Computational methods & modelling 3

Study guide

# Table of Contents

**Contents**

[Table of Contents 1](#_Toc152601925)

[Guide to Using Python 4](#_Toc152601926)

[For Loops 4](#_Toc152601927)

[While Loops 4](#_Toc152601928)

[Break and continue 4](#_Toc152601929)

[Importing Data in Python 4](#_Toc152601930)

[Text file 4](#_Toc152601931)

[Input Command 4](#_Toc152601932)

[Definition of Functions in Python 4](#_Toc152601933)

[Defining Function 4](#_Toc152601934)

[Lambda Function 4](#_Toc152601935)

[Symbolic Computation 4](#_Toc152601936)

[Algebraic Computation 4](#_Toc152601937)

[Differentiating wrt. X 4](#_Toc152601938)

[Integrating wrt. X 4](#_Toc152601939)

[Simplifies Expression 4](#_Toc152601940)

[Finding limit as x -> 0 5](#_Toc152601941)

[Solve 5](#_Toc152601942)

[Solving Equations 5](#_Toc152601943)

[Sympy solving multiple equations 5](#_Toc152601944)

[Fsolve for solving multiple equation systems (nonlinear) 5](#_Toc152601945)

[Linalg for solving multiple equation systems (linear equations) 5](#_Toc152601946)

[Standard Functions 5](#_Toc152601947)

[Quad 5](#_Toc152601948)

[Dblquad 5](#_Toc152601949)

[Minimise (Optimisation) 5](#_Toc152601950)

[Root Finding 8](#_Toc152601951)

[Bracketing method 8](#_Toc152601952)

[Bisection Technique 8](#_Toc152601953)

[False Position Method 8](#_Toc152601954)

[Modified FPM 9](#_Toc152601955)

[Incremental Search Techniques 9](#_Toc152601956)

[Open Root Finding Methods 10](#_Toc152601957)

[Differences in Open and Closed Root Finding Methods 10](#_Toc152601958)

[Two Point Graphical Method 10](#_Toc152601959)

[Newton Raphson Method 11](#_Toc152601960)

[Comparing Secant Technique and False Position 12](#_Toc152601961)

[Modified Secant Technique 12](#_Toc152601962)

[Inverse Quadratic 13](#_Toc152601963)

[Interpolation Without Real Roots 13](#_Toc152601964)

[Multiple Repeated Root Technique 14](#_Toc152601965)

[Ralston-Rabinowitz Method 14](#_Toc152601966)

[Supplement & Root Finding Examples 15](#_Toc152601967)

[Engineering Examples 15](#_Toc152601968)

[Common Roots of Equation Systems (Lecture 3, Root Finding 2) 17](#_Toc152601969)

[Solution 17](#_Toc152601970)

[Roots of Equation Systems 17](#_Toc152601971)

[Roots of Polynomial Equations 17](#_Toc152601972)

[Regression and Interpolation (Lecture 4) 18](#_Toc152601973)

[Minimizing the sum of errors for all available data 18](#_Toc152601974)

[Minimax Criterion 18](#_Toc152601975)

[Minimizing the sum of squares of the residuals 19](#_Toc152601976)

[Polynomial Regression 19](#_Toc152601977)

[Quadratic Regression 19](#_Toc152601978)

[General nonlinear regression 19](#_Toc152601979)

[Linear interpolation with Newton polynomials 19](#_Toc152601980)

[Using Interpolation to approximate a function 20](#_Toc152601981)

[First and Second order Interpolation 20](#_Toc152601982)

[Splines 20](#_Toc152601983)

[Linear Algebra 21](#_Toc152601984)

[Solutions of Systems of Linear Equations 21](#_Toc152601985)

[Naïve (Simple) Gauss Elimination 23](#_Toc152601986)

[NGE Disadvantages 24](#_Toc152601987)

[Gauss Elimination with Partial Pivoting 24](#_Toc152601988)

[Gauss Elimination Applied to a Physics Problem 25](#_Toc152601989)

[Calculate Cord Tensions in a Tandem Team of Parachutists 25](#_Toc152601990)

[Ordinary Differential Equations 1 (Lecture 6) 26](#_Toc152601991)

