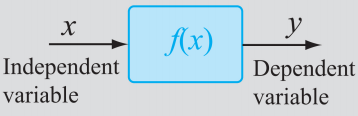
Formulae that are highlighted in green will be given in the exam, unless they are to be derived.

02 Mathematical Background

# Functions



The span of is called the **domain**.

The span of is called the **range**.

The span can also be called an **interval**:

* Open interval: or
* Closed interval:

### Limit of a Function

For continuous functions:

may be discontinuous or singular at . In these cases, the limit is said **not to exist**.

### Continuity of a Function

A function at is said to be continuous if:

* exists

### The Intermediate Value Theorem

If both:

1. is continuous in
2. is any number between and

then such that .

### Derivatives of a Function

The derivative of a function is denoted as either:

It is defined as:

is the slope of the secant connecting and .

In the limit as , we get a slope of tangent at .

This slope is equivalent to the rate of change of with respect to at .

### The Chain Rule

Example:

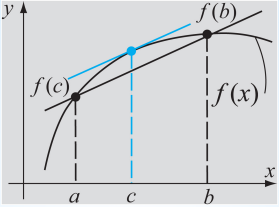
|  |
| --- |
|  |

### The Product Rule

Example:

|  |
| --- |
|  |

### The Mean Value Theorem (For Derivatives)



If is a continuous function in and is differentiable in the open interval , then such that:

In other words, at some point in the interval, the slope of the tangent line will equal the slope of the secant line between and .

### Integral / Antiderivative of a Function

If then:

* The **indefinite integral** of is :
* The **definite integral** of is :

Assuming is defined and continuous over then:

### Fundamental Theorems of Calculus

and

### The Mean Value Theorem (For Integrals)

If is continuous over then such that:

The **average value** of over the interval is:

In other words, there must be some value within the interval that takes on the average value of over the interval .

# Vectors

Vectors are quantities that have a **magnitude** and **direction**.

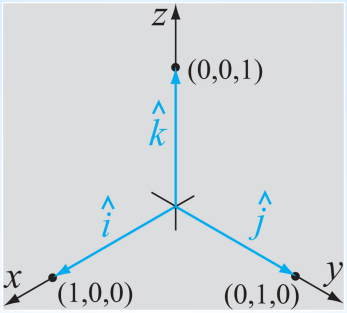
Scalars are quantities with **magnitude only**.

To denote the vector we write (or in these notes).

The **magnitude** of a vector is denoted or .

A vector may be represented graphically as a **directed line segment**.

Projections of the vector onto each of the coordinate axes define the **components** of the vector.



, and are the **unit vectors** in the , and directions respectively.

A unit vector is a vector with a **magnitude of unity** in a particular direction:

We can therefore write:

where is the -component of (the projection of onto the -axis) and so on.

A vector may be written by listing the **magnitudes of its components** in a row or column:

e.g.

The **magnitude** of a vector in three-dimensional Cartension space is its **length**:

The unit vector in the direction of is:

In mathematics, a vector is a set or list of numbers written in a row or column:

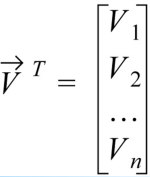
e.g.

### Addition and Subtraction of Vectors

### Multiplication by a Scalar

### Transpose of a Vector

If then:



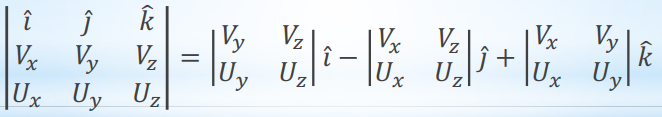
### Multiplication of Two Vectors

**Dot Product / Scalar Product**:

or

(Multiply the magnitudes of both vectors and by )

**Cross Product / Vector Product**:

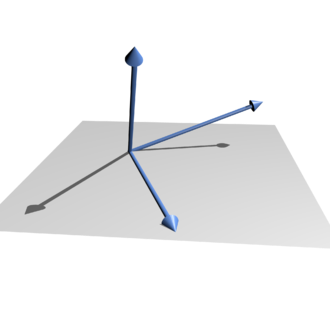


### Linear Dependence and Independence on a Set of Vectors

A set of vectors is said to be **linearly independent** if:

is satisfied .

Otherwise, the set of vectors is said to be **linearly dependent**.

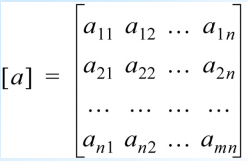


i.e. If one of the vectors in the set can be defined as a linear combination of the others, then the set of vectors is said to be linearly dependent.

### The Triangle Inequality

# Matrices & Linear Algebra

A matrix is a rectangular array of numbers. The size of the matrix refers to the number of rows and columns that it contains.



An element refers to row and column .

A **row vector** is a matrix.

A **column vector** is a matrix.

### Multiplication by a Scalar

If is a matrix, and is a scalar, then is obtained by **multiplying every element** of by .

### Addition and Subtraction of Matrices

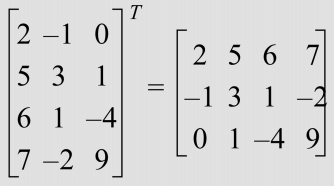
Matrices can only be added or subtracted if they are both of the **same size**.

This is done by adding / subtracting the **corresponding entries** of both matrices.

The resulting matrix is the same size where:

### Transpose of a Matrix

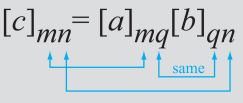
The transpose of a matrix is a matrix with the **rows and columns interchanged**.



### Multiplication of Matrices

The multiplication is defined only if the numbers of columns of equals the number of rows of matrix .

The resulting matrix has the same number of rows as and the same number of columns as .



If is and is then is .

For example:

|  |  |
| --- | --- |
|  |  |

### Special Matrices

|  |  |
| --- | --- |
| **Square Matrix**:  Has the same number of columns as rows.  Entries are known as the **diagonal elements**, and all others are the **off-diagonal elements**. |  |
| **Diagonal Matrix**:  A square matrix with diagonal elements that are non-zero, and off-diagonal elements that are zero. |  |
| **Upper-Triangular Matrix**:  A square matrix whose **sub-diagonal** entries are all zero. |  |
| **Lower-Triangular Matrix**:  A square matrix whose **super-diagonal** entries are all zero. |  |
| **Identity Matrix**:  A square matrix whose diagonal elements are all ones, and off-diagonal entries are all zero.  Any matrix multiplied by the identity matrix remains **unchanged**. |  |
| **Zero Matrix**:  A matrix whose entries are all zero.  Any matrix multiplied by a zero matrix will yield a zero matrix. |  |
| **Symmetric Matrix**:  A square matrix whose lower-diagonal entries mirror its upper-diagonal entries.  The transpose of a symmetric matrix is the matrix itself. |  |

### Inverse of a Matrix

A matrix is invertible (its multiplicative inverse can be found) if there exists a square matrix of the same size such that .

### Determinant of a Matrix

This is a useful quantity to determine if a matrix has an inverse.

