Semester - III

		L	T	P	C
17MMU301	**NUMERICAL METHODS**	**4**	**0**	**0**	**4**

Scope: This course provides a deep knowledge to the learners to understand the basic concepts of Numerical Methods which utilize computers to solve Engineering Problems that are not easily solved or even impossible to solve by analytical means.

Objectives: To enable the students to study numerical techniques as powerful tool in scientific computing.

UNIT I

Convergence, Errors: Relative, Absolute, Round off, Truncation. Transcendental and Polynomial equations: Bisection method - Newton's method - False Position method - Secant method - Rate of convergence of these methods.

UNIT II

System of linear algebraic equations: Gaussian Elimination - Gauss Jordan methods - Gauss Jacobi method - Gauss Seidel method and their convergence analysis – LU decomposition - Power method.

UNIT III

Interpolation: Lagrange and Newton's methods. Error bounds - Finite difference operators. Gregory forward and backward difference interpolation – Newton's divided difference – Central difference – Lagrange and inverse Lagrange interpolation formula.

UNIT IV

Numerical Differentiation and Integration: Gregory's Newton's forward and backward differentiation- Trapezoidal rule, Simpson's rule, Simpsons 3/8th rule, Boole's Rule. Midpoint rule, Composite Trapezoidal rule, Composite Simpson's rule.

UNIT V

Ordinary Differential Equations: Taylor's series - Euler's method – modified Euler's method - Runge-Kutta methods of orders two and four.

SUGGESTED READINGS

TEXT BOOK

1. Jain. M.K., Iyengar. S.R.K.,and Jain R.K., (2012). Numerical Methods for Scientific and Engineering Computation, New Age International Publishers, New Delhi .

REFERNCES

1. Bradie B., (2007).A Friendly Introduction to Numerical Analysis, Pearson Education, India,

2.Gerald C.F., and Wheatley P.O., (2006). Applied Numerical Analysis, Sixth Edition, Dorling Kindersley (India) Pvt. Ltd., New Delhi.

3. Uri M. Ascher and Chen Greif., (2013). A First Course in Numerical Methods, Seventh Edition., PHI Learning Private Limited.

4. John H., Mathews and Kurtis D. Fink., (2012). Numerical Methods using Matlab, Fourth Edition., PHI Learning Private Limited.

5. Sastry S.S., (2008). Introductory methods of Numerical Analysis, Fourth edition, Prentice Hall of India, New Delhi.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021.

LECTURE PLAN
DEPARTMENT OF MATHEMATICS

STAFF NAME: M.SANTHI
SUBJECT NAME: NUMERICAL METHODS SUB.CODE:18MMU401
SEMESTER: IV CLASS: I B.SC MATHEMATICS

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
		UNIT-I	
1.	1	Introduction to Convergence	T1:ch-1, Pg.No:12-15
2.	1	Convergence, Errors: Relative, Absolute, Round off, Truncation	T1:ch -1,Pg.No:7-8
3.	1	Solution of Algebraic and Transcendental Equation -Bisection Method	T1: ch -2,Pg.No:20-22
4.	1	Newton's method and its rate of convergence- problems	R2:Ch 1,Pg.No:48-49
5.	1	Continuous on Newton's method and rate of convergence problems	R2:Ch 1,Pg.No:50-51
6.	1	False Position method and its rate of convergence related examples	T1: ch -2,Pg.No:23-24
7.	1	Continuous on False Position method and its rate of convergence related examples	T1: ch -2,Pg.No:25-26
8.	1	Secant method related problems and its rate of convergence	R5:ch-2,Pg.No:43-44
9.	1	Recapitulation and Discussion of possible questions	
	Total No of Hours Planned For Unit I=9		
		UNIT-II	
1.	1	Introduction to Solution of Simultaneous Linear algebraic Equations	T1:ch -3,Pg.No:114-115
2.	1	Gauss Elimination Method: Procedure	T1:ch -3,Pg.No:116-117
3.	1	Gauss Jordan Method and their convergence related examples	T1:ch -3,Pg.No:119-120

4.	1	Gauss Jordan Method and its convergence related examples	R1:chater-3,Pg.No:216-224
5.	1	Gauss Jacobic Method and its convergence related examples	T1:ch -3,Pg.No:146-149
6.	1	Gauss Seidal Method and its convergence problems	T1:ch -3,Pg.No:150-152
7.	1	Continuation of Problems on Gauss Seidal Method	R2:ch-2,Pg.No:129-134
8.	1	LU Decomposition related problems	R3: ch -5 Pg.No:100-105
9.	1	Power Method with examples	T1: ch -3,Pg.No:192-194
10.	1	Recapitulation and Discussion of possible questions	
	Total No of Hours Planned For Unit II=10		
		UNIT-III	
1.	1	Introduction on Interpolation and its formulas	T1: ch -4,Pg.No: 212-214
2.	1	Lagrange and Newton's Methods related problmes	T1: ch -4, Pg.No: 215-216
3.	1	Continuous on Lagrange and Newton's Methods related problmes	T1: ch -4, Pg.No: 216-217
4.	1	Error bounds - Finite difference operators related examples	T1: ch -4,Pg.No: 218-220
5.	1	Continuous on Error bounds - Finite difference operators related examples	T1: ch -4,Pg.No: 221-224
6.	1	Gregory Forward and backward difference Interpolation related examples	T1: ch -4, Pg.No: 230-236
7.	1	Newton's Divided difference and its problems	T1: ch -4,Pg.No: 226-229
8.	1	Central difference	R3:ch -10,Pg.No:306-310
9.	1	Lagrange and Inverse Interpolation formula	R4: ch -6,Pg.No:334-335
10.	1	Recapitulation and Discussion of possible questions	
	Total No of Hours Planned For Unit III=10		
		UNIT-IV	
1.	1	Introduction to Numerical Differentiation and Integration	T1: ch -5,Pg.No: 320-322
2.	1	Gregory 's Newton's Forward and Backward differentiation	T1: ch -5,Pg.No: 323-324
3.	1	Continuous on Gregory 's Newton's Forward and Backward differentiation	T1: ch -5, Pg.No: 325-326
4.	1	Trapezoidal rule and its examples	T1: ch -5,Pg.No:350-352
5.	1	Simpson's 1/3 rule and Simpson's	T1: ch -5,Pg.No:353-355

		3/8 rule-Problems	
6.	1	Boole's Rule & Midpoint rule related problems	R5:ch-5,Pg.No:200-202
7.	1	Composite Trapezoidal rule and its problems	T1: ch 5,Pg.No:386-387
8.	1	Composite Simpson' rule related examples	T1: ch 5,Pg.No:388-390
9.	1	Recapitulation and Discussion of possible questions	
	Total No of Hours Planned For Unit IV=9		
		UNIT-V	
1.	1	Introduction to Ordinary Differential Equations	R4:ch 9,Pg.No:451-453
2.		Taylor's series with examples	R4:ch 9,Pg.No:454-456
3.	1	Euler's method and modified Euler's method with problems	T1: ch -6, Pg.No:425-430
4.	1	Continuous on Euler's method and modified Euler's method with problems	R2:ch:6,Pg.No:455-458
5.	1	Runge-Kutta methods of orders two and four with problems	T1: ch -6, Pg.No:451-456
6.	1	Milne's predictor – corrector method & Adam's Bashforth predictor – corrector method and its examples	R2:ch:6,Pg.No:467-468 T1: ch -6,Pg.No:487-492
7.	1	Recapitulation and Discussion of possible questions	
8.	1	Discuss on Previous ESE Question Papers	
9.	1	Discuss on Previous ESE Question Papers	
10.	1	Discuss on Previous ESE Question Papers	
	Total No of Hours Planned for unit V=10		
Total Planned Hours	**48**		

SUGGESTED READINGS

TEXT BOOK

T1. Jain. M.K., Iyengar. S.R.K.,and Jain R.K., (2012). Numerical Methods for Scientific and Engineering Computation, New Age International Publishers, New Delhi .

REFERNCES

R1. Bradie B., (2007).A Friendly Introduction to Numerical Analysis, Pearson Education, India,

R2. Gerald C.F. and Wheatley P.O., (2006). Applied Numerical Analysis, Sixth Edition, Dorling Kindersley (India) Pvt. Ltd., New Delhi.

R3. Uri M. Ascher and Chen Greif., (2013). A First Course in Numerical Methods, Seventh Edition., PHI Learning Private Limited.

R4. John H., Mathews and Kurtis D. Fink., (2012). Numerical Methods using Matlab, Fourth Edition., PHI Learning Private Limited.

R5. Sastry S.S., (2008). Introductory methods of Numerical Analysis, Fourth edition, Prentice Hall of India, New Delhi.

Name and signature
of the student Representative

Name and Signature
Course Faculty

Name and Signature of the Class Tutor

Name and signature of Coordinator

Head of the Department

UNIT - I

Solution of Algebraic and Transcendental Equations

- Solution of Algebraic and Transcendental Equations
  - Bisection Method
  - Method of False Position
  - The Iteration Method
  - Newton Raphson Method
- Summary
- Solved University Questions (JNTU)
- Objective Type Questions

1.1 Solution of Algebraic and Transcendental Equations

1.1.1 Introduction

A polynomial equation of the form

$$f(x) = p_n(x) = a_0 x^{n-1} + a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1} x + a_n = 0 \qquad \ldots..(1)$$

is called an Algebraic equation. For example,

$x^4 - 4x^2 + 5 = 0$, $4x^2 - 5x + 7 = 0$; $2x^3 - 5x^2 + 7x + 5 = 0$ are algebraic equations.

An equation which contains polynomials, trigonometric functions, logarithmic functions, exponential functions etc., is called a Transcendental equation. For example,

$\tan x - e^x = 0$; $\sin x - xe^{2x} = 0$; $x\,e^x = \cos x$

are transcendental equations.

Finding the roots or zeros of an equation of the form $f(x) = 0$ is an important problem in science and engineering. We assume that $f(x)$ is continuous in the required interval. A root of an equation $f(x) = 0$ is the value of x, say $x = \alpha$ for which $f(\alpha) = 0$. Geometrically, a root of an equation $f(x) = 0$ is the value of x at which the graph of the equation $y = f(x)$ intersects the x – axis (see Fig. 1)

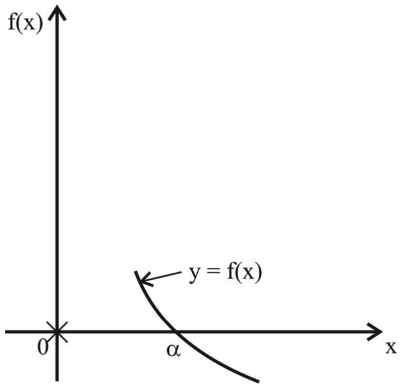


Fig. 1 Geometrical Interpretation of a root of $f(x) = 0$

A number α is a simple root of $f(x) = 0$; if $f(\alpha) = 0$ and $f'(\alpha) \neq 0$. Then, we can write $f(x)$ as,

$$f(x) = (x - \alpha)\, g(x),\ g(\alpha) \neq 0 \qquad \ldots..(2)$$

A number α is a multiple root of multiplicity m of $f(x) = 0$, if $f(\alpha) = f^1(\alpha) = \ldots. = f^{(m-1)}(\alpha) = 0$ and $f^m(\alpha) = 0$.

Then, $f(x)$ can be writhen as,

$$f(x) = (x - \alpha)^m\, g(x),\ g(\alpha) \neq 0 \qquad \ldots..(3)$$

A polynomial equation of degree n will have exactly n roots, real or complex, simple or multiple. A transcendental equation may have one root or no root or infinite number of roots depending on the form of $f(x)$.

The methods of finding the roots of $f(x) = 0$ are classified as,

1. Direct Methods
2. Numerical Methods.

Direct methods give the exact values of all the roots in a finite number of steps. Numerical methods are based on the idea of successive approximations. In these methods, we start with one or two initial approximations to the root and obtain a sequence of approximations x_0, x_1, … x_k which in the limit as $k \to \infty$ converge to the exact root $x = a$.

There are no direct methods for solving higher degree algebraic equations or transcendental equations. Such equations can be solved by Numerical methods. In these methods, we first find an interval in which the root lies. If a and b are two numbers such that $f(a)$ and $f(b)$ have opposite signs, then a root of $f(x) = 0$ lies in between a and b. We take a or b or any valve in between a or b as first approximation x_1. This is further improved by numerical methods. Here we discuss few important Numerical methods to find a root of $f(x) = 0$.

1.1.2 Bisection Method

This is a very simple method. Identify two points $x = a$ and $x = b$ such that $f(a)$ and $f(b)$ are having opposite signs. Let $f(a)$ be negative and $f(b)$ be positive. Then there will be a root of $f(x) = 0$ in between a and b.

Let the first approximation be the mid point of the interval (a, b). i.e.

$$x_1 = \frac{(a+b)}{2}$$

If $f(x_1) = 0$, then x_1 is a root, other wise root lies between a and x_1 or x_1 and b according as $f(x_1)$ is positive or negative. Then again we bisect the interval and continue the process until the root is found to desired accuracy. Let f (x_1) is positive, then root lies in between a and x_1 (see fig.2.). The second approximation to the root is given by,

$$x_2 = \frac{(a+x_1)}{2}$$

If $f(x_2)$ is negative, then next approximation is given by

$$x_3 = \frac{(x_2+x_1)}{2}$$

Similarly we can get other approximations. This method is also called Bolzano method.

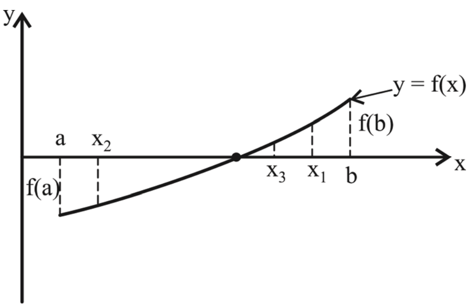


Fig. 2 Bisection Method

***Note*:** The interval width is reduced by a factor of one–half at each step and at the end of the n^{th} step, the new interval will be $[a_n, b_n]$ of length $\frac{|b-a|}{2^n}$. The number of iterations n required to achieve an accuracy $\in$ is given by,

$$n \geq \frac{\log_e\left(\frac{|b-a|}{\in}\right)}{\log_e 2} \qquad \text{.....(4)}$$

EXAMPLE 1

Find a real root of the equation $f(x) = x^3 - x - 1 = 0$, using Bisection method.

SOLUTION

First find the interval in which the root lies, by trail and error method.

$f(1) = 1^3 - 1 - 1 = -1$, which is negative

$f(2) = 2^3 - 2 - 1 = 5$, which is positive

$\therefore$ A root of $f(x) = x^3 - x - 1 = 0$ lies in between 1 and 2.

$$\therefore \quad x_1 = \frac{(1+2)}{2} = \frac{3}{2} = 1.5$$

$f(x_1) = f(1.5) = (1.5)^3 - 1.5 - 1 = 0.875$, which is positive.

Hence, the root lies in between 1 and 1.5

$$\therefore \quad x_2 = \frac{(1+1.5)}{2} = 1.25$$

$f(x_2) = f(1.25) = (1.25)^3 - 1.25 - 1 = -0.29$, which is negative.

Hence, the root lies in between 1.25 and 1.5

$\therefore \qquad x_3 = \dfrac{(1.25+1.5)}{2} = 1.375$

Similarly, we get $x_4 = 1.3125$, $x_5 = 1.34375$, $x_6 = 1.328125$ etc.

EXAMPLE 2

Find a root of $f(x) = xe^x - 1 = 0$, using Bisection method, correct to three decimal places.

SOLUTION

$$f(0) = 0.e^0 - 1 = -1 < 0$$

$$f(1) = 1.e^1 - 1 = 1.7183 > 0$$

Hence a root of $f(x) = 0$ lies in between 0 and 1.

$\therefore \qquad x_1 = \dfrac{(0+1)}{2} = 0.5$

$$f(0.5) = 0.5\, e^{0.5} - 1 = -0.1756$$

Hence the root lies in between 0.5 and 1

$\therefore \qquad x_2 = \dfrac{(0.5+1)}{2} = 0.75$

Proceeding like this, we get the sequence of approximations as follows.

$x_3 = 0.625$
$x_4 = 0.5625$
$x_5 = 0.59375$
$x_6 = 0.5781$
$x_7 = 0.5703$
$x_8 = 0.5664$
$x_9 = 0.5684$
$x_{10} = 0.5674$
$x_{11} = 0.5669$
$x_{12} = 0.5672$,
$x_{13} = 0.5671$,

Hence, the required root correct to three decimal places is, $x = 0.567$.

1.1.3 Method of False Position

This is another method to find the roots of $f(x) = 0$. This method is also known as Regular False Method.

In this method, we choose two points a and b such that $f(a)$ and $f(b)$ are of opposite signs. Hence a root lies in between these points. The equation of the chord joining the two points,

$(a, f(a))$ and $(b, f(b))$ is given by

$$\frac{y - f(a)}{x - a} = \frac{f(b) - f(a)}{b - a} \qquad .....(5)$$

We replace the part of the curve between the points $[a, f(a)]$ and $[b, f(b)]$ by means of the chord joining these points and we take the point of intersection of the chord with the x axis as an approximation to the root (see Fig.3). The point of intersection is obtained by putting y = 0 in (5), as

$$x = x_1 = \frac{a\,f(b) - b\,f(a)}{f(b) - f(a)} \qquad .....(6)$$

x_1 is the first approximation to the root of $f(x) = 0$.

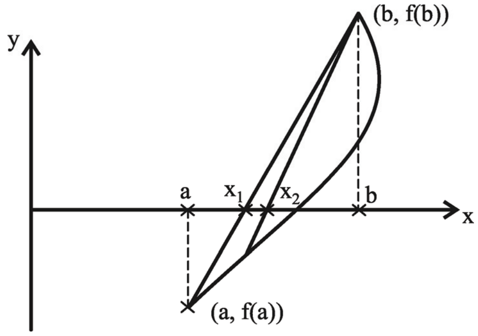


Fig. 3 Method of False Position

If f (x_1) and f (a) are of opposite signs, then the root lies between a and x_1 and we replace b by x_1 in (6) and obtain the next approximation x_2. Otherwise, we replace a by x_1 and generate the next approximation. The procedure is repeated till the root is obtained to the desired accuracy. This method is also called linear interpolation method or chord method.

EXAMPLE 3

Find a real root of the equation $f(x) = x^3 - 2x - 5 = 0$ by method of False position.

SOLUTION

$$f(2) = -1 \text{ and } f(3) = 16$$

Hence the root lies in between 2 and 3.

Take $a = 2$, $b = 3$.

$$\therefore \quad x_1 = \frac{a\,f(b) - b\,f(a)}{f(b) - f(a)}$$

$$= \frac{2(16) - 3(-1)}{16 - (-1)} = \frac{35}{17} = 2.058823529.$$

$$f(x_1) = f(2.058823529) = -0.390799917 < 0.$$

Therefore the root lies between 0.058823529 and 3. Again, using the formula, we get the second approximation as,

$$x_2 = \frac{2.058823529(16) - 3(-0.390799917)}{16 - (-0.390799917)} = 2.08126366$$

Proceeding like this, we get the next approximation as,

$$x_3 = 2.089639211,$$
$$x_4 = 2.092739575,$$
$$x_5 = 2.09388371,$$
$$x_6 = 2.094305452,$$
$$x_7 = 2.094460846$$

EXAMPLE 4

Determine the root of the equation $\cos x - x\,e^x = 0$ by the method of False position.

SOLUTION

$$f(0) = 1 \text{ and } f(1) = -2.177979523$$

$\therefore$ $a = 0$ and $b = 1$. The root lies in between 0 and 1

$$\therefore \quad x_1 = \frac{0(-2.177979523) - 1(1)}{-2.177979523 - 1} = 0.3146653378$$

$$f(x_1) = f(0.314653378) = 0.51986.$$

$\therefore$ The root lies in between 0.314653378 and 1.

