MATH 350: Introduction to Computational Mathematics

Chapter VII: Numerical Differentiation and Solution of Ordinary Differential Equations

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Spring 2011

Outline

Outline

Introduction

In this chapter we are mostly concerned with the numerical solution of ODEs.

We've already seen that many mathematical models lead to (systems of) differential equations. Recall our examples in Chapter 1 on modeling.

Introduction

In this chapter we are mostly concerned with the numerical solution of ODEs.

We've already seen that many mathematical models lead to (systems of) differential equations. Recall our examples in Chapter 1 on modeling.

Now we finally want to tackle these problems!

Introduction

In this chapter we are mostly concerned with the numerical solution of ODEs.

We've already seen that many mathematical models lead to (systems of) differential equations. Recall our examples in Chapter 1 on modeling.

Now we finally want to tackle these problems!

However, first we think about how to deal with derivatives numerically since being able to do this accurately is often essential for a good ODE or PDE solver.

Recall the definition of the derivative:

Recall the definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

Recall the definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

As with the Riemann sum definition of integrals, we turn this into a numerical method by dropping the limit:

$$f'(x) \approx D_h f(x) = \frac{f(x+h) - f(x)}{h},$$

the forward difference method.

Recall the definition of the derivative:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}.$$

As with the Riemann sum definition of integrals, we turn this into a numerical method by dropping the limit:

$$f'(x) \approx D_h f(x) = \frac{f(x+h) - f(x)}{h},$$

the forward difference method.

Example

Note that this method is exact for linear functions.
Let $f(x) = mx + b$ so that $f'(x) = m$. Then

Recall the definition of the derivative:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}.$$

As with the Riemann sum definition of integrals, we turn this into a numerical method by dropping the limit:

$$f'(x) \approx D_h f(x) = \frac{f(x+h)-f(x)}{h},$$

the forward difference method.

Example

Note that this method is exact for linear functions.
Let $f(x) = mx + b$ so that $f'(x) = m$. Then

$$D_h f(x) = \frac{f(x+h)-f(x)}{h}$$

Recall the definition of the derivative:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}.$$

As with the Riemann sum definition of integrals, we turn this into a numerical method by dropping the limit:

$$f'(x) \approx D_h f(x) = \frac{f(x+h) - f(x)}{h},$$

the forward difference method.

Example

Note that this method is exact for linear functions.
Let $f(x) = mx + b$ so that $f'(x) = m$. Then

$$D_h f(x) = \frac{f(x+h) - f(x)}{h} = \frac{[m(x+h) + b] - [mx + b]}{h}$$

Recall the definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

As with the Riemann sum definition of integrals, we turn this into a numerical method by dropping the limit:

$$f'(x) \approx D_h f(x) = \frac{f(x+h) - f(x)}{h},$$

the forward difference method.

Example

Note that this method is exact for linear functions.
Let $f(x) = mx + b$ so that $f'(x) = m$. Then

$$D_h f(x) = \frac{f(x+h) - f(x)}{h} = \frac{[m(x+h) + b] - [mx + b]}{h} = \frac{mh}{h}$$

Recall the definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

As with the Riemann sum definition of integrals, we turn this into a numerical method by dropping the limit:

$$f'(x) \approx D_h f(x) = \frac{f(x+h) - f(x)}{h},$$

the forward difference method.

Example

Note that this method is exact for linear functions.
Let $f(x) = mx + b$ so that $f'(x) = m$. Then

$$D_h f(x) = \frac{f(x+h) - f(x)}{h} = \frac{[m(x+h) + b] - [mx + b]}{h} = \frac{mh}{h} = m.$$

How accurate are forward differences in general?

How accurate are forward differences in general?

If f is at least twice differentiable, we can use a Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi),$$

where ξ is somewhere between x and $x+h$.

How accurate are forward differences in general?

If f is at least twice differentiable, we can use a Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi),$$

where ξ is somewhere between x and $x+h$.
Then

$$D_h f(x) = \frac{f(x+h) - f(x)}{h}$$

How accurate are forward differences in general?

If f is at least twice differentiable, we can use a Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi),$$

where ξ is somewhere between x and $x+h$.
Then

$$\begin{aligned} D_h f(x) &= \frac{f(x+h) - f(x)}{h} \\ &= \frac{\left[f(x) + hf'(x) + \frac{h^2}{2} f''(\xi)\right] - f(x)}{h} \end{aligned}$$

How accurate are forward differences in general?

If f is at least twice differentiable, we can use a Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi),$$

where ξ is somewhere between x and $x+h$.
Then

$$\begin{aligned} D_h f(x) &= \frac{f(x+h) - f(x)}{h} \\ &= \frac{[f(x) + hf'(x) + \frac{h^2}{2}f''(\xi)] - f(x)}{h} \\ &= \frac{hf'(x) + \frac{h^2}{2}f''(\xi)}{h} = f'(x) + \frac{h}{2}f''(\xi) \end{aligned}$$

How accurate are forward differences in general?

If f is at least twice differentiable, we can use a Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi),$$

where ξ is somewhere between x and $x+h$.
Then

$$\begin{aligned} D_h f(x) &= \frac{f(x+h) - f(x)}{h} \\ &= \frac{[f(x) + hf'(x) + \frac{h^2}{2}f''(\xi)] - f(x)}{h} \\ &= \frac{hf'(x) + \frac{h^2}{2}f''(\xi)}{h} = f'(x) + \frac{h}{2}f''(\xi) \end{aligned}$$

Therefore, the error for $D_h f$ is

How accurate are forward differences in general?

If f is at least twice differentiable, we can use a Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi),$$

where ξ is somewhere between x and $x+h$.
Then

$$\begin{aligned} D_h f(x) &= \frac{f(x+h) - f(x)}{h} \\ &= \frac{\left[f(x) + hf'(x) + \frac{h^2}{2} f''(\xi)\right] - f(x)}{h} \\ &= \frac{hf'(x) + \frac{h^2}{2} f''(\xi)}{h} = f'(x) + \frac{h}{2} f''(\xi) \end{aligned}$$

Therefore, the error for $D_h f$ is

$$E_h(x) = f'(x) - D_h f(x) = -\frac{h}{2} f''(\xi) = \mathcal{O}(h).$$

Example

Consider the function

$$f(x) = \arctan(x).$$

What is $f'(\sqrt{2})$?

Example

Consider the function

$$f(x) = \arctan(x).$$

What is $f'(\sqrt{2})$?

Solution

We can use the forward difference method to estimate this.

$$f'(\sqrt{2}) \approx D_h f(\sqrt{2}) = \frac{f(\sqrt{2}+h) - f(\sqrt{2})}{h}$$

Example

Consider the function

$$f(x) = \arctan(x).$$

What is $f'(\sqrt{2})$?

Solution

We can use the forward difference method to estimate this.

$$f'(\sqrt{2}) \approx D_h f(\sqrt{2}) = \frac{f(\sqrt{2}+h) - f(\sqrt{2})}{h} = \frac{\arctan(\sqrt{2}+h) - \arctan\sqrt{2}}{h}.$$

Example

Consider the function

$$f(x) = \arctan(x).$$

What is $f'(\sqrt{2})$?

Solution

We can use the forward difference method to estimate this.

$$f'(\sqrt{2}) \approx D_h f(\sqrt{2}) = \frac{f(\sqrt{2}+h) - f(\sqrt{2})}{h} = \frac{\arctan(\sqrt{2}+h) - \arctan\sqrt{2}}{h}.$$

On the other hand, from calculus we know that $f'(x) = \dfrac{1}{1+x^2}$

Example

Consider the function
$$f(x) = \arctan(x).$$
What is $f'(\sqrt{2})$?

Solution

We can use the forward difference method to estimate this.

$$f'(\sqrt{2}) \approx D_h f(\sqrt{2}) = \frac{f(\sqrt{2}+h) - f(\sqrt{2})}{h} = \frac{\arctan(\sqrt{2}+h) - \arctan\sqrt{2}}{h}.$$

On the other hand, from calculus we know that $f'(x) = \frac{1}{1+x^2}$ and so
$$f'(\sqrt{2}) = \frac{1}{1+(\sqrt{2})^2} = \frac{1}{3}.$$

Example

Consider the function

$$f(x) = \arctan(x).$$

What is $f'(\sqrt{2})$?

Solution

We can use the forward difference method to estimate this.

$$f'(\sqrt{2}) \approx D_h f(\sqrt{2}) = \frac{f(\sqrt{2}+h) - f(\sqrt{2})}{h} = \frac{\arctan(\sqrt{2}+h) - \arctan\sqrt{2}}{h}.$$

On the other hand, from calculus we know that $f'(x) = \frac{1}{1+x^2}$ and so

$$f'(\sqrt{2}) = \frac{1}{1+(\sqrt{2})^2} = \frac{1}{3}.$$

This is illustrated in the Maple worksheet `Differentiation.mw`.

A More Accurate Method

Let's consider the following two Taylor expansions:

A More Accurate Method

Let's consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2) \end{aligned}$$

A More Accurate Method

Let's consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \\ -f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_2) \end{aligned}$$

A More Accurate Method

Let's consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \\ -f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_2) \end{aligned}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{6}\left[f'''(\xi_1) + f'''(\xi_2)\right].$$

A More Accurate Method

Let's consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \\ -f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_2) \end{aligned}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{6}\left[f'''(\xi_1) + f'''(\xi_2)\right].$$

Solving for $f'(x)$ yields

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right].$$

A More Accurate Method

Let's consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1) \\ -f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_2) \end{aligned}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{6}\left[f'''(\xi_1) + f'''(\xi_2)\right].$$

Solving for $f'(x)$ yields

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right].$$

This gives us a new symmetric difference method:

$$f'(x) \approx \overline{D}_h f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

Our derivation of the symmetric difference shows that it can be considered as the average of a forward and backward difference method.

Our derivation of the symmetric difference shows that it can be considered as the average of a forward and backward difference method.

How accurate is this average?

Our derivation of the symmetric difference shows that it can be considered as the average of a forward and backward difference method.

How accurate is this average?

Since

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right]$$

and

$$\overline{D}_h f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

the symmetric difference method has an approximation error of

$$E_h(x) = f'(x) - \overline{D}_h f(x)$$

Our derivation of the symmetric difference shows that it can be considered as the average of a forward and backward difference method.

How accurate is this average?

Since

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right]$$

and

$$\overline{D}_h f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

the symmetric difference method has an approximation error of

$$E_h(x) = f'(x) - \overline{D}_h f(x) = -\frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right]$$

Our derivation of the symmetric difference shows that it can be considered as the average of a forward and backward difference method.

How accurate is this average?

Since

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right]$$

and

$$\overline{D}_h f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

the symmetric difference method has an approximation error of

$$E_h(x) = f'(x) - \overline{D}_h f(x) = -\frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right] = \mathcal{O}(h^2).$$

Our derivation of the symmetric difference shows that it can be considered as the average of a forward and backward difference method.

How accurate is this average?

Since

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right]$$

and

$$\overline{D}_h f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

the symmetric difference method has an approximation error of

$$E_h(x) = f'(x) - \overline{D}_h f(x) = -\frac{h^2}{12}\left[f'''(\xi_1) + f'''(\xi_2)\right] = \mathcal{O}(h^2).$$

This is illustrated in the Maple worksheet `Differentiation.mw`.

Richardson Extrapolation

Recall from Chapter 6 that Richardson extrapolation can be used to improve the accuracy of any numerical method that has an error of the form $\mathcal{O}(h^p)$.

Richardson Extrapolation

Recall from Chapter 6 that Richardson extrapolation can be used to improve the accuracy of any numerical method that has an error of the form $\mathcal{O}(h^p)$.

Therefore, we can apply Richardson extrapolation to boost the

- $\mathcal{O}(h)$ accuracy of forward differences ($p = 1$), and
- $\mathcal{O}(h^2)$ accuracy of symmetric differences ($p = 2$).

Richardson Extrapolation

Recall from Chapter 6 that Richardson extrapolation can be used to improve the accuracy of any numerical method that has an error of the form $\mathcal{O}(h^p)$.

Therefore, we can apply Richardson extrapolation to boost the

- $\mathcal{O}(h)$ accuracy of forward differences ($p = 1$), and
- $\mathcal{O}(h^2)$ accuracy of symmetric differences ($p = 2$).

This is illustrated in the Maple worksheet `Differentiation.mw`.

Richardson Extrapolation

Recall from Chapter 6 that Richardson extrapolation can be used to improve the accuracy of any numerical method that has an error of the form $\mathcal{O}(h^p)$.

Therefore, we can apply Richardson extrapolation to boost the

- $\mathcal{O}(h)$ accuracy of forward differences ($p = 1$), and
- $\mathcal{O}(h^2)$ accuracy of symmetric differences ($p = 2$).

This is illustrated in the Maple worksheet `Differentiation.mw`.

Since we have an “exact” error estimate for these methods we can be more precise about the impact of Richardson extrapolation.

Example

Had we taken the full Taylor series expansions in the derivation of symmetric differences then

Example

Had we taken the full Taylor series expansions in the derivation of symmetric differences then

$$
\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) + \ldots \\
f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) - \frac{h^5}{120}f^{(5)}(x) + \ldots
\end{aligned}
$$

Example

Had we taken the full Taylor series expansions in the derivation of symmetric differences then

$$
\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) + \dots \\
-f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) - \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) - \dots
\end{aligned}
$$

Example

Had we taken the full Taylor series expansions in the derivation of symmetric differences then

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) + \ldots \\ -f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) - \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) - \ldots \end{aligned}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3}f'''(x) + \frac{h^5}{60}f^{(5)}(x) + \ldots$$

Example

Had we taken the full Taylor series expansions in the derivation of symmetric differences then

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) + \ldots \\ -f(x-h) &= -f(x) + hf'(x) - \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) - \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) - \ldots \end{aligned}$$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3}f'''(x) + \frac{h^5}{60}f^{(5)}(x) + \ldots$$

or

$$f'(x) = \underbrace{\frac{f(x+h) - f(x-h)}{2h}}_{=\overline{D}_h f(x)} + k_2 h^2 + k_4 h^4 + \ldots,$$

where k_2, k_4, etc., are appropriate constants independent of h.