Example:

|  |
| --- |
|  |

### Cramer’s Rule and Solution to a System of Linear Equations

Given a set of simultaneous equations with unknowns:

|  |
| --- |
|  |

The system of equations can be written compactly using matrices:



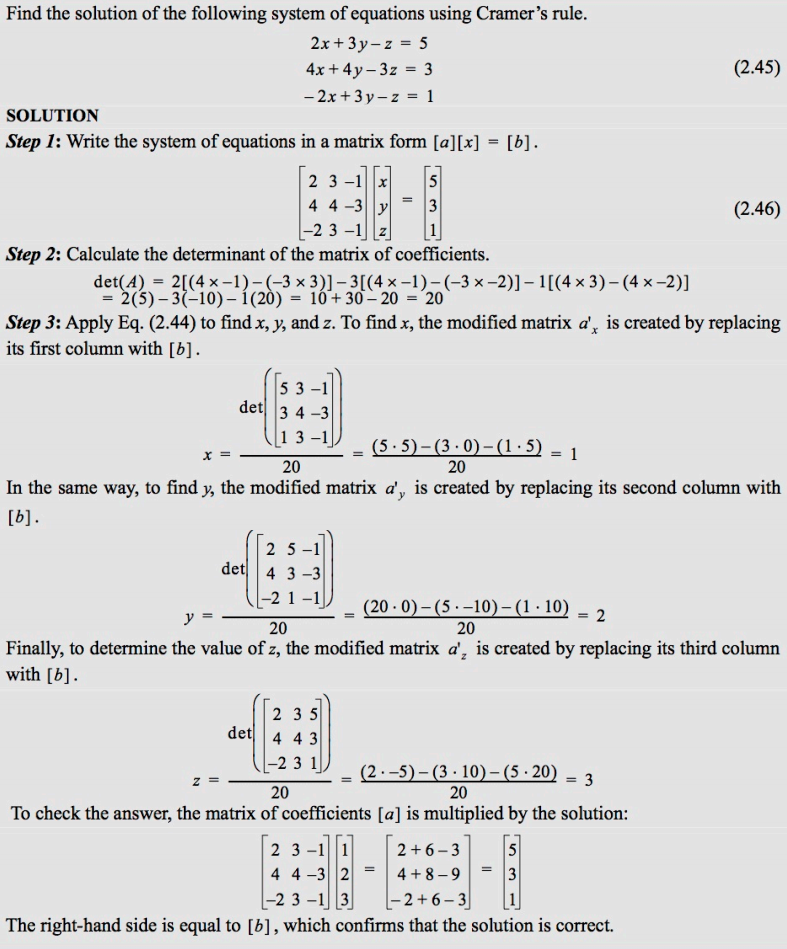
i.e.

Cramer’s Rule states that the solution for , if it exists, is given by:

where is the matrix formed by replacing the th column of the matrix with the column vector .

*Note*: exists

if one or more columns or rows of are **not** linearly independent.

Example:

# Ordinary Differential Equations (ODEs)

An ordinary differential equation is one that contains:

1. **One dependent variable** (e.g. ).
2. **One independent variable** (e.g. ).
3. **Ordinary derivatives** of the dependent variable (as opposed to partial).

Examples:

|  |
| --- |
|  |

An ordinary differential equation is **nonlinear** if any of the **coefficients** are **functions of the dependent variable**.

Examples:

|  |
| --- |
|  |

An ODE is said to be **homogenous** if the **coefficient of the independent variables is zero**. Otherwise, it is said to be nonhomogeneous.

Examples:

|  |
| --- |
| is a **linear**, **nonhomogeneous** ODE.  is a **nonlinear**, **homogeneous** ODE. |

The **order** of an ODE is the **highest derivative** that appears in the equation.

Example:

|  |
| --- |
| is a **second order** ODE. |

### Boundary Conditions & Initial Conditions

To eliminate integration constraints when solving a differential equation we apply **constraints**.

These are referred to as **boundary conditions** (solutions to the DE at particular points) and in the case of **time-dependent DEs**, **initial conditions** (or the solution to the DE at ).

Examples:

|  |
| --- |
| **Nonhomogeneous Linear First-Order ODE**:  Multiply both sides of the equation by the following integrating factor:  This gives:  Integrating both sides gives:  Dividing through by gives:  The constant of integration is determined from a constraint which is problem-dependent. |

|  |
| --- |
| **Homogeneous Linear Second-Order ODE**:  Consider:  where and are constants.  The general solution to this equation is found by substituting .  The resulting equation is called the **characteristic equation**:  The solution is obtained from the quadratic formula:  The general solution is then:  If we get: |

### Functions of Two or More Independent Variables

Consider the function:

This is a function of two variables where is the dependent variable and and are the independent variables.

### The Partial Derivative

For , the first partial derivative of with respect to is denoted by or and is defined by:

if the limit exists.

Also:

The partial derivative with respect to a certain variable (say ) simply by treating (or holding) all other independent variables (say and ) to constants.

Example:

|  |
| --- |
|  |

The **total differential** (or exact differential) of a function of two variables, say is given by:

Using the formula for the total differential we can state:

where and are both functions of the independent variable .

Consider where then:

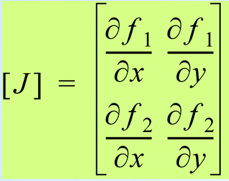
Consider where then:

or

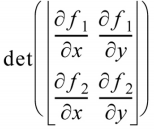
### The Jacobian

The Jacobian is a quantity that arises when solving **systems of nonlinear equations**.

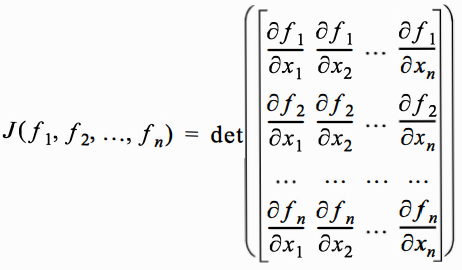
Given and (where and are constants) then the **Jacobian matrix** is:



The **Jacobian determinant** or simply the **Jacobian** is:



In general:



### Taylor Series Expansion of Functions

Used to represent a function as an **infinite power series**.

This can be useful when trying to **integrate** certain non-analytic functions since a truncated series may result in a good analytic **approximation** of the function.

It may also be used in approximating functions to make them more easily manipulated for certain purposes such as **increasing computational speed**.

Given a function that is differentiable times in an interval containing , Taylor’s theorem states that such that:

where called the **remainder** is given by:

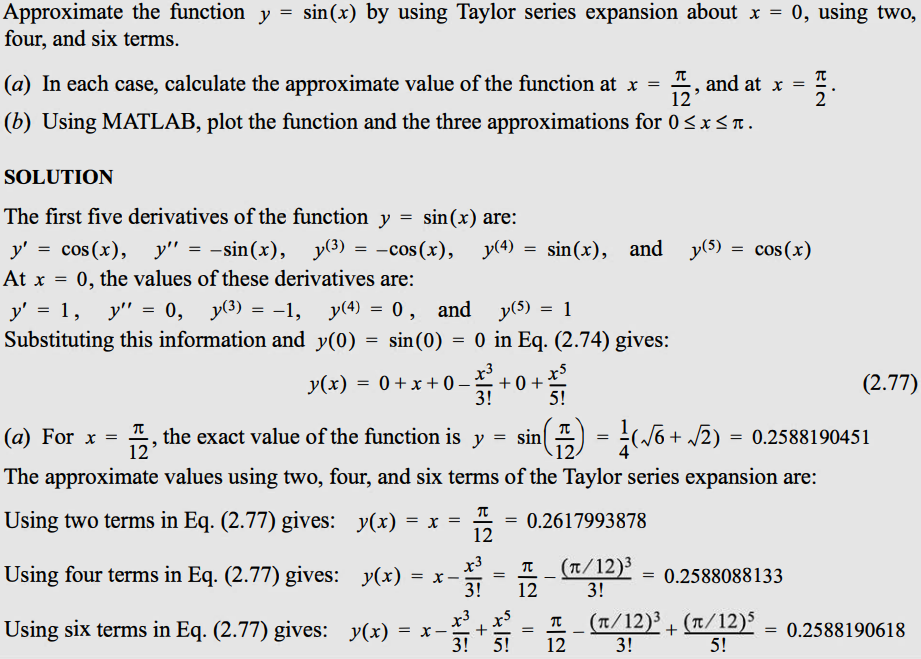
The value of the remainder cannot actually be calculated since the value of is not known.

For , Taylor’s theorem reduces to:

or

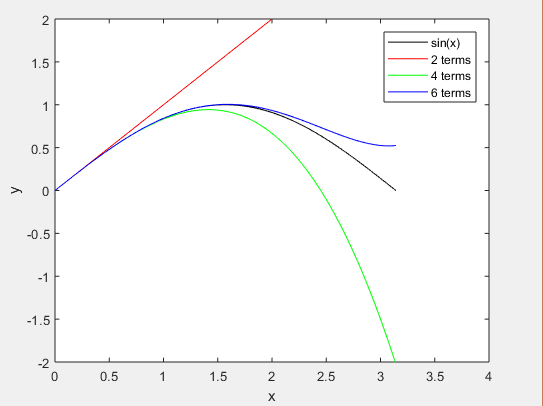
which is a statement of the **mean value theorem for derivatives**.

Example:



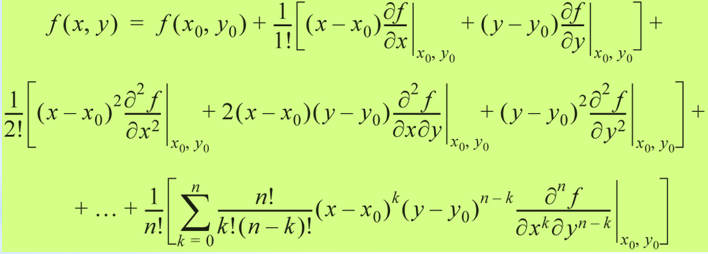
Matlab:

|  |
| --- |
| % Generate a linearly-spaced vector between 0 and pi with 100 elements x = linspace(0, pi, 100); y = sin(x); % Function we are approximating with Taylor series  % Terms of Taylor series y2 = x; % First two terms. y4 = y2 - (x.^3)/factorial(3); % .^ is element-wise power y6 = y4 + (x.^5)/factorial(5); % First six terms.  % Plot graph plot(x, y, 'k', ... % black line  x, y2, 'r', ... % ... allows multi-line functions  x, y4, 'g', ...  x, y6, 'b');   axis([0, 4, -2, 2]); % 0<=x<=4, -2<=y<=2  legend('sin(x)', ... % line 1  '2 terms', ... % line 2  '4 terms', ...  '6 terms');  xlabel('x'); ylabel('y'); |



### Taylor Series for a Function of Two Variables

Taylor’s expansion for a function of two variables is done the same way as for a function of one independent variable, except that the differentiation involves **partial derivatives**.



### Inner Product & Orthogonality

The **inner product** (scalar product or dot product) of two vectors and is denoted:

oror

Say and

Then:

The vectors are said to be **orthogonal** to each other if:

and **parallel** if:

The inner product of two functions and over an interval is given by:

Again, they are said to be orthogonal if:

The **sine** and **cosine** functions are orthogonal at all frequencies:

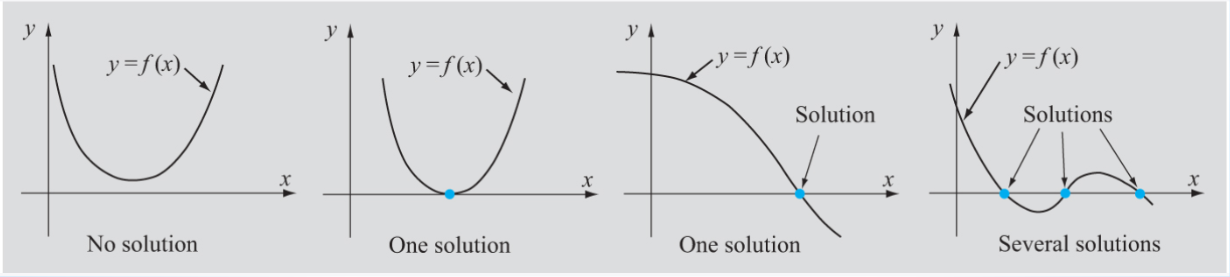
for both and

03 Solving Nonlinear Equations

An equation of one variable can be written in the form:

A solution to the equation (a root of the equation) is a numerical value of that satisfies the equation.

Graphically, this is the point (or points) where the function intersects the -axis.



Sometimes the solutions can be obtained analytically (such as using the formula for solving a quadratic equation), but sometimes there is **no analytic solution** and the equation must be solved **numerically**.

A numerical solution of an equation is a value of that satisfies the equation **approximately**.

When is substituted in the equation, the value of is **close** to zero, but not exactly.

We will always have an error in our approximation. This error **must be quantified** as it tells us the range in which the exact solution lies.

### Estimation of Errors in Numerical Solutions

Let:

1. be the true (exact) solution such that
2. be the numerical solution such that where is a small number.

### True Error

In general is **unknown** so we need to use other measures.

### Tolerance (in )

If it is known that the solution is in the domain then:

plus or minus a tolerance of:

### True Relative Error

but since is unknown we use the estimated relative error.

### Estimated Relative Error

When two numerical estimates for the solution are known (from multiple iterations):

When the numerical solution is close to the true solution then:

i.e. the difference is small compared to the value of .

The estimated relative error is approximately the same as the true relative error.

The methods for solving nonlinear equations are divided into:

1. Bracketing methods
2. Open methods

|  |  |
| --- | --- |
| **Bracketing Methods** | **Open Methods** |
| In **bracketing methods**, the **interval** containing the solution is identified.  The size of the interval is successively **reduced** until the distance between the endpoints is less that the desired accuracy of the solution. | In open methods, an **initial guess** is made at the solution before narrowing down the search through **further guesswork**.  Solutions that have a **higher error** than the current solution are **discarded**. |
| **Always** converge to a solution. | More **efficient**, but **may not** yield a solution. |
| **Examples**:   1. Bisection Method 2. Regula Falsi Method | **Examples**:   1. Newton’s Method 2. Secant Method 3. Fixed-Point Iteration |
|  |  |

All these methods assume that :

1. Is **continuous**.
2. Is **differentiable** over the appropriate interval.
3. Has a **solution**.

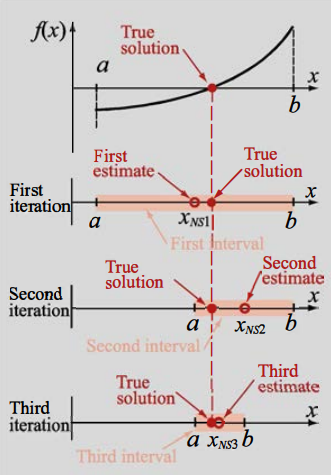
# Bracketing Methods

### 1) The Bisection Method

Used for finding the solution of when it is known that the solution exists within a given interval .

**Algorithm**:

1. Choose the **first interval** by finding points and such that a solution exists in this domain.  
    and have **different signs** so that .
2. The first estimate of the numerical solution is then:
3. Determine whether the true solution is between ( and ) or ( and b).  
   This is done by checking the sign of :
   1. If , then the true solution is between and .
   2. If , then the true solution is between and .
4. Select the subinterval that contains the true solution.  
   *Go to step 2*.



Matlab:

|  |
| --- |
| % Define an anonymous function f(x)=8-4.5(x-sin(x)) f = @ (x) 8 - 4.5\*(x - sin(x));  % Define initial interval a<=x<=3 a = 2; b = 3;  maxIterations = 100; % Perform at most 100 iterations. tolerance = 1e-10; % Terminate after tolerance is reached.  % Make sure that initial interval is valid. if (f(a)\*f(b)>0)  disp("Error, a and b are invalid bounds.")  disp(strcat("f(a)=", num2str(f(a))));  disp(strcat("f(b)=", num2str(f(b)))); end  % Perform Bisection method until either max iterations or tolerance are % reached. for i = 0 : maxIterations  mid = (a+b)/2;    % Stop iterating if within tolerance.  if abs(f(mid)) < tolerance  break;  end;    % Calculate new interval.  sign = f(a)\*f(mid);  if (sign < 0)  b = mid;  else  a = mid;  end end  % Print results of x, f(x) and number of iterations. disp(strcat("x=", num2str(mid))); disp(strcat("f(x)=", num2str(f(mid)))); disp(strcat("iterations=", num2str(i))); |

### 2) The Regula Falsi Method

Approximates the solution by calculating the intercept of the line containing and with the interval where the solution is known to exist.

The equation of the line that connects and is:

()

Its intercept with the -axis is determined by subbing into the previous formula and solving the equation for :

**Algorithm**:

1. Choose the **first interval** by finding points and such that a solution exists in this domain.  
    and have **different signs** so that .
2. The first estimate of the numerical solution is then:
3. Determine whether the true solution is between ( and ) or ( and b).  
   This is done by checking the sign of :
   1. If , then the true solution is between and .
   2. If , then the true solution is between and .
4. Select the subinterval that contains the true solution.  
   *Go to step 2*.



# Open Methods

### 1) Newton’s Method (Newton-Raphson Method)

Approximates the solution initially as the intercept of the tangent to the function, at an initial guess-point, with the -axis.

Subsequent iterations approximate the solution as the intercept of the tangent to the function, at the point defined by the previous estimate, with the -axis.

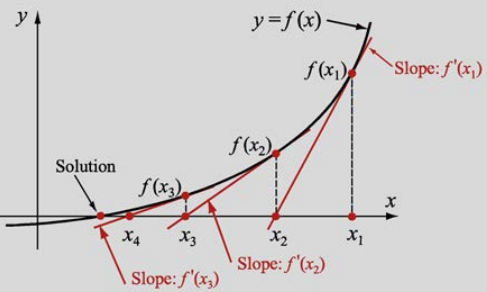
This algorithm does **not** necessarily converge.

**Algorithm**:

1. Choose as a starting point near the solution.
2. For the first iteration, the slope of the tangent at is given by:  
    (rise over run, slope between and )  
     
   Solving for gives:
3. Generalising the previous formula, we can **repeat step 3**:

Newton’s method may not converge, in particular if in the vicinity of the solution.

If and are continuous, at the solution and if the starting value is near the solution, then the method converges.



Matlab:

|  |
| --- |
| % Define an anonymous function f(x)=8-4.5(x-sin(x)) f = @ (x) 8 - 4.5\*(x - sin(x)); % f(x) dfdx = @ (x) -4.5 + 4.5\*cos(x); % f’(x)  % Define starting point (x, f(x)) x = 5;  maxIterations = 100; % Perform at most 100 iterations. tolerance = 1e-10; % Terminate after tolerance is reached.  % Perform Newton's method until either max iterations or tolerance are % reached. for i = 0 : maxIterations  % Calculate numerical solution.  x = x - f(x) / dfdx(x);    % Stop iterating if within tolerance.  if abs(f(x)) < tolerance  break;  end; end  % Print results of x, f(x) and number of iterations. disp(strcat("x=", num2str(x))); disp(strcat("f(x)=", num2str(f(x)))); disp(strcat("iterations=", num2str(i))); |

### 2) The Secant Method

Similar to Newton’s Method, however we use a **secant** to the function instead of a tangent.

This means we initially need **two points** near the solution.

**Algorithm**:

1. Choose , as starting points near the solution.
2. For the first iteration, the slope of the secant line is given by:  
     
     
   Solving for gives:
3. Generalising the previous formula, we can **repeat step 3**:



Matlab:

|  |
| --- |
| % Define an anonymous function f(x)=8-4.5(x-sin(x)) f = @ (x) 8 - 4.5\*(x - sin(x)); % f(x)  % Define starting points (x1, f(x1)), (x2, f(x2)) x1 = 6; x2 = 5;  maxIterations = 100; % Perform at most 100 iterations. tolerance = 1e-10; % Terminate after tolerance is reached.  % Perform Secant method until either max iterations or tolerance are % reached. for i = 0 : maxIterations  % Calculate numerical solution.  new = x2 - (f(x2)\*(x1-x2)) / (f(x1)-f(x2));  x1 = x2;  x2 = new;    % Stop iterating if within tolerance.  if abs(f(x2)) < tolerance  break;  end; end  % Print results of x, f(x) and number of iterations. disp(strcat("x2=", num2str(x2))); disp(strcat("f(x2)=", num2str(f(x2)))); disp(strcat("iterations=", num2str(i))); |

### 3) The Fixed-Point Iteration Method

Used for solving an equation of the form:

This function is written in the form:

.

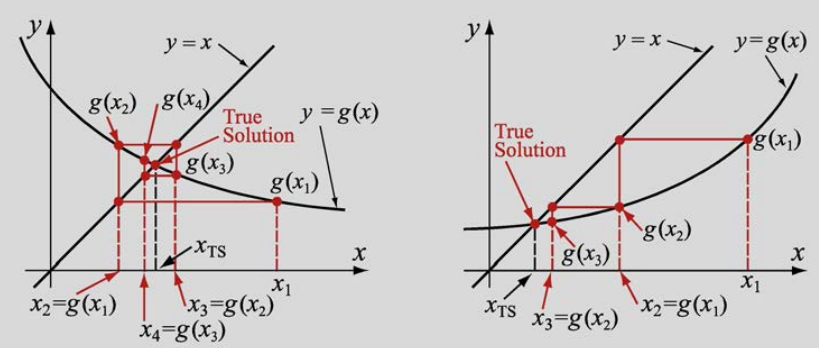
This equation holds true at the **intersection** of the line and the function .

This intersection is called the fixed point.

The problem boils down to obtaining the -coordinate of this point of intersection.

**Algorithm**:

1. Take a value of near the solution (the solution is the point of intersection between the line and the function ) .  
   This is our first estimate of the solution.
2. Find .
3. Use this -ordinate to obtain an -ordinate using the line .  
   *Go to step 2*.



This method is prone to diverge depending on the manner in which is expressed.

If the function is slowly varying such that in the neighbourhood of the solution:

then the process will converge to the correct solution.

**Example**:

Considering

We can write a number of different ways:

Only when expressed as the second option do we get:

Matlab:

|  |
| --- |
| % f(x) = 8-4.5(x-sin(x)) = 0 % rewrite into form x = g(x)  % Define an anonymous function g(x) g = @ (x) sin(x) + 8/4.5;  % Define starting x value x = 5;  maxIterations = 100; % Perform at most 100 iterations. tolerance = 1e-10; % Terminate after tolerance is reached.  % Perform Secant method until either max iterations or tolerance are % reached. for i = 0 : maxIterations  % Calculate numerical solution.  y = g(x);  x = y;    % Stop iterating if within tolerance.  if abs(g(x) - x) < tolerance  break;  end; end  % Print results of x, g(x) and number of iterations. disp(strcat("x=", num2str(x))); disp(strcat("g(x)=", num2str(g(x)))); disp(strcat("iterations=", num2str(i))); |

# Systems of Nonlinear Equations

A system of nonlinear equations consists of **two or more nonlinear equations** that have to be solved simultaneously.

### 1) Newton’s Method (For Systems of Nonlinear Equations)

A system of two equations with two unknowns and can be written as:

The solution process begins by choosing an **estimated solution**  and .

If and are the true (unknown) solutions to the system and are sufficiently close to and , then the values of and at and can be expressed using a **Taylor series expansion** of the functions and about .

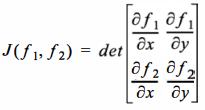
Since and are close to and , approximate values for and can be calculated by **neglecting the higher-order terms**.

Since and , our Taylor-expanded equations can be rewritten as:

where and .

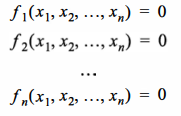
The system can be solved using Cramer’s rule:

where:

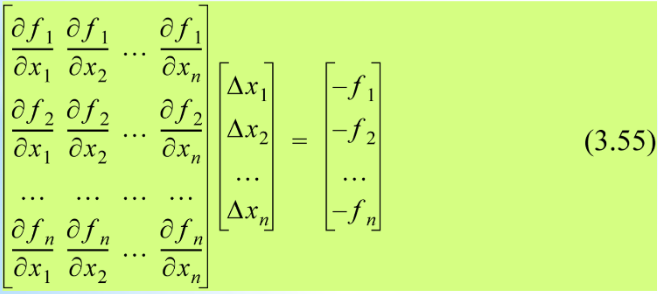


is the Jacobian. Once and are known, the values of and are calculated by:

**Newton’s method** can be generalised for a system of nonlinear equations which have the form:



which gives:

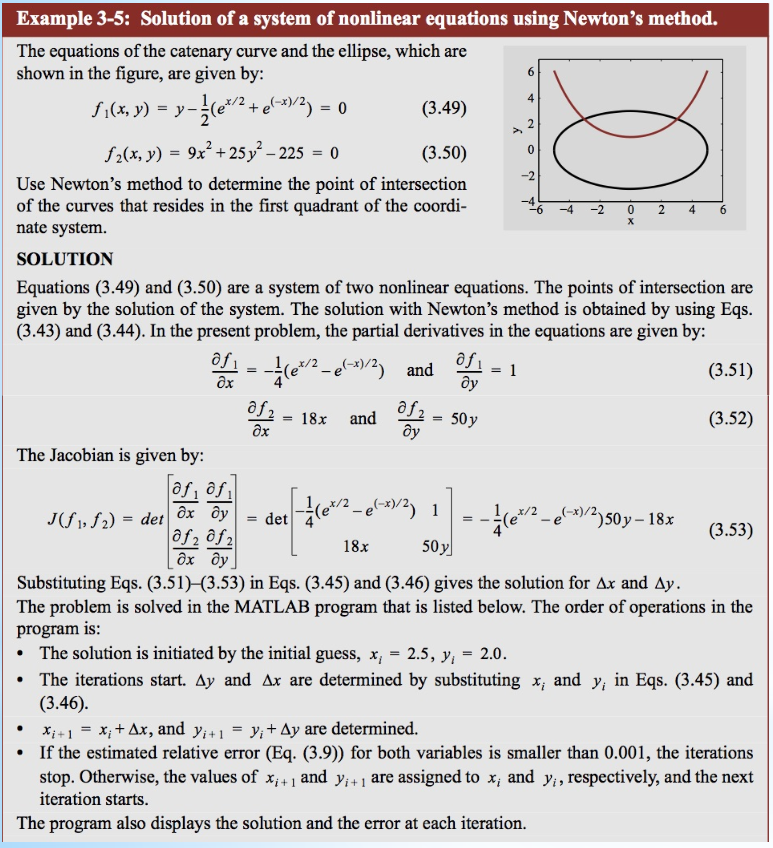


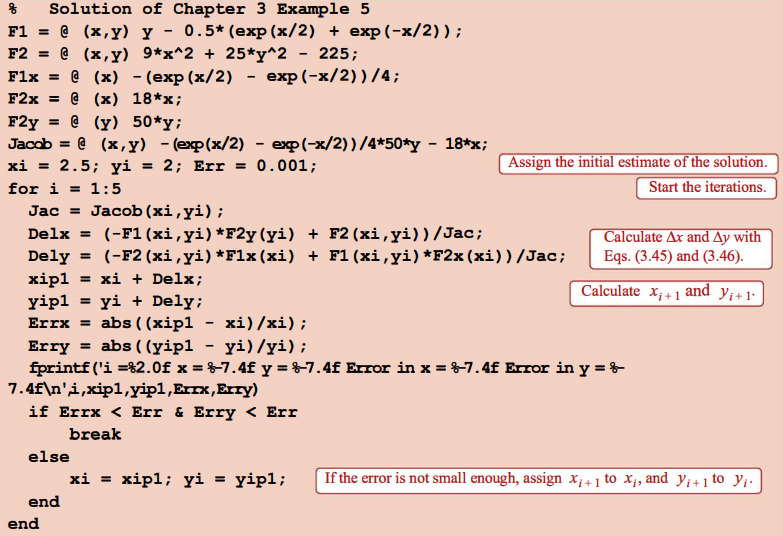
**Algorithm**:

1. Estimate an initial solution:
2. Solve equation 3.55 for
3. Calculate a new estimate of the solution using:
4. Repeat until the desired accuracy is reached.

Newton’s method does not necessarily converge, but when it does it does so quickly.

The Jacobian must be non-zero and the initial estimates must be close to the solution.





### 2) Fixed-Point Method (For Systems of Nonlinear Equations)

**Algorithm**:

1. We begin with:
2. This system can be rewritten in the form:
3. We then choose an initial estimate for the solution which is substituted into the RHS of our rewritten system.  
   The LHS is our new second estimate of the solution.
4. We continue this process until a desired accuracy is reached.

The fixed-point iteration method will converge under the following (but not necessary) conditions:

1. If the functions and their derivatives are continuous in the neighbourhood of the solution.
2. If
3. The initial estimate is close to the solution.

04 Solving a System of Linear Equations

The general form of a system of linear algebraic equations is:

|  |
| --- |
|  |

Which can be written as:



### Methods for Solving Systems of Linear Equations

These equations can be solved with **analytic**, **direct** or **iterative** methods. However, analytic methods such as Kramer’s rule become impossible to compute when the number of equations is large.

|  |  |
| --- | --- |
| **Direct methods** | **Iterative methods** |
| The solution is calculated using an efficient numerical algorithm which performs **arithmetic operations** on equations. | An initial approximate solution is assumed and is then used in an iterative process to obtain **successively more-accurate** results. |

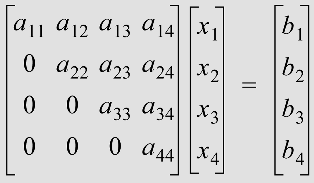
# Direct Methods

In direct methods, the initial system of equations is manipulated into an **equivalent** system of equations that can be easily solved.

Three systems of linear equations that can be **easily solved** are:

1. Upper triangular form
2. Lower triangular form
3. Diagonal form

### Upper-Triangular Form



This equation can be easily solved by **back-substitution**:

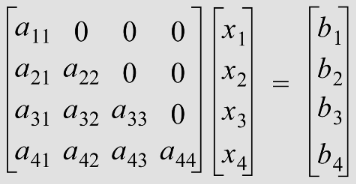
1. We start with the last equation and solve for .
2. Knowing , we can then solve for and so on.

The general form for the solution using back-substitution is:

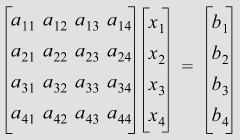
|  |
| --- |
|  |

### Lower-Triangular Form

Likewise, a lower-triangular form can be solved using forward substitution.

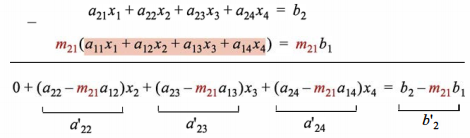
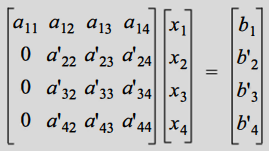


### Gaussian Elimination



Here, the full set of linear equations is manipulated to be in **upper-triangular form**, which can then be solved using back-substitution.

**Algorithm**:

1. We take the **first equation**, called the **pivot equation**.  
    is called the pivot coefficient or **pivot element**.  
   The pivot element is always a diagonal element.  
     
   To eliminate the term , we multiply the pivot equation by .  
   The resulting equation is subtracted from the second equation thus eliminating .  
     
   Note that the **pivot equation remains unchanged**.  
   The second equation now has new coefficients.  
     
   The same process is used to eliminate from all subsequent equations using for the th equation giving:  
   
2. The process is now repeated operating on the third and subsequent equations using the second as the pivot equation and as the pivot element.  
   This eliminates from these equations.  
     
   Continuing in this fashion yields an upper-triangular system which can be solved using back-substitution.

Problems with zero or near-zero pivoting elements can be remedied by **changing the order** of the system of equations such that **no zero or near-zero elements appear on the diagonal**.

This process is known as **pivoting**.

### Gauss-Jordan Elimination

Gauss-Jordan elimination is similar to Gaussian elimination except that:

1. The pivot equation is normalised by **dividing all the terms** in the pivot equation by the **pivot coefficient**. This makes the pivot coefficient equal to 1.
2. The pivot equation is used to eliminate the off-diagonal terms in **all equations**.

The result is a **diagonal matrix** that can be easily solved.

Pivoting applies as with Gaussian elimination.

### LU Decomposition

In this method, we begin with the equation to be solved:

where and are vectors.

Then

where and are lower and upper triangular matrices respectively.

We then write:

We then say:

and

is then first solved for thus enabling to be solved for .

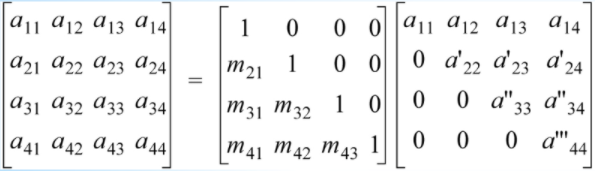
The advantage of this method is that can be easily solved by back and forward substitution.

### LU Decomposition Using The Gaussian Elimination Procedure

When the Gaussian Elimination procedure is applied to a matrix , the elements of the matrices and are actually calculated.

The upper-triangular matrix is obtained at the **end** of the Gaussian Elimination procedure.

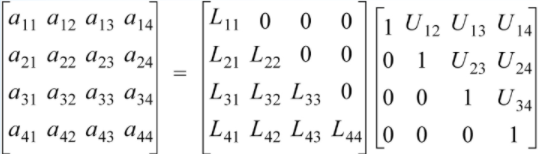
The lower-triangular matrix is a matrix with 1s as its diagonal elements and the multipliers that multiply the pivot equation as its off-diagonal elements.



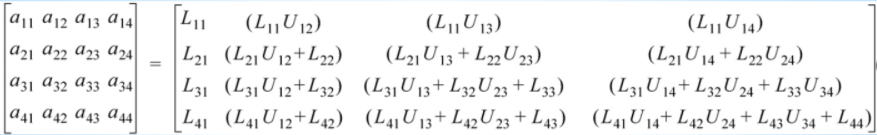
### LU Decomposition Using Crout’s Method

Crout’s method is particularly advantageous since we do not have to use vector at all, which means that we can solve for **multiple** s once and have been determined.

In Crout’s method, the LU decomposition matrix has the form:



Executing the matrix multiplication on the RHS gives:



This equation is then solved by **equating** the elements of the product with that of matrix .

In general:

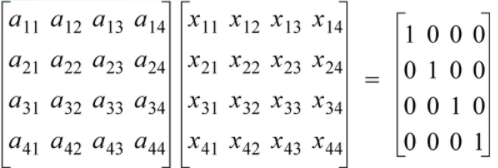
1. Calculate the **first column** of using:
2. Substitute 1s in the **diagonal** of :
3. Calculate the elements of the **first row** of **except** for :
4. Then, calculate:

### Inverse of a Matrix

This is the most advantageous since it can easily be reused.

Consider:

For we have:



We then have:



Solving these four systems of equations gives the four columns of .

Among other methods, this can be done by **Gauss-Jordan elimination** and by **LU decomposition**.

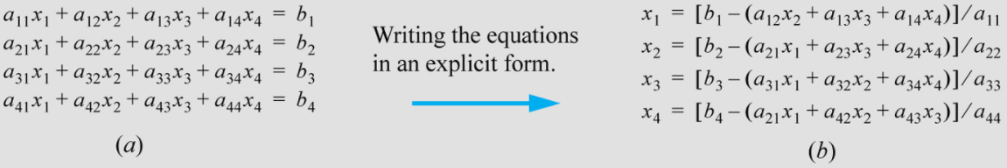
### Inverse of a Matrix by LU Decomposition & Gauss-Jordan Elimination

We obtain each column of the inverse matrix by solving the system of equations.

This can be done by **Gauss-Jordan elimination** or by **LU decomposition** on each of the four equations.

# Iterative Methods

For the Jacobi and Gauss-Seidel iterative methods, the system of equations is written in **explicit form**:



The idea is to **assume an initial solution** for , and then **recursively substitute** these values in to the RHS of the above explicit form and so obtain new estimates for .

**(4.51)**

A sufficient (but not necessary) **condition for convergence** is:

### The Jacobi Iterative Method

An **initial solution** for is chosen. If no information is available then .

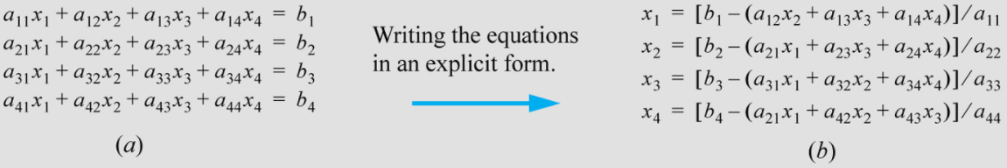
The values for are substituted into the RHS of the explicit representation of the system of equations and a new estimate for is found. This process is repeated recursively.

That is:

The process is continued until the absolute value of the estimated relative error of all unknowns is less than some predetermined value:

**(4.55)**

### The Gauss-Seidel Iterative Method



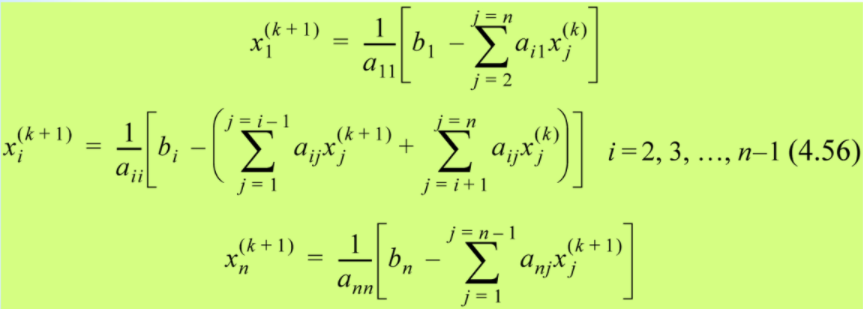
**Algorithm**:

1. **First iteration**:  
   To begin with, all unknowns except for are initialised with some approximate values. These can be zero if approximate values are unknown.  
     
    **(4.51 again)**  
     
   Using these values, with in equation (4.51), we calculate .  
   With in equation (4.51), we calculate a new value for and so on.
2. **Second iteration**:  
   The next iteration uses in equation (4.51), where we calculate a new value for.  
   With in equation (4.51), we calculate a new value for and so on.

These iterations are repeated until:

**(4.55 again)**

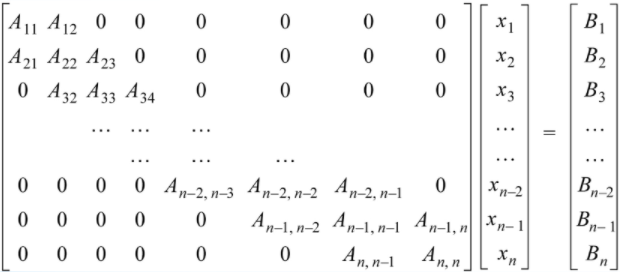
Mathematically, the Gauss-Seidel algorithm can be written:



The Gauss-Seidel algorithm **converges faster** than the Jacobi method.

Like the Jacobi method, the sufficient but not necessary condition for convergence is that is diagonally dominant.

### Tridiagonal Systems of Equations



These can be readily solved using previous methods but many elements of are zero.

This means more storage space and many redundant zero multiplications and additions.

The **Thomas algorithm** is a “common sense” approach to the problem.

Instead of performing Gaussian elimination on as a whole, we store the **diagonal** and **off-diagonal** entries as **separate vectors**. We then apply **Gaussian elimination** as before.

### Error & Residual

Consider the equation

The **true error** is the vector :

*(TS = True Solution and NS = Numerical solution)*

But because cannot be calculated in general, we instead use the **residual error** :

The smaller the elements in the better satisfies the equation. However, this does not necessarily tell us how close is to .

# Norms

The norm of a vector or matrix is a number that in some way indicates collectively the **size** of the vector or matrix **elements**.

There are various definitions for the norm of a vector or matrix .

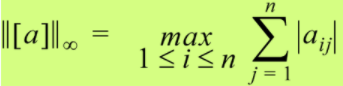
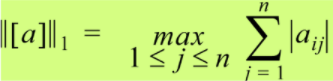
We will concern ourselves with the **Euclidean 2-norm** on this course.

**Vector norms**:

1. The **infinity norm** ():
2. The **1-norm** ():
3. The **Euclidean 2-norm** ():

The Euclidean 2-norm is also referred to as the **magnitude of the vector** .

**Matrix norms**:

1. The **infinity norm** ():  
   
2. The **1-norm** ():  
   
3. The **2-norm** ():  
    where is an **eigenvector** of .
4. The **Euclidean norm**:  
    *(i.e. The root of the sum of squares of the elements.)*

### Properties of Norms

1. The norm of a vector or matrix , denoted , is a **positive quantity** and is equal to zero iff .
2. This means that
4. **The triangle inequality**:

### Using Norms to Determine Error Bounds

It can be shown that:

This useful result gives an upper and lower bound to error relative to the true solution.

### The Condition Number of a Matrix

This tells us **how stable the solution** to a system of equations is.

This number shows how a small perturbation in will affect the solution. If the condition number is much **greater** than 1, then a small perturbation in will **greatly** affect the solution.

In this case, the matrix is said to be **ill-conditioned**.

By contrast, if a small perturbation in does **not** affect the solution greatly, then is said to be **well-conditioned**.

The condition number of a matrix is given by:

For example, take the following system of equations:

|  |
| --- |
|  |

The solution of this system is:

If we now change slightly to , then:

It is clear that is **ill-conditioned**. In fact it has a condition number greater than 1500 regardless of which norm is used.

05 Eigenvalues and Eigenvectors

For a given square matrix , the number is an **eigenvalue** of the matrix if:

**(5.1)**

The vector is a common vector called the **eigenvector associated with the eigenvalue** .

There are usually **more than one** eigenvalue and eigenvector. For an matrix, there are eigenvalues and an infinite number of eigenvectors.

In general:

**(5.2)**

where is some mathematical operator.

Equation (5.2) is a general statement of an eigenvalue problem where is the eigenvalue associated with the operator .

For example, consider:

where:

* is the second derivative with respect to
* is the **eigenfunction**
* is the **eigenvalue** associated with

We will concern ourselves with **matrix operations** only.

### Finding Eigenvalues and Eigenvectors

**(5.1 again)**

Equation (5.1) can be rewritten as:

If has an inverse, then .

If on the other hand has no inverse, then a non-trivial solution for is possible.

Another way of stating this problem is to use **Cramer’s rule**.

has no inverse (it is singular) if:

This equation is known as the **characteristic equation** for .

The characteristic equation for yields a polynomial equation in whose roots are the eigenvalues.

Once the eigenvalues are known then the eigenvectors can be determined by substituting the eigenvalues one at a time into equation (5.1) and solving for .

This process is relatively straightforward for small matrices, but for large matrices we need to use **numerical methods** such as the **power method** or the **QR factorisation method**.

### The Basic Power Method

This method is an iterative procedure for determining the **largest real eigenvalue** and **corresponding eigenvector** of a matrix.

**Proof**:

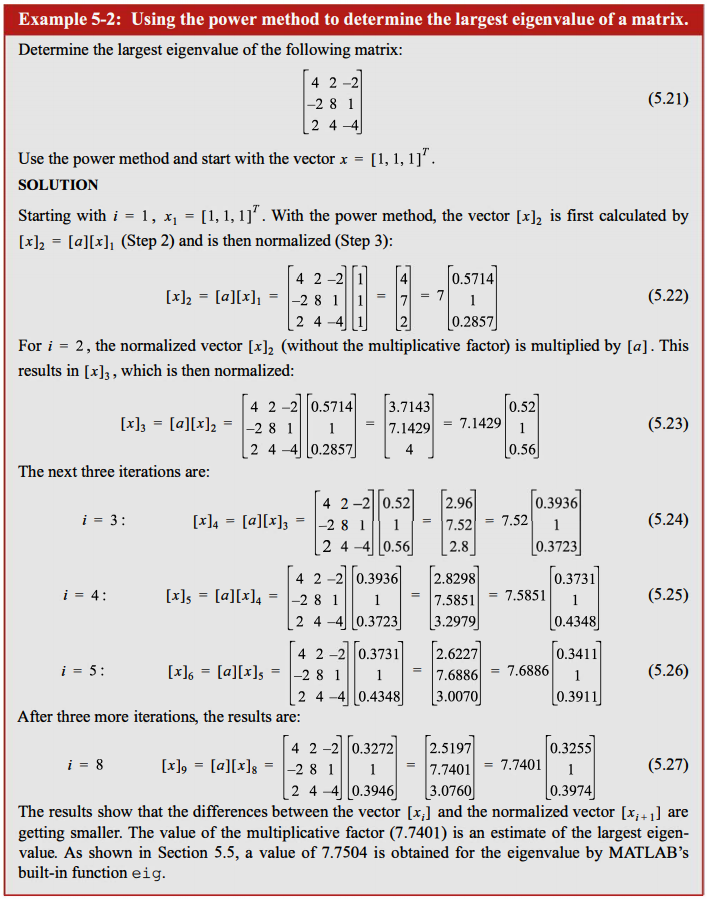
|  |
| --- |
| Let be an matrix with eigenvalues which:   1. Are not necessarily distinct. 2. Satisfy the relations   The eigenvalue , which is the **largest in magnitude**, is known as the **dominant eigenvalue** of the matrix .  Assume that the associated eigenvectors are **linearly independent**, and therefore form a basis for .  It should be noted that **not all matrices** have eigenvalues and eigenvectors which satisfy the conditions we have assumed here.  Let be a **nonzero** element of . Since the eigenvectors of form a basis for , it follows that can be written as a **linear combination** of .  That is, there exists **constants**  such that:  Next, construct the sequence of vectors according to the rule for .  By **direct calculation** we find that:  and **in general**:  In deriving these expressions, we have made repeated use of the relation , which follows from the fact that is an **eigenvector** associated with the **eigenvalue** .  Factoring from the RHS of the equation for gives:  **(1)**  By assumption, for each , so as .  It therefore follows that: |

|  |
| --- |
| Since any nonzero constant times an eigenvector is still an eigenvector associated with the same eigenvalue, we see that the scaled sequence  converges to an eigenvector associated with the **dominant eigenvalue** provided that .  Furthermore, convergence toward the eigenvector is **linear** with asymptotic error constant .  An approximation for the dominant eigenvalue of A can be obtained from the sequence as follows:  Let be an index for which , and consider the ratio of the th element from the vector to the th element from .  By equation (1):  ,  provided that where denotes the th element from the vector .  Hence the ratio converges towards the **dominant eigenvalue**, and the convergence is **linear** with asymptotic rate constant . |

### The Basic Power Method Algorithm

**Algorithm**:

1. Choose a column vector of length . The vector can be any non-zero vector.
2. Multiply this vector by . This gives a column vector
3. Normalise this column vector.
4. *Go to step 2* with
5. Continue with *steps 2 - 4* until a desired accuracy is reached for both the **eigenvalue** and **eigenvector**.



### The Inverse Power Method

This method is used for finding the **smallest** eigenvalue by applying the power method to .

The eigenvalues of are the reciprocals of the eigenvalues of .

This can be seen as follows:

|  |  |
| --- | --- |
| Then:  Hence:  or  This shows that is an eigenvalue of . | *where is the eigenvalue associated with the eigenvector* |

Thus the Inverse Power Method can be used to find the largest eigenvalue of , which is the smallest eigenvalue of .

**Algorithm**:

1. Choose a column vector of length . The vector can be any non-zero vector.
2. Multiply this vector by . This gives a column vector
3. Normalise this column vector and multiply it by again:  
     
     
   To avoid calculating , we use:  
     
     
   This can be solved for using, for example, LU Decomposition.
4. *Go to step 2* with
5. Continue with *steps 2 - 4* until a desired accuracy is reached for both the **eigenvalue** and **eigenvector**.

### The Shifted Power Method

Given , then if is the largest (or smallest) eigenvalue obtained using the Power Method (or Inverse Power Method), then the eigenvalues of a new shifted matrix formed by are:

The reason for this is as follows:

where the s are the eigenvalues of .

But also , so:

where

**Algorithm**:

1. Determine the **largest eigenvalue**  of using the Power Method.
2. Determine the **largest eigenvalue**  of the shifted matrix using the power method.  
   Then from which can be determined.
3. Continue in this fashion until **all the eigenvalues** of are determined.  
   This is done in a total of steps where is an matrix.

The Shifted Power Method is an **inefficient process**. A preferred method for obtaining all eigenvalues of a matrix is the QR Factorisation Method.

### The QR Factorisation Method

Some definitions:

|  |
| --- |
| Two square matrices and are **similar** if:  where is an **invertible matrix**.  and have the **same eigenvalues**. |
| A matrix is **orthogonal** if: |
| A matrix is **symmetric** if: |
| The **eigenvalues** of an **upper-triangular matrix** are its **diagonal elements**. |

QR Factorisation is used to find **all the eigenvalues** of a matrix.

We begin with the matrix , whose eigenvalues are to be determined.

**Iteration 1**:

Let

where:

* is an **orthogonal** matrix
* is an **upper-triangular** matrix

We now define

But

So we can write:

This means that and are similar, thus having the **same eigenvalues**.

**Iteration 2**:

We now write

Then define where so that as in the first iteration.

These iterations continue until the sequence of matrices generates results in an **upper-triangular matrix** whose eigenvalues are its **diagonal elements** which are the same as the eigenvalues of .

We now obtain and ( and for the th iteration).

To do this, we use the **Householder matrix** that has the form:

where is the identity matrix and is an -element column vector given by:

where:

Note that is a **scalar**, but is an matrix.

The **Householder matrix** is both **symmetric** and **orthogonal**. This means that:

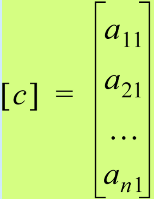
Hence yields a matrix that is similar to .

### QR Factorisation

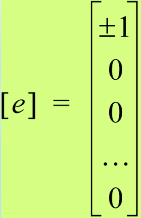
Factoring the matrix into an orthogonal matrix and an upper-triangular matrix such that is done in steps.

**Step 1**:

We choose to be the first column of :



The vector is defined as:



The first element of is if the first element of is positive. Otherwise, it is negative.

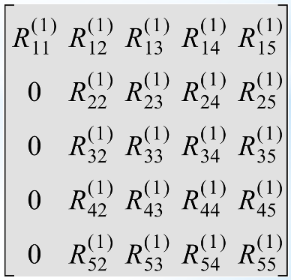
Now we have enough information to calculate .

The matrix is factored into where:

and

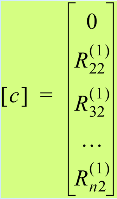
The matrix is orthogonal because is orthogonal.

is a matrix with zeroes as the elements in the first column below the diagonal.

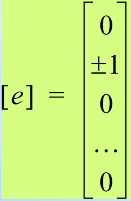


**Step 2**:

The vector is defined as the second column of the matrix with its first element set to zero:



The vector is:



The second element of is if the second element of is positive. Otherwise, it is negative.

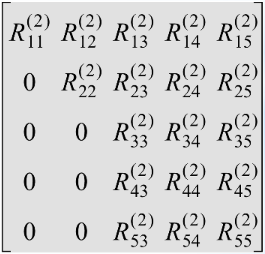
Like before in *step 1*, a new Householder matrix is constructed.

We factor the matrix into where:

and

The matrix is orthogonal because is orthogonal.

is a matrix of zeros as the elements in the second column below the diagonal.



**Step 3**:

Moving to the third column of , the vectors and are defined as:

|  |  |
| --- | --- |
|  |  |

where the sign rule applies as before.

is then:



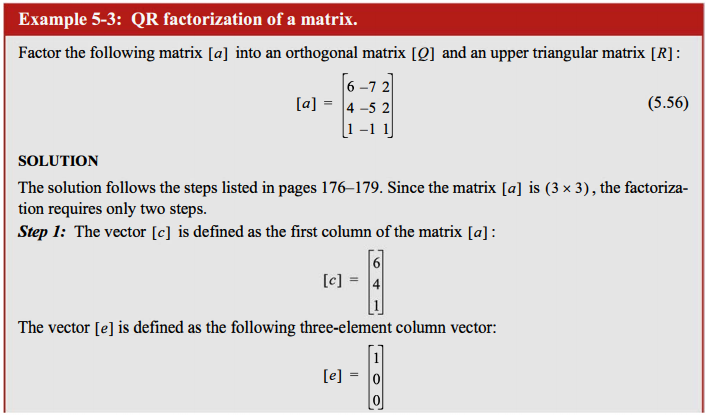
We continue in this latter fashion until is **upper-triangular**.

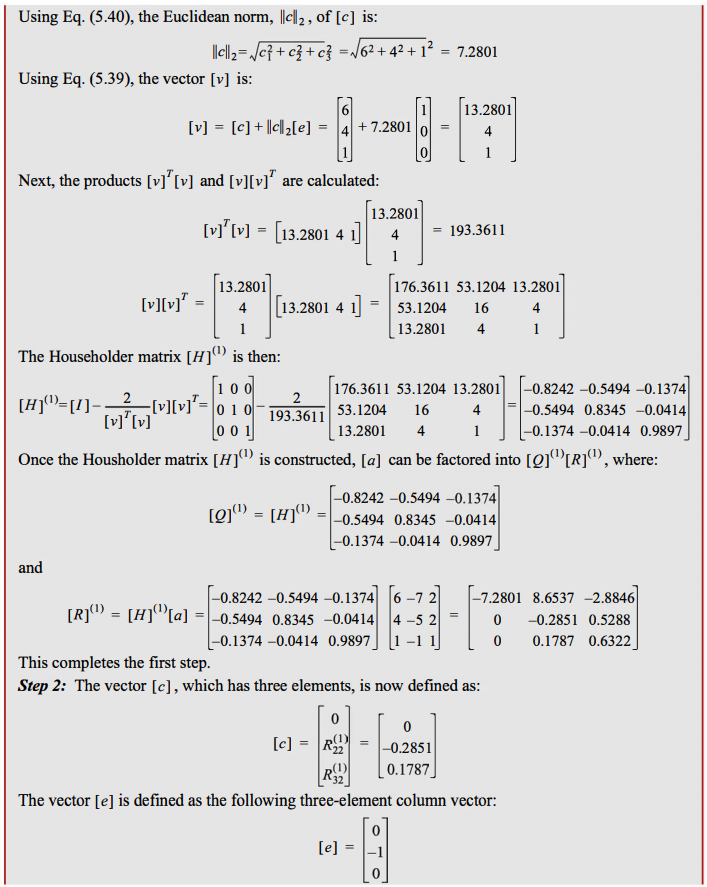
Then we have:

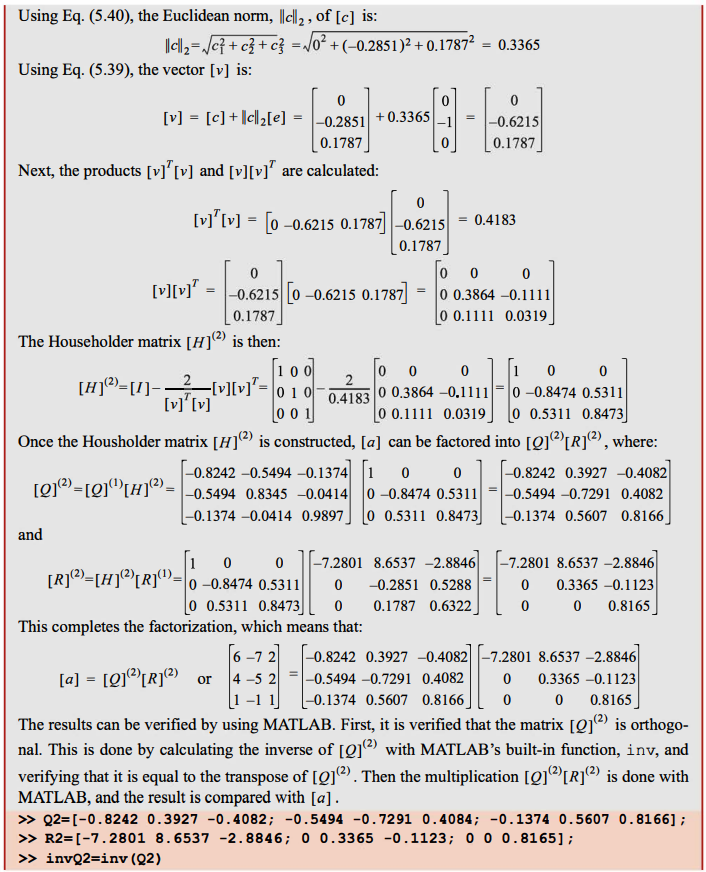
This completes the factorisation of .

**Algorithm**:

1. Factor into an orthogonal matrix and an upper triangular matrix .  
   This is done in steps using a Householder matrix such that:
2. Calculate:
3. *Repeat steps 1 and 2* until is reached.  
   At this point, is upper-triangular and so its eigenvalues (which are the same as those of ) appear on its diagonal.







Google Docs is getting sluggish AF, so the rest of the notes will be in “All Notes 2/2”.