Because the presentation of this material in lecture will differ from that in the book, I felt that notes that closely follow the lecture presentation might be appreciated.

Contents

5. First-Order Equations: General Theory
   5.1. Well-Posed Problems 2
   5.2. Existence and Uniqueness 3

6. First-Order Equations: Graphical Methods
   6.1. Phase-Line Portraits for Autonomous Equations 4
   6.2. Plots of Explicit Solutions 6
   6.3. Contour Plots of Implicit Solutions 8
   6.4. Direction Fields 11

7. First-Order Equations: Numerical Methods
   7.1. Numerical Approximations 13
   7.2. Explicit and Implicit Euler Methods 14
   7.3. Explicit Methods Based on Taylor Approximation 15
      7.3.1. Explicit Euler Method Revisited 15
      7.3.2. Local and Global Errors 15
      7.3.3. Higher-Order Taylor-Based Methods 16
   7.4. Explicit Methods Based on Quadrature 17
      7.4.1. Explicit Euler Method Revisited Again 17
      7.3.2. Heun-Trapezoidal Method 18
      7.3.3. Heun-Midpoint Method 19
      7.3.4. Runge-Kutta Method 20
5. FIRST-ORDER EQUATIONS: GENERAL THEORY

So far we have used analytical methods to construct solutions of first-order differential equations and solve initial-value problems. These methods have required the equation to be either linear or separable. More generally, they can be applied to any equation that can be transformed into either of those forms. They also have required finding certain primitives. We will learn other analytic methods later, but they also will require the equation to satisfy certain restrictions and finding certain primitives. However, most first-order differential equations either will not satisfy any of these restrictions or will lead to a primitive that cannot be found analytically. Moreover, even when analytic methods can work, they can produce complicated expressions which are not easy to understand. It is therefore helpful to have other methods that can be applied to a broad class of first-order equations.

Subsequent sections will develop graphical and numerical methods that can be applied to a broad class of first-order equations. These methods require a little theoretical groundwork so that you have a clear understanding of when they can be applied.

5.1. Well-Posed Problems. The notion of a well-posed problem is central to science and engineering. It motivated by the idea that mathematical problems in science and engineering are used to predict or explain something. A problem is called well-posed if

(i) the problem has a solution,
(ii) the solution is unique,
(iii) the solution depends continuously the problem.

The motivations for the first two points are fairly clear: a problem with no solution will not give a prediction, and a problem with many solutions gives too many. The third point is crucial. It recognizes that a mathematical problem is always a model of reality. Some nearby mathematical problems will be equally valid models. To have predictive value, we should be confident that the solutions of these other models lie close the solution of our model. For example, if our model is an initial-value problem associated with a differential equation then we would like to know that its solution would not change much if the initial value was a bit different or if a coefficient in the differential equation were a bit different. This is what is meant by saying the solution depends continuously the problem.

The solution of a well-posed problem can be approximated accurately by a wealth of techniques. The solution of a problem that is not well-posed is very difficult, if not impossible to approximate accurately. This is why scientists and engineers want to know which problems are well-posed and which are not.

In this section we consider initial-value problems of the form

\[
\frac{dy}{dt} = f(t, y), \quad y(t_I) = y_I.
\]

We will give conditions on \( f(t, y) \) that insure this problem has a unique solution. In subsequent sections we will use this theory to develop methods by which we can study the solution to this problem when analytical methods either do not apply or become complicated.
5.2. Existence and Uniqueness. Here we will address only the existence and uniqueness of solutions. We begin with a definition that a picture should help clarify.

Definition 5.1. Let $S$ be a set in the $ty$-plane. A point $(t_o, y_o)$ is said to be in the interior of $S$ if there exists a open rectangle $(t_L, t_R) \times (y_L, y_R)$ that contains the point $(t_o, y_o)$ and also lies within the set $S$.

Our basic existence and uniqueness theorem is the following.

Theorem 5.1. Let $f(t, y)$ be a function defined over a set $S$ in the $ty$-plane such that

- $f$ is continuous over $S$,
- $f$ is differentiable with respect to $y$ over $S$,
- $\partial_y f$ is continuous over $S$.

Then for every initial time $t_I$ and every initial value $y_I$ such that $(t_I, y_I)$ is in the interior of $S$ there exists a unique solution $y = Y(t)$ to initial-value problem (5.1) that is defined over some time interval $(a, b)$ such that

- $t_I$ is in $(a, b)$,
- $\{(t, Y(t)) : t \in (a, b)\}$ lies within the interior of $S$.

Moreover, $Y(t)$ extends to the largest such interval and $Y'(t)$ is continuous over that interval.

Remark. This is not the most general theorem we could state, but it is one that applies to most equations you will face in this course, and is easy to apply. It asserts that $Y(t)$ will exist until $(t, Y(t))$ leaves $S$.

Example. Determine how Theorem 5.1 applies to the initial-value problem

$$\frac{dy}{dt} = \sqrt{1 + t^2 + y^2}, \quad y(t_I) = y_I.$$

Solution. Because $f(t, y) = \sqrt{1 + t^2 + y^2}$ is defined over $(-\infty, \infty) \times (-\infty, \infty)$, we first try taking $S = (-\infty, \infty) \times (-\infty, \infty)$. Clearly, $f$ is continuous over $S$, $f$ is differentiable with respect to $y$ over $S$ with

$$\partial_y f(t, y) = \frac{y}{\sqrt{1 + t^2 + y^2}},$$

and $\partial_y f$ is continuous over $S$. Every point in $S$ is also in the interior of $S$. Therefore any initial data $(t_I, y_I)$ is in the interior of $S$. Theorem 5.1 therefore insures that the initial-value problem has a unique solution $y = Y(t)$ that is defined over some time interval $(a, b)$ that contains $t_I$. Either the solution extends to all $t$ or $Y(t)$ blows up in finite time because those are the only ways for $(t, Y(t))$ to leave $S$. (In fact, it extends to all $t$.)

Remark. The initial-value problem in the example above cannot be solved by analytic methods. However, Theorem 5.1 insures that its solution exists and is unique. In the next sections we will see how to visualize and approximate it.
6. First-Order Equations: Graphical Methods

Sometimes the best way to understand the solution of a differential equation is by graphical methods. Of course sometimes these methods can be applied when analytic methods fail to yield explicit solutions. But even when analytic methods can yield explicit solutions, it is often better to gain some understanding of the solutions through a graphical method. Often you can find out everything you need to know graphically, thereby saving yourself from a complicated analytic calculation.

6.1. Phase-Line Portraits for Autonomous Equations. This method can be applied to autonomous equations of the form

\[ \frac{dy}{dt} = g(y). \]

It has the virtue that it can often be carried out quickly without the aid of a calculator or computer. It requires that \( g \) be continuous over an interval \((y_L, y_R)\), that \( g \) be differentiable at each of its zeros. Then by Theorem 4.1 every point of \((y_L, y_R)\) has a unique solution of (6.1) passing through it.

Any solution \( y = Y(t) \) of (6.1) can be viewed as giving the position of a point moving along the interval \((y_L, y_R)\) as a function of time. We can determine the direction that this point moves as time increases from the sign of \( g(y) \):

- where \( g(y) = 0 \) the point does not move because \( Y''(t) = \frac{dy}{dt} = 0 \),
- where \( g(y) > 0 \) the point moves to the right because \( Y''(t) = \frac{dy}{dt} > 0 \),
- where \( g(y) < 0 \) the point moves to the left because \( Y''(t) = \frac{dy}{dt} < 0 \).

We can present the sign analysis of \( g(y) \) on a graph of the interval \((y_L, y_R)\) as follows.

