x')
      set(get(gca,'YLabel'),'String','time, t')
      set(get(gca,'ZLabel'),'String','dependent variable,
          u(t,x)')
```

```
        set(gca,'XLim',[-10 40],'YLim',[0 30],'ZLim', [0 2]);
        axis tight
        view(-10,45)
        colormap('cool');   % set color map
      end
      Frames=getframe(h1, frameRect); % Create image in Frames
                                      % array
      aviMov=addframe(aviMov,Frames); % Add frame to avi movie
                                      % file
      fprintf('\n    Plot number: %d created', frameCount);
      qq=frameStep;  % Reset qq
    end
      qq=qq-1;
      aviMov=close(aviMov); % Close avi movie file
      fprintf('\navi movie created in file: %s \n', avi_fName);
%
% Print image to file
  print -dpng -r300 SchrodingersEqnPlot
```

Listing A.6.4. Code to be used following a Schrödinger equation simulation run to produce a variety of 2D and 3D plots

An example 3D plot is shown in Chapter 6.

EXAMPLE – KdV EQUATION MOVIE

The following code generates an animation movie file in AVI format and utilizes Matlab workspace data populated by a previous simulation run; that is, this code should be run following the KdV code detailed in Chapter 7.

Depending on whether the initial simulation run was based on a single- or two-soliton solution, the user is presented with a dialog box from which to choose either

1. Single-soliton solution, `ncase=1`
   (a) Two-dimensional plot
   (b) Two-dimensional plus analytical solution plots
   (c) Three-dimensional surface plot
2. Two-soliton solution, `ncase=2`
   (a) Two-dimensional plot
   (b) Three-dimensional waterfall plot
   (c) Three-dimensional surface plot

The file name to save the movie to is initially set to `KdV_Eqn_01.avi`. However, if this file already exists, then the user is prompted to choose to either supply a new file name or overwrite the existing file. The movie-generating process then proceeds and the final frame is saved as an image at a resolution of 300 dpi to file `KdVEqnPlot.png`. A detailed description of the code will not be provided as ample comments are included (see Listing A.6.5).

```
%
% Code to generate Korteweg de Vries equation movie
% Run after pde_1_main.m
% Generates animation movies
% User should select from choices presented in dialog boxes!
%
% Get screen size
  scrsz=get(0,'ScreenSize');
%
% Define required position for figure
  rect=[0 scrsz(4) scrsz(3)/2 scrsz(4)/3];
%
% Create figure at required position
  h1=figure('Position',rect);
  set(h1,'DoubleBuffer','on');
%
% Define required dimensions of image  to be saved within
% figure
  frameRect=[0 0 scrsz(3)/2 scrsz(4)/3];
  plotType=1;       % Set plot type, 1=2D, 2=3D
  if ncase==1
  plotType=menu([{'KdV Equation - Single soliton solution.'}, ...
           {'Choose Plot Type:'}], ...
           '2D','2D+Analytical Solution','3D');
  else
    plotType=menu([{'KdV Equation - Two soliton solution.'}, ...
             {'Choose Plot Type:'}], ...
             '2D','3D Surface', '3D waterfall');
  end
%
% Set up animation parameters
  [mm nn]=size(u);
  if ncase==2 && plotType==3
    frameSteps=floor(mm/20);
  else
    frameSteps=2;
  end
%
% Plot image every frameSteps
  nframes=ceil(mm/frameSteps);   % Number of frames in the
                                 % movie
  Frames=moviein(nframes);       % Initialize the matrix
                                 % 'Frames'
  avi_fName='KdV_Eqn_01.avi';    % Movie file name
  if exist(avi_fName, 'file') ~= 0
    fprintf('File: %s, already exists!\n\n', avi_fName);
```