Now we can look at the accuracy of the Richardson extrapolant ($p = 2$) $\frac{4}{3}\overline{D}_{\frac{h}{2}} - \frac{1}{3}\overline{D}_h$ for symmetric differences.

Now we can look at the accuracy of the Richardson extrapolant ($p = 2$) $\frac{4}{3}\overline{D}_{\frac{h}{2}} - \frac{1}{3}\overline{D}_h$ for symmetric differences.
From the previous slide we have

$$\begin{aligned}\overline{D}_{\frac{h}{2}} f(x) &= f'(x) - k_2 \left(\frac{h}{2}\right)^2 - k_4 \left(\frac{h}{2}\right)^4 - \ldots \\ \overline{D}_h f(x) &= f'(x) - k_2 h^2 - k_4 h^4 - \ldots\end{aligned}$$

Now we can look at the accuracy of the Richardson extrapolant ($p = 2$) $\frac{4}{3}\overline{D}_{\frac{h}{2}} - \frac{1}{3}\overline{D}_h$ for symmetric differences.

From the previous slide we have

$$\begin{aligned}\overline{D}_{\frac{h}{2}} f(x) &= f'(x) - \frac{1}{4}k_2 h^2 - \frac{1}{16}k_4 h^4 - \ldots \\ \overline{D}_h f(x) &= f'(x) - k_2 h^2 - k_4 h^4 - \ldots\end{aligned}$$

Now we can look at the accuracy of the Richardson extrapolant ($p = 2$) $\frac{4}{3}\overline{D}_{\frac{h}{2}} - \frac{1}{3}\overline{D}_h$ for symmetric differences.
From the previous slide we have

$$\begin{aligned}\overline{D}_{\frac{h}{2}} f(x) &= f'(x) - \frac{1}{4}k_2 h^2 - \frac{1}{16}k_4 h^4 - \ldots \\ \overline{D}_h f(x) &= f'(x) - k_2 h^2 - k_4 h^4 - \ldots\end{aligned}$$

Multiplying by the appropriate factors gives

$$\frac{4}{3}\overline{D}_{\frac{h}{2}} f(x) = \frac{4}{3}f'(x) - \frac{1}{3}k_2 h^2 - \frac{1}{12}k_4 h^4 - \ldots \quad (1)$$

$$\frac{1}{3}\overline{D}_h f(x) = \frac{1}{3}f'(x) - \frac{1}{3}k_2 h^2 - \frac{1}{3}k_4 h^4 - \ldots \quad (2)$$

Now we can look at the accuracy of the Richardson extrapolant ($p = 2$) $\frac{4}{3}\overline{D}_{\frac{h}{2}} - \frac{1}{3}\overline{D}_h$ for symmetric differences.
From the previous slide we have

$$\begin{aligned} \overline{D}_{\frac{h}{2}} f(x) &= f'(x) - \frac{1}{4}k_2 h^2 - \frac{1}{16}k_4 h^4 - \ldots \\ \overline{D}_h f(x) &= f'(x) - k_2 h^2 - k_4 h^4 - \ldots \end{aligned}$$

Multiplying by the appropriate factors gives

$$\frac{4}{3}\overline{D}_{\frac{h}{2}} f(x) = \frac{4}{3}f'(x) - \frac{1}{3}k_2 h^2 - \frac{1}{12}k_4 h^4 - \ldots \quad (1)$$

$$\frac{1}{3}\overline{D}_h f(x) = \frac{1}{3}f'(x) - \frac{1}{3}k_2 h^2 - \frac{1}{3}k_4 h^4 - \ldots \quad (2)$$

Subtracting (2) from (1) yields

$$\frac{4}{3}\overline{D}_{\frac{h}{2}} f(x) - \frac{1}{3}\overline{D}_h f(x) = f'(x) + \frac{1}{4}k_4 h^4 + \ldots$$

Now we can look at the accuracy of the Richardson extrapolant ($p = 2$) $\frac{4}{3}\overline{D}_{\frac{h}{2}} - \frac{1}{3}\overline{D}_h$ for symmetric differences.
From the previous slide we have

$$\begin{aligned} \overline{D}_{\frac{h}{2}} f(x) &= f'(x) - \frac{1}{4}k_2 h^2 - \frac{1}{16}k_4 h^4 - \dots \\ \overline{D}_h f(x) &= f'(x) - k_2 h^2 - k_4 h^4 - \dots \end{aligned}$$

Multiplying by the appropriate factors gives

$$\frac{4}{3}\overline{D}_{\frac{h}{2}} f(x) = \frac{4}{3}f'(x) - \frac{1}{3}k_2 h^2 - \frac{1}{12}k_4 h^4 - \dots \quad (1)$$

$$\frac{1}{3}\overline{D}_h f(x) = \frac{1}{3}f'(x) - \frac{1}{3}k_2 h^2 - \frac{1}{3}k_4 h^4 - \dots \quad (2)$$

Subtracting (2) from (1) yields

$$\frac{4}{3}\overline{D}_{\frac{h}{2}} f(x) - \frac{1}{3}\overline{D}_h f(x) = f'(x) + \frac{1}{4}k_4 h^4 + \dots$$

So symmetric differences with Richardson extrapolation are $\mathcal{O}(h^4)$ accurate (see `Differentiation.mw`).

An Estimate for $f''(x)$

Let's again consider the following two Taylor expansions:

An Estimate for $f''(x)$

Let's again consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(\xi_1) \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(\xi_2) \end{aligned}$$

An Estimate for $f''(x)$

Let's again consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_1) \\ +f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_2) \end{aligned}$$

An Estimate for $f''(x)$

Let's again consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_1) \\ +f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_2) \end{aligned}$$

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{24}\left[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)\right].$$

An Estimate for $f''(x)$

Let's again consider the following two Taylor expansions:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_1) \\ +f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_2) \end{aligned}$$

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{24}\left[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)\right].$$

Solving for $f''(x)$ yields

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{24}\left[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)\right].$$

An Estimate for $f''(x)$

Let's again consider the following two Taylor expansions:

$$
\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_1) \\
+f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(\xi_2)
\end{aligned}
$$

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{24}\left[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)\right].$$

Solving for $f''(x)$ yields

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{24}\left[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)\right].$$

This gives us

$$f''(x) \approx D_h^{(2)} f(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*
   - *Replace f by a generic polynomial p that interpolates f at a chosen set of nodes $x_1, x_2, \ldots, x_n$.*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*
   - *Replace f by a generic polynomial p that interpolates f at a chosen set of nodes $x_1, x_2, \ldots, x_n$.*
   - *Differentiate p.*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*
   - *Replace f by a generic polynomial p that interpolates f at a chosen set of nodes $x_1, x_2, \ldots, x_n$.*
   - *Differentiate p.*
   - *Specify a particular choice of nodes $x_1, x_2, \ldots, x_n$ so that one of them equals the evaluation point x (see example on next slide).*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*
   - *Replace f by a generic polynomial p that interpolates f at a chosen set of nodes $x_1, x_2, \ldots, x_n$.*
   - *Differentiate p.*
   - *Specify a particular choice of nodes $x_1, x_2, \ldots, x_n$ so that one of them equals the evaluation point x (see example on next slide).*
   - *This yields the approximate formula for $f'(x)$.*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*
   - *Replace f by a generic polynomial p that interpolates f at a chosen set of nodes $x_1, x_2, \ldots, x_n$.*
   - *Differentiate p.*
   - *Specify a particular choice of nodes $x_1, x_2, \ldots, x_n$ so that one of them equals the evaluation point x (see example on next slide).*
   - *This yields the approximate formula for $f'(x)$.*
3. *The accuracy of such a method can be obtained via error estimates for the polynomial interpolant (which we didn't study).*

Remark

1. *Our approximation for f'' has $\mathcal{O}(h^2)$ accuracy. There is no analogue of symmetric differences that is more accurate.*
2. *All numerical differentiation methods that we derived can also be obtained via polynomial interpolation. The idea is similar to what we did for numerical integration:*
   - *Replace f by a generic polynomial p that interpolates f at a chosen set of nodes $x_1, x_2, \ldots, x_n$.*
   - *Differentiate p.*
   - *Specify a particular choice of nodes $x_1, x_2, \ldots, x_n$ so that one of them equals the evaluation point x (see example on next slide).*
   - *This yields the approximate formula for $f'(x)$.*
3. *The accuracy of such a method can be obtained via error estimates for the polynomial interpolant (which we didn't study).*
4. *The approach based on polynomial interpolation is very general and can be used for arbitrary degree (i.e., arbitrarily many points), arbitrarily spaced points (symmetric, non-symmetric, etc.), and arbitrary order derivatives.*

Example

- The linear interpolant of f at $(x-h, f(x-h))$ and $(x+h, f(x+h))$ has slope

$$\frac{\Delta y}{\Delta x}$$

Example

- The linear interpolant of f at $(x-h, f(x-h))$ and $(x+h, f(x+h))$ has slope

$$\frac{\Delta y}{\Delta x} = \frac{f(x+h) - f(x-h)}{(x+h) - (x-h)}$$

Example

- The linear interpolant of f at $(x-h, f(x-h))$ and $(x+h, f(x+h))$ has slope

$$\frac{\Delta y}{\Delta x} = \frac{f(x+h) - f(x-h)}{(x+h) - (x-h)} = \frac{f(x+h) - f(x-h)}{2h} = \overline{D}_h f(x).$$

Example

- The linear interpolant of f at $(x-h, f(x-h))$ and $(x+h, f(x+h))$ has slope

$$\frac{\Delta y}{\Delta x} = \frac{f(x+h)-f(x-h)}{(x+h)-(x-h)} = \frac{f(x+h)-f(x-h)}{2h} = \overline{D}_h f(x).$$

- Differentiation of the quadratic interpolant

$$p(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}f(x_1) + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}f(x_2) + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}f(x_3)$$

to f followed by evaluation at $x_1 = x-h$, $x_2 = x$ and $x_3 = x+h$ will provide the formula for $f''(x)$ from the previous slide (for details see HW).

Outline

First-order initial value problems

We will consider problems of the form

$$\begin{aligned}\frac{\mathrm{d}y(t)}{\mathrm{d}t} &= f(t, y(t)) \\ y(t_0) &= y_0.\end{aligned}$$

Here f can take many different forms.

First-order initial value problems

We will consider problems of the form

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = f(t, y(t))$$
$$y(t_0) = y_0.$$

Here f can take many different forms.

Example

If $f(t, y(t)) = a(t)y(t) + b(t)$ with given functions a and b, then we have a linear first-order initial value problem that can be solved using integrating factors.

First-order initial value problems

We will consider problems of the form

$$\begin{aligned}\frac{\mathrm{d}y(t)}{\mathrm{d}t} &= f(t, y(t)) \\ y(t_0) &= y_0.\end{aligned}$$

Here f can take many different forms.

Example

If $f(t, y(t)) = a(t)y(t) + b(t)$ with given functions a and b, then we have a linear first-order initial value problem that can be solved using integrating factors.

Namely,

$$y(t) = \frac{1}{\mu(t)} \int_{t_0}^{t} \mu(\tau) b(\tau) \mathrm{d}\tau$$

with integrating factor

$$\mu(t) = \mathrm{e}^{-\int a(t)\mathrm{d}t}.$$

Example

If, for example, $f(t, y(t)) = -[y(t)]^2$, then the problem is nonlinear.

Example

If, for example, $f(t, y(t)) = -[y(t)]^2$, then the problem is nonlinear.
This particular problem can be solved by separation.

Example

If, for example, $f(t, y(t)) = -[y(t)]^2$, then the problem is nonlinear.
This particular problem can be solved by separation.
Namely,

$$\frac{dy}{y^2} = -dt$$

Example

If, for example, $f(t, y(t)) = -[y(t)]^2$, then the problem is nonlinear.
This particular problem can be solved by separation.
Namely,

$$\frac{\mathrm{d}y}{y^2} = -\mathrm{d}t \quad \Longrightarrow \quad -\frac{1}{y} = -t + C$$

Example

If, for example, $f(t, y(t)) = -[y(t)]^2$, then the problem is nonlinear.
This particular problem can be solved by separation.
Namely,

$$\frac{\mathrm{d}y}{y^2} = -\mathrm{d}t \quad \Longrightarrow \quad -\frac{1}{y} = -t + C \quad \Longrightarrow \quad y(t) = \frac{1}{t+c}.$$

Example

If, for example, $f(t, y(t)) = -[y(t)]^2$, then the problem is nonlinear.
This particular problem can be solved by separation.
Namely,

$$\frac{dy}{y^2} = -dt \quad \Longrightarrow \quad -\frac{1}{y} = -t + C \quad \Longrightarrow \quad y(t) = \frac{1}{t + c}.$$

Remark

In many cases f will not be as simple, and numerical methods are required.

The theory — in particular existence and uniqueness theorems — for differential equations is generally a rather difficult subject.

The theory — in particular existence and uniqueness theorems — for differential equations is generally a rather difficult subject.
For first-order initial value problems we have

Theorem

Let $f = f(t, y)$ and $\frac{\partial f}{\partial y}$ be continuous near the initial point (t_0, y_0). Then there is a unique solution y defined on the interval $[t_0 - \alpha, t_0 + \alpha]$ for some α such that

$$\begin{aligned} \frac{\mathrm{d}y(t)}{\mathrm{d}t} &= f(t, y(t)) \\ y(t_0) &= y_0. \end{aligned}$$

The theory — in particular existence and uniqueness theorems — for differential equations is generally a rather difficult subject.
For first-order initial value problems we have

Theorem

Let $f = f(t, y)$ and $\frac{\partial f}{\partial y}$ be continuous near the initial point (t_0, y_0). Then there is a unique solution y defined on the interval $[t_0 - \alpha, t_0 + \alpha]$ for some α such that

$$\begin{aligned} \frac{\mathrm{d}y(t)}{\mathrm{d}t} &= f(t, y(t)) \\ y(t_0) &= y_0. \end{aligned}$$

Remark

This theorem can be found in any introductory ODE book (such as [Zill]). For a proof see for example [Boyce and DiPrima].