1. Find all the zeros of \( g(y) \). These are the stationary points of the equation. They usually are isolated. Plot these points on the interval \((y_L, y_R)\). They partition \((y_L, y_R)\) into subintervals.

2. Determine the sign of \( g(y) \) on each of these subintervals. Plot right arrows on each subinterval where \( g(y) \) is positive and left arrows on each subinterval where \( g(y) \) is negative. These arrows indicate the direction that solutions of the equation will move along \((y_L, y_R)\) as time increases.

The resulting graph is called the phase portrait for equation (6.1) along the interval \((y_L, y_R)\) of the phase-line. It gives a rather complete picture of how all solutions of (6.1) behave when they take values in \((y_L, y_R)\).

Example. Describe the behavior of all solutions of the equation

\[ \frac{dy}{dt} = 4y - y^3. \]

Solution. Because \( 4y - y^3 = y(2 + y)(2 - y) \), the stationary points of this equation are \( y = -2, y = 0, \) and \( y = 2 \). To get a complete picture you should sketch a phase portrait over an interval that includes all of these points, say \((-4, 4)\). Clearly \( g(y) \) is positive over \((-4, -2)\), negative over \((-2, 0)\), positive over \((0, 2)\) and negative over \((2, 4)\). The phase portrait is therefore
It is clear from this portrait that:

- every solution \( y(t) \) that initially lies within \((−∞, −2)\) will move towards \(-2\) as \( t \) increases with \( y(t) \to −2 \) as \( t \to ∞ \);
- every solution \( y(t) \) that initially lies within \((-2, 0)\) will move towards \(-2\) as \( t \) increases with \( y(t) \to −2 \) as \( t \to ∞ \);
- every solution \( y(t) \) that initially lies within \((0, 2)\) will move towards \(2\) as \( t \) increases with \( y(t) \to 2 \) as \( t \to ∞ \);
- every solution \( y(t) \) that initially lies within \((2, ∞)\) will move towards \(2\) as \( t \) increases with \( y(t) \to 2 \) as \( t \to ∞ \).

Of course, all of this information can be read off from the analytic general solution we had worked out earlier. But if this is all you wanted to know, sketching the phase portrait is certainly a faster way to get it.

The above portrait shows that all solutions near \(-2\) and \(2\) move towards them while all solutions near \(0\) move away from it. We say that the stationary points \(-2\) and \(2\) are stable, while the stationary point \(0\) is unstable. A stationary point that has some solutions that move towards it and others that move away from it are called semistable. Phase portraits allow you to quickly classify the stability of every stationary point.

**Example.** Classify the stability of all the stationary solutions of

\[
\frac{dx}{dt} = x(2 - x)(4 - x)^2.
\]

Describe the behavior of all solutions.

**Solution.** The stationary points of this equation are \( x = 0 \), \( x = 2 \), and \( x = 4 \). A sign analysis of \( x(2 - x)(4 - x)^2 \) shows that the phase portrait for this equation is

\[
\begin{array}{ccc}
\longrightarrow & \bullet & \longrightarrow \\
\bullet & & \bullet \\
\longrightarrow & \bullet & \longrightarrow \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 2 & 4 \\
\text{unstable} & \text{stable} & \text{semistable} \\
\end{array}
\]

We thereby classify the stability of the stationary as indicated above. Moreover:

- every solution \( x(t) \) that initially lies within \((-∞, 0)\) will move towards \(-∞\) as \( t \) increases with \( x(t) \to −∞ \) as \( t \to t_∗ \), where \( t_∗ \) is some finite “blow-up” time;
- every solution \( x(t) \) that initially lies within \((0, 2)\) will move towards \(2\) as \( t \) increases with \( x(t) \to 2 \) as \( t \to ∞ \);
- every solution \( x(t) \) that initially lies within \((2, 4)\) will move towards \(2\) as \( t \) increases with \( x(t) \to 2 \) as \( t \to ∞ \);
- every solution \( x(t) \) that initially lies within \((4, ∞)\) will move towards \(4\) as \( t \) increases with \( y(t) \to 4 \) as \( t \to ∞ \).

Perhaps the only one of these statements that might be a bit surprising is the last one. By just looking at the arrows on the phase-line you might have thought that such solutions would move past \(4\) and continue down to \(2\). But you have to remember that Theorem 4.1 tells us that nonstationary solutions never hit the stationary solutions. They merely approach them as \( t \to ∞ \).
6.2. **Plots of Explicit Solutions.** This method can be applied whenever you can obtain an explicit formula for a solution or a family of solutions. For example, if you want to see what a solution \( y = Y(t) \) looks like over the time interval \([t_L, t_R]\) then the simplest thing to do when you already have an explicit expression for \( Y(t) \) is use the MATLAB command `ezplot`. In general, you plot \( Y(t) \) over a time interval \([t_L, t_R]\) by

\[
\text{>> ezplot('} Y(x) \text{', [} t_L, t_R \text{])}
\]

\[
\text{>> xlabel 't', ylabel 'y'}
\]

\[
\text{>> title 'Plot of } y = Y(t)\text{'}
\]

You should get into the habit of labeling each axis and titling each graph.

**Example.** Consider the linear initial-value problem

\[
\frac{dy}{dt} + y = 5 \sin(2t), \quad y(0) = 1.
\]

You have found that its solution is

\[
y = 3e^{-t} - 2 \cos(2t) + \sin(2t).
\]

Plot this solution over the time interval \([0, 5]\).

**Solution.** You can input the explicit expression directly into `ezplot` as a symbolic expression. For example,

\[
\text{>> ezplot('}3*exp(-x) - 2*cos(2*x) + sin(2*x)\text{', [}0, 5\text{])}
\]

\[
\text{>> xlabel 't', ylabel 'y'}
\]

\[
\text{>> title '} Plot of y = 3 \exp(-t) - 2 \cos(2t) + \sin(2t)\text{'}
\]

**Remark.** The first argument of `ezplot` must be a symbolic expression with one variable. Here we used \( x \) as the variable.

**Alternative Solution.** You can also input the explicit expression indirectly into `ezplot` as a symbolic expression. For example,

\[
\text{>> sym} t; \text{sol} = 3*exp(-t) - 2*cos(2*t) + sin(2*t);
\]

\[
\text{>> ezplot(sol, [0 5])}
\]

\[
\text{>> xlabel 't', ylabel 'y'}
\]

\[
\text{>> title '} Plot of y = 3 \exp(-t) - 2 \cos(2t) + \sin(2t)\text{'}
\]

**Remark.** We will take the second approach when plotting members of a family of solutions.

You can also use `ezplot` when the input is a symbolic solution generated by the MATLAB command `dsolve`.

**Example.** Consider the linear initial-value problem

\[
\frac{dy}{dt} + y = 5 \sin(2t), \quad y(0) = 1.
\]

Plot its solution over the time interval \([0, 5]\).

**Solution.** You can input the initial-value problem into `dsolve` as symbolic strings. For example,

\[
\text{>> sol} = \text{dsolve('}Dy + y = 5 \sin(2t)\text{', 'y(0) = 1', 't');}
\]

\[
\text{>> ezplot(sol, [0 5])}
\]

\[
\text{>> xlabel 't', ylabel 'y'}
\]

\[
\text{>> title '} Solution of Dy + y = 5 \sin(2t), y(0) = 1\text{'}
\]

**Remark.** The semicolon after the `dsolve` command suppresses its explicit output.
If \( y = Y(t, c) \) is a family of solutions to a differential equation and you want to see how these solutions look over the time interval \([t_L, t_R]\) for several different values of \( c \) then you can change the values of \( c \) in ezplot by using the MATLAB command \texttt{subs} inside a loop.