```
    fileMsg=sprintf('File: %s, already exists - Choose to:', ...
                    avi_fName);
    namChoice=menu(fileMsg,'Overwrite','New Name');
    if namChoice==1
      delete(avi_fName);
    else
    newName=inputdlg('Movie file name:', ...
                     'File Name Request',1,{avi_fName});
    if isempty(newName{1})
      delete(avi_fName);
    else
      [pathstr,name,ext,versn]=fileparts(newName{1});
      if strcmp('.avi',ext)==0
        avi_fName=strcat(newName{1},'.avi');
      else
        avi_fName=newName{1};
      end
    end
    if exist(avi_fName, 'file')
      delete(avi_fName);
    end
  end
%
% avi movie parameters
  fps=1;                        % Frames per second
  aviQuality=100;               % Image quality, 0-100
                                % (100 best)
%
% aviCompression='Cinepak';     % Image compression mode - see
                                % help
                                % Options On Windows:
                                % 'Indeo3'
                                % 'Indeo5'
                                % 'Cinepak'
                                % 'MSVC'
                                % 'None'
  aviCompression='Indeo5'; % Image compression mode - Large
                           % file!
  aviMov=avifile(avi_fName,'compression',aviCompression, ...
         'quality', aviQuality);
%
% Movie calculations
  qq=1;              % Set frame skip counter
  frameCount=0;      % Initialise frame counter
  u1=zeros(mm,n);
  hbar=waitbar(0,'please wait...');
  for it=1:mm
```

```
    qq=qq-1;
    if qq==0
      frameCount=frameCount+1;
      if plotType==1 || (plotType==2 && ncase==1)
%
%       2D Plot
        if plotType==1
          h2D=plot(x,u(it,:));
        else
          h2D=plot(x,u(it,:),'o',x,u_anal(it,:),'-');
          legend('Numerical solution', ...
                 'Analytical solution', ...
                 'location', 'northeast');
        end
        xlabel('x')
        ylabel('dependent variable, u(x,t)')
        if ncase==1
          axis([-30 70 0 0.6]);
        else
          axis([-30 70 0 1.0]);
        end
          set(h2D,'LineWidth',2)
          title('Korteweg de Vries (KdV) equation');
          grid on
      else
%
%     Update u1 for each frame
      if it <= frameSteps      % First frame
        u1(it,:)=u(it,:);
      else                     % Subsequent frames
        u1(it-frameSteps:it,:)=u(it-frameSteps:it,:);
      end
      if plotType==2
        h3D=surfl(x,t,u1, 'light');
        shading interp
        axis tight
      else
        h3D=waterfall(x,t(it),u1(it,:));
        set(h3D,'LineWidth',2);
        axis([xl xu t0 tf 0 1])
        hold on;
      end
      title('Korteweg deVries (KdV) equation');
      set(get(gca,'XLabel'),'String','space, x')
      set(get(gca,'YLabel'),'String','time, t')
      set(get(gca,'ZLabel'),'String', ...
          'dependent variable,u(t,x)')
```

```
      view(-10,70);
      colormap('cool');   % set color map
    end
    Frames=getframe(h1, frameRect);   % Create image in Frames
                                      % array
    aviMov=addframe(aviMov,Frames);   % Add frame to avi movie
                                      % file
    disp(['    Plot number: ', num2str(frameCount),...
          ' created']);
    qq=frameSteps;  % Reset qq
    waitbar(it/mm);
  end
  close(hbar);
  aviMov=close(aviMov); % Close avi movie file
  fprintf('\navi movie created in file: %s \n', avi_fName);
%
% Print image to file
% print -dpng -r300 KdVEqnPlot
```

Listing A.6.5. Code to be used following a KdV equation simulation run to produce a variety of 2D and 3D plots

An example of 3D plot is shown in Chapter 7.

ANIMATED GIF FILES

As an alternative to generating AVI movie files, it is also possible to generate *animated GIF files*. These files, which also produce a movie, are usually much smaller than AVI files and are particularly useful for displaying simulation results in Web pages. The process is simple and straightforward and involves saving image files at each simulation step and then assembling them into a composite animated GIF file. The one drawback is that GIF files cannot be generated directly from within Matlab using standard functions. However, many graphics packages provide this facility and a simple search will reveal a number of *freeware* and/or *shareware* utilities that can be downloaded from the Web and can perform the GIF animation task, for example, *VideoMach-standard*, a shareware product that can be downloaded from `http://www.gromada.com/videomach.html`.

EXAMPLE – 3D LAPLACE EQUATION MOVIE

The GIF animation procedure will be illustrated using Matlab code that can be used following the 3D Laplace equation code detailed in Chapter 11. This is an interesting graphical image problem as it relates to the solution of a 3D Laplace equation problem whereby the solution is obtained by converting it into a *pseudo time-dependent 3D problem*; that is, a 4D problem, and letting the simulation proceed to steady state

that represents the final converged solution. Now, as mentioned in Chapter 11, it is not possible to view a 4D problem in the usual way, so we have to find a different approach.

The way we approach this difficult problem is to use the volume `slice` function in Matlab to produce a sequence of slices through the 3D volume at each time step and saving the corresponding image files. We then use these images to generate an animated sequence. This produces a clear idea of what is happening.

The following code generates a sequence of images from Matlab workspace data populated by a previous simulation run; that is, this code should be run following the Laplace 3D simulation code detailed in Chapter 11. A detailed description of the code will not be provided as ample comments are included (Listing A.6.6).