Hence,
$$x_2 = \frac{0.3146653378(-2.177979523) - 1(0.51986)}{-2.177979523 - 0.51986} = 0.44673$$

Proceeding like this, we get

$$x_3 = 0.49402,$$
$$x_4 = 0.50995,$$
$$x_5 = 0.51520,$$
$$x_6 = 0.51692,$$

EXAMPLE 5

Determine the smallest positive root of $x - e^{-x} = 0$, correct of three significant figures using Regula False method.

SOLUTION

Here,
$$f(0) = 0 - e^{-0} = -1$$

and $f(1) = 1 - e^{-1} = 0.63212.$

$\therefore$ The smallest positive root lies in between 0 and 1. Here a = 0 and b = 1

$$\therefore \quad x_1 = \frac{0(0.63212) - 1(-1)}{0.63212 + 1} = 0.6127$$

$$f(0.6127) = 0.6127 - e^{-(0.6127)} = 0.0708$$

Hence, the next approximation lies in between 0 and 0.6127. Proceeding like this, we get

$x_2 = 0.57219,0$ $\quad x_3 = 0.5677,$ $\quad x_4 = 0.5672,$ $\quad x_5 = 0.5671,$

Hence, the smallest positive root, which is correct up to three decimal places is,

$x = 0.567$

1.1.4 The Iteration Method

In the previous methods, we have identified the interval in which the root of $f(x) = 0$ lies, we discuss the methods which require one or more starting values of x, which need not necessarily enclose the root of $f(x) = 0$. The iteration method is one such method, which requires one starting value of x.

We can use this method, if we can express $f(x) = 0$, as

$$x = \phi(x) \qquad \text{..... (1)}$$

We can express $f(x) = 0$, in the above form in more than one way also. For example, the equation $x^3 + x^2 - 1 = 0$ can be expressed in the following ways.

$$x = (1+x)^{\frac{-1}{2}}$$

$$x = (1-x^3)^{1/2}$$

$$x = (1-x^2)^{1/3}$$

and so on

Let x_0 be an approximation to the desired root ξ, which we can find graphically or otherwise. Substituting x_0 in right hand side of (1), we get the first approximation as

$$x_1 = \phi(x_0) \qquad \text{.....(2)}$$

The successive approximations are given by

$$x_2 = \phi(x_1)$$

$$x_3 = \phi(x_2) \qquad \text{.....(3)}$$

.

.

.

$$x_n = \phi(x_{n-1})$$

Note**:** The sequence of approximations $x_0, x_1, x_2 \ldots x_n$ given by (3) converges to the root ξ in a interval I, if $|\phi'(x)| < 1$ for all x in I.

EXAMPLE 6

Using the method of iteration find a positive root between 0 and 1 of the equation

$$x\,e^x = 1$$

SOLUTION

The given equation can be writhen as $x = e^{-x}$

$\therefore \quad \phi(x) = e^{-x}$.

Here $|\phi'(x)| < 1$ for $x < 1$

$\therefore$ We can use iterative method

Let $\quad x_0 = 1$

$$\therefore \quad x_1 = e^{-1} = \frac{1}{e} = 0.3678794.$$

$$x_2 = e^{-0.3678794} = 0.6922006.$$

$$x_3 = e^{-0.6922006} = 0.5004735$$

Proceeding like this, we get the required root as $x = 0.5671$.

EXAMPLE 7

Find the root of the equation $2x = \cos x + 3_1$ correct to three decimal places using Iteration method.

SOLUTION

Given equation can be written as

$$x = \frac{(\cos x + 3)}{2}$$

$$|\phi'(x)| = \left|\frac{\sin x}{2}\right| < 1$$

Hence iteration method can be applied

Let $x_0 = \frac{\pi}{2}$

$$\therefore \quad x_1 = \frac{1}{2}\left(\cos\frac{\pi}{2} + 3\right) = 1.5$$

$$x_2 = \frac{1}{2}(\cos 1.5 + 3) = 1.535$$

Similarly,

$x_3 = 1.518$,
$x_4 = 1.526$,
$x_5 = 1.522$,
$x_6 = 1.524$,
$x_7 = 1.523$,
$x_8 = 1.524$.

$\therefore$ The required root is $x = 1.524$

EXAMPLE 8

Find a real root of $2x - \log_{10} x = 7$ by the iteration method

SOLUTION

The given equation can be written as,

$$x = \frac{1}{2}(\log_{10} x + 7)$$

Let $x_0 = 3.8$

$$\therefore \quad x_1 = \frac{1}{2}(\log_{10} 3.8 + 7) = 3.79$$

$$x_2 = \frac{1}{2}(\log_{10} 3.79 + 7) = 3.7893$$

$$x_3 = \frac{1}{2}(\log_{10} 3.7893 + 7) = 3.7893.$$

$\therefore$ $x = 3.7893$ is a root of the given equation which is correct to four significant digits.

1.1.5 Newton Raphson Method

This is another important method. Let x_0 be approximation for the root of $f(x) = 0$. Let $x_1 = x_0 + h$ be the correct root so that $f(x_1) = 0$. Expanding $f(x_1) = f(x_0 + h)$ by Taylor series, we get

$$f(x_1) = f(x_1 + h) = f(x_0) + h\, f'(x_0) + \frac{h^2}{2!} f''(x_0) + \ldots\ldots = 0 \qquad \ldots..(1)$$

For small valves of h, neglecting the terms with h^2, h^3 ….. etc,. We get

$$\therefore \quad f(x_0) + h\, f'(x_0) = 0 \qquad \ldots..(2)$$

and $$h = -\frac{f(x_0)}{f^1(x_0)}$$

$\therefore$ $$x_1 = x_0 + h$$
$$= x_0 - \frac{f(x_0)}{f^{'}(x_0)}$$

Proceeding like this, successive approximation $x_2, x_3, \ldots x_{n+1}$ are given by,

$$x_{n+1} = x_n - \frac{f(x_n)}{f^{'}(x_n)}. \qquad \ldots..(3)$$

For n = 0, 1, 2, ……

***Note*:**

(i) The approximation x_{n+1} given by (3) converges, provided that the initial approximation x_0 is chosen sufficiently close to root of $f(x) = 0$.

(ii) Convergence of Newton-Raphson method: Newton-Raphson method is similar to iteration method

$$\phi(x) = x - \frac{f(x)}{f^{'}(x)} \qquad \ldots..(1)$$

differentiating (1) w.r.t to 'x' and using condition for convergence of iteration method i.e.

$$|\phi'(x)| < 1,$$

We get

$$\left|1 - \frac{f'(x).f'(x) - f(x)f''(x)}{[f'(x)]^2}\right| < 1$$

Simplifying we get condition for convergence of Newton-Raphson method is

$$|f(x).f''(x)| < [f(x)]^2$$

EXAMPLE 9

Find a root of the equation $x^2 - 2x - 5 = 0$ by Newton – Raphson method.

SOLUTION

Here $f(x) = x^3 - 2x - 5$.

$\therefore$ $$f^1(x) = 3x^2 - 2$$

Newton – Raphson method formula is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$\therefore \qquad x_{n+1} = x_n - \frac{x_n^3 - 2x_n - 5}{3x_n^2 - 2}, \qquad n = 0, 1, 2, \ldots \qquad \ldots\ldots(1)$$

Let $x_0 = 2$

$$\therefore \qquad f(x_0) = f(2) = 2^3 - 2\,(2) - 5 = -1$$

and $\qquad f^1(x_0) = f^1(2) = 3\,(2)^2 - 2 = 10$

Putting n = 0 in (I), we get

$$x_1 = 2 - \left(\frac{-1}{10}\right) = 2.1$$

$$f(x_1) = f(2.1) = (2.1)^3 - 2\,(2.1) - 5 = 0.061$$

$$f^1(x_1) = f^1(2.1) = 3\,(2.1)^2 - 2 = 11.23$$

$$\therefore \qquad x_2 = 2.1 - \frac{0.061}{11.23} = 2.094568$$

Similarly, we can calculate x_3, x_4 ……

EXAMPLE 10

Find a root of $x \sin x + \cos x = 0$, using Newton – Raphson method

---

SOLUTION

$$f(x) = x \sin x + \cos x.$$

$$\therefore \qquad f'(x) = \sin x + x \cos x - \sin x = x \cos x$$

The Newton – Raphson method formula is,

$$x_{n+1} = x_n - \frac{x_n \sin x_n + \cos x_n}{x_n \cos x_n}, \qquad n = 0, 1, 2, \ldots.$$

Let $\qquad x_0 = \pi = 3.1416.$

$$\therefore \qquad x_1 = 3.1416 - \frac{3.1416 \sin \pi + \cos \pi}{3.1416 \cos \pi} = 2.8233.$$

Similarly,

$x_2 = 2.7986$

$x_3 = 2.7984$

$x_4 = 2.7984$

$\therefore$ $x = 2.7984$ can be taken as a root of the equation $x \sin x + \cos x = 0$.

EXAMPLE 11

Find the smallest positive root of $x - e^{-x} = 0$, using Newton – Raphson method.

SOLUTION

Here $f(x) = x - e^{-x}$

$f^1(x) = 1 + e^{-x}$

$f(0) = -1$ and $f(1) = 0.63212$.

$\therefore$ The smallest positive root of $f(x) = 0$ lies in between 0 and 1.

Let $x_0 = 1$

The Newton – Raphson method formula is,

$$x_{n+1} = x_n - \frac{x_n - e^{-x}n}{1 + e^{-x}n}, \ n = 0, 1, 2, \ldots\ldots$$

$f(0) = f(1) = 0.63212$

$f'(0) = f'(1) = 1.3679$

$$\therefore \quad x_1 = x_0 - \frac{x_0 - e^{-x_0}}{1 + e^{-x_0}} = 1 - \frac{0.63212}{1.3679} = 0.5379.$$

$f(0.5379) = -0.0461$

$f'(0.5379) = 1.584.$

$$\therefore \quad x_2 = 0.5379 + \frac{0.0461}{1.584} = 0.567$$

Similarly, $x_3 = 0.56714$

$\therefore$ $x = 0.567$ can be taken as the smallest positive root of $x - e^{-x} = 0$., correct to three decimal places.

***Note*:** A method is said to be of order P or has the rate of convergence P, if P is the largest positive real number for which there exists a finite constant $c \neq 0$, such that

$$|\in_{K+1}| \leq c|\in_K|^P \qquad \ldots\ldots (A)$$

Where $\in_K = x_K - \xi$ is the error in the k^{th} iterate. C is called Asymptotic Error constant and depends on derivative of $f(x)$ at $x = \xi$. It can be shown easily that the order of convergence of Newton – Raphson method is 2.

Exercise - 1.1

1. Using Bisection method find the smallest positive root of $x^3 - x - 4 = 0$ which is correct to two decimal places.

   [**Ans:** 1.80]

2. Obtain a root correct to three decimal places of $x^3 - 18 = 0$, using Bisection Method.

   [**Ans:** 2.621]

3. Find a root of the equation $xe^x - 1 = 0$ which lies in (0, 1), using Bisection Method.

   [**Ans:** 0.567]

4. Using Method of False position, obtain a root of $x^3 + x^2 + x + 7 = 0$, correct to three decimal places.

   [**Ans:** – 2.105]

5. Find the root of $x^3 - 2x^2 + 3x - 5 = 0$, which lies between 1 and 2, using Regula False method.

   [**Ans:** 1.8438]

6. Compute the real root of $x \log x - 1.2 = 0$, by the Method of False position.

   [**Ans:** 2.740]

7. Find the root of the equation $\cos x - x\,e^x = 0$, correct to four decimal places by Method of False position

   [**Ans:** 0.5178]

8. Using Iteration Method find a real root of the equation $x^3 - x^2 - 1 = 0$.

   [**Ans:** 1.466]

9. Find a real root of $\sin^2 x = x^2 - 1$, using iteration Method.

   [**Ans:** 1.404]

10. Find a root of $\sin x = 10\,(x - 1)$, using Iteration Method.

    [**Ans:** 1.088]

11. Find a real root of $\cot x = e^x$, using Iteration Method.

    [**Ans:** 0.5314]

12. Find a root of $x^4 - x - 10 = 0$ by Newton – Raphson Method.

    [**Ans:** 1.856]

13. Find a real root of $x - \cos x = 0$ by Newton – Raphson Method.

[**Ans:** 0.739]

14. Find a root of $2x - 3 \sin x - 5 = 0$ by Newton – Raphson Method.

[**Ans:** 2.883238]

15. Find a smallest positive root of $\tan x = x$ by Newton – Raphson Method.

[**Ans:** 4.4934]

Summary

Solution of algebraic and transcendental equations

1. The numerical methods to find the roots of $f(x) = 0$

   (i) *Bisection method:* If a function $f(x)$ is continuous between a and b, $f(a)$ and $f(b)$ are of apposite sign then there exists at least one root between a and b. The approximate value of the root between them is $x_0 = \frac{a+b}{2}$

   If $f(x_0) = 0$ then the x_0 is the correct root of $f(x) = 0$. If $f(x_0) \neq 0$, then the root either lies in between $\left(a, \frac{a+b}{2}\right)$ or $\left(\frac{a+b}{2}, b\right)$ depending on whether $f(x_0)$ is negative or positive. Again bisection the interval and repeat same method until the accurate root is obtained.

   (ii) *Method of false position:* (Regula false method): This is another method to find the root of $f(x) = 0$. In this method, we choose two points a and b such that $f(a)$, $f(b)$ are of apposite signs. Hence the root lies in between these points $[a, f(a)]$, $[b, f(b)]$ using equation of the chord joining these points and taking the point of intersection of the chord with the x-axis as an approximate root (using $y = 0$ on x– axis) is $x_1 = \frac{a\,f(b) - b\,f(a)}{f(b) - f(a)}$

   Repeat the same process till the root is obtained to the desired accuracy.

   (iii) *Newton Raphson method:* The successive approximate roots are given by

   $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},\ n = 0, 1, 2 - - - - -$$

   provided that the initial approximate root x_0 is choosen sufficiently close to root of $f(x) = 0$

Solved University Questions

1. Find the root of the equation $2x - \log x = 7$ which lies between 3.5 and 4 by Regula–False method. (JNTU 2006)

Solution

Given $f(x) = 2x - \log x_{10} = 7$ …..(1)

Take $x_0 = 3.5$, $x_1 = 4$

Using Regula Falsi method

$$x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x)} . f(x_0)$$

$$x_2 = 3.5 - \frac{4 - 3.5}{(0.3979 + 0.5441)} (-0.5441)$$

$$x_2 = 3.7888$$

Now taking $x_0 = 3.7888$ and $x_1 = 4$

$$x_3 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} . f(x_0)$$

$$x_3 = 3.7888 - \frac{4 - 3.7888}{0.3988} (-0.0009)$$

$$x_3 = 3.7893$$

The required root is = 3.789

2. Find a real root of $xe^x = 3$ using Regula-Falsi method. (JNTU – 2006)

Solution

Given $f(x) = x\,e^x - 3 = 0$

$f(1) = e - 3 = -0.2817 < 0$

$f(2) = 2e^2 - 3 = 11.778 > 0$

$\therefore$ One root lies between 1 and 2

Now taking $x_0 = 1$, $x_1 = 2$

Using Regula – Falsi method

$$x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0)$$

$$\therefore \quad x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

$$x_2 = \frac{1(11.778) - 2(-0.2817)}{11.778 + 0.2817}$$

$$x_2 = 1.329$$

Now $f(x_2) = f(1.329) = 1.329\ e^{1.329} - 3 = 2.0199 > 0$

$f(1) = -0.2817 < 0$

$\therefore$ The root lies between 1 and 1.329 taking $x_0 = 1$ and $x_2 = 1.329$

$\therefore$ Taking $x_0 = 1$ and $x_2 = 1.329$

$$\therefore \qquad x_3 = \frac{x_0 f(x_2) - x_2 f(x_0)}{f(x_2) - f(x_0)}$$

$$= \frac{1(2.0199) + (1.329)(0.2817)}{(2.0199) + (0.2817)}$$

$$= \frac{2.3942}{2.3016} = 1.04$$

Now $f(x^3) = 1.04\ e^{1.04} - 3 = -0.05 < 0$

The root lies between x^2 and x^3

i.e., 1.04 and 1.329 $[\because f(x_2) > 0 \text{ and } f(x_3) < 0]$

$$\therefore \qquad x_4 = \frac{x_2 f(x_3) - x_3 f(x_2)}{f(x_3) - f(x_2)} = \frac{(1.04)(-0.05) - (1.329)(2.0199)}{(-0.05) - (2.0199)}$$

$x_4 = 1.08$ is the approximate root

3. Find a real root of $e^x \sin x = 1$ using Regula – Falsi method (JNTU 2006)

Solution

Given $f(x) = e^x \sin x - 1 = 0$

Consider $x_0 = 2$

$f(x_0) = f(2) = e^2 \sin 2 - 1 = -0.7421 < 0$

$f(x_1) = f(3) = e^3 \sin 3 - 1 = 0.511 > 0$

$\therefore$ The root lies between 2 and 3

Using Regula – Falsi method

$$x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

$$x_2 = \frac{2(0.511)+3(0.7421)}{0.511+0.7421}$$

$x_2 = 2.93557$

$f(x_2) = e^{2.93557}\sin(2.93557) - 1$

$f(x_2) = -0.35538 < 0$

$\therefore$ Root lies between x_2 and x_1

i.e., lies between 2.93557 and 3

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

$$= \frac{(2.93557)(0.511) - 3(-35538)}{0.511 + 0.35538}$$

$x_3 = 2.96199$

$f(x_3) = e^{2.90199}\sin(2.96199) - 1 = -0.000819 < 0$

$\therefore$ root lies between x_3 and x_1

$$x_4 = \frac{x_3 f(x_1) - x_1 f(x_3)}{f(x_1) - f(x_3)}$$

$$x_4 = \frac{2.96199(0.511) + 3(0.000819)}{0.511 + 0.000819} = 2.9625898$$

$f(x^4) = e^{2.9625898}\sin(2.9625898) - 1$

$f(x^4) = -0.0001898 < 0$

$\therefore$ The root lies between x_4 and x_1

$$x_5 = \frac{x_4 f(x_1) - x_1 f(x_4)}{f(x_1) - f(x_4)}$$

$$= \frac{2.9625898(0.511) + 3(0.0001898)}{0.511 + (0.0001898)}$$

$x_5 = 2.9626$

we have

$x_4 = 2.9625$

$x_5 = 2.9626$

$\therefore \quad x_5 = x_4 = 2.962$

$\therefore$ The root lies between 2 and 3 is 2.962

4. Find a real root of $x\,e^x = 2$ using Regula – Falsi method (JNTU 2007)

Solution

$f(x) = x\,e^x - 2 = 0$

$f(0) = -2 < 0$, $f(1) = $ i.e., $-2 = (2.7183)-2$

$f(1) = 0.7183 > 0$

$\therefore$ The root lies between 0 and 1

Considering $x_0 = 0$, $x_1 = 1$

$f(0) = f(x_0) = -2$; $f(1) = f(x_1) = 0.7183$

By Regula – Falsi method

$$x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

$$x_2 = \frac{0(0.7183) - 1(-2)}{0.7183 - (-2)} = \frac{2}{2.7183}$$

$$x_2 = 0.73575$$

Now $f(x^2) = f(0.73575) = 0.73575\,e^{0.73575} - 2$

$f(x_2) = -0.46445 < 0$

and $f(x_1) = 0.7183 > 0$

$\therefore$ The root x_3 lies between x_1 and x_2

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

$$x_3 = \frac{(0.73575)(0.7183)}{0.7183 + 0.46445}$$

$$x_3 = \frac{0.52848 + 0.46445}{1.18275}$$

$$x_3 = \frac{0.992939}{1.18275}$$

$$x_3 = 0.83951 \quad f(x^3) = \frac{(0.83951)}{(0.83951)e^{-2}}$$

$f(x_3) = (0.83951)\,e^{0.83951} - 2$

$f(x_3) = -0.056339 < 0$

∴ One root lies between x_1 and x_3

$$x_4 = \frac{x_3 f(x_1) - x_1 f(x_3)}{f(x_1) - f(x_3)} = \frac{(0.83951)(0.7183) - 1(-0.056339)}{0.7183 + 0.056339}$$

$$x_4 = \frac{0.65935}{0.774639} = 0.851171$$

$f(x_4) = 0.851171 \text{ e}0.851171 - 2 = -0.006227 < 0$

Now x_5 lies between x_1 and x_4

$$x_5 = \frac{x_4 f(x_1) - x_1 f(x_4)}{f(x_1) - f(x_4)}$$

$$x_5 = \frac{(0.851171)(0.7183) + (.006227)}{0.7183 + 0.006227}$$

$$x_5 = \frac{0.617623}{0.724527} = 0.85245$$

Now $f(x_5) = 0.85245 \text{ e}^{0.85245} \text{ e}^{0.85245} - 2 = -0.0006756 < 0$

∴ One root lies between x_1 and x_5, (i.e., x_6 lies between x_1 and x_5)

Using Regula – Falsi method

$$x_6 = \frac{(0.85245)(0.7183) + 0.0006756}{0.7183 + 0.0006756}$$

$x_6 = 0.85260$

Now $f(x_6) = -0.00006736 < 0$

∴ One root x_7 lies between x_1 and x_6

By Regula – Falsi method

$$x_7 = \frac{x_6 f(x_1) - x_1 f(x_6)}{f(x_1) - f(x_6)}$$

$$x_7 = \frac{(0.85260)(0.7183) + 0.0006736}{0.7183 + 0.0006736}$$

$x_7 = 0.85260$

From $x^6 = 0.85260$ and $x_7 = 0.85260$

∴ A real root of the given equation is 0.85260

5. Using Newton-Raphson method (a) Find square root of a number (b) Find a reciprocal of a number [JNTU 2008]

Solution

(a) Let n be the number

and $x = \sqrt{n} \Rightarrow x^2 = n$

If $f(x) = x^2 - n = 0$ …..(1)

Then the solution to $f(x) = x^2 - n = 0$ is $x = \sqrt{n}$.