[Taylor’s Theorem and Taylor Series 26](#_Toc152601992)

[Taylor series term-by-term 26](#_Toc152601993)

[Truncation Error 27](#_Toc152601994)

[Numerical Derivative 27](#_Toc152601995)

[One-Step Methods for ODEs 27](#_Toc152601996)

[Euler Method 28](#_Toc152601997)

[Runge-Kutta Method 28](#_Toc152601998)

[Systems of ODEs and Stiff Equations (Lecture 7) 28](#_Toc152601999)

[Stiff ODEs 28](#_Toc152602000)

[Implicit Euler 28](#_Toc152602001)

[Numerical Integration (Lecture 9) 28](#_Toc152602002)

[Quadrature Techniques 28](#_Toc152602003)

[Three Main Quadrature Rules 29](#_Toc152602004)

[Three Eighths Simpson’s Rule 29](#_Toc152602005)

[Adaptive Algorithm Numerical Methods 29](#_Toc152602006)

[Optimisation (Lecture 10) 29](#_Toc152602007)

[Golden Search Technique 30](#_Toc152602008)

[Parabolic Interpolation 31](#_Toc152602009)

[Newtons Method 31](#_Toc152602010)

# Guide to Using Python

## For Loops

For (i = 1; I <=10; i+-):

<loop conditions>

*here i will start at 1 and will remain less than or equal to 10. The value of i increases by 1 in every step.*

For I in range(5):

<loop conditions>

*here i will loop through every value from 0 to 5.*

## While Loops

While loops specify a continuing condition for the duration of a loop, such as:

i = 1

while i <6:

print(i)

if i ==3:

break

i +=1

## Break and continue

“break” stops a loop and returns the first valid value, “continue” stops the current iteration and starts the next iteration of a loop.

## Importing Data in Python

### Text file

File = open(“sample.txt”)

Data = file.read()

Print(data)

File.close

#### Alternatively

With open(“welcome.txt”) as file:

Object data = file.read()

*This is more efficient as it closes the file after use.*

### Input Command

Value = input(“Please enter a string: \n”)

Print(f”You entered {value}”)

## Definition of Functions in Python

### Defining Function

Def my\_function(x):

Return 5 \* x

Print(my\_function(6))

### Lambda Function

X = lambda a,b: a\*b

Print(x(5,6)

This method has more advantages as lambda can be used repeatedly to define numerous functions.

### Symbolic Computation

From sympy import \*

X = Symbol(“x”\_, y = symbol(“y”)

### Algebraic Computation

Print(2\*x + 3\*x – y)

### Differentiating wrt. X

Print(diff(x\*\*2,x)

### Integrating wrt. X

Print(integrate(cos(x),x)

### Simplifies Expression

Print(simplify((x\*\*2 + x\*\*3)/x\*\*2))

### Finding limit as x -> 0

Print(limit(sin(x)/x, x, 0))

### Solve

Print(solve(5\*x – 15, x))

## Solving Equations

### Sympy solving multiple equations

A diagram of equations and formulas

Description automatically generated with medium confidence

### A screenshot of a computer program Description automatically generatedFsolve for solving multiple equation systems (nonlinear)

### Linalg for solving multiple equation systems (linear equations)

A screenshot of a computer program

Description automatically generated

## Standard Functions

### Quad

General Purpose integration

#### Example

From scipy.integrate import quad

Def integrand(x, a, b):

Return a\*x\*\*2 + b

a = 2

b = 1

I = quad(integrand, 0, 1, args=(a,b))

### Dblquad

Performs double integration on a lambda function

#### Example

A math equations and formulas

Description automatically generated with medium confidence

From scipy.integrate import dblquad

Area = dblquad(lambda x, y: x\*y, 0, 0.5, lambda x:0, lambda x: 1-2\*y)

#### NOTE: The routine defines outer integral with fixed bounds first, then uses lambda to define the non-fixed bounds of the integral

### Minimise (Optimisation)

For unconstrained multivariate optimisation

#### Rosenbrock Example

Import numpy as np

From scipy.optimize import minimize

Def rosen(x): “””The Rosenbrock function”””