All numerical solvers that we will consider (especially those provided by MATLAB) will work not only for single first-order initial-value problems, but also for systems of first-order initial-value problems.

All numerical solvers that we will consider (especially those provided by MATLAB) will work not only for single first-order initial-value problems, but also for systems of first-order initial-value problems.

Example

The predator-prey model from Chapter 1 provides such a system:

$$\begin{aligned} \frac{\mathrm{d}H(t)}{\mathrm{d}t} &= aH(t) - bH(t)L(t) \\ \frac{\mathrm{d}L(t)}{\mathrm{d}t} &= -cL(t) + dH(t)L(t) \\ H(t_0) &= H_0, \quad L(t_0) = L_0, \end{aligned}$$

where t denotes time, H population of hares, L population of lynx, and a, b, c, d, H_0, L_0 are given constants.
This system is coupled and nonlinear and does not possess an analytical solution.

In addition to considering outright systems of first-order IVPs we can also rewrite any higher-order IVP as a system of first-order IVPs.

In addition to considering outright systems of first-order IVPs we can also rewrite any higher-order IVP as a system of first-order IVPs.

Example

The angle θ that a pendulum with mass m and length ℓ makes with the vertical satisfies the following second-order IVP (which is based on Newton's second law of motion):

$$\begin{aligned} m\ell\frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} &= -mg\sin(\theta(t)) \\ \theta(t_0) &= \theta_0 \\ \theta'(t_0) &= v_0. \end{aligned}$$

Here g is the gravitational constant, θ_0 is the initial angle, and v_0 the initial angular velocity.

In addition to considering outright systems of first-order IVPs we can also rewrite any higher-order IVP as a system of first-order IVPs.

Example

The angle θ that a pendulum with mass m and length ℓ makes with the vertical satisfies the following second-order IVP (which is based on Newton's second law of motion):

$$\begin{aligned} m\ell\frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} &= -mg\sin(\theta(t)) \\ \theta(t_0) &= \theta_0 \\ \theta'(t_0) &= v_0. \end{aligned}$$

Here g is the gravitational constant, θ_0 is the initial angle, and v_0 the initial angular velocity.
This problem can be converted to a system of two first-order equations.

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{\mathrm{d}\theta(t)}{\mathrm{d}t}]^T$.

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{d\theta(t)}{dt}]^T$. Then

$$\frac{d\boldsymbol{y}(t)}{dt} = \begin{bmatrix} \frac{d\theta(t)}{dt} \\ \frac{d^2\theta(t)}{dt^2} \end{bmatrix}$$

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{\mathrm{d}\theta(t)}{\mathrm{d}t}]^T$. Then

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -\frac{g}{\ell}\sin(y_1(t)) \end{bmatrix}$$

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{\mathrm{d}\theta(t)}{\mathrm{d}t}]^T$. Then

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -\frac{g}{\ell}\sin(y_1(t)) \end{bmatrix}$$

with

$$\boldsymbol{y}_0 = \begin{bmatrix} \theta(t_0) \\ \frac{\mathrm{d}\theta}{\mathrm{d}t}(t_0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ v_0 \end{bmatrix}.$$

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{d\theta(t)}{dt}]^T$. Then

$$\frac{d\boldsymbol{y}(t)}{dt} = \begin{bmatrix} \frac{d\theta(t)}{dt} \\ \frac{d^2\theta(t)}{dt^2} \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -\frac{g}{\ell}\sin(y_1(t)) \end{bmatrix}$$

with

$$\boldsymbol{y}_0 = \begin{bmatrix} \theta(t_0) \\ \frac{d\theta}{dt}(t_0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ v_0 \end{bmatrix}.$$

The same principle works for any higher-order ODE initial value problem — even for systems of higher-order ODEs.

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{\mathrm{d}\theta(t)}{\mathrm{d}t}]^T$. Then

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \begin{bmatrix} \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} \\ \frac{\mathrm{d}^2\theta(t)}{\mathrm{d}t^2} \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -\frac{g}{\ell}\sin(y_1(t)) \end{bmatrix}$$

with

$$\boldsymbol{y}_0 = \begin{bmatrix} \theta(t_0) \\ \frac{\mathrm{d}\theta}{\mathrm{d}t}(t_0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ v_0 \end{bmatrix}.$$

The same principle works for any higher-order ODE initial value problem — even for systems of higher-order ODEs.
Therefore, all we need to numerically solve any ODE initial value problem is a vectorized first-order solver.

Example (cont.)

We introduce a vector $\boldsymbol{y}(t) = [y_1(t), y_2(t)]^T$ of new variables such that $[y_1(t), y_2(t)]^T = [\theta(t), \frac{d\theta(t)}{dt}]^T$. Then

$$\frac{d\boldsymbol{y}(t)}{dt} = \begin{bmatrix} \frac{d\theta(t)}{dt} \\ \frac{d^2\theta(t)}{dt^2} \end{bmatrix} = \begin{bmatrix} y_2(t) \\ -\frac{g}{\ell}\sin(y_1(t)) \end{bmatrix}$$

with

$$\boldsymbol{y}_0 = \begin{bmatrix} \theta(t_0) \\ \frac{d\theta}{dt}(t_0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ v_0 \end{bmatrix}.$$

The same principle works for any higher-order ODE initial value problem — even for systems of higher-order ODEs.

Therefore, all we need to numerically solve any ODE initial value problem is a vectorized first-order solver.

The simplest such solver is given by `Euler.m` used in Chapter 1 (see details below).

Example

Rewrite the coupled system of second-order initial-value problems

$$[x''(t)]^2 + te^{y(t)} + y'(t) = x'(t) - x(t)$$
$$y'(t)y''(t) - \cos(x(t)y(t)) + \sin(tx'(t)y(t)) = x(t)$$
$$x(0) = a, \quad x'(0) = b, \quad y(0) = c, \quad y'(0) = d$$

as a first-order system.

Example

Rewrite the coupled system of second-order initial-value problems

$$[x''(t)]^2 + te^{y(t)} + y'(t) = x'(t) - x(t)$$
$$y'(t)y''(t) - \cos(x(t)y(t)) + \sin(tx'(t)y(t)) = x(t)$$
$$x(0) = a, \quad x'(0) = b, \quad y(0) = c, \quad y'(0) = d$$

as a first-order system.

Solution

Since there are two second-order equations we will need four new variables leading to four first-order equations.
Therefore we take $\boldsymbol{y}(t) = [y_1(t), y_2(t), y_3(t), y_4(t)]^T$ with

$$[y_1(t), y_2(t), y_3(t), y_4(t)]^T = [x(t), x'(t), y(t), y'(t)]^T.$$

Solution (cont.)

Since the ODEs are

$$[x''(t)]^2 + te^{y(t)} + y'(t) = x'(t) - x(t)$$
$$y'(t)y''(t) - \cos(x(t)y(t)) + \sin(tx'(t)y(t)) = x(t)$$

we have

$$\frac{d\boldsymbol{y}(t)}{dt} = \begin{bmatrix} x'(t) \\ x''(t) \\ y'(t) \\ y''(t) \end{bmatrix}$$

Solution (cont.)

Since the ODEs are

$$[x''(t)]^2 + te^{y(t)} + y'(t) = x'(t) - x(t)$$
$$y'(t)y''(t) - \cos\left(x(t)y(t)\right) + \sin\left(tx'(t)y(t)\right) = x(t)$$

we have

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \begin{bmatrix} x'(t) \\ x''(t) \\ y'(t) \\ y''(t) \end{bmatrix} = \begin{bmatrix} y_2(t) \\ \sqrt{y_2(t) - y_1(t) - te^{y_3(t)} - y_4(t)} \\ y_4(t) \\ \frac{y_1(t)+\cos(y_1(t)y_3(t))-\sin(ty_2(t)y_3(t))}{y_4(t)} \end{bmatrix}$$

Solution (cont.)

Since the ODEs are

$$[x''(t)]^2 + te^{y(t)} + y'(t) = x'(t) - x(t)$$
$$y'(t)y''(t) - \cos(x(t)y(t)) + \sin(tx'(t)y(t)) = x(t)$$

we have

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \begin{bmatrix} x'(t) \\ x''(t) \\ y'(t) \\ y''(t) \end{bmatrix} = \begin{bmatrix} y_2(t) \\ \sqrt{y_2(t) - y_1(t) - te^{y_3(t)} - y_4(t)} \\ y_4(t) \\ \frac{y_1(t)+\cos(y_1(t)y_3(t))-\sin(ty_2(t)y_3(t))}{y_4(t)} \end{bmatrix}$$

and

$$\boldsymbol{y}_0 = \begin{bmatrix} x(0) \\ x'(0) \\ y(0) \\ y'(0) \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}.$$

Outline

From IVPs to Integral Equations

As before, we consider the IVP

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned}$$

and integrate both sides of the differential equation from t to $t + h$ to obtain

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau)) \mathrm{d}\tau. \tag{3}$$

From IVPs to Integral Equations

As before, we consider the IVP

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned}$$

and integrate both sides of the differential equation from t to $t+h$ to obtain

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \tag{3}$$

Therefore, the solution to our IVP can be obtained by solving the integral equation (3).

From IVPs to Integral Equations

As before, we consider the IVP

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned}$$

and integrate both sides of the differential equation from t to $t+h$ to obtain

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau)) \mathrm{d}\tau. \quad (3)$$

Therefore, the solution to our IVP can be obtained by solving the integral equation (3).

Of course, we can use numerical integration to do this.

From IVPs to Integral Equations

As before, we consider the IVP

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned}$$

and integrate both sides of the differential equation from t to $t+h$ to obtain

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau. \qquad (3)$$

Therefore, the solution to our IVP can be obtained by solving the integral equation (3).
Of course, we can use numerical integration to do this.

Remark

For simplicity we limit our discussion to single ODEs of one variable. However, everything goes through analogously for the first-order systems discussed on the previous slides.

Example

Apply the left endpoint rule $\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \underbrace{h}_{=\frac{b-a}{n}} f(x_{i-1})$ on a single interval, i.e., with $n = 1$, and $a = t$, $b = t + h$ to the RHS of (3).

Example

Apply the left endpoint rule $\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \underbrace{h}_{=\frac{b-a}{n}} f(x_{i-1})$ on a single interval, i.e., with $n = 1$, and $a = t$, $b = t + h$ to the RHS of (3).

Solution

In this case the left endpoint rule is

$$L_1(f) = \sum_{i=1}^{1} \frac{b-a}{1} f(x_{i-1})$$

Example

Apply the left endpoint rule $\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \underbrace{h}_{=\frac{b-a}{n}} f(x_{i-1})$ on a single interval, i.e., with $n = 1$, and $a = t$, $b = t + h$ to the RHS of (3).

Solution

In this case the left endpoint rule is

$$L_1(f) = \sum_{i=1}^{1} \frac{b-a}{1} f(x_{i-1}) = ((t+h) - t)\, f(x_0)$$

Example

Apply the left endpoint rule $\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \underbrace{h}_{=\frac{b-a}{n}} f(x_{i-1})$ on a single interval, i.e., with $n = 1$, and $a = t$, $b = t + h$ to the RHS of (3).

Solution

In this case the left endpoint rule is

$$L_1(f) = \sum_{i=1}^{1} \frac{b-a}{1} f(x_{i-1}) = ((t+h)-t)\, f(x_0) = hf(t, y(t)).$$

Example

Apply the left endpoint rule $\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \underbrace{h}_{=\frac{b-a}{n}} f(x_{i-1})$ on a single interval, i.e., with $n = 1$, and $a = t$, $b = t + h$ to the RHS of (3).

Solution

In this case the left endpoint rule is

$$L_1(f) = \sum_{i=1}^{1} \frac{b-a}{1} f(x_{i-1}) = ((t+h)-t)\, f(x_0) = hf(t, y(t)).$$

Therefore we get $$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t, y(t)).$$

Example

Apply the left endpoint rule $\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \underbrace{h}_{=\frac{b-a}{n}} f(x_{i-1})$ on a single interval, i.e., with $n = 1$, and $a = t$, $b = t + h$ to the RHS of (3).

Solution

In this case the left endpoint rule is

$$L_1(f) = \sum_{i=1}^{1} \frac{b-a}{1} f(x_{i-1}) = ((t+h) - t)\, f(x_0) = hf(t, y(t)).$$

Therefore we get
$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t, y(t)).$$

Thus, solving (3) with the left endpoint rule is equivalent to Euler's method (see also Chapter 6).

Euler's Method

In order to advance the solution of the IVP in time we introduce a sequence of points $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ that divide a time interval $[t_0, t_N]$ into N equal subintervals (i.e., $h = \frac{t_N - t_0}{N}$).

Euler's Method

In order to advance the solution of the IVP in time we introduce a sequence of points $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ that divide a time interval $[t_0, t_N]$ into N equal subintervals (i.e., $h = \frac{t_N - t_0}{N}$).
With this notation the approximate solution of the integral equation (3)

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t, y(t))$$

immediately leads to an iterative algorithm:

Euler's Method

In order to advance the solution of the IVP in time we introduce a sequence of points $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ that divide a time interval $[t_0, t_N]$ into N equal subintervals (i.e., $h = \frac{t_N - t_0}{N}$).
With this notation the approximate solution of the integral equation (3)

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t, y(t))$$

immediately leads to an iterative algorithm:
Algorithm (see also `Euler.m`)

- Input t_0, y_0, f, h, N
- for $n = 0$ to $N-1$ do
  - $y_{n+1} = y_n + hf(t_n, y_n)$
  - $t_{n+1} = t_n + h$
- end

Euler's Method

In order to advance the solution of the IVP in time we introduce a sequence of points $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ that divide a time interval $[t_0, t_N]$ into N equal subintervals (i.e., $h = \frac{t_N - t_0}{N}$).
With this notation the approximate solution of the integral equation (3)

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t, y(t))$$

immediately leads to an iterative algorithm:

Algorithm (see also `Euler.m`)

- Input t_0, y_0, f, h, N
- for $n = 0$ to $N - 1$ do
  - $y_{n+1} = y_n + hf(t_n, y_n)$
  - $t_{n+1} = t_n + h$
- end

Here we obtain approximations $y_{n+1} \approx y(t_{n+1}) = y(t_n + h)$ of the unknown true solution y.

