Imperial College London

Department of Mechanical Engineering
South Kensington, London, SW7 2AZ

ME4 Computational Fluid Dynamics

1 Dimensional Convection Diffusion Equation

Project Assignment I

Author – Henry Hart
CID – 01190775
Course Leader – Prof. Pavlos Aleiferis
Date – 16th January 2020
Word Count – 3,548
Word Limit – 4,000

Abstract

Computational Fluid Dynamics (CFD) is a powerful tool that is used to solve some real engineering problems more cost effectively than the alternatives. This report compares three of the most basic discretisation systems used in CFD codes against an analytical solution to a simple convection diffusion problem. These three schemes are Central Differencing (CDS), Upwind Differencing (UDS) and Power Law Differencing (PLDS).

CDS has the disadvantage that it is unstable at high local Peclet numbers, while UDS has the disadvantage of being inaccurate due to amplified false diffusion, especially when compared to alternatives at low Peclet numbers. This report finds that PLDS outperforms CDS and UDS in terms of error across a range of Peclet numbers and grid resolutions. PLDS and UDS guarantee a stable solution, so reflecting robust commercial CFD codes.

The only drawback to PLDS demonstrated herein, in comparison to CDS and UDS, is the longer computation times inherent in this more complicated scheme for the same grid resolution and physical parameters.

Contents

1. Introduction

As part of the Master's level module 'Computational Fluid Dynamics' for Mechanical Engineering at Imperial College, students are required to demonstrate their understanding of the taught material [1] by submitting two pieces of coursework. This report constitutes the discussion of the results for the first coursework submission. This report is intended for assessment in parallel with the source code [2] as required by the assignment [3]. The source code was written by the author in the $MATLAB$ environment and used to produce results in graphical form. The problem to be investigated is a 1-dimensional convection diffusion problem. This has been covered in lectures [1] and alternative insight is given in the previous iteration of notes for the CFD course [4].

This report will outline the nature of the physical problem whose governing ODE has an analytical solution. This analytical solution will be discussed in relation to its dependence on the non-dimensional Peclet number. Computational methods used to discretise and solve the governing ODE will then be discussed in terms of their relative advantages and disadvantages. The results of these schemes are then compared, both qualitatively and then with respect to their respective errors, to demonstrate the aforementioned advantages and disadvantages.

2. Physical Problem Overview

The problem of interest is one where a property ϕ is transported by convection, controlled by velocity u, and diffusion, controlled by diffusion coefficient Γ_ϕ. The layout of the physical problem is shown in figure 2.1.

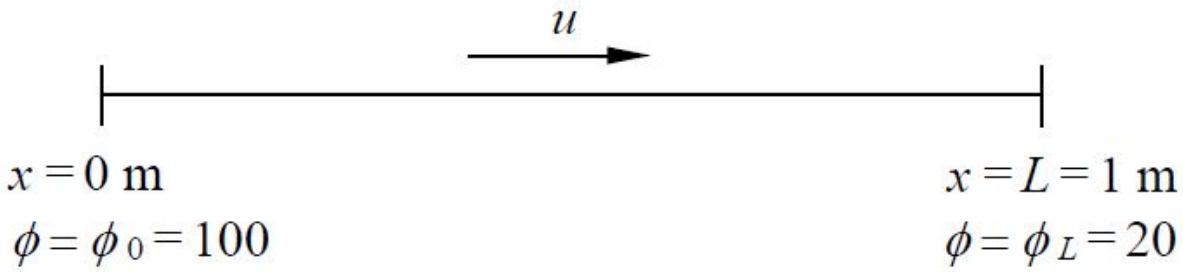

Figure 2.1: Layout of Physical Problem [3];

The general form of the convection diffusion conservation equation is given below:

$$\frac{\partial}{\partial t}(\rho\phi) + \frac{\partial}{\partial x_j}(\rho u_j \phi) = \frac{\partial}{\partial x_j}(\Gamma_\phi \frac{\partial \phi}{\partial x_j}) + S_\phi \tag{2.1}$$

The terms from left to right in equation 2.1 represent rate of change, convection, diffusion and the source term. In this problem, steady state is assumed, and the existence of a source in the domain is neglected. As such, the unsteady and source terms can be taken as zero. Since the domain is only one dimensional, change in the y and z directions can be taken as zero, i.e.:

$$\frac{\partial}{\partial t}(\psi) = \frac{\partial}{\partial y}(\psi) = \frac{\partial}{\partial z}(\psi) = 0 \quad for \;\; any \;\; \psi \tag{2.2}$$

This leads to the steady 1-dimensional convection diffusion equation below, with differentials written with 'straight' d as all variables are functions of x only:

$$\frac{d}{dx}(\rho u \phi) = \frac{d}{dx}(\Gamma_\phi \frac{d\phi}{dx}) \tag{2.3}$$

This problem has two boundary conditions, both of which are Dirichlet conditions where the value of ϕ is specified at the boundary. A physical interpretation of this problem therefore

could be two large reservoirs of fluid each kept at a prescribed temperature and linked by a pipe which transports the fluid at a velocity u. When $u = 0$, this problem becomes a pure convection problem, and it can be shown that this produces a constant variation in ϕ along the domain, which is intuitively true (see figure 3.1). When the effect of convection dominates over diffusion, i.e. $\rho u L >> \Gamma_\phi$, one would expect the temperature from the 'upwind' direction to dominate the domain, and this report will validate this intuition with analytical and computational solutions (sections 3 & 4) to the problem at different global Peclet numbers, as defined below:

$$Pe_{global} = \frac{\rho u L}{\Gamma_\phi} \tag{2.4}$$

3. Analytical Solution

3.1 ODE Solution

The exact analytical solution to the problem with boundary conditions given in figure 2.1 and characteristic equation 2.3 is given in equation 3.1. This report will not derive this result, however it can be easily validated by inserting boundary values of x to retrieve boundary values of ϕ and confirming that the lhs and rhs of equation 2.3 are equal for the value of ϕ given below:

$$\phi = \phi_0 + \frac{e^{xPe_{global}/L} - 1}{e^{Pe_{global}} - 1}(\phi_L - \phi_0) \tag{3.1}$$

Given that the length and boundary conditions are specified as constant (figure 2.1), ϕ is a function only of the global Peclet number and x. Analytical solutions to the problem are shown in figure 3.1 for different Peclet numbers. The Peclet numbers of adjacent solutions shown differ by 1. Careful inspection of equation 3.1 reveals that the result for $Pe_{global} = 0$ is undefined unless the limit of the fraction of exponentials is taken. In the code [2], a small value ($Pe_{global} = 0.001$) has been used to represent this limit.

3.2 Turbulence

Turbulence, and it's modelling in the context of Computational Fluid Dynamics, is central to Fluid Mechanics research. In many problems, it is important to model the turbulent viscosity in order to more accurately compute drag. In this problem, turbulence would act to increase the value of the diffusion coefficient differentially along the domain. Fortunately, this is of no concern to this scenario due to the problem being defined as 1-dimensional. The incompressible continuity equation is given below:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{3.2}$$

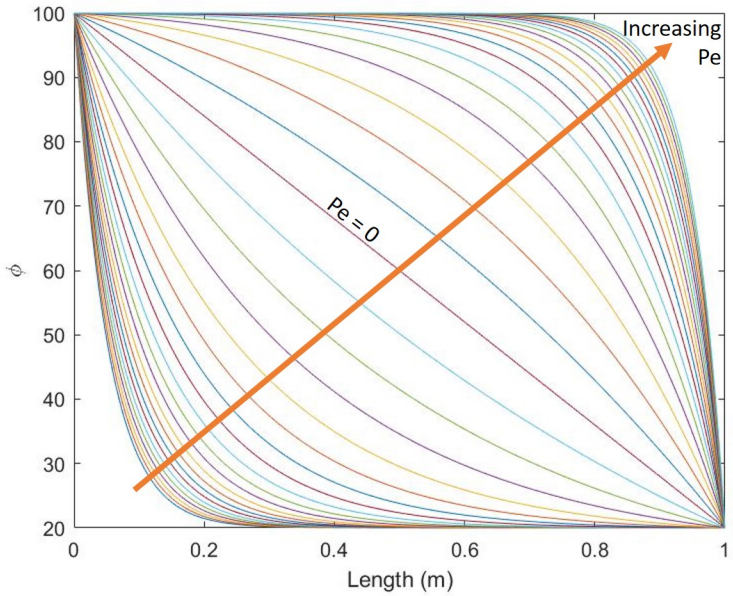

Figure 3.1: Variation of ϕ with Global Peclet Number;

The Reynolds decomposition is given below, where $\bar{u}$ is the time average and u' is the fluctuating term [5]:

$$u = \bar{u} + u'; \quad v = \bar{v} + v'; \quad w = \bar{w} + w' \tag{3.3}$$

Time averaging the laminar continuity equation and subtracting it from a Reynolds averaged continuity equation implies the following formulation of the fluctuating continuity equation:

$$\frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} + \frac{\partial w'}{\partial z} = 0 \tag{3.4}$$

Since $v = v' = w = w' = 0$ due to 1 dimensionality, this implies that u' is constant along the domain implying no variability in the diffusion coefficient.