Return(100.0\*(x[1:]-x[:-1]\*\*2.0)\*\*2.0 + (1-x[:-1])\*\*2)

X0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])

Res = minimize(rosen, x0, method=”nelder-mead”, options={“xatol”: 1e-8, “disp”:True})

#### Rosen hess Example

For unconstrained multivariate optimisation using gradients

Def rosen\_hess(x):

x = np.asarray(x)

H = np.diag(-400\*x[:-1],1) – np.diag(400\*x[:-1],-1)

diagonal = np.zeros\_like(x)

diagonal[0] = 1200\*x[0]\*\*2 – 400\*x[1] + 2

diagonal[-1] = 200

diagonal[1:-1] = 202 + 1200\*x[1:-1]\*\*2 – 400\*x[2:]

H = H + np.diag(diagonal

Return H

Res = minimize(rosen, x0, method=”Newton-CG”, jac=rosen\_der, hess=rosen\_hess, options={“xtol”: 1e-8, “disp”: True})

Res.x array([1., 1., 1., 1., 1.])

#### Nonlinear Constraint Example

For constrained multivariate optimisation using gradients

def cons\_f(x):

return [x[0]\*\*2 + x[1], x[0]\*\*2 - x[1]]

def cons\_J(x):

return [[2\*x[0], 1], [2\*x[0], -1]]

def cons\_H(x, v):

return v[0]\*np.array([[2, 0], [0, 0]]) + v[1]\*np.array([[2, 0], [0, 0]])

from scipy.optimize import NonlinearConstraint

nonlinear\_constraint = NonlinearConstraint(cons\_f, - np.inf, 1, jac=cons\_J, hess=cons\_H)

#### Constrained multivariate optimisation using gradients

for constrained multivariate optimisation using gradients

For the previous codes, boundaries and linear constraints for the optimisation routine are specified with commands such as these.

from scipy.optimize import Bounds >>>

bounds = Bounds([0, -0.5], [1.0, 2.0])

from scipy.optimize import LinearConstraint >>>

linear\_constraint = LinearConstraint([[1, 2], [2, 1]], [-np.inf, 1], [1, 1])

#### All arguments of the minimise function have been defined

x0 = np.array([0.5, 0])

res = minimize(rosen, x0, method='trust-constr',

jac=rosen\_der, hess=rosen\_hess,

constraints=[linear\_constraint,nonlinear\_constraint],

options={'verbose': 1}, bounds=bounds)

#### Solution of ODEs by odeint

import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt

# function that returns dy/dt

def model(y,t):

k = 0.3

dydt = -k \* y

return dydt

# initial condition

y0 = 5

# time points

t = np.linspace(0,20)

# solve ODE

y = odeint(model,y0,t)

# plot results

plt.plot(t,y)

plt.xlabel('time')

plt.ylabel('y(t)')

plt.show()

#### A family of solutions can be solved for different values of the parameter, k as below:

import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt

# function that returns dy/dt

def model(y,t,k): dydt = -k \* y

return dydt

# initial condition

y0 = 5

# time points

t = np.linspace(0,20)

# solve ODEs

k = 0.1

y1 = odeint(model,y0,t,args=(k,))

k = 0.2

y2 = odeint(model,y0,t,args=(k,))

k = 0.5

y3 = odeint(model,y0,t,args=(k,))

# plot results

plt.plot(t,y1,'r-',linewidth=2,label='k=0.1')

plt.plot(t,y2,'b--',linewidth=2,label='k=0.2')

plt.plot(t,y3,'g:',linewidth=2,label='k=0.5')

plt.xlabel('time')

plt.ylabel('y(t)')

plt.legend()

plt.show()

# Root Finding

## Bracketing method

Root finding with graphs can be fast, but inaccurate without testing estimated roots against function value.

Evaluating root estimates based solely on function values can be laborious and haphazard without a reasonable algorithm to save computational steps.

## Bisection Technique

Works relatively well but is comparatively less efficient. This is a brute force technique that relies on enough iterations to achieve a true answer, or one with an acceptable error margin.

A white text with black text

Description automatically generated

This algorithm could proceed for large iterations in order to determine an accurate/exact answer.