Euler's Method

In order to advance the solution of the IVP in time we introduce a sequence of points $t_n = t_0 + nh$, $n = 0, 1, \ldots, N$ that divide a time interval $[t_0, t_N]$ into N equal subintervals (i.e., $h = \frac{t_N - t_0}{N}$).
With this notation the approximate solution of the integral equation (3)

$$y(t+h) - y(t) = \int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t, y(t))$$

immediately leads to an iterative algorithm:
Algorithm (see also `Euler.m`)

- Input t_0, y_0, f, h, N
- for $n = 0$ to $N - 1$ do
  - $y_{n+1} = y_n + hf(t_n, y_n)$
  - $t_{n+1} = t_n + h$
- end

Here we obtain approximations $y_{n+1} \approx y(t_{n+1}) = y(t_n + h)$ of the unknown true solution y.
Euler's method is illustrated in the MATLAB script `EulerDemo350.m`.

Graphical interpretation of Euler's method

Graphically, Euler's method comes down to taking straight line approximations of the unknown solution y over small time intervals from t_n to $t_{n+1} = t_n + h$.

Graphical interpretation of Euler's method

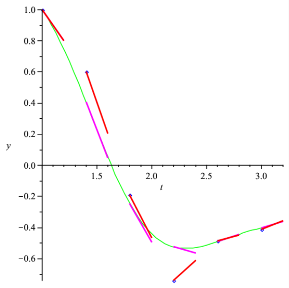
Graphically, Euler's method comes down to taking straight line approximations of the unknown solution y over small time intervals from t_n to $t_{n+1} = t_n + h$.

The slopes of these lines are given by the differential equation since it tells us that $y'(t) = f(t, y(t))$.

Graphical interpretation of Euler's method

Graphically, Euler's method comes down to taking straight line approximations of the unknown solution y over small time intervals from t_n to $t_{n+1} = t_n + h$.

The slopes of these lines are given by the differential equation since it tells us that $y'(t) = f(t, y(t))$.

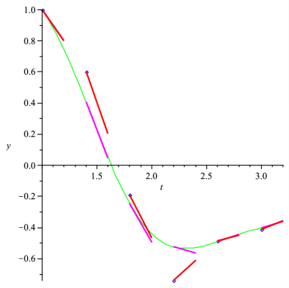


Graphical interpretation of Euler's method

Graphically, Euler's method comes down to taking straight line approximations of the unknown solution y over small time intervals from t_n to $t_{n+1} = t_n + h$.

The slopes of these lines are given by the differential equation since it tells us that $y'(t) = f(t, y(t))$.

Note, however, that at each point (t_n, y_n) the new "marching direction" for Euler's method is only close to the slope of the solution at t_n since in general f depends on t and y, and y (the unknown solution) is only approximately known.

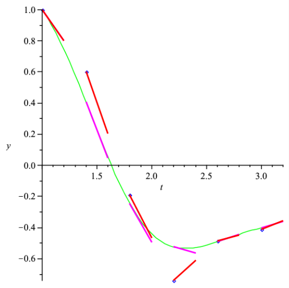


Graphical interpretation of Euler's method

Graphically, Euler's method comes down to taking straight line approximations of the unknown solution y over small time intervals from t_n to $t_{n+1} = t_n + h$.

The slopes of these lines are given by the differential equation since it tells us that $y'(t) = f(t, y(t))$.

Note, however, that at each point (t_n, y_n) the new "marching direction" for Euler's method is only close to the slope of the solution at t_n since in general f depends on t and y, and y (the unknown solution) is only approximately known.



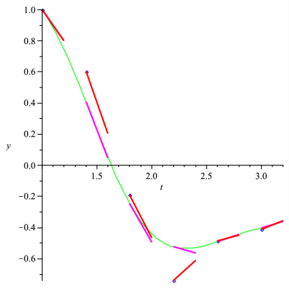
In fact, the slope we use as "marching direction" is that of the tangent line to a nearby solution (corresponding to a different initial condition).

Graphical interpretation of Euler's method

Graphically, Euler's method comes down to taking straight line approximations of the unknown solution y over small time intervals from t_n to $t_{n+1} = t_n + h$.

The slopes of these lines are given by the differential equation since it tells us that $y'(t) = f(t, y(t))$.

Note, however, that at each point (t_n, y_n) the new "marching direction" for Euler's method is only close to the slope of the solution at t_n since in general f depends on t and y, and y (the unknown solution) is only approximately known.



In fact, the slope we use as "marching direction" is that of the tangent line to a nearby solution (corresponding to a different initial condition).

See the Maple worksheet `EulerDemo.mw`.

Example

Apply the basic trapezoidal rule $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}[f(a)+f(b)]$ with $a = t$ and $b = t+h$ to the RHS of (3).

Example

Apply the basic trapezoidal rule $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\left[f(a)+f(b)\right]$ with $a=t$ and $b=t+h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \frac{h}{2}\left[f(t, y(t)) + f(t+h, y(t+h))\right].$$

Example

Apply the basic trapezoidal rule $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\left[f(a)+f(b)\right]$ with $a = t$ and $b = t + h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \frac{h}{2}\left[f(t, y(t)) + f(t+h, y(t+h))\right].$$

The corresponding IVP solver is therefore

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

Example

Apply the basic trapezoidal rule $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\left[f(a)+f(b)\right]$ with $a = t$ and $b = t + h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \frac{h}{2}\left[f(t, y(t)) + f(t+h, y(t+h))\right].$$

The corresponding IVP solver is therefore

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

Note that we have a y_{n+1} term on both sides of the equation (and cannot explicitly solve for it).

Example

Apply the basic trapezoidal rule $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\left[f(a)+f(b)\right]$ with $a = t$ and $b = t + h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \frac{h}{2}\left[f(t, y(t)) + f(t+h, y(t+h))\right].$$

The corresponding IVP solver is therefore

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

Note that we have a y_{n+1} term on both sides of the equation (and cannot explicitly solve for it). This means that we have an implicit method.

Example

Apply the basic trapezoidal rule $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2}\left[f(a)+f(b)\right]$ with $a = t$ and $b = t + h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx \frac{h}{2}\left[f(t, y(t)) + f(t+h, y(t+h))\right].$$

The corresponding IVP solver is therefore

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

Note that we have a y_{n+1} term on both sides of the equation (and cannot explicitly solve for it). This means that we have an implicit method. This method is also called trapezoidal rule (for IVPs).

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.
We can't simply time-advance the solution in a for-loop as we did for Euler's method.

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.

We can't simply time-advance the solution in a for-loop as we did for Euler's method.

Here are three commonly used approaches to deal with this difficulty:

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.
We can't simply time-advance the solution in a for-loop as we did for Euler's method.
Here are three commonly used approaches to deal with this difficulty:

- Modify the method to make it explicit.

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.
We can't simply time-advance the solution in a for-loop as we did for Euler's method.
Here are three commonly used approaches to deal with this difficulty:

- Modify the method to make it explicit.
- Couple it with an explicit method to create a so-called predictor-corrector method.

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.
We can't simply time-advance the solution in a for-loop as we did for Euler's method.
Here are three commonly used approaches to deal with this difficulty:

- Modify the method to make it explicit.
- Couple it with an explicit method to create a so-called predictor-corrector method.
- Use a nonlinear equation solver such as Newton's method at each time step.

How to deal with an implicit IVP solver?

The fact that in an implicit method the unknown value y_{n+1} appears on both sides of the equation causes serious problems.
We can't simply time-advance the solution in a for-loop as we did for Euler's method.
Here are three commonly used approaches to deal with this difficulty:

- Modify the method to make it explicit.
- Couple it with an explicit method to create a so-called predictor-corrector method.
- Use a nonlinear equation solver such as Newton's method at each time step.

Remark

We will not focus on implicit methods. They are more difficult to implement, but have better stability properties (see MATH 478). In MATLAB *we find them as stiff solvers.*

In order to make the trapezoidal rule

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

explicit we can use Euler's method to replace y_{n+1} on the right-hand side by

$$y_{n+1} = y_n + h f(t_n, y_n).$$

In order to make the trapezoidal rule

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

explicit we can use Euler's method to replace y_{n+1} on the right-hand side by

$$y_{n+1} = y_n + hf(t_n, y_n).$$

Then we end up with the method

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_n + hf(t_n, y_n))$$

In order to make the trapezoidal rule

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

explicit we can use Euler's method to replace y_{n+1} on the right-hand side by

$$y_{n+1} = y_n + hf(t_n, y_n).$$

Then we end up with the method

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_n + hf(t_n, y_n))$$

or

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1). \end{aligned}$$

In order to make the trapezoidal rule

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

explicit we can use Euler's method to replace y_{n+1} on the right-hand side by

$$y_{n+1} = y_n + hf(t_n, y_n).$$

Then we end up with the method

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_n + hf(t_n, y_n))$$

or

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1). \end{aligned}$$

This is known as the classical second-order Runge-Kutta method.

In order to make the trapezoidal rule

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_{n+1}).$$

explicit we can use Euler's method to replace y_{n+1} on the right-hand side by

$$y_{n+1} = y_n + hf(t_n, y_n).$$

Then we end up with the method

$$y_{n+1} = y_n + \frac{h}{2} f(t_n, y_n) + \frac{h}{2} f(t_{n+1}, y_n + hf(t_n, y_n))$$

or

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1). \end{aligned}$$

This is known as the classical second-order Runge-Kutta method.

For an example see the MATLAB files `RK2.m` and `RK2Demo.m`.

Graphical Interpretation of Classical 2nd-Order RK

Since the 2nd-order RK method is given by

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1). \end{aligned}$$

we see that it corresponds to

Graphical Interpretation of Classical 2nd-Order RK

Since the 2nd-order RK method is given by

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1). \end{aligned}$$

we see that it corresponds to

- taking a tentative Euler step with slope s_1 resulting in

$$\tilde{y}_{n+1} = y_n + hf(t_n, y_n) = y_n + hs_1$$

  so that we obtain an approximate slope s_2 at $t_n + h$.

Graphical Interpretation of Classical 2nd-Order RK

Since the 2nd-order RK method is given by

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1). \end{aligned}$$

we see that it corresponds to

- taking a tentative Euler step with slope s_1 resulting in

  $$\tilde{y}_{n+1} = y_n + hf(t_n, y_n) = y_n + hs_1$$

  so that we obtain an approximate slope s_2 at $t_n + h$.
- The actual Euler step then uses the average of the slopes s_1 at t_n and s_2 at $t_n + h$ to obtain

  $$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}.$$

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + h$ to the RHS of (3).

Example

Apply the basic midpoint rule $\int_a^b f(x)dx \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + h$ to the RHS of (3).

Solution

This gives us $\int_t^{t+h} f(\tau, y(\tau))d\tau \approx hf(t + \frac{h}{2}, y(t + \frac{h}{2}))$.

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + h$ to the RHS of (3).

Solution

This gives us $\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t + \frac{h}{2}, y(t + \frac{h}{2}))$.

Now the term $y(t + \frac{h}{2})$ on the right-hand side is unknown.

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + h$ to the RHS of (3).

Solution

This gives us $\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t + \frac{h}{2}, y(t + \frac{h}{2}))$.

Now the term $y(t + \frac{h}{2})$ on the right-hand side is unknown.

We can use Euler's method with step size $\frac{h}{2}$ to approximate this value:

$$y(t + \frac{h}{2}) \approx y(t) + \frac{h}{2} f(t, y(t)).$$

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + h$ to the RHS of (3).

Solution

This gives us $\displaystyle\int_t^{t+h} f(\tau, y(\tau))\mathrm{d}\tau \approx hf(t + \frac{h}{2}, y(t + \frac{h}{2}))$.

Now the term $y(t + \frac{h}{2})$ on the right-hand side is unknown.

We can use Euler's method with step size $\frac{h}{2}$ to approximate this value:

$$y(t + \frac{h}{2}) \approx y(t) + \frac{h}{2}f(t, y(t)).$$

Therefore we get

$$y_{n+1} = y_n + hf(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)).$$

This is known as the modified Euler method or midpoint rule (for IVPs).

If we write the modified Euler method (midpoint rule)

$$y_{n+1} = y_n + hf(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2} f(t_n, y_n))$$

as

$$y_{n+1} = y_n + hs_2$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2} s_1) \end{aligned}$$

then this is also a second-order Runge Kutta method.

If we write the modified Euler method (midpoint rule)

$$y_{n+1} = y_n + hf(t_n + \frac{h}{2}, y_n + \frac{h}{2} f(t_n, y_n))$$

as

$$y_{n+1} = y_n + hs_2$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} s_1) \end{aligned}$$

then this is also a second-order Runge Kutta method.

Remark

Runge-Kutta methods are characterized by having several stages ($s_1, s_2, \ldots$) for each time step.

Graphical Interpretation of Modified Euler

Based on the Runge-Kutta formulation of the modified Euler method/midpoint rule we can see that it corresponds to

- taking an Euler step with only half the step length, $\frac{h}{2}$, resulting in

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} f(t_n, y_n)$$

Graphical Interpretation of Modified Euler

Based on the Runge-Kutta formulation of the modified Euler method/midpoint rule we can see that it corresponds to

- taking an Euler step with only half the step length, $\frac{h}{2}$, resulting in

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} f(t_n, y_n)$$

- followed by an Euler step with full step length h using the slope at the half-way point $(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$ so that

$$y_{n+1} = y_n + hf(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}).$$

Here $t_{n+\frac{1}{2}}$ is used symbolically to denote the time $t_n + \frac{h}{2}$.

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + 2h$ to the RHS of (3).

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + 2h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+2h} f(\tau, y(\tau))\mathrm{d}\tau \quad \approx \quad 2hf(t+h, y(t+h)).$$

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + 2h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+2h} f(\tau, y(\tau))\mathrm{d}\tau \quad \approx \quad 2hf(t+h, y(t+h)).$$

Thus, we have the explicit midpoint rule

$$y_{n+2} = y_n + 2hf(t_{n+1}, y_{n+1}).$$

Example

Apply the basic midpoint rule $\int_a^b f(x)\mathrm{d}x \approx (b-a)f\left(\frac{a+b}{2}\right)$ with $a = t$, $b = t + 2h$ to the RHS of (3).