4. Computational Solutions

The finite volume method of computing results for the conservation equation defined in equation 2.3 begins by integrating the equation over the grid control volume corresponding to cell P (see figure 4.1).

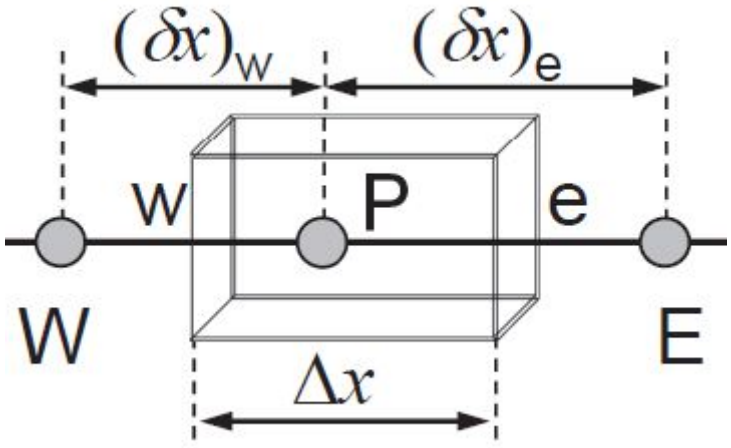

Figure 4.1: Control Volume for Finite Volume Grid [1, p. 191];

Integration of equation 2.3 and application of Gauss' divergence theorem [6] is shown in equation 4.1 which yields equation 4.2 when discretised. $\hat{n}$ represents the outward facing normal vector for each surface portion.

$$\iiint_V \frac{d(\rho u\phi)}{dx}dV = \oiint_S \rho\vec{u}\cdot\hat{n}\phi dS = \oiint_S \Gamma_\phi \frac{\vec{d\phi}}{dx}\cdot\hat{n}dS = \iiint_V \frac{d}{dx}\Gamma_\phi(\frac{d\phi}{dx})dV \tag{4.1}$$

$$(\rho u\phi)_e - (\rho u\phi)_w = \frac{(\Gamma_\phi)_e(\phi_E - \phi_P)}{(\delta x)_e} - \frac{(\Gamma_\phi)_w(\phi_P - \phi_W)}{(\delta x)_w} \tag{4.2}$$

This equation is not yet suitable for implementation in a code which stores values of ϕ at nodes, rather than faces. The face values (ϕ_e & ϕ_w) can be estimated using a variety of schemes, three of which are discussed as follows:

1. Central Differencing Scheme (section 4.1)
2. Upwind Differencing Scheme (section 4.2)

3. Power Law Differencing Scheme (section 4.3)

Finding nodal values of ϕ leads to a relationship between a central node and its two neighbours as shown below:

$$a_P\phi_P = a_E\phi_E + a_W\phi_W \tag{4.3}$$

The coefficients defining this relationship depend on the differencing scheme used to find face values and will be defined in the relevant sections.

4.1 Central Differencing Scheme

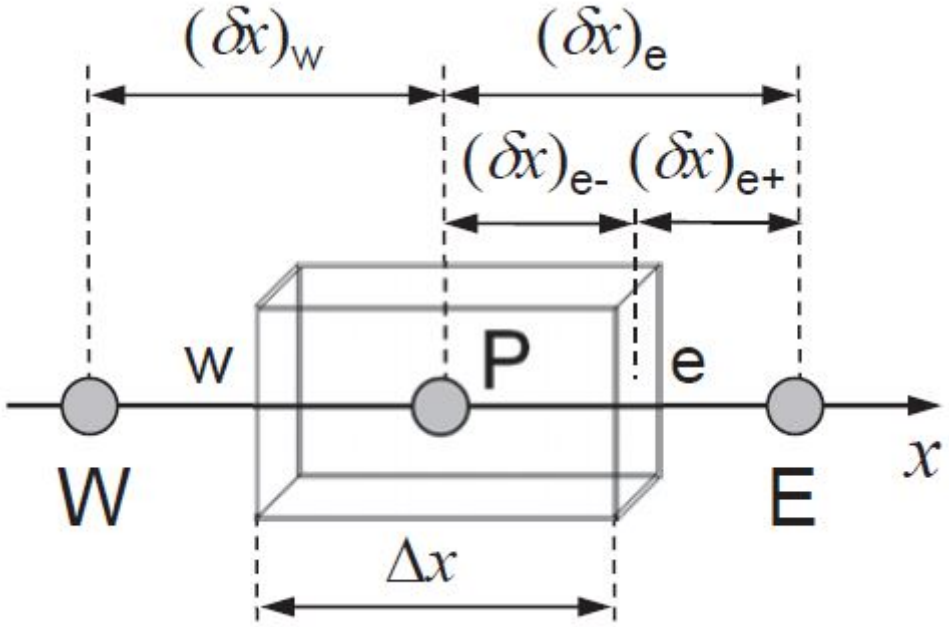

Figure 4.2: Control Volume for CDS [1, p. 224];

The central differencing scheme estimates the face value of ϕ by taking a distance weighted average of the neighbouring nodal values as shown below:

$$\phi_e = \frac{(\delta x)_{e+}}{\delta x_e}\phi_P + \frac{(\delta x)_{e-}}{\delta x_e}\phi_E; \quad \phi_w = \frac{(\delta x)_{w+}}{\delta x_w}\phi_W + \frac{(\delta x)_{w-}}{\delta x_w}\phi_P \tag{4.4}$$

This formulation of face values means that the central differencing scheme is second order accurate. This can be shown with a Taylor series expansion.

The 'half grid spacings', $(\delta x)_e-$ on the East face for example, used in equation 4.4 are defined by figure 4.2. The $+/-$ symbols imply the side of the face with respect to the direction of the x axis.

In the case of an evenly spaced grid, as will be implemented in this assignment, the

formulation in equation 4.4 simply describes taking the mean of nodal values. In this case, the coefficients defining equation 4.3 are given below:

$$a_E = \frac{(\Gamma_\phi)_e}{(\delta x)_e} - \frac{(\rho u)_e}{2}; \quad a_W = \frac{(\Gamma_\phi)_w}{(\delta x)_w} + \frac{(\rho u)_w}{2}; \quad a_P = a_E + a_W + (\rho u)_e - (\rho u)_w \tag{4.5}$$

For an incompressible flow, the mass flux terms in a_P in equation 4.5 cancel out to yield $a_P = a_E + a_W$.

Observing these coefficients, it is clear that they may become negative should the local Peclet number exceed a magnitude of 2. Negative coefficients are not desirable as it makes the solution unstable as demonstrated in section 5. The local Peclet number is defined as follows:

$$Pe_{local} = \frac{\rho u \delta x}{\Gamma_\phi} \tag{4.6}$$

4.2 Upwind Differencing Scheme

Upwind differencing scheme estimates that the face values of ϕ are given by the nodal value upwind of that face. Computationally, this is realised using a $max(A, B) \equiv \langle A, B \rangle$ function as below, which returns the maximum of its two arguments:

$$(\rho u \phi)_e = \phi_P \langle (\rho u)_e, 0 \rangle + \phi_E \langle -(\rho u)_e, 0 \rangle; \qquad (\rho u \phi)_w = \phi_W \langle (\rho u)_w, 0 \rangle + \phi_P \langle -(\rho u)_w, 0 \rangle \tag{4.7}$$

The coefficients in equation 4.3 can therefore be expressed as follows:

$$a_E = \frac{(\Gamma_\phi)_e}{(\delta x)_e} + \langle -(\rho u)_e, 0 \rangle; \quad a_W = \frac{(\Gamma_\phi)_w}{(\delta x)_w} + \langle (\rho u)_w, 0 \rangle; \quad a_P = a_E + a_W + (\rho u)_e - (\rho u)_w \tag{4.8}$$

Observation of the coefficients in equation 4.8 reveals that they cannot become negative, regardless of the value of the local Peclet number. The advantage of this is that the solution never becomes unstable which can be seen in section 5.

4.3 Power Law Differencing Scheme

PLDS is a scheme which sets diffusivity to zero for high local Peclet number flows (i.e. greater than 10), and uses a quintic function otherwise to determine the coefficients for equation 4.3 as below:

$$a_E = \frac{(\Gamma_\phi)_e}{(\delta x)_e} \langle (1 - 0.1|Pe_e|)^5, 0 \rangle + \langle -(\rho u)_e, 0 \rangle; \quad a_W = \frac{(\Gamma_\phi)_w}{(\delta x)_w} \langle (1 - 0.1|Pe_w|)^5, 0 \rangle + \langle (\rho u)_w, 0 \rangle \tag{4.9}$$

$$a_P = a_E + a_W + (\rho u)_e - (\rho u)_w \tag{4.10}$$

In high local Peclet number flows, false diffusion is a problematic numerically induced error which is remedied by tapering off the effect of diffusion in high Peclet flows. This is an underlying reason behind PLDS generally showing more accurate results, especially at high Peclet numbers (see section 5). Again, it is clear from the formulation of the coefficients in equations 4.9 and 4.10 that these cannot become negative, and so PLDS always produces stable solutions.