In practice, a termination criteria should be used to prevent it going for infinity.

A close-up of a text

Description automatically generated

#### Acceptable Error

This code then needs an acceptable error which, when achieved, terminates the program

#### Number of Iterations

To estimate the number of iterations based on the knowledge of the initial bracket size

## False Position Method

Takes into account the magnitude of the function values on either bound of the interval.

This technique is dependent on the shape of the function cure, and therefore won’t always be the fastest way on convergence.

A graph of a graph of a line

Description automatically generated with medium confidenceExample: The code cannot “see” past the long, flat section and therefore must crawl along the curve.

#### NOTE: it is essential to plot a graph of a function before choosing a suitable estimation technique.

## Modified FPM

An unmodified FPM has the issue that one bound may remain completely unchanged through many iterations.

A modified FPM halves the function value at the “stuck” bound.

#### Modified FPM Example

MAX\_ITER = 1000000

# The function is x^3 - x^2 + 2

def func( x ):

return (x\*\*3 - 4\*(x\*2) + 10)

# Prints root of func(x) in interval [a, b

]def regulaFalsi( a , b):

if func(a) \* func(b) >= 0:

print("You have not assumed right a and b")

return -1c = a

# Initialize result

for i in range(MAX\_ITER):

# Find the point that touches x axis

c = (a \* func(b) - b \* func(a))/ (func(b) -func(a))

# Check if the above found point is root

if func(c) == 0:

break

# Decide the side to repeat the steps

elif func(c) \* func(a) < 0:

b = c

else:

a = c

print("The value of root is : " , '%.4f' %c)

# Test the function by setting a and b and calling the code by the filename

# Initial values assumed

a =-200

b = 300

regulaFalsi(a, b)

## Incremental Search Techniques

These techniques choose one point at either end of an interval of interest around a root of a function. Then at arbitrary increments (moving in one direction), the function is evaluated until the value is within the required error tolerance.

In cases where there are very close roots, if too big of a step interval is chosen then the roots may be skipped all together.

#### Example

A screenshot of a computer program

Description automatically generated

## Open Root Finding Methods

This technique only requires 1 starting point.

### Differences in Open and Closed Root Finding Methods

Closed methods will always converge to a solution, provided that the root or roots actually lie in the domain of x chosen for investigation.

Open methods, by contrast, may not converge towards a root but if they do they tend to be far more efficient in reaching a solution in fewer steps than bisection

#### Single Fixed Point Iteration Example

A screenshot of a computer program

Description automatically generated

## Two Point Graphical Method

The function above can easily be split into two further equations, as shown

The root of the main function is then the x-value of where these functions intersect. This technique cannot be used uncritically in all situations.

A graph of x and x

Description automatically generated

## Newton Raphson Method

An initial root value is used to evaluate a function value from which a tangent line is drawn to the x-axis to produce a next estimate for the root. This continues until the function approximates to 0 within an error. This has a much faster convergence and a far lower error than the single point fixed method.

#### Newton Raphson Example

A screenshot of a computer program

Description automatically generated

#### Functions that are not solveable

These functions include asymptotic functions, functions with multiple minima, cyclic/periodic functions or functions with no real root.

#### Measures to Mitigate

* Graph the function to see the shape
  + Establish whether a root exists in the domain of interest
* Check each solution to establish its closeness to 0
* Include an upper limit in the number of iterations to prevent infinite cycling

#### Complications

One of the complications of this technique is that it requires the calculation of an exact derivative. For known functions, this is fine as they are analytically differentiable but for those that are this method is problematic. The secant method may then be used instead (this calculates an approximation to the point derivative).

#### Secant Technique Example

A screenshot of a computer program

Description automatically generated

### Comparing Secant Technique and False Position

These techniques are very similar, however False Position always sets the new right bound of the new line to be at the new estimate for the root. The secant method uses the new estimate to cut the curve above it. False position is guaranteed to find the root, however secant is not.

A diagram of a graph

Description automatically generated with medium confidence

## Modified Secant Technique

This involves the alteration of how derivatives are evaluated. In the secant, two different points are used to compute the derivative for the given step. However, in the modified secant, the new derivative is calculated by retaining one old point and taking a small increment on this value as the second reference point.