$f^1(x) = 2x$

by Newton Raphson method

$$x_{i+1} = x_i - \frac{f(x_i)}{f^1(x_i)} = x_i - \left(\frac{x_i^2 - n}{2x_i}\right)$$

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{x}{x_i}\right) \quad \text{…..(2)}$$

using the above formula the square root of any number 'n' can be found to required accuracy

(b) To find the reciprocal of a number 'n'

$$f(x) = \frac{1}{x} - n = 0 \quad \text{…..(1)}$$

$\therefore$ solution of (1) is $x = \dfrac{1}{n}$

$$f^1(x) = -\frac{1}{x^2}$$

Now by Newton-Raphson method, $x_{i+1} = x_i - \left(\dfrac{f(x_i)}{f^1(x_i)}\right)$

$$x_{i+1} = x_i - \left(\frac{\frac{1}{x_i} - N}{-\frac{1}{x_1^2}}\right)$$

$$x_{i+1} = x_i\,(2 - x_i\, n)$$

using the above formula the reciprocal of a number can be found to required accuracy.

6. Find the reciprocal of 18 using Newton–Raphson method [JNTU 2004]

Solution

The Newton-Raphson method

$$x_{i+1} = x_i\,(2 - x_i\, n) \qquad \text{.....(1)}$$

considering the initial approximate value of x as $x_0 = 0.055$ and given $n = 18$

$\therefore \quad x_1 = 0.055\,[2 - (0.055)\,(18)]$

$\therefore \quad x_1 = 0.0555$

$x_2 = 0.0555\,[2 - 0.0555 \times 18]$

$x_2 = (0.0555)\,(1.001)$

$x_2 = 0.0555$

Hence $x_1 = x_2 = 0.0555$

$\therefore$ The reciprocal of 18 is 0.0555

7. Find a real root for $x \tan x + 1 = 0$ using Newton–Raphson method [JNTU 2006]

Solution

Given $f(x) = x \tan x + 1 = 0$

$f^1(x) = x \sec^2 x + \tan x$

$f(2) = 2 \tan 2 + 1 = -3.370079 < 0$

$f(3) = 2 \tan 3 + 1 = -0.572370 > 0$

$\therefore$ The root lies between 2 and 3

Take $\quad x_0 = \dfrac{2+3}{2} = 2.5 \qquad$ (average of 2 and 3)

By Newton-Raphson method

$$x_{i+1} = x_i - \left(\frac{f(x_i)}{f^1(x_i)}\right)$$

$$x_1 = x_0 - \left(\frac{f(x_0)}{f^1(x_0)}\right)$$

$$x_1 = 2.5 - \frac{(-0.86755)}{3.14808}$$

$$x_1 = 2.77558$$

$$x_2 = x_1 - \frac{f(x_i)}{f^1(x_i)} \;;$$

$f(x_1) = -0.06383, \qquad f^1(x_1) = 2.80004$

$$x_2 = 2.77558 - \frac{(-0.06383)}{2.80004}$$

$$x_2 = 2.798$$

$f(x_2) = -0.001080, \qquad f^1(x_2) = 2.7983$

$$x_3 = x_2 - \frac{f(x_2)}{f^1(x_2)} = 2.798 - \frac{[-0.001080]}{2.7983}$$

$$x_3 = 2.798.$$

$\therefore \qquad x_2 = x_3$

$\therefore$ The real root of $x \tan x + 1 = 0$ is 2.798

8. Find a root of $e^x \sin x = 1$ using Newton–Raphson method [JNTU 2006]

Solution

Given $f(x) = e^x \sin x - 1 = 0$

$$f^1(x) = e^x \sec x + ex \cos x$$

Take $\qquad x_1 = 0, x_2 = 1$

$$f(0) = f(x_1) = e^0 \sin 0 - 1 = -1 < 0$$

$$f(1) = f(x_2) = e^1 \sin(1) - 1 = 1.287 > 0$$

The root of the equation lies between 0 and 1

Using Newton-Raphson method

$$x_{i+1} = x_i - \frac{f(x_i)}{f^1(x_i)}$$

Now consider x_0 = average of 0 and 1

$$x_0 = \frac{1+0}{2} = 0.5$$

$$x_0 = 0.5$$

$$f(x_0) = e^{0.5} \sin(0.5) - 1$$

$$f^1(x_0) = e^{0.5} \sin(0.5) + e^{0.5} \cos(0.5) = 2.2373$$

$$x_1 = x_0 - \frac{f(x_0)}{f^1(x_0)} = 0.5 - \frac{(-0.20956)}{2.2373}$$

$x_1 = 0.5936$

$f(x_1) = e^{0.5936} \sin(0.5936) - 1 = 0.0128$

$f^1(x_1) = e^{0.5936} \sin(0.5936) + e^{0.5936} \cos(0.5936) = 2.5136$

$$x_2 = x_1 - \frac{f(x_1)}{f^1(x_1)} = 0.5936 - \frac{(0.0128)}{2.5136}$$

$\therefore \quad x_2 = 0.58854$

similarly $\quad x_3 = x_2 - \dfrac{f(x_1)}{f^1(x_1)}$

$f(x_2) = e^{0.58854} \sin(0.58854) - 1 = 0.0000181$

$f^1(x_2) = e^{0.58854} \sin(0.58854) + e^{0.58854} \cos(0.58854)$

$f(x_2) = 2.4983$

$$\therefore \quad x_3 = 0.58854 - \frac{0.0000181}{2.4983}$$

$x_3 = 0.5885$

$\therefore \quad x_2 - x_3 = 0.5885$

0.5885 is the root of the equation $e^x \sin x - 1 = 0$

9. Find a real root of the equation $xe^x - \cos x = 0$ using Newton-Raphson method [JNTU-2006]

Solution

Given $f(x) = \mathrm{e}^x - \cos x = 0$

$f^1(x) = x\mathrm{e}^x + \mathrm{e}^x + \sin x = (x + 1)\, e^x + \sin x$

Take $\quad f(0) = 0 - \cos 0 = -1 < 0$

$f(1) = e - \cos 1 = 2.1779 > 0$

$\therefore$ The root lies between 0 and 1

Let $\quad x_0 = \dfrac{0+1}{2} = 0.5 \quad$ (average of 0 and 1)

Newton-Raphson method

$$x_{i+1} = x_i - \frac{f(x_i)}{f^1(x_i)}$$

$$x_{i+1} = x_0 - \frac{f(x_0)}{f^1(x_0)} = 0.5 - \frac{(-0.053221)}{(1.715966)}$$

$x_1 = 0.5310$

$f(x_1) = 0.040734, \qquad f^1(x_1) = 3.110063$

$$x_2 = x_1 - \frac{f(x_1)}{f^1(x_1)} = 0.5310 - \frac{0.040734}{3.110064}$$

$\therefore \quad x_2 = 0.5179 ; \quad f(x_2) = 0.0004339, \qquad f^1(x_2) = 3.0428504$

$$x_3 = 0.5179 - \frac{(0.0004339)}{3.0428504}$$

$x_3 = 0.5177$

$\therefore \quad f(x_3) = 0.000001106$

$f(x_3) = 3.04214$

$$x_4 = x_3 - \frac{f(x_3)}{f(x_3)} = 0.5177 - \frac{0.000001106}{3.04212}$$

$x_4 = 0.5177$

$\therefore \quad x_3 = x_4 = 0.5177$

$\therefore$ The root of $xe^x - \cos x = 0$ is 0.5177

10. Find a root of the equation $x^4 - x - 10 = 0$ using Bisection method correct to 2 decimal places. [JNTU 2008]

Solution

Let $f(x) = x^4 - x - 10 = 0$ be the given equation. We observe that f(1) < 0, then f(2) >0. So one root lies between 1 and 2.

$\therefore$ Let $x_0 = 1$, $x_1 = 2$;

Take $x_2 = \dfrac{x_0 + x_1}{2} = 1.5$; $\qquad f(1.5) < 0$;

$\therefore$ The root lies between 1.5 and 2

Let us take $x_3 = \dfrac{1.5 + 2}{2} = 1.75$; we find that $f(1.75) < 0$,

$\therefore$ The root lies between 1.75 and 2

So we take now $x_4 = \dfrac{1.75 + 1.875}{2} = 1.8125 = 1.81$ can be taken as the root of the given equation.

11. Find a real root of equation $x^3 - x - 11 = 0$ by Bisection method. [JNTU-2007]

Solution

Given equation is $f(x) = x^3 - x - 11 = 0$

We observe that $f(2) = -5 < 0$ and $f(3) = 13 > 0$.

$\therefore$ A root of (1) lies between 2 and 3; take $x_0 = 2$, $x = 3$;

Let $x_2 = \dfrac{x_0 + x_1}{2} = \dfrac{2+3}{2} = 2.5$; Since $f(2.5) > 0$, the root lies between 2 and 2.5

$\therefore$ Taking $x_3 = \dfrac{2+2.5}{2} = 2.25$, we note that $f(2.25) < 0$;

$\therefore$ The root can be taken as lying between 2.25 and 2.5.

$\therefore$ The root = $\dfrac{2.25+2.5}{2} = 2.375$

12. Find a real root of $x^3 - 5x + 3 = 0$ using Bisection method. [JNTU-2007]

Solution

Let $f(x) = x^3 - 5x + 3 = 0$ be the equation given

Since $f(1) = -1 < 0$ and $f(2) = 1 > 0$, a real root lies between 1 and 2.

i.e., $x_0 = 1$, $x_1 = 2$; take $x_2 = \dfrac{1+2}{2} = 1.5$; $f(1.5) = -1.25 < 0$

$\therefore$ The root lies between 1.5 and 2;

$\therefore$ Take $x_3 = \dfrac{1.5+2}{2} = 1.75$

Now $f(1.75) = \left(\dfrac{7}{4}\right)^3 - 5\left(\dfrac{7}{4}\right) + 3 = -\text{ve}$;

$\therefore$ The root lies between 1.75 and 2

Let $x_4 = \dfrac{1.75+2}{2} = 1.875$;

We find that $f(1.875) = (1.875)^3 - 5(1.875) + 3 > 0$

$\therefore$ The root of the given equation lies between 1.75 and 1.875

$\therefore$ The root = $\dfrac{1.75+1.875}{2} = 1.813$

13. Find a real root of the equation $x^3 - 6x - 4 = 0$ by Bisection method [JNTU-2006]

Solution

Here $f(x) = x^3 - 6x - 4$

Take $x_0 = 2, x_1 = 3$; $(\because f(2) < 0, f(3) > 0)$

$x_1 = 2.5; f(x_1) < 0$; take $x_3 = \dfrac{2.5+3}{2} = 2.75$

$f(2.75) > 0 \Rightarrow x_4 = \dfrac{2.5+2.75}{2} = 2.625$

$f(2.625) < 0 \Rightarrow$ Root lies between 2.625 and 2.75

$\therefore$ Approximately the root will be $= \dfrac{2.625+2.75}{2} = 2.69$

Objective Type Questions

I. Choose correct answer:

1. An example of an algebraic equation is

(1) $\tan x = e^x$ (2) $x = \log x$ (3) $x^3 - 5x + 3 = 0$ (4) None

[***Ans:*** (3)]

2. An example of a transcendental equation is

(1) $x^3 - 2x - 10 = 0$ (2) $x^3 e^x = 5$

(3) $x^2 + 11x - 1 = 0$ (4) None

[***Ans:*** (2)]

3. In finding a real root of the equation $x^3 - x - 10 = 0$ by bisection, if the root lies between $x_0 = 2$ and $x_1 = 3$, then, $x_2 =$

(1) 2.5 (2) 2.75 (3) 2.60 (4) None

[***Ans:*** (1)]

4. If ϕ(a) and ϕ(b) are of opposite signs and the real root of the equation $\phi(x) = 0$ is found by false position method, the first approximation x_1, of the root is

(1) $\dfrac{a\,\phi(b) + b\,\phi(a)}{\phi(b) + \phi(a)}$ (2) $\dfrac{a\,\phi'(b) + b\,\phi'(a)}{\phi(b) + \phi(a)}$

(3) $\dfrac{ab\,\phi(a)\,\phi(b)}{\phi(a) - \phi(b)}$ (4) $\dfrac{a\,\phi(b) - b\,\phi(a)}{\phi(b) - \phi(a)}$

[***Ans:*** (4)]

5. The two initial values of the roots of the equation $x^3 - x - 3 = 0$ are

(1) (–1, 0) (2) 1, 2 (3) –2, 1 (4) (1, 0)

[***Ans:*** (2)]

6. The iteration method is said to have p^{th} order convergence if for any finite constant $K \neq 0$

(1) $|e_n| \leq K|e_{n-1}|^P$ (2) $|e_n| \leq K|e_{n+1}|^P$

(3) $|e_n + 1| \leq K|e_0|^P$ (4) None

[***Ans:*** (1)]

7. Newton-Raphson method formula to find $(n + 1)^{th}$ approximation of root of $f(x) = 0$ is

(1) $x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$ (2) $x_{n+1} = \frac{x_n f(x_n)}{f'(x_n)}$

(3) $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ (4) None

[***Ans:*** (3)]

8. In the bisection method e_0 is the initial error and e_n is the error in n^{th} iteration

(1) $\frac{1}{2}$ (2) 1 (3) $\frac{1}{2^n}$ (4) None

[***Ans:*** (3)]

9. Which of the following methods has linear rate of convergence

(1) Regular flase (2) Bisection

(3) Newton-Raphson (4) None

[***Ans:*** (1)]

10. A non linear equation $x^3 + x^2 - 1 = 0$ is $x = \phi(x)$, then the choice of $\phi(x)$ for which the iteration scheme $x_n = \phi(x_{n-1})$ $x_0 = 1$ converge is $\phi(x)$=

(1) $(1 - x^2)^{1/3}$ (2) $\frac{1}{\sqrt{1+x}}$ (3) $\sqrt{1-x^3}$ (d) None

[***Ans:*** (2)]

Introduction to Numerical Methods

Lecture notes for MATH 3311

Jeffrey R. Chasnov

The Hong Kong University of
Science and Technology

The Hong Kong University of Science and Technology
Department of Mathematics
Clear Water Bay, Kowloon
Hong Kong

Preface

What follows are my lecture notes for Math 3311: *Introduction to Numerical Methods*, taught at the Hong Kong University of Science and Technology. Math 3311, with two lecture hours per week, is primarily for non-mathematics majors and is required by several engineering departments.

All web surfers are welcome to download these notes at

http://www.math.ust.hk/~machas/numerical-methods.pdf

and to use the notes freely for teaching and learning. I welcome any comments, suggestions or corrections sent by email to jeffrey.chasnov@ust.hk.

Contents

Chapter 1

IEEE Arithmetic

Definitions

Bit	=	0 or 1	
Byte	=	8 bits	
Word	=	Reals:	4 bytes (single precision)
			8 bytes (double precision)
	=	Integers:	1, 2, 4, or 8 byte signed
			1, 2, 4, or 8 byte unsigned

Numbers with a decimal or binary point

	Q	Q	Q	Q	.	Q	Q	Q	Q
Decimal:	10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}	10^{-4}
Binary:	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}

Examples of binary numbers

Decimal	Binary
1	1
2	10
3	11
4	100
0.5	0.1
1.5	1.1

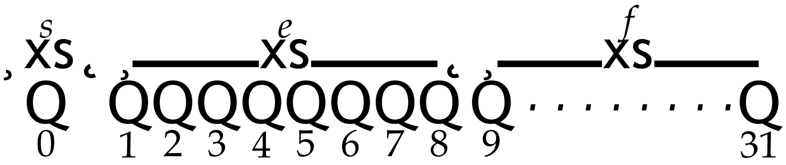
Hex numbers

$\{0, 1, 2, 3, \ldots, 9, 10, 11, 12, 13, 14, 15\} = \{0, 1, 2, 3.......9, a,b,c,d,e,f\}$

1.54-bit unsigned integers as hex numbers

Decimal	Binary	Hex
1	0001	1
2	0010	2
3	0011	3
.	.	.
10	1010	a
.	.	.
15	1111	f

IEEE single precision format:



$$\# = (-1)^s \times 2^{e-127} \times 1.f$$

where s = sign
e = biased exponent
p=e-127 = exponent
1.f = significand (use binary point)

Special numbers

Smallest exponent: e = 0000 0000, represents denormal numbers ($1.f \rightarrow 0.f$)
Largest exponent: e = 1111 1111, represents $\pm\infty$, if f = 0
e = 1111 1111, represents NaN, if f $\neq$ 0

Number Range: e = 1111 1111 = 2^8 - 1 = 255 reserved
e = 0000 0000 = 0 reserved
so, p = e - 127 is
$1 - 127 \leq p \leq 254-127$
$-126 \leq p \leq 127$

Smallest positive normal number
= $1.0000\ 0000 \cdots\cdots 0000 \times 2^{-126}$
$\approx 1.2 \times 10^{-38}$
bin: 0000 0000 1000 0000 0000 0000 0000 0000
hex: 00800000
MATLAB: realmin('single')

Largest positive number
= $1.1111\ 1111 \cdots\cdots 1111 \times 2^{127}$
= $(1 + (1 - 2^{-23})) \times 2^{127}$
$\approx 2^{128} \approx 3.4 \times 10^{38}$
bin: 0111 1111 0111 1111 1111 1111 1111 1111
hex: 7f7fffff
MATLAB: realmax('single')

Zero
bin: 0000 0000 0000 0000 0000 0000 0000 0000
hex: 00000000

Subnormal numbers
Allow $1.f \rightarrow 0.f$ (in software)
Smallest positive number = $0.0000\ 0000 \cdots\cdots 0001 \times 2^{-126}$
$= 2^{-23} \times 2^{-126} \approx 1.4 \times 10^{-45}$

Examples of computer numbers

What is 1.0, 2.0 & 1/2 in hex ?