Solution

This gives us

$$\int_t^{t+2h} f(\tau, y(\tau))\mathrm{d}\tau \quad \approx \quad 2hf(t+h, y(t+h)).$$

Thus, we have the explicit midpoint rule

$$y_{n+2} = y_n + 2hf(t_{n+1}, y_{n+1}).$$

This is an explicit 2-step method (and *not* a Runge-Kutta method). In the context of PDEs this method appears as the leapfrog method.

Example

There are many more examples connecting numerical integration methods with a solver for first-order initial value problems:

Example

There are many more examples connecting numerical integration methods with a solver for first-order initial value problems:

- The right endpoint rule will give rise to the so-called backward Euler method

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

— an implicit method.

Example

There are many more examples connecting numerical integration methods with a solver for first-order initial value problems:

- The right endpoint rule will give rise to the so-called backward Euler method

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

  — an implicit method.
- Simpson's rule yields the classical fourth-order Runge-Kutta method (see below) in case there is no dependence of f on y.

Example

There are many more examples connecting numerical integration methods with a solver for first-order initial value problems:

- The right endpoint rule will give rise to the so-called backward Euler method

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

  — an implicit method.
- Simpson's rule yields the classical fourth-order Runge-Kutta method (see below) in case there is no dependence of f on y.
- Gauss quadrature leads to so-called Gauss-Runge-Kutta or Gauss-Legendre methods. One such method is the implicit midpoint rule

$$y_{n+1} = y_n + hf(t_n + \tfrac{h}{2}, \tfrac{1}{2}(y_n + y_{n+1})).$$

Outline

Methods for which the value y_{n+1} depends only on the previous time level t_n are called single step methods. All methods mentioned so far (except for the explicit midpoint rule) fall into this category.

Methods for which the value y_{n+1} depends only on the previous time level t_n are called single step methods. All methods mentioned so far (except for the explicit midpoint rule) fall into this category.

We now take a closer look at the family of Runge-Kutta methods named after the late 19th/early 20th century German mathematicians Carl Runge and Martin Kutta.

Second-order Runge-Kutta Methods

We already met the classical second-order Runge-Kutta (improved Euler) method

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1) \end{aligned}$$

Second-order Runge-Kutta Methods

We already met the classical second-order Runge-Kutta (improved Euler) method

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1) \end{aligned}$$

and the modified Euler method (midpoint rule)

$$y_{n+1} = y_n + hs_2$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1). \end{aligned}$$

Second-order Runge-Kutta Methods

We already met the classical second-order Runge-Kutta (improved Euler) method

$$y_{n+1} = y_n + h\frac{s_1 + s_2}{2}$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + h, y_n + hs_1) \end{aligned}$$

and the modified Euler method (midpoint rule)

$$y_{n+1} = y_n + hs_2$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1). \end{aligned}$$

What do they have in common?

Both methods use two intermediate stages s_1 and s_2 to advance the solution.

Both methods use two intermediate stages s_1 and s_2 to advance the solution.

The stages correspond to different estimates for the slope of the solution.

Both methods use two intermediate stages s_1 and s_2 to advance the solution.

The stages correspond to different estimates for the slope of the solution.

As mentioned earlier,

- In the classical RK2 (improved Euler) method we average the slopes s_1 at t_n and s_2 at $t_n + h$,

Both methods use two intermediate stages s_1 and s_2 to advance the solution.

The stages correspond to different estimates for the slope of the solution.

As mentioned earlier,

- In the classical RK2 (improved Euler) method we average the slopes s_1 at t_n and s_2 at $t_n + h$,
- while for the modified Euler method we use s_1 at t_n to take a half-step to $t_n + \frac{h}{2}$, compute s_2 and then take the full step.

Both methods use two intermediate stages s_1 and s_2 to advance the solution.

The stages correspond to different estimates for the slope of the solution.

As mentioned earlier,

- In the classical RK2 (improved Euler) method we average the slopes s_1 at t_n and s_2 at $t_n + h$,
- while for the modified Euler method we use s_1 at t_n to take a half-step to $t_n + \frac{h}{2}$, compute s_2 and then take the full step.

One can imagine many other possibilities to complete these two stages.

A general explicit two-stage Runge-Kutta method is of the form

$$y_{n+1} = y_n + h\left[\gamma_1 s_1 + \gamma_2 s_2\right]$$

where

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \alpha_2 h, y_n + h\beta_{21} s_1), \end{aligned}$$

with $\alpha_2 = \beta_{21}$ (which ensures that the method is consistent[1] or first-order).

[1] Consistency is necessary for convergence (more details in MATH 478)

A general explicit two-stage Runge-Kutta method is of the form

$$y_{n+1} = y_n + h\left[\gamma_1 s_1 + \gamma_2 s_2\right]$$

where

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \alpha_2 h, y_n + h\beta_{21} s_1), \end{aligned}$$

with $\alpha_2 = \beta_{21}$ (which ensures that the method is consistent[1] or first-order).

Clearly, this is a generalization of the classical Runge-Kutta method since the choice $\gamma_1 = \gamma_2 = \frac{1}{2}$ and $\alpha_2 = \beta_{21} = 1$ yields that case.

[1] Consistency is necessary for convergence (more details in MATH 478)

A general explicit two-stage Runge-Kutta method is of the form

$$y_{n+1} = y_n + h\left[\gamma_1 s_1 + \gamma_2 s_2\right]$$

where

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \alpha_2 h, y_n + h\beta_{21} s_1), \end{aligned}$$

with $\alpha_2 = \beta_{21}$ (which ensures that the method is consistent[1] or first-order).

Clearly, this is a generalization of the classical Runge-Kutta method since the choice $\gamma_1 = \gamma_2 = \frac{1}{2}$ and $\alpha_2 = \beta_{21} = 1$ yields that case.

Remark

The somewhat arbitrary notation comes from a more general discussion that includes implicit as well as higher-order methods.

[1] Consistency is necessary for convergence (more details in MATH 478)

Butcher Tableaux

It is customary to arrange the coefficients α_i, β_{ij}, and γ_i in a so-called Runge-Kutta or Butcher tableaux.

Butcher Tableaux

It is customary to arrange the coefficients α_i, β_{ij}, and γ_i in a so-called Runge-Kutta or Butcher tableaux.
An explicit two-stage RK method will always look like

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \alpha_2 & \beta_{21} & 0 \\ \hline & \gamma_1 & \gamma_2 \end{array}$$

with $\alpha_2 = \beta_{21}$.

Butcher Tableaux

It is customary to arrange the coefficients α_i, β_{ij}, and γ_i in a so-called Runge-Kutta or Butcher tableaux.
An explicit two-stage RK method will always look like

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \alpha_2 & \beta_{21} & 0 \\ \hline & \gamma_1 & \gamma_2 \end{array}$$

with $\alpha_2 = \beta_{21}$.
Using Taylor series expansions one can show (see MATH 478) that for the method to be second-order it needs to also satisfy

$$\gamma_1 + \gamma_2 = 1$$
$$\beta_{21}\gamma_2 = \frac{1}{2}$$

— a system of two nonlinear equations in three unknowns. It is not difficult to generate solutions of this system.

Remark

The choice $\gamma_1 = 1$, $\gamma_2 = 0$ leads to Euler's method. However, since now we can't have $\beta_{21}\gamma_2 = \frac{1}{2}$ Euler's method is only a first-order method.

Remark

The choice $\gamma_1 = 1$, $\gamma_2 = 0$ leads to Euler's method. However, since now we can't have $\beta_{21}\gamma_2 = \frac{1}{2}$ Euler's method is only a first-order method.

Example

The Butcher tableaux for the classical RK2 method is

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

Remark

The choice $\gamma_1 = 1$, $\gamma_2 = 0$ leads to Euler's method. However, since now we can't have $\beta_{21}\gamma_2 = \frac{1}{2}$ Euler's method is only a first-order method.

Example

The Butcher tableaux for the classical RK2 method is

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2}. \end{array}$$

Example

The Butcher tableaux for the modified Euler method is

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1. \end{array}$$

Example

Another interesting second-order Runge-Kutta method has the tableaux

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array}.$$

Example

Another interesting second-order Runge-Kutta method has the tableaux

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & \frac{3}{4} \end{array}.$$

We will see later how it can be embedded into a third-order method that uses the same slopes as the second-order method (plus one additional one) resulting in an adaptive method.

Fourth-order Runge-Kutta Methods

Probably the most famous Runge-Kutta method is the four-stage classical fourth-order method:

$$y_{n+1} = y_n + \frac{h}{6}\left[s_1 + 2s_2 + 2s_3 + s_4\right]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_2\right) \\ s_4 &= f\left(t_n + h, y_n + hs_3\right) \end{aligned}$$

Fourth-order Runge-Kutta Methods

Probably the most famous Runge-Kutta method is the four-stage classical fourth-order method:

$$y_{n+1} = y_n + \frac{h}{6}\left[s_1 + 2s_2 + 2s_3 + s_4\right]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_2\right) \\ s_4 &= f\left(t_n + h, y_n + hs_3\right) \end{aligned}$$

and Butcher tableaux

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}. \end{array}$$

Fourth-order Runge-Kutta Methods

Probably the most famous Runge-Kutta method is the four-stage classical fourth-order method:

$$y_{n+1} = y_n + \frac{h}{6}\left[s_1 + 2s_2 + 2s_3 + s_4\right]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_2\right) \\ s_4 &= f\left(t_n + h, y_n + hs_3\right) \end{aligned}$$

and Butcher tableaux

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}. \end{array}$$

For an example see the MATLAB files `RK4.m` and `RK4Demo.m`.

Convergence Experiments

In the MATLAB script `EulerRKConvergenceDemo.m` we compare the orders (convergence rates) of three single step methods:

- Euler's method (first-order),
- classical second-order Runge-Kutta (or improved Euler) method (second-order),
- classical fourth-order Runge-Kutta method (fourth-order).

Outline

In order to be able to implement an adaptive step size control as we did for numerical integration in `quadtx` we need to have two IVP solvers of different orders that can be paired together efficiently.

In order to be able to implement an adaptive step size control as we did for numerical integration in `quadtx` we need to have two IVP solvers of different orders that can be paired together efficiently. There are several ways to do this:

- using embedded Runge-Kutta methods (such as the MATLAB functions `ode23`, `ode45`, `ode23s`[2]),

[2] the `s` indicates a stiff solver

In order to be able to implement an adaptive step size control as we did for numerical integration in `quadtx` we need to have two IVP solvers of different orders that can be paired together efficiently. There are several ways to do this:

- using embedded Runge-Kutta methods (such as the MATLAB functions `ode23`, `ode45`, `ode23s`[2]),
- using adaptive order methods which are usually based on multistep methods (such as `ode113`, `ode15s`),

[2]the `s` indicates a stiff solver

In order to be able to implement an adaptive step size control as we did for numerical integration in `quadtx` we need to have two IVP solvers of different orders that can be paired together efficiently. There are several ways to do this:

- using embedded Runge-Kutta methods (such as the MATLAB functions `ode23`, `ode45`, `ode23s`[2]),
- using adaptive order methods which are usually based on multistep methods (such as `ode113`, `ode15s`),
- using so-called predictor-corrector methods (also based on multistep methods such as Adams methods),

[2]the `s` indicates a stiff solver

In order to be able to implement an adaptive step size control as we did for numerical integration in `quadtx` we need to have two IVP solvers of different orders that can be paired together efficiently. There are several ways to do this:

- using embedded Runge-Kutta methods (such as the MATLAB functions `ode23`, `ode45`, `ode23s`[2]),
- using adaptive order methods which are usually based on multistep methods (such as `ode113`, `ode15s`),
- using so-called predictor-corrector methods (also based on multistep methods such as Adams methods),
- using two totally different methods (such as `ode23t` and `ode23tb` which both couple the trapezoidal rule with something else).

[2]the `s` indicates a stiff solver

The second-order Runge-Kutta methods we discussed earlier require two evaluations of f, i.e., two stages, per time step. Similarly, the fourth-order method required four function evaluations.

The second-order Runge-Kutta methods we discussed earlier require two evaluations of f, i.e., two stages, per time step. Similarly, the fourth-order method required four function evaluations.

For more general Runge-Kutta methods the situation is as follows:

# of stages per time step	2	3	4	5	6	7	8	9	10	11
maximum order achievable	2	3	4	4	5	6	6	7	7	8

The second-order Runge-Kutta methods we discussed earlier require two evaluations of f, i.e., two stages, per time step. Similarly, the fourth-order method required four function evaluations.

For more general Runge-Kutta methods the situation is as follows:

# of stages per time step	2	3	4	5	6	7	8	9	10	11
maximum order achievable	2	3	4	4	5	6	6	7	7	8

This shows that higher-order (> 4) Runge-Kutta methods are relatively inefficient.

The second-order Runge-Kutta methods we discussed earlier require two evaluations of f, i.e., two stages, per time step. Similarly, the fourth-order method required four function evaluations.

For more general Runge-Kutta methods the situation is as follows:

# of stages per time step	2	3	4	5	6	7	8	9	10	11
maximum order achievable	2	3	4	4	5	6	6	7	7	8

This shows that higher-order (> 4) Runge-Kutta methods are relatively inefficient.

However, certain higher-order methods are still useful if we want to construct adaptive embedded Runge-Kutta methods.