4.4 Tri-Diagonal Matrix Algorithm

For any choice of discretisation scheme, the result is a system of equations defined by equation 4.3. An obvious way to solve this system of equations would be a for loop which creates a matrix to be inverted. While simple and easy to programme, this approach is not efficient. The tri-diagonal matrix algorithm (TDMA) is more efficient for large solution domains. Using the notation of figure 4.1, TDMA calculates the value of ϕ at each node using its Eastern neighbour (see equation 4.11). The TDMA coefficients (P and Q) used in the equation however are calculated using coefficients determined at the Western neighbour. In practical programming terms this means that two loops are set up which run through the domain in opposite directions one after another which results in an array of ϕ, which is the ultimate output [2]. In equation 4.11, b_P is a source coefficient which is zero in this problem.

$$\phi_P = P_P \phi_E + Q_P; \quad P_P = \frac{a_E}{a_P - a_W P_W}; \quad Q_P = \frac{b_P + a_W Q_W}{a_P - a_W P_W} \tag{4.11}$$

In this problem, where the boundary conditions are both Dirichlet type, ϕ at the boundaries is known. This implies that $P = 0$ and $Q = \phi$ at the boundaries.

5. Comparison of Results

Results were calculated and plotted using the $MATLAB$ environment using the computation methods described in section 4. This section attempts to use these results to demonstrate some trademark features of the different discretisation schemes employed. In all results, Γ_ϕ and ρ will be held constant at a value of 1 (all units are standard SI, not that CFD cares) while u and Δx (= δx for an evenly spaced grid) will be varied.

5.1 Pure Results

This section will focus on the pure results ($\phi(x)$) and examine the qualitative suitability of the solutions. Section 5.2 will compare the errors in each scheme.

5.1.1 CDS Instability

As described in section 4.1, the coefficients for equation 4.3 formulated using CDS can turn negative for high local Peclet numbers.

To demonstrate this, let us begin with a fairly forgiving example with 11 nodes (including boundaries) and $u = -5$, which has a local Peclet number magnitude of 0.5. This is shown in figure 5.1.

Now if the local Peclet number's magnitude is increased to above 2, the solution becomes unstable and obviously incorrect as a result, as shown in figure 5.2.

This instability can be resolved for the same velocity by reducing the grid resolution to restore a small local Peclet number as shown in figure 5.3.

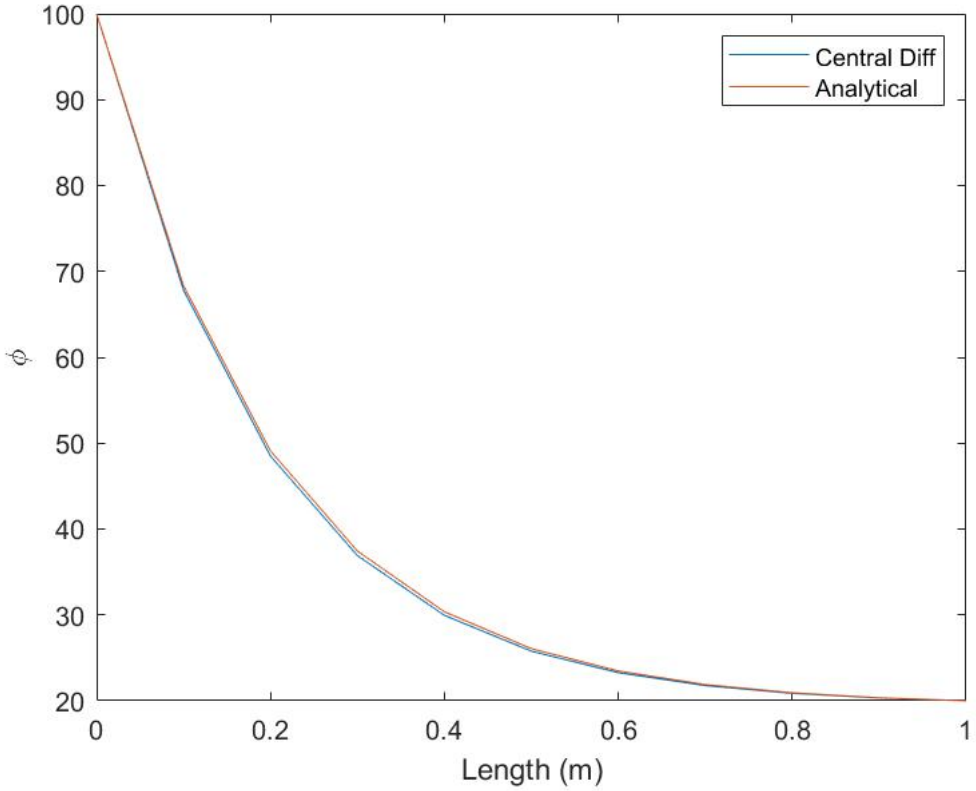

Figure 5.1: Central Differencing Scheme: $u = -5, \delta x = 0.1, |Pe_{local}| = 0.5$*;*

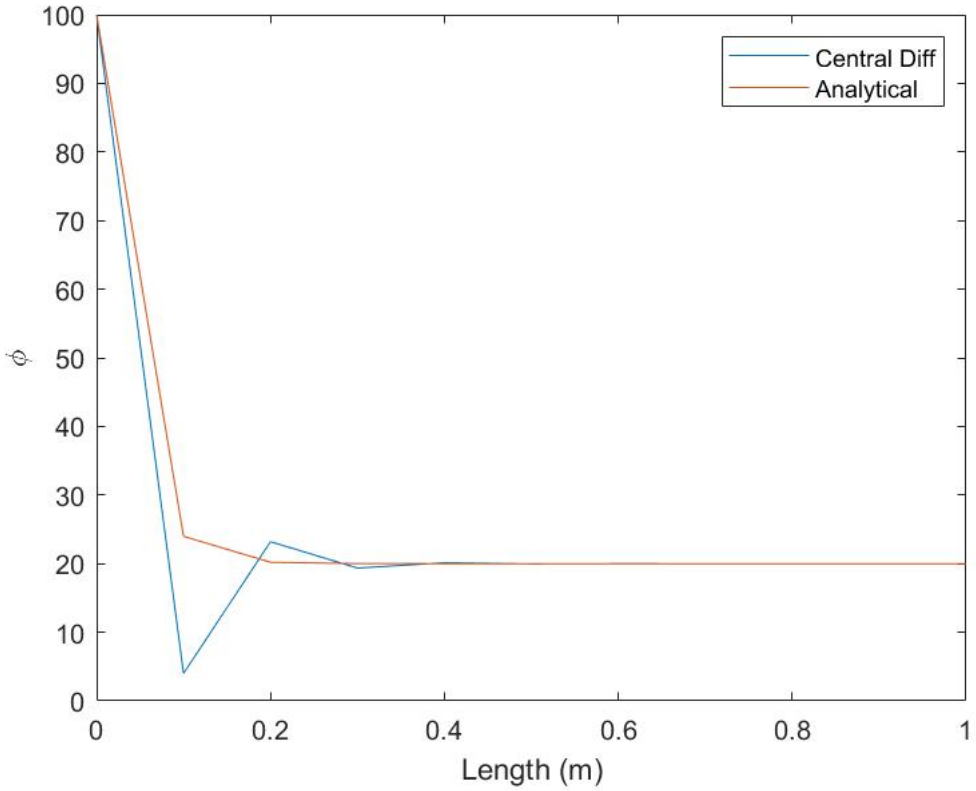

Figure 5.2: Central Differencing Scheme: $u = -30, \delta x = 0.1, |Pe_{local}| = 3$*;*