```
%
% Run after pde_1_main.m
% Generates a sequence of images from Matlab workspace data
% populated by a previous simulation run
  delta=1/(nx-1);
  [x,y,z]=meshgrid(0:delta:1,0:delta:1,0:delta:1);
  figure(1); clf; hold on; axis equal; axis on;
%
% Set suitable color map
  colormap(jet(16));
%
% Set Shading
  shading faceted
  for it=1:nout
%
%   Set 3D data at step 'it'
    w(:,:,:)=u(it,:,:,:);
%
%   Define slice planes to include in plot
    xslice=[0.1,.5,0.9]; yslice=[0]; zslice=[0,1];
%
%   Create slice image
    slice(x,y,z,w,xslice,yslice,zslice)
    xlabel('x');ylabel('y');zlabel('z');
    tStr=sprintf('Step No: %d',it);
    title(tStr);
    caxis([0,1]);
%
%   Include colorbar - scale
    colorbar
%
%   Set viewpoint
```

```
    view(-161,8);
%
%   Set file name with numeric increment
    file=sprintf('Movie_frame%d.png', 1000+it);
    disp(sprintf('Saving to %s', file));
%
%   Save image file
    print('-dpng','-zbuffer','-r100',file);
  end
```

Listing A.6.6. Code to be used following a 3D Laplace equation simulation run to produce a sequence of 3D `slice` plots

The code given in Listing A.6.6 generates in a set of image files that will form the basis of the animated GIF file. Shown in Figure A.6.1 is a composite image showing the results of steps 2–5 (where the simulation was previously run with a step of 0.04). It will be observed that at each step the color coding of each slice changes.

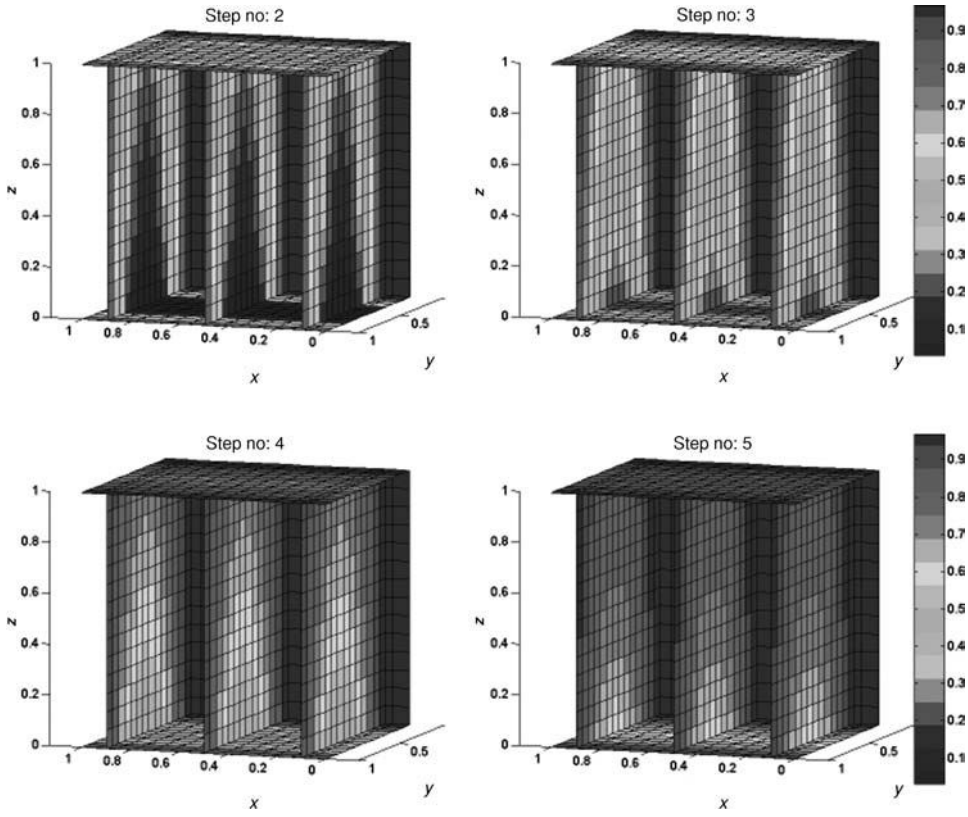

Figure A.6.1. Composite image showing results of steps 2 to 5 from Listing A.6.6 code run following `pde_1_main` of Chapter 11. (See also color plate in Color Plate Section)

Eventually, when fully converged to the solution (after approximately 12 steps), the images all become the same color corresponding to $u(x, y, z) = 1$.

The next step is to assemble into an animated GIF file using a suitable utility, such as the one mentioned earlier. This process is straightforward and most packages provide the facility to resize the images and set the number of frames to be displayed per second and other parameters that enable the user to achieve the desired overall aesthetic appearance of the animation.

EXAMPLE – SPHERICAL DIFFUSION EQUATION MOVIE

This example relates to the spherical coordinates diffusion problem described in Chapter 14. In order to display the results properly, we therefore need to perform a polar to Cartesian conversion on these data before creating the image. We will illustrate this by reference to the simulation performed by `pde_2_main`: first by the code to generate static plots of the inhomogeneous term $f(r, \theta, \phi)$, and second by the code to generate an animated GIF file of the simulation results $u(r, \theta, t)$.

The code given in Listing A.6.7 generates two static images side by side of the inhomogeneous function: one for the r–ϕ plane and one for the r–θ plane.