Example

Earlier we mentioned the second-order Runge-Kutta method

$$y_{n+1} = y_n + \frac{h}{4}\left[s_1 + 3s_2\right]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}hs_1). \end{aligned}$$

Example

Earlier we mentioned the second-order Runge-Kutta method

$$y_{n+1} = y_n + \frac{h}{4}\left[s_1 + 3s_2\right]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}hs_1). \end{aligned}$$

It can be paired with a third-order method that looks like

$$y_{n+1} = y_n + \frac{h}{8}\left[2s_1 + 3s_2 + 3s_3\right]$$

Example

Earlier we mentioned the second-order Runge-Kutta method

$$y_{n+1} = y_n + \frac{h}{4}[s_1 + 3s_2]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}hs_1). \end{aligned}$$

It can be paired with a third-order method that looks like

$$y_{n+1} = y_n + \frac{h}{8}[2s_1 + 3s_2 + 3s_3]$$

with the same s_1 and s_2 and

$$s_3 = f(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}hs_2).$$

Example

Earlier we mentioned the second-order Runge-Kutta method

$$y_{n+1} = y_n + \frac{h}{4}\left[s_1 + 3s_2\right]$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}hs_1). \end{aligned}$$

It can be paired with a third-order method that looks like

$$y_{n+1} = y_n + \frac{h}{8}\left[2s_1 + 3s_2 + 3s_3\right]$$

with the same s_1 and s_2 and

$$s_3 = f(t_n + \tfrac{2}{3}h, y_n + \tfrac{2}{3}hs_2).$$

The combination uses only three function evaluations per time step.

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.

Other popular examples are:

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.

Other popular examples are:

- The MATLAB solver `ode23` by Bogacki and Shampine (which pairs a three-stage second-order method with a four-stage third-order method – see next section).

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.

Other popular examples are:

- The MATLAB solver `ode23` by Bogacki and Shampine (which pairs a three-stage second-order method with a four-stage third-order method – see next section).
- The classical fourth-fifth-order Runge-Kutta-Fehlberg method (which couples a five-stage fourth-order method with a six-stage fifth-order method). This is Maple's default numerical IVP solver `RKF45`.

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.

Other popular examples are:

- The MATLAB solver `ode23` by Bogacki and Shampine (which pairs a three-stage second-order method with a four-stage third-order method – see next section).
- The classical fourth-fifth-order Runge-Kutta-Fehlberg method (which couples a five-stage fourth-order method with a six-stage fifth-order method). This is Maple's default numerical IVP solver `RKF45`.
- The `ode45` code in MATLAB uses a different pair found by Dormand and Prince.

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.

We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.

Other popular examples are:

- The MATLAB solver `ode23` by Bogacki and Shampine (which pairs a three-stage second-order method with a four-stage third-order method – see next section).
- The classical fourth-fifth-order Runge-Kutta-Fehlberg method (which couples a five-stage fourth-order method with a six-stage fifth-order method). This is Maple's default numerical IVP solver `RKF45`.
- The `ode45` code in MATLAB uses a different pair found by Dormand and Prince.
- Mathematica has a fourth-fifth-order pair discovered by Bogacki and Shampine.

In general, for an embedded Runge-Kutta method we compute the value y_{n+1} with two different methods.
We need to pair up methods of different orders that use the same function evaluations, i.e., the function evaluations used for the lower-order method are embedded in the second higher-order method.
Other popular examples are:

- The MATLAB solver `ode23` by Bogacki and Shampine (which pairs a three-stage second-order method with a four-stage third-order method – see next section).
- The classical fourth-fifth-order Runge-Kutta-Fehlberg method (which couples a five-stage fourth-order method with a six-stage fifth-order method). This is Maple's default numerical IVP solver `RKF45`.
- The `ode45` code in MATLAB uses a different pair found by Dormand and Prince.
- Mathematica has a fourth-fifth-order pair discovered by Bogacki and Shampine.
- There also is a fifth-sixth-order pair by Dormand and Prince.

In [NCM] we can find a detailed discussion of the textbook version `ode23tx` of the Bogacki-Shampine BS23 method.

In [NCM] we can find a detailed discussion of the textbook version `ode23tx` of the Bogacki-Shampine BS23 method.

A seemingly inefficient three-stage second-order method is given by

$$y_{n+1} = y_n + \frac{h}{9}(2s_1 + 3s_2 + 4s_3)$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \tfrac{3}{4}h, y_n + \tfrac{3}{4}hs_2\right) \end{aligned}$$

In [NCM] we can find a detailed discussion of the textbook version `ode23tx` of the Bogacki-Shampine BS23 method.
A seemingly inefficient three-stage second-order method is given by

$$y_{n+1} = y_n + \frac{h}{9}(2s_1 + 3s_2 + 4s_3)$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \tfrac{3}{4}h, y_n + \tfrac{3}{4}hs_2\right) \end{aligned}$$

while the related four-stage third-order method is

$$\tilde{y}_{n+1} = y_n + \frac{h}{24}(7s_1 + 6s_2 + 8s_3 + 3s_4)$$

In [NCM] we can find a detailed discussion of the textbook version `ode23tx` of the Bogacki-Shampine BS23 method.
A seemingly inefficient three-stage second-order method is given by

$$y_{n+1} = y_n + \frac{h}{9}(2s_1 + 3s_2 + 4s_3)$$

with

$$\begin{aligned} s_1 &= f(t_n, y_n) \\ s_2 &= f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}s_1\right) \\ s_3 &= f\left(t_n + \tfrac{3}{4}h, y_n + \tfrac{3}{4}hs_2\right) \end{aligned}$$

while the related four-stage third-order method is

$$\tilde{y}_{n+1} = y_n + \frac{h}{24}(7s_1 + 6s_2 + 8s_3 + 3s_4)$$

with s_1, s_2 and s_3 as above, and

$$s_4 = f(t_{n+1}, y_{n+1})$$

using the second-order approximation y_{n+1}.

Adaptive step size control

Using a technique similar to Richardson extrapolation we can obtain an error estimate which we can use to adaptively control the stepsize:

$$e_{n+1} = \frac{h}{72}(-5s_1 + 6s_2 + 8s_3 - 9s_4).$$

Adaptive step size control

Using a technique similar to Richardson extrapolation we can obtain an error estimate which we can use to adaptively control the stepsize:

$$e_{n+1} = \frac{h}{72}(-5s_1 + 6s_2 + 8s_3 - 9s_4).$$

How is this error estimate used?

Adaptive step size control

Using a technique similar to Richardson extrapolation we can obtain an error estimate which we can use to adaptively control the stepsize:

$$e_{n+1} = \frac{h}{72}(-5s_1 + 6s_2 + 8s_3 - 9s_4).$$

How is this error estimate used?

- If the error estimate is less than a specified tolerance, then we accept the new value y_{n+1} given by the more conservative lower-order three-stage method.

Adaptive step size control

Using a technique similar to Richardson extrapolation we can obtain an error estimate which we can use to adaptively control the stepsize:

$$e_{n+1} = \frac{h}{72}(-5s_1 + 6s_2 + 8s_3 - 9s_4).$$

How is this error estimate used?

- If the error estimate is less than a specified tolerance, then we accept the new value y_{n+1} given by the more conservative lower-order three-stage method.
  Note that the fourth stage is not wasted since it is used as the first stage for the next time step. Thus, only three function evaluations are required per time step.

Adaptive step size control

Using a technique similar to Richardson extrapolation we can obtain an error estimate which we can use to adaptively control the stepsize:

$$e_{n+1} = \frac{h}{72}(-5s_1 + 6s_2 + 8s_3 - 9s_4).$$

How is this error estimate used?

- If the error estimate is less than a specified tolerance, then we accept the new value y_{n+1} given by the more conservative lower-order three-stage method.
  Note that the fourth stage is not wasted since it is used as the first stage for the next time step. Thus, only three function evaluations are required per time step.
- If the error is too large, then we forget the y_{n+1} calculation and try again with a smaller value of h.

Slightly simplified main ingredients of `ode23tx.m`

Function header:

```
function [tout,yout] = ode23tx(F,tspan,y0,arg4,varargin)
```

Slightly simplified main ingredients of ode23tx.m

Function header:

```
function [tout,yout] = ode23tx(F,tspan,y0,arg4,varargin)
```

Initialize a few variables:

```
rtol = 1.e-3,  atol = 1.e-6;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);      % "forward" or "backward"
threshold = atol / rtol;
hmax = abs(0.1*(tfinal-t0));
t = t0,  y = y0(:);
```

Slightly simplified main ingredients of `ode23tx.m`

Function header:

```
function [tout,yout] = ode23tx(F,tspan,y0,arg4,varargin)
```

Initialize a few variables:

```
rtol = 1.e-3,  atol = 1.e-6;
t0 = tspan(1);
tfinal = tspan(2);
tdir = sign(tfinal - t0);      % "forward" or "backward"
threshold = atol / rtol;
hmax = abs(0.1*(tfinal-t0));
t = t0,  y = y0(:);
```

Set initial time step size depending on scale of the problem:

```
s1 = F(t, y, varargin{:});
r = norm(s1./max(abs(y),threshold),inf) + realmin;
h = tdir*0.8*rtol^(1/3)/r;
```

The cube root appears because we have a third-order method.

Main loop

```
while t ~= tfinal
    hmin = 16*eps*abs(t);
    if abs(h) > hmax, h = tdir*hmax; end
    if abs(h) < hmin, h = tdir*hmin; end
    % Stretch the step if t is close to tfinal.
    if 1.1*abs(h) >= abs(tfinal - t)
        h = tfinal - t;
    end
```

Main loop

```
while t ~= tfinal
    hmin = 16*eps*abs(t);
    if abs(h) > hmax, h = tdir*hmax; end
    if abs(h) < hmin, h = tdir*hmin; end
    % Stretch the step if t is close to tfinal.
    if 1.1*abs(h) >= abs(tfinal - t)
        h = tfinal - t;
    end
```

The Runge-Kutta step

```
    s2 = F(t+h/2, y+h/2*s1, varargin{:});
    s3 = F(t+3*h/4, y+3*h/4*s2, varargin{:});
    tnew = t + h;
    ynew = y + h*(2*s1 + 3*s2 + 4*s3)/9;
    s4 = F(tnew, ynew, varargin{:});
```

Error estimate scaled to match the tolerances (realmin prevents err from being exactly zero):

```
e = h*(-5*s1 + 6*s2 + 8*s3 - 9*s4)/72;
err = norm(e./max(max(abs(y),abs(ynew)),threshold),
        ... inf) + realmin;
```

Error estimate scaled to match the tolerances (realmin prevents err from being exactly zero):

```
e = h*(-5*s1 + 6*s2 + 8*s3 - 9*s4)/72;
err = norm(e./max(max(abs(y),abs(ynew)),threshold),
        ... inf) + realmin;
```

See if we advance or repeat

```
if err <= rtol
    t = tnew;
    y = ynew;
    tout(end+1,1) = t;
    yout(end+1,:) = y.';
    s1 = s4; % Reuse final function value in new step
% else forget the latest calculation
end
```

Compute new step size:

```
    h = h*min(5,0.8*(rtol/err)^(1/3));
end   % of function ode23tx
```

Here

$$\texttt{rtol/err} \begin{cases} > 1 & \text{if advance} \\ < 1 & \text{otherwise,} \end{cases}$$

and the factors 0.8 and 5 prevent excessive changes in step size.

We illustrate the use of `ode23tx` in several examples.

We illustrate the use of `ode23tx` in several examples.

Example

The trivial initial value problem

$$\begin{aligned}\frac{dy}{dt} &= 0, \quad 0 \le t \le 10 \\ y(0) &= 1\end{aligned}$$

has solution $y(t) = 1$.

We illustrate the use of `ode23tx` in several examples.

Example

The trivial initial value problem

$$\begin{aligned} \frac{dy}{dt} &= 0, \quad 0 \le t \le 10 \\ y(0) &= 1 \end{aligned}$$

has solution $y(t) = 1$.
In MATLAB we can use `ode23tx` to solve this problem by running this MATLAB code:

```
f = @(t,y) 0;
ode23tx(f,[0 10],1);
```

Example

The harmonic oscillator (see Pendulum example)

$$\begin{aligned} \frac{d^2}{dt^2} y(t) &= -y(t), \quad 0 \le t \le 2\pi \\ y(0) &= 1 \quad y'(0) = 0 \end{aligned}$$

is solved in MATLAB by

Example

The harmonic oscillator (see ▸ Pendulum example)

$$\begin{aligned}\frac{\mathrm{d}^2}{\mathrm{d}t^2}y(t) &= -y(t), \quad 0 \le t \le 2\pi \\ y(0) &= 1 \quad y'(0) = 0\end{aligned}$$

is solved in MATLAB by first converting it to a system of first-order initial value problems:

Example

The harmonic oscillator (see Pendulum example)

$$\begin{aligned} \frac{d^2}{dt^2} y(t) &= -y(t), \quad 0 \leq t \leq 2\pi \\ y(0) &= 1 \quad y'(0) = 0 \end{aligned}$$

is solved in MATLAB by first converting it to a system of first-order initial value problems:

$$\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= -y_1(t) \end{aligned}$$

Example

The harmonic oscillator (see ‣ Pendulum example)

$$\begin{aligned}\frac{d^2}{dt^2}y(t) &= -y(t), \quad 0 \le t \le 2\pi \\ y(0) &= 1 \quad y'(0) = 0\end{aligned}$$

is solved in MATLAB by first converting it to a system of first-order initial value problems:

$$\begin{aligned}y_1'(t) &= y_2(t) \\ y_2'(t) &= -y_1(t)\end{aligned}$$

We can then use `ode23tx` and ‣ run this MATLAB code:

```
f = @(t,y) [y(2); -y(1)];
ode23tx(f,[0 2*pi],[1; 0]);
```

Example (cont.)

In order to get a phase plane plot, i.e., a plot of the y_2 component (velocity) vs. the y_1 component (position) parametrized by time, we use

Run this MATLAB code

```
f = @(t,y) [y(2); -y(1)];
[t,y] = ode23tx(f,[0 2*pi],[1; 0]);
plot(y(:,1),y(:,2),'-o')
axis([-1.2 1.2 -1.2 1.2])
axis square
```

Example (cont.)

In order to get a phase plane plot, i.e., a plot of the y_2 component (velocity) vs. the y_1 component (position) parametrized by time, we use

▸ Run this MATLAB code

```
f = @(t,y) [y(2); -y(1)];
[t,y] = ode23tx(f,[0 2*pi],[1; 0]);
plot(y(:,1),y(:,2),'-o')
axis([-1.2 1.2 -1.2 1.2])
axis square
```

A few other ways to achieve this (such as by defining your own plotting function) are described in [NCM].