**Example.** Consider the general initial-value problem

\[
\frac{dy}{dt} + y = 5 \sin(2t), \quad y(0) = c.
\]

You have found that its solution is

\[
y = (c + 2)e^{-t} - 2 \cos(2t) + \sin(2t).
\]

Plot this solution over the time interval \([0, 5]\) for \( c = -3, -1, 1, \) and 3.

**Solution.** We can define the family as a symbolic expression outside the loop as

\[
\texttt{syms t c}
\]

\[
\texttt{gensol = (c + 2)*exp(-t) - 2*cos(2*t) + sin(2*t);}
\]

\[
\texttt{figure; hold on}
\]

\[
\texttt{for cval = -3:2:3}
\]

\[
\texttt{ezplot(subs(gensol, 'c', cval), [0 5])}
\]

\[
\texttt{end}
\]

\[
\texttt{axis tight, xlabel 't', ylabel 'y'}
\]

\[
\texttt{title 'Plot of } y = (c + 2)e^{-t} - 2 \cos(2t) + \sin(2t) \text{ for } c = -3, -1, 1, 3'
\]

\[
\texttt{hold off}
\]

**Remark.** Notice that we have plotted the general solution for only four values of \( c \). Plotting the solution for many more values of \( c \) can make the resulting graph look cluttered.

You can do a similar thing when the input is a symbolic family of solutions generated by the MATLAB command \texttt{dsolve}.

**Example.** Consider the general linear initial-value problem

\[
\frac{dy}{dt} + y = 5 \sin(2t), \quad y(0) = c.
\]

Plot its solution over the time interval \([0, 5]\) for \( c = -3, -1, 1, \) and 3.

**Solution.** Simply modify the previous solution as

\[
\texttt{gensol = dsolve('Dy + y = 5*sin(2*t)', 'y(0) = c', 't');}
\]

\[
\texttt{figure; hold on}
\]

\[
\texttt{for cval = -3:2:3}
\]

\[
\texttt{ezplot(subs(gensol, 'c', cval), [0 5])}
\]

\[
\texttt{end}
\]

\[
\texttt{axis tight, xlabel 't', ylabel 'y'}
\]

\[
\texttt{title 'Solution of Dy + y = 5 \sin(2t), y(0) = c for c = -3, -1, 1, 3'}
\]

\[
\texttt{hold off}
\]

**Remark.** You can find more examples of how to use ezplot to graph explicit solutions of first-order differential equations in our MATLAB book, *Differential Equations with MATLAB* by Hunt, Lipsman, Osborn, and Rosenberg.
6.3. Contour Plots of Implicit Solutions. This method can be applied anytime the solutions of a differential equation are implicitly given by an equation of the form

\[(6.2) \quad H(x, y) = c.\]

This situation might arise when solving a separable equation for which you can analytically find the primitives \(F(x)\) and \(G(y)\), but \(G(y)\) is so complicated that you cannot analytically compute the inverse function \(G^{-1}\). In that case (6.2) takes on the special form

\[F(x) - G(y) = c.\]

We will soon learn other methods that can lead to implicit solutions of the form (6.1) for any function \(H(x, y)\) with continuous second partial derivatives. We will assume that \(H(x, y)\) has continuous second partial derivatives.

Recall from multivariable calculus that the critical points of \(H\) are those points in the \(xy\)-plane where the gradient of \(H\) vanishes — i.e. those points \((x, y)\) where

\[\partial_x H(x, y) = \partial_y H(x, y) = 0.\]

The value of \(H(x, y)\) at a critical point is called a critical value of \(H\). We will assume that each of critical point of \(H\) is nondegenerate. This means that at each of critical point of \(H\) the Hessian matrix \(H(x, y)\) of second partial derivatives has a nonzero determinant — i.e. at every critical point \((x, y)\) we have

\[\det(H(x, y)) \neq 0, \quad \text{where} \quad H(x, y) = \begin{pmatrix} \partial_{xx} H(x, y) & \partial_{xy} H(x, y) \\ \partial_{yx} H(x, y) & \partial_{yy} H(x, y) \end{pmatrix}.\]

Because they are nondegenerate, the critical points of \(H\) can be classified as follows.

- If \(\det(H(x, y)) > 0\) and \(\partial_{xx} H(x, y) > 0\) then \((x, y)\) is a local minimizer of \(H\).
- If \(\det(H(x, y)) > 0\) and \(\partial_{xx} H(x, y) < 0\) then \((x, y)\) is a local maximizer of \(H\).
- If \(\det(H(x, y)) < 0\) then \((x, y)\) is a saddle point of \(H\).

**Remark.** Nondegenerate critical points of \(H\) are isolated. This means that each of critical point of \(H\) is contained within a rectangle \((a, b) \times (c, d)\) that contains no other critical points. This is a consequence of the Implicit Function Theorem of multivariable calculus.

Given a value for \(c\), solutions of the differential equation lie on the set in the \(xy\)-plane given by

\[\{(x, y) : H(x, y) = c\}.\]

This is the so-called level set of \(H(x, y)\) associated with \(c\). Whenever this set has at least one point in it, the Implicit Function Theorem of multivariable calculus tells us the following.

- If \(c\) is not a critical value then its level set will look like one or more curves in the \(xy\)-plane that never meet. These curves will either be loops that close on themselves or extend to infinity. They will not have endpoints.
- If \(c\) is a critical value then its level set will look like one or more local extremizers of \(H\) plus some curves in the \(xy\)-plane that might meet only at saddle points. These curves will either be loops that close on themselves, extend to infinity, or have an endpoint at a saddle point.

The idea is to plot one or more level sets of \(H\) within a bounded rectangle in the \(xy\)-plane that is of interest. When many level sets are used the result is called a contour plot of \(H\).
Remark. If we consider $H(x, y)$ to give the height of the graph of $H$ over the $xy$-plane then a contour plot shows the height of this graph in exactly the same way a contour map shows the elevation of topographical features.

We can produce a contour plot of a function $H(x, y)$ over the rectangle $[x_L, x_R] \times [y_L, y_R]$ by using the MATLAB commands \texttt{meshgrid} and \texttt{contour} as follows.

\begin{verbatim}
>> [X, Y] = meshgrid(x_L:h:x_R,y_L:k:y_R);
>> contour(X, Y, H(X,Y))
>> axis square, xlabel 'x', ylabel 'y'
>> title 'Contour Plot of $H(x, y)$'
\end{verbatim}

Here $h$ and $k$ are the resolutions of the intervals $[x_L, x_R]$ and $[y_L, y_R]$ respectively, which should have values of the form

$$h = \frac{x_R - x_L}{m}, \quad k = \frac{y_R - y_L}{n},$$

where $m$ and $n$ are positive integers.

The \texttt{meshgrid} command creates an array of grid points in the rectangle $[x_L, x_R] \times [y_L, y_R]$ given by $(x_i, y_j)$ where

$$x_i = x_L + ih \text{ for } i = 0, 1, \cdots m, \quad y_j = y_L + jk \text{ for } j = 0, 1, \cdots n.$$  

More precisely, \texttt{meshgrid} creates two arrays; the array X contains $x_i$ in its $ij$-th entry while the array Y contains $y_j$ in its $ij$-th entry.

The \texttt{contour} command computes $H(x_i, y_j)$ at each of these grid points, uses these values to construct an approximation to $H(x, y)$ over $[x_L, x_R] \times [y_L, y_R]$ by interpolation, selects nine values for $c$ that are evenly spaced between the minimum and maximum values of this interpolation, and then constructs an approximation to the level set of each $c$ based on this interpolation. Exactly how \texttt{contour} does all this is beyond the scope of this course, so we will not discuss it further. You can learn more about such algorithms in a numerical analysis course. Here all you need to know is that MATLAB produces an approximation to each level set. This approximation can be improved by making $h$ and $k$ smaller. The resulting contour plot will give you a good idea of what $H(x, y)$ looks like over the rectangle $[x_L, x_R] \times [y_L, y_R]$ provided $h$ and $k$ are small enough. Typical values for $m$ and $n$ run between 50 and 200.