#### Modified Secant Technique Example

A screenshot of a computer program

Description automatically generated

## Inverse Quadratic

It is more effective to use a second order curve to model a curve that is third order or higher than it is to use the linear secant method. (GOOD FOR IMAGINARY)

A graph of a function

Description automatically generated with medium confidenceInverse quadratic interpolation uses a parabola x = f(y) (a parabola on its side) to model a function

## Interpolation Without Real Roots

A graph of a function

Description automatically generatedIn this example, an attempted interpolation function gives a parabola with no real roots.

This method crashes immediately.

#### Inverse Quadratic Interpolation Example

A screenshot of a computer screen

Description automatically generated

## Multiple Repeated Root Technique

Multiple roots exist for the condition that a point on the function touches the x-axis.

This can be solved with using open search methods. (Not bracketing as they do not change sign). It is easiest to use the Ralston-Rabinowitz method

### Ralston-Rabinowitz Method

The first step is to define a function as follows:

Then substituting u(x) and u’(x) for f(x) and f’(x)

Then, to find u’(x) in terms of f’(x) and f(x)

Substituting into Newton-Raphson we get:

# Supplement & Root Finding Examples

### Engineering Examples

We are given function:

#### Solution

To solve this problem, we must calculate as many variables as possible. The first obvious variable to calculate is Re:

We can then substitute this into Colebrook[[1]](#footnote-1) to get the following:

#### STEP ONE: graph the function

A graph with a line

Description automatically generatedThis gives us the following information:

* A solution actually exists for g(f) = 0
* The solution is in the region of f = 0.03
* The function is continuous and differentiable
* Can use bracketing or open method to solve this
* Known initial values to use

#### Bisection to solve Colebrook

A screenshot of a computer

Description automatically generated

#### False Position Code

A white background with black text

Description automatically generated

#### Newton Raphson

A screenshot of a computer program

Description automatically generated

#### Secant Technique

A white screen with black text

Description automatically generated

# Common Roots of Equation Systems (Lecture 3, Root Finding 2)

In scenarios where it is necessary to determine the common roots of multiple equations simultaneously.

For the following quadratics:

We can use fixed point iteration or the newton-raphson technique

## Solution

Take initial guesses of x & y values (x0 and y0), in this case x0 = 1.5 and y0 = 3.5 and rearrange the equations as follows to get a better x(i+1) estimate

The same is done simultaneously for the y-coordinate, as follows:

Then, substituting the guesses an improved value of xi, xi+1 is given

If this value is off, then the equations can be rearranged differently, as follows to give more accurate results:

## Roots of Equation Systems

The condition for fixed point iteration to converge on a solution for two non-linear equations is:

#### NOTE: test the conditions above prior to using this technique

An alternative is to use Newton-Raphson method. This is difficult to implement, and therefore isn’t used.

### Roots of Polynomial Equations

Number of computations for an nth order polynomial is:

#### NOTE: To reduce the number of computations, employ a nested version of the polynomial, as follows

# Regression and Interpolation (Lecture 4)

A diagram of a linear function

Description automatically generated with medium confidence

The mathematical expression for a straight line is:

Where a0 and a1 are coefficients for the intercept and gradient and e is the error. We want to minimize the error

## Minimizing the sum of errors for all available data

This is inadequate

A diagram of a line graph

Description automatically generated

As shown, obviously the best fit is the solid line between points. However, any straight line passing through the midpoint results in a minimum error of 0 because the errors cancel.

## Minimax Criterion

This chooses the line that minimizes the max distance among points. This isn’t effective as it gives undue influence on an outlier

A diagram of a line graph

Description automatically generated

To overcome this, we can minimize the sum of squares of the residuals

## Minimizing the sum of squares of the residuals

This has many advantages, including that it yields a unique line for a given set of data

### Example

To use the following model to fit data:

We need to determine the a0 and a1 so that the least-square error is minimized.

To do this:

Differentiate Sr with respect to each coefficient

Setting these equal to 0 will result in a minimum Sr

A math equations on a white background

Description automatically generated

## Polynomial Regression

It is possible to fit polynomials of any order m to data using polynomial regression

## Quadratic Regression

To get a minimum Sr, you must set the respective derivatives equal to 0. Then you have three linear equations and three unknowns.