$1.0 = (-1)^0 \times 2^{(127-127)} \times 1.0$
Therefore, $s = 0$, $e = 0111\ 1111$, $f = 0000\ 0000\ 0000\ 0000\ 0000\ 000$
bin: 0011 1111 1000 0000 0000 0000 0000 0000
hex: 3f80 0000

$2.0 = (-1)^0 \times 2^{(128-127)} \times 1.0$
Therefore, $s = 0$, $e = 1000\ 0000$, $f = 0000\ 0000\ 0000\ 0000\ 0000\ 000$
bin: 0100 00000 1000 0000 0000 0000 0000 0000
hex: 4000 0000

$1/2 = (-1)^0 \times 2^{(126-127)} \times 1.0$
Therefore, $s = 0$, $e = 0111\ 1110$, $f = 0000\ 0000\ 0000\ 0000\ 0000\ 000$
bin: 0011 1111 0000 0000 0000 0000 0000 0000
hex: 3f00 0000

Inexact numbers

Example:

$$\frac{1}{3} = (-1)^0 \times \frac{1}{4} \times (1 + \frac{1}{3}),$$

so that $p = e - 127 = -2$ and $e = 125 = 128 - 3$, or in binary, $e = 0111\ 1101$. How is $f = 1/3$ represented in binary? To compute binary number, multiply successively by 2 as follows:

0.333 . . .	0.
0.666 . . .	0.0
1.333 . . .	0.01
0.666 . . .	0.010
1.333 . . .	0.0101
etc.	

so that 1/3 exactly in binary is 0.010101 . . . . With only 23 bits to represent f, the number is inexact and we have

$$f = 01010101010101010101011,$$

where we have rounded to the nearest binary number (here, rounded up). The machine number 1/3 is then represented as

$$00111110101010101010101010101011$$

or in hex

$$3eaaaaab.$$

1.9.1Find smallest positive integer that is not exact in single precision

Let N be the smallest positive integer that is not exact. Now, I claim that

$$N - 2 = 2^{23} \times 1.11\ldots1,$$

and

$$N - 1 = 2^{24} \times 1.00\ldots0.$$

The integer N would then require a one-bit in the 2^{-24} position, which is not available. Therefore, the smallest positive integer that is not exact is $2^{24}+1 = 16\,777\,217$. In MATLAB, single(2^{24}) has the same value as single($2^{24}+1$). Since single($2^{24}+1$) is exactly halfway between the two consecutive machine numbers 2^{24} and $2^{24}+2$, MATLAB rounds to the number with a final zero-bit in f, which is 2^{24}.

Machine epsilon

Machine epsilon (ϵ_{mach}) is the distance between 1 and the next largest number. If $0 \le \delta < \epsilon_{\text{mach}}/2$, then $1 + \delta = 1$ in computer math. Also since

$$x + y = x(1 + y/x),$$

if $0 \le y/x < \epsilon_{\text{mach}}/2$, then $x + y = x$ in computer math.

Find ϵ_{mach}

The number 1 in the IEEE format is written as

$$1 = 2^0 \times 1.000\ldots0,$$

with 23 0's following the binary point. The number just larger than 1 has a 1 in the 23rd position after the decimal point. Therefore,

$$\epsilon_{\text{mach}} = 2^{-23} \approx 1.192 \times 10^{-7}.$$

What is the distance between 1 and the number just smaller than 1? Here, the number just smaller than one can be written as

$$2^{-1} \times 1.111\ldots1 = 2^{-1}(1 + (1 - 2^{-23})) = 1 - 2^{-24}$$

Therefore, this distance is $2^{-24} = \epsilon_{\text{mach}}/2$.

The spacing between numbers is uniform between powers of 2, with logarithmic spacing of the powers of 2. That is, the spacing of numbers between 1 and 2 is 2^{-23}, between 2 and 4 is 2^{-22}, between 4 and 8 is 2^{-21}, etc. This spacing changes for denormal numbers, where the spacing is uniform all the way down to zero.

Find the machine number just greater than 5

A rough estimate would be $5(1 + \epsilon_{\text{mach}}) = 5 + 5\epsilon_{\text{mach}}$, but this is not exact. The exact answer can be found by writing

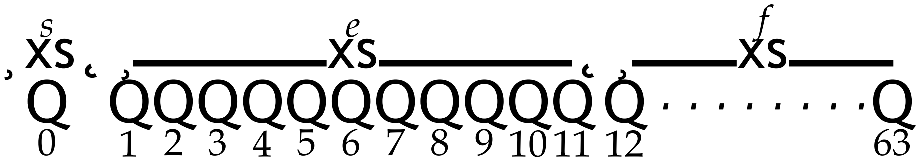
$$5 = 2^2(1 + \frac{1}{4}),$$

so that the next largest number is

$$2^2(1 + \frac{1}{4} + 2^{-23}) = 5 + 2^{-21} = 5 + 4\epsilon_{\text{mach}}.$$

IEEE double precision format

Most computations take place in double precision, where round-off error is reduced, and all of the above calculations in single precision can be repeated for double precision. The format is



$$\# = (-1)^s \times 2^{e-1023} \times 1.\text{f}$$

where s = sign
e = biased exponent
p=e-1023 = exponent
1.f = significand (use binary point)

Roundoff error example

Consider solving the quadratic equation

$$x^2 + 2bx - 1 = 0,$$

where b is a parameter. The quadratic formula yields the two solutions

$$x_\pm = -b \pm \sqrt{b^2 + 1}.$$

Consider the solution with $b > 0$ and $x > 0$ (the x_+ solution) given by

$$x = -b + \sqrt{b^2 + 1}. \tag{1.1}$$

As $b \to \infty$,

$$\begin{aligned} x &= -b + \sqrt{b^2 + 1} \\ &= -b + b\sqrt{1 + 1/b^2} \\ &= b\left(\sqrt{1 + 1/b^2} - 1\right) \\ &\approx b\left(\frac{1}{2b^2}\right) \\ &= \frac{1}{2b}. \end{aligned}$$

Now in double precision, realmin $\approx 2.2 \times 10^{-308}$ and we would like x to be accurate to this value before it goes to 0 via denormal numbers. Therefore, x should be computed accurately to $b \approx 1/(2 \times \text{realmin}) \approx 2 \times 10^{307}$. What happens if we compute (1.1) directly? Then $x = 0$ when $b^2 + 1 = b^2$, or $1 + 1/b^2 = 1$. That is $1/b^2 = \epsilon_{\text{mach}}/2$, or $b = \sqrt{2}/\sqrt{\epsilon_{\text{mach}}} \approx 10^8$.

For a subroutine written to compute the solution of a quadratic for a general user, this is not good enough. The way for a software designer to solve this problem is to compute the solution for x as

$$x = \frac{1}{b + \sqrt{b^2 + 1}}.$$

In this form, if $b^2 + 1 = b^2$, then $x = 1/2b$ which is the correct asymptotic form.

Chapter 2

Root Finding

Solve $f(x) = 0$ for x, when an explicit analytical solution is impossible.

Bisection Method

The bisection method is the easiest to numerically implement and almost always works. The main disadvantage is that convergence is slow. If the bisection method results in a computer program that runs too slow, then other faster methods may be chosen; otherwise it is a good choice of method.

We want to construct a sequence $x_0, x_1, x_2, \ldots$ that converges to the root $x = r$ that solves $f(x) = 0$. We choose x_0 and x_1 such that $x_0 < r < x_1$. We say that x_0 and x_1 bracket the root. With $f(r) = 0$, we want $f(x_0)$ and $f(x_1)$ to be of opposite sign, so that $f(x_0) f(x_1) < 0$. We then assign x_2 to be the midpoint of x_0 and x_1, that is $x_2 = (x_0 + x_1)/2$, or

$$x_2 = x_0 + \frac{x_1 - x_0}{2}.$$

The sign of $f(x_2)$ can then be determined. The value of x_3 is then chosen as either the midpoint of x_0 and x_2 or as the midpoint of x_2 and x_1, depending on whether x_0 and x_2 bracket the root, or x_2 and x_1 bracket the root. The root, therefore, stays bracketed at all times. The algorithm proceeds in this fashion and is typically stopped when the increment to the left side of the bracket (above, given by $(x_1 - x_0)/2$) is smaller than some required precision.

Newton's Method

This is the fastest method, but requires analytical computation of the derivative of $f(x)$. Also, the method may not always converge to the desired root.

We can derive Newton's Method graphically, or by a Taylor series. We again want to construct a sequence $x_0, x_1, x_2, \ldots$ that converges to the root $x = r$. Consider the x_{n+1} member of this sequence, and Taylor series expand $f(x_{n+1})$ about the point x_n. We have

$$f(x_{n+1}) = f(x_n) + (x_{n+1} - x_n) f'(x_n) + \ldots.$$

To determine x_{n+1}, we drop the higher-order terms in the Taylor series, and assume $f(x_{n+1}) = 0$. Solving for x_{n+1}, we have

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Starting Newton's Method requires a guess for x_0, hopefully close to the root $x = r$.

Secant Method

The Secant Method is second best to Newton's Method, and is used when a faster convergence than Bisection is desired, but it is too difficult or impossible to take an analytical derivative of the function $f(x)$. We write in place of $f'(x_n)$,

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Starting the Secant Method requires a guess for both x_0 and x_1.

2.3.1 Estimate $\sqrt{2} = 1.41421356$ using Newton's Method

The $\sqrt{2}$ is the zero of the function $f(x) = x^2 - 2$. To implement Newton's Method, we use $f'(x) = 2x$. Therefore, Newton's Method is the iteration

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n}.$$

We take as our initial guess $x_0 = 1$. Then

$$x_1 = 1 - \frac{-1}{2} = \frac{3}{2} = 1.5,$$

$$x_2 = \frac{3}{2} - \frac{\frac{9}{4} - 2}{3} = \frac{17}{12} = 1.416667,$$

$$x_3 = \frac{17}{12} - \frac{\frac{17^2}{12^2} - 2}{\frac{17}{6}} = \frac{577}{408} = 1.41426.$$

2.3.2 Example of fractals using Newton's Method

Consider the complex roots of the equation $f(z) = 0$, where

$$f(z) = z^3 - 1.$$

These roots are the three cubic roots of unity. With

$$e^{i2\pi n} = 1, \quad n = 0, 1, 2, \ldots$$

the three unique cubic roots of unity are given by

$$1, \quad e^{i2\pi/3}, \quad e^{i4\pi/3}.$$

With

$$e^{i\theta} = \cos\theta + i\sin\theta,$$

and $\cos(2\pi/3) = -1/2$, $\sin(2\pi/3) = \sqrt{3}/2$, the three cubic roots of unity are

$$r_1 = 1, \quad r_2 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i, \quad r_3 = -\frac{1}{2} - \frac{\sqrt{3}}{2}i.$$

The interesting idea here is to determine which initial values of z_0 in the complex plane converge to which of the three cubic roots of unity.

Newton's method is

$$z_{n+1} = z_n - \frac{z_n^3 - 1}{3z_n^2}.$$

If the iteration converges to r_1, we color z_0 red; r_2, blue; r_3, green. The result will be shown in lecture.

Order of convergence

Let r be the root and x_n be the nth approximation to the root. Define the error as

$$\epsilon_n = r - x_n.$$

If for large n we have the approximate relationship

$$|\epsilon_{n+1}| = k|\epsilon_n|^p,$$

with k a positive constant, then we say the root-finding numerical method is of order p. Larger values of p correspond to faster convergence to the root. The order of convergence of bisection is one: the error is reduced by approximately a factor of 2 with each iteration so that

$$|\epsilon_{n+1}| = \frac{1}{2}|\epsilon_n|.$$

We now find the order of convergence for Newton's Method and for the Secant Method.

Newton's Method

We start with Newton's Method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Subtracting both sides from r, we have

$$r - x_{n+1} = r - x_n + \frac{f(x_n)}{f'(x_n)},$$

or

$$\epsilon_{n+1} = \epsilon_n + \frac{f(x_n)}{f'(x_n)}. \tag{2.1}$$

We use Taylor series to expand the functions $f(x_n)$ and $f'(x_n)$ about the root r, using $f(r) = 0$. We have

$$\begin{aligned} f(x_n) &= f(r) + (x_n - r)f'(r) + \frac{1}{2}(x_n - r)^2 f''(r) + \ldots, \\ &= -\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \ldots; \\ f'(x_n) &= f'(r) + (x_n - r)f''(r) + \frac{1}{2}(x_n - r)^2 f'''(r) + \ldots, \\ &= f'(r) - \epsilon_n f''(r) + \frac{1}{2}\epsilon_n^2 f'''(r) + \ldots . \end{aligned} \tag{2.2}$$

To make further progress, we will make use of the following standard Taylor series:

$$\frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \ldots, \tag{2.3}$$

which converges for $|\epsilon| < 1$. Substituting (2.2) into (2.1), and using (2.3) yields

$$\begin{aligned}
\epsilon_{n+1} &= \epsilon_n + \frac{f(x_n)}{f'(x_n)} \\
&= \epsilon_n + \frac{-\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \dots}{f'(r) - \epsilon_n f''(r) + \frac{1}{2}\epsilon_n^2 f'''(r) + \dots} \\
&= \epsilon_n + \frac{-\epsilon_n + \frac{1}{2}\epsilon_n^2 \frac{f''(r)}{f'(r)} + \dots}{1 - \epsilon_n \frac{f''(r)}{f'(r)} + \dots} \\
&= \epsilon_n + \left(-\epsilon_n + \frac{1}{2}\epsilon_n^2 \frac{f''(r)}{f'(r)} + \dots\right)\left(1 + \epsilon_n \frac{f''(r)}{f'(r)} + \dots\right) \\
&= \epsilon_n + \left(-\epsilon_n + \epsilon_n^2\left(\frac{1}{2}\frac{f''(r)}{f'(r)} - \frac{f''(r)}{f'(r)}\right) + \dots\right) \\
&= -\frac{1}{2}\frac{f''(r)}{f'(r)}\epsilon_n^2 + \dots
\end{aligned}$$

Therefore, we have shown that

$$|\epsilon_{n+1}| = k|\epsilon_n|^2$$

as $n \to \infty$, with

$$k = \frac{1}{2}\left|\frac{f''(r)}{f'(r)}\right|,$$

provided $f'(r) \neq 0$. Newton's method is thus of order 2 at simple roots.

Secant Method

Determining the order of the Secant Method proceeds in a similar fashion. We start with

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1})f(x_n)}{f(x_n) - f(x_{n-1})}.$$

We subtract both sides from r and make use of

$$\begin{aligned}
x_n - x_{n-1} &= (r - x_{n-1}) - (r - x_n) \\
&= \epsilon_{n-1} - \epsilon_n,
\end{aligned}$$

and the Taylor series

$$\begin{aligned}
f(x_n) &= -\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \dots, \\
f(x_{n-1}) &= -\epsilon_{n-1} f'(r) + \frac{1}{2}\epsilon_{n-1}^2 f''(r) + \dots,
\end{aligned}$$

so that

$$f(x_n) - f(x_{n-1}) = (\epsilon_{n-1} - \epsilon_n)f'(r) + \frac{1}{2}(\epsilon_n^2 - \epsilon_{n-1}^2) f''(r) + \dots$$

$$= (c_{n-1} - c_n)\left[f'(r) - \frac{1}{2}(c_{n-1} + c_n) f''(r) + \dots \right].$$

We therefore have

$$\begin{aligned}
\epsilon_{n+1} &= \epsilon_n + \frac{-\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \dots}{f'(r) - \frac{1}{2}(\epsilon_{n-1} + \epsilon_n) f''(r) + \dots} \\
&= \epsilon_n - \epsilon_n \frac{1 - \frac{1}{2}\epsilon_n \frac{f''(r)}{f'(r)} + \dots}{1 - \frac{1}{2}(\epsilon_{n-1} + \epsilon_n)\frac{f''(r)}{f'(r)} + \dots} \\
&= \epsilon_n - \epsilon_n \left(1 - \frac{1}{2}\epsilon_n \frac{f''(r)}{f'(r)} + \dots\right)\left(1 + \frac{1}{2}(\epsilon_{n-1} + \epsilon_n)\frac{f''(r)}{f'(r)} + \dots\right) \\
&= -\frac{1}{2}\frac{f''(r)}{f'(r)}\epsilon_{n-1}\epsilon_n + \dots,
\end{aligned}$$

or to leading order

$$|\epsilon_{n+1}| = \left|\frac{f''(r)}{2f'(r)}\right| |\epsilon_{n-1}||\epsilon_n|. \tag{2.4}$$

The order of convergence is not yet obvious from this equation, and to determine the scaling law we look for a solution of the form

$$|\epsilon_{n+1}| = k|\epsilon_n|^p.$$

From this ansatz, we also have

$$|\epsilon_n| = k|\epsilon_{n-1}|^p,$$

and therefore

$$|\epsilon_{n+1}| = k^{p+1}|\epsilon_{n-1}|^{p^2}.$$

Substitution into (2.4) results in

$$k^{p+1}|\epsilon_{n-1}|^{p^2} = \frac{k}{2}\left|\frac{f''(r)}{f'(r)}\right| |\epsilon_{n-1}|^{p+1}.$$

Equating the coefficient and the power of ϵ_{n-1} results in

$$k^p = \frac{1}{2}\left|\frac{f''(r)}{f'(r)}\right|,$$

and

$$p^2 = p + 1.$$

The order of convergence of the Secant Method, given by p, therefore is determined to be the positive root of the quadratic equation $p^2 - p - 1 = 0$, or

$$p = \frac{1 + \sqrt{5}}{2} \approx 1.618,$$

which coincidentally is a famous irrational number that is called The Golden Ratio, and goes by the symbol Φ. We see that the Secant Method has an order of convergence lying between the Bisection Method and Newton's Method.

Chapter 3

Systems of equations

Consider the system of n linear equations and n unknowns, given by

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2, \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned}$$

We can write this system as the matrix equation

$$\mathrm{A}\mathbf{x} = \mathbf{b},$$

with

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Gaussian Elimination

The standard numerical algorithm to solve a system of linear equations is called Gaussian Elimination. We can illustrate this algorithm by example.

Consider the system of equations

$$\begin{aligned} -3x_1 + 2x_2 - x_3 &= -1, \\ 6x_1 - 6x_2 + 7x_3 &= -7, \\ 3x_1 - 4x_2 + 4x_3 &= -6. \end{aligned}$$

To perform Gaussian elimination, we form an Augmented Matrix by combining the matrix A with the column vector $\mathbf{b}$:

$$\begin{pmatrix} -3 & 2 & -1 & -1 \\ 6 & -6 & 7 & -7 \\ 3 & -4 & 4 & -6 \end{pmatrix}.$$

Row reduction is then performed on this matrix. Allowed operations are (1) multiply any row by a constant, (2) add multiple of one row to another row, (3) interchange the order of any rows. The goal is to convert the original matrix into an upper-triangular matrix.

We start with the first row of the matrix and work our way down as follows. First we multiply the first row by 2 and add it to the second row, and add the first row to the third row:

$$\begin{pmatrix} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 0 & -2 & 3 & -7 \end{pmatrix}.$$

We then go to the second row. We multiply this row by -1 and add it to the third row:

$$\begin{pmatrix} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 0 & 0 & -2 & 2 \end{pmatrix}.$$

The resulting equations can be determined from the matrix and are given by

$$\begin{aligned} -3x_1 + 2x_2 - x_3 &= -1 \\ -2x_2 + 5x_3 &= -9 \\ -2x_3 &= 2. \end{aligned}$$

These equations can be solved by backward substitution, starting from the last equation and working backwards. We have

$$\begin{aligned} -2x_3 &= 2 \rightarrow x_3 = -1 \\ -2x_2 &= -9 - 5x_3 = -4 \rightarrow x_2 = 2, \\ -3x_1 &= -1 - 2x_2 + x_3 = -6 \rightarrow x_1 = 2. \end{aligned}$$

Therefore,

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}.$$

LU decomposition

The process of Gaussian Elimination also results in the factoring of the matrix A to

$$A = LU,$$

where L is a lower triangular matrix and U is an upper triangular matrix. Using the same matrix A as in the last section, we show how this factorization is realized. We have

$$\begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} \rightarrow \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix} = M_1A,$$

where

$$M_1A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} = \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix}.$$

Note that the matrix M_1 performs row elimination on the first column. Two times the first row is added to the second row and one times the first row is added to the third row. The entries of the column of M_1 come from $2 = -(6/-3)$ and $1 = -(3/-3)$ as required for row elimination. The number 3 is called the pivot.

The next step is

$$\begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix} = M_2(M_1A),$$

where

$$M_2(M_1A) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{pmatrix} = \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix}.$$

Here, M_2 multiplies the second row by $-1 = -(-2/-2)$ and adds it to the third row. The pivot is -2.