Example. Produce a contour plot of $H(x, y) = x^2 + y^2$ in the rectangle $[-5, 5] \times [-5, 5]$.

Solution. If we select $m = n = 100$, so that $h = k = .1$, then our MATLAB program becomes

\begin{verbatim}
>> [X, Y] = meshgrid(-5:0.1:5,-5:0.1:5);
>> contour(X, Y, X.^2 + Y.^2)
>> axis square, xlabel 'x', ylabel 'y'
>> title 'Contour Plot of $x^2 + y^2$'
\end{verbatim}

Because the minimum and maximum of $x^2 + y^2$ over the rectangle $[-5, 5] \times [-5, 5]$ are 0 and 50 respectively, the values of $c$ MATLAB will choose are $5l$ where $l = 0, 1, 2, \cdots, 10$. The resulting contour plot will show a point at the origin plus the intersections of the rectangle $[-5, 5] \times [-5, 5]$ with ten concentric circles centered at the origin that have radii $\sqrt{5}$, $\sqrt{10}$, $\sqrt{15}$, $2\sqrt{5}$, $5$, $\sqrt{30}$, $\sqrt{35}$, $2\sqrt{10}$, $3\sqrt{5}$, and $5\sqrt{2}$.

Remark. The dots that appear in $X.^2 + Y.^2$ tell MATLAB that we want to square each entry in the arrays X and Y rather than square the arrays themselves by matrix multiplication.
If we are investigating an initial-value problem with initial data \((x_I, y_I)\) inside a rectangle \([x_L, x_R] \times [y_L, y_R]\) then the only level set that we want to plot is the one corresponding to \(c = H(x_I, y_I)\). This level set might show more solutions than the one we seek, so be careful! We can plot just this level set by the following.

```matlab
>>> c = H(x_I, y_I);
>>> [X, Y] = meshgrid(x_L:h:x_R,y_L:k:y_R);
>>> contour(X, Y, H(X,Y), [c c])
>>> axis square, xlabel 'x', ylabel 'y'
>>> title 'Level Set for H(x, y) = H(x_I, y_I)'
```

**Example.** Consider the initial-value problem

\[
\frac{dx}{dt} = \frac{e^x \cos(t)}{1 + x}, \quad x(0) = -2.
\]

Graph its solution over the time interval \([-4, 4]\).

**Solution.** In Section 4.4 we used the fact this equation is separable to show that the solution of this initial-value problem satisfies

\[
\sin(t) = -(2 + x)e^{-x}.
\]

The analysis in Section 4.4 showed moreover that there is a unique oscillatory solution \(x = X(t) < -1\) to this equation with interval of definition \((-\infty, \infty)\). However, other solutions lie on this level set that have \(x > -1\). We can plot the level set by the following.

```matlab
>>> [T, X] = meshgrid(-4:0.1:4,-4:0.1:4);
>>> contour(T, X, sin(T) + (2 + X).*exp(-X), [0 2 3/exp(1) 4/exp(2)])
>>> axis square, xlabel 't', ylabel 'x'
>>> title 'Solutions of \(dy/dt = \exp(-x) \cos(x) / (1 + x)\), \(x(0) = -2\)'
```

We can plot the level sets for the \(c\) values \(c_1, c_2,\) and \(c_3\) on a single graph by the following.

```matlab
>>> [X, Y] = meshgrid(x_L:h:x_R,y_L:k:y_R);
>>> contour(X, Y, H(X,Y), [c1 c2 c3])
>>> axis square, xlabel 'x', ylabel 'y'
>>> title 'Level Sets for H(x, y) = c_1, c_2, c_3'
```

**Example.** Consider the general initial-value problem

\[
\frac{dx}{dt} = \frac{e^x \cos(t)}{1 + x}, \quad x(0) = x_I.
\]

Graph its solutions over the time interval \([-4, 4]\) for \(x_I = -2, 0, 1,\) and \(2\).

**Solution.** In Section 4.4 we showed that the solution of this initial-value problem satisfies

\[
\sin(t) = (2 + x_I)e^{-x_I} - (2 + x)e^{-x}.
\]

This equation has the form

\[
\sin(t) + (2 + x)e^{-x} = c,
\]

where the values of \(c\) corresponding to \(x_I = -2, 0, 1,\) and \(2\) are \(c = 0, 2, 3/e\) and \(4/e^2\) respectively. We can plot these level sets by the following.

```matlab
>>> [T, X] = meshgrid(-4:0.1:4,-4:0.1:4);
>>> contour(T, X, sin(T) + (2 + X).*exp(-X), [0 2 3/exp(1) 4/exp(2)])
>>> axis square, xlabel 't', ylabel 'x'
>>> title 'Solutions of \(dy/dt = \exp(-x) \cos(x) / (1 + x)\), \(x(0) = x_I\)'
```
6.4. Direction Fields. This is the crudest tool in your toolbox. Its virtue is that it can be applied to almost any first-order equation

$$\frac{dy}{dt} = f(t, y).$$

We assume that the function $f(t, y)$ is defined over a set $S$ in the $ty$-plane such that

- $f$ is continuous over $S$,
- $f$ is differentiable with respect to $y$ over $S$,
- $\partial_y f$ is continuous over $S$.

Moreover, we assume that every point in a rectangle $[t_L, t_R] \times [y_L, y_R]$ is in the interior of $S$. Then by Theorem 5.1 every point $(t_I, y_I)$ in $[t_L, t_R] \times [y_L, y_R]$ has a unique curve $(t, Y(t))$ passing through it such that $y = Y(t)$ is a solution of (6.3). This curve can be extended to the largest time interval $[a, b]$ such that $(t, Y(t))$ remains within the rectangle $[t_L, t_R] \times [y_L, y_R]$.

If we cannot find an explicit or implicit solution of (6.3) then we cannot plot the curve $(t, Y(t))$ by the methods of the previous two sections. However, if $f(t, y)$ meets the criteria given above then we know by Theorem 5.1 that the curve exists and has a tangent vector given by

$$\frac{d}{dt} \begin{pmatrix} 1 \\ Y(t) \end{pmatrix} = \begin{pmatrix} 1 \\ Y'(t) \end{pmatrix} = \begin{pmatrix} 1 \\ f(t, Y(t)) \end{pmatrix}.$$ 

In other words, the unique solution that goes through any point $(t, y)$ in the rectangle $[t_L, t_R] \times [y_L, y_R]$ has the tangent vector $(1, f(t, y))$. A direction field for equation (6.3) over the rectangle $[t_L, t_R] \times [y_L, y_R]$ is a plot that shows the direction of this tangent vector with an arrow at each point of a grid in the rectangle $[t_L, t_R] \times [y_L, y_R]$. The idea is that these arrows might give us a correct picture of how the orbits move inside the rectangle.

We can produce such a direction field by using the MATLAB commands `meshgrid` and `quiver` as follows.

```matlab
>> [T, Y] = meshgrid(t_L:h:t_R, y_L:k:y_R);
>> S = f(T, Y);
>> L = sqrt(1 + S.^2);
>> quiver(T, Y, 1./L, S./L, ℓ)
>> axis tight, xlabel 't', ylabel 'y'
>> title 'Direction Field for dy/dt = f(t, y)'
```

Here $h$ and $k$ are the grid spacings for the intervals $[t_L, t_R]$ and $[y_L, y_R]$ respectively, which should have values of the form

$$h = \frac{t_R - t_L}{m}, \quad k = \frac{y_R - y_L}{n},$$

where $m$ and $n$ are positive integers.