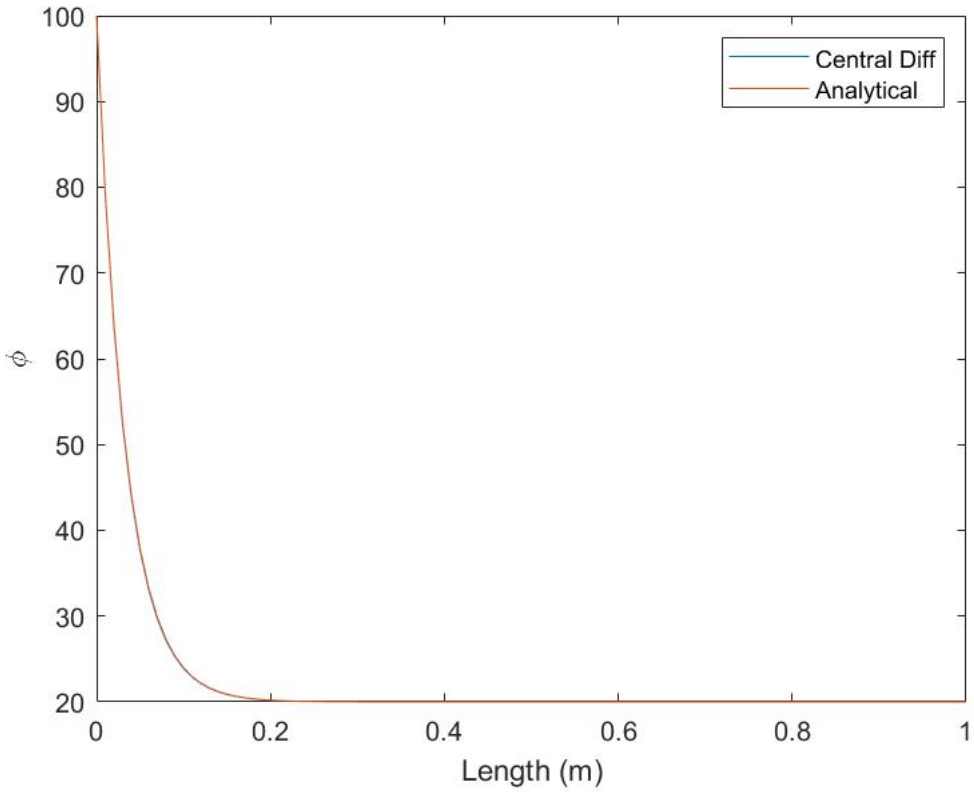

Figure 5.3: Central Differencing Scheme: $u = -30, \delta x = 0.01, |Pe_{local}| = 0.3$*;*

5.1.2 UDS Inaccuracy

While compared to CDS, UDS has the advantage of always being stable, this scheme is often the least accurate at low local Peclet numbers. The stark difference in accuracy when compared with CDS and PLDS is shown in figures 5.4, 5.5 & 5.6 which are the same results with progressively more magnification as indicated by the rectangles. This error is induced by numerical, or false, diffusion. This phenomenon is diffusion caused exclusively by the imposition of a grid, and it affects UDS greatly because of the effective amplification of the diffusion term (i.e. $\frac{\Gamma_\phi}{\delta x}$) by the mass flux term (i.e. ρu).

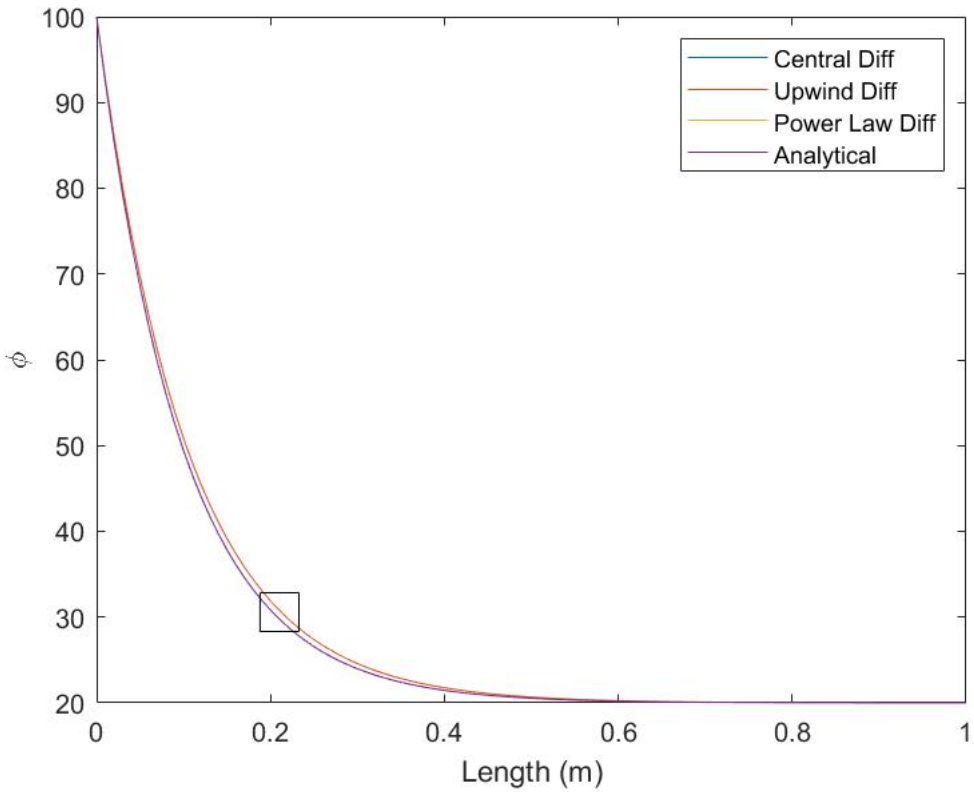

Figure 5.4: Differencing Scheme Comparison Full: $u = -10, \delta x = 0.01, |Pe_{local}| = 0.1$*;*

The absolute error in each scheme is discussed further in section 5.2.1.

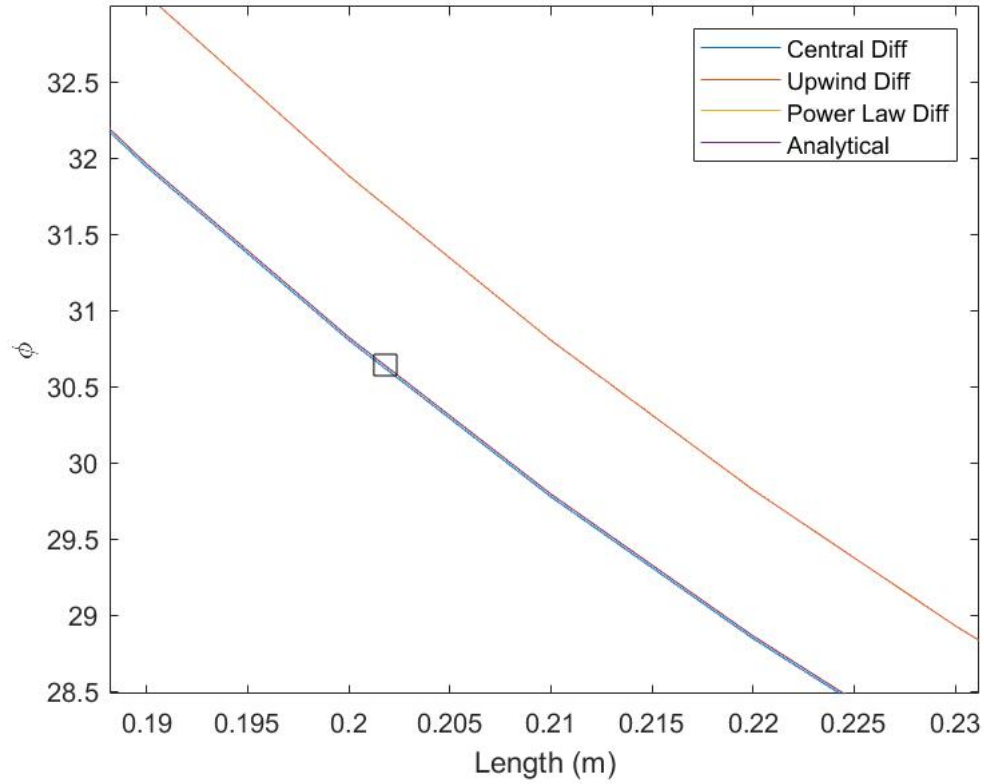

Figure 5.5: Differencing Scheme Comparison Zoom 1: $u = -10, \delta x = 0.01, |Pe_{local}| = 0.1$*;*

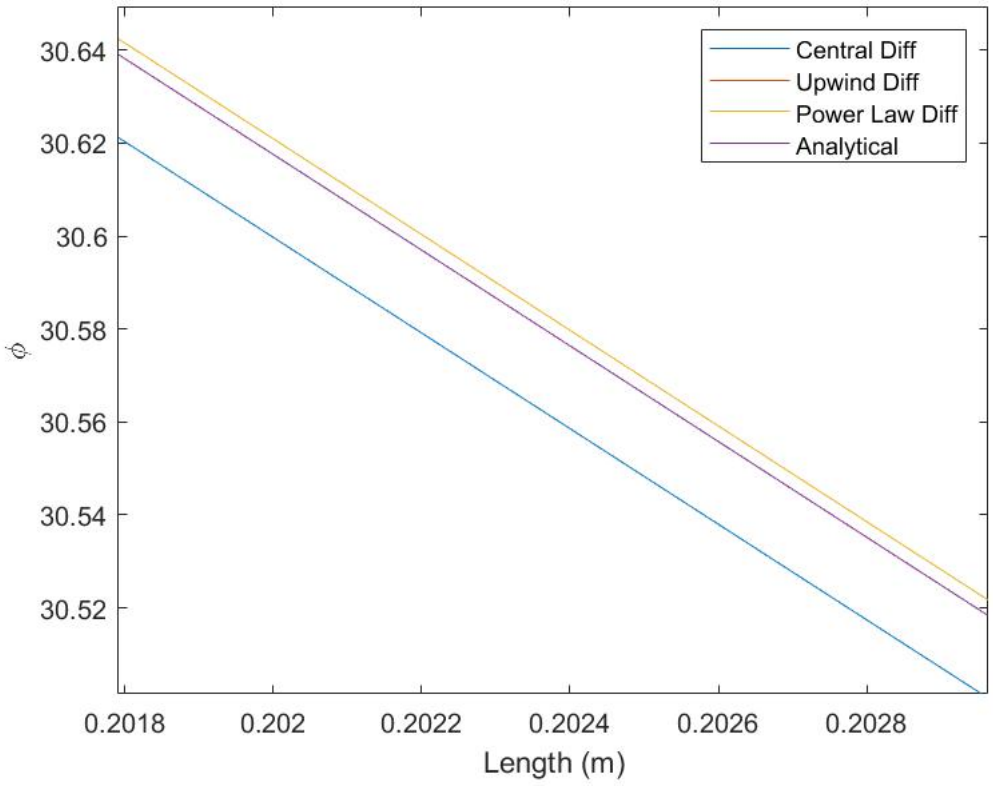

Figure 5.6: Differencing Scheme Comparison Zoom 2: $u = -10, \delta x = 0.01, |Pe_{local}| = 0.1$*;*