We now have

$$M_2M_1A = U$$

or

$$A = M_1^{-1}M_2^{-1}U.$$

The inverse matrices are easy to find. The matrix M_1 multiples the first row by 2 and adds it to the second row, and multiplies the first row by 1 and adds it to the third row. To invert these operations, we need to multiply the first row by -2 and add it to the second row, and multiply the first row by -1 and add it to the third row. To check, with

$$M_1M_1^{-1} = I,$$

we have

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Similarly,

$$M_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Therefore,

$$L = M_1^{-1}M_2^{-1}$$

is given by

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix},$$

which is lower triangular. The off-diagonal elements of M_1^{-1} and M_2^{-1} are simply combined to form L. Our LU decomposition is therefore

$$\begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}\begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix}.$$

Another nice feature of the LU decomposition is that it can be done by overwriting A, therefore saving memory if the matrix A is very large.

The LU decomposition is useful when one needs to solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ when A is fixed and there are many different $\mathbf{b}$'s. First one determines L and U using Gaussian elimination. Then one writes

$$(LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b}.$$

We let

$$\mathbf{y} = U\mathbf{x},$$

and first solve

$$L\mathbf{y} = \mathbf{b}$$

for $\mathbf{y}$ by forward substitution. We then solve

$$U\mathbf{x} = \mathbf{y}$$

for $\mathbf{x}$ by backward substitution. When we count operations, we will see that solving $(\mathrm{LU})\mathbf{x} = \mathbf{b}$ is significantly faster once L and U are in hand than solving $\mathrm{A}\mathbf{x} = \mathbf{b}$ directly by Gaussian elimination.

We now illustrate the solution of $\mathrm{LU}\mathbf{x} = \mathbf{b}$ using our previous example, where

$$\mathrm{L} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}, \quad \mathrm{U} = \begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} -1 \\ -7 \\ -6 \end{pmatrix}.$$

With $\mathbf{y} = \mathrm{U}\mathbf{x}$, we first solve $\mathrm{L}\mathbf{y} = \mathbf{b}$, that is

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} -1 \\ -7 \\ -6 \end{pmatrix}.$$

Using forward substitution

$$\begin{aligned} y_1 &= -1, \\ y_2 &= -7 + 2y_1 = -9, \\ y_3 &= -6 + y_1 - y_2 = 2. \end{aligned}$$

We now solve $\mathrm{U}\mathbf{x} = \mathbf{y}$, that is

$$\begin{pmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ -9 \\ 2 \end{pmatrix}.$$

Using backward substitution,

$$\begin{aligned} -2x_3 &= 2 \to x_3 = -1, \\ -2x_2 &= -9 - 5x_3 = -4 \to x_2 = 2, \\ -3x_1 &= -1 - 2x_2 + x_3 = -6 \to x_1 = 2, \end{aligned}$$

and we have once again determined

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}.$$

Partial pivoting

When performing Gaussian elimination, the diagonal element that one uses during the elimination procedure is called the pivot. To obtain the correct multiple, one uses the pivot as the divisor to the elements below the pivot. Gaussian elimination in this form will fail if the pivot is zero. In this situation, a row interchange must be performed.

Even if the pivot is not identically zero, a small value can result in big round-off errors. For very large matrices, one can easily lose all accuracy in the solution. To avoid these round-off errors arising from small pivots, row interchanges are made, and this technique is called partial pivoting (partial pivoting is in contrast to complete pivoting, where both rows and columns are interchanged). We will illustrate by example the LU decomposition using partial pivoting.

Consider

$$A = \begin{pmatrix} -2 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -8 & 4 \end{pmatrix}.$$

We interchange rows to place the largest element (in absolute value) in the pivot, or a_{11}, position. That is,

$$A \to \begin{pmatrix} 6 & -6 & 7 \\ -2 & 2 & -1 \\ 3 & -8 & 4 \end{pmatrix} = P_{12}A,$$

where

$$P_{12} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

is a permutation matrix that when multiplied on the left interchanges the first and second rows of a matrix. Note that $P_{12}^{-1} = P_{12}$. The elimination step is then

$$P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & 0 & 4/3 \\ 0 & -5 & 1/2 \end{pmatrix} = M_1P_{12}A,$$

where

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}.$$

The final step requires one more row interchange:

$$M_1P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & -5 & 1/2 \\ 0 & 0 & 4/3 \end{pmatrix} = P_{23}M_1P_{12}A = U.$$

Since the permutation matrices given by P are their own inverses, we can write our result as

$$(P_{23}M_1P_{23})P_{23}P_{12}A = U.$$

Multiplication of M on the left by P interchanges rows while multiplication on the right by P interchanges columns. That is,

$$P_{23}\begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 0 & 1 \\ 1/3 & 1 & 0 \end{pmatrix}P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/3 & 0 & 1 \end{pmatrix}.$$

The net result on M_1 is an interchange of the nondiagonal elements $1/3$ and $-1/2$.

We can then multiply by the inverse of $(P_{23}M_1P_{23})$ to obtain

$$P_{23}P_{12}A = (P_{23}M_1P_{23})^{-1}U,$$

which we write as

$$PA = LU.$$

Instead of L, MATLAB will write this as

$$A = (P^{-1}L)U.$$

For convenience, we will just denote $(P^{-1}L)$ by L, but by L here we mean a permutated lower triangular matrix.

For example, in MATLAB, to solve A**x** = **b** for **x** using Gaussian elimination, one types

```
x = A \ b;
```

where \ solves for **x** using the most efficient algorithm available, depending on the form of A. If A is a general $n \times n$ matrix, then first the LU decomposition of A is found using partial pivoting, and then x is determined from permuted forward and backward substitution. If A is upper or lower triangular, then forward or backward substitution (or their permuted version) is used directly.

If there are many different right-hand-sides, one would first directly find the LU decomposition of A using a function call, and then solve using \. That is, one would iterate for different **b**'s the following expressions:

```
[LU] = lu(A);
y = L \ b;
x = U \ y;
```

where the second and third lines can be shortened to

```
x = U \ (L \ b);
```

where the parenthesis are required. In lecture, I will demonstrate these solutions in MATLAB using the matrix A = [−2, 2, −1; 6, −6, 7; 3, −8, 4]; which is the example in the notes.

Operation counts

To estimate how much computational time is required for an algorithm, one can count the number of operations required (multiplications, divisions, additions and subtractions). Usually, what is of interest is how the algorithm scales with the size of the problem. For example, suppose one wants to multiply two full $n \times n$ matrices. The calculation of each element requires n multiplications and $n - 1$ additions, or say $2n - 1$ operations. There are n^2 elements to compute so that the total operation count is $n^2(2n - 1)$. If n is large, we might want to know what will happen to the computational time if n is doubled. What matters most is the fastest-growing, leading-order term in the operation count. In this matrix multiplication example, the operation count is $n^2(2n - 1) = 2n^3 - n^2$, and the leading-order term is $2n^3$. The factor of 2 is unimportant for the scaling, and we say that the algorithm scales like $O(n^3)$, which is read as "big Oh of n cubed." When using the big-Oh notation, we will drop both lower-order terms and constant multipliers.

The big-Oh notation tells us how the computational time of an algorithm scales. For example, suppose that the multiplication of two large $n \times n$ matrices took a computational time of T_n seconds. With the known operation count going like $O(n^3)$, we can write

$$T_n = kn^3$$

for some unknown constant k. To determine how long multiplication of two $2n \times 2n$

matrices will take, we write

$$\begin{aligned} T_{2n} &= k(2n)^3 \\ &= 8kn^3 \\ &= 8T_n, \end{aligned}$$

so that doubling the size of the matrix is expected to increase the computational time by a factor of $2^3 = 8$.

Running MATLAB on my computer, the multiplication of two 2048×2048 matrices took about 0.75 sec. The multiplication of two 4096×4096 matrices took about 6 sec, which is 8 times longer. Timing of code in MATLAB can be found using the built-in stopwatch functions tic and toc.

What is the operation count and therefore the scaling of Gaussian elimination? Consider an elimination step with the pivot in the ith row and ith column. There are both $n - i$ rows below the pivot and $n - i$ columns to the right of the pivot. To perform elimination of one row, each matrix element to the right of the pivot must be multiplied by a factor and added to the row underneath. This must be done for all the rows. There are therefore $(n - i)(n - i)$ multiplication-additions to be done for this pivot. Since we are interested in only the scaling of the algorithm, I will just count a multiplication-addition as one operation.

To find the total operation count, we need to perform elimination using $n - 1$ pivots, so that

$$\begin{aligned} \text{op. counts} &= \sum_{i=1}^{n-1} (n-i)^2 \\ &= (n-1)^2 + (n-2)^2 + \ldots (1)^2 \\ &= \sum_{i=1}^{n-1} i^2. \end{aligned}$$

Three summation formulas will come in handy. They are

$$\begin{aligned} \sum_{i=1}^{n} 1 &= n, \\ \sum_{i=1}^{n} i &= \frac{1}{2} n(n+1), \\ \sum_{i=1}^{n} i^2 &= \frac{1}{6} n(2n+1)(n+1), \end{aligned}$$

which can be proved by mathematical induction, or derived by some tricks.

The operation count for Gaussian elimination is therefore

$$\begin{aligned} \text{op. counts} &= \sum_{i=1}^{n-1} i^2 \\ &= \frac{1}{6}(n-1)(2n-1)(n). \end{aligned}$$

The leading-order term is therefore $n^3/3$, and we say that Gaussian elimination scales like $O(n^3)$. Since LU decomposition with partial pivoting is essentially Gaussian elimination, that algorithm also scales like $O(n^3)$.

However, once the LU decomposition of a matrix A is known, the solution of A**x** = **b** can proceed by a forward and backward substitution. How does a backward substitution, say, scale? For backward substitution, the matrix equation to be solved is of the form

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \cdots & 0 & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}.$$

The solution for x_i is found after solving for x_j with $j > i$. The explicit solution for x_i is given by

$$x_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=i+1}^{n} a_{i,j}x_j\right).$$

The solution for x_i requires $n - i + 1$ multiplication-additions, and since this must be done for n such i's, we have

$$\begin{aligned} \text{op. counts} &= \sum_{i=1}^{n} n - i + 1 \\ &= n + (n-1) + \cdots + 1 \\ &= \sum_{i=1}^{n} i \\ &= \frac{1}{2}n(n+1). \end{aligned}$$

The leading-order term is $n^2/2$ and the scaling of backward substitution is $O(n^2)$. After the LU decomposition of a matrix A is found, only a single forward and backward substitution is required to solve A**x** = **b**, and the scaling of the algorithm to solve this matrix equation is therefore still $O(n^2)$. For large n, one should expect that solving A**x** = **b** by a forward and backward substitution should be substantially faster than a direct solution using Gaussian elimination.

System of nonlinear equations

A system of nonlinear equations can be solved using a version of Newton's Method. We illustrate this method for a system of two equations and two unknowns. Suppose that we want to solve

$$f(x, y) = 0, \quad g(x, y) = 0,$$

for the unknowns x and y. We want to construct two simultaneous sequences $x_0, x_1, x_2, \ldots$ and $y_0, y_1, y_2, \ldots$ that converge to the roots. To construct these sequences, we Taylor series expand $f(x_{n+1}, y_{n+1})$ and $g(x_{n+1}, y_{n+1})$ about the point (x_n, y_n). Using the partial derivatives $f_x = \partial f/\partial x$, $f_y = \partial f/\partial y$, etc., the two-dimensional Taylor series expansions, displaying only the linear terms, are given by

$$f(x_{n+1}, y_{n+1}) = f(x_n, y_n) + (x_{n+1} - x_n) f_x(x_n, y_n) \\ + (y_{n+1} - y_n) f_y(x_n, y_n) + \dots$$

$$g(x_{n+1}, y_{n+1}) = g(x_n, y_n) + (x_{n+1} - x_n)g_x(x_n, y_n) + (y_{n+1} - y_n)g_y(x_n, y_n) + \dots .$$

To obtain Newton's method, we take $f(x_{n+1}, y_{n+1}) = 0$, $g(x_{n+1}, y_{n+1}) = 0$ and drop higher-order terms above linear. Although one can then find a system of linear equations for x_{n+1} and y_{n+1}, it is more convenient to define the variables

$$\Delta x_n = x_{n+1} - x_n, \qquad \Delta y_n = y_{n+1} - y_n.$$

The iteration equations will then be given by

$$x_{n+1} = x_n + \Delta x_n, \qquad y_{n+1} = y_n + \Delta y_n;$$

and the linear equations to be solved for Δx_n and Δy_n are given by

$$\begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \end{pmatrix} = \begin{pmatrix} -f \\ -g \end{pmatrix},$$

where f, g, f_x, f_y, g_x, and g_y are all evaluated at the point (x_n, y_n). The two-dimensional case is easily generalized to n dimensions. The matrix of partial derivatives is called the Jacobian Matrix.

We illustrate Newton's Method by finding the steady state solution of the Lorenz equations, given by

$$\begin{aligned} \sigma(y - x) &= 0, \\ rx - y - xz &= 0, \\ xy - bz &= 0, \end{aligned}$$

where x, y, and z are the unknown variables and σ, r, and b are the known parameters. Here, we have a three-dimensional homogeneous system $f = 0$, $g = 0$, and $h = 0$, with

$$\begin{aligned} f(x, y, z) &= \sigma(y - x), \\ g(x, y, z) &= rx - y - xz, \\ h(x, y, z) &= xy - bz. \end{aligned}$$

The partial derivatives can be computed to be

$$\begin{aligned} f_x &= -\sigma, & f_y &= \sigma, & f_z &= 0, \\ g_x &= r - z, & g_y &= -1, & g_z &= -x, \\ h_x &= y, & h_y &= x, & h_z &= -b. \end{aligned}$$

The iteration equation is therefore

$$\begin{pmatrix} -\sigma & \sigma & 0 \\ r - z_n & -1 & -x_n \\ y_n & x_n & -b \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \\ \Delta z_n \end{pmatrix} = - \begin{pmatrix} \sigma(y_n - x_n) \\ rx_n - y_n - x_n z_n \\ x_n y_n - bz_n \end{pmatrix},$$

with

$$\begin{aligned} x_{n+1} &= x_n + \Delta x_n, \\ y_{n+1} &= y_n + \Delta y_n, \\ z_{n+1} &= z_n + \Delta z_n. \end{aligned}$$

The MATLAB program that solves this system is contained in newton_system.m.

Chapter 4

Least-squares approximation

The method of least-squares is commonly used to fit a parameterized curve to experimental data. In general, the fitting curve is not expected to pass through the data points, making this problem substantially different from the one of interpolation.

We consider here only the simplest case of the same experimental error for all the data points. Let the data to be fitted be given by (x_i, y_i), with $i = 1$ to n.

Fitting a straight line

Suppose the fitting curve is a line. We write for the fitting curve

$$y(x) = \alpha x + \beta.$$

The distance r_i from the data point (x_i, y_i) and the fitting curve is given by

$$\begin{aligned} r_i &= y_i - y(x_i) \\ &= y_i - (\alpha x_i + \beta). \end{aligned}$$

A least-squares fit minimizes the sum of the squares of the r_i's. This minimum can be shown to result in the most probable values of α and β.

We define

$$\begin{aligned} \rho &= \sum_{i=1}^{n} r_i^2 \\ &= \sum_{i=1}^{n} \left(y_i - (\alpha x_i + \beta) \right)^2. \end{aligned}$$

To minimize ρ with respect to α and β, we solve

$$\frac{\partial \rho}{\partial \alpha} = 0, \quad \frac{\partial \rho}{\partial \beta} = 0.$$

Taking the partial derivatives, we have

$$\begin{aligned} \frac{\partial \rho}{\partial \alpha} &= \sum_{i=1}^{n} 2(-x_i) \left(y_i - (\alpha x_i + \beta) \right) = 0, \\ \frac{\partial \rho}{\partial \beta} &= \sum_{i=1}^{n} 2(-1) \left(y_i - (\alpha x_i + \beta) \right) = 0. \end{aligned}$$

These equations form a system of two linear equations in the two unknowns α and β, which is evident when rewritten in the form

$$\begin{aligned} \alpha \sum_{i=1}^{n} x_i^2 + \beta \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} x_i y_i, \\ \alpha \sum_{i=1}^{n} x_i + \beta n &= \sum_{i=1}^{n} y_i. \end{aligned}$$

These equations can be solved either analytically, or numerically in MATLAB, where the matrix form is

$$\begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & n \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{pmatrix}.$$

A proper statistical treatment of this problem should also consider an estimate of the errors in α and β as well as an estimate of the goodness-of-fit of the data to the model. We leave these further considerations to a statistics class.

Fitting to a linear combination of functions

Consider the general fitting function

$$y(x) = \sum_{j=1}^{m} c_j f_j(x),$$

where we assume m functions $f_j(x)$. For example, if we want to fit a cubic polynomial to the data, then we would have $m = 4$ and take $f_1 = 1$, $f_2 = x$, $f_3 = x^2$ and $f_4 = x^3$. Typically, the number of functions f_j is less than the number of data points; that is, $m < n$, so that a direct attempt to solve for the c_j's such that the fitting function exactly passes through the n data points would result in n equations and m unknowns. This would be an over-determined linear system that in general has no solution.

We now define the vectors

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix},$$

and the n-by-m matrix

$$\mathrm{A} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_m(x_n) \end{pmatrix}. \tag{4.1}$$

With $m < n$, the equation $\mathrm{A}\mathbf{c} = \mathbf{y}$ is over-determined. We let

$$\mathbf{r} = \mathbf{y} - \mathrm{A}\mathbf{c}$$

be the residual vector, and let

$$\rho = \sum_{i=1}^{n} r_i^2.$$

The method of least squares minimizes ρ with respect to the components of $\mathbf{c}$. Now, using T to signify the transpose of a matrix, we have

$$\begin{aligned} \rho &= \mathbf{r}^T \mathbf{r} \\ &= (\mathbf{y} - \mathrm{A}\mathbf{c})^T (\mathbf{y} - \mathrm{A}\mathbf{c}) \end{aligned}$$

$$= \mathbf{y}^T \mathbf{y} - \mathbf{c}^T \mathrm{A}^T \mathbf{y} - \mathbf{y}^T \mathrm{A}\mathbf{c} + \mathbf{c}^T \mathrm{A}^T \mathrm{A}\mathbf{c}.$$

4.2. FITTING TO A LINEAR COMBINATION OF FUNCTIONS

Since ρ is a scalar, each term in the above expression must be a scalar, and since the transpose of a scalar is equal to the scalar, we have

$$\mathbf{c}^T A^T \mathbf{y} = \left(\mathbf{c}^T A^T \mathbf{y}\right)^T = \mathbf{y}^T A\mathbf{c}.$$

Therefore,

$$\rho = \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T A\mathbf{c} + \mathbf{c}^T A^T A\mathbf{c}.$$

To find the minimum of ρ, we will need to solve $\partial\rho/\partial c_j = 0$ for $j = 1, \ldots, m$. To take the derivative of ρ, we switch to a tensor notation, using the Einstein summation convention, where repeated indices are summed over their allowable range. We can write

$$\rho = y_i y_i - 2y_i A_{ik} c_k + c_i A^T_{ik} A_{kl} c_l.$$

Taking the partial derivative, we have

$$\frac{\partial\rho}{\partial c_j} = -2y_i A_{ik}\frac{\partial c_k}{\partial c_j} + \frac{\partial c_i}{\partial c_j} A^T_{ik} A_{kl} c_l + c_i A^T_{ik} A_{kl} \frac{\partial c_l}{\partial c_j}.$$

Now,

$$\frac{\partial c_i}{\partial c_j} = \begin{cases} 1, & \text{if } i = j; \\ 0, & \text{otherwise.} \end{cases}$$

Therefore,

$$\frac{\partial\rho}{\partial c_j} = -2y_i A_{ij} + A^T_{jk} A_{kl} c_l + c_i A^T_{ik} A_{kj}.$$

Now,

$$\begin{aligned} c_i A^T_{ik} A_{kj} &= c_i A_{ki} A_{kj} \\ &= A_{kj} A_{ki} c_i \\ &= A^T_{jk} A_{ki} c_i \\ &= A^T_{jk} A_{kl} c_l. \end{aligned}$$

Therefore,

$$\frac{\partial\rho}{\partial c_j} = -2y_i A_{ij} + 2A^T_{jk} A_{kl} c_l.$$

With the partials set equal to zero, we have

$$A^T_{jk} A_{kl} c_l = y_i A_{ij},$$

or

$$A^T_{jk} A_{kl} c_l = A^T_{ji} y_i,$$

In vector notation, we have

$$A^T A\mathbf{c} = A^T\mathbf{y}. \tag{4.2}$$

Equation (4.2) is the so-called normal equation, and can be solved for **c** by Gaussian elimination using the MATLAB backslash operator. After constructing the matrix A given by (4.1), and the vector **y** from the data, one can code in MATLAB

```
c = (A'A)\(A'y);
```

But in fact the MATLAB back slash operator will automatically solve the normal equations when the matrix A is not square, so that the MATLAB code

```
c = A\y;
```

yields the same result.