The meshgrid command creates an array of grid points in the rectangle $[t_L, t_R] \times [y_L, y_R]$ given by $(t_i, y_j)$ where

$$t_i = t_L + ih \text{ for } i = 0, 1, \cdots, m, \quad y_j = y_L + jk \text{ for } j = 0, 1, \cdots, n.$$ 

More precisely, meshgrid creates two arrays; the array $T$ contains $t_i$ in its $ij^{th}$-entry while the array $Y$ contains $y_j$ in its $ij^{th}$-entry. Next, an array $S$ is computed that contains the slope $f(t_i, y_j)$ in its $ij^{th}$-entry. Then an array $L$ is computed that contains the length of the tangent vector $(1, f(t_i, y_j))$ in its $ij^{th}$-entry.
Finally, the quiver command plots an array of arrows of length $\ell$ so that the $ij$th-arrow is centered at the grid point $(t_i, y_j)$ and is pointing in the direction of the unit tangent vector

$$\left(\frac{1}{\sqrt{1 + f(t_i, y_j)^2}}, \frac{f(t_i, y_j)}{\sqrt{1 + f(t_i, y_j)^2}}\right).$$

The length $\ell$ should be smaller than $h$ or $k$ so that the plotted arrows will not overlap. Typically $m$ and $n$ will be about 20 to insure there will be enough arrows to give a complete picture of the direction field, but not so many that the plot becomes cluttered.

**Remark.** Often it is hard to figure out how the orbits move from the arrows in a direction field. Therefore they should be used only as a tool of last resort. They should never by used for autonomous equations because phase-line portraits are much easier to use.

**Example.** Describe the solutions of the equation

$$\frac{dy}{dt} = \sqrt{1 + t^2 + y^2}.$$ 

**Solution.** This equation is not linear or separable, so it cannot be solved by the analytic methods we have studied. Earlier we showed that this equation meets the criteria of Theorem 5.1, so there is a unique solution that passes through every point in the $ty$-plane. Therefore the only method we have discussed that applies to this equation is direction fields.

Before applying the method of direction fields, we should see what information can be seen directly from the equation. Its right-hand side satisfies $\sqrt{1 + t^2 + y^2} \geq 1$. This means its solutions will all be increasing functions of $t$. If our direction fields are not consistent with this observation then we will know we have made a mistake in our MATLAB program.

We can produce a direction field for this equation in the rectangle $[-5, 5] \times [-5, 5]$ with a grid spacing of 0.5 (by taking $m = n = 20$) and arrows of length 0.35 as follows.

```matlab
>> [T, Y] = meshgrid(-5:0.5:5,-5:0.5:5);
>> S = sqrt(1 + T.^2 + Y.^2);
>> L = sqrt(1 + S.^2);
>> quiver(T, Y, 1./L, S./L, 0.35);
>> axis tight, xlabel 't', ylabel 'y'
>> title 'Direction Field for $dy/dt = \sqrt{1 + t^2 + y^2}$'
```

In this case it is easy to figure out how the orbits move from the direction field.
7. First-Order Equations: Numerical Methods

7.1. Numerical Approximations. Analytic methods are either difficult or impossible to apply to many first-order differential equations. In such cases direction fields might be the only graphical method that we have covered that can be applied. However, it can be very hard to understand how any particular solution behaves from the direction field of an equation. If one is interested in understanding how a particular solution behaves it is often easiest to first use numerical methods to construct accurate approximations to the solution. These approximations can then be graphed as you would an explicit solution.

Suppose we are interested in the solution $Y(t)$ of the initial-value problem
\[ \frac{dy}{dt} = f(t, y), \quad y(t_I) = y_I, \]
over the time interval $[t_I, t_F]$ — i.e. for $t_I \leq t \leq t_F$. Here $t_I$ is called the *initial time* while $t_F$ is called the *final time*. A numerical method selects times $\{t_n\}_{n=0}^N$ such that
\[ t_I = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = t_F, \]
and computes values $\{y_n\}_{n=0}^N$ such that
\[ y_0 = Y(t_0) = y_I, \]
\[ y_n \text{ approximates } Y(t_n) \text{ for } n = 1, 2, \ldots, N. \]

For good numerical methods, these approximations will improve as $N$ increases. So for sufficiently large $N$ you can plot the points $\{(t_n, y_n)\}_{n=0}^N$ in the $(t, y)$-plane and “connect the dots” to get an accurate picture of how $Y(t)$ behaves over the time interval $[t_I, t_F]$.

Here we will introduce a few basic numerical methods in simple settings. The numerical methods used in software packages such as MATLAB are generally far more sophisticated than those we will study here. They are however built upon the same fundamental ideas as the simpler methods we will study. Throughout this section we will make the following two basic simplifications.

- We will employ *uniform time steps*. This means that given $N$ we set
\[ h = \frac{t_F - t_I}{N}, \quad \text{and} \quad t_n = t_I + nh \quad \text{for } n = 0, 1, \ldots, N; \]
where $h$ is called the *time step*.

- We will employ *one-step methods*. This means that given $f(t, y)$ and $h$ the value of $y_{n+1}$ for $n = 0, 1, \ldots, N - 1$ will depend only on $y_n$.

Sophisticated software packages use methods in which the time step is chosen adaptively. In other words, the choice of $t_{n+1}$ will depend on the behavior of recent approximations — for example, on $(t_n, y_n)$ and $(t_{n-1}, y_{n-1})$. Employing uniform time steps will greatly simplify the algorithms, and thereby simplify the programming you will have to do. If you do not like the way a run looks, you will simply try again with a larger $N$.

Similarly, sophisticated software packages will sometimes use so-called *multi-step methods* for which the value of $y_{n+1}$ for $n = m, m+1, \ldots, N - 1$ will depend on $y_n, y_{n-1}, \ldots, y_{n-m}$ for some positive integer $m$. Employing one-step methods will again simplify the algorithms, and thereby simplify the programming you will have to do.
7.2. Explicit and Implicit Euler Methods. The simplest (and the least accurate) numerical methods are the Euler methods. These can be derived in many ways. Here we give a simple approach based on the definition of the derivative through difference quotients.

If we start with the fact that
\[
\lim_{h \to 0} \frac{Y(t+h) - Y(t)}{h} = \frac{dY}{dt}(t) = f(t, Y(t)),
\]
then for small positive \( h \) one has
\[
Y(t+h) - Y(t) \approx \frac{f(t, Y(t))}{h},
\]
Upon solving this for \( Y(t+h) \) we find that
\[
Y(t+h) \approx Y(t) + hf(t, Y(t)).
\]
If we let \( t = t_n \) above (so that \( t + h = t_{n+1} \)) this is equivalent to
\[
Y(t_{n+1}) \approx Y(t_n) + hf(t_n, Y(t_n)).
\]
Because \( y_n \) and \( y_{n+1} \) approximate \( Y(t_n) \) and \( Y(t_{n+1}) \) respectively, this suggests setting
\[
y_{n+1} = y_n + hf(t_n, y_n) \quad \text{for } n = 0, 1, \ldots, N - 1.
\]
This so-called Euler method was introduced by Euler in the mid 1700’s.