```
% Creates x-y and x-z plane polar plots of inhomogeneous 2D
% Gaussian source term 'f(r,theta,phi)' from pde_2_main.m
% code of Chapter 14
% Note: f is symmetrical about phi.
%
% Clear previous files
  clc; clear all
%
% Model parameters
  D=0.1;
  r0=5; th0=2*pi;
  std_0=1.0; std_pi2=2.0;
  tau=1.0;
  NN=101;
%
% Radial grid
  nr=NN; r=linspace(0,r0,nr);
%
% Angular grid
  nth=NN; th=linspace(0,th0,nth);
%
% Create polar mesh
  [R, Theta]=meshgrid(r, th);
  [X,Y]=pol2cart(Theta,R); % Convert to Cartesian coordinates
  [X,Z]=pol2cart(Theta,R); % Convert to Cartesian coordinates
%
```

```
% Calculate Inhomogeneous function data
  for i=1:nr, for j=1:nth
    F1(i,j)=exp(-(r(i)/std_0 )^2); % Symmetrical about phi
    F2(i,j)=exp(-(r(i)*sin(th(j))/std_pi2)^2 ...
               -(r(i)*cos(th(j))/std_0  )^2);
    end, end
%
% Convert function data to be suitable for use with
% the meshgrid using cubic interpolation
  Fxy=interp2(th,r,F1,Theta,R,'cubic');
  Fxz=interp2(th,r,F2,Theta,R,'cubic');
%
% Plot r-phi (x-y) plane of inhomogeneous source,
% f(r,theta,phi) over four quadrants (uses Cartesian
% coordinate data)
  figure()
  subplot(1,2,1)
  surf(X, Y, Fxy); shading interp;
  axis square; axis tight
  caxis([0,1]); colormap jet
  xlabel('X'); ylabel('Y'); zlabel('f(r,\theta)');
  title([{'f(r,\theta,\phi) - (r- \phi) Plane'}]);
  view([0,90]);
%
% Plot r-theta (x-z) plane of inhomogeneous source,
% f(r,theta,phi) over four quadrants  (uses Cartesian
% coordinate data)
  subplot(1,2,2)
  surf(X, Z, Fxz); shading interp;
  axis square; axis tight
  caxis([0,1]); colormap jet
  xlabel('X'); ylabel('Z'); zlabel('f(r,\theta,phi)');
  title([{'f(r,\theta,\phi) - (r- \theta) Plane'}]);
  view([0,90]);
% print -dpng  -r300 func_f1.png % Save image file
```

Listing A.6.7. Code to generate r–ϕ plane and r–θ plane plots of the inhomogeneous function $f(r, \theta, \phi)$

We can note the following points about this code section:

1. A polar coordinate mesh is created, based on grid variables `r` and `th`, which is then converted to Cartesian coordinates for subsequent use by the plot function `surf`.