Chapter 5

Interpolation

Consider the following problem: Given the values of a *known* function $y = f(x)$ at a sequence of ordered points $x_0, x_1, \ldots, x_n$, find $f(x)$ for arbitrary x. When $x_0 \leq x \leq x_n$, the problem is called interpolation. When $x < x_0$ or $x > x_n$ the problem is called extrapolation.

With $y_i = f(x_i)$, the problem of interpolation is basically one of drawing a smooth curve through the known points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$. This is not the same problem as drawing a smooth curve that approximates a set of data points that have experimental error. This latter problem is called least-squares approximation.

Here, we will consider three interpolation algorithms: (1) polynomial interpolation; (2) piecewise linear interpolation, and; (3) cubic spline interpolation.

Polynomial interpolation

The $n + 1$ points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ can be interpolated by a unique polynomial of degree n. When $n = 1$, the polynomial is a linear function; when $n = 2$, the polynomial is a quadratic function. There are three standard algorithms that can be used to construct this unique interpolating polynomial, and we will present all three here, not so much because they are all useful, but because it is interesting to learn how these three algorithms are constructed.

When discussing each algorithm, we define $P_n(x)$ to be the unique nth degree polynomial that passes through the given $n + 1$ data points.

Vandermonde polynomial

This Vandermonde polynomial is the most straightforward construction of the interpolating polynomial $P_n(x)$. We simply write

$$P_n(x) = c_0x^n + c_1x^{n-1} + \cdots + c_n.$$

Then we can immediately form $n + 1$ linear equations for the $n + 1$ unknown coefficients $c_0, c_1, \ldots, c_n$ using the $n + 1$ known points:

$$\begin{aligned}
y_0 &= c_0x_0^n + c_1x_0^{n-1} + \cdots + c_{n-1}x_0 + c_n \\
y_2 &= c_0x_1^n + c_1x_1^{n-1} + \cdots + c_{n-1}x_1 + c_n \\
&\vdots \\
y_n &= c_0x_n^n + c_1x_n^{n-1} + \cdots + c_{n-1}x_n + c_n.
\end{aligned}$$

The system of equations in matrix form is

$$\begin{pmatrix}
x_0^n & x_0^{n-1} & \cdots & x_0 & 1 \\
x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
x_n^n & x_n^{n-1} & \cdots & x_n & 1
\end{pmatrix}
\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix}
=
\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The matrix is called the Vandermonde matrix, and can be constructed using the MATLAB function vander.m. The system of linear equations can be solved in MATLAB using the \ operator, and the MATLAB function polyval.m can used to interpolate using the c coefficients. I will illustrate this in class and place the code on the website.

Lagrange polynomial

The Lagrange polynomial is the most clever construction of the interpolating polynomial $P_n(x)$, and leads directly to an analytical formula. The Lagrange polynomial is the sum of $n + 1$ terms and each term is itself a polynomial of degree n. The full polynomial is therefore of degree n. Counting from 0, the ith term of the Lagrange polynomial is constructed by requiring it to be zero at x_j with $j = i$, and equal to y_i when $j = i$. The polynomial can be written as

$$
\begin{aligned}
P_n(x) = {} & \frac{(x - x_1)(x - x_2) \cdots (x - x_n) y_0}{(x_0 - x_1)(x_0 - x_2) \cdots (x_0 - x_n)} + \frac{(x - x_0)(x - x_2) \cdots (x - x_n) y_1}{(x_1 - x_0)(x_1 - x_2) \cdots (x_1 - x_n)} \\
& + \cdots + \frac{(x - x_0)(x - x_1) \cdots (x - x_{n-1}) y_n}{(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1})}.
\end{aligned}
$$

It can be clearly seen that the first term is equal to zero when $x = x_1, x_2, \ldots, x_n$ and equal to y_0 when $x = x_0$; the second term is equal to zero when $x = x_0, x_2, \ldots x_n$ and equal to y_1 when $x = x_1$; and the last term is equal to zero when $x = x_0, x_1, \ldots x_{n-1}$ and equal to y_n when $x = x_n$. The uniqueness of the interpolating polynomial implies that the Lagrange polynomial must be the interpolating polynomial.

Newton polynomial

The Newton polynomial is somewhat more clever than the Vandermonde polynomial because it results in a system of linear equations that is lower triangular, and therefore can be solved by forward substitution. The interpolating polynomial is written in the form

$$
P_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0) \cdots (x - x_{n-1}),
$$

which is clearly a polynomial of degree n. The $n + 1$ unknown coefficients given by the c's can be found by substituting the points (x_i, y_i) for $i = 0, \ldots, n$:

$$
\begin{aligned}
y_0 &= c_0, \\
y_1 &= c_0 + c_1(x_1 - x_0), \\
y_2 &= c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1), \\
&\;\;\vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
y_n &= c_0 + c_1(x_n - x_0) + c_2(x_n - x_0)(x_n - x_1) + \cdots + c_n(x_n - x_0) \cdots (x_n - x_{n-1}).
\end{aligned}
$$

This system of linear equations is lower triangular as can be seen from the matrix form

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & (x_1 - x_0) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \cdots & (x_n - x_0)\cdots(x_n - x_{n-1}) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

and so theoretically can be solved faster than the Vandermonde polynomial. In practice, however, there is little difference because polynomial interpolation is only useful when the number of points to be interpolated is small.

Piecewise linear interpolation

Instead of constructing a single global polynomial that goes through all the points, one can construct local polynomials that are then connected together. In the the section following this one, we will discuss how this may be done using cubic polynomials. Here, we discuss the simpler case of linear polynomials. This is the default interpolation typically used when plotting data.

Suppose the interpolating function is $y = g(x)$, and as previously, there are $n + 1$ points to interpolate. We construct the function $g(x)$ out of n local linear polynomials. We write

$$g(x) = g_i(x), \quad \text{for } x_i \le x \le x_{i+1},$$

where

$$g_i(x) = a_i(x - x_i) + b_i,$$

and $i = 0, 1, \ldots, n - 1$.

We now require $y = g_i(x)$ to pass through the endpoints (x_i, y_i) and (x_{i+1}, y_{i+1}). We have

$$\begin{aligned} y_i &= b_i, \\ y_{i+1} &= a_i(x_{i+1} - x_i) + b_i. \end{aligned}$$

Therefore, the coefficients of $g_i(x)$ are determined to be

$$a_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad b_i = y_i.$$

Although piecewise linear interpolation is widely used, particularly in plotting routines, it suffers from a discontinuity in the derivative at each point. This results in a function which may not look smooth if the points are too widely spaced. We next consider a more challenging algorithm that uses cubic polynomials.

Cubic spline interpolation

The $n + 1$ points to be interpolated are again

$$(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n).$$

Here, we use n piecewise cubic polynomials for interpolation,

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, \qquad i = 0, 1, \ldots, n - 1,$$

with the global interpolation function written as

$$g(x) = g_i(x), \quad \text{for } x_i \le x \le x_{i+1}.$$

To achieve a smooth interpolation we impose that $g(x)$ and its first and second derivatives are continuous. The requirement that $g(x)$ is continuous (and goes through all $n + 1$ points) results in the two constraints

$$g_i(x_i) = y_i, \quad i = 0 \text{ to } n - 1, \tag{5.1}$$

$$g_i(x_{i+1}) = y_{i+1}, \quad i = 0 \text{ to } n - 1. \tag{5.2}$$

The requirement that $g'(x)$ is continuous results in

$$g_i'(x_{i+1}) = g_{i+1}'(x_{i+1}), \quad i = 0 \text{ to } n - 2. \tag{5.3}$$

And the requirement that $g''(x)$ is continuous results in

$$g_i''(x_{i+1}) = g_{i+1}''(x_{i+1}), \quad i = 0 \text{ to } n - 2. \tag{5.4}$$

There are n cubic polynomials $g_i(x)$ and each cubic polynomial has four free coefficients; there are therefore a total of $4n$ unknown coefficients. The number of constraining equations from (5.1)-(5.4) is $2n + 2(n - 1) = 4n - 2$. With $4n - 2$ constraints and $4n$ unknowns, two more conditions are required for a unique solution. These are usually chosen to be extra conditions on the first $g_0(x)$ and last $g_{n-1}(x)$ polynomials. We will discuss these extra conditions later.

We now proceed to determine equations for the unknown coefficients of the cubic polynomials. The polynomials and their first two derivatives are given by

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, \tag{5.5}$$

$$g_i'(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i, \tag{5.6}$$

$$g_i''(x) = 6a_i(x - x_i) + 2b_i. \tag{5.7}$$

We will consider the four conditions (5.1)-(5.4) in turn. From (5.1) and (5.5), we have

$$d_i = y_i, \quad i = 0 \text{ to } n - 1, \tag{5.8}$$

which directly solves for all of the d-coefficients.

To satisfy (5.2), we first define

$$h_i = x_{i+1} - x_i,$$

and

$$f_i = y_{i+1} - y_i.$$

Now, from (5.2) and (5.5), using (5.8), we obtain the n equations

$$a_i h_i^3 + b_i h_i^2 + c_i h_i = f_i, \qquad i = 0 \text{ to } n-1. \tag{5.9}$$

From (5.3) and (5.6) we obtain the $n-1$ equations

$$3a_i h_i^2 + 2b_i h_i + c_i = c_{i+1}, \qquad i = 0 \text{ to } n-2. \tag{5.10}$$

From (5.4) and (5.7) we obtain the $n-1$ equations

$$3a_i h_i + b_i = b_{i+1} \qquad i = 0 \text{ to } n-2. \tag{5.11}$$

It is will be useful to include a definition of the coefficient b_n, which is now missing. (The index of the cubic polynomial coefficients only go up to $n-1$.) We simply extend (5.11) up to $i = n-1$ and so write

$$3a_{n-1}h_{n-1} + b_{n-1} = b_n, \tag{5.12}$$

which can be viewed as a definition of b_n.

We now proceed to eliminate the sets of a- and c-coefficients (with the d-coefficients already eliminated in (5.8)) to find a system of linear equations for the b-coefficients. From (5.11) and (5.12), we can solve for the n a-coefficients to find

$$a_i = \frac{1}{3h_i}(b_{i+1} - b_i), \qquad i = 0 \text{ to } n-1. \tag{5.13}$$

From (5.9), we can solve for the n c-coefficients as follows:

$$\begin{aligned} c_i &= \frac{1}{h_i}\left(f_i - a_i h_i^3 - b_i h_i^2\right) \\ &= \frac{1}{h_i}\left(f_i - \frac{1}{3h_i}(b_{i+1} - b_i)h_i^3 - b_i h_i^2\right) \\ &= \frac{f_i}{h_i} - \frac{1}{3}h_i(b_{i+1} + 2b_i), \qquad i = 0 \text{ to } n-1. \end{aligned} \tag{5.14}$$

We can now find an equation for the b-coefficients by substituting (5.8), (5.13) and (5.14) into (5.10):

$$3\left(\frac{1}{3h_i}(b_{i+1} - b_i)\right)h_i^2 + 2b_i h_i + \left(\frac{f_i}{h_i} - \frac{1}{3}h_i(b_{i+1} + 2b_i)\right) = \left(\frac{f_{i+1}}{h_{i+1}} - \frac{1}{3}h_{i+1}(b_{i+2} + 2b_{i+1})\right),$$

which simplifies to

$$\frac{1}{3}h_i b_i + \frac{2}{3}(h_i + h_{i+1})b_{i+1} + \frac{1}{3}h_{i+1}b_{i+2} = \frac{f_{i+1}}{h_{i+1}} - \frac{f_i}{h_i}, \tag{5.15}$$

an equation that is valid for $i = 0$ to $n-2$. Therefore, (5.15) represent $n-1$ equations for the $n+1$ unknown b-coefficients. Accordingly, we write the matrix equation for the b-coefficients, leaving the first and last row absent, as

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \text{missing} & \cdots & \cdots \\ \frac{1}{3}h_0 & \frac{2}{3}(h_0+h_1) & \frac{1}{3}h_1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{3}h_{n-2} & \frac{2}{3}(h_{n-2}+h_{n-1}) & \frac{1}{3}h_{n-1} \\ \cdots & \cdots & \cdots & \cdots & \text{missing} & \cdots & \cdots \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix} = \begin{pmatrix} \text{missing} \\ \frac{f_1}{h_1} - \frac{f_0}{h_0} \\ \vdots \\ \frac{f_{n-1}}{h_{n-1}} - \frac{f_{n-2}}{h_{n-2}} \\ \text{missing} \end{pmatrix}.$$

Once the missing first and last equations are specified, the matrix equation for the b-coefficients can be solved by Gaussian elimination. And once the b-coefficients are determined, the a- and c-coefficients can also be determined from (5.13) and (5.14), with the d-coefficients already known from (5.8). The piecewise cubic polynomials, then, are known and $g(x)$ can be used for interpolation to any value x satisfying $x_0 \le x \le x_n$.

The missing first and last equations can be specified in several ways, and here we show the two ways that are allowed by the MATLAB function spline.m. The first way should be used when the derivative $g'(x)$ is known at the endpoints x_0 and x_n; that is, suppose we know the values of α and β such that

$$g_0'(x_0) = \alpha, \quad g_{n-1}'(x_n) = \beta.$$

From the known value of α, and using (5.6) and (5.14), we have

$$\begin{aligned} \alpha &= c_0 \\ &= \frac{f_0}{h_0} - \frac{1}{3}h_0(b_1 + 2b_0). \end{aligned}$$

Therefore, the missing first equation is determined to be

$$\frac{2}{3}h_0 b_0 + \frac{1}{3}h_0 b_1 = \frac{f_0}{h_0} - \alpha. \tag{5.16}$$

From the known value of β, and using (5.6), (5.13), and (5.14), we have

$$\begin{aligned} \beta &= 3a_{n-1}h_{n-1}^2 + 2b_{n-1}h_{n-1} + c_{n-1} \\ &= 3\left(\frac{1}{3h_{n-1}}(b_n - b_{n-1})\right)h_{n-1}^2 + 2b_{n-1}h_{n-1} + \left(\frac{f_{n-1}}{h_{n-1}} - \frac{1}{3}h_{n-1}(b_n + 2b_{n-1})\right), \end{aligned}$$

which simplifies to

$$\frac{1}{3}h_{n-1}b_{n-1} + \frac{2}{3}h_{n-1}b_n = \beta - \frac{f_{n-1}}{h_{n-1}}, \tag{5.17}$$

3 3 h_{n-1}

to be used as the missing last equation.

The second way of specifying the missing first and last equations is called the *not-a-knot* condition, which assumes that

$$g_0(x) = g_1(x), \quad g_{n-2}(x) = g_{n-1}(x).$$

Considering the first of these equations, from (5.5) we have

$$a_0(x - x_0)^3 + b_0(x - x_0)^2 + c_0(x - x_0) + d_0$$
$$= a_1(x - x_1)^3 + b_1(x - x_1)^2 + c_1(x - x_1) + d_1.$$

Now two cubic polynomials can be proven to be identical if at some value of x, the polynomials and their first three derivatives are identical. Our conditions of continuity at $x = x_1$ already require that at this value of x these two polynomials and their first two derivatives are identical. The polynomials themselves will be identical, then, if their third derivatives are also identical at $x = x_1$, or if

$$a_0 = a_1.$$

From (5.13), we have

$$\frac{1}{3h_0}(b_1 - b_0) = \frac{1}{3h_1}(b_2 - b_1),$$

or after simplification

$$h_1 b_0 - (h_0 + h_1)b_1 + h_0 b_2 = 0, \tag{5.18}$$

which provides us our missing first equation. A similar argument at $x = x_{n-1}$ also provides us with our last equation,

$$h_{n-1}b_{n-2} - (h_{n-2} + h_{n-1})b_{n-1} + h_{n-2}b_n = 0. \tag{5.19}$$

The MATLAB subroutines spline.m and ppval.m can be used for cubic spline interpolation (see also interp1.m). I will illustrate these routines in class and post sample code on the course web site.

Multidimensional interpolation

Suppose we are interpolating the value of a function of two variables,

$$z = f(x, y).$$

The known values are given by

$$z_{ij} = f(x_i, y_j),$$

with $i = 0, 1, \ldots, n$ and $j = 0, 1, \ldots, n$. Note that the (x, y) points at which $f(x, y)$ are known lie on a grid in the $x - y$ plane.

Let $z = g(x, y)$ be the interpolating function, satisfying $z_{ij} = g(x_i, y_j)$. A two-dimensional interpolation to find the value of g at the point (x, y) may be done by first performing $n + 1$ one-dimensional interpolations in y to find the value of g at the $n + 1$ points $(x_0, y), (x_1, y), \ldots, (x_n, y)$, followed by a single one-dimensional interpolation in x to find the value of g at (x, y).

In other words, two-dimensional interpolation on a grid of dimension $(n + 1) \times (n + 1)$ is done by first performing $n + 1$ one-dimensional interpolations to the value y followed by a single one-dimensional interpolation to the value x. Two-dimensional interpolation can be generalized to higher dimensions. The MATLAB functions to perform two- and three-dimensional interpolation are interp2.m and interp3.m.

Chapter 6

Integration

We want to construct numerical algorithms that can perform definite integrals of the form

$$I = \int_a^b f(x)dx. \tag{6.1}$$

Calculating these definite integrals numerically is called *numerical integration*, *numerical quadrature*, or more simply *quadrature*.