Alternatively, we could have started with the fact that
\[
\lim_{h \to 0} \frac{Y(t) - Y(t-h)}{h} = \frac{dY}{dt}(t) = f(t, Y(t)),
\]
then for small positive \( h \) one has
\[
Y(t) - Y(t-h) \approx \frac{f(t, Y(t))}{h},
\]
Upon solving this for \( Y(t-h) \) we find that
\[
Y(t-h) \approx Y(t) - hf(t, Y(t)).
\]
If we let \( t = t_{n+1} \) above (so that \( t - h = t_n \)) this is equivalent to
\[
Y(t_{n+1}) - hf(t_{n+1}, Y(t_{n+1})) \approx Y(t_n).
\]
Because \( y_n \) and \( y_{n+1} \) approximate \( Y(t_n) \) and \( Y(t_{n+1}) \) respectively, this suggests setting
\[
y_{n+1} - hf(t_{n+1}, y_{n+1}) = y_n \quad \text{for } n = 0, 1, \ldots, N - 1.
\]
This method is called the implicit Euler or backward Euler method. It is called the implicit Euler method because equation (7.4) implicitly relates \( y_{n+1} \) to \( y_n \). It is called the backward Euler method because the difference quotient upon which it is based steps backward in time (from \( t \) to \( t-h \)). In contrast, the Euler method (7.3) sometimes called the explicit Euler or forward Euler method because it gives \( y_{n+1} \) explicitly and because the difference quotient upon which it is based steps forward in time (from \( t \) to \( t+h \)).

The implicit Euler method can be very inefficient unless equation (7.4) can be explicitly solved for \( y_{n+1} \). This can be done when \( f(t, y) \) is a fairly simple function if \( y \). For example, it can be done when \( f(t, y) \) is linear or quadratic in either \( y \) or \( \sqrt{y} \). However, there are equations for which the implicit Euler method will outperform the (explicit) Euler method.
7.3. Explicit Methods Based on Taylor Approximation. The explicit (or forward) Euler method can be understood as the first in a sequence of explicit methods that can be derived from the Taylor approximation formula.

7.3.1. Explicit Euler Method Revisited. The explicit Euler method can be derived from the first-order Taylor approximation, which is also known as the tangent line approximation. This approximation states that if \( Y(t) \) is twice continuously differentiable then

\[
Y(t + h) = Y(t) + h \frac{dY}{dt}(t) + O(h^2).
\]

Here the \( O(h^2) \) means that the remainder vanishes at least as fast as \( h^2 \) as \( h \) tends to zero. It is clear from (7.5) that for small positive \( h \) one has

\[
Y(t + h) \approx Y(t) + h \frac{dY}{dt}(t).
\]

Because \( Y(t) \) satisfies (7.1), this is the same as

\[
Y(t + h) \approx Y(t) + hf(t, Y(t)).
\]

If we let \( t = t_n \) above (so that \( t + h = t_{n+1} \)) this is equivalent to

\[
Y(t_{n+1}) \approx Y(t_n) + hf(t_n, Y(t_n)).
\]

Because \( y_n \) and \( y_{n+1} \) approximate \( Y(t_n) \) and \( Y(t_{n+1}) \) respectively, this suggests setting

\[
y_{n+1} = y_n + hf(t_n, y_n) \quad \text{for } n = 0, 1, \ldots, N - 1,
\]

which is exactly the Euler method (7.3).

7.3.2. Local and Global Errors. One advantage of viewing the Euler method through the tangent line approximation (7.5) is that you gain some understanding of how its error behaves as you increase \( N \), the number of time steps — or what is equivalent by (7.2), as you decrease \( h \), the time step. The \( O(h^2) \) term in (7.5) represents the local error, which is error the approximation makes at each step.

Roughly speaking, if you halve the time step \( h \) then by (7.5) the local error will reduce by a factor of four, while by (7.2) the number of steps \( N \) you must take to get to a prescribed time (say \( t_F \)) will double. If we assume that the errors add (which is often the case) then the error at \( t_F \) will reduce by a factor of two. In other words, doubling the number of time steps will reduce the error by about a factor of two. Similarly, tripling the number of time steps will reduce the error by about a factor of three. Indeed, it can be shown (but we will not do so) that the error of the explicit Euler method is \( O(h) \) over the interval \([t_I, t_F]\). The best way to think about this is that if you take \( N \) steps and the error made at each step is \( O(h^2) \) then you can expect that the accumulation of the local errors will lead to a global error of \( O(h^2)N \). Because (7.2) states that \( hN = t_F - t_I \), which is a fixed number that is independent of \( h \) and \( N \), you thereby see that global error of the explicit Euler method is \( O(h) \). Moreover, the error of the implicit Euler method behaves the same way.

Because the global error tells you how fast a method converges over the entire interval \([t_I, t_F]\), it is a more meaningful concept than local error. We therefore identify the order of a method by the order of its global error. In particular, methods like the Euler methods with global errors of \( O(h) \) are first-order methods. By the reasoning of the previous paragraph, methods whose local error is \( O(h^{m+1}) \) will have a global error of \( O(h^{m+1})N = O(h^m) \) and are thereby \( m^{th} \)-order methods.
Higher order methods are more complicated than the explicit Euler method. The hope is that this cost is overcome by the fact that its error improves faster as you increase $N$ — or what is equivalent by (7.2), as you decrease $h$. For example, if you halve the time step $h$ of a fourth-order method then the global error will reduce by a factor of sixteen. Similarly, tripling the number of time steps will reduce the error by about a factor of 81.

7.3.3. **Higher-Order Taylor-Based Methods.** The second-order Taylor approximation states that if $Y(t)$ is thrice continuously differentiable then

$$
(7.7) \quad Y(t + h) = Y(t) + h \frac{dY}{dt}(t) = \frac{1}{2} h^2 \frac{d^2Y}{dt^2}(t) + O(h^3).
$$

Here the $O(h^3)$ means that the remainder vanishes at least as fast as $h^3$ as $h$ tends to zero. It is clear from (7.7) that for small positive $h$ one has

$$
(7.8) \quad Y(t + h) \approx Y(t) + h \frac{dY}{dt}(t) + \frac{1}{2} h^2 \frac{d^2Y}{dt^2}(t).
$$

Because $Y(t)$ satisfies (7.1), we see by the chain rule from multivariable calculus that

$$
\frac{d^2Y}{dt^2}(t) = \frac{d}{dt} \left( \frac{dY}{dt}(t) \right) = \frac{d}{dt} f(t, Y(t)) = \partial_t f(t, Y(t)) + \frac{dY}{dt}(t) \partial_y f(t, Y(t))
$$

$$
= \partial_t f(t, Y(t)) + f(t, Y(t)) \partial_y f(t, Y(t)).
$$

Hence, equation (7.8) is the same as

$$
Y(t + h) \approx Y(t) + h f(t, Y(t)) + \frac{1}{2} h^2 \left( \partial_t f(t, Y(t)) + f(t, Y(t)) \partial_y f(t, Y(t)) \right).
$$

If we let $t = t_n$ above (so that $t + h = t_{n+1}$) this is equivalent to

$$
Y(t_{n+1}) \approx Y(t_n) + h f(t_n, Y(t_n)) + \frac{1}{2} h^2 \left( \partial_t f(t_n, Y(t_n)) + f(t_n, Y(t_n)) \partial_y f(t_n, Y(t_n)) \right).
$$

Because $y_n$ and $y_{n+1}$ approximate $Y(t_n)$ and $Y(t_{n+1})$ respectively, this suggests setting

$$
(7.9) \quad y_{n+1} = y_n + h f(t_n, y_n) + \frac{1}{2} h^2 \left( \partial_t f(t_n, y_n) + f(t_n, y_n) \partial_y f(t_n, y_n) \right)
$$

for $n = 0, 1, \ldots, N - 1$.

We call this the second-order Taylor-based method.