### General nonlinear regression

In many engineering cases, nonlinear models must be fit to data. These models depend on their parameters, for example an exponential:

Here we use a numerical optimisation method.

### Linear interpolation with Newton polynomials

The first order interpolation with a Newton polynomial is:

A math equations with numbers

Description automatically generated with medium confidenceIn general, the smaller the interval between points – the better the approximation.

### Using Interpolation to approximate a function

To approximate a function, we sample the function at two locations and use linear interpolation to approximate the function between these locations.

If two points are close, we obtain a better approximation.A graph of a function

Description automatically generated with medium confidence

### A diagram of a function Description automatically generatedFirst and Second order Interpolation

First order, we approximate the underlying function with a straight line.

Second order, we approximate the underlying function with a parabola.

For second order we need 3 points to construct a parabola that goes through them.

Second order is usually more accurate

### Splines

Spline interpolation is very flexible and powerful.

The interpolant is a special kind of piecewise polynomial called a spline.

Spline interpolation fits low-degree polynomials to small subsets of the values. The cubic spline is most common. This can be made very accurate.

#### NOTE: avoid the oscillatory behaviour that occurs if we fit a single high-order polynomial using many data points

A graph of a function

Description automatically generated with medium confidence

## Linear Algebra

### Solutions of Systems of Linear Equations

Usually systems of three or less equations can be solved graphically by Cramer’s Rule or elimination of unknown variables.

If there are common roots of any two equations, they can be evaluated at the intersection point of the two functions when they are plotted.

Consider:

With two equations and two unknowns, so that it will be possible to solve for both unknowns x1 and x2

A diagram of equations and graphs

Description automatically generated

Not all two-equation systems behave this way, as shown:

A diagram of equations and equations

Description automatically generated with medium confidence

A and B are not resolvable as they are intrinsic properties of the equation system. C, however, is solvable and there needs to be found a suitable non-graphical technique to determines the solution.

#### Cramer’s Rule

A white background with black text

Description automatically generatedA math equations on a white background

Description automatically generated

If the determinant is 0, the system is singular, and if it is very close to 0, the problem is said to be ill-conditioned and almost singular.

#### Cramer’s Rule for up to Three Equations

A math equations and numbers

Description automatically generated with medium confidence

A white paper with black and white numbers and numbers

Description automatically generated

#### Elimination of Unknowns

Cramer’s Rule is 100% effective in cases with solutions. (Non-singular system). It is not computationally efficient for systems involving more than 3 equations.

Here, we use the elimination of unknowns. Example below.

Here we can times the first equation by a21 and the second equation by a11 to eliminate x1 and solve for x2.

### Naïve (Simple) Gauss Elimination

This systemizes the process of forward elimination and reverse substitution.

A math equations on a white background

Description automatically generated

A math equations on a white background

Description automatically generatedThe first equation remains unchanged in the revised matrix after this and subsequent operations.

A diagram of a mathematical equation

Description automatically generated

After the forward eliminations are complete, the unknown variable in the last modified equation can be solved for directly. (In this case x3) Once this is solved, the remaining unknown in the previous equation is solved etc etc. This same process applies to any system of n linear equations that one solves using NGE. This can be computerised, unlike crammer’s as it has a recognisable repeatable algorithm.

### NGE Disadvantages

* NGE and all elimination techniques will crash as they cannot handle divisions by 0
* NGE can accumulate inaccuracies due to round-off error
  + Every decimal operation incurs some intrinsic relative error, and these accumulate to high total error over multiple calculations.
* Ill-Conditioned Systems are not suited to solution using NGE (and elimination methods) because of round off error
  + This is where small changes in coefficients induce large changes in solution.

#### Techniques for improving solutions to LESs

* Use of more significant figures
  + Reduces round-off errors
* Use of partial or full pivoting
  + This is the practice of identifying zero or near-zero coefficients of elements in a given row and switching the row with one containing much larger coefficients in the same position.