5.1.3 Computational Time

After results were produced, a short experiment was run to get an estimate the computational strain of implementing each of the discretisation schemes. The $MATLAB$ function $tictoc$ was used to measure how long the computer took to calculate, for each scheme, the a coefficients from equation 4.3, the coefficients for TDMA (equation 4.11) and finally the ϕ distribution along the domain. The results are shown in figure 5.7. The parameters of the experiment were as follows: $u = -100$, $\delta x = 0.04$.

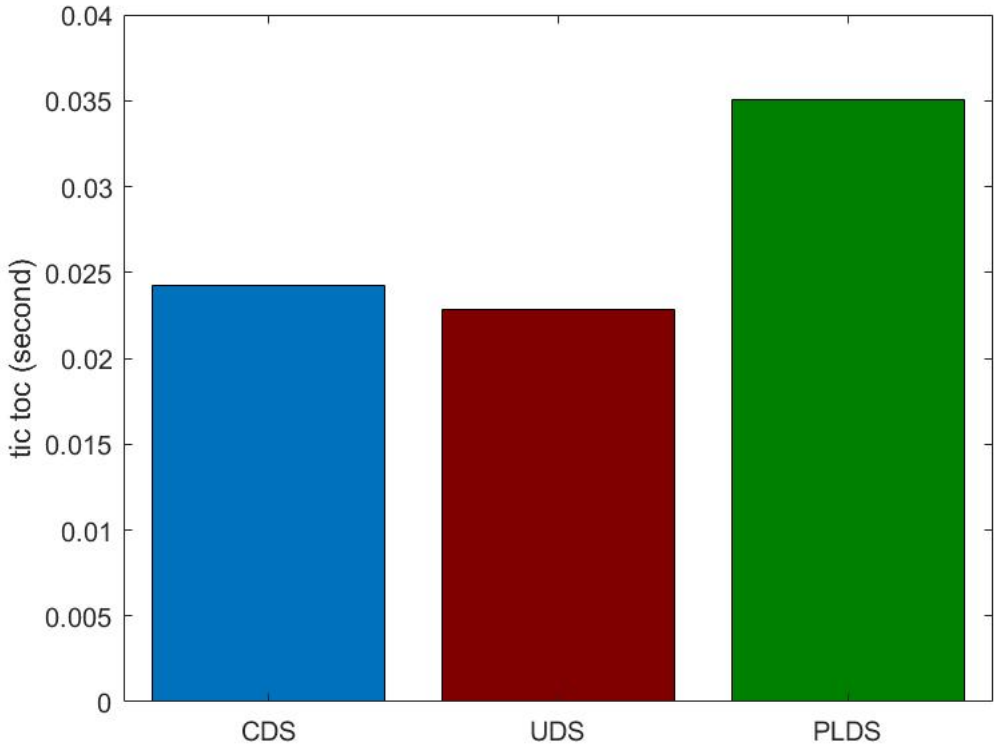

Figure 5.7: Computational Times;

Figure 5.7 shows that PLDS is significantly slower than UDS and CDS, which is to be expected given the number of different and complicated mathematical operations that are required in this scheme (for example raising a value to the power of 5).

5.2 Error Analysis

5.2.1 Error Along Solution Domain

For the parameters of the problem discussed in section 5.1.2, the absolute error of each scheme in units of ϕ is given as a function of length along the solution domain in figure 5.8.

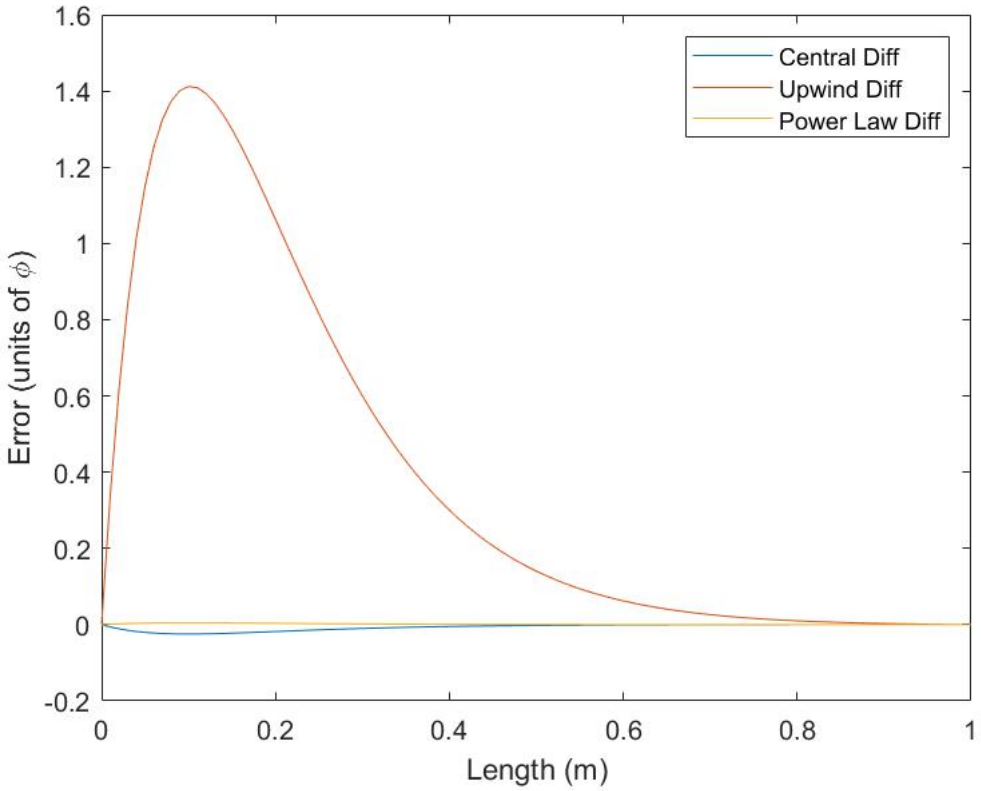

Figure 5.8: Scheme Errors along Domain: $u = -10, \delta x = 0.01, |Pe_{local}| = 0.1$*;*

Figure 5.8 validates the conclusion of section 5.1.2 that UDS is inaccurate compared to the other schemes. That conclusion also requires the condition that CDS is not caused to become unstable and thereby become the scheme with the largest errors as shown in figure 5.9. In both cases, PLDS has a comparatively low error along the solution domain.

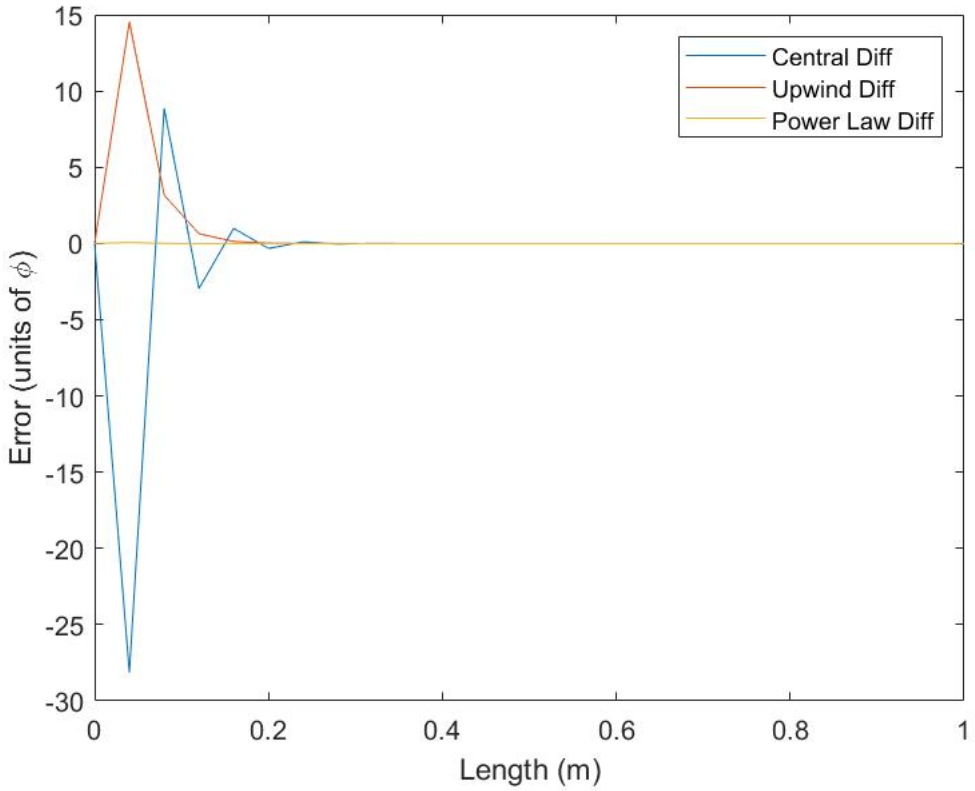

Figure 5.9: Scheme Errors along Domain: $u = -100, \delta x = 0.04, |Pe_{local}| = 4$*;*