**Remark.** We can generalize our derivation of the second-order Taylor-based method by using the $n^{th}$-order Taylor approximation to derive an explicit numerical method whose error is $O(h^n)$ over the interval $[t_f, t_p]$ — a so-called $n^{th}$-order method. However, the formulas for these methods grow in complexity. For example, the third-order method is

$$
y_{n+1} = y_n + h f(t_n, y_n) + \frac{1}{2} h^2 \left( \partial_t f(t_n, y_n) + f(t_n, y_n) \partial_y f(t_n, y_n) \right)
$$

$$
+ \frac{1}{6} h^3 \left[ \partial_{tt} f(t_n, y_n) + 2 f(t_n, y_n) \partial_y f(t_n, y_n) + f(t_n, y_n)^2 \partial_{yy} f(t_n, y_n) \right]
$$

$$
+ \left( \partial_t f(t_n, y_n) + f(t_n, y_n) \partial_y f(t_n, y_n) \right) \partial_x f(t_n, y_n)
$$

for $n = 0, 1, \ldots, N - 1$.

This complexity makes them far less practical for general algorithms than the next class of methods we will study.
7.4. **Explicit Methods Based on Quadrature.** The starting point for our next class of methods will be the Fundamental Theorem of Calculus — specifically, the fact that

\[ Y(t + h) - Y(t) = \int_t^{t+h} \frac{dY}{dt}(s) \, ds. \]

Because \( Y(t) \) satisfies (7.1), this becomes

\[ Y(t + h) - Y(t) = \int_t^{t+h} f(s, Y(s)) \, ds. \]  

(7.11)

The idea is to replace the definite integral on the right-hand side above with a numerical approximation — a so-called numerical quadrature. Specifically, we will employ four basic numerical quadrature rules that are covered in most calculus courses: the left-hand rule, the trapezoidal rule, the midpoint rule, and the Simpson rule.

7.4.1. **Explicit Euler Method Revisited Again.** The left-hand rule approximates the definite integral on the right-hand side of (7.11) as

\[ \int_t^{t+h} f(s, Y(s)) \, ds = hf(t, Y(t)) + O(h^2), \]

whereby you see that (7.11) becomes

\[ Y(t + h) = Y(t) + hf(t, Y(t)) + O(h^2). \]

If we let \( t = t_n \) above (so that \( t + h = t_{n+1} \)) this is equivalent to

\[ Y(t_{n+1}) = Y(t_n) + hf(t_n, Y(t_n)) + O(h^2). \]

Because \( y_n \) and \( y_{n+1} \) approximate \( Y(t_n) \) and \( Y(t_{n+1}) \) respectively, this suggests setting

\[ y_{n+1} = y_n + hf(t_n, y_n) \quad \text{for } n = 0, 1, \ldots, N - 1, \]

which is exactly the forward Euler method (7.3).

In practice, the forward Euler method is implemented by initializing \( y_0 = y_I \) and then for \( n = 0, \ldots, N - 1 \) cycling through the instructions

\[ f_n = f(t_n, y_n), \quad y_{n+1} = y_n + hf_n, \]

where \( t_n = t_I + nh. \)

**Example.** Let \( Y(t) \) be the solution of the initial-value problem

\[ \frac{dy}{dt} = t^2 + y^2, \quad y(0) = 1. \]

Use the forward Euler method with \( h = .1 \) to approximate \( Y(.2). \)

**Solution.** We initialize \( t_0 = 0 \) and \( y_0 = 1. \) The forward Euler method then gives

\[ f_0 = f(t_0, y_0) = 0^2 + 1^2 = 1 \]
\[ y_1 = y_0 + hf_0 = 1 + .1 \cdot 1 = 1.1 \]
\[ f_1 = f(t_1, y_1) = (.1)^2 + (1.1)^2 = .01 + 1.21 = 1.22 \]
\[ y_2 = y_1 + hf_1 = 1.1 + .1 \cdot 1.22 = 1.1 + .122 = 1.222 \]

Therefore \( Y(.2) \approx y_2 = 1.222. \)
7.4.2. **Heun-Trapezoidal Method.** The trapezoidal rule approximates the definite integral on the right-hand side of (7.11) as
\[
\int_{t}^{t+h} f(s, Y(s)) \, ds = \frac{h}{2} [f(t, Y(t)) + f(t + h, Y(t + h))] + O(h^3).
\]
whereby you see that (7.11) becomes
\[
Y(t + h) = Y(t) + \frac{h}{2} [f(t, Y(t)) + f(t + h, Y(t + h))] + O(h^3).
\]
If you approximate the \(Y(t + h)\) on the right-hand side by the forward Euler method then
\[
Y(t + h) = Y(t) + \frac{h}{2} [f(t, Y(t)) + f(t + h, Y(t) + hf(t, Y(t)))] + O(h^3).
\]
If we let \(t = t_n\) above (so that \(t + h = t_{n+1}\)) this is equivalent to
\[
Y(t_{n+1}) = Y(t_n) + \frac{h}{2} [f(t_n, Y(t_n)) + f(t_{n+1}, Y(t_n) + hf(t_n, Y(t_n)))] + O(h^3).
\]
Because \(y_n\) and \(y_{n+1}\) approximate \(Y(t_n)\) and \(Y(t_{n+1})\) respectively, this suggests setting
\[
y_{n+1} = y_n + \frac{h}{2} [f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n))]
\text{ for } n = 0, 1, \ldots , N - 1.
\]
The book calls this the **Improved Euler** method. That name is sometimes used for other methods. Moreover, it is not very descriptive. We will call it the **Heun-trapezoidal** method, which makes its origins clearer.

In practice, the Heun-trapezoidal method is implemented by initializing \(y_0 = y_I\) and then for \(n = 0, \ldots , N - 1\) cycling through the instructions
\[
f_n = f(t_n, y_n), \quad \bar{y}_{n+1} = y_n + hf_n, \\
\tilde{f}_{n+1} = f(t_{n+1}, \bar{y}_{n+1}), \quad y_{n+1} = y_n + \frac{h}{2} [f_n + \tilde{f}_{n+1}],
\]
where \(t_n = t_I + nh\).

**Example.** Let \(y(t)\) be the solution of the initial-value problem
\[
\frac{dy}{dt} = t^2 + y^2, \quad y(0) = 1.
\]
Use the Heun-Trapezoidal method with \(h = 0.2\) to approximate \(y(2)\).

**Solution.** We initialize \(t_0 = 0\) and \(y_0 = 1\). The Heun-Trapezoidal method then gives
\[
f_0 = f(t_0, y_0) = 0^2 + 1^2 = 1, \\
\bar{y}_1 = y_0 + hf_0 = 1 + 0.2 \cdot 1 = 1.2, \\
\tilde{f}_1 = f(t_1, \bar{y}_1) = (0.2)^2 + (1.2)^2 = 0.04 + 1.44 = 1.48, \\
y_1 = y_0 + \frac{h}{2} [f_0 + \tilde{f}_1] = 1 + 0.1 \cdot (1 + 1.24) = 1 + 0.1 \cdot 2.24 = 1.224
\]
We then have \(y(2) \approx y_1 = 1.224\).

**Remark.** Notice that two steps of the forward Euler method with \(h = 0.1\) yielded the approximation \(y(2) = 1.222\), while one step of the Heun-trapezoidal method with \(h = 0.2\) yielded the approximation \(y(2) = 1.224\), which is closer to the exact value. As these two calculations required roughly the same computational effort, this shows the advantage of using the second-order method.
7.4.3. Heun-Midpoint Method. The midpoint rule approximates the definite integral on the right-hand side of (7.11) as
\[ \int_{t}^{t+h} f(s, Y(s)) \, ds = hf(t + \frac{1}{2}h, Y(t + \frac{1}{2}h)) + O(h^3). \]
whereby you see that (7.11) becomes
\[ Y(t + h) = Y(t) + hf(t + \frac{1}{2}h, Y(t + \frac{1}{2}h)) + O(h^3). \]
If you approximate \( Y(t + \frac{1}{2}h) \) by the forward Euler method then
\[ Y(t + h) = Y(t) + hf(t + \frac{1}{2}h, Y(t)) + O(h^3). \]
If we let \( t = t_n \) above (so that \( t + h = t_{n+1} \)) this is equivalent to
\[ Y(t_{n+1}) = Y(t_n) + hf(t_{n+\frac{1}{2}}, Y(t_n)) + O(h^3), \]
where \( t_{n+\frac{1}{2}} = t_n + \frac{1}{2}h = t_f + (n + \frac{1}{2})h. \) Because \( y_n \) and \( y_{n+1} \) approximate \( Y(t_n) \) and \( Y(t_{n+1}) \) respectively, this suggests setting
\[ y_{n+1} = y_n + hf(t_{n+\frac{1}{2}}, y_n + \frac{1}{2}hf(t_n, y_n)) \quad \text{for } n = 0, 1, \ldots, N - 1. \]
The book calls this the Modified Euler method. That name is sometimes used for other methods. Moreover, it is not very descriptive. We will call it the Heun-midpoint method, which makes its origins clearer.