Example

In the two-body problem we model the orbit of a small body (such as a spaceship) as it moves under the gravitational attraction of a much heavier body (a planet).
A model for the path of the small body (specified by the Cartesian coordinates of its position at time t relative to the large body) is

$$
\begin{aligned}
x''(t) &= -\frac{x(t)}{r(t)^3} \\
y''(t) &= -\frac{y(t)}{r(t)^3}
\end{aligned}
$$

where

$$r(t) = \sqrt{x(t)^2 + y(t)^2}.$$

Example (cont.)

Since this is a system of two second-order equations we rewrite them as (cf. ▸ earlier example)

Example (cont.)

Since this is a system of two second-order equations we rewrite them as (cf. earlier example)

$$\boldsymbol{y}(t) = \begin{bmatrix} x(t) \\ x'(t) \\ y(t) \\ y'(t) \end{bmatrix}$$

so that

Example (cont.)

Since this is a system of two second-order equations we rewrite them as (cf. earlier example)

$$\boldsymbol{y}(t) = \begin{bmatrix} x(t) \\ x'(t) \\ y(t) \\ y'(t) \end{bmatrix}$$

so that

$$\boldsymbol{y}'(t) = \begin{bmatrix} x'(t) \\ -x(t)/r(t)^3 \\ y'(t) \\ -y(t)/r(t)^3 \end{bmatrix}$$

Example (cont.)

Since this is a system of two second-order equations we rewrite them as (cf. earlier example)

$$\boldsymbol{y}(t) = \begin{bmatrix} x(t) \\ x'(t) \\ y(t) \\ y'(t) \end{bmatrix}$$

so that

$$\boldsymbol{y}'(t) = \begin{bmatrix} x'(t) \\ -x(t)/r(t)^3 \\ y'(t) \\ -y(t)/r(t)^3 \end{bmatrix}$$

This is illustrated in `twobody.m` and `TwobodyDemo.m`.

Example

If one lights a match, the ball of flame grows rapidly until it reaches a critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface.

Example

If one lights a match, the ball of flame grows rapidly until it reaches a critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface.
A mathematical model for this is given by the nonlinear first-order equation

$$\begin{aligned} y'(t) &= y^2(t) - y^3(t), \quad 0 \le t \le \frac{2}{\delta} \\ y(0) &= \delta, \end{aligned}$$

Example

If one lights a match, the ball of flame grows rapidly until it reaches a critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface.
A mathematical model for this is given by the nonlinear first-order equation

$$\begin{aligned} y'(t) &= y^2(t) - y^3(t), \quad 0 \le t \le \tfrac{2}{\delta} \\ y(0) &= \delta, \end{aligned}$$

- $y(t)$: radius of the ball of flame at time t,
- y^2: comes from surface area,
- y^3: comes from volume,
- δ: initial radius (assumed to be “small”, transition to critical size occurs at $\frac{1}{\delta}$).

Example (cont.)

The exact solution is given by

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}, \quad a = \frac{1}{\delta} - 1,$$

where W is the Lambert W function (the solution of the equation $W(z)e^{W(z)} = z$, see also the Maple worksheet `MatchDemo.mw`).

Example (cont.)

The exact solution is given by

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}, \quad a = \tfrac{1}{\delta} - 1,$$

where W is the Lambert W function (the solution of the equation $W(z)e^{W(z)} = z$, see also the Maple worksheet `MatchDemo.mw`).
The numerical solution is illustrated in the MATLAB script `MatchDemo.m` and in in the NCM file `flame.m`.

Example (cont.)

The exact solution is given by

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}, \quad a = \frac{1}{\delta} - 1,$$

where W is the Lambert W function (the solution of the equation $W(z)e^{W(z)} = z$, see also the Maple worksheet `MatchDemo.mw`).
The numerical solution is illustrated in the MATLAB script `MatchDemo.m` and in in the NCM file `flame.m`.
Note how it takes `ode23tx` longer and longer to obtain a solution for decreasing values of δ.

Example (cont.)

The exact solution is given by

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}, \quad a = \frac{1}{\delta} - 1,$$

where W is the Lambert W function (the solution of the equation $W(z)e^{W(z)} = z$, see also the Maple worksheet `MatchDemo.mw`).
The numerical solution is illustrated in the MATLAB script `MatchDemo.m` and in in the NCM file `flame.m`.
Note how it takes `ode23tx` longer and longer to obtain a solution for decreasing values of δ.
This problem is initially well-behaved, but becomes stiff as the solution approaches the steady state of 1. See an illustration of the stiffness in `MatchDemo.mw`.

Example (cont.)

The exact solution is given by

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}, \quad a = \frac{1}{\delta} - 1,$$

where W is the Lambert W function (the solution of the equation $W(z)e^{W(z)} = z$, see also the Maple worksheet `MatchDemo.mw`).

The numerical solution is illustrated in the MATLAB script `MatchDemo.m` and in in the NCM file `flame.m`.

Note how it takes `ode23tx` longer and longer to obtain a solution for decreasing values of δ.

This problem is initially well-behaved, but becomes stiff as the solution approaches the steady state of 1. See an illustration of the stiffness in `MatchDemo.mw`.

The stiff solver `ode23s` used to solve this problem much more efficiently is an embedded second-third order implicit Runge-Kutta (or Rosenbrock) method.

Using `odeset` to add special options to MATLAB's IVP solvers

Example

In the match problem we used the `odeset` function to reduce the default relative tolerance from 10^{-3} to 10^{-4} via

```
tol = 1e-4;
opts = odeset('RelTol',tol);
ode23tx(f,[t0 tmax],y0,opts);
```

Example

Let's consider again the skydive model of Chapters 1 and 4:

$$\frac{\mathrm{d}v}{\mathrm{d}t}(t) = \frac{F_g + F_d}{m} = g - \frac{c}{m}v^2(t), \quad v(0) = v_0 = 0.$$

Here we used the second model according to which the drag force due to air resistance is proportional to the square of the velocity.

Example

Let's consider again the skydive model of Chapters 1 and 4:

$$\frac{\mathrm{d}v}{\mathrm{d}t}(t) = \frac{F_g + F_d}{m} = g - \frac{c}{m}v^2(t), \quad v(0) = v_0 = 0.$$

Here we used the second model according to which the drag force due to air resistance is proportional to the square of the velocity.
In addition, let's assume that the gravitational "constant" g depends on the altitude x according to Newton's inverse square law of gravitational attraction

$$g(x) = g(0)\frac{R^2}{(R + x(t))^2}$$

with

- $R \approx 6.37 \times 10^6 (m)$: earth's radius,
- $g(0) = 9.81 (m/s^2)$: value of the gravitational constant at the earth's surface.

Example (cont.)

Combining the above information we get

$$\frac{\mathrm{d}v}{\mathrm{d}t}(t) = g(0)\frac{R^2}{(R + x(t))^2} - \frac{c}{m}v^2(t), \quad v(0) = v_0 = 0.$$

Example (cont.)

Combining the above information we get

$$\frac{\mathrm{d}v}{\mathrm{d}t}(t) = g(0)\frac{R^2}{(R + x(t))^2} - \frac{c}{m}v^2(t), \quad v(0) = v_0 = 0.$$

Since $v = -\frac{\mathrm{d}x}{\mathrm{d}t}$ (with "$-$" indicating downward motion) we actually end up either with

Example (cont.)

Combining the above information we get

$$\frac{\mathrm{d}v}{\mathrm{d}t}(t) = g(0)\frac{R^2}{(R+x(t))^2} - \frac{c}{m}v^2(t), \quad v(0) = v_0 = 0.$$

Since $v = -\frac{\mathrm{d}x}{\mathrm{d}t}$ (with "$-$" indicating downward motion) we actually end up either with a second-order IVP, or with the first-order system

Example (cont.)

Combining the above information we get

$$\frac{\mathrm{d}v}{\mathrm{d}t}(t) = g(0)\frac{R^2}{(R+x(t))^2} - \frac{c}{m}v^2(t), \quad v(0) = v_0 = 0.$$

Since $v = -\frac{\mathrm{d}x}{\mathrm{d}t}$ (with "$-$" indicating downward motion) we actually end up either with a second-order IVP, or with the first-order system

$$\begin{aligned}
\frac{\mathrm{d}x}{\mathrm{d}t}(t) &= -v(t) \\
\frac{\mathrm{d}v}{\mathrm{d}t}(t) &= g(0)\frac{R^2}{(R+x(t))^2} - \frac{c}{m}v^2(t) \\
x(0) &= x_0 \\
v(0) &= 0.
\end{aligned}$$

A standard jumping altitude is about $x_0 = 4000(m)$.

Example (cont.)

In Chapter 4 we said that in order to decide when the skydiver will hit the ground we

- first need to solve the coupled second-order IVP for the position (altitude).

Example (cont.)

In Chapter 4 we said that in order to decide when the skydiver will hit the ground we

- first need to solve the coupled second-order IVP for the position (altitude).
- Then we need to find the root of the position function using a root-finding algorithm.

Example (cont.)

In Chapter 4 we said that in order to decide when the skydiver will hit the ground we

- first need to solve the coupled second-order IVP for the position (altitude).
- Then we need to find the root of the position function using a root-finding algorithm.

This is quite complicated, and luckily MATLAB offers a simpler approach based on event handling.

Example (cont.)

In Chapter 4 we said that in order to decide when the skydiver will hit the ground we

- first need to solve the coupled second-order IVP for the position (altitude).
- Then we need to find the root of the position function using a root-finding algorithm.

This is quite complicated, and luckily MATLAB offers a simpler approach based on event handling.

The MATLAB programs `Skydive3Demo.m`, `Skydive3.m`, and `Skydive3Event.m` illustrate how this works.

Example (cont.)

Event handling for the skydive problem works as follows:
In the main program we call `ode23`

```
opts = odeset('events',@Skydive3Event);
[t,y,te,ye] = ode23(@Skydive3,[t0 tend],y0,opts,g,c,m,R);
```

where `t0,tend,y0,g,c,m,R` are specified values.

Example (cont.)

Event handling for the skydive problem works as follows:
In the main program we call `ode23`

```
opts = odeset('events',@Skydive3Event);
[t,y,te,ye] = ode23(@Skydive3,[t0 tend],y0,opts,g,c,m,R);
```

where `t0,tend,y0,g,c,m,R` are specified values.
The event handler `Skydive3Event.m` consists of

```
function [stopval,isterm,dir] = Skydive3Event(t,y,g,c,m,R)
stopval = y(1); % the value we want to make 0
isterm = 1; % stop when stopval is 0
dir = []; % direct. from which we approach 0 irrelevant
```

Example (cont.)

Event handling for the skydive problem works as follows:
In the main program we call `ode23`

```
opts = odeset('events',@Skydive3Event);
[t,y,te,ye] = ode23(@Skydive3,[t0 tend],y0,opts,g,c,m,R);
```

where `t0,tend,y0,g,c,m,R` are specified values.
The event handler `Skydive3Event.m` consists of

```
function [stopval,isterm,dir] = Skydive3Event(t,y,g,c,m,R)
stopval = y(1); % the value we want to make 0
isterm = 1; % stop when stopval is 0
dir = []; % direct. from which we approach 0 irrelevant
```

- `y(1)` contains the altitude, so the solver stops when this value becomes zero.
- `isterm` specifies whether we stop when the stop event occurs, or continue to `tend` (`isterm = 0`).

Example

As a small challenge you should try to modify the previous skydive example to cover the model of Problem 3 of Homework Assignment 1 and Problem 1 of Computer Assignment 1 in which the skydiver was allowed to use a parachute.

Outline

What is Stiffness?

In our earlier match lighting example we stated that the problem eventually becomes stiff. What did we mean by this?

What is Stiffness?

In our earlier match lighting example we stated that the problem eventually becomes stiff. What did we mean by this?

In [NCM] we read:

> *A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.*

What is Stiffness?

In our earlier match lighting example we stated that the problem eventually becomes stiff. What did we mean by this?
In [NCM] we read:

> *A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.*

Other similar descriptions from the literature are:

- A problem is stiff if it contains widely varying time scales, i.e., some components of the solution decay much more rapidly than others.

What is Stiffness?

In our earlier match lighting example we stated that the problem eventually becomes stiff. What did we mean by this?
In [NCM] we read:

> *A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.*

Other similar descriptions from the literature are:

- A problem is stiff if it contains widely varying time scales, i.e., some components of the solution decay much more rapidly than others.
- A problem is stiff if the stepsize is dictated by stability requirements rather than by accuracy requirements.

What is Stiffness?

In our earlier match lighting example we stated that the problem eventually becomes stiff. What did we mean by this?
In [NCM] we read:

> *A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.*

Other similar descriptions from the literature are:

- A problem is stiff if it contains widely varying time scales, i.e., some components of the solution decay much more rapidly than others.
- A problem is stiff if the stepsize is dictated by stability requirements rather than by accuracy requirements.
- A problem is stiff if explicit methods don't work, or work only extremely slowly.

Stiff ODEs arise in many applications; e.g.,

- when modeling chemical reactions,
- in control theory,
- in network analysis and simulation problems,
- in electrical circuits.

The van der Pol Equation

Example

The van der Pol equation is a generalization of the simple harmonic oscillator (obtained by setting $\mu = 0$ below). It models oscillations in which energy is fed into small oscillations and removed from large ones.

The van der Pol Equation

Example

The van der Pol equation is a generalization of the simple harmonic oscillator (obtained by setting $\mu = 0$ below). It models oscillations in which energy is fed into small oscillations and removed from large ones.

This results in a second-order nonlinear IVP

$$y''(t) - \mu \left(1 - y^2(t)\right) y'(t) + y(t) = 0,$$

$$y(0) = y_0, \; y'(0) = yp_0,$$

where μ is a parameter that indicates the amount of damping.

For positive values of μ the solution describes deterministic chaos and ends up in a limit cycle.

The van der Pol Equation

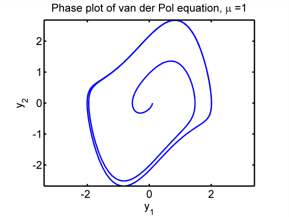
Example

The van der Pol equation is a generalization of the simple harmonic oscillator (obtained by setting $\mu = 0$ below). It models oscillations in which energy is fed into small oscillations and removed from large ones.