Elementary formulas

We first consider integration from 0 to h, with h small, to serve as the building blocks for integration over larger domains. We here define I_h as the following integral:

$$I_h = \int_0^h f(x)dx. \tag{6.2}$$

To perform this integral, we consider a Taylor series expansion of $f(x)$ about the value $x = h/2$:

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

Midpoint rule

The midpoint rule makes use of only the first term in the Taylor series expansion. Here, we will determine the error in this approximation. Integrating,

$$\begin{aligned} I_h = hf(h/2) + \int_0^h \Big(&(x - h/2)f'(h/2) + \frac{(x-h/2)^2}{2}f''(h/2) \\ &+ \frac{(x-h/2)^3}{6}f'''(h/2) + \frac{(x-h/2)^4}{24}f''''(h/2) + \ldots \Big) dx. \end{aligned}$$

Changing variables by letting $y = x - h/2$ and $dy = dx$, and simplifying the integral depending on whether the integrand is even or odd, we have

$$\begin{aligned} I_h &= hf(h/2) \\ &\quad + \int_{-h/2}^{h/2} \Big(yf'(h/2) + \frac{y^2}{2}f''(h/2) + \frac{y^3}{6}f'''(h/2) + \frac{y^4}{24}f''''(h/2) + \ldots \Big) dy \\ &= hf(h/2) + \int_0^{h/2} \Big(y^2 f''(h/2) + \frac{y^4}{12}f''''(h/2) + \ldots \Big) dy. \end{aligned}$$

The integrals that we need here are

$$\int_0^{\frac{h}{2}} y^2 dy = \frac{h^3}{24}, \quad \int_0^{\frac{h}{2}} y^4 dy = \frac{h^5}{160}.$$

Therefore,

$$I_h = hf(h/2) + \frac{h^3}{24}f''(h/2) + \frac{h^5}{1920}f''''(h/2) + \dots. \tag{6.3}$$

Trapezoidal rule

From the Taylor series expansion of $f(x)$ about $x = h/2$, we have

$$f(0) = f(h/2) - \frac{h}{2}f'(h/2) + \frac{h^2}{8}f''(h/2) - \frac{h^3}{48}f'''(h/2) + \frac{h^4}{384}f''''(h/2) + \dots,$$

and

$$f(h) = f(h/2) + \frac{h}{2}f'(h/2) + \frac{h^2}{8}f''(h/2) + \frac{h^3}{48}f'''(h/2) + \frac{h^4}{384}f''''(h/2) + \dots.$$

Adding and multiplying by $h/2$ we obtain

$$\frac{h}{2}\left(f(0) + f(h)\right) = hf(h/2) + \frac{h^3}{8}f''(h/2) + \frac{h^5}{384}f''''(h/2) + \dots.$$

We now substitute for the first term on the right-hand-side using the midpoint rule formula:

$$\begin{aligned}\frac{h}{2}\left(f(0) + f(h)\right) = &\left(I_h - \frac{h^3}{24}f''(h/2) - \frac{h^5}{1920}f''''(h/2)\right) \\ &+ \frac{h^3}{8}f''(h/2) + \frac{h^5}{384}f''''(h/2) + \dots,\end{aligned}$$

and solving for I_h, we find

$$I_h = \frac{h}{2}\left(f(0) + f(h)\right) - \frac{h^3}{12}f''(h/2) - \frac{h^5}{480}f''''(h/2) + \dots. \tag{6.4}$$

Simpson's rule

To obtain Simpson's rule, we combine the midpoint and trapezoidal rule to eliminate the error term proportional to h^3. Multiplying (6.3) by two and adding to (6.4), we obtain

$$3I_h = h\left(2f(h/2) + \frac{1}{2}(f(0) + f(h))\right) + h^5\left(\frac{2}{1920} - \frac{1}{480}\right)f''''(h/2) + \dots,$$

or

$$I_h = \frac{h}{6}\left(f(0) + 4f(h/2) + f(h)\right) - \frac{h^5}{2880}f''''(h/2) + \dots.$$

Usually, Simpson's rule is written by considering the three consecutive points 0, h and $2h$. Substituting $h \to 2h$, we obtain the standard result

$$I_{2h} = \frac{h}{3}\left(f(0) + 4f(h) + f(2h)\right) - \frac{h^5}{90}f''''(h) + \dots. \tag{6.5}$$

6.2 Composite rules

We now use our elementary formulas obtained for (6.2) to perform the integral given by (6.1).

Trapezoidal rule

We suppose that the function $f(x)$ is known at the $n+1$ points labeled as $x_0, x_1, \ldots, x_n$, with the endpoints given by $x_0 = a$ and $x_n = b$. Define

$$f_i = f(x_i), \quad h_i = x_{i+1} - x_i.$$

Then the integral of (6.1) may be decomposed as

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx \\ &= \sum_{i=0}^{n-1} \int_0^{h_i} f(x_i + s)ds,\end{aligned}$$

where the last equality arises from the change-of-variables $s = x - x_i$. Applying the trapezoidal rule to the integral, we have

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \frac{h_i}{2}\left(f_i + f_{i+1}\right). \tag{6.6}$$

If the points are not evenly spaced, say because the data are experimental values, then the h_i may differ for each value of i and (6.6) is to be used directly.

However, if the points are evenly spaced, say because $f(x)$ can be computed, we have $h_i = h$, independent of i. We can then define

$$x_i = a + ih, \quad i = 0, 1, \ldots, n;$$

and since the end point b satisfies $b = a + nh$, we have

$$h = \frac{b-a}{n}.$$

The composite trapezoidal rule for evenly space points then becomes

$$\begin{aligned}\int_a^b f(x)dx &= \frac{h}{2}\sum_{i=0}^{n-1}\left(f_i + f_{i+1}\right) \\ &= \frac{h}{2}\left(f_0 + 2f_1 + \cdots + 2f_{n-1} + f_n\right). \end{aligned} \tag{6.7}$$

The first and last terms have a multiple of one; all other terms have a multiple of two; and the entire sum is multiplied by $h/2$.

Simpson's rule

We here consider the composite Simpson's rule for evenly space points. We apply Simpson's rule over intervals of $2h$, starting from a and ending at b:

$$\int_a^b f(x)dx = \frac{h}{3}\left(f_0 + 4f_1 + f_2\right) + \frac{h}{3}\left(f_2 + 4f_3 + f_4\right) + \ldots + \frac{h}{3}\left(f_{n-2} + 4f_{n-1} + f_n\right).$$

Note that n must be even for this scheme to work. Combining terms, we have

$$\int_a^b f(x)dx = \frac{h}{3}\left(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 4f_{n-1} + f_n\right).$$

The first and last terms have a multiple of one; the even indexed terms have a multiple of 2; the odd indexed terms have a multiple of 4; and the entire sum is multiplied by $h/3$.

Local versus global error

Consider the elementary formula (6.4) for the trapezoidal rule, written in the form

$$\int_0^h f(x)dx = \frac{h}{2}\left(f(0) + f(h)\right) - \frac{h^3}{12}f''(\xi),$$

where ξ is some value satisfying $0 \leq \xi \leq h$, and we have used Taylor's theorem with the mean-value form of the remainder. We can also represent the remainder as

$$-\frac{h^3}{12}f''(\xi) = O(h^3),$$

where $O(h^3)$ signifies that when h is small, halving of the grid spacing h decreases the error in the elementary trapezoidal rule by a factor of eight. We call the terms represented by $O(h^3)$ the *Local Error*.

More important is the *Global Error* which is obtained from the composite formula (6.7) for the trapezoidal rule. Putting in the remainder terms, we have

$$\int_a^b f(x)dx = \frac{h}{2}\left(f_0 + 2f_1 + \cdots + 2f_{n-1} + f_n\right) - \frac{h^3}{12}\sum_{i=0}^{n-1} f''(\xi_i),$$

where ξ_i are values satisfying $x_i \leq \xi_i \leq x_{i+1}$. The remainder can be rewritten as

$$-\frac{h^3}{12}\sum_{i=0}^{n-1} f''(\xi_i) = -\frac{nh^3}{12}\left\langle f''(\xi_i)\right\rangle,$$

where $\left\langle f''(\xi_i)\right\rangle$ is the average value of all the $f''(\xi_i)$'s. Now,

$$n = \frac{b-a}{h},$$

so that the error term becomes

$$\begin{aligned} -\frac{nh^3}{12}\left\langle f''(\xi_i)\right\rangle &= -\frac{(b-a)h^2}{12}\left\langle f''(\xi_i)\right\rangle \\ &= O(h^2). \end{aligned}$$

Therefore, the global error is $O(h^2)$. That is, a halving of the grid spacing only decreases the global error by a factor of four.

Similarly, Simpson's rule has a local error of $O(h^5)$ and a global error of $O(h^4)$.

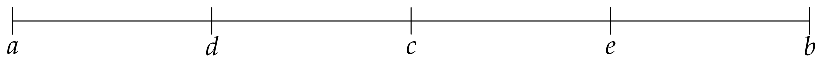


Figure 6.1: Adaptive Simpson quadrature: Level 1.

Adaptive integration

The useful MATLAB function quad.m performs numerical integration using adaptive Simpson quadrature. The idea is to let the computation itself decide on the grid size required to achieve a certain level of accuracy. Moreover, the grid size need not be the same over the entire region of integration.

We begin the adaptive integration at what is called Level 1. The uniformly spaced points at which the function $f(x)$ is to be evaluated are shown in Fig.6.1. The distance between the points a and b is taken to be $2h$, so that

$$h = \frac{b-a}{2}.$$

Integration using Simpson's rule (6.5) with grid size h yields

$$I = \frac{h}{3}\left(f(a) + 4f(c) + f(b)\right) - \frac{h^5}{90} f''''(\xi),$$

where ξ is some value satisfying $a \le \xi \le b$.

Integration using Simpson's rule twice with grid size $h/2$ yields

$$I = \frac{h}{6}\left(f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)\right) - \frac{(h/2)^5}{90} f''''(\xi_l) - \frac{(h/2)^5}{90} f''''(\xi_r),$$

with ξ_l and ξ_r some values satisfying $a \le \xi_l \le c$ and $c \le \xi_r \le b$.

We now define

$$S_1 = \frac{h}{3}\left(f(a) + 4f(c) + f(b)\right),$$
$$S_2 = \frac{h}{6}\left(f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)\right),$$
$$E_1 = -\frac{h^5}{90} f''''(\xi),$$
$$E_2 = -\frac{h^5}{2^5 \cdot 90}\left(f''''(\xi_l) + f''''(\xi_r)\right).$$

Now we ask whether S_2 is accurate enough, or must we further refine the calculation and go to Level 2? To answer this question, we make the simplifying approximation that all of the fourth-order derivatives of $f(x)$ in the error terms are equal; that is,

$$f''''(\xi) = f''''(\xi_l) = f''''(\xi_r) = C.$$

Then

$$E_1 = -\frac{h^5}{90} C,$$
$$E_2 = -\frac{h^5}{2^4 \cdot 90} C = \frac{1}{16} E_1.$$

Then since

$$S_1 + E_1 = S_2 + E_2,$$

and

$$E_1 = 16 E_2,$$

we have for our estimate for the error term E_2,

$$E_2 = \frac{1}{15}(S_2 - S_1).$$

Therefore, given some specific value of the tolerance tol, if

$$\left|\frac{1}{15}(S_2 - S_1)\right| < \text{tol},$$

then we can accept S_2 as I. If the tolerance is not achieved for I, then we proceed to Level 2.

The computation at Level 2 further divides the integration interval from a to b into the two integration intervals a to c and c to b, and proceeds with the above procedure independently on both halves. Integration can be stopped on either half provided the tolerance is less than tol/2 (since the sum of both errors must be less than tol). Otherwise, either half can proceed to Level 3, and so on.

As a side note, the two values of I given above (for integration with step size h and $h/2$) can be combined to give a more accurate value for I given by

$$I = \frac{16 S_2 - S_1}{15} + O(h^7),$$

where the error terms of $O(h^5)$ approximately cancel. This free lunch, so to speak, is called Richardson's extrapolation.

Chapter 7

Ordinary differential equations

We now discuss the numerical solution of ordinary differential equations. These include the initial value problem, the boundary value problem, and the eigenvalue problem. Before proceeding to the development of numerical methods, we review the analytical solution of some classic equations.

Examples of analytical solutions

Initial value problem

One classic initial value problem is the *RC* circuit. With *R* the resistor and *C* the capacitor, the differential equation for the charge *q* on the capacitor is given by

$$R\frac{dq}{dt} + \frac{q}{C} = 0. \tag{7.1}$$

If we consider the physical problem of a charged capacitor connected in a closed circuit to a resistor, then the initial condition is $q(0) = q_0$, where q_0 is the initial charge on the capacitor.

The differential equation (7.1) is separable, and separating and integrating from time $t = 0$ to t yields

$$\int_{q_0}^{q} \frac{dq}{q} = -\frac{1}{RC}\int_0^t dt,$$

which can be integrated and solved for $q = q(t)$:

$$q(t) = q_0 e^{-t/RC}.$$

The classic second-order initial value problem is the *RLC* circuit, with differential equation

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} = 0.$$

Here, a charged capacitor is connected to a closed circuit, and the initial conditions satisfy

$$q(0) = q_0, \quad \frac{dq}{dt}(0) = 0.$$

The solution is obtained for the second-order equation by the ansatz

$$q(t) = e^{rt},$$

which results in the following so-called characteristic equation for *r*:

$$Lr^2 + Rr + \frac{1}{C} = 0.$$

If the two solutions for *r* are distinct and real, then the two found exponential solutions can be multiplied by constants and added to form a general solution. The constants can then be determined by requiring the general solution to satisfy the two initial conditions. If the roots of the characteristic equation are complex or degenerate, a general solution to the differential equation can also be found.

Boundary value problems

The dimensionless equation for the temperature $y = y(x)$ along a linear heat-conducting rod of length unity, and with an applied external heat source $f(x)$, is given by the differential equation

$$-\frac{d^2y}{dx^2} = f(x), \tag{7.2}$$

with $0 \le x \le 1$. Boundary conditions are usually prescribed at the end points of the rod, and here we assume that the temperature at both ends are maintained at zero so that

$$y(0) = 0, \quad y(1) = 0.$$

The assignment of boundary conditions at two separate points is called a two-point boundary value problem, in contrast to the initial value problem where the boundary conditions are prescribed at only a single point. Two-point boundary value problems typically require a more sophisticated algorithm for a numerical solution than initial value problems.

Here, the solution of (7.2) can proceed by integration once $f(x)$ is specified. We assume that

$$f(x) = x(1 - x),$$

so that the maximum of the heat source occurs in the center of the rod, and goes to zero at the ends.

The differential equation can then be written as

$$\frac{d^2y}{dx^2} = -x(1 - x).$$

The first integration results in

$$\begin{aligned}\frac{dy}{dx} &= \int (x^2 - x)dx \\ &= \frac{x^3}{3} - \frac{x^2}{2} + c_1,\end{aligned}$$

where c_1 is the first integration constant. Integrating again,

$$\begin{aligned}y(x) &= \int \left(\frac{x^3}{3} - \frac{x^2}{2} + c_1\right) dx \\ &= \frac{x^4}{12} - \frac{x^3}{6} + c_1 x + c_2,\end{aligned}$$

where c_2 is the second integration constant. The two integration constants are determined by the boundary conditions. At $x = 0$, we have

$$0 = c_2,$$

and at $x = 1$, we have

$$0 = \frac{1}{12} - \frac{1}{6} + c_1,$$

so that $c_1 = 1/12$. Our solution is therefore

$$\begin{aligned}y(x) &= \frac{x^4}{12} - \frac{x^3}{6} + \frac{x}{12} \\ &= \frac{1}{12}x(1 - x)(1 + x - x^2).\end{aligned}$$

The temperature of the rod is maximum at $x = 1/2$ and goes smoothly to zero at the ends.

Eigenvalue problem

The classic eigenvalue problem obtained by solving the wave equation by separation of variables is given by

$$\frac{d^2y}{dx^2} + \lambda^2 y = 0,$$

with the two-point boundary conditions $y(0) = 0$ and $y(1) = 0$. Notice that $y(x) = 0$ satisfies both the differential equation and the boundary conditions. Other nonzero solutions for $y = y(x)$ are possible only for certain discrete values of λ. These values of λ are called the eigenvalues of the differential equation.

We proceed by first finding the general solution to the differential equation. It is easy to see that this solution is

$$y(x) = A \cos \lambda x + B \sin \lambda x.$$

Imposing the first boundary condition at $x = 0$, we obtain

$$A = 0.$$

The second boundary condition at $x = 1$ results in

$$B \sin \lambda = 0.$$

Since we are searching for a solution where $y = y(x)$ is not identically zero, we must have

$$\lambda = \pi, 2\pi, 3\pi, \ldots.$$

The corresponding negative values of λ are also solutions, but their inclusion only changes the corresponding values of the unknown B constant. A linear superposition of all the solutions results in the general solution

$$y(x) = \sum_{n=1}^{\infty} B_n \sin n\pi x.$$

For each eigenvalue $n\pi$, we say there is a corresponding eigenfunction $\sin n\pi x$. When the differential equation can not be solved analytically, a numerical method should be able to solve for both the eigenvalues and eigenfunctions.

Numerical methods: initial value problem

We begin with the simple Euler method, then discuss the more sophisticated Runge-Kutta methods, and conclude with the Runge-Kutta-Fehlberg method, as implemented in the MATLAB function ode45.m. Our differential equations are for $x = x(t)$, where the time t is the independent variable, and we will make use of the notation $\dot{x} = dx/dt$. This notation is still widely used by physicists and descends directly from the notation originally used by Newton.

Euler method

The Euler method is the most straightforward method to integrate a differential equation. Consider the first-order differential equation

$$\dot{x} = f(t, x), \tag{7.3}$$

with the initial condition $x(0) = x_0$. Define $t_n = n\Delta t$ and $x_n = x(t_n)$. A Taylor series expansion of x_{n+1} results in

$$\begin{aligned} x_{n+1} &= x(t_n + \Delta t) \\ &= x(t_n) + \Delta t \dot{x}(t_n) + O(\Delta t^2) \\ &= x(t_n) + \Delta t f(t_n, x_n) + O(\Delta t^2). \end{aligned}$$

The Euler Method is therefore written as

$$x_{n+1} = x(t_n) + \Delta t f(t_n, x_n).$$

We say that the Euler method steps forward in time using a time-step Δt, starting from the initial value $x_0 = x(0)$. The local error of the Euler Method is $O(\Delta t^2)$. The global error, however, incurred when integrating to a time T, is a factor of $1/\Delta t$ larger and is given by $O(\Delta t)$. It is therefore customary to call the Euler Method a *first-order method*.

Modified Euler method

This method is of a type that is called a predictor-corrector method. It is also the first of what are Runge-Kutta methods. As before, we want to solve (7.3). The idea is to average the value of $\dot{x}$ at the beginning and end of the time step. That is, we would like to modify the Euler method and write

$$x_{n+1} = x_n + \frac{1}{2}\Delta t \left(f(t_n, x_n) + f(t_n + \Delta t, x_{n+1}) \right).$$

The obvious problem with this formula is that the unknown value x_{n+1} appears on the right-hand-side. We can, however, estimate this value, in what is called the predictor step. For the predictor step, we use the Euler method to find

$$x^p_{n+1} = x_n + \Delta t f(t_n, x_n).$$

The corrector step then becomes

$$x_{n+1} = x_n + \frac{1}{2}\Delta t \left(f(t_n, x_n) + f(t_n + \Delta t, x^p_{n+1}) \right).$$

The Modified Euler Method can be rewritten in the following form that we will later identify as a Runge-Kutta method:

$$\begin{aligned} k_1 &= \Delta t f(t_n, x_n), \\ k_2 &= \Delta t f(t_n + \Delta t, x_n + k_1), \\ x_{n+1} &= x_n + \frac{1}{2}(k_1 + k_2). \end{aligned} \tag{7.4}$$

Second-order Runge-Kutta methods

We now derive all second-order Runge-Kutta methods. Higher-order methods can be similarly derived, but require substantially more algebra.

We consider the differential equation given by (7.3). A general second-order Runge-Kutta method may be written in the form

$$\begin{aligned} k_1 &= \Delta t f(t_n, x_n), \\ k_2 &= \Delta t f(t_n + \alpha\Delta t, x_n + \beta k_1), \\ x_{n+1} &= x_n + ak_1 + bk_2, \end{aligned} \tag{7.5}$$

with α, β, a and b constants that define the particular second-order Runge-Kutta method. These constants are to be constrained by setting the local error of the second-order Runge-Kutta method to be $O(\Delta t^3)$. Intuitively, we might guess that two of the constraints will be $a + b = 1$ and $\alpha = \beta$.