### Gauss Elimination with Partial Pivoting

A screenshot of a computer program

Description automatically generated

## Gauss Elimination Applied to a Physics Problem

### Calculate Cord Tensions in a Tandem Team of Parachutists

A screenshot of a computer

Description automatically generated

A diagram of a parachutist

Description automatically generated

A math equations with numbers and symbols

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated

# Ordinary Differential Equations 1 (Lecture 6)

## Taylor’s Theorem and Taylor Series

If the function of independent variable x and its n+1 derivatives are continuous in an interval containing both points xi and xi+1 = xi + h, f(x) can be expanded in the following series:

A math equations on a white background

Description automatically generated

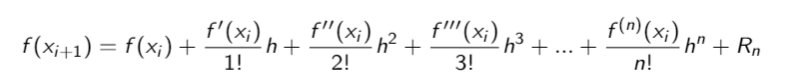
Rn can also be expressed in Lagrangian form, and is often called the Truncation Error.

A math equation with numbers and symbols

Description automatically generated

### Taylor series term-by-term

Defining h = xi+1 – xi the Taylor series can be written as:



Depending on the number of terms kept in the series, we have different levels of approximation.

* Zero-order approximation
  + F(xi+1) = f(xi)
    - If f(x) is constant, this is a perfect estimate
* First order approximation
  + F(xi+1) = f(xi) + f’(xi)h
    - This can predict a change in the function but is exact only if the function is linear
* Order n approximation
  + A math equations on a white background

    Description automatically generated

## A math equation with numbers Description automatically generated with medium confidenceTruncation Error

Cannot be determined since  is not known. We only know that it lies between xi and xi+1

We have control over h. For different orders, the error decreases in different ways if we decrease h.

A black text on a white background

Description automatically generated

## Numerical Derivative

Considering,

A math equations on a white background

Description automatically generatedThis equation can be solved for A math equation with black text

Description automatically generated with medium confidence

With an estimate of error:

A math equation with black text

Description automatically generated with medium confidence

in the usual more compact formA math equations with black text

Description automatically generated with medium confidence

where :

* ∆fi is the first forward difference
* h is the step size
* ∆fi/h is the first forward divided difference
  + This is one of many ways to approximate the derivative using the taylor series.

## One-Step Methods for ODEs

To solve equations in the form:

With a given initial condition y0 = y(x0)

To solve this equation with a numerical method on a set of discrete points, we need to be able to extrapolate from a value yi to a new value yi+1 over a step h:

Where is an estimate of an appropriate slope of the function y over the step h.

The simplest approach is to estimate the slope from the differential equation itself as the first derivative of y at the point xi, which is the Euler Method

## Euler Method

A new value of y is computed extrapolating linearly over the step h using a slope approximated with the derivative in the original point xi, where the solution and its derivatives are known.

## Runge-Kutta Method

Achieves high accuracy without the use of higher order derivatives like with the Taylor series.

Many versions exist but it can be cast as:

Where is an increment function

#### NOTE: a runge-kutta method with n =1 is the Euler method

# Systems of ODEs and Stiff Equations (Lecture 7)

## Stiff ODEs

ODE or a system of ODEs where fast and slow components exist.

* Slow component: we need to solve the equation over a large interval
* Fast component: we usually need a small step h to capture the fast component
* Long interval with small steps means many steps.

## Implicit Euler

Implicit methods employ information at locations that have not yet been computed. For the implicit euler method, we use the derivative in the point x(i+1) to estimate the slope. To compute y(i+1) we must find the root of the function F(yi+1)

### Implicit vs explicit

The implicit Euler method requires the solution of the (in general, non-linear) equation F(yi+1) = 0. This requires a root finding method, for example the Secant method in this case.

# Numerical Integration (Lecture 9)

### Quadrature Techniques

Quadrature is the process of evaluating the integral of the curve – it is not exact, it is approximate.