```
%
% Create polar mesh
  [R, Theta]=meshgrid(r, th);
  [X,Y]=pol2cart(Theta,R); % Convert to Cartesian coordinates
  [X,Z]=pol2cart(Theta,R); % Convert to Cartesian coordinates
```

2. The inhomogeneous function is evaluated over the r–ϕ plane and the r–θ plane.

```
%
% Calculate Inhomogeneous function data
  for i=1:nr, for j=1:nth
    F1(i,j)=exp(-(r(i)/std_0 )^2); % Symmetrical about phi
    F2(i,j)=exp(-(r(i)*sin(th(j))/std_pi2)^2 ...
              -(r(i)*cos(th(j))/std_0  )^2);
    end, end
```

Note: The preceding code could have been implemented more efficiently using the following vectorized code, but we wish to emphasize the use of polar coordinates.

```
%
% Calculate Inhomogeneous function data
  F1=exp(-(Y/std_0).^2   -(X/std_0  ).^2);
  F2=exp(-(Z/std_pi2).^2 -(X/std_0  ).^2);
```

3. The function f data are now converted using the Matlab function `interp2` with *cubic interpolation* so as to be suitable for use with the meshes `[X,Y]` and `[X,Z]`. This step is required because the data `F1` and `F2` correspond to grid variables `r` and `th`, not the `meshgrid` variables `X`, `Y`, and `Z`.

