Part 6
Chapter 21
Adaptive Methods and Stiff Systems
Chapter Objectives

• Understanding how the Runge-Kutta Fehlberg methods use RK methods of different orders to provide error estimates that are used to adjust step size.
• Familiarizing yourself with the built-in MATLAB function for solving ODEs.
• Learning how to adjust options for MATLAB’s ODE solvers.
• Learning how to pass parameters to MATLAB’s ODE solvers.
• Understanding what is meant by stiffness and its implications for solving ODEs.
Adaptive Runge-Kutta Methods

- The solutions to some ODE problems exhibit multiple time scales - for some parts of the solution the variable changes slowly, while for others there are abrupt changes.
- Constant step-size algorithms would have to apply a small step-size to the entire computation, wasting many more calculations on regions of gradual change.
- Adaptive algorithms, on the other hand, can change step-size depending on the region.
Approaches to Adaptive Methods

- There are two primary approaches to incorporate adaptive step-size control:
  - *Step halving* - perform the one-step algorithm two different ways, once with a full step and once with two half-steps, and compare the results.
  - *Embedded RK methods* - perform two RK iterations of different orders and compare the results. This is the preferred method.
MATLAB Functions

- MATLAB’s `ode23` function uses second- and third-order RK functions to solve the ODE and adjust step sizes.
- MATLAB’s `ode45` function uses fourth- and fifth-order RK functions to solve the ODE and adjust step sizes. This is recommended as the first function to use to solve a problem.
- MATLAB’s `ode113` function is a multistep solver useful for computationally intensive ODE functions.
Using \texttt{ode} Functions

- The functions are generally called in the same way; \texttt{ode45} is used as an example:
  \[
  [t, \ y] = \texttt{ode45}(\texttt{odefun}, \ tspan, \ y0)
  \]
  - \texttt{y}: solution array, where each column represents one of the variables and each row corresponds to a time in the \texttt{t} vector
  - \texttt{odefun}: function returning a column vector of the right-hand-sides of the ODEs
  - \texttt{tspan}: time over which to solve the system
    - If \texttt{tspan} has two entries, the results are reported for those times as well as several intermediate times based on the steps taken by the algorithm
    - If \texttt{tspan} has more than two entries, the results are reported only for those specific times
  - \texttt{y0}: vector of initial values
Example - Predator-Prey

- Solve: \[
\frac{dy_1}{dt} = 1.2y_1 - 0.6y_1y_2 \quad \frac{dy_2}{dt} = 0.8y_2 + 0.3y_1y_2
\]

with \(y_1(0)=2\) and \(y_2(0)=1\) for 20 seconds

- predprey.m M-file:

```matlab
function yp = predprey(t, y)
    yp = [1.2*y(1)-0.6*y(1)*y(2);…
          -0.8*y(2)+0.3*y(1)*y(2)];
end
```

- tspan = [0 20];
  y0 = [2, 1];
  [t, y] = ode45(@predprey, tspan, y0);
  figure(1); plot(t,y); figure(2); plot(y(:,1),y(:,2));
ODE Solver Options

• Options to ODE solvers may be passed as an optional fourth argument, and are generally created using the `odeset` function:
  ```matlab
  options=odeset('par1', 'val1', 'par2', 'val2',...)
  ```

• Commonly used parameters are:
  - `'RelTol'`: adjusts relative tolerance
  - `'AbsTol'`: adjusts absolute tolerance
  - `'InitialStep'`: sets initial step size
  - `'MaxStep'`: sets maximum step size (default: one tenth of `tspan` interval)
Multistep Methods

- **Multistep methods** are based on the insight that, once the computation has begun, valuable information from the previous points exists.

- One example is the non-self-starting Heun’s Method, which has the following predictor and corrector equations:
  
  \[
  \begin{align*}
  (a) \text{ Predictor} & \quad y_{i+1}^0 = y_i^m + f(t_i, y_i^m) \Delta h \\
  (b) \text{ Corrector} & \quad y_{i+1}^j = y_i^m + \frac{f(t_i, y_i^m) + f(t_{i+1}, y_{i+1}^{j-1})}{2}
  \end{align*}
  \]
Stiffness

- A *stiff system* is one involving rapidly changing components together with slowly changing ones.
- An example of a single stiff ODE is:
  \[
  \frac{dy}{dt} = -1000y + 3000 - 2000e^{-t}
  \]
whose solution if \(y(0)=0\) is:
\[
y = 3 - 0.998e^{-1000t} - 2.002e^{-t}
\]
MATLAB Functions for Stiff Systems

- MATLAB has a number of built-in functions for solving stiff systems of ODEs, including `ode15s`, `ode23`, `ode23t`, and `ode23tb`.
- The arguments for the stiff solvers are the same as those for previous solvers.