This results in a second-order nonlinear IVP

$$y''(t) - \mu\left(1 - y^2(t)\right)y'(t) + y(t) = 0,$$

$$y(0) = y_0,\ y'(0) = yp_0,$$



where μ is a parameter that indicates the amount of damping.

For positive values of μ the solution describes deterministic chaos and ends up in a limit cycle.

For large values of μ the equation becomes stiff.

Example (cont.)

Written as a first-order system we have

Example (cont.)

Written as a first-order system we have

$$
\begin{aligned}
y_1'(t) &= y_2(t) \\
y_2'(t) &= \mu\left(1 - y_1^2(t)\right) y_2(t) - y_1(t) \\
y_1(0) &= y_0 \\
y_2(0) &= yp_0.
\end{aligned}
$$

Example (cont.)

Written as a first-order system we have

$$\begin{aligned} y_1'(t) &= y_2(t) \\ y_2'(t) &= \mu \left(1 - y_1^2(t)\right) y_2(t) - y_1(t) \\ y_1(0) &= y_0 \\ y_2(0) &= yp_0. \end{aligned}$$

Solution of this system is illustrated in the MATLAB script `VanderPolDemo.m`.

Example (cont.)

Written as a first-order system we have

$$
\begin{aligned}
y_1'(t) &= y_2(t) \\
y_2'(t) &= \mu \left(1 - y_1^2(t)\right) y_2(t) - y_1(t) \\
y_1(0) &= y_0 \\
y_2(0) &= yp_0.
\end{aligned}
$$

Solution of this system is illustrated in the MATLAB script `VanderPolDemo.m`.
Additional plots are provided in `VanderPolPlots.m`.

Remark

While our explicit solvers `ode23` *and* `ode45` *use adaptive stepsizes, there are stability constraints we have not discussed which prevent them from taking very large time steps — even if the problem would seem to allow this.*

Remark

While our explicit solvers `ode23` *and* `ode45` *use adaptive stepsizes, there are stability constraints we have not discussed which prevent them from taking very large time steps — even if the problem would seem to allow this.*

This is why explicit solvers don't work for stiff problems.

Remark

While our explicit solvers `ode23` *and* `ode45` *use adaptive stepsizes, there are stability constraints we have not discussed which prevent them from taking very large time steps — even if the problem would seem to allow this.*

This is why explicit solvers don't work for stiff problems.

Implicit solvers have much better stability properties, and therefore adaptive implicit solvers (such as `ode23s` *and* `ode15s`*) can be used much more efficiently to deal with stiff problems.*

Outline

A single-step numerical method uses only the most recent history, y_n, to obtain the next value y_{n+1}.

A single-step numerical method uses only the most recent history, y_n, to obtain the next value y_{n+1}.

With an s-step method, on the other hand, more of the history of the solution will affect the next value, i.e., y_{n+s} depends on s previous values, $y_{n+s-1}, y_{n+s-2}, \ldots, y_{n+1}, y_n$.

A single-step numerical method uses only the most recent history, y_n, to obtain the next value y_{n+1}.

With an s-step method, on the other hand, more of the history of the solution will affect the next value, i.e., y_{n+s} depends on s previous values, $y_{n+s-1}, y_{n+s-2}, \ldots, y_{n+1}, y_n$.

In its most general form an s-step method looks like

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

where the coefficients a_m and b_m, $m = 0, 1, \ldots, s$, are independent of h, n, and the underlying ODE.

A single-step numerical method uses only the most recent history, y_n, to obtain the next value y_{n+1}.

With an s-step method, on the other hand, more of the history of the solution will affect the next value, i.e., y_{n+s} depends on s previous values, $y_{n+s-1}, y_{n+s-2}, \ldots, y_{n+1}, y_n$.

In its most general form an s-step method looks like

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

where the coefficients a_m and b_m, $m = 0, 1, \ldots, s$, are independent of h, n, and the underlying ODE.

Usually, the formula is normalized so that $a_s = 1$.

A single-step numerical method uses only the most recent history, y_n, to obtain the next value y_{n+1}.

With an s-step method, on the other hand, more of the history of the solution will affect the next value, i.e., y_{n+s} depends on s previous values, $y_{n+s-1}, y_{n+s-2}, \dots, y_{n+1}, y_n$.

In its most general form an s-step method looks like

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \dots,$$

where the coefficients a_m and b_m, $m = 0, 1, \dots, s$, are independent of h, n, and the underlying ODE.

Usually, the formula is normalized so that $a_s = 1$.

Different choices of the a_m and b_m yield different numerical methods.

A single-step numerical method uses only the most recent history, y_n, to obtain the next value y_{n+1}.

With an s-step method, on the other hand, more of the history of the solution will affect the next value, i.e., y_{n+s} depends on s previous values, $y_{n+s-1}, y_{n+s-2}, \ldots, y_{n+1}, y_n$.

In its most general form an s-step method looks like

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

where the coefficients a_m and b_m, $m = 0, 1, \ldots, s$, are independent of h, n, and the underlying ODE.

Usually, the formula is normalized so that $a_s = 1$.

Different choices of the a_m and b_m yield different numerical methods.

We have a true s-step formula if either a_0 or b_0 is different from zero.

General s-step method:

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

If $b_s = 0$ the method is explicit (otherwise implicit).

General s-step method:

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

If $b_s = 0$ the method is explicit (otherwise implicit).

Explicit s-step methods can be accurate at most of order s.

General s-step method:

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

If $b_s = 0$ the method is explicit (otherwise implicit).

Explicit s-step methods can be accurate at most of order s.

For implicit s-step methods this can increase to $s+1$ (if s odd) or $s+2$ (s even).

General s-step method:

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

If $b_s = 0$ the method is explicit (otherwise implicit).

Explicit s-step methods can be accurate at most of order s.

For implicit s-step methods this can increase to $s+1$ (if s odd) or $s+2$ (s even).

Adam-Bashforth methods are optimal order explicit methods.
Adam-Moulton methods with odd s are optimal order implicit methods.

General s-step method:

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m}), \qquad n = 0, 1, \ldots,$$

If $b_s = 0$ the method is explicit (otherwise implicit).

Explicit s-step methods can be accurate at most of order s.

For implicit s-step methods this can increase to $s + 1$ (if s odd) or $s + 2$ (s even).

Adam-Bashforth methods are optimal order explicit methods.
Adam-Moulton methods with odd s are optimal order implicit methods.

Multistep methods require additional startup values. These are frequently obtained using one step of a higher-order single-step method (such as a Runge-Kutta method).

Explicit methods

Example

- First-order Adams-Bashforth method (Euler):

$$y_{n+1} = y_n + hf(t_n, y_n)$$

- Second-order Adams-Bashforth method:

$$y_{n+2} = y_{n+1} + \frac{h}{2}\left[3f(t_{n+1}, y_{n+1}) - f(t_n, y_n)\right]$$

- Third-order Adams-Bashforth method:

$$y_{n+3} = y_{n+2} + \frac{h}{12}\left[23f(t_{n+2}, y_{n+2}) - 16f(t_{n+1}, y_{n+1}) + 5f(t_n, y_n)\right]$$

Implicit methods

Example

- First-order Adams-Moulton method (backward Euler):

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

- Second-order Adams-Moulton method (trapezoidal method, note only 1-step):

$$y_{n+2} = y_{n+1} + \frac{h}{2}\left[f(t_{n+2}, y_{n+2}) + f(t_{n+1}, y_{n+1})\right]$$

- Third-order Adams-Moulton method:

$$y_{n+3} = y_{n+2} + \frac{h}{12}\left[5f(t_{n+3}, y_{n+3}) + 8f(t_{n+2}, y_{n+2}) - f(t_{n+1}, y_{n+1})\right]$$

Usually one combines an implicit Adams-Moulton method with an explicit Adams-Bashforth method as a predictor-corrector pair. See, e.g., MATLAB's `ode113`, or

Usually one combines an implicit Adams-Moulton method with an explicit Adams-Bashforth method as a predictor-corrector pair. See, e.g., MATLAB's `ode113`, or

Example

- Predictor (AB2): $\tilde{y}_{n+2} = y_{n+1} + \frac{h}{2}\left[3f(t_{n+1}, y_{n+1}) - f(t_n, y_n)\right]$

Usually one combines an implicit Adams-Moulton method with an explicit Adams-Bashforth method as a predictor-corrector pair. See, e.g., MATLAB's `ode113`, or

Example

- Predictor (AB2): $\tilde{y}_{n+2} = y_{n+1} + \frac{h}{2}\left[3f(t_{n+1}, y_{n+1}) - f(t_n, y_n)\right]$
- Corrector (AM2): $y_{n+2} = y_{n+1} + \frac{h}{2}\left[f(t_{n+1}, y_{n+1}) + f(t_{n+2}, \tilde{y}_{n+2})\right]$

Usually one combines an implicit Adams-Moulton method with an explicit Adams-Bashforth method as a predictor-corrector pair. See, e.g., MATLAB's `ode113`, or

Example

- Predictor (AB2): $\tilde{y}_{n+2} = y_{n+1} + \frac{h}{2}\left[3f(t_{n+1}, y_{n+1}) - f(t_n, y_n)\right]$
- Corrector (AM2): $y_{n+2} = y_{n+1} + \frac{h}{2}\left[f(t_{n+1}, y_{n+1}) + f(t_{n+2}, \tilde{y}_{n+2})\right]$
- Error estimator for adaptive step size control: $\kappa = \frac{1}{6}|\tilde{y}_{n+2} - y_{n+2}|$

Usually one combines an implicit Adams-Moulton method with an explicit Adams-Bashforth method as a predictor-corrector pair. See, e.g., MATLAB's `ode113`, or

Example

- Predictor (AB2): $\tilde{y}_{n+2} = y_{n+1} + \frac{h}{2}\left[3f(t_{n+1}, y_{n+1}) - f(t_n, y_n)\right]$
- Corrector (AM2): $y_{n+2} = y_{n+1} + \frac{h}{2}\left[f(t_{n+1}, y_{n+1}) + f(t_{n+2}, \tilde{y}_{n+2})\right]$
- Error estimator for adaptive step size control: $\kappa = \frac{1}{6}|\tilde{y}_{n+2} - y_{n+2}|$

Remark

Multistep methods tend to be more efficient than single-step methods for problems with smooth solutions and high accuracy requirements. For example, the orbits of planets and deep space probes are computed with multistep methods.

BDF multistep methods are implemented in MATLAB as `ode15s`. Many more details on multistep methods are provided in MATH 478.

Outline

Things to remember about this chapter

- The derivative estimates are important in and of themselves, but also play a fundamental role in finite difference solvers for BVPs and PDEs (see below).
- The ODE solvers we looked at are limited to initital value problems.
- Always convert your problem to a (system of) first-order ODEs before applying one of the standard solvers.
- Choose your method according to the guidelines on the next slide (from MATLAB's Help documentation).

MATLAB ODE Solvers

- `ode45`: Nonstiff problems, medium accuracy. Use most of the time. This should be the first solver you try.
- `ode23`: Nonstiff problems, low accuracy. Use for large error tolerances or moderately stiff problems.
- `ode113`: Nonstiff problems, low to high accuracy. Use for stringent error tolerances or computationally intensive ordinary differential equation functions.
- `ode15s` Stiff problems, low to medium accuracy. Use if ode45 is slow (stiff systems) or there is a mass matrix.
- `ode23s` Stiff problems, low accuracy. Use for large error tolerances with stiff systems or with a constant mass matrix.
- `ode23t` Moderately stiff problems, low accuracy. Use for moderately stiff problems where you need a solution without numerical damping.
- `ode23tb` Stiff problems, low accuracy. Use for large error tolerances with stiff systems or if there is a mass matrix.

Other things we did not discuss

- Accuracy issues, such as local vs. global truncation errors. See MATH 478 for details (or Section 7.13 in [NCM]).

Other things we did not discuss

- Accuracy issues, such as local vs. global truncation errors. See MATH 478 for details (or Section 7.13 in [NCM]).
- Boundary value problems in one variable, such as

$$\begin{aligned} \frac{d^2}{dt^2} y(t) &= -y(t), \quad 0 \le t \le 2\pi \\ y(0) &= 1, \quad y(2\pi) = 0 \end{aligned}$$

MATLAB provides the finite difference solver `bvp4c` for such problems. Other popular methods are shooting methods, or collocation methods such as spectral methods. See MATH 478 for more on this.

Other things we did not discuss

- Accuracy issues, such as local vs. global truncation errors. See MATH 478 for details (or Section 7.13 in [NCM]).
- Boundary value problems in one variable, such as

$$\begin{aligned} \frac{d^2}{dt^2} y(t) &= -y(t), \quad 0 \le t \le 2\pi \\ y(0) &= 1, \quad y(2\pi) = 0 \end{aligned}$$

  MATLAB provides the finite difference solver `bvp4c` for such problems. Other popular methods are shooting methods, or collocation methods such as spectral methods. See MATH 478 for more on this.
- Partial differential equations, such as vibration of a string

$$\begin{aligned} \frac{\partial^2}{\partial t^2} u(x,t) &= c^2 \frac{\partial^2}{\partial x^2} u(x,t), \quad 0 \le x \le L,\ 0 \le t \le T \\ u(0,t) &= 0, \quad u(L,t) = 0, \quad 0 \le t \le T \\ u(x,0) &= f(x), \quad \frac{\partial}{\partial t} u(x,0) = g(x), \quad 0 \le x \le L. \end{aligned}$$

  A bit more is discussed in MATH 478 (and then MATH 589). See also Chapter 11 of [NCM].

References I

W. E. Boyce and R. C. DiPrima.
Elementary Differential Equations and Boundary Value Problems.
Wiley, 7th ed., 2001.

C. Moler.
Numerical Computing with MATLAB.
SIAM, Philadelphia, 2004.
Also http://www.mathworks.com/moler/.

D. G. Zill.
A First Course in Differential Equations with Modeling Applications.
Brooks/Cole, 8th ed., 2005.