5.2.2 Error Variation with Grid Resolution

While graphs showing variation in error along the solution domain (section 5.2.1) are useful for comparing discretisation schemes, it does not give a single measure of total error for a given grid resolution. One measure of overall accuracy is suggested in the assignment [3] as below, where n is the number of nodes:

$$\%\ \ error = \frac{100}{n} \sum \left| \frac{\phi_{computational} - \phi_{analytical}}{\phi_{analytical}} \right| \tag{5.1}$$

When this estimate of error is calculated and plotted for the different schemes, the conclusions suggested previously in section 5.1 are validated. The data in question are shown in figures 5.10 to 5.12. As foreshadowed in section 5.2.1, at fine grid resolutions, PLDS is the most accurate, ahead of CDS and then UDS which is far less accurate. At coarser grid resolutions, CDS becomes unstable and less accurate even than UDS. Interestingly, at fine grid resolutions, all scheme errors have a constant exponential dependency on resolution, with CDS and PLDS having equal exponential dependencies (see figure 5.12).

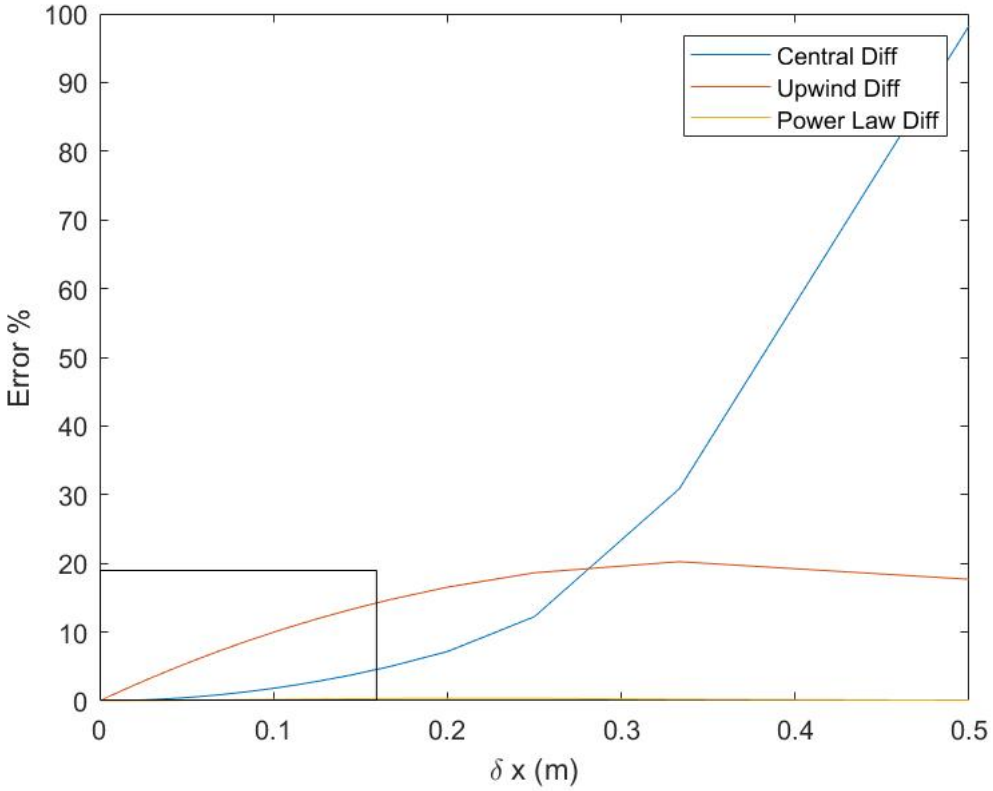

Figure 5.10: Error Variation with Grid Resolution Full: $u = -10$*;*

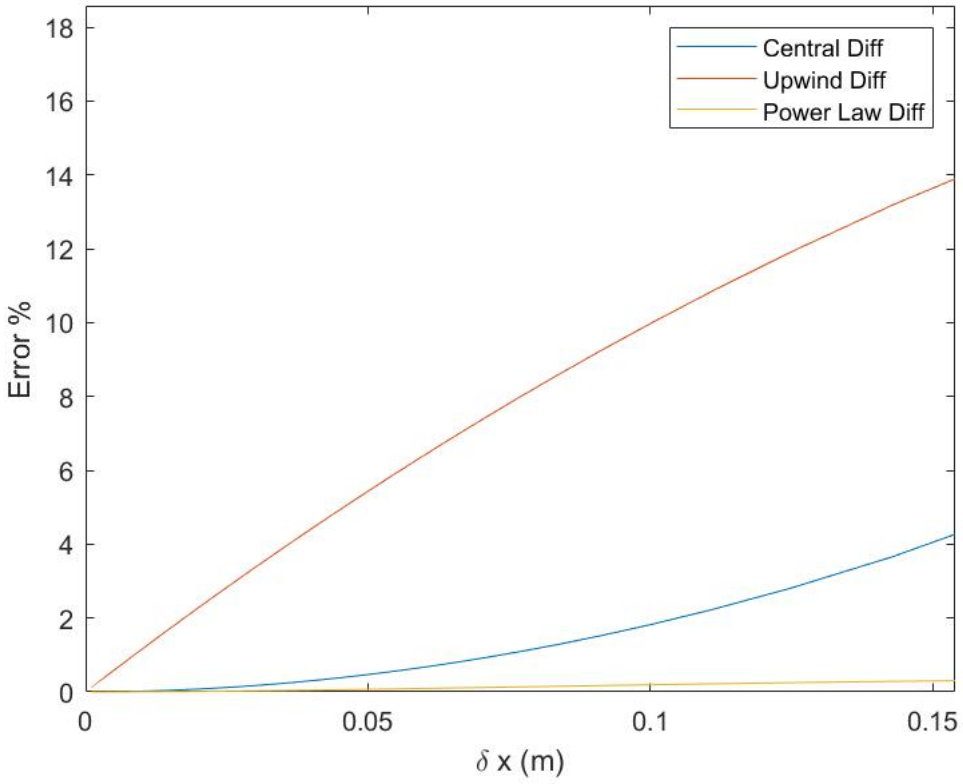

Figure 5.11: Error Variation with Grid Resolution Zoom: $u = -10$*;*

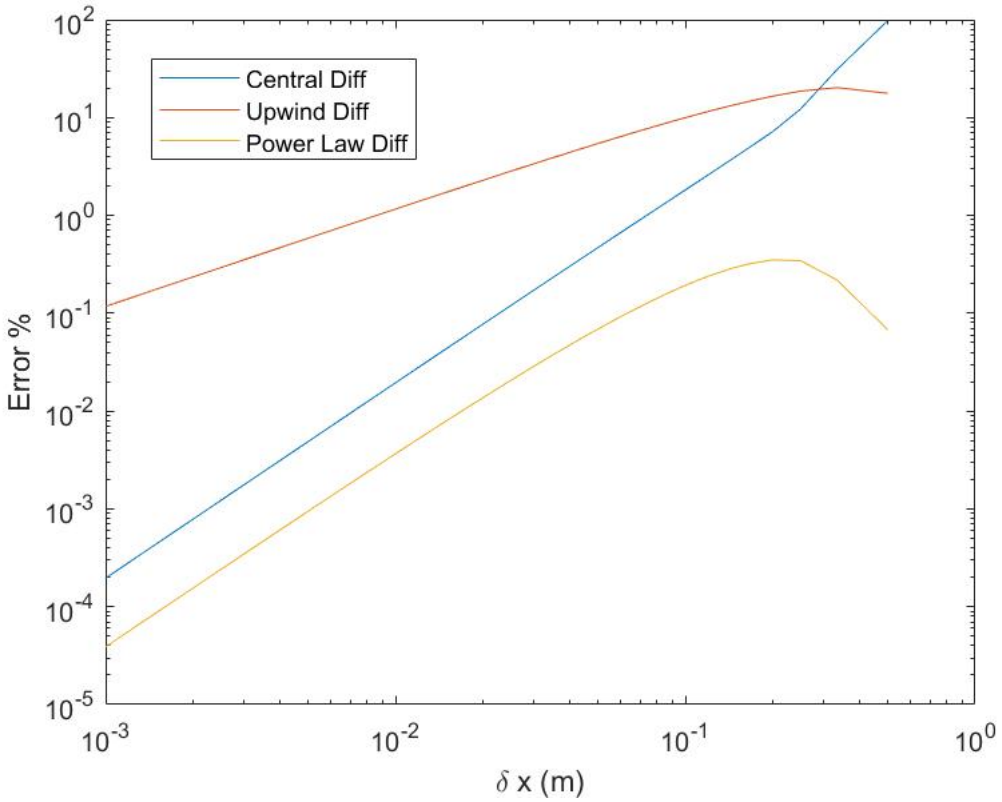

Figure 5.12: Error Variation with Grid Resolution Log-Log Scale: $u = -10$*;*

6. Conclusion

Figures 5.10 and 5.11 confirm the following conclusions:

1. CDS is unstable at high local Peclet numbers and so produces a high total error at high grid spacing (see section 5.1.1).
2. UDS is inaccurate compared with other schemes at low Peclet Numbers (i.e. low grid spacing) due to false diffusion (see section 5.1.2). UDS does however guarantee a stable solution.
3. PLDS is the most accurate scheme for all Peclet numbers (see section 4.3). It also guarantees a stable solution.
4. PLDS does have the drawback that it is moderately slower than CDS or UDS (see section 5.1.3).

Therefore for a problem such as this, the Power Law Differencing Scheme should always be considered the most accurate due to its unconditional stability and false diffusion correction characteristic. However, if computational time is critical, users should consider using other schemes as this extra time could be used to implement a finer grid resolution, amongst other strategies.

7. References

[1] P. Aleiferis. Computational Fluid Dynamics (CFD) (MECH97012/MECH97052, ME4 & MSc), 2019.

[2] H. Hart. CFD19-20_Project1_Code_Hart_01190775.zip. Uploaded for Assessment, 2019.

[3] P. Aleiferis. Computational Fluid Dynamics (ME4 & MSc) Project Assignment 1. 2019.

[4] B. van Wachem and F. Denner. Computational Fluid Dynamics ME4 and MSc. 2015.

[5] O. Reynolds. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society of London A.*, 186:123–164, 1895. doi: $10.1098/\mathrm{rsta}.1895.0004$.

[6] C.F. Gauss. Theoria attractionis corporum sphaeroidicorum ellipticorum homogeneorum methodo nova tractata. *Commentationes societatis regiae scientiarium Gottingensis recentiores*, 2:355 – 378, 1813.

A. Source Code

A.1 Figure 3.1

The source code to produce figure 3.1 is given below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script plots analytical variation of phi along domain for
    ↪ different %
% Peclet numbers
    ↪                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        H.R.HART 09−12−2019          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
c=1
for j = −20.001:19.999 %loop through Peclet numbers / speeds
    %DEFINE SYSTEM PARAMETERS%
    L = 1; %length in metres
    phi_0 = 100; %phi at 0m
    phi_L = 20; %phi at end
    n = 1001; %number of nodes
    u(1:n) = j; %speed from left to right
    rho(1:n) = 1; %density in kg/m^3
    gamma_phi(1:n) = 1; %diffusivity in kg/m/s^2
```

```
    Pec = rho(1)*u(1)*L/gamma_phi(1) %global peclet number for
        ↪ analytical solution
    %ANALYTICAL SOLUTION%%%
    for i = 1:n; %loop to determine analytical phi
        x(i) = L*(i-1)/(n-1);
        phi_an(i,c) = phi_0 +
            ↪ (exp(x(i)*Pec/L)-1)*(phi_L-phi_0)/(exp(Pec)-1); %as
            ↪ given in notes
    end
    c = c+1
end
plot(x,phi_an)
xlabel('Length (m)')
ylabel('\phi')
```

A.2 Figures 5.1 to 5.6 and 5.8 to 5.9

The source code to produce figures 5.1 to 5.6 and 5.8 to 5.9 is given below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script plots phi and error variation along solution domain for
    ↪ the three %
% discretisation schemes and analytical solution for comparison.
    ↪ User can  %
% vary system parameters in first block.
    ↪                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          H.R.HART 09-12-2019             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
%DEFINE SYSTEM PARAMETERS%
L = 1; %length in metres
phi_0 = 100; %phi at 0m, boundary condition
phi_L = 20; %phi at end, boundary condition
n = 26; %number of nodes
u(1:n) = -100; %speed from left to right
rho(1:n) = 1; %density in kg/m^3
gamma_phi(1:n) = 1; %diffusivity in kg/m/s^2
Pec = rho(1)*u(1)*L/gamma_phi(1) %global Peclet (Pec) number for
    ↪ analytical solution
%ANALYTICAL SOLUTION%%%
for i = 1:n;
    x(i) = L*(i-1)/(n-1);
    phi_an(i) = phi_0 +
        ↪ (exp(x(i)*Pec/L)-1)*(phi_L-phi_0)/(exp(Pec)-1); %as given
        ↪ in notes
end
```

```
%NUMERICAL SOLUTION%%%
for i = 1:n; %Loop to define array of delta x and local Peclet (Pe)
    ↪ numbers
    dx(i) = x(2)-x(1);
end
Pe = rho.*u.*dx./gamma_phi;
Pe_Local = mean(Pe)
tic
%CENTRAL DIFFERENCING SCHEME%%%
P_c(1)= 0; %Boundary TDMA coefficients
Q_c(1)= phi_0; %Boundary TDMA coefficients
P_c(n) = 0; %Boundary TDMA coefficients
Q_c(n) = phi_L; %Boundary TDMA coefficients
for i = 2:n-1; %loop from West to East to determine TDMA
    ↪ coefficients
    b_c(i) = gamma_phi(i+1)/dx(i+1) -
        ↪ rho(i+1)*mean([u(i+1),u(i)])/2;
    c_c(i) = gamma_phi(i-1)/dx(i-1) +
        ↪ rho(i-1)*mean([u(i-1),u(i)])/2;
    a_c(i) = b_c(i) + c_c(i) + rho(i+1)*mean([u(i+1),u(i)]) -
        ↪ rho(i-1)*mean([u(i-1),u(i)]);
    d_c(i) = 0;
    P_c(i) = b_c(i)/(a_c(i)-c_c(i)*P_c(i-1));
    Q_c(i) = (d_c(i) + c_c(i)*Q_c(i-1))/(a_c(i)-c_c(i)*P_c(i-1));
end
phi_num_c(1) = phi_0; %Boundary condition
phi_num_c(n) = phi_L; %Boundary condition
for i = n-1:-1:2; %Loop from East to West get phi using TDMA
    ↪ coefficients
    phi_num_c(i) = P_c(i)*phi_num_c(i+1) + Q_c(i);
end
toc
toc_c = toc;
tic
```

```
%UPWIND DIFFERENCING SCHEME%%%
P_u(1)= 0; %Boundary TDMA coefficients
Q_u(1)= phi_0; %Boundary TDMA coefficients
P_u(n) = 0; %Boundary TDMA coefficients
Q_u(n) = phi_L; %Boundary TDMA coefficients
for i = 2:n-1; %loop from West to East to determine TDMA
    ↪ coefficients
    b_u(i) = gamma_phi(i+1)/dx(i+1) +
        ↪ max(-rho(i+1)*mean([u(i+1),u(i)]),0);
    c_u(i) = gamma_phi(i-1)/dx(i-1) +
        ↪ max(rho(i-1)*mean([u(i-1),u(i)]),0);
    a_u(i) = b_u(i) + c_u(i) + rho(i+1)*mean([u(i+1),u(i)]) -
        ↪ rho(i-1)*mean([u(i-1),u(i)]);
    d_u(i) = 0;
    P_u(i) = b_u(i)/(a_u(i)-c_u(i)*P_u(i-1));
    Q_u(i) = (d_u(i) + c_u(i)*Q_u(i-1))/(a_u(i)-c_u(i)*P_u(i-1));
end
phi_num_u(1) = phi_0; %Boundary condition
phi_num_u(n) = phi_L; %Boundary condition
for i = n-1:-1:2; %Loop from East to West get phi using TDMA
    ↪ coefficients
    phi_num_u(i) = P_u(i)*phi_num_u(i+1) + Q_u(i);
end
toc
toc_u = toc;
tic
%POWER LAW DIFFERENCING SCHEME%%%
P_p(1)= 0; %Boundary TDMA coefficients
Q_p(1)= phi_0; %Boundary TDMA coefficients
P_p(n) = 0; %Boundary TDMA coefficients
Q_p(n) = phi_L; %Boundary TDMA coefficients
for i = 2:n-1; %loop from West to East to determine TDMA
    ↪ coefficients
```

```
    b_p(i) = (gamma_phi(i+1)/dx(i+1))*max((1-0.1*abs(Pe(i+1)))^5,0)
        ↪ + max(-rho(i+1)*mean([u(i+1),u(i)]),0);
    c_p(i) = (gamma_phi(i-1)/dx(i-1))*max((1-0.1*abs(Pe(i-1)))^5,0)
        ↪ + max(rho(i-1)*mean([u(i-1),u(i)]),0);
    a_p(i) = b_p(i) + c_p(i) + rho(i+1)*mean([u(i+1),u(i)]) -
        ↪ rho(i-1)*mean([u(i-1),u(i)]);
    d_p(i) = 0;
    P_p(i) = b_p(i)/(a_p(i)-c_p(i)*P_p(i-1));
    Q_p(i) = (d_p(i) + c_p(i)*Q_p(i-1))/(a_p(i)-c_p(i)*P_p(i-1));
end
phi_num_p(1) = phi_0; %Boundary condition
phi_num_p(n) = phi_L; %Boundary condition
for i = n-1:-1:2; %Loop from East to West get phi using TDMA
    ↪ coefficients
    phi_num_p(i) = P_p(i)*phi_num_p(i+1) + Q_p(i);
end
toc
toc_p = toc;
%%%CALCULATE ERRORS%%%
err_c = phi_num_c - phi_an;
err_u = phi_num_u - phi_an;
err_p = phi_num_p - phi_an;
%%%PLOT RESULTS%%%
% ... statements allow series to be turned off in plot with
    ↪ prefixed '...'
figure('Name','Phi␣Variation')
plot(...
x,phi_num_c,...
x,phi_num_u,...
x,phi_num_p,...
x,phi_an...
)
legend(...
'Central␣Diff',...
```

```
'Upwind Diff',...
'Power Law Diff',...
'Analytical'...
)
xlabel('Length (m)')
ylabel('\phi')
pause(1)
%%%PLOT ERRORS%%%
% ... statements allow series to be turned off in plot with
    ↪ prefixed '...'
figure('Name','Error Variation')
plot(...
x,err_c,...
x,err_u,...
x,err_p...
)
legend(...
'Central Diff',...
'Upwind Diff',...
'Power Law Diff'...
)
xlabel('Length (m)')
ylabel('Error (units of \phi)')
```