```
%
% Convert function data to be suitable for use with the
% meshgrid using cubic interpolation
  Fxy=interp2(th,r,F1,Theta,R,'cubic');
  Fxz=interp2(th,r,F2,Theta,R,'cubic');
```

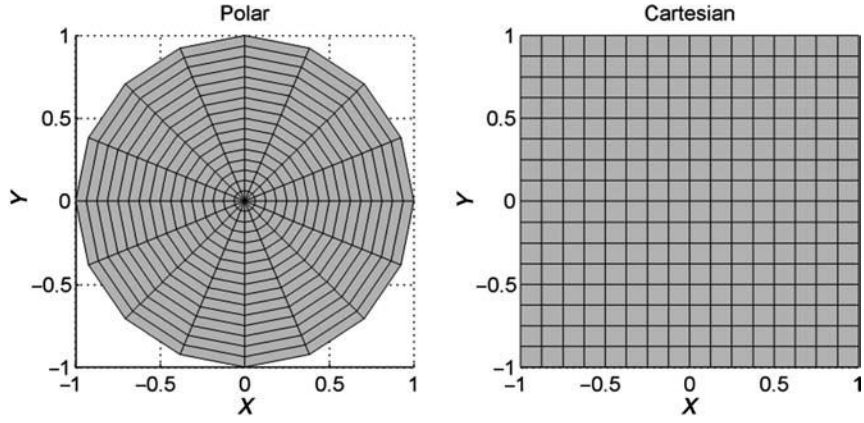

Figure A.6.2. Polar and Cartesian grid plots. (See also color plate in Color Plate Section)

That this last operation is necessary is seen from Figure A.6.2, which illustrates that conversion from a polar coordinate grid to a Cartesian coordinate grid requires interpolation; that is, there is not a simple mapping from each polar grid point to a corresponding Cartesian grid point or vice versa. Thus, to obtain $f(x, y)$ on the Cartesian grid at points (x_i, y_j), first we calculate the corresponding locations on the polar grid, $r_a = \sqrt{(x_i^2 + y_j^2)}$ and $\theta_b = \arctan(y_j / x_i)$, from which we obtain $f(x_i, y_j) = f(r_a, \theta_b)$ by interpolation of the function $f(r, \theta)$. The result is shown in Figure A.6.3.

The remainder of the code should be understandable from the embedded comments. Having demonstrated how polar coordinate data can be converted to Cartesian coordinate data and plotted using the Matlab `surf` function, we now proceed to the main task of this example, which is to create an animation of the spherical coordinate diffusion process described in Chapter 14.

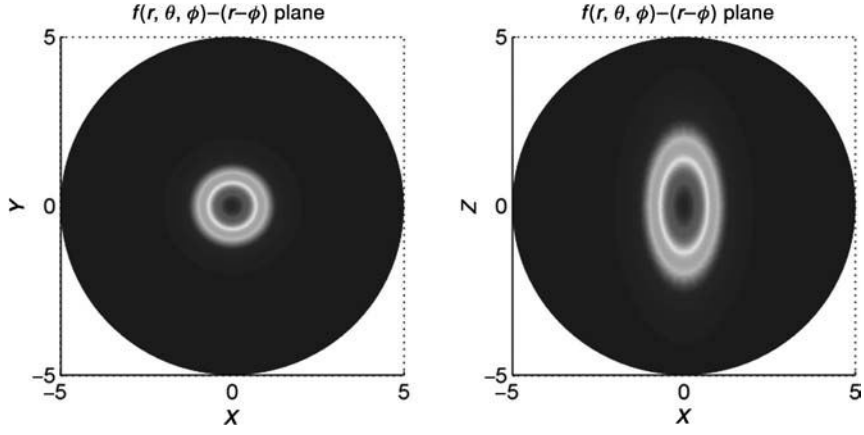

Figure A.6.3. Plots from Listing A.6.7 code showing Chapter 14 inhomogeneous function $f(r, \theta, \phi)$ data; *left* – $r-\phi$ plane, *right* – $r-\theta$ plane. (See also color plate in Color Plate Section)

The code given in Listing A.6.8 generates a sequence of images from Matlab workspace data populated by a previous simulation run; that is, this code should be run following the diffusion equation in spherical coordinates simulation code pde_2_main detailed in Chapter 14.

```
% Run after pde_2_main.m code from Chapter 14
% Generates a sequence of images from Matlab workspace data
% populated by a previous simulation run. Images can then be
% used to create an animated gif file.
%
% Set Up Plotting Vars
  r1=linspace(-r0,r0,2*nr);
  th1=linspace(0,2*pi,2*nr);
%
% Create mesh for single quadrant image
  [R1, Theta1]=meshgrid(r, th);
%
% Create mesh for four-quadrant mesh
  [R2, Theta2]=meshgrid(r1, th1);
%
% Convert meshes to Cartesian form
  [X1,Z1]=pol2cart(Theta1,R1);
  [X2,Z2]=pol2cart(Theta2,R2);
  plotType = 2; % Set plot type: 1=single-quadrant,
                % 2=four-quadrant
%
% Plot dependant variable, u(r,theta) over one quadrant
% at t=tf
  figure();
  for it=1:1:nout
%
% Create 2D matrix of simulation output data at t(it)
  U0(:,:)=u(it,:,:);
%
% Convert results data to be suitable for use with the
% meshgrid using cubic interpolation
  U1=interp2(th,r,U0,Theta1,R1,'cubic');
  if plotType == 1
%
%   Create plot from single quadrant data
    surf(Z1, X1, U1); hold on;
  else
%
%   Create four-quadrant data set from single quadrant data
%   by flipping (vertical/horizontal) and shifting operations
```