We compute the Taylor series of x_{n+1} directly, and from the Runge-Kutta method, and require them to be the same to order Δt^2. First, we compute the Taylor series of x_{n+1}. We have

$$\begin{aligned} x_{n+1} &= x(t_n + \Delta t) \\ &= x(t_n) + \Delta t \dot{x}(t_n) + \frac{1}{2}(\Delta t)^2 \ddot{x}(t_n) + O(\Delta t^3). \end{aligned}$$

Now,

$$\dot{x}(t_n) = f(t_n, x_n).$$

The second derivative is more complicated and requires partial derivatives. We have

$$\begin{aligned} \ddot{x}(t_n) &= \left.\frac{d}{dt} f(t, x(t))\right]_{t=t_n} \\ &= f_t(t_n, x_n) + \dot{x}(t_n) f_x(t_n, x_n) \\ &= f_t(t_n, x_n) + f(t_n, x_n) f_x(t_n, x_n). \end{aligned}$$

Therefore,

$$x_{n+1} = x_n + \Delta t f(t_n, x_n) + \frac{1}{2}(\Delta t)^2 \left(f_t(t_n, x_n) + f(t_n, x_n) f_x(t_n, x_n) \right). \tag{7.6}$$

Second, we compute x_{n+1} from the Runge-Kutta method given by (7.5). Substituting in k_1 and k_2, we have

$$x_{n+1} = x_n + a\Delta t f(t_n, x_n) + b\Delta t f\left(t_n + \alpha\Delta t, x_n + \beta\Delta t f(t_n, x_n)\right).$$

We Taylor series expand using

$$\begin{aligned} & f\left(t_n + \alpha\Delta t, x_n + \beta\Delta t f(t_n, x_n)\right) \\ &\qquad = f(t_n, x_n) + \alpha\Delta t f_t(t_n, x_n) + \beta\Delta t f(t_n, x_n) f_x(t_n, x_n) + O(\Delta t^2). \end{aligned}$$

The Runge-Kutta formula is therefore

$$\begin{aligned} x_{n+1} &= x_n + (a + b)\Delta t f(t_n, x_n) \\ &\qquad + (\Delta t)^2 \left(\alpha b f_t(t_n, x_n) + \beta b f(t_n, x_n) f_x(t_n, x_n) \right) + O(\Delta t^3). \end{aligned} \tag{7.7}$$

Comparing (7.6) and (7.7), we find

$$a + b = 1,$$
$$\alpha b = 1/2,$$
$$\beta b = 1/2.$$

There are three equations for four parameters, and there exists a family of second-order Runge-Kutta methods.

The Modified Euler Method given by (7.4) corresponds to $\alpha = \beta = 1$ and $a = b = 1/2$. Another second-order Runge-Kutta method, called the Midpoint Method, corresponds to $\alpha = \beta = 1/2$, $a = 0$ and $b = 1$. This method is written as

$$k_1 = \Delta t f(t_n, x_n),$$
$$k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right),$$
$$x_{n+1} = x_n + k_2.$$

Higher-order Runge-Kutta methods

The general second-order Runge-Kutta method was given by (7.5). The general form of the third-order method is given by

$$k_1 = \Delta t f(t_n, x_n),$$
$$k_2 = \Delta t f(t_n + \alpha\Delta t, x_n + \beta k_1),$$
$$k_3 = \Delta t f(t_n + \gamma\Delta t, x_n + \delta k_1 + \epsilon k_2),$$
$$x_{n+1} = x_n + ak_1 + bk_2 + ck_3.$$

The following constraints on the constants can be guessed: $\alpha = \beta$, $\gamma = \delta + \epsilon$, and $a + b + c = 1$. Remaining constraints need to be derived.

The fourth-order method has a k_1, k_2, k_3 and k_4. The fifth-order method requires up to k_6. The table below gives the order of the method and the number of stages required.

order	2	3	4	5	6	7	8
minimum # stages	2	3	4	6	7	9	11

Because of the jump in the number of stages required between the fourth-order and fifth-order method, the fourth-order Runge-Kutta method has some appeal. The general fourth-order method starts with 13 constants, and one then finds 11 constraints. A particularly simple fourth-order method that has been widely used is given by

$$k_1 = \Delta t f(t_n, x_n),$$
$$k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right),$$
$$k_3 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_2\right),$$
$$k_4 = \Delta t f(t_n + \Delta t, x_n + k_3),$$
$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Adaptive Runge-Kutta Methods

As in adaptive integration, it is useful to devise an ode integrator that automatically finds the appropriate Δt. The Dormand-Prince Method, which is implemented in MATLAB's ode45.m, finds the appropriate step size by comparing the results of a fifth-order and fourth-order method. It requires six function evaluations per time step, and constructs both a fifth-order and a fourth-order method from these function evaluations.

Suppose the fifth-order method finds x_{n+1} with local error $O(\Delta t^6)$, and the fourth-order method finds x'_{n+1} with local error $O(\Delta t^5)$. Let ε be the desired error tolerance of the method, and let e be the actual error. We can estimate e from the difference between the fifth- and fourth-order methods; that is,

$$e = |x_{n+1} - x'_{n+1}|.$$

Now e is of $O(\Delta t^5)$, where Δt is the step size taken. Let $\Delta\tau$ be the estimated step size required to get the desired error ε. Then we have

$$e/\varepsilon = (\Delta t)^5/(\Delta\tau)^5,$$

or solving for $\Delta\tau$,

$$\Delta\tau = \Delta t \left(\frac{\varepsilon}{e}\right)^{1/5}.$$

On the one hand, if $e < \varepsilon$, then we accept x_{n+1} and do the next time step using the larger value of $\Delta\tau$. On the other hand, if $e > \varepsilon$, then we reject the integration step and redo the time step using the smaller value of $\Delta\tau$. In practice, one usually increases the time step slightly less and decreases the time step slightly more to prevent the waste of too many failed time steps.

System of differential equations

Our numerical methods can be easily adapted to solve higher-order differential equations, or equivalently, a system of differential equations. First, we show how a second-order differential equation can be reduced to two first-order equations. Consider

$$\ddot{x} = f(t, x, \dot{x}).$$

This second-order equation can be rewritten as two first-order equations by defining $u = \dot{x}$. We then have the system

$$\begin{aligned}\dot{x} &= u,\\ \dot{u} &= f(t, x, u).\end{aligned}$$

This trick also works for higher-order equation. For another example, the third-order equation

$$\dddot{x} = f(t, x, \dot{x}, \ddot{x}),$$

can be written as

$$\begin{aligned}\dot{x} &= u,\\ \dot{u} &= v,\\ \dot{v} &= f(t, x, u, v).\end{aligned}$$

Now, we show how to generalize Runge-Kutta methods to a system of differential equations. As an example, consider the following system of two odes,

$$
\begin{aligned}
\dot{x} &= f(t, x, y), \\
\dot{y} &= g(t, x, y),
\end{aligned}
$$

with the initial conditions $x(0) = x_0$ and $y(0) = y_0$. The generalization of the commonly used fourth-order Runge-Kutta method would be

$$
\begin{aligned}
k_1 &= \Delta t f(t_n, x_n, y_n), \\
l_1 &= \Delta t g(t_n, x_n, y_n),
\end{aligned}
$$

$$
\begin{aligned}
k_2 &= \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1, y_n + \frac{1}{2}l_1\right), \\
l_2 &= \Delta t g\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1, y_n + \frac{1}{2}l_1\right),
\end{aligned}
$$

$$
\begin{aligned}
k_3 &= \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_2, y_n + \frac{1}{2}l_2\right), \\
l_3 &= \Delta t g\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_2, y_n + \frac{1}{2}l_2\right),
\end{aligned}
$$

$$
\begin{aligned}
k_4 &= \Delta t f(t_n + \Delta t, x_n + k_3, y_n + l_3), \\
l_4 &= \Delta t g(t_n + \Delta t, x_n + k_3, y_n + l_3),
\end{aligned}
$$

$$
\begin{aligned}
x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
y_{n+1} &= y_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4).
\end{aligned}
$$

Numerical methods: boundary value problem

Finite difference method

We consider first the differential equation

$$
-\frac{d^2y}{dx^2} = f(x), \quad 0 \le x \le 1, \tag{7.8}
$$

with two-point boundary conditions

$$
y(0) = A, \quad y(1) = B.
$$

Equation (7.8) can be solved by quadrature, but here we will demonstrate a numerical solution using a finite difference method.

We begin by discussing how to numerically approximate derivatives. Consider the Taylor series approximation for $y(x + h)$ and $y(x - h)$, given by

$$
\begin{aligned}
y(x + h) &= y(x) + hy'(x) + \frac{1}{2}h^2y''(x) + \frac{1}{6}h^3y'''(x) + \frac{1}{24}h^4y''''(x) + \dots, \\
y(x - h) &= y(x) - hy'(x) + \frac{1}{2}h^2y''(x) - \frac{1}{6}h^3y'''(x) + \frac{1}{24}h^4y''''(x) + \dots.
\end{aligned}
$$

The standard definitions of the derivatives give the first-order approximations

$$y'(x) = \frac{y(x+h) - y(x)}{h} + O(h),$$
$$y'(x) = \frac{y(x) - y(x-h)}{h} + O(h).$$

The more widely-used second-order approximation is called the central difference approximation and is given by

$$y'(x) = \frac{y(x+h) - y(x-h)}{2h} + O(h^2).$$

The finite difference approximation to the second derivative can be found from considering

$$y(x+h) + y(x-h) = 2y(x) + h^2 y''(x) + \frac{1}{12} h^4 y''''(x) + \ldots,$$

from which we find

$$y''(x) = \frac{y(x+h) - 2y(x) + y(x-h)}{h^2} + O(h^2).$$

Sometimes a second-order method is required for x on the boundaries of the domain. For a boundary point on the left, a second-order forward difference method requires the additional Taylor series

$$y(x+2h) = y(x) + 2hy'(x) + 2h^2 y''(x) + \frac{4}{3} h^3 y'''(x) + \ldots.$$

We combine the Taylor series for $y(x+h)$ and $y(x+2h)$ to eliminate the term proportional to h^2:

$$y(x+2h) - 4y(x+h) = -3y(x) - 2hy'(x) + O(h^3).$$

Therefore,

$$y'(x) = \frac{-3y(x) + 4y(x+h) - y(x+2h)}{2h} + O(h^2).$$

For a boundary point on the right, we send $h \to -h$ to find

$$y'(x) = \frac{3y(x) - 4y(x-h) + y(x-2h)}{2h} + O(h^2).$$

We now write a finite difference scheme to solve (7.8). We discretize x by defining $x_i = ih$, $i = 0, 1, \ldots, n+1$. Since $x_{n+1} = 1$, we have $h = 1/(n+1)$. The functions $y(x)$ and $f(x)$ are discretized as $y_i = y(x_i)$ and $f_i = f(x_i)$. The second-order differential equation (7.8) then becomes for the interior points of the domain

$$-y_{i-1} + 2y_i - y_{i+1} = h^2 f_i, \qquad i = 1, 2, \ldots n,$$

with the boundary conditions $y_0 = A$ and $y_{n+1} = B$. We therefore have a linear system of equations to solve. The first and nth equation contain the boundary conditions and are given by

$$2y_1 - y_2 = h^2 f_1 + A,$$
$$-y_{n-1} + 2y_n = h^2 f_n + B.$$

The second and third equations, etc., are

$$
\begin{aligned}
-y_1 + 2y_2 - y_3 &= h^2 f_2, \\
-y_2 + 2y_3 - y_4 &= h^2 f_3, \\
&\dots
\end{aligned}
$$

In matrix form, we have

$$
\begin{pmatrix}
2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\
-1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\
0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\
0 & 0 & 0 & 0 & \dots & 0 & -1 & 2
\end{pmatrix}
\begin{pmatrix}
y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n
\end{pmatrix}
=
\begin{pmatrix}
h^2 f_1 + A \\ h^2 f_2 \\ h^2 f_3 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n + B
\end{pmatrix}.
$$

The matrix is tridiagonal, and a numerical solution by Guassian elimination can be done quickly. The matrix itself is easily constructed using the MATLAB function diag.m and ones.m. As excerpted from the MATLAB help page, the function call ones(m,n) returns an m-by-n matrix of ones, and the function call diag(v,k), when v is a vector with n components, is a square matrix of order n+abs(k) with the elements of v on the k-th diagonal: $k = 0$ is the main diagonal, $k > 0$ is above the main diagonal and $k < 0$ is below the main diagonal. The $n \times n$ matrix above can be constructed by the MATLAB code

```
M=diag(-ones(n-1,1),-1)+diag(2*ones(n,1),0)+diag(-ones(n-1,1),1); .
```

The right-hand-side, provided f is a given n-by-1 vector, can be constructed by the MATLAB code

```
b=h^2*f; b(1)=b(1)+A; b(n)=b(n)+B;
```

and the solution for u is given by the MATLAB code

```
y=M\b;
```

Shooting method

The finite difference method can solve linear odes. For a general ode of the form

$$
\frac{d^2y}{dx^2} = f(x, y, dy/dx),
$$

with $y(0) = A$ and $y(1) = B$, we use a shooting method. First, we formulate the ode as an initial value problem. We have

$$\frac{dy}{dx} = z,$$
$$\frac{dz}{dx} = f(x, y, z).$$

The initial condition $y(0) = A$ is known, but the second initial condition $z(0) = b$ is unknown. Our goal is to determine b such that $y(1) = B$.

In fact, this is a root-finding problem for an appropriately defined function. We define the function $F = F(b)$ such that

$$F(b) = y(1) - B.$$

In other words, $F(b)$ is the difference between the value of $y(1)$ obtained from integrating the differential equations using the initial condition $z(0) = b$, and B. Our root-finding routine will want to solve $F(b) = 0$. (The method is called *shooting* because the slope of the solution curve for $y = y(x)$ at $x = 0$ is given by b, and the solution hits the value $y(1)$ at $x = 1$. This looks like pointing a gun and trying to shoot the target, which is B.)

To determine the value of b that solves $F(b) = 0$, we iterate using the Secant method, given by

$$b_{n+1} = b_n - F(b_n)\frac{b_n - b_{n-1}}{F(b_n) - F(b_{n-1})}.$$

We need to start with two initial guesses for b, solving the ode for the two corresponding values of $y(1)$. Then the Secant Method will give us the next value of b to try, and we iterate until $|y(1) - B| <$ tol, where tol is some specified tolerance for the error.

Numerical methods: eigenvalue problem

For illustrative purposes, we develop our numerical methods for what is perhaps the simplest eigenvalue ode. With $y = y(x)$ and $0 \leq x \leq 1$, this simple ode is given by

$$y'' + \lambda^2 y = 0. \tag{7.9}$$

To solve (7.9) numerically, we will develop both a finite difference method and a shooting method. Furthermore, we will show how to solve (7.9) with homogeneous boundary conditions on either the function y or its derivative y'.

Finite difference method

We first consider solving (7.9) with the homogeneous boundary conditions $y(0) = y(1) = 0$. In this case, we have already shown that the eigenvalues of (7.9) are given by $\lambda = \pi, 2\pi, 3\pi, \ldots$.

With n interior points, we have $x_i = ih$ for $i = 0, \ldots, n + 1$, and $h = 1/(n + 1)$. Using the centered-finite-difference approximation for the second derivative, (7.9) becomes

$$y_{i-1} - 2y_i + y_{i+1} = -h^2\lambda^2 y_i. \tag{7.10}$$

Applying the boundary conditions $y_0 = y_{n+1} = 0$, the first equation with $i = 1$, and the last equation with $i = n$, are given by

$$\begin{aligned} -2y_1 + y_2 &= -h^2\lambda^2 y_1, \\ y_{n-1} - 2y_n &= -h^2\lambda^2 y_n. \end{aligned}$$

The remaining $n - 2$ equations are given by (7.10) for $i = 2, \ldots, n - 1$.

It is of interest to see how the solution develops with increasing n. The smallest possible value is $n = 1$, corresponding to a single interior point, and since $h = 1/2$ we have

$$-2y_1 = -\frac{1}{4}\lambda^2 y_1,$$

so that $\lambda^2 = 8$, or $\lambda = 2\sqrt{2} = 2.8284$. This is to be compared to the first eigenvalue which is $\lambda = \pi$. When $n = 2$, we have $h = 1/3$, and the resulting two equations written in matrix form are given by

$$\begin{pmatrix} -2 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = -\frac{1}{9}\lambda^2 \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

This is a matrix eigenvalue problem with the eigenvalue given by $\mu = -\lambda^2/9$. The solution for μ is arrived at by solving

$$\det \begin{pmatrix} -2-\mu & 1 \\ 1 & -2-\mu \end{pmatrix} = 0,$$

with resulting quadratic equation

$$(2 + \mu)^2 - 1 = 0.$$

The solutions are $\mu = -1, -3$, and since $\lambda = 3\sqrt{-\mu}$, we have $\lambda = 3, 3\sqrt{3} = 5.1962$. These two eigenvalues serve as rough approximations to the first two eigenvalues π and 2π.

With A an n-by-n matrix, the MATLAB variable mu=eig(A) is a vector containing the n eigenvalues of the matrix A. The built-in function eig.m can therefore be used to find the eigenvalues. With n grid points, the smaller eigenvalues will converge more rapidly than the larger ones.

We can also consider boundary conditions on the derivative, or mixed boundary conditions. For example, consider the mixed boundary conditions given by $y(0) = 0$ and $y'(1) = 0$. The eigenvalues of (7.9) can then be determined analytically to be $\lambda_i = (i - 1/2)\pi$, with i a natural number.

The difficulty we now face is how to implement a boundary condition on the derivative. Our computation of y'' uses a second-order method, and we would like the computation of the first derivative to also be second order. The condition $y'(1) = 0$ occurs on the right-most boundary, and we can make use of the second-order backward-difference approximation to the derivative that we have previously derived. This finite-difference approximation for $y'(1)$ can be written as

$$y'_{n+1} = \frac{3y_{n+1} - 4y_n + y_{n-1}}{2h}. \tag{7.11}$$

Now, the nth finite-difference equation was given by

$$y_{n-1} - 2y_n + y_{n+1} = -h^2 y_n,$$

and we now replace the value y_{n+1} using (7.11); that is,

$$y_{n+1} = \frac{1}{3}\left(2hy'_{n+1} + 4y_n - y_{n-1}\right).$$

Implementing the boundary condition $y'_{n+1} = 0$, we have

$$y_{n+1} = \frac{4}{3}y_n - \frac{1}{3}y_{n-1}.$$

Therefore, the nth finite-difference equation becomes

$$\frac{2}{3}y_{n-1} - \frac{2}{3}y_n = -h^2\lambda^2 y_n.$$

For example, when $n = 2$, the finite difference equations become

$$\begin{pmatrix} -2 & 1 \\ \frac{2}{3} & -\frac{2}{3} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = -\frac{1}{9}\lambda^2 \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

The eigenvalues of the matrix are now the solution of

$$(\mu + 2)\left(\mu + \frac{2}{3}\right) - \frac{2}{3} = 0,$$

or

$$3\mu^2 + 8\mu + 2 = 0.$$

Therefore, $\mu = (-4 \pm \sqrt{10})/3$, and we find $\lambda = 1.5853, 4.6354$, which are approximations to $\pi/2$ and $3\pi/2$.

Shooting method

We apply the shooting method to solve (7.9) with boundary conditions $y(0) = y(1) = 0$. The initial value problem to solve is

$$y' = z,$$
$$z' = -\lambda^2 y,$$

with known boundary condition $y(0) = 0$ and an unknown boundary condition on $y'(0)$. In fact, any nonzero boundary condition on $y'(0)$ can be chosen: the differential equation is linear and the boundary conditions are homogeneous, so that if $y(x)$ is an eigenfunction then so is $Ay(x)$. What we need to find here is the value of λ such that $y(1) = 0$. In other words, choosing $y'(0) = 1$, we solve

$$F(\lambda) = 0, \tag{7.12}$$

where $F(\lambda) = y(1)$, obtained by solving the initial value problem. Again, an iteration for the roots of $F(\lambda)$ can be done using the Secant Method. For the eigenvalue problem, there are an infinite number of roots, and the choice of the two initial guesses for λ will then determine to which root the iteration will converge.

For this simple problem, it is possible to write explicitly the equation $F(\lambda) = 0$. The general solution to (7.9) is given by

$$y(x) = A\cos(\lambda x) + B\sin(\lambda x).$$

The initial condition $y(0) = 0$ yields $A = 0$. The initial condition $y'(0) = 1$ yields

$$B = 1/\lambda.$$

Therefore, the solution to the initial value problem is

$$y(x) = \frac{\sin(\lambda x)}{\lambda}.$$

The function $F(\lambda) = y(1)$ is therefore given by

$$F(\lambda) = \frac{\sin\lambda}{\lambda},$$

and the roots occur when $\lambda = \pi, 2\pi, \ldots$.

If the boundary conditions were $y(0) = 0$ and $y'(1) = 0$, for example, then we would simply redefine $F(\lambda) = y'(1)$. We would then have

$$F(\lambda) = \frac{\cos\lambda}{\lambda},$$

and the roots occur when $\lambda = \pi/2, 3\pi/2, \ldots$.