A.3 Figure 5.7

The source code to produce figure 5.7 is given below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script simply plots bar graph of computational times %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          H.R.HART 09-12-2019          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

CDS = 0.024269;
UDS = 0.022851;
PLDS = 0.035049;
x = categorical({'CDS','UDS','PLDS'});
x = reordercats(x,{'CDS','UDS','PLDS'});
y = [CDS,UDS,PLDS];
b = bar(x,y)
b.FaceColor = 'flat';
b.CData(2,:) = [0.5 0 0]
b.CData(3,:) = [0 0.5 0]
ylabel('tic toc (second)')
```

A.4 Figures 5.10 to 5.12

The source code to produce figures 5.10 to 5.12 is given below:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Script loops through grid resolutions to calculate '%error' as
    ↪          %
% defined in the assignment for the three discretisation schemes.
    ↪ '%error' %
% is then plotted against grid spacing for all three.
    ↪                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       H.R.HART 09-12-2019          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
n_coarse = 3;
n_fine = 1001;
n_range = n_coarse:n_fine;
for aye = n_coarse:n_fine
    %DEFINE SYSTEM PARAMETERS%
    L = 1; %length in metres
    phi_0 = 100; %phi at 0m
    phi_L = 20; %phi at end
    n = aye; %number of nodes
    u(1:n) = -10; %speed from left to right
    rho(1:n) = 1; %density in kg/m^3
    gamma_phi(1:n) = 1; %diffusivity in kg/m/s^2
    Pec = rho(1)*u(1)*L/gamma_phi(1); %global peclet number for
        ↪ analytical solution
    %ANALYTICAL SOLUTION%%%
    for i = 1:n;
        x(i) = L*(i-1)/(n-1);
```

```
    phi_an(i) = phi_0 +
        ↪ (exp(x(i)*Pec/L)-1)*(phi_L-phi_0)/(exp(Pec)-1); %as
        ↪ given in notes
end
%NUMERICAL SOLUTION%%%
%define array of delta x and local Peclet numbers
for i = 1:n;
    dx(i) = x(2)-x(1);
end
Pe = rho.*u.*dx./gamma_phi;
Pe_Local = mean(Pe);
%CENTRAL DIFFERENCING SCHEME%%%
P_c(1)= 0;
Q_c(1)= phi_0;
P_c(n) = 0;
Q_c(n) = phi_L;
for i = 2:n-1;
    b_c(i) = gamma_phi(i+1)/dx(i+1) -
        ↪ rho(i+1)*mean([u(i+1),u(i)])/2;
    c_c(i) = gamma_phi(i-1)/dx(i-1) +
        ↪ rho(i-1)*mean([u(i-1),u(i)])/2;
    a_c(i) = b_c(i) + c_c(i) + rho(i+1)*mean([u(i+1),u(i)]) -
        ↪ rho(i-1)*mean([u(i-1),u(i)]);
    d_c(i) = 0;
    P_c(i) = b_c(i)/(a_c(i)-c_c(i)*P_c(i-1));
    Q_c(i) = (d_c(i) +
        ↪ c_c(i)*Q_c(i-1))/(a_c(i)-c_c(i)*P_c(i-1));
end
phi_num_c(1) = phi_0;
phi_num_c(n) = phi_L;
for i = n-1:-1:2;
    phi_num_c(i) = P_c(i)*phi_num_c(i+1) + Q_c(i);
end
%UPWIND DIFFERENCING SCHEME%%%
```

```
P_u(1)= 0;
Q_u(1)= phi_0;
P_u(n) = 0;
Q_u(n) = phi_L;
for i = 2:n-1;
    b_u(i) = gamma_phi(i+1)/dx(i+1) +
        ↪ max(-rho(i+1)*mean([u(i+1),u(i)]),0);
    c_u(i) = gamma_phi(i-1)/dx(i-1) +
        ↪ max(rho(i-1)*mean([u(i-1),u(i)]),0);
    a_u(i) = b_u(i) + c_u(i) + rho(i+1)*mean([u(i+1),u(i)]) -
        ↪ rho(i-1)*mean([u(i-1),u(i)]);
    d_u(i) = 0;
    P_u(i) = b_u(i)/(a_u(i)-c_u(i)*P_u(i-1));
    Q_u(i) = (d_u(i) +
        ↪ c_u(i)*Q_u(i-1))/(a_u(i)-c_u(i)*P_u(i-1));
end
phi_num_u(1) = phi_0;
phi_num_u(n) = phi_L;
for i = n-1:-1:2;
    phi_num_u(i) = P_u(i)*phi_num_u(i+1) + Q_u(i);
end
%POWER LAW DIFFERENCING SCHEME%%%
P_p(1)= 0;
Q_p(1)= phi_0;
P_p(n) = 0;
Q_p(n) = phi_L;
for i = 2:n-1;
    b_p(i) =
        ↪ (gamma_phi(i+1)/dx(i+1))*max((1-0.1*abs(Pe(i+1)))^5,0)
        ↪ + max(-rho(i+1)*mean([u(i+1),u(i)]),0);
    c_p(i) =
        ↪ (gamma_phi(i-1)/dx(i-1))*max((1-0.1*abs(Pe(i-1)))^5,0)
        ↪ + max(rho(i-1)*mean([u(i-1),u(i)]),0);
```

```
        a_p(i) = b_p(i) + c_p(i) + rho(i+1)*mean([u(i+1),u(i)]) -
            ↪ rho(i-1)*mean([u(i-1),u(i)]);
        d_p(i) = 0;
        P_p(i) = b_p(i)/(a_p(i)-c_p(i)*P_p(i-1));
        Q_p(i) = (d_p(i) +
            ↪ c_p(i)*Q_p(i-1))/(a_p(i)-c_p(i)*P_p(i-1));
    end
    phi_num_p(1) = phi_0;
    phi_num_p(n) = phi_L;
    for i = n-1:-1:2;
        phi_num_p(i) = P_p(i)*phi_num_p(i+1) + Q_p(i);
    end
    %%%CALCULATE ERRORS%%%
    err_c = phi_num_c - phi_an;
    err_u = phi_num_u - phi_an;
    err_p = phi_num_p - phi_an;
    %%%CALCULATE ERROR AS DEFINED%%%
    error_c(aye) = 100*(n^(-1))*sum(abs(err_c./phi_an));
    error_u(aye) = 100*(n^(-1))*sum(abs(err_u./phi_an));
    error_p(aye) = 100*(n^(-1))*sum(abs(err_p./phi_an));
    delta_x(aye) = dx(1);
end
%%%PLOT ERRORS AS DEFINED%%%
% ... statements allow series to be turned off in plot with comments
figure('Name','Error␣Variation␣with␣dx')
semilogy(...
delta_x(n_coarse:n_fine),error_c(n_coarse:n_fine),...
delta_x(n_coarse:n_fine),error_u(n_coarse:n_fine),...
delta_x(n_coarse:n_fine),error_p(n_coarse:n_fine)...
)
legend(...
'Central␣Diff',...
'Upwind␣Diff',...
'Power␣Law␣Diff'...
```

```
)
xlabel('\delta x (m)')
ylabel('Error %')
```