In practice, the Heun-midpoint method is implemented by initializing \( y_0 = y_t \) and then for \( n = 0, \ldots, N - 1 \) cycling through the instructions
\[ f_n = f(t_n, y_n), \quad y_{n+\frac{1}{2}} = y_n + \frac{1}{2}hf_n, \]
\[ f_{n+\frac{1}{2}} = f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}), \quad y_{n+1} = y_n + hf_{n+\frac{1}{2}}, \]
where \( t_n = t_f + nh \) and \( t_{n+\frac{1}{2}} = t_f + (n + \frac{1}{2})h. \)

Example. Let \( y(t) \) be the solution of the initial-value problem
\[ \frac{dy}{dt} = t^2 + y^2, \quad y(0) = 1. \]
Use the Heun-Midpoint method with \( h = .2 \) to approximate \( y(.2) \).

Solution. We initialize \( t_0 = 0 \) and \( y_0 = 1 \). The Heun-Midpoint method then gives
\[ f_0 = f(t_0, y_0) = 0^2 + 1^2 = 1 \]
\[ \tilde{y}_{\frac{1}{2}} = y_0 + \frac{1}{2}hf_0 = 1 + .1 \cdot 1 = 1.1 \]
\[ f_{\frac{1}{2}} = f(t_{\frac{1}{2}}, \tilde{y}_{\frac{1}{2}}) = (.1)^2 + (1.1)^2 = .01 + 1.21 = 1.22 \]
\[ y_1 = y_0 + hf_{\frac{1}{2}} = 1 + .2 \cdot (1.22) = 1 + .244 = 1.244 \]
We then have \( y(.2) \approx y_1 = 1.244. \)

Remark. Notice that one step of the Heun-trapezoidal method with \( h = .2 \) yielded the approximation \( y(.2) = 1.224 \), while one step of the Heun-midpoint method with \( h = .2 \) yielded the approximation \( y(.2) = 1.244 \), which is closer to the exact value. Even though both methods are second-order, the Heun-midpoint method will often outperform the Heun-trapezoidal by roughly a factor of two.
Remark. One step of the Runge-Kutta method with such arithmetic calculations to nine decimal places on an exam. We initialize $y(t) = 0$ and then for $n = 0, \cdots, N - 1$ cycling through the instructions

\begin{align*}
    f_n &= f(t_n, y_n), \\
    2 f_n &= f(t_n + \frac{1}{2} h, \tilde{y}_n), \\
    f_n + \frac{1}{2} f_n &= f(t_n + \frac{1}{2} h, \frac{1}{2} y_n + \frac{1}{2} \tilde{y}_n), \\
    y_n + \frac{1}{2} h f_n &= y_n + h f_n + \frac{1}{2} \tilde{y}_n + \tilde{f}_n + \tilde{f}_n + 1,
\end{align*}

where $t_n = t_l + nh$ and $t_{n+\frac{1}{2}} = t_l + (n + \frac{1}{2})h$.

Example. Let $y(t)$ be the solution of the initial-value problem

\[ \frac{dy}{dt} = t^2 + y^2, \quad y(0) = 1. \]

Use the Runge-Kutta method with $h = .2$ to approximate $y(.2)$.

Solution. We initialize $t_0 = 0$ and $y_0 = 1$. The Runge-Kutta method then gives

\begin{align*}
    f_0 &= f(t_0, y_0) = 0^2 + 1^2 = 1, \\
    \tilde{y}_0 &= y_0 + \frac{1}{2} h f_0 = 1 + .1 \cdot 1 = 1.1, \\
    \tilde{f}_0 &= f(t_0 + \frac{1}{2} h, \tilde{y}_0) = (.1)^2 + (1.1)^2 = .01 + 1.21 = 1.22, \\
    y_0 &= y_0 + \frac{1}{2} h \tilde{f}_0 = 1 + .1 \cdot 1.22 = 1.122, \\
    f_0 &= f(t_0 + \frac{1}{2} h, y_0) = (.1)^2 + (1.122)^2 = .01 + 1.258884 = 1.268884, \\
    y_1 &= y_0 + h f_0 = 1 + .2 \cdot 1.268884 = 1 + .2517768 = 1.2517768, \\
    \tilde{f}_1 &= f(t_0 + h, y_1) = (.2)^2 + (1.2517768)^2 \approx .04 + 1.566945157 = 1.606945157, \\
    y_1 &= y_0 + \frac{1}{6} h [f_0 + 2 \tilde{f}_0 + 2 \tilde{f}_0 + \tilde{f}_1] \\
    &\approx 1 + .0333333333 [1 + 2 \cdot 1.22 + 2 \cdot 1.26888 + 1.606945157].
\end{align*}

We then have $y(.2) \approx y_1 \approx 1.252823772$. Of course, you would not be expected to carry out such arithmetic calculations to nine decimal places on an exam.

Remark. One step of the Runge-Kutta method with $h = .2$ yielded the approximation $y(.2) \approx 1.252823772$. This is more accurate than the approximations we had obtained with either second-order method. However, that is not a fair comparison because the Runge-Kutta method required roughly twice the computational work. A better comparison would be with the approximation produced by two steps of either second-order method with $h = .1$. 

\[ \int_t^{t+h} f(s, Y(s)) ds = \frac{h}{6} \left[ f(t, Y(t)) + 4 f(t + \frac{1}{3} h, Y(t + \frac{1}{3} h)) + \frac{5}{6} f(t + h, Y(t + h)) \right] + O(h^5), \]

whereby you see that $(7.11)$ becomes

\[ Y(t + h) = Y(t) + \frac{h}{6} \left[ f(t, Y(t)) + 4 f(t + \frac{1}{3} h, Y(t + \frac{1}{3} h)) + \frac{5}{6} f(t + h, Y(t + h)) \right] + O(h^5). \]

This leads to the Runge-Kutta method. We will not give more details about the derivation this method. You can find them in books on numerical methods.

In practice the Runge-Kutta method is implemented by initializing $y_0 = y_I$ and then for $n = 0, \cdots, N - 1$ cycling through the instructions

\begin{align*}
    f_n &= f(t_n, y_n), \\
    \tilde{f}_n &= f(t_n + \frac{1}{2} h, \tilde{y}_n), \\
    \tilde{f}_n &= f(t_n + \frac{1}{2} h, \frac{1}{2} y_n + \frac{1}{2} \tilde{y}_n), \\
    y_n + \frac{1}{2} h \tilde{f}_n &= y_n + h f_n + \frac{1}{2} \tilde{y}_n + \tilde{f}_n + \tilde{f}_n + 1,
\end{align*}

where $t_n = t_l + nh$ and $t_{n+\frac{1}{2}} = t_l + (n + \frac{1}{2})h$. 

\[ \frac{dy}{dt} = t^2 + y^2, \quad y(0) = 1. \]