```
    U2=quadJoin(U1);
%
%   Create plot from four-quadrant data
    surf(Z2, X2, U2); hold on;
  end
  shading  interp
  colormap jet
  axis tight;  axis square
  colorbar(128); caxis([0,1]);  % Add color key
  xlabel('X'); ylabel('Z'); zlabel('u(r,\theta)');
  tmpStr=sprintf('at t=%2.1f',t(it));  % Prepare title text
  title([{'u(r,\theta), r=(X^2+Z^2)^{0.5},\theta=arctan(Z/X)'}
         {tmpStr}]);
  view([0,90]);  % Set azimuth and elevation
%
% Set file name with numeric increment
  file=sprintf('sphericalMov%d.png', 100+it);
  disp(sprintf('Saving image to file: %s', file));
%
% Save image file
  print( '-dpng', '-zbuffer', '-r100', file);
end
```

Listing A.6.8. Code to be used following the spherical coordinate diffusion process simulation described in Chapter 14. It produces a sequence of r–θ plane plots of the result $u(r, \theta, t)$

We can note the following points about this code section:

1. Because the problem is symmetrical about ϕ, the simulation only needs to generate results $u(r, \theta, \phi, t)$ for one quadrant of the r–θ plane. The remaining quadrants can be obtained by a simple transformation of this single quadrant. By setting `plotType=1`, a sequence of single-quadrant images is generated or by setting `plotType=2`, a sequence of four-quadrant images is generated, as shown in Figure A.6.4.

```
%
plotType=2; % Set plot type: 1=single-quadrant,2=four-quadrant
```

2. The following call to function `quadJoin`, discussed subsequently, performs the necessary transformations on the single-quadrant data `U1` to create the four-quadrant data `U2`.

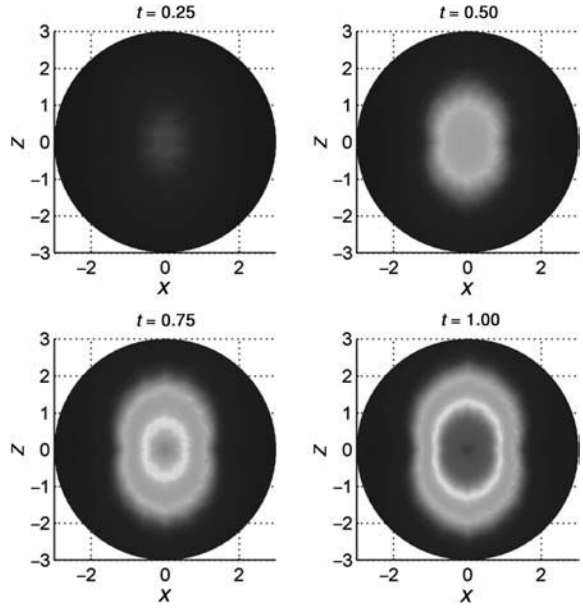

Figure A.6.4. Plots from Listing A.6.8. code showing time sequence of $r-\theta$ plane results from program `pde_2_main` of Chapter 14. (See also color plate in Color Plate Section)

```
%
%   Create four-quadrant data set from single quadrant data
%   by flipping (vertical/horizontal) and shifting operations
    U2=quadJoin(U1);
```

The function `quadJoin` given in Listing A.6.9 performs combinations of horizontal/vertical flips/shifts on single-quadrant data and combines the results to form a set of corresponding four-quadrant data.

```
  function B=quadJoin(A)
%
% Creates a four-quadrant image from a single quadrant image
% To be used with sphericalPlot.m
  [m,n]=size(A);
  B=zeros(2*n,2*n); % Pre-allocate array for four-quadrant
                    % data
  for i=1:n
    for j=1:n
      B(i      , n+1-j)= A(i,j);   % Top left quadrant
```

```
    B(i      , n+j  )= A(i,j);   % Top right quadrant
    B(2*n+1-i, n+1-j)= A(i,j);   % Bottom left quadrant
    B(2*n+1-i, n+j  )= A(i,j);   % Bottom right quadrant
  end
end
```

Listing A.6.9. Code for function `quadJoin`

REFERENCES

[1] Biran, A. and M. Briener (1998), *Matlab 5 for Engineers*, Addison-Wesley, UK
[2] Hanselman, D. and B. Littlefield (2004), *Mastering MATLAB 7*, Prentice Hall, New Jersey
[3] Part-Enander, E. and A. Sjoberg (1999), *The Matlab 5 Handbook*, Addison-Wesley, UK