M – midpoint rule T = Trapezodial Rule S – true value of integral

#### A group of graphs with different colored lines Description automatically generatedSimpsons Third Rule

Gives an exact answer for a third order integral. (Not exact for powers > 4)

#### Simpsons Third Rule quadratic interpolation

Between limits a, b and their midpoint c

### Three Main Quadrature Rules

#### A diagram of a function Description automatically generated with medium confidenceRectangle/Midpoint Rule

#### Trapezoidal Rule

#### Composite Simpson’s Rule

### Three Eighths Simpson’s Rule

A more sophisticated model of Simpson third rule. Resulting in

### Adaptive Algorithm Numerical Methods

Quadrature techniques work well for certain functions and less well for others. Adaptive algorithms allow intervals to be subdivided during the numerical integration if accuracy is not achieved after n steps. The adaptive algorithm approach can be applied dynamically within any of the three quadrature techniques. The “basic” equation is shown below.

Which we can refine by halving the step size to give:

The error can be defined as the difference between integral estimates

## Optimisation (Lecture 10)

Two types of optimisations:

* One-dimensional optimisation – a curve in a plane (2D)
* Two-dimensional optimisation (3D functional surface)

If both the function and constraints are linear then the optimisation is an example of linear programming. If f(x) is a quadratic but its constraints are linear, we have quadratic programming. If both f(x) and constraints are non-linear, we have non-linear programming.

The DoF of a system in an optimisation problem is calculated by the term n-p-m. Where:

* N – the number of dimensions in the x vector
* P – number of equality constraints
* M – number of inequality constraints

To obtain a solution, m + p < n. If m + p >n, the optimisation is overconstrained.

#### NOTE: Pre-assess if a problem is over-constrained

### Golden Search Technique

The most basic unconstrained one-dimensional search technique. Modelled closely on the bisection method used to find a function root (except this time for minimum/maximum)

A diagram of a function

Description automatically generatedGS relies on selecting two estimate points either side of a maximum or minimum. An effective strategy for selection is by using the golden ratio

Then using the quadratic equation to solve for R gives our upper and lower estimates.

If f(x1)>f(x2), then all points left of x2 can be eliminated as no maximum will occur here. X2 becomes the new xl. If f(x1) < f(x2), then all points right of x1 can be eliminated from the region of interest. Xl becomes the new xu for the next iteration. Then we can calculate a new x1.

This golden ration halves the number of necessary function evaluations needed to complete the algorithm. Convergence is guaranteed, although the rate of convergence is infinite.

The approximation error for the optimal x value.

### Parabolic Interpolation

A diagram of a function

Description automatically generatedWhere x0, x1 & x2 are the initial guesses and x3 is calculated approximation to the point xopt. The new points can then be done by reassigning sequentially.

### Newtons Method

For finding a root approximation:

When f’(x) = 0, there is a minimum/maximum

### Brent’s Method

Combines parabolic interpolation and golden search techniques in one code.

### Newtons Method (two – variable optimisation)

The absolute value of the Hessian is a test point for maximum or minimum.

A close up of a number

Description automatically generated

## Optimisation (Lecture 11)

### Steepest Ascent (Hill Climb)

The solution to a multivariative problem is equivalent to finding the shortest route to the top of a high mountain.

The first step is to identify where you are on the surface, then the curve of travel in any direction is expressible by a function. At the peak of the optimisation path function, g(h), the derivative is 0. The value of h at this point is used to calculate the next point.

The path of steepest ascent is shown below.

A diagram of a mathematical equation

Description automatically generated

### Constrained Non-Linear Optimisation (finding min & maxima)

#### Tank Example

A diagram of a cylinder

Description automatically generatedA text on a white paper

Description automatically generatedA math equations with text

Description automatically generated with medium confidenceA math equations and formulas

Description automatically generated with medium confidence

To solve this we can use the in-built solver of Microsoft ExcelA screenshot of a computer

Description automatically generatedA screenshot of a spreadsheet

Description automatically generated

### Example 1 Lagrange Multiplier Technique for Multivariate Constrained Optimisation

The use of a Lagrange Multiplier solves a constrained multivariate optimisation problem more efficiently by incorporating the constraint function into the objective function.

A compound expression for the objective and constrain functions gives the following:

A math problem with equations

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

### Example 2 Maximising Revenue of an Engineering Project

A math equations and numbers

Description automatically generatedA paper with text and numbers

Description automatically generatedA math equations on a white background

Description automatically generated

1. Used to calculate Darcy friction Factor. [↑](#footnote-ref-1)