FLIGHT DYNAMICS AND SIMULATION – PART I EXERCISES

AE3-303P

With an introduction into the use of Matlab/Simulink

A.M. Kraeger
A.C. in 't Veld

February 2007

The laboratory aircraft of the faculty of Aerospace Engineering.

1. INTRODUCTION **5**

2. MATLAB **7**

2.1. INTRODUCTION 7
2.2. INTRODUCING MATRICES 8
2.2.1. Input using the keyboard *8*
2.2.2. Input from an external file *9*
2.3. MANIPULATING MATRICES 11
2.3.1. Index notation *11*
2.3.2. Interval notation *12*
2.3.3. Generating matrices *14*
2.3.3.1. Generating matrices from sub-matrices 14
2.3.3.2. Generating matrices using Matlab functions 16
2.4. SCALAR OPERATION 18
2.4.1. Arithmetic operations *18*
Examples 18
Examples 19
2.4.2. Elementary functions *20*
2.4.3. Logical and relational operators *21*
2.5. MATRIX OPERATIONS 22
2.5.1. Mathematical operations *22*
2.5.2. Elementary matrix functions *22*
2.5.3. Logical and relational operators *22*
2.6. POLYNOMIALS 23
2.6.1. Basics *23*
Representation and input of polynomials 23
Polynomial multiplication 23
Polynomial division 23
Polynomial roots 24
Single point evaluation of a polynomial 24
The derivative of a polynomial 25
Partial fraction decomposition 25
2.6.2. Interpolation *26*
2.7. GRAPHICS 28
2.7.1. Two dimensional figures *28*
2.7.1.1. Plotting basics 28
2.7.1.2. Axis scaling 30
2.7.1.3. Multiple figures 31
2.7.1.4. More plot commands 31
2.7.2. Three dimensional figures *32*
2.7.2.1. Functions of two variables 32
2.7.2.2. Parametric representation 35
2.7.2.3. Rotational bodies 36
2.7.2.4. Using color 37
2.8. PROBLEMS 40

3. PROBLEM-SOLVING USING MATLAB **42**

3.1. INTRODUCTION 42
3.2. MATLAB PROGRAMMING STRUCTURE 42
3.3. MATLAB M-FILES 43
3.3.1. Script files *43*
3.3.2. Matlab functions *44*
3.4. PROBLEM SOLVING METHODOLOGY 45
3.4.1. Problem analysis *45*
3.4.2. Problem definition *45*
3.4.3. Logic design *46*
3.4.4. Physical design *46*
3.4.5. Testing and checking *46*

4. EXAMPLES OF SIMULATIONS IN MATLAB **47**

4.1 EXAMPLE 1: OSCILLATIONS OF A PENDULUM 47
4.1.1 Derivation of the linearised equation of motion *47*
4.1.1.1 Motion without friction 48
4.1.1.2 Motion in a medium with viscosity 50
4.1.2 Implementing the linearised equation of motion in Matlab *50*
4.1.2.1 Problem analysis 51
4.1.2.2 Problem definition 51
4.1.2.3 Logic design 51
4.1.2.4 Physical design 51
4.1.2.5 Testing and checking 51
4.1.3 Questions *52*
4.2 EXAMPLE 2: A SYSTEM OF FIRST ORDER LINEAR DIFFERENTIAL EQUATIONS 54
4.3 EXAMPLE 3: THE MOTION OF A POINTMASS IN A FORCE FIELD 57
4.4 EXAMPLE 4: MASS-SPRING SYSTEM 61

5. SIMULINK 64

5.1. INTRODUCTION 64
5.2. SIMULINK EXAMPLE: THE TRAJECTORY OF A METEORITE 64
5.2.1. Derivation of the equations of motion *64*
5.2.2. Building the simulation with Simulink *67*
5.2.3. Running the simulation from a Matlab m-file *70*
5.2.4. Simulation results *73*
5.2.5. The simulation exercise *74*
5.3. SIMULINK EXERCISE: SYMMETRIC MOTION OF A RIGID AIRSHIP 76
REFERENCE FRAMES 76
FORCES 76
MOMENTS 77
5.4. SIMULINK EXERCISE: SYMMETRIC MOTION OF AN AIRCRAFT 80
5.4.1. Reference frames *80*
5.4.2. Forces *80*
5.4.3. Moments *81*
5.4.4. Force and moment coefficients *82*
5.4.5. Questions *85*

6. THE FLIGHT TEST 89

6.1. INTRODUCTION 89
6.2. EXERCISE 1: ELEVATOR TRIM AND ELEVATOR CONTROL FORCE CURVES 89
WEIGHTS 94
STATIONARY MEASUREMENTS 94
SHIFT IN CENTER OF GRAVITY 95
REGISTRATIONS OF MANOEUVRES 95
6.3. EXERCISE 2: LINEAR SIMULATIONS OF THE AIRCRAFT EIGENMOTIONS 96

APPENDIX A: HOW TO USE THE MATLAB NOTEBOOK? 99

APPENDIX B: WHAT IS NICE TO KNOW ABOUT MATLAB? 100

1. INTRODUCTION

The exercise Flight Dynamics and Simulation serves different goals.

First of all, it is taught how to handle the computer program Matlab/Simulink. The combination of Matlab and Simulink forms a true engineering tool when investigating systems. One of the most effective ways to study a system is by simulating it. During this exercise you will learn how to make simple simulations of a variety of systems.

In Aerospace Engineering especially mechanical systems play an important role. After all, the motion of an aircraft or a spacecraft must be regarded as the motion of a (whether or not stiff) body in space under the influence of forces and moments acting on it. That is the reason why in this exercise mainly mechanical systems are simulated.

In the course of the exercise various links to other courses will be made. During the practical some examples are treated that interrelate with the courses Mechanics and Differential Equations. The latter course plays an important role in solving the differential equations that govern the system which has to be simulated. The introduction in the use of Matlab is also convenient for the exercise Numerical Analysis, which is also held in the first semester of the third year, and where Matlab is used as well. The other way around, the knowledge gained in Numerical Analysis may be of use in solving the differential equations that may occur in this exercise.

Above all, the Exercise Flight Dynamics and Simulation serves to illustrate the course Flight Dynamics. Various theoretic subjects that are treated in this course return when making the simulations or when executing the flight test, which is included in this practical. For example, during the flight test the eigenmotions of the Cessna Citation II research aircraft are demonstrated in practice and the aircraft responses to certain control surface deflections are measured and recorded.

The aircraft responses, which are measured during real flight, are compared to the responses that follow from the simulations. In this manner the simulations can be directly validated against the real flight, which is a valuable component of this exercise.

The exercise consists of fourteen practical mornings or afternoons, which will be preceded by a general introduction where a new subject will be introduced, an example will be treated, or a homework assignment will be discussed. During the practical mornings or afternoons, which will be held in the project rooms, programming and simulating with Matlab/Simulink will be practised. Examples from practical situations will be used, for example the processing of previously recorded measurements from real flights with the laboratory aircraft. During the practical mornings or afternoons, support from student-assistants is available, one on each floor.

In this manual in chapter 2 and 3 the basics of using Matlab are dealt with. Subsequently, chapter 4 comes up with building simulations and applying them to the flight test. Chapter 5 deals with the use of Simulink in simulations.

The laboratory aircraft of the Faculty of Aerospace Engineering fitted out as "flying classroom".

2. MATLAB

2.1. Introduction

The work of an engineer typically consists of finding solutions for every day problems in all kinds of technical areas, using knowledge from different fields of science such as mathematics, physics, chemistry and computer science. Because of the ever increasing availability of computing power, solving these problems is done more and more using computer simulation.

Some of the advantages of performing simulations are the low risk and low cost involved as compared to performing physical experiments. Another major advantage is the physical insight one can gain in the behavior of the system under different circumstances and different values of the characteristic parameters of the system. This can already play a major role during the design phase of a system, as experiments are already possible using the simulation before the system has been actually built. Good examples are e.g. the design of an aircraft or an autopilot before the first sheet of aluminum has been cut in the factory.

Sooner or later these simulations will have to be validated against the real world, to show the applicability of the simulation. This is exactly what we will do during the course of Flight Dynamics and Simulation Flight Test. We will start by investigating the responses of a mathematical model of the Cessna Citation to deflections of its control surfaces. Later we will validate the model by taking measurements during an actual flight. The faculty's research aircraft has been equipped with a measurement system that accurately measures the aircraft's motion and steer inputs.

Of course simulations can be performed in a number of ways, but the use of computers is the norm nowadays. We will make use of the Matlab/Simulink package, as this a widely used engineering tool and has proven to be a very powerful tool for solving different kinds of engineering problems.

This chapter treats the most elementary Matlab commands. The different examples and exercises can be used to practice these commands during the first couple of lectures. It is important to first learn how to use Matlab and become proficient in Matlab basics, before we can start building the simulations we will need later on.

Matlab commands are displayed in **`green`** and Matlab output in `blue`.

2.2. INTRODUCING MATRICES

Matlab's basic working element is the matrix. A scalar is regarded to be a 1x1 matrix, so scalar operations within Matlab are treated as a special case of general matrix manipulation. In the same spirit vectors are defined as 1xn or nx1 matrices.

This paragraph describes two ways to introduce matrices into the Matlab *workspace*: entry by using the keyboard and by reading an external file in which the elements of the matrix have been stored.

2.2.1. Input using the keyboard

The input is straightforward. The Matlab command:

`A = 3` <Enter> results in the following output on the screen:

```
A =
     3
```

The value 3 has now been stored in variable A. This variable will remain in the memory of the computer until the explicit command is given to remove it from memory. The command `clear A` is used for this. The command `clear A B C` removes variables A, B and C from the working memory of the computer. `clear` by itself will remove *all* the variables from memory.

Entering matrices is a straightforward matter in Matlab. The following rules should be kept in mind:

- The elements of a row have to be separated either by a space or a comma
- The last element of each row has to be followed by a semi-colon
- The list of all elements has to be enclosed by square brackets

Example: **`B = [1 2 3 ; 4 5 6 ; 7 8 9]`** <Enter> produces the following result:

```
B =
     1     2     3
     4     5     6
     7     8     9
```

Matrix B has now been stored in the working memory and will remain available for calculations until it is erased using the command `clear.`

Now that it is clear how to define a matrix, the creation of vectors in Matlab is very straightforward. A vector is a 1xn or nx1 matrix. A 1xn row-vector can easily be transformed into an nx1 column-vector by using the transpose function. The apostrophe ` is used for this.

Example: **`C = [16 9 3 5]`** <Enter> yields the 1x4 row vector:

```
C =
    16     9     3     5
```

This can be turned into a column vector:

`C_t = C'` <Enter>

```
C_t =
    16
     9
     3
     5
```

Using the command `who` an overview can be obtained of all the variables currently stored in the working memory. A more elaborate version of `who` is `whos` which generates more information on each variable. Apart from the variable names, `whos` also gives the dimensions (e.g. 1x3, 1x1, 4x8), the number of bytes the variable is occupying and the variable type (e.g. double array, character array).

```
who

Your variables are:

A          B          C          C_t

whos

  Name        Size            Bytes  Class

  A           1x1                 8  double array
  B           3x3                72  double array
  C           1x4                32  double array
  C_t         4x1                32  double array

Grand total is 18 elements using 144 bytes
```

2.2.2. Input from an external file

The data that needs to be analyzed using Matlab, will often have been generated by other sources than Matlab, such as measurements or a computer program written in e.g. Pascal, FORTRAN, C or C++. This will also be the case during the flight test that is part of this project. During the flight a number of parameters will be sampled and stored on an on-board hard disk. To be able to analyze and process these measurements using Matlab, all participants will receive a copy of the measurement data on a floppy disk or similar means, after the flight. The following paragraphs explain the method to import this data into Matlab.

First of all the working directory needs to be set to the directory containing the data file. The command `cd` is available to change directories. The command `cd` without any arguments returns the current working directory. The default directory when Matlab is launched for the first time is P:\matlab\bin. The data file in this case will be in the directory P:\ae3-303P\Data, so the working directory has to be changed:

```
cd

C:\MATLAB5\bin

cd P:\ae3-303P\Data

cd

P:\ae3-303P\Data

dir

.              Takeoff.dat    Landing.dat

load Takeoff.dat
```

```
whos

  Name           Size           Bytes  Class

  A              1x1                8  double array
  B              3x3               72  double array
  C              1x4               32  double array
  C_t            4x1               32  double array
  Takeoff      300x16           38400  double array

Grand total is 4818 elements using 38544 bytes
```

This data contains a number of parameters that have been measured during a take-off of faculty's Cessna Citation, the PH-LAB. The matrix *Takeoff* contains all this data and is now available in the working memory. Using the command `whos`, we find that this matrix consists of 16 columns and 300 rows. The columns contain the registered parameters:

1. pressure altitude in feet
2. static airpressure in Pascal
3. true airspeed in knots
4. static air temperature in degrees Celsius
5. time in hours, minutes and seconds in UTC acoording to the Flight Management Computer
6. latitude of the aircraft in degrees
7. longitude of the aircraft in degrees
8. groundspeed in knots
9. true heading of the aircraft in degrees
10. windspeed in knots
11. wind direction in degrees with regards to the magnetic North
12. altitude above mean sea level according to the Global Positioning System (GPS)
13. time in hours, minutes and seconds UTC according to the GPS
14. angle of attack in degrees
15. air pressure inside the cabin in Pascal
16. registration date

These 16 parameters have been sampled and stored at 1 second intervals. The matrix Takeoff consists of 300 rows, which means that each parameter has been measured 300 times during a total of 299 seconds (5 minutes). Afterwards a number of manipulations can be performed on these data.

2.3. Manipulating matrices

Matlab offers many different ways of manipulating matrices. Examples include the automatic generation of matrices and vectors, building matrices from sub-matrices en separating rows, columns or sub-matrices from existing matrices. This section deals with a number of useful Matlab commands for manipulating matrices.

2.3.1. Index notation

The elements of a matrix can be accessed using indices. It is possible to take from matrix A the element on the i^th row and j^th column and store it in a variable a. Take for example the following matrix A:

```
A = [11 12 13; 21 22 23; 31 32 33]

A =
    11    12    13
    21    22    23
    31    32    33
```

Elements a12, a22 and a23 can be lifted out of matrix A as follows:

```
a12 = A(1,2)

a12 =
    12

a22 = A(2,2)

a22 =
    22

a23 = A(2,3)

a23 =
    23
```

Suppose we have defined vector v:

```
v = [4/7 8/3 2/9]

v =
    0.5714    2.6667    0.2222
```

Suppose we need to add a new element to this vector v, the following command will take care of that:

```
v(5) = 10

v =
    0.5714    2.6667    0.2222         0   10.0000
```

All this shows that it is not necessary to define the dimensions of a matrix beforehand and that it is possible to change the matrix dimensions at any time. Note that Matlab automatically assigned a value of zero to the fourth element, as there was no information on v(4).

Watch what happens when you execute the nest command:

```
v(1) = 6/5
```

```
v =
    1.2000    2.6667    0.2222         0   10.0000
```

The dimension of v has not changed because v(1) already existed, but the value of the first element did change.

Last example:

```
v(6) = v(4)

v =
    1.2000    2.6667    0.2222         0   10.0000         0
```

Suppose you made a mistake and you need to remove the fourth element of v, which has gotten the value of zero. This is how you do it:

```
v(4) = []

v =
    1.2000    2.6667    0.2222   10.0000         0
```

The command `v = []` removes all the elements of v, resulting in an empty vector v:

```
v = []

v =
     []
```

However, vector v has not been removed from the workspace as can be checked using the command `who`:

```
who

Your variables are:

A         v
```

As we have seen earlier, we can remove the variable v from Matlab's workspace using `clear v`.

2.3.2. Interval notation

The interval notation is an extension on the index notation. Within Matlab an interval is a row of numbers, defined by a starting element, an end element and a step size. Using interval notation offers a very efficient way of referring to vectors and matrices. The colon ":" plays an important role as we will show next.

The general expression for an interval is:

$x = x_{min} : step : x_{max}$

x_{min} represents the lower boundary of the interval and x_{max} the upper boundary. The step size is given by the variable `step`. The next expressions define row vectors using interval notation:

```
1:1:10

ans =
     1     2     3     4     5     6     7     8     9    10
```

```
1:2:10

ans =
     1     3     5     7     9

1:8

ans =
     1     2     3     4     5     6     7     8

108:-20:-17

ans =
   108    88    68    48    28     8   -12

17:-1:120

ans =
   Empty matrix: 1-by-0

0:pi/4:2*pi

ans =
  Columns 1 through 7
         0    0.7854    1.5708    2.3562    3.1416    3.9270    4.7124
  Columns 8 through 9
    5.4978    6.2832
```

This last expression uses the Matlab constant `pi` which has a value of π. We can see that negative step sizes are allowed and that when the step size is omitted Matlab automatically assumes a step size of 1.

Not only can we use interval notation to define vectors, but also to manipulate matrices. In fact, one of the most powerful features of Matlab is its ability to separate sub-matrices from matrices. Earlier we have seen how to lift a single element out of a matrix. The expression `aij = A(i,j)` assigns the value of the element on the ith row and the jth column of matrix A to the variable aij.

If i and j are vectors, then the column vector `A(i,1)` ill contain the elements `A(i(1),1),A(i(2),1),...,A(i(n),1)`. The expression `A(i,j)` will contain the submatrix of A of which the rows are given by the elements of vector I and the columns are given by the elements of vector j.

To illustrate this we define a 5x6 matrix A.

```
A = [11 12 13 14 15 16
21 22 23 24 25 26
31 32 33 34 35 36
41 42 43 44 45 46
51 52 53 54 55 56]

A =
    11    12    13    14    15    16
    21    22    23    24    25    26
    31    32    33    34    35    36
    41    42    43    44    45    46
    51    52    53    54    55    56
```

We can separate the first three rows and the first three columns as follows:

```
A(1:3,1:3)

ans =
    11    12    13
    21    22    23
    31    32    33

A(3:4,4:6)

ans =
    34    35    36
    44    45    46

A(1:2:5,1:5)

ans =
    11    12    13    14    15
    31    32    33    34    35
    51    52    53    54    55

A(5:-1:1,5:-1:1)

ans =
    55    54    53    52    51
    45    44    43    42    41
    35    34    33    32    31
    25    24    23    22    21
    15    14    13    12    11
```

To indicate that all rows or columns should be included in the sub-matrix, the colon ":" can be used without beginning or end element. Example:

```
A(2:4,:)

ans =
    21    22    23    24    25    26
    31    32    33    34    35    36
    41    42    43    44    45    46
```

2.3.3. Generating matrices

2.3.3.1. Generating matrices from sub-matrices

Not only is it possible to build a matrix using its elements as we have seen in paragraph 2.2.1., we can also build it from sub-matrices. Suppose matrices A and B are defined by specifying their respective elements:

```
A = [1 2; 3 4]

A =
     1     2
     3     4

B = [5 6; 7 8]
```

```
B =
     5     6
     7     8
```

Matrices A and B can be used to construct other matrices:

```
C = [A B]

C =
     1     2     5     6
     3     4     7     8

D = [A; B]

D =
     1     2
     3     4
     5     6
     7     8

E = [A B-4; A' B']

E =
     1     2     1     2
     3     4     3     4
     1     3     5     7
     2     4     6     8
```

Should you loose track of the size of the matrix after a number of manipulations, the command `size` can be used to retrieve the matrix dimensions.

```
[m, n] = size(E)

m =
     4
n =
     4
```

The command `length` returns either the number of rows or the number of columns of the matrix, whichever is larger.
Suppose we have a vector v:

```
v = [1 2 3 4 5 6 7]

v =
     1     2     3     4     5     6     7

length(v)

ans =
     7

length(C)

ans =
     4

lD = length(D)

lD =
     4
```

2.3.3.2. Generating matrices using Matlab functions

Matlab offers an array of possibilities to generate matrices. This paragraph treats a number of them.

Zeros creates a matrix filled with zeros

```
Z = zeros(3,5)

Z =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Ones creates a matrix filled with ones

```
O = ones(2,8)

O =
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
```

Eye creates an identity matrix

```
E = eye(3,3)

E =
     1     0     0
     0     1     0
     0     0     1
```

Randn creates a matrix filled with normally distributed random numbers.

```
R = randn(5,5)

R =
   -0.4326    1.1909   -0.1867    0.1139    0.2944
   -1.6656    1.1892    0.7258    1.0668   -1.3362
    0.1253   -0.0376   -0.5883    0.0593    0.7143
    0.2877    0.3273    2.1832   -0.0956    1.6236
   -1.1465    0.1746   -0.1364   -0.8323   -0.6918
```

Diag creates a diagonal matrix using the elements of a known vector.

```
v = [3 2 1]

v =
     3     2     1

D0 = diag(v)

D0 =
     3     0     0
     0     2     0
     0     0     1
```

```
D1 = diag(v,1)

D1 =
     0     3     0     0
     0     0     2     0
     0     0     0     1
     0     0     0     0

D2 = diag(v,2)

D2 =
     0     0     3     0     0
     0     0     0     2     0
     0     0     0     0     1
     0     0     0     0     0
     0     0     0     0     0
```

`diag` can also be used to retrieve the diagonal elements of a matrix:

```
diag(D0)'

ans =
     3     2     1

diag(D1)'

ans =
     0     0     0     0

diag(D2)'

ans =
     0     0     0     0     0
```

Find out for yourself what the effect is of the functions `rot90`, `fliplr`, `flipud`, `flipdim` and `reshape`. During your investigations, you will find Matlab's `help` function to be very valuable.

2.4. SCALAR OPERATION

So far methods to import and manipulate data have been treated. In this paragraph arithmetic and logical scalar operators will be introduced. These operators can also be used on matrices, in which case the operations will be done on the elements of the matrices. The next paragraph will deal with matrix operations that are not executed element wise, such as matrix multiplication.

2.4.1. Arithmetic operations

Within Matlab the following operators are available:

+	addition
-	subtraction
*	multiplication
/	division
^	power
sqrt	square root
\	left division

Examples

Two scalars a and b and two vectors x and y are defined:

```
a = 3
b = 5
x = [1 0 2]
y = [3 2 4]

a =
     3
b =
     5
x =
     1     0     2
y =
     3     2     4
```

Investigate the value of:

```
a*b

ans =
    15

a/b

ans =
    0.6000

a\b

ans =
    1.6667

b^a

ans =
   125
```

```
x+y

ans =
     4     2     6

a*x

ans =
     3     0     6

a*x+b*y

ans =
    18    10    26

sqrt(b)

ans =
    2.2361

sqrt(y)

ans =
    1.7321    1.4142    2.0000
```

Notice that some operators in these examples can also be used on matrices. In these cases the (scalar) operator is applied on the elements of matrices, preserving the scalar character of the operator.

However, the operators mentioned above do also conform to the rules of linear algebra. This can lead to some confusion when these operators are applied to vectors. For instance, the product of scalar a and vector x is a vector where each element of x has been multiplied by a. But what happens if vector x is multiplied by vector y? As will be shown in the next paragraph, the result of such a multiplication is the inner product (dot product) of these vectors.

When the required result is a vector whose elements are the product of the respective elements of both vectors, the dot-operator "." can be used. The dot can be used in combination with the operators *,/,\ and ^.

Examples

```
[2 3 4]*[4 2 1]'

ans =
    18

[2 3 4].*[4 2 1]

ans =
     8     6     4

[2 3 4].^3

ans =
     8    27    64

[2 3 4].^[4 2 1]

ans =
    16      9      4
```

For exponential and logarithmic calculations the following operators are available:

`pow2`	base 2 exponential power
`exp`	base e exponential power
`log`	natural logarithm
`log2`	base 2 logarithm
`log10`	base 10 logarithm

Finally a number of trigonometric functions are worth mentioning:

`sin`	sine
`cos`	cosine
`tan`	tangent
`asin`	arcsine
`acos`	arccosine
`atan`	arctangent
`sinh`	hyperbolic sine
`cosh`	hyperbolic cosine
`tanh`	hyperbolic tangent
`asinh`	inverse hyperbolic sine
`acosh`	inverse hyperbolic cosine
`atanh`	inverse hyperbolic tangent

2.4.2. Elementary functions

Of the many elementary functions available in Matlab, a number are mentioned here. If the description is not totally clear, investigate these functions yourself using Matlab.

`round`	round to the nearest integer
`fix`	round to the nearest integer, toward zero
`floor`	round to the smaller nearest integer
`ceil`	round to the bigger nearest integer
`rem`	remainder after division
`rats`	approximates a real number by a rational number
`gcd`	greatest common divisor
`lcm`	least common multiple
`real`	real part of a complex number
`imag`	imaginary part of a complex number
`conj`	complex conjugate of a complex number
`abs`	absolute value or complex magnitude
`angle`	phase angle

Within a Matlab a number of coordinate transformations have been defined:

`cart2pol`	Cartesian coordinates to polar or cylindrical coordinates
`pol2cart`	polar or cylindrical coordinates to Cartesian coordinates
`cart2sph`	Cartesian coordinates to spherical coordinates
`sph2cart`	spherical coordinates to Cartesian coordinates

2.4.3. Logical and relational operators

Matlab knows the following logical operators:

&	AND
|	OR
`xor`	EXCLUSIVE OR
~	NOT

Relational operators:

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
~=	not equal

The result FALSE is indicated by 0, the result TRUE by 1. The operators can also be used on matrices.

Examples:

```
1+1 == 2

ans =
     1

2+2 ~= 4

ans =
     0

A = [1 2 3 4; 5 6 7 8]

A =
     1     2     3     4
     5     6     7     8

P = (rem(A,2)~=0)

P =
     1     0     1     0
     1     0     1     0
```

In the last example all uneven elements of matrix A are located. It is interesting to see that in the expression a matrix, `rem(A,2)`, is compared to a scalar. To be able to understand this, the next paragraph will explain the way Matlab treats matrices.

2.5. MATRIX OPERATIONS

As mentioned before, the basic element Matlab uses is the matrix. A scalar is nothing more than a 1x1 matrix. This implies that the scalar operations from the previous paragraph can be readily expanded into matrix operations. However, matrix operations are not always the same as scalar operations on the element of a matrix. The next paragraph sows this.

2.5.1. Mathematical operations

The most important arithmetic operators are `+`, `-` , `*`, `/`, `\` and `^`.

Matrix addition and subtraction is performed elementwise. Naturally the dimensions of the matrices must match. There is only one exception to this rule: within Matlab the addition or subtraction of a matrix A and a scalar a is also defined. In this case the scalar is transformed into a matrix of suitable dimensions of which all elements are equal to a. So `A + a` is calculated as `A + a*ones(size(A))`.

Matrix multiplication is not an elementwise operation. The * sign represents matrix multiplication as defined in linear algebra. Again the user must assure that the dimensions of the matrices match. The number of columns of the first matrix must equal the number of rows of the second matrix. The product of a scalar and a matrix is defined as a matrix of the same size whose elements are equal to the product of the scalar and the elements of the original matrix. So `a*A` is calculated as `a*ones(size(A)).*A`.

There are two different symbols for matrix "division": / and `\`:

`X = A\B` calculates $X = A^{-1} \cdot B$ solving the system $A \cdot X = B$, while
`X = A/B` calculates $X = A \cdot B^{-1}$ solving the system $X \cdot B = A$.

Exponentiation is nothing more than repeated matrix multiplication. Raising a matrix to the power –1 has the same result as the calculation of the inverse of the matrix, so `A^-1 = inverse(A)`.

2.5.2. Elementary matrix functions

Matlab knows many matrix functions. Some of the more useful are: `max`, `min`, `sort`, `sum`, `mean`, `rank`, `det`, `inv`, `poly`, `trace`, `norm`.

Also a number of exponential functions are available, e.g.: `expm`, `logm`, `sqrtm`, `expm1`, `expm2`, `expm3`.

Find the purpose of a number of these functions, using for instance Matlab's help function.

2.5.3. Logical and relational operators

A whole array of logical and relational operators is at the disposal of the Matlab user. Some examples include: `any`, `all`, `find`, `exist`, `isnan`, `finite`, `isinf`, `isempty`, `isieee`, `isstr`. Investigate the function of number of these Matlab commands.

2.6. Polynomials

It is possible to work with polynomials in Matlab. Polynomials play an important part in describing systems in terms of transfer functions. MATLAB represents polynomials as row vectors containing coefficients ordered by descending powers.

2.6.1. Basics

Representation and input of polynomials

First of all the entry of polynomials in Matlab is explained. Suppose we have two polynomials a(s) and b(s):

$a(s) = s^3 + 4s^2 + 2s + 5$
$b(s) = s^3 + 1$

These polynomials are represented by row vectors:

```
a = [1 4 2 5]

a =
     1     4     2     5

b = [1 0 0 1]

b =
     1     0     0     1
```

It is also possible to enter a polynomial by entering its roots. Say the polynomial u(s) has roots in $s = -3 \pm 2j$ en $s = -5$, in other words $u(s) = (s + 3 + 2j)(s + 3 - 2j)(s + 5)$. The polynomial u(s) can be found by:

```
r = [-3-2j, -3+2j, -5]

r =
  -3.0000 - 2.0000i  -3.0000 + 2.0000i  -5.0000

u = poly(r)

u =
     1    11    43    65
```

So the polynomial u(s) can be written as $u(s) = s^3 + 11s^2 + 43s + 65$.

Polynomial multiplication

The command `conv` calculates the product of two polynomials.

```
c = conv(a,b)

c =
     1     4     2     6     4     2     5
```

As expected v(s) is a polynomial of order six.

Polynomial division

For the inverse operation, the division of polynomials, the command `deconv` is used.

```
[q,r] = deconv(a,b)

q =
     1
r =
     0     4     2     4
```

The quotient q(s) and the remainder r(s) are determined in such a way that:

a(s) = q(s) b(s) + r(s)

When b(s) is a multiple of a(s), the remainder r(s) = 0.

Polynomial roots

The roots of a polynomial can be calculated using the command `roots`.

```
roots(u)

ans =
  -5.0000
  -3.0000 + 2.0000i
  -3.0000 - 2.0000i
```

Single point evaluation of a polynomial

polyval(p,x) calculates the value of the polynomial p in point x, or p(x). The power of Matlab lies in the fact that not only can x be chosen to be a scalar, but also a vector and even a matrix.

```
polyval(u,1)

ans =
   120

polyval([3 5 7],2)

ans =
    29

s = [1:10]

s =
     1     2     3     4     5     6     7     8     9    10

d = polyval([3 5 7],s)

d =
    15    29    49    75   107   145   189   239   295   357
```

The command `polyval` should not be confused with the command `polyvalm`. See the next example:

Let p(s) = s^2 + s + 1, so

```
p=[1 1 1]

p =
     1     1     1
```

and let matrix A be:

```
A = [2 3 ; 5 4]

A =
     2     3
     5     4
```

then `polyval(p,A)` = $\begin{pmatrix} p(2) & p(3) \\ p(5) & p(4) \end{pmatrix}$

```
polyval(p,A)

ans =
     7    13
    31    21
```

However, polyvalm(p,A) prodcues the more familiar result from linear algebra $A^2 + A + I$
(Be careful: I is the identity matrix, not the number 1)

```
polyvalm(p,A)

ans =
    22    21
    35    36
```

The derivative of a polynomial

A common polynomial operation is the determination of its derivative. Matlab uses the function `polyder` to calculate the derivative of a polynomial. This function can be used in different ways:

`q = polyder(p)`	returns the derivative of the polynomial p(s)
`q = polyder(a,b)`	returns the derivative of the product of the polynomials a(s) and b(s)
`[q,d] = polyder(a,b)`	returns the numerator q and denominator d of the derivative of the polynomial quotient b/a as a rational function: $\frac{q(s)}{d(s)} = \frac{d}{ds}\left(\frac{a(s)}{b(s)}\right)$

Partial fraction decomposition

Suppose the quotient of two polynomials a(s) and b(s) needs to be decomposed in partial fractions:

$$\frac{b(s)}{a(s)} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + \ldots\ldots + \frac{r_n}{s - p_n} + k(s)$$

We can use the Matlab `residue`:

```
[r,p,k] = residue(b,a)
```

The residue function works both ways, i.e. the following is valid Matlab command:

```
(b,a) = residue(r,p,k)
```

Suppose:

```
a = [1 8 23 26 10]

a =
     1     8    23    26    10
```

```
b = [1 -1]

b =
     1    -1

[r,p,k] = residue(b,a)

r =
  -0.2600 + 0.3200i
  -0.2600 - 0.3200i
   0.5200
  -0.4000
p =
  -3.0000 + 1.0000i
  -3.0000 - 1.0000i
  -1.0000
  -1.0000
k =
     []

[b,a] = residue(r,p,k)

b =
    0.0000    0.0000    1.0000   -1.0000
a =
    1.0000    8.0000   23.0000   26.0000   10.0000
```

2.6.2. Interpolation

We can use Matlab to interpolate between two (measuring) points. If vector x defines an independent variable and vector y contains the values between which we want to interpolate, then the Matlab command `p = polyfit(x,y,n)` generates the polynomial p(x) of the n^{th} order that interpolates between the points (x_i,y_i) using a least squares fit.

Other Matlab commands that are related to interpolation are:
`spline`, `interpft`, `interp1` and `interp2`.

Example:
Suppose we launched a rocket and we made a registration of how air density ρ [kg/m^3] varies with altitude h [km] in the atmosphere. Our registration contains 13 different points.

```
h = [7 10 15 21 27 34 39 43 47 51 55 59 61]

h =
  Columns 1 through 12
     7    10    15    21    27    34    39    43    47    51    55    59
  Column 13
    61

rho = [556 369 191 75 26.2 9.9 4.4 2.3 1.4 0.8 0.5 0.33 0.25]*10^-3

rho =
  Columns 1 through 7
    0.5560    0.3690    0.1910    0.0750    0.0262    0.0099    0.0044
  Columns 8 through 13
    0.0023    0.0014    0.0008    0.0005    0.0003    0.0003
```

We can fit a line through these points using a least square fit.

```
p = polyfit(h,rho,3)
```

```
p =
   -0.0131    1.7017  -70.8466  941.9188
```

2.7. Graphics

Some of the powerful aspects of Matlab are the very flexible approach to matrix manipulation in the Matlab workspace, the ability to work with infinity in the form of the Matlab constant `inf`, and as we will explore in this paragraph, the graphical possibilities. In our example in paragraph 2.6.2. it would have been interesting to display both the original measurement points and the calculated polynomial in one graph to show the errors of the fit. We will discuss some of the plotting capabilities of Matlab.

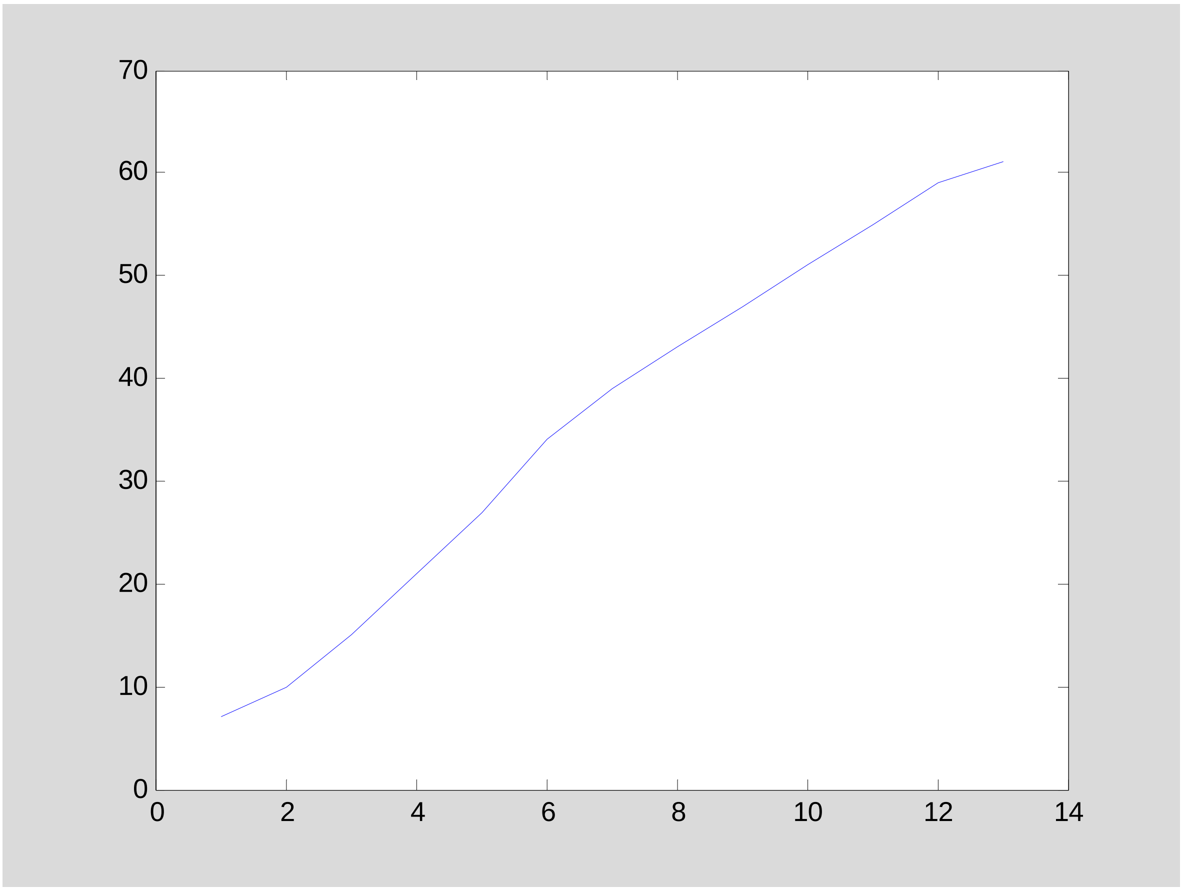
2.7.1. Two dimensional figures

2.7.1.1. Plotting basics

Suppose we want to present in a plot the altitude range we covered during our measurements. This is a quite straightforward affair:

```
h = [7 10 15 21 27 34 39 43 47 51 55 59 61];
rho = [556 369 191 75 26.2 9.9 4.4 2.3 1.4 0.8 0.5 0.33 0.25]*10^-3;

plot(h)
```



A number of things are evident from generating this plot. First of all there is no need to explicitly open a figure window. Matlab will open the window 'Figure No. 1' automatically. Furthermore the axes are scaled automatically and our points are connected with a line. The vertical axis shows the altitude h and the horizontal axis represents the corresponding element number of row vector h.

The plot is still a bit non-descript and probably someone with no knowledge of our experiment will not understand what it represents. First we want to indicate the actual measurement points separately. The `plot` command can be extended with a parameter that controls the line style, marker type and color. Say we want to show our measurement points using green circles. This how to accomplish that:

```
plot(h,'og')
```

The following table shows some of the possibilities.

Line style		Marker type		Color	
solid line	`-`	dot	`.`	yellow	`y`
dashed line	`--`	plus	`+`	magenta	`m`
dotted line	`:`	star	`*`	cyan	`c`
dash-dot line	`-.`	circle	`o`	red	`r`
		cross	`x`	green	`g`
				blue	`b`
				white	`w`
				black	`k`

Tabel 2.1: Matlab plot options.

Of course our graph needs a title and labels on the axes. We can also add a grid to the background:

```
title('figure 1: the altitudes where air density measurements were
taken')
xlabel('measurement point nr.')
ylabel('altitude h [km]')
grid
```

If we want to include even more text in the plot we can use the command `text`.

```
text(8,2,'measurement date: 060206')
```

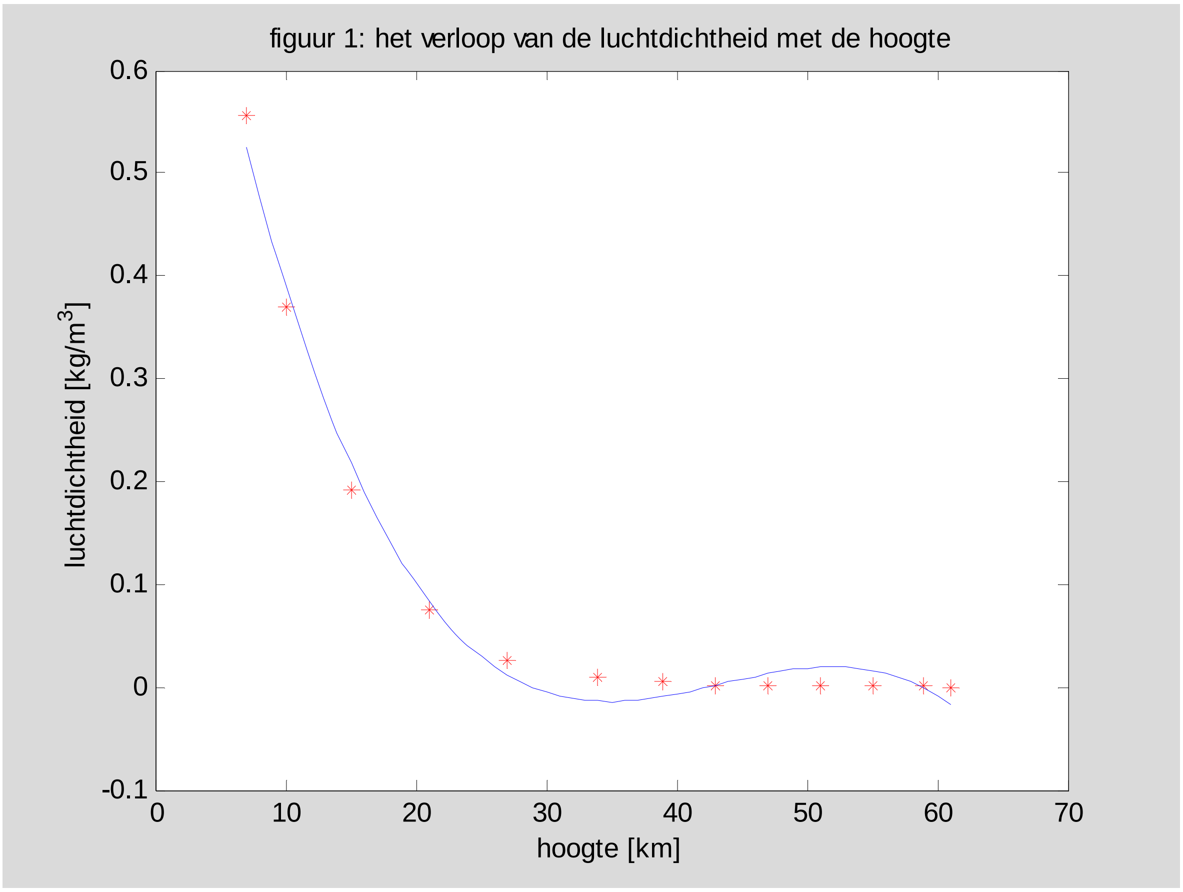
You can use the mouse to position the text at the right spot. Find out for yourself how you can also use `gtext` to do this. All you need to do to get the graph window in the foreground is type `figure(1)`.

Now we want to plot the how the air density varies with altitude:

```
plot(h,rho,'*r')
title('figure 1: air density variation with altitude')
xlabel('altitude [km]')
ylabel('air density [kg/m^3]')
```

Finally we want to include our estimated polynomial p from paragraph 2.6.2. into the same figure. In order to do so we need to somehow keep our current plot of the data points in our new plot. This can be accomplished using the command `hold on`. This command allows us to add a plot to a figure without erasing the previous graph.

```
p = polyfit(h,rho,3);
hold on
plot(min(h):max(h),polyval(p,min(h):max(h)))
```



2.7.1.2. Axis scaling

As we have seen before, Matlab will automatically scale the axes. However, sometimes the Matlab user might want to determine the scaling himself. The `axis` command has three different functions to accomplish this:

- `axis([xmin xmax ymin ymax])` defines extreme values for the x- and y-axis.
- `axis(axis)` 'freezes' the current scaling.
- `axis('auto')` restores automatic scaling.

`axis` is both a command and a variable. The command `v = axis` stores the current scaling of the axes into the variable v. In case one of the elements of the vector axis is infinity (∞), the corresponding scaling limit will once again be automatic. In other words: `axis([-inf inf -inf inf]) = axis('auto')`.

Other ways to use `axis` include:

`axis('ij')`	uses matrix coordinates, the vertical axis is pointed downwards
`axis('xy')`	uses Cartesian coordinates, the vertical axis is pointed upwards
`axis('square')`	assures both axes appear equal in length in the graph
`axis ('equal')`	uses the same scaling for both axes
`axis('off')`	switches off all axis lines, tick marks and labels
`axis('on')`	switches these options back on

To investigate details of the current figure you can use the `zoom` command. After entering the `zoom` command, clicking the left mouse button will enlarge the area of the figure under the mouse cursor. To zoom out, click the right mouse button. Entering `zoom off` will switch the zooming off.

2.7.1.3. Multiple figures

It is possible to present multiple graphs at once. Each plot can be presented in its own window. The command `figure(n)` can be used to open a new window or activate an existing window. When a plot command is given, Matlab will generate the plot in the window that is active at that moment. If there are no graphics windows yet, one will automatically be created. Each subsequent plot command will overwrite the previous plot, unless the command `hold on` has been entered.

Another possibility is to present multiple graphs within the same window, using the command `subplot`. Entering `subplot(m,n,p)` will subdivide the current figure window in m rows and n columns and will select the pth subplot for the next plot command.

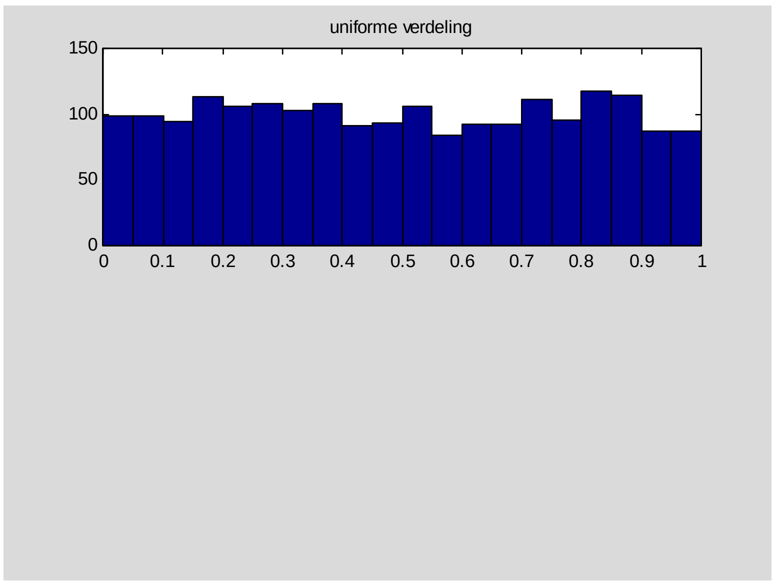
2.7.1.4. More plot commands

There are many more commands that enable flexible plotting. Some of the more useful commands follow below:

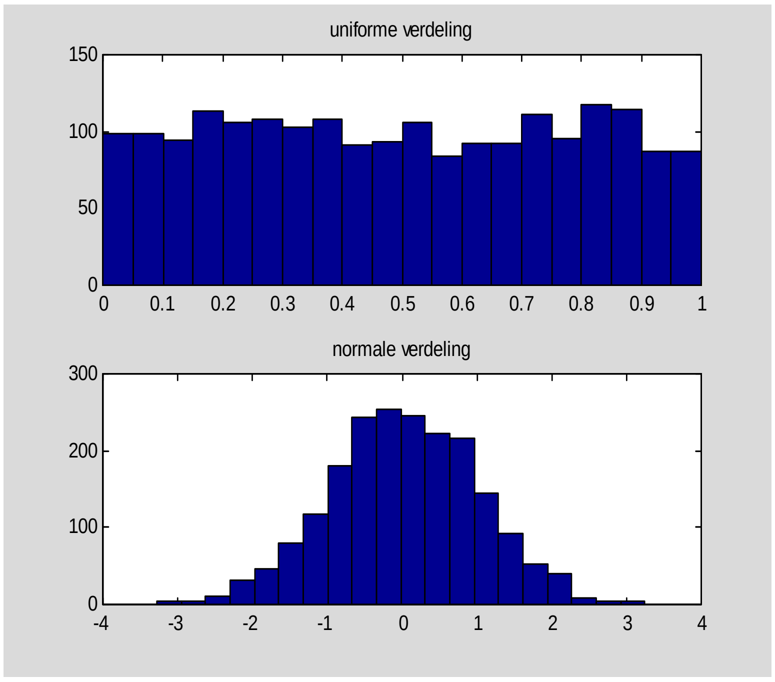
`semilogx`	plots the x-axis on a ^{10}log scale and the y-axis on linear scale. Parameters are he same as with the normal `plot` command.
`semilogy`	same, but with x- and y-axis reversed.
`loglog`	same, except both axes are now log scale.
`polar(theta,rho)`	generates a plot using polar coordinates, i.e. the angle theta (in radians) is plotted against the radius rho.
`bar,stairs,hist`	offer all kinds of bar graphs.

<u>Example</u>:

```
clear
close all
hold off
figure(1)
subplot(2,1,1)
n = 2000;
y1 = rand(n,1);
hist(y1,20)
title('uniform distribution')
```



```
figure(1)
subplot(212)
y2 = randn(n,1);
hist(y2,20)
title('normal distribution')
```



Try to find out the meaning of these histograms by varying the parameters in the previous example and observe the results.

2.7.2. Three dimensional figures

Matlab offers an array of possibilities of generating plots from more than two variables. However, 3D plots are a lot more complicated than 2D plots. Just changing the point of view on a 3D plot can generate a totally different image.

3D plots can be divided into three categories:

- surfaces generated by a function of two variables z = f(x,y)
- curves or surfaces defined by a parameter representation x = x(t), y = y(t), z = z(t)
- rotational bodies, generated by rotating a curve around a fixed axis

The next paragraphs will treat each of these categories separately and will also discuss the use of color in 3D plots.

2.7.2.1. Functions of two variables

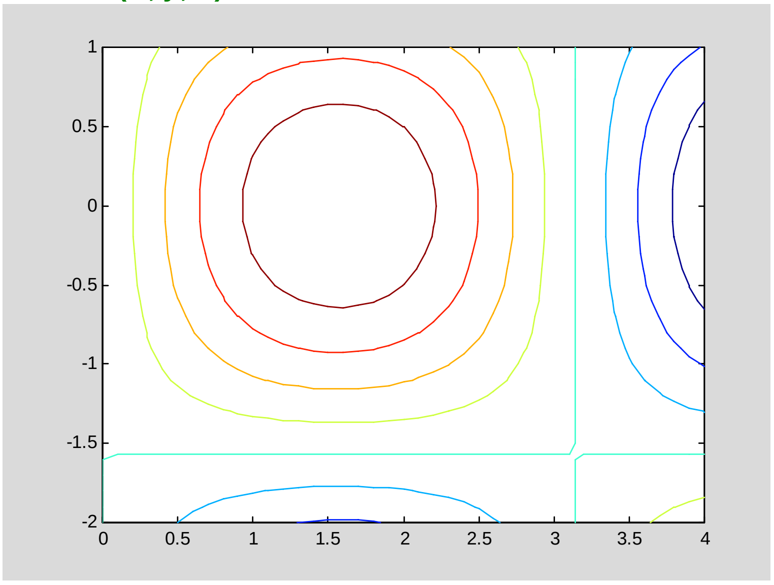
Suppose we want to plot the real function of two variables z = f(x,y). First order of business is defining the intervals of x and y over which we want to plot the function. These intervals will be stored in the variables x and y. Next we need to build to matrices X and Y, using the command `meshgrid`. The rows of matrix X containing the interval x, the columns of matrix Y contain the interval y. In other words, when two row vectors x and y define the x and y interval, the command `[X,Y] = meshgrid(x,y)`will generate a matrix X containing a number of rows equal to the length of y and a matrix Y with a number of columns equal to the length of x.

Now we can determine the values of the function z = sinx cosy with x ∈ [0,4] and y ∈ [-2,1]:

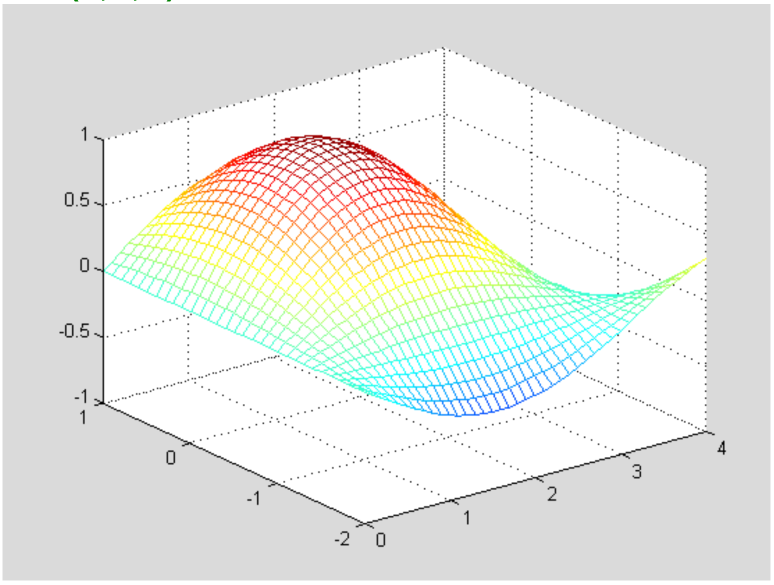
```
close all
x = 0:.1:4;
y = -2:.1:1;
[X,Y] = meshgrid(x,y);
Z = sin(X).*cos(Y);
```

Now that the domain and the function values have been determined, all we have to do is generate the 3D plot. To do so we have number of options. Pay attention to the differences in the following plot commands:

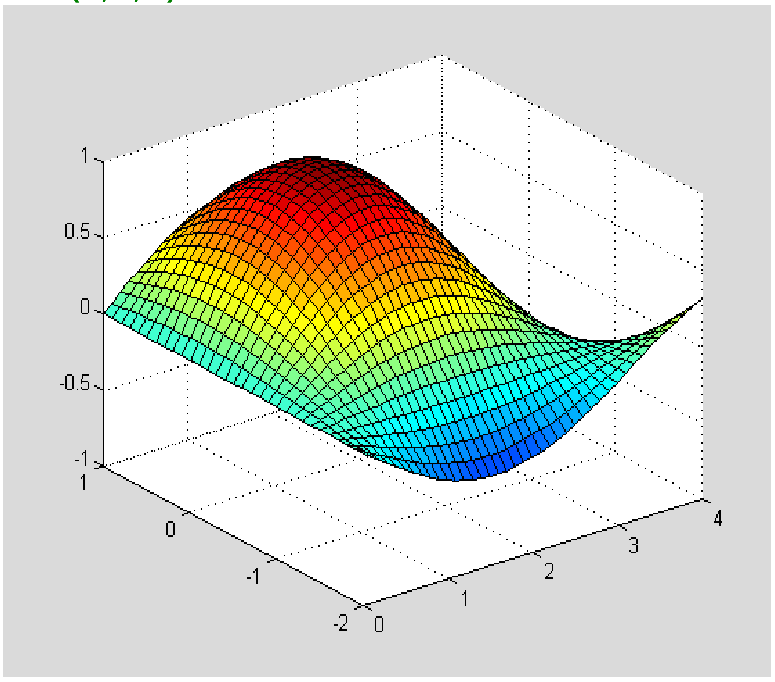
```
contour(x,y,Z)
```



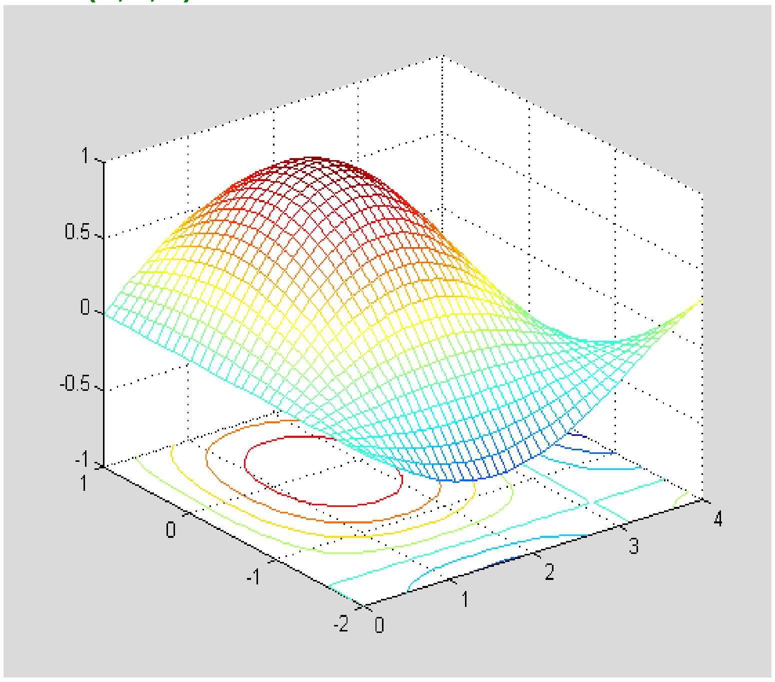
```
mesh(X,Y,Z)
```



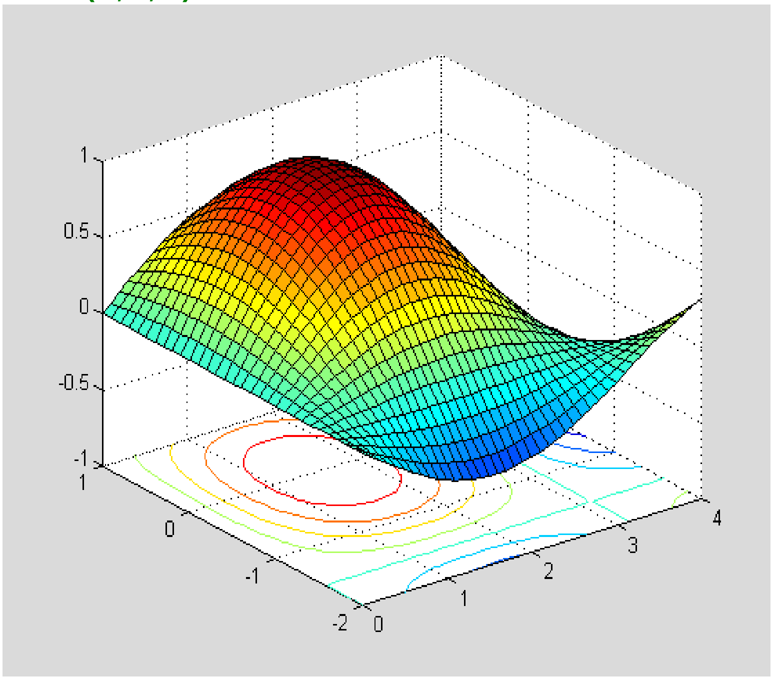
```
surf(X,Y,Z)
```



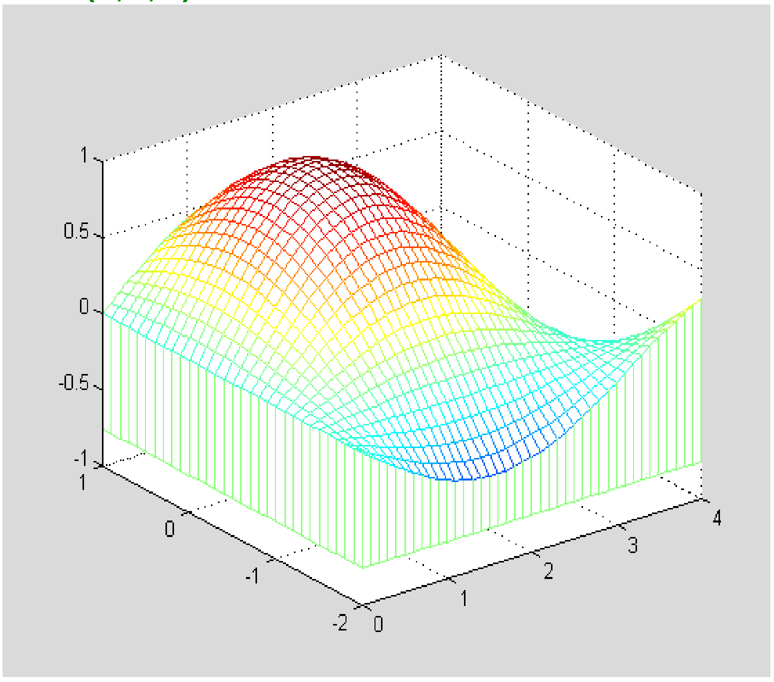
```
meshc(X,Y,Z)
```



```
surfc(X,Y,Z)
```



```
meshz(X,Y,Z)
```



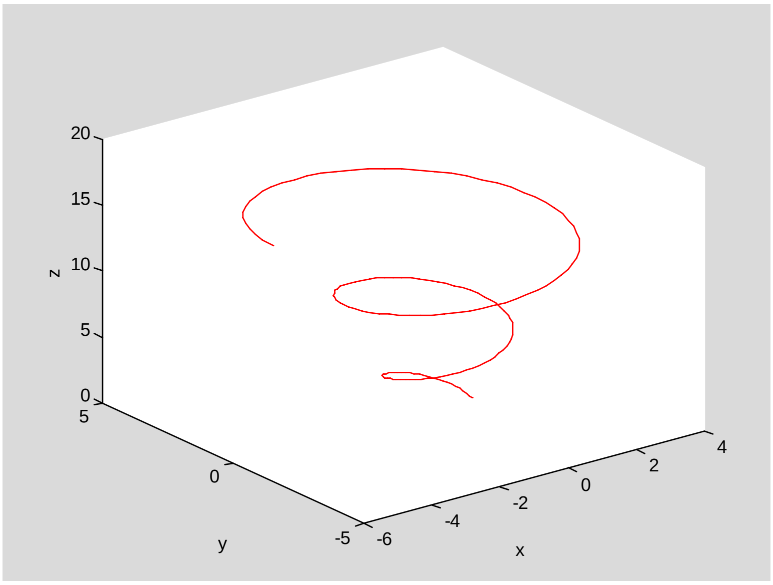
2.7.2.2. Parametric representation

A parametric representation is a simple display of curves and points in a three-dimensional space. Using the `plot3` command, it is easy to visualize parametric curves and surfaces. The syntax is the same as for the normal `plot` command, so it is easy to use different colors and line types.

<u>Example</u>: Generate a 3D visualization of the following parameter curve:

$x = r\cos(t)$
$y = r\sin(t)$
$z = t$ with $t \in [0 \quad 5\pi]$

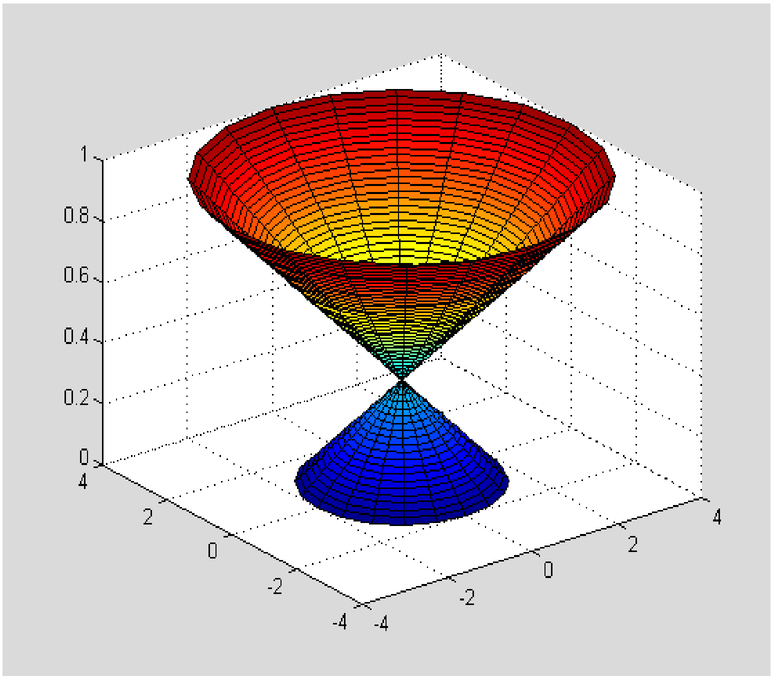
```
close all
t = 0:.1:5*pi;
r = exp(t/10);
x = r.*cos(t);
y = r.*sin(t);
z = t;
plot3(x,y,z,'-r');
xlabel('x');
ylabel('y');
zlabel('z');
```



2.7.2.3. Rotational bodies

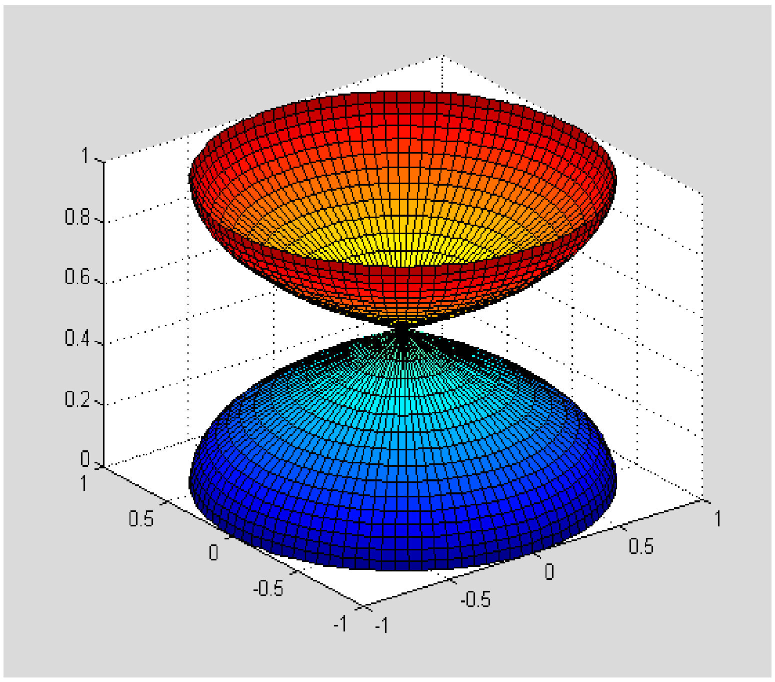
The command `cylinder(r,n)` produces a rotational body by rotating the curve of vector r around the x-axis. Matlab normally calculates 20 points on the circumference, but this number can be changed by specifying the parameter n.

```
cylinder(-2:.1:4);
```



`[X,Y,Z] = cylinder(r,n)` stores the points that define the rotational body in the matrices X, Y and Z.

```
[X,Y,Z] = cylinder(cos(0:.1:pi),100);
surf(X,Y,Z);
```



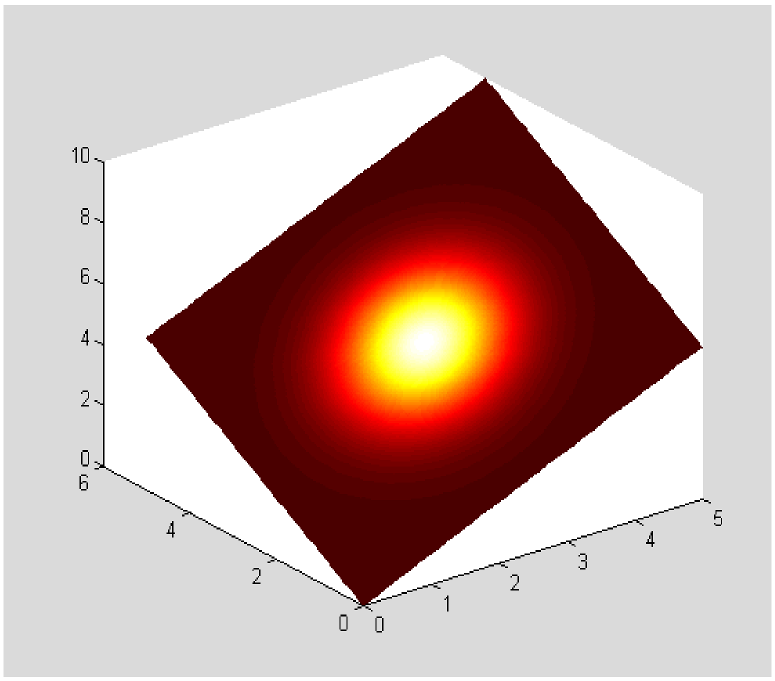
2.7.2.4. Using color

Instructions like `mesh` and the previously covered 3D plot commands allow the use of a fourth parameter matrix C. This extra parameter matrix makes it possible to select the colors to be used on the surface of the plot.

commands: `caxis, colormap, shading interp, shading flat, shading faceted, pcolor, image`

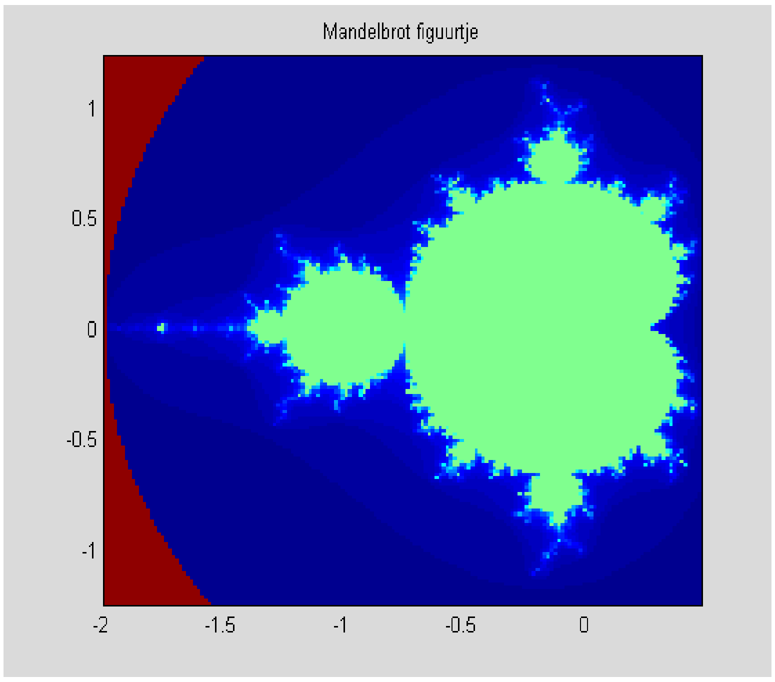
Example:

```
x = 0:.1:5;
[X,Y] = meshgrid(x);
Z = X+Y;
W = exp(-(X-2.5).^2-(Y-2.5).^2);
surf(X,Y,Z,W);
colormap(hot);
caxis([-.11 1]);
grid;
shading interp;
```



Example:

```
clear;
close all;
hold off;
[X,Y] = meshgrid(-2:.015:.5,-1.25:.015:1.25);
C = X + j*Y;
W = 100*ones(size(C));
Z = zeros(size(C));
for n = 1:50,
  Z = Z.*Z + C;
  h = find(abs(Z)<2);
  if ~isempty(h),
    W(h) = n*ones(size(h));
  else
    break
  end;
end;
clear C Z
pcolor(X,Y,W)
shading flat;
title('Mandelbrot set')
```


Mandelbrot figuurtje
1
0.5
0
-0.5
-1
-2
-1.5
-1
-0.5
0

2.8. Problems

1. Define a 3x3 matrix X and investigate the following operations on matrix X:

`X'`, `sum(X)`, `sum(X,1)`, `sum(X,2)`, `sum(X')`, `sum(X',1)`, `sum(X',2)`.

Make a comparison between the results of `sum(X,1)` and `sum(X',2)`. You may want to use the command `help sum`.

2. Define a vector A as A = [-1.383   3.945   5.636   -4.531]. Try the effect of changing numeric format by subsequently entering the following commands:

```
format short
format long
format bank
format short e
format long e
format compact
format loose
format +
```

Check the representation of vector A on the Matlab Command Window after each command. Conclusions?

3. When the vectors A and B are defined as A = [-2 -1 2 0] and B = [2 5 3 -1], give the values of the elements of vector C in the following statements:

```
C = A – B + 4
C = A./B
C = A.^B
C = 2^A + B
C = 2*B/3.*A
```

Check your answers using Matlab.

4. Generate a conversion table between the vertical velocity of an aircraft in ft/min and m/sec. Let the rate of climb (ROC) range from 0 to 3000 ft/min with steps of 100 ft/min. Store the table in a matrix of appropriate size and let the first column represent the ROC in ft/min and the second column the ROC in m/sec. One ft equals 0.3048 m.

5. Generate a conversion table between the elevator deflection angle in degrees and the elevator deflection angle in radians. Remember π radians equal 180°. Let the angle in degrees range from 0 to 360° with steps of one degree.

6. Check if $e^{a+bi} = e^{a}(\cos b + i \sin b)$ when a = 2 and $b = \frac{1}{3}\pi$ using one single Matlab statement.

7. Suppose the radius r ranges from 0 to 1 and the angle $\theta = \pi$*r. Make a polar plot of θ and r. Transform the complex numbers $re^{\theta i}$ in polar form to complex numbers a+bi in rectangular co-ordinate form. Make use of a predefined Matlab transformation. Make a plot of a+bi in a Cartesian co-ordinate system.

8. Investigate the function $f(x) = \dfrac{e^{x} - e^{-x}}{2}$ using Matlab

a. find the zeroes of the function f(x) using the command `fzero`
b. what is the value of the function in x = 3 ?
c. make a plot of the function f

9. Consider the next polynomial functions g_1 and g_2:

$$g1(x) = x^3 - 3x^2 - x - 3$$
$$g_2(x) = x^3 - 8x^2 + 20x - 16$$

a. find the zeros of both functions
b. calculate $g_1(5)$ and $g_2(12)$
c. find an expression for the product g_1*g_2
d. find the derivative of $g_1(x)$, $g_2(x)$ and $g_1(x)/g_2(x)$
e. plot $g_1(x)$ and $g_2(x)$ in one figure on the interval [–2 4]
f. what are the points of intersection of $g_1(x)$ and $g_2(x)$?
g. what is the value of $\dfrac{3g_1(x) - g_2(x)}{x-2}$ in x = {0, -3, 24, -1, -38, 3.2372*10^{23}, ∞, -∞, }

10. Make a three-dimensional plot of the function $f(x,y) = x^2 \sin y$ on an appropriate interval of x and y. Try different ways to visualise the surface.

11. Load the datafile landing.dat. This file contains different parameters that were registered during the final part of the landing of the Cessna Citation laboratory aircraft at Schiphol Airport. The parameters are arranged in columns:

column 01	pressure altitude	[ft]
column 03	indicated airspeed	[kts]
column 24	angle of attack	[degr]
column 26	load factor	[g m/sec^2]
column 31	theta	[degr]
column 43	relative time	[sec]

a. what is the size of the data file?
b. make a plot of the altitude versus time in the window "figure(1)"
c. make a plot of the indicated airspeed versus altitude in the window "figure(2)"
d. make a plot of theta versus time in the window "figure(3)"

There was some turbulence on final, as can for example be seen from the registration of the load factor.

e. make a plot of the loadfactor versus time in the window "figure(4)". Can you distinguish the moment the main gear of the aircraft touches the runway? And the moment the nose wheel touches the ground? You can also make use of the plot of theta.
f. calculate the mean load factor during the final approach before the aircraft touches the runway
g. find the maximum load factor and the minimum load factor when the aircraft is still airborne
h. What is the load factor when the aircraft touches the runway?

12. During a flight of the Cessna Citation II, the elevator angle required to maintain constant speed was measured at different speeds. The engine thrust was maintained constant throughout the experiment. The results of the measurements are listed below.

$\delta_e\,[\deg] =$ [-0.48; -0.88; -1.18; -1.68; -2.28; -0.08; 0.12; 0.33]

$V\,[m/\sec] =$[83.9; 78.7; 73.5; 68.8; 63.6; 91.0; 95.2; 100.0]

a. Plot δ_e against V, label the axes and give the plot the title "Citation elevator trim curve".
b. Fit a curve through the measurement points and indicate each measurement by a blue cross. Use a second order polynomial that fits the data in a least squares sense.
c. Somewhere in the plot add the text "measured at constant thrust".

3. PROBLEM-SOLVING USING MATLAB

3.1. INTRODUCTION

So far Matlab has been used in an interactive manner where every command is directly given in the Matlab command window with the keyboard. This method uses the computer as a sort of advanced pocket calculator and is fast and efficient. However when some commands have to be given repeatedly, while some parameters in the command must be changed simultaneously, the process becomes tedious.

As a solution Matlab offers the possibility to create so-called *m-files*. M-files contain subsequent Matlab commands that form a computer program. They are called m-files because they carry the extension `.m`. In these files all Matlab commands are available and a programming structure can be used. The Matlab programming structure is dealt with in section 3.2. In section 3.3 two types of m-files are discussed: the script file and the Matlab function. Finally, section 3.4. gives a methodology for problem solving with the help of Matlab.

3.2. MATLAB PROGRAMMING STRUCTURE

Matlab has a complete programming structure. A programming structure requires four fundamental operations, all available in Matlab:

- The order in commands is simply determined by the order the Matlab commands are given in the m-file or in the Matlab command window.
- The assignment statement is possible by using the " = " sign. This statement assigns a value to a variable when the variable was not previously defined and overwrites the value of a variable that had already been defined. When choosing variable names, one should be aware that Matlab differs between uppercase and lowercase variable names. It is common practise to use capitals for matrices.
- The choice in Matlab is possible with the classical operator:

```
if [condition1]
  instructionset1
elseif [condition2]
  instructionset2
else
  instructionset3
end
```

- Repeated commands are possible with the following operators:

```
for i = 1:n,                   while [condition]
  instructions                   instructions
end                            end
```

The command `if` can be used in any desirable form: as a single `if`, as an `if – else` command or as an `if – elseif – else` command, but is always followed by `end`.

The counter variable i, which is used in the for-loop can take on a more general form as mentioned above. The command `for i = A`, where A is a matrix, is possible. In this case the counter i gets the value of the subsequent numbers in the columns of matrix A.

Loops can be halted with the command `break`. In nested loops the `break` command only stops the inner loop.

When using loops, one must be aware that the execution of repeated commands is slow in Matlab. At this point Matlab differs from other programming languages. The application of loops must be avoided wherever possible. In some cases the command `find` offers a way out.

For example, compare the following lines where all elements of the matrices A and B which are greater than 1 are substituted with NaN.

```
tic
A = randn(500,800);
[m,n] = size(A);
for i = 1:m
  for j = 1:n
    if A(i,j)>1,
    A(i,j) = NaN;
    end
  end
end
toc

elapsed_time =
    2.9640

tic
B = randn(500,800);
i = find(B>1);
B(i) = NaN*ones(size(i));
toc

elapsed_time =
    0.2400
```

It can be concluded that in the second case, where no repeated statements have been applied, the substitution of the elements of the matrix is much faster than in the first case.

3.3. MATLAB M-FILES

3.3.1. Script files

A script file is a Matlab procedure that can be built with any text editor and contains a cascade of Matlab commands. Such m-files must be saved under a name with the extension .m. M-files can be run from the Matlab prompt in the command window and can be seen as Matlab commands made by yourself. This way a Matlab user can build his personal "toolbox" of m-files.

In Matlab a new m-file can be created, or an existing m-file can be changed from the File-menu. Select File, New, M-file from the Matlab command window to start a new m-file. The Matlab Editor/Debugger is automatically started and one can start typing command lines immediately. See to it that the new m-file is saved in a directory, which is contained in the Matlabpath, otherwise the m-file cannot be run from Matlab. Adding a directory to the Matlabpath is possible by using the command `addpath`, or by using the Pathbrowser available from the Set Path… option in the File menu.

```
help addpath

 ADDPATH Add directory to search path.
    ADDPATH DIRNAME  prepends the specified directory to the current
    matlabpath.  Surround the DIRNAME in quotes if the name contains a
    space.

    ADDPATH DIR1 DIR2 DIR3 ...  prepends all the specified directories to
    the path.

    ADDPATH ... -END   appends the specified directories.
    ADDPATH ... -BEGIN prepends the specified directories.
    Use the functional form of ADDPATH, such as
ADDPATH('dir1','dir2',...),
    when the directory specification is stored in a string.

    Examples
       addpath c:\matlab\work
       addpath 'Macintosh HD:Matlab:My Tools'
       addpath /home/user/matlab

    See also RMPATH.
```

Apart from setting the correct path to the script files in use, it is wise to start a main script file with a few commands to set the programming environment. The Matlab workspace must be cleared from all variables, the screen must be emptied, all figures must be closed, and the hold-option for plot commands should be turned to off. Therefore all main script files should start with at least the following commands:

```
clear;
clc;
close all;
hold off;
```

Of course, be careful with these commands in m-files that are called from within the main program.

To prevent that all calculated variables are presented on screen, close every command in the script file with the semicolon sign " `;` ".

3.3.2. Matlab functions

A function is created in Matlab as a script file starting with the following line:

`function [`*output variables*`]` = *Function Name* `(`*input variables*`)`

After this starting line defining the input- and output variables, a series of Matlab commands may follow. All temporary variables used within the function are *local* variables. After executing the function they are removed from the Matlab workspace. A variable A used within a function may be declared as a global variable by using the command `global` just before calling the function from an m-file. The variable A must also be declared as global within the function. To remove the variable from the workspace the command `clear global` A must be used.

In Matlab many different routines are pre-programmed. These routines are available from script files and m-functions. An example is the function `trace`, which is saved in the m-file trace.m in the directory C:\MATLAB\toolbox\matlab\matfun\. This function determines the sum of the elements on the main diagonal of a matrix.

```
edit trace.m
```

```
function t = trace(a)
%TRACE  Sum of diagonal elements.
%   TRACE(A) is the sum of the diagonal elements of A, which is
%   also the sum of the eigenvalues of A.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.4 $  $Date: 1997/11/21 23:38:57 $

t = sum(diag(a));
```

The second till the sixth line contain comments. In this case the core of the function consists of a single line. The comment lines immediately after the first line of a function are available in the on-line help of the function. For example the command `help trace` gives the following explanation of the function `trace`:

```
help trace

 TRACE  Sum of diagonal elements.
    TRACE(A) is the sum of the diagonal elements of A, which is
    also the sum of the eigenvalues of A.
```

3.4. PROBLEM SOLVING METHODOLOGY

Problem solving is a key part in engineering. This section aims to provide an engineering problem solving methodology, which is usable throughout many fields of science. The object is to learn to create m-files in an efficient and structured manner.

Five different steps in the process may be recognised:

1. Analyse the task
2. Define the problem clearly
3. Build the logic design of the program
4. Build the physical design
5. Test the (Matlab) code with all available data and check the results

Following this schedule it is possible to safe a lot of time in programming. The process of problem solving may be iterative, however a thorough analysis of the task and a clear problem definition will limit the number of required iterations considerably. Therefore it pays off to invest time in the first three steps of the process. In the following sections each of these five steps will be discussed in more detail.

3.4.1. Problem analysis

In the first phase of analysing the problem, one must give an answer to the question what problem must be solved. Very often a problem is stated vaguely and incomplete, or even contains contradictions. A poor defined problem is not a good starting point for creating a computer program. By systematically reviewing the problem, its clarity must be improved.

In this phase the theoretical background which is required to solve the problem must be studied in order to determine whether the desired solution of the problem can be obtained from the available data or that additional data/measurements are required.

The result of the first phase should be an accurately and clearly defined problem without any contradictions, or the conclusion that such a definition is impossible.

3.4.2. Problem definition

The second phase is closely related to the problem analysis. The goal of the problem definition is to make a description of the problem, which may serve as a basis for the next phase, the logic design of the program. The difference with the problem analysis phase is that during the analysis the problem is *reshaped* where necessary and in the problem definition phase the problem is made explicit and *described* in detail.

In this phase a description of the required input and the desired output information is very important. The available format of the input data and the desired format of the output data of the program must be considered and defined properly. Also it must be decided what plots will have to be presented by the program.

In this phase, it is good practise to write down a detailed description of the problem, the specifications of the input data and the desired results, which form the solution to the problem.

3.4.3. Logic design

In this phase a method is designed to create the desired output from the input data. This method is called an algorithm and is the basis for the program to be developed. It is essential to know that the algorithm is independent of the programming language and the computer system that will eventually be used.

The question of what the program must achieve is the central issue, i.e. what will be the architecture of the program. So in this phase the logistics of the program are constructed by making a structured diagram of all necessary calculations to obtain the desired output from the input. The emphasis lies on *structured*, because in order to keep the final program clear and efficient, it must have a well-developed structure.

3.4.4. Physical design

In this phase the question of how the program must achieve the solution to the problem is the central issue. The diagram containing the logic design of the program must be translated into Matlab commands. The logic design may be used as remarks in the m-file to emphasise the structure of the program as determined in the logical design of the program. By adding explanatory text preceded by the percent sign, an unfamiliar user of the program is able to understand the program more easily.

3.4.5. Testing and checking

Testing of the program is done in two steps. First the program (m-file) is run and Matlab may come up with syntax errors. Most of the time these errors can be solved easily. Matlab has a built-in debugger as an aid in correcting syntax errors. In this manual the Matlab debugger is not treated, because normally it is not needed for correcting the syntax errors.

After all syntax errors are corrected, Matlab is able to run the m-file(s). Errors in the logic or physical design may become exposed if the output of the program does not meet the expectations. Also, errors may occur because the input data is not in the specified form. Depending on the type of error, one may have to return to one or more of the preceding phases. As explained before, the process of creating a computer program that meets all requirements is often an iterative process.

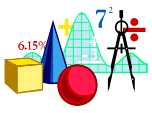
4. EXAMPLES OF SIMULATIONS IN MATLAB

In this chapter the knowledge obtained from the preceding chapters will be applied in practice. By making simulations of mechanical systems in Matlab one encounters different aspects of engineering practise. In this chapter the problem solving methodology from chapter 3 will be used when creating computer programs in Matlab. The computer programs will simulate different mechanical systems. To solve the equations of motion, some links with the courses Differential Equations or Numerical Analysis are inevitable.

By making simulations of systems one gets a more thorough investigation of the system compared to some theoretical lectures. The aim is to eventually simulate the aircraft response to control surface deflections in close relation to the course ae3-302 "Flight Dynamics I" and the flight test comprised in this practical.

While making the simulations try to notice the similarity in the equations of motion between different mechanical systems. In the first example the oscillations of a pendulum are considered. Although the motion of a pendulum does not seem to be associated with the motion of an aircraft, it will later become clear that the oscillations of the pendulum show a resemblance with some of the eigenmotions of the Cessna Citation aircraft, as demonstrated during the ae3-303P flight test experiment.

4.1 EXAMPLE 1: OSCILLATIONS OF A PENDULUM



In this example we consider the motion of a pendulum following different initial conditions. First the motion is considered to be without friction, but later friction is introduced. It is the first example where we use Matlab as an aid in simulating systems.

The objective is to study the motion of the pendulum by making a simulation of it in Matlab. In order to make the simulations, the equation of motion that describes the motion of the pendulum must be derived. The equation of motion will be implemented in Matlab and simulations of the pendulum can be made using different initial conditions and different values for the cord length, viscosity, gravitation constant and mass.

4.1.1 Derivation of the linearised equation of motion

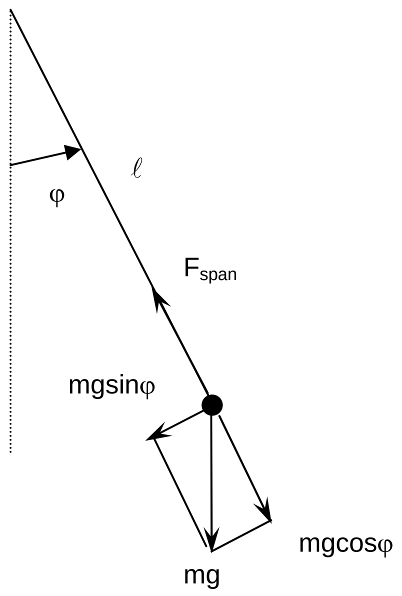


Figure 4.1: The forces acting on the pendulum.

4.1.1.1 Motion without friction

First we consider the equations of motion of the pendulum without friction. The accelerations in tangential, respectively radial direction are:

$$a_t = \ell\ddot{\varphi}$$
$$a_r = \frac{(\ell\dot{\varphi})^2}{\ell} = \ell\dot{\varphi}^2$$

So the equilibrium of forces in either direction gives:

tangential direction: $m\,\ell\ddot{\varphi} = -mg\sin\varphi$, or in rewritten form $\ddot{\varphi} = -\frac{g}{\ell}\sin\varphi$

radial direction: $m\ell\dot{\varphi}^2 = F_{span} - mg\cos\varphi$, or in rewritten form $\dot{\varphi}^2 = \frac{F_{span}}{m\ell} - \frac{g}{\ell}\cos\varphi$

From the first equation it can be seen that there is an obvious equilibrium position in $\varphi = 0$. The non-trivial solution of the equation gives the motion of the pendulum as long as the mass moves about the arc of a circle, i.e. as long as it has only one degree of freedom φ.

The second equation, the radial equation, can be used to determine the tension in the cord. From this equation the tensile force in the equilibrium situation can be found to be:

$$\frac{F_{span,0}}{m\ell} - \frac{g}{\ell}\cos\varphi_0 = 0 \quad\Rightarrow\quad F_{span,0} = mg$$

However, as long as we do not consider looping were the tension in the cord could become zero, the only equation of motion we need to describe the motion of the pendulum, is the tangential equation.

The equation of motion, a second order differential equation, could be solved numerically. However in this case we will only consider small deviations from the equilibrium position $\varphi_0 = 0$. In this case we can *linearise* the equations of motion around this equilibrium position. Notice that we will use exactly the same way of linearising the equations as is done in the course Flight Dynamics (ae3-302) where the full set of equations of motion of an aircraft is linearised. Using the same procedure we can state:

$$\varphi = \varphi_0 + d\varphi$$
$$\varphi_0 = 0$$
$$\dot{\varphi}_0 = 0$$

And from this it follows:

$$\sin\varphi = \sin(\varphi_0 + d\varphi) = \sin(\varphi_0)\cos(d\varphi) + \cos(\varphi_0)\sin(d\varphi) = 0\cdot\cos(d\varphi) + 1\cdot\sin(d\varphi) = \sin(d\varphi) = d\varphi$$
$$\cos\varphi = \cos(\varphi_0 + d\varphi) = \cos(\varphi_0)\cos(d\varphi) - \sin(\varphi_0)\sin(d\varphi) = 1\cdot\cos(d\varphi) + 0\cdot\sin(d\varphi) = \cos(d\varphi) = 1$$

Linearisation of the equations means that products of small quantities are neglected, so from the original equations we find the linearised equation of motion and the tensile force equation:

$$d\ddot{\varphi} = -\frac{g}{\ell}d\varphi$$
$$(d\dot{\varphi})^2 = \frac{F_{span,0} + dF_{span}}{m\ell} - \frac{g}{\ell} = \frac{dF_{span}}{ml}$$

For simplicity we rename dφ into φ and dF_{span} into F_{span}. One should realise that φ is now a small deviation from the equilibrium situation $\varphi = \varphi_0$ and F_{span} is a small deviation from the tensile force in the cord in the equilibrium situation $F_{span,0}$ = mg. Doing so, we arrive at the following linearised equations:

$$\ddot{\varphi} + \frac{g}{l}\varphi = 0$$
$$F_{span} = ml\dot{\varphi}^2$$

The solution of the second order differential equation $\ddot{\varphi} + \frac{g}{\ell}\varphi = 0$ has the general form of $\varphi = Ce^{\lambda t}$. Differentiating the solution with respect to time yields:

$$\dot{\varphi} = C\lambda e^{\lambda t}$$
$$\ddot{\varphi} = C\lambda^2 e^{\lambda t}$$

Substitution into the equation of motion gives:

$$C\lambda^2 e^{\lambda t} + \frac{g}{\ell}Ce^{\lambda t} = 0 \quad \Rightarrow \quad \lambda^2 = -\frac{g}{\ell} \quad < 0$$

$$\lambda = \pm\, j\sqrt{\frac{g}{\ell}} = \pm\, jp$$

The solution is now found to be:

$\varphi = C_1 e^{jpt} + C_2 e^{-jpt}$, where $p = \sqrt{\frac{g}{\ell}}$ and C_1 and C_2 are determined from the initial condition.

Suppose the initial condition at t = 0 is $\varphi = \frac{\pi}{6}$ and $\dot{\varphi} = 0$, then it follows:

$$C_1 + C_2 = \frac{\pi}{6}$$

And from differentiating the solution for φ with respect to time:

$\dot{\varphi} = jC_1 pe^{jpt} - jC_2 pe^{-jpt}$, the second equation in C_1 and C_2 is found:

$$C_1 - C_2 = 0$$

$$\varphi = \frac{\pi}{12}e^{jpt} + \frac{\pi}{12}e^{-jpt}$$

Remember that

$$e^{+jpt} = \cos pt + j\sin pt$$
$$e^{-jpt} = \cos pt - j\sin pt$$

Substitution yields the general solution for the deflection from the equilibrium situation:

$$\varphi = \frac{\pi}{6}\cos pt = \frac{\pi}{6}\cos\sqrt{\frac{g}{\ell}}\ t$$

The amplitude of the motion is $\pi/6$, the period is $P = \frac{2\pi}{\sqrt{\frac{g}{\ell}}}$ and the frequency is $f = \frac{\sqrt{\frac{g}{\ell}}}{2\pi}$.

4.1.1.2 Motion in a medium with viscosity

Now consider a pendulum placed in a medium with a certain viscosity, like air or water. The movement of the pendulum is now also subject to friction. Suppose the friction force is given by:

$$F_W = -\nu\ell\dot{\varphi}$$

After linearisation the equation of motion is:

$$\ddot{\varphi} = -\frac{g}{\ell}\varphi - \frac{\nu}{m}\dot{\varphi}$$

Again, the solution is of general form:

$$\varphi = Ce^{\lambda t}$$
$$\dot{\varphi} = C\lambda e^{\lambda t}$$
$$\ddot{\varphi} = C\lambda^2 e^{\lambda t}$$

Substitution yields:

$$\lambda^2 = -\frac{g}{\ell} - \frac{\nu}{m}\lambda \quad \Rightarrow \quad \lambda^2 + \frac{\nu}{m}\lambda + \frac{g}{\ell} = 0 \quad \Rightarrow \quad \lambda_{1,2} = \frac{-\frac{\nu}{m} \pm \sqrt{\frac{\nu^2}{m^2} - \frac{4g}{\ell}}}{2}$$

Since we found two solutions for λ the solution looks like:

$$\varphi = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}$$
$$\dot{\varphi} = C_1\lambda_1 e^{\lambda_1 t} + C_2\lambda_2 e^{\lambda_2 t}$$

$$t = 0,\ \varphi = \varphi(0) \quad \Rightarrow \quad C_1 + C_2 = \varphi(0)$$
$$t = 0,\ \dot{\varphi} = \dot{\varphi}(0) \quad \Rightarrow \quad \lambda_1 C_1 + \lambda_2 C_2 = \dot{\varphi}(0)$$

From these initial conditions the constants C_1 and C_2 can be determined:

$$C_1 = \varphi(0) - C_2$$
$$C_2 = \frac{\lambda_1\varphi(0) - \dot{\varphi}(0)}{\lambda_1 - \lambda_2}$$

4.1.2 Implementing the linearised equation of motion in Matlab

The solution of the equation of motion describing the damped motion of the pendulum is now determined analytically in time t and can be presented using Matlab. In order to implement the simulation of the pendulum in Matlab, we proceed according the problem solving methodology as presented in chapter 3. The goal is to create a script file, which simulates the pendulum under different conditions and presents the simulation results to the user. Below, all five steps of the problem solving process are discussed. Pass through these steps carefully because eventually it will safe you time.

4.1.2.1 Problem analysis

Most of the problem analysis has already been done in section 4.1.1 when deriving the linearised equation of motion. To complete the problem analysis, summarise the equations that you will need to implement in Matlab on a piece of paper. Before continuing with the problem definition, read the questions that will be asked about the simulation in section 4.1.3. In part f. of this section, for example, it is asked to derive an expression for the kinetic and potential energy of the pendulum. To make your m-file as complete as possible, derive the equations for the kinetic and potential energy now. Add the required equations to the summary of equations you will need to implement.

4.1.2.2 Problem definition

To define the problem clearly, make a list of required input data for the simulation. Think about parameters defining the geometry and mechanical properties of the pendulum and the initial conditions of the mass.

Subsequently, make a list of the desired output of the m-file. What variables do you want to present in figures and how do you want the figures to look like? Do you want to plot more than one variable in a figure? Refer to the questions that will be asked later on as much as possible.

Summarise the results of your investigation in a detailed description of the problem containing a specification of input data and desired results in the form of figures or otherwise.

4.1.2.3 Logic design

Here you will have to specify the algorithm you are going to use to obtain the desired output of the program from the input data. Write down in plain language <u>what</u> must be done in subsequent steps. In this phase the fundaments for the structure of your m-file are laid. Try to make your m-file as efficient as possible. This way you will not have to continuously change your m-file while answering the questions in section 4.1.3.

4.1.2.4 Physical design

The central issue in this phase is <u>how</u> the program must obtain the output from the input. Here the logic design is translated into subsequent Matlab commands. To emphasise the structure of your m-file add comments to groups of Matlab commands. The comments will tell an unfamiliar user of the m-file what is done in a certain part of the m-file. Add the description of the problem which is solved by the m-file at the top of the script file. Typing `help <m-file name>` at the command prompt will give this description. Use comments also to indicate the units of variables. The percent sign must precede comments.

4.1.2.5 Testing and checking

After saving the m-file under the name pendulum.m, the file can be tried from the command prompt by typing `pendulum`, provided that the Matlabpath is set correctly. Probably Matlab will come up with syntax errors. After correcting these errors, the m-file can be run. Now, one has to determine whether the simulation gives the desired output and whether the output is trustworthy. You can check this by trying different initial conditions and different mechanical properties of the pendulum, while bearing in mind the physical background of the pendulum.

Remember the design of a computer program is an iterative process. While answering the questions in the next section you might have to improve the m-file according to new insights in the problem. However by sticking to the five phases in creating a computer program as much as possible you can reduce the number of iterations and safe time.

4.1.3 Questions

Answer the following questions concerning the pendulum.

a. Consider the motion of a pendulum in a medium with a certain viscosity ν . Implement the linearised equation of motion in an m-file. Plot the angle φ in degrees against time t in seconds. This m-file should be able to handle different initial conditions $\varphi(0)$ and $\dot{\varphi}(0)$, but also different values of the cord length ℓ in [m], the mass m in [kg], de gravity constant g in [m/sec^2] and viscosity ν in [kg/sec]. For this first plot assume the following constants and initial conditions:

m = 1 kg	ℓ = 1 m	$\varphi(0) = \frac{\pi}{6}$
g = 10 m/sec^2	ν = 1 kg/sec	$\dot{\varphi}(0) = 0$

b. In the same figure make a plot of the angular velocity $\dot{\varphi}$ against time t assuming the same constants and initial conditions. Make sure you state the parameters and units used along the axes. Insert the text "phi" and "phidot" along the corresponding line in the plot.

c. Try the effect of changing initial conditions and the values of all constants on the amplitude, the period, the frequency and the damping of the movement. What parameters effect the amplitude, what parameters the period and what parameters the damping of the motion? Make a list of the dependencies.

d. By trial and error find the value of ν where the motion is no longer periodic, using your own m-file. Use the same values of constants and initial conditions as under a. Use the command `zoom` to zoom in on an existing figure to check the periodicity. After you found the value of ν investigate what constants and/or initial conditions influence the value of ν that you found.

e. By investigating the equation of motion, are you able to confirm this value of ν from a theoretical point of view? Give the exact expression for ν where the motion loses its periodic nature in terms of all relevant parameters.

f. Derive an expression for the kinetic and potential energy of the mass m as a function of time. Choose the equilibrium position of the mass as a reference for zero potential energy. Make a plot containing both kinetic and potential energy. In the same figure also plot the total energy of the system. Does the plot indicate that the system is conservative when $\nu = 1$? What if the viscosity $\nu = 0$? How do you explain this result? Do you think the initial deflection of 30° is too great for the linear simulation to be valid? Try smaller initial deflections.

The linear equation of motion is a second order differential equation. You know from theory that any second order differential equation can be split in a system of two first order differential equations. This can be accomplished by the substitution $\dot{\varphi} = \omega$, where omega can be physically interpreted as the angular velocity of the mass.

g. Perform the substitution and write the system of equations in a matrix notation.

The solution of these first order differential equations has the general form:

$$\begin{bmatrix} \varphi \\ \omega \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} e^{\lambda t}$$

h. Substitute this general solution in the system of first order equations describing the pendulum in a medium with viscosity ν and show this is an eigenvalue problem. Maintain the matrix notation throughout the derivation. What are the eigenvalues? What is the significance of the real and imaginary part of the eigenvalues for the motion of the pendulum? How does the nature of the motion (periodic or aperiodic, damped or not damped) depend on the value of the eigenvalues? Can the real part of the eigenvalues ever be positive? Why is instability impossible in this case?

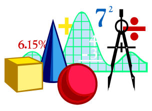
i. The eigenvalues can be written in the form $\lambda = \xi + i\eta$. What is the relation between the imaginary part η of the eigenvalues and the period P of the movement? Find an expression for the period in terms of the real and/or imaginary part of the eigenvalues. You may need the following relations:

$$e^{+jpt} = \cos pt + j\sin pt$$
$$e^{-jpt} = \cos pt - j\sin pt$$

j. To characterise the motion a number of quantities can be defined. For example the half time $T_{1/2}$ is defined as $\varphi(t + T_{1/2}) = \frac{1}{2}\varphi(t)$. $C_{1/2}$ is the number of periods during which the amplitude has diminished to half of the original value. The damping is defined from $\lambda = -\zeta\omega_0 \pm j\omega_0\sqrt{1-\zeta^2}$, ω_n is defined in $P = \frac{2\pi}{\omega_n} = \frac{2\pi}{\eta}$ and therefore $\omega_n = \omega_0\sqrt{1-\zeta^2}$, $\omega_0 = \sqrt{\xi^2 + \eta^2}$ and $\zeta = \frac{-\xi}{\sqrt{\xi^2 + \eta^2}}$

Calculate ζ when the same constants and initial conditions as under part a. are chosen. What is the value of the damping when the viscosity has the value as found in part e. and f. and the rest of the parameters are unchanged?

4.2 EXAMPLE 2: A SYSTEM OF FIRST ORDER LINEAR DIFFERENTIAL EQUATIONS



In the next example a system of homogeneous, linear differential equations with constant coefficients will be considered. Such a system has the following form:

$$\dot{\mathbf{x}} = \mathbf{A}\ \mathbf{x}$$

In which the matrix **A** is a constant nxn matrix. In the course Flight Dynamics (ae3-302) it will be shown that the linearised equations of motion of an aircraft have a similar representation. Therefore it is interesting to derive the general solution of an example of such a system. Matlab is a useful tool to accomplish this goal.

Suppose we would like to derive the general solution of the following first order system:

$$\dot{\mathbf{x}} = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix} \mathbf{x}$$

In the course "Differential Equations" it has been shown that the general solution is found in the form $\mathbf{x} = \mathbf{v}e^{\lambda t}$, in which λ is a scalar and **v** is a 2x1 column vector.

a. Substitute this general solution in the original set of equations and show that we are dealing with an eigenvalue problem.

In other words: we are looking for the eigenvalues λ and the eigenvectors **v** of matrix **A**. Matlab offers the possibility to find both with the command `[V,D] = eig(A)`. The columns of the matrix **V** contain the eigenvectors of **A** and the diagonal matrix **D** contains the eigenvalues of **A**, such that **AV** = **VD**.

b. Find the eigenvalues and corresponding eigenvectors using the above-mentioned Matlab command.

Notice that Matlab determines the eigenvectors such that the norm of the eigenvectors equals 1.

c. Check this by calculating the norm of the columns of **V**.

The solutions of the set of equations are determined by the fundamental set:

$$\mathbf{x_1}(t) = \mathbf{v_1}\ e^{\lambda_1 t} \text{ and } \mathbf{x_2}(t) = \mathbf{v_2}\ e^{\lambda_2 t}$$

The general solution is a linear combination of both solutions:

$$\mathbf{x}(t) = c_1\mathbf{x_1}(t) + c_2\mathbf{x_2}(t)$$

Because the vector **x** is a two dimensional vector, this vector can be depicted in a flat surface at any point in time t. The general solution of the system of differential equations can be presented as a parametric figure in t. For each value of c_1 and c_2 a trajectory can be determined.

d. Make a Matlab m-file that plots these trajectories. For the values of the constants c_1 and c_2 choose the interval –2:1:2. Label the axes. In this case you will have to make use of a for-loop and the command `hold on`.

The following figure shows the result of a Matlab program that plots these trajectories:

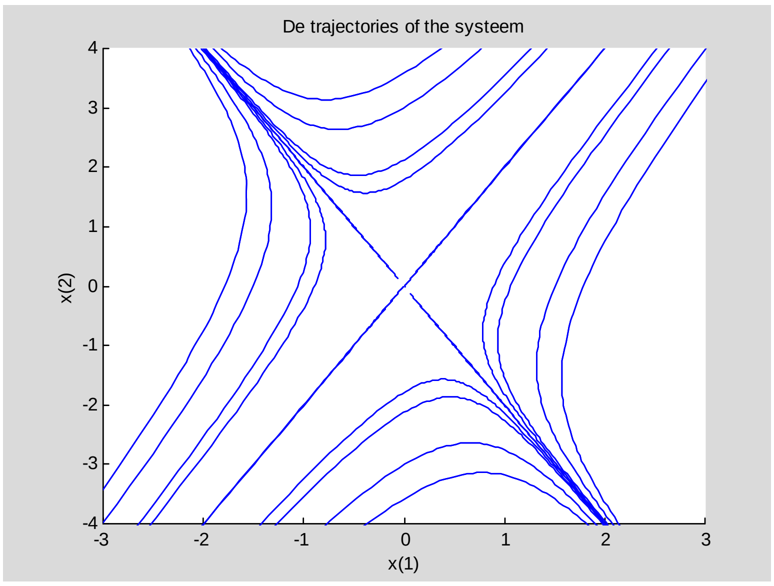


Figure 4.2: The trajectories of the system of two first order linear differential equations.

It is possible to assign a direction to the trajectories.

e. Determine the direction of the above trajectories yourself by finding the dominant part of the solution when $t \to \infty$. Why is the origin a so-called saddle point?

We can try to plot the trajectories in a three dimensional figure. In such a plot the course of time is made explicit, and so the direction of the trajectories will become clear.

f. Determine a set of parametric curves representing the trajectories. The parametric curves can be found by using the command plot3, see also section 2.7.2.2. of the Matlab introduction. Time is plotted along the vertical axis. For the values of the constants c_1 and c_2 choose the interval –2e-12:0.5e-12:2e-12 and –10:2:10 respectively.

The following figure shows the result of a Matlab program that plots these parametric curves:

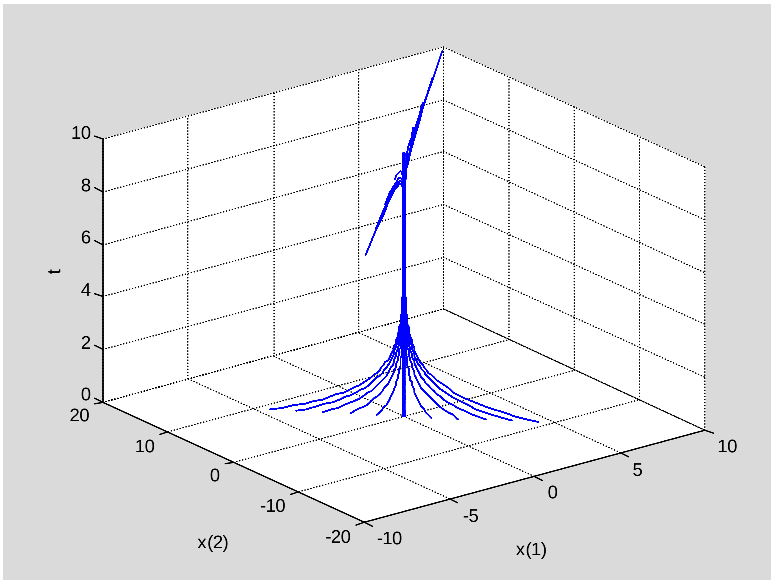


Figure 4.3: parametric curves representing the system of two first order linear differential equations

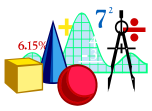
g. Answer the questions a. till f. in case the matrices $\mathbf{A}_{1\ldots5}$ are defined as:

$$\mathbf{A}_1 = \begin{pmatrix} -1 & -1 \\ 4 & -1 \end{pmatrix},\ \mathbf{A}_2 = \begin{pmatrix} 1 & 1 \\ -4 & -1 \end{pmatrix},\ \mathbf{A}_3 = \begin{pmatrix} 1 & 1 \\ -4 & 1 \end{pmatrix},\ \mathbf{A}_4 = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix},\ \mathbf{A}_5 = \begin{pmatrix} 8 & 1 \\ -8 & 2 \end{pmatrix}$$

Tip: Make your m-file such that it can handle any 2x2 matrix A.

h. Compare the eigenvalues of $\mathbf{A}_{1\ldots5}$ with the plots of the trajectories generated by your m-file. What conclusions can you draw from this exercise?

4.3 Example 3: The motion of a pointmass in a force field



This example treats the motion of particle under the influence of a two-dimensional force field. The force on the particle is a function of its position in the force field only. What fysically causes this force is not relevant to the problem.

Suppose the field force is given by:

$$\begin{aligned} F_x &= -cx - \sqrt{3}cy \\ F_y &= -\sqrt{3}cx - 3cy \end{aligned}$$

a. Determine, using Matlab where possible, what this force field looks like. In which points is the field force zero? What is the direction of the field? Plot a three-dimensional picture of the magnitude of the force as a function of the position in the force field.

<u>Hint</u>: use the functions `meshgrid`, `quiver` and `contour` to visualize the direction and magnitude of the force field.

Using Newton's second law, the equations of motion can be written as follows:

$$\ddot{x} = -\frac{c}{m}x - \frac{\sqrt{3}c}{m}y$$
$$\ddot{y} = -\frac{\sqrt{3}c}{m}x - \frac{3c}{m}y$$

These equations of motion are readily represented by a matrix notation:

$\ddot{\mathbf{x}} - \mathbf{A}\,\mathbf{x} = \mathbf{0}$, where matrix $\mathbf{A} = \frac{c}{m}\begin{pmatrix} -1 & -\sqrt{3} \\ -\sqrt{3} & -3 \end{pmatrix}$ and vector $\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$.

The solution has the form $\mathbf{x} = \mathbf{v}e^{\lambda t}$, since the equations of motion form a system of second-order homogeneous lineair differential equations with constant coefficients. (See W.E. Boyce and R.C. DiPrima, *Elementary differential equations and boundary value problems*). Substitution of this solution into the system of equations yields:

$$\lambda^2\mathbf{v}e^{\lambda t} - \mathbf{A}\,\mathbf{v}e^{\lambda t} = \mathbf{0}$$

$$\lambda^2\,\mathbf{v} - \mathbf{A}\,\mathbf{v} = \mathbf{0}$$

$$\left(\mathbf{A} - \lambda^2\mathbf{I}\right)\mathbf{v} = \mathbf{0}$$

This is an eigenvalue problem. To rule out the trivial solution the following determinant has to be assumed zero:

$$\mathbf{v} \neq \mathbf{0} \quad \Rightarrow \quad \left|\mathbf{A} - \lambda^2\mathbf{I}\right| = 0$$

b. Perform this calculation by hand and determine the eigenvalues and eigenvectors for a mass m = 1 kg and a fieldconstant c = 100 N/m.

Use can be made of the Matlab function eig to solve the eigenvalue problem.

```
help eig

 EIG    Eigenvalues and eigenvectors.
    E = EIG(X) is a vector containing the eigenvalues of a square
    matrix X.

    [V,D] = EIG(X) produces a diagonal matrix D of eigenvalues and a
    full matrix V whose columns are the corresponding eigenvectors so
    that X*V = V*D.

    [V,D] = EIG(X,'nobalance') performs the computation with balancing
    disabled, which sometimes gives more accurate results for certain
    problems with unusual scaling.

    E = EIG(A,B) is a vector containing the generalized eigenvalues
    of square matrices A and B.

    [V,D] = EIG(A,B) produces a diagonal matrix D of generalized
    eigenvalues and a full matrix V whose columns are the
    corresponding eigenvectors so that A*V = B*V*D.

    See also CONDEIG, EIGS.

 Overloaded methods
    help sym/eig.m
    help lti/eig.m
```

c. Calculate the eigenvalues using Matlab and verify the results against your own caulculations.

d. Matlab determines eigenvectors in such a way that the norm of the eigenvectors equals one. Verify this by taking the norm of the columns of matrix **V**.

Because this problem has four eigenvalues, the generic solution depends on four constants c_1, c_2, c_3 en c_4.

$\mathbf{x} = c_1\mathbf{v_1}e^{\lambda_1 t} + c_2\mathbf{v_1}e^{\lambda_2 t} + c_3\mathbf{v_2}e^{\lambda_3 t} + c_4\mathbf{v_2}e^{\lambda_4 t}$, keeping in mind that there are two identical eigenvalues ($\lambda_1 = \lambda_2 = 0$), yields:

$$\begin{pmatrix} x \\ y \end{pmatrix} = c_1\begin{pmatrix} -0.8660 \\ 0.5000 \end{pmatrix} + c_2\begin{pmatrix} -0.8660 \\ 0.5000 \end{pmatrix} t + c_3\begin{pmatrix} -0.5000 \\ -0.8660 \end{pmatrix} e^{2\sqrt{\frac{c}{m}}i\cdot t} + c_4\begin{pmatrix} -0.5000 \\ -0.8660 \end{pmatrix} e^{-2\sqrt{\frac{c}{m}}i\cdot t}$$

The exponents can be written as a complex sum of sines and cosines:

$$x = -0.8660c_1 - 0.8660c_2 t - 0.5000c_3\left(\cos 2\sqrt{\tfrac{c}{m}}t + i\sin 2\sqrt{\tfrac{c}{m}}t\right) - 0.5000c_4\left(\cos 2\sqrt{\tfrac{c}{m}}t - i\sin 2\sqrt{\tfrac{c}{m}}t\right)$$

$$y = +0.5000c_1 + 0.5000c_2 t - 0.8660c_3\left(\cos 2\sqrt{\tfrac{c}{m}}t + i\sin 2\sqrt{\tfrac{c}{m}}t\right) - 0.8660c_4\left(\cos 2\sqrt{\tfrac{c}{m}}t - i\sin 2\sqrt{\tfrac{c}{m}}t\right)$$

Rearranging terms:

$$x = -0.8660\,c_1 - 0.8660\,c_2 t - 0.5000(c_3 + c_4)\cos 2\sqrt{\tfrac{c}{m}}t - 0.5000(c_3 - c_4)i\sin 2\sqrt{\tfrac{c}{m}}t$$

$$y = +0.5000\,c_1 + 0.5000\,c_2 t - 0.8660(c_3 + c_4)\cos 2\sqrt{\tfrac{c}{m}}t - 0.8660(c_3 - c_4)i\sin 2\sqrt{\tfrac{c}{m}}t$$

Renaming constants $c_3^* = c_3 + c_4$ and $c_4^* = (c_3 - c_4)i$ results in:

$$x = -0.8660\,c_1 - 0.8660\,c_2 t - 0.5000 c_3^* \cos 2\sqrt{\tfrac{c}{m}}t - 0.5000 c_4^* \sin 2\sqrt{\tfrac{c}{m}}t$$
$$y = +0.5000\,c_1 + 0.5000\,c_2 t - 0.8660 c_3^* \cos 2\sqrt{\tfrac{c}{m}}t - 0.8660 c_4^* \sin 2\sqrt{\tfrac{c}{m}}t$$

The constants c_1, c_2, c_3^* and c_4^* can be determined from the intial values of the problem. In order to do this, an expression for the velocity components $\dot{x}$ and $\dot{y}$ has to be determined, which can be achieved by differentiating the vector equation.

$$\dot{x} = -0.8660\,c_2 - 0.5000 \cdot 2\sqrt{\tfrac{c}{m}}\,c_3^* \sin 2\sqrt{\tfrac{c}{m}}t - 0.5000 \cdot 2\sqrt{\tfrac{c}{m}}\,c_4^* \cos 2\sqrt{\tfrac{c}{m}}t$$
$$\dot{y} = +0.5000\,c_2 - 0.8660 \cdot 2\sqrt{\tfrac{c}{m}}\,c_3^* \sin 2\sqrt{\tfrac{c}{m}}t - 0.8660 \cdot 2\sqrt{\tfrac{c}{m}}\,c_4^* \cos 2\sqrt{\tfrac{c}{m}}t$$

At time t = 0 the velocity of the particle is given by:

$$\dot{x}_0 = -0.8660\,c_2 - 0.5000 \cdot 2\sqrt{\tfrac{c}{m}}\,c_4^*$$
$$\dot{y}_0 = +0.5000\,c_2 - 0.8660 \cdot 2\sqrt{\tfrac{c}{m}}\,c_4^*$$

Furthermore, the position of the particle at time t = 0 is:

$$x_0 = -0.8660\,c_1 - 0.5000 c_3^*$$
$$y_0 = +0.5000\,c_1 - 0.8660 c_3^*$$

Apparently the initial condition of the particle is fully determined by its position and its velocity. Once the initial condition is known, any further movement of the particle can be determined as well. It can therefore by concluded that the position (2 coordinates) and velocity (2 components) of the particle at any given time, fully determine any further movement of the particle. This leads us to the introduction of the notion of *state*. In this example, the *state* of the particle is determined by four parameters. Although it is possible to choose different parameters, you will always need four to completely describe the state.

e. How many parameters would be needed to describe the state of a particle that moves in a three-dimensional force field?

The initial condition can be described by a vector $\mathbf{y_0}$:

$\mathbf{y_0} = [x_0 \quad y_0 \quad \dot{x}_0 \quad \dot{y}_0]^T$ where

$$x_0 = -0.8660\,c_1 - 0.5000 c_3^*$$
$$y_0 = +0.5000\,c_1 - 0.8660 c_3^*$$
$$\dot{x}_0 = -0.8660\,c_2 - 0.5000 \cdot 2\sqrt{\tfrac{c}{m}}\,c_4^*$$
$$\dot{y}_0 = +0.5000\,c_2 - 0.8660 \cdot 2\sqrt{\tfrac{c}{m}}\,c_4^*$$

This can be written using matrix notation:

$$\begin{bmatrix} x_0 \\ y_0 \\ \dot{x}_0 \\ \dot{y}_0 \end{bmatrix} = \begin{bmatrix} -0.8660 & 0 & -0.5000 & 0 \\ +0.5000 & 0 & -0.8660 & 0 \\ 0 & -0.8660 & 0 & -0.5000 \cdot 2\sqrt{\frac{c}{m}} \\ 0 & +0.5000 & 0 & -0.8660 \cdot 2\sqrt{\frac{c}{m}} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3^* \\ c_4^* \end{bmatrix}$$

f. Write an m-file to determine the constants c_1, c_2, c_3^* en c_4^* for a given initial condition. Write the m-file in such a way that the values of the constants can be determined for *any* initial condition.

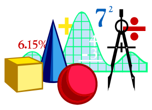
g. Include in your m-file code to calculate the trajectory of the particle, depending on the initial condition. Make a plot of the trajectory.

h. Is it possible to choose the initial conditions in such a way that the particle will continue to move indefinitely, without going to infinity?

i. The construction of this simulation had been very cumbersome. Would it help to use the Laplace transform? Compare this example to the example of the pendulum. What are the similarities in the equations of motion?

This problem can also be solved by rewriting the second-order differential equations as first-order differential equations. This results in a set of 4 first-order differential equations which are a lot easier to solve using Matlab. Note that each state of the particle corresponds to only one first-order equation. This makes it a lot easier to incorporate the initial conditions. It will be shown later that it is also possible to use Simulink to solve this problem.

4.4 EXAMPLE 4: MASS-SPRING SYSTEM



In the following example a mass-spring system is considered. The spring constant is c and the mass is m. The input to the spring is a time dependent force Fsinωt. At t = 0 the deflection x and the velocity of the mass are zero. There is no gravity in this example.

a. Find the equations of motion of the mass-spring system.

As is shown during the course "Differential equations" (wi225LR) the differential equation can be solved by applying the Laplace transform.

b. Derive the Laplace transformed equation of motion. Take the initial conditions into account.

The Laplace transformed deflection X(s) can be programmed in Matlab, making use of polynomials. Make use of the substitution $s^2 = u$. The polynomials n1 en n2 in the denominator can be presented in the variable u.

For now, suppose $\omega = 4\pi$ rad/sec, F = 1 N, c = 10 N/m en m = 0.1 kg.

c. Create an m-file where you find an expression for the numerator and the denominator of the Laplace transformed deflection X(s) making use of polynomials. Use the Matlab command `conv`.

To find the deflection x(t) as a function of time, X(s) must be inverse Laplace transformed. To find the inverse transform X(s) must be decomposed into partial fractions.

d. Execute this in your m-file. Use the Matlab command `residue`.

The inverse Laplace transform of X(s) can be found by looking it up in a table. In this way the deflection x of the mass can be found as a function of time.

e. Complete the m-file with the expression you found for x(t). Make sure you determine x(t) for all possible values of ω, F, c en m by defining these quantities at the beginning of the m-file. The initial conditions however remain unchanged during this exercise.

With help of Matlab this function x(t) can be presented in a plot.

f. Make a plot of the deflection x as a function of time. Consider the first 10 seconds of the motion.

g. By trial and error try to find the value of ω where the deflection of the mass is maximum. Use your own m-file. What happens when ω is a little smaller or greater than the value you found? Check the response over a greater time interval than 10 seconds, if necessary.

h. Now consider the mass-spring system <u>without</u> exciting force F. Rewrite the equation of motion (a second order differential equation) into a system of two first order differential equations by substituting $\dot{x} = v$ and determining $\dot{v}$. Bring the set of two first order differential equations in the form $\dot{u} = Au$, where A is a 2x2 matrix with constant coefficients.

i. Apply the Laplace transform to this set of equations and show this leads to an eigenvalue problem. Solve the eigenvalue problem. What are the eigenvalues of the system?

j. Determine the period of the free response of the mass using the eigenvalues. Compare the result with the value you found for ω under g. Can you explain the result from a physical point of view?

4.5 EXAMPLE 5: AIRCRAFT LINEAR EQUATIONS OF MOTION

The linear equations of motion derived in the course Flight Dynamics are based on a number of assumptions:

- the aircraft is represented as a single rigid body
- the mass of the aircraft does not change

The equations of motion are linearised, allowing only small deviations around an equilibrium state of the aircraft. The symmetric equations of motion as derived in "Flight Dynamics" are:

$$\begin{bmatrix} C_{X_u}-2\mu_c D_c & C_{X_\alpha} & C_{Z_0} & C_{X_q} \\ C_{Z_u} & C_{Z_\alpha}+\left(C_{Z_{\dot\alpha}}-2\mu_c\right)D_c & -C_{X_0} & C_{Z_q}+2\mu_c \\ 0 & 0 & -D_c & 1 \\ C_{m_u} & C_{m_\alpha}+C_{m_{\dot\alpha}}D_c & 0 & C_{m_q}-2\mu_c K_{yy} D_c \end{bmatrix} \cdot \begin{bmatrix} \hat{u} \\ \alpha \\ \theta \\ \dfrac{q\bar{c}}{V} \end{bmatrix} = \begin{bmatrix} -C_{X_\delta} \\ -C_{Z_\delta} \\ 0 \\ -C_{m_\delta} \end{bmatrix} \cdot \delta_e$$

In this equation *D*$_c$ is a differential operator, $D_c = \frac{\bar{c}}{V}\frac{d}{dt}$ and $\hat{u}$ is the dimensionless speed, $\hat{u} = \frac{u}{V}$ with *V* the true airspeed of the aircraft of the stationary flight condition. The equation can be rewritten using the speed deviation u instead of the dimensionless speed $\hat{u}$ and the normal angular pitch rate q , instead of the dimensionless version $\frac{q\bar{c}}{V}$. Implementing this substitution in the above equation yields the following equation:

$$C_1\dot{\bar{x}} + C_2\bar{x} + C_3\bar{u} = \bar{0}$$

Here $\bar{u}$ is the input vector and $\bar{x}$ is the state vector;

$$\bar{u} = [\delta_e]$$
$$\bar{x} = [u \quad \alpha \quad \theta \quad q]^T$$

Derive the matrices *C*$_1$, *C*$_2$ and *C*$_3$ yourself. Rewrite the equation very carefully, because any mistake, however small, will lead to faulty results in the simulations.

The form of the equation we are aiming at is the *state space representation* of the system:

$$\dot{\bar{x}} = A\bar{x} + B\bar{u}$$
$$\bar{y} = C\bar{x} + D\bar{u}$$

With $\bar{x}$ the state vector, $\bar{u}$ the input vector and $\bar{y}$ the output vector. The system matrices A and B can be determined from:

$$A = -C_1^{-1}C_2$$
$$B = -C_1^{-1}C_3$$

In a similar manner the state-space equations can be derived for the asymmetric motions of the aircraft. Here the differential operator D_b is equal to $D_b = \frac{b}{V}\frac{d}{dt}$, with *b* equal to the span of the aircraft. The starting point for the derivation is the linearised equation describing the asymmetric motions of the aircraft:

$$\begin{bmatrix} C_{Y_\beta} + \left(C_{Y_{\dot\beta}} - 2\mu_b\right) D_b & C_L & C_{Y_p} & C_{Y_r} - 4\mu_b \\ 0 & -\frac{1}{2}D_b & 1 & 0 \\ C_{\ell_\beta} & 0 & C_{\ell_p} - 4\mu_b K_{xx} D_b & C_{\ell_r} - 4\mu_b K_{xz} D_b \\ C_{n_\beta} + C_{n_{\dot\beta}} D_b & 0 & C_{n_p} - 4\mu_b K_{xz} D_b & C_{n_r} - 4\mu_b K_{zz} D_b \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \varphi \\ \frac{pb}{2V} \\ \frac{rb}{2V} \end{bmatrix} =$$

$$= \begin{bmatrix} -C_{y_{\delta_a}} \\ 0 \\ -C_{\ell_{\delta_a}} \\ -C_{n_{\delta_a}} \end{bmatrix} \cdot \delta_a + \begin{bmatrix} -C_{y_{\delta_r}} \\ 0 \\ -C_{\ell_{\delta_r}} \\ -C_{n_{\delta_r}} \end{bmatrix} \cdot \delta_r$$

The equation can be rewritten using the normal angular roll and yaw rate *p* and *r*, instead of the dimensionless versions $\frac{pb}{2V}$ and $\frac{rb}{2V}$. Again, derive matrices C_1, C_2 and C_3 and implement in Matlab the state-space representation:

$$\dot{\bar{x}} = A\bar{x} + B\bar{u}$$
$$\bar{y} = C\bar{x} + D\bar{u}$$

There are a few ways to obtain simpler equations of motion, among others:

Motion described	**assumption**
short period pitching oscillation	$\hat{u} = 0$
short period pitching oscillation	$\hat{u} = 0$ and $\gamma = \theta - \alpha = 0$
phugoid	$\dot{q} = 0$ and $\alpha = 0$
phugoid	$\dot{q} = 0$ and $\dot{\alpha} = 0$
aperiodic roll motion	$\beta = 0$ and $r = 0$
dutch roll motion	$\varphi = 0$ and $p = 0$

Compare the responses with the full equations of motion for the phugoid motion and another motion of your choice.

5. SIMULINK

5.1. Introduction

Simulink is an acronym of the words "Simulation" en "Link". This name expresses what Simulink has been designed for, the artificial reproduction of physical phenomena by means of simulations. Simulink can be seen as a toolbox belonging to Matlab.

Simulink has a graphic user interface. The user can build the simulation from pre-programmed modules called *blocks*.

The best way to learn about Simulink is to work with it. Therefore this chapter will not give such an extensive description of Simulink as the description of Matlab in chapter 2. The use of Simulink will be demonstrated with the help of an example.

5.2. Simulink example: the trajectory of a meteorite

This example treats the simulation of the path, which a meteorite would follow in the gravitational field of the Earth depending on the initial condition of the meteorite. The mass of the meteorite m is negligible in comparison with the mass of the Earth M. The goal is to make simulations of the trajectories and to draw some conclusions out of them.

5.2.1. Derivation of the equations of motion

First, the equations of motion of the meteorite have to be derived. For this purpose we need a clear drawing of the situation.

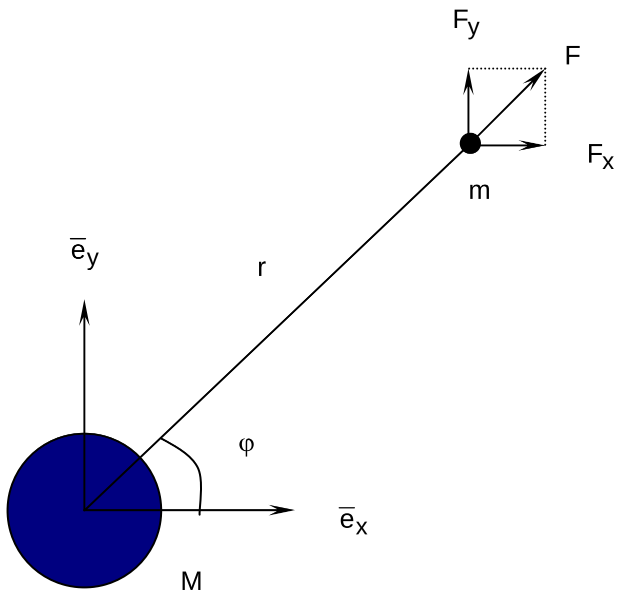


Figure 5.1: A meteorite in the gravitational field of the Earth.

The meteorite is attracted by the mass M with a force F, given by:

$$F = -G\frac{Mm}{r^2} = -\frac{\mu m}{r^2}$$

G is the universal gravity constant and $\mu = GM$. The components of F along the axes are:

$$F_x = -\frac{\mu m}{r^2}\cos\varphi$$

$$F_y = -\frac{\mu m}{r^2}\sin\varphi$$

The position of the mass m can be given in Cartesian co-ordinates as follows:

$$x = r\cos\varphi$$
$$y = r\sin\varphi$$

The velocity of the mass m can be found as the derivative of the position with respect to time t:

$$\dot{x} = \dot{r}\cos\varphi - r\sin\varphi\,\dot{\varphi}$$
$$y = \dot{r}\sin\varphi + r\cos\varphi\,\dot{\varphi}$$

In the same way the acceleration of the mass m can be found as the derivative of the velocity with respect to time t:

$$\ddot{x} = \ddot{r}\cos\varphi - \dot{r}\sin\varphi\,\dot{\varphi} - \dot{r}\sin\varphi\,\dot{\varphi} - r(\cos\varphi\,\dot{\varphi}\,\dot{\varphi} + \sin\varphi\,\ddot{\varphi})$$
$$\ddot{y} = \ddot{r}\sin\varphi + \dot{r}\cos\varphi\,\dot{\varphi} + \dot{r}\cos\varphi\,\dot{\varphi} + r(-\sin\varphi\,\dot{\varphi}\,\dot{\varphi} + \cos\varphi\,\ddot{\varphi})$$

Now that the kinematics of the problem is determined, the equations of motion are easy to find:

$$\ddot{x} = \frac{F_x}{m} \quad\Rightarrow\quad \ddot{r}\cos\varphi - 2\dot{r}\sin\varphi\,\dot{\varphi} - r\cos\varphi\,\dot{\varphi}^2 - r\sin\varphi\,\ddot{\varphi} = -\frac{\mu}{r^2}\cos\varphi$$
$$\ddot{y} = \frac{F_y}{m} \quad\Rightarrow\quad \ddot{r}\sin\varphi + 2\dot{r}\cos\varphi\,\dot{\varphi} - r\sin\varphi\,\dot{\varphi}^2 + r\cos\varphi\,\ddot{\varphi} = -\frac{\mu}{r^2}\sin\varphi$$

Rewriting the equations gives:

$$(\ddot{r} - r\,\dot{\varphi}^2 + \frac{\mu}{r^2})\cos\varphi \quad = \quad (2\dot{r}\,\dot{\varphi} + r\,\ddot{\varphi})\sin\varphi$$
$$-(2\dot{r}\,\dot{\varphi} + r\,\ddot{\varphi})\cos\varphi \quad = \quad (\ddot{r} - r\,\dot{\varphi}^2 + \frac{\mu}{r^2})\sin\varphi$$

Multiplying crosswards:

$$(\ddot{r} - r\,\dot{\varphi}^2 + \frac{\mu}{r^2})^2 = -(2\dot{r}\,\dot{\varphi} + r\,\ddot{\varphi})^2 \quad\wedge\quad \sin\varphi\cos\varphi \neq 0$$

Since both squares are always non-negative and φ is not generally equal to an integer multiple of $\frac{1}{2}\pi$, the only possible solution for the equations of motion is:

$$\ddot{r} - r\,\dot{\varphi}^2 + \frac{\mu}{r^2} \quad = \quad 0$$
$$r\,\ddot{\varphi} + 2\dot{r}\,\dot{\varphi} \quad = \quad 0$$

Show that these equations can also be derived by using polar co-ordinates in a reference system with unity vectors $\overline{e_r}$ and $\overline{e_\varphi}$. Mind the unity vectors $\overline{e_r}$ and $\overline{e_\varphi}$ are *not* time independent. They vary with angle φ however not with radius r.

We found two second order differential equations describing the motion of the meteorite in a flat surface. If we can solve these equations for r(t) and φ(t) the trajectory is known. However it is much easier to transform the equations into four first order equations using a substitution.

First realise how many initial values must be known before the trajectory of the meteorite is fully determined. It's easy to see that one would have to specify the starting position and the initial velocity of the meteorite. Both position and velocity are two-dimensional vectors, so the problem requires four initial values to be given. This brings us back to the concept of the *state* of a mechanical system. Apparently the state of this system is determined by four independent variables. It would be handy if we would be able to rewrite the second order equations we found into four first order differential equations, each describing the course in time of a state of the system.

There is freedom in choosing the state variables, as long as there are four. Suppose one chooses the radius r, the angle φ, the radial velocity V_r and tangential velocity V_φ, so the states are:

$$r\,;\ \varphi\,;\ V_r = \dot{r}\,;\ V_\varphi = r\,\dot{\varphi}$$

From this it can be deduced that:

$$\dot{\varphi} = \frac{V_\varphi}{r} \quad \Rightarrow \quad \ddot{\varphi} = \frac{\dot{V}_\varphi\, r - V_\varphi V_r}{r^2}$$
$$\dot{r} = V_r \quad \Rightarrow \quad \ddot{r} = \dot{V}_r$$

Substituting the expressions for the states in the original equations of motion gives:

$$\dot{V}_r - \frac{V_\varphi^2}{r} = -\frac{\mu}{r^2}$$
$$\frac{\dot{V}_\varphi\, r - V_\varphi V_r}{r} + 2\frac{V_r V_\varphi}{r} = 0$$

or:

$$\dot{V}_r = \frac{V_\varphi^2}{r} - \frac{\mu}{r^2}$$
$$\dot{V}_\varphi = \frac{-V_r V_\varphi}{r}$$

Additionally it was defined:

$$\dot{\varphi} = \frac{V_\varphi}{r}$$
$$\dot{r} = V_r$$

These are four first order differential equations describing the course of the chosen states in time. Note the time derivatives of the states are expressed in the states themselves. Also notice, the time derivatives of the states do not depend on the angle φ. Can this be understood from a physical point of view?

The system is now ready for implementation in the Matlab/Simulink environment.

When the states are collected in the statevector x, the problem to be solved can be summarised as follows:

$$\dot{x} = \begin{bmatrix} \dot{r} \\ \dot{\varphi} \\ \dot{V}_r \\ \dot{V}_\varphi \end{bmatrix} = \begin{bmatrix} V_r \\ \dfrac{V_\varphi}{r} \\ \dfrac{V_\varphi^2}{r} - \dfrac{\mu}{r^2} \\ \dfrac{-V_r V_\varphi}{r} \end{bmatrix}$$

5.2.2. Building the simulation with Simulink

Integrating the state derivatives in $\dot{x}$ gives the states themselves. Of course four initial conditions must be given before integrating. Note that the equations are non-linear and cannot be written in the form of $\dot{x}$ ■Ax, where A is a matrix with constant coefficients. Simulink is very suitable for handling such non-linear problems in a quick and easy way.

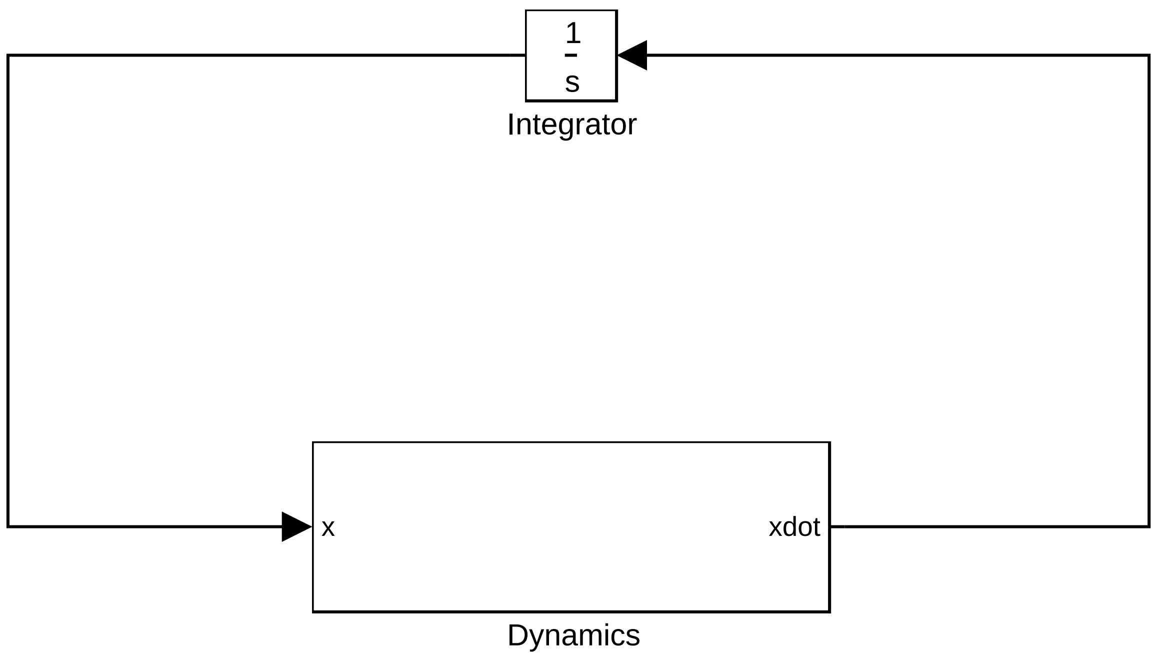


Figure 5.2: Schematic of the way the state trajectories can be determined using Simulink.

In the black box "Dynamics", the derivatives of the state variables are determined from the states themselves using the above formula for $\dot{x}$, which contains the states and a constant ◡ only. The output of the black box is the vector $\dot{x}$. This vector may be seen as a time dependent signal. This signal is integrated with respect to time by the integrator, giving the states as a function of time and enabling to calculate $\dot{x}$. This process continues until the simulation is ended. Simulink has various built-in integration routines, so the user does not have to program any integration algorithms.

Now build the simulation yourself, following these guidelines:

Start Matlab and change to the Matlab command window. Type `simulink` at the Matlab command prompt to start Simulink. Two new windows will be opened, one is the simulink library window and the other is a yet untitled window where the simulation can be built, using blocks from the library. The library contains six sub-libraries, *sources*, *sinks*, *discrete*, *linear*, *non-linear* and *connections*, where many predefined blocks can be found by double-clicking on them with the mouse.

First collect all the blocks that you need from the library. From the sinks menu, drag the To Workspace block to the empty Simulink system window; from the linear menu, drag the Integrator block to the system window; from the Non-linear menu drag the Function (Fcn) block to the system window; and from the connections menu drag the Mux block to the workspace. Consequently build the system as depicted in figure 5.3 below. Don't forget to save your work; name the Simulink system "meteor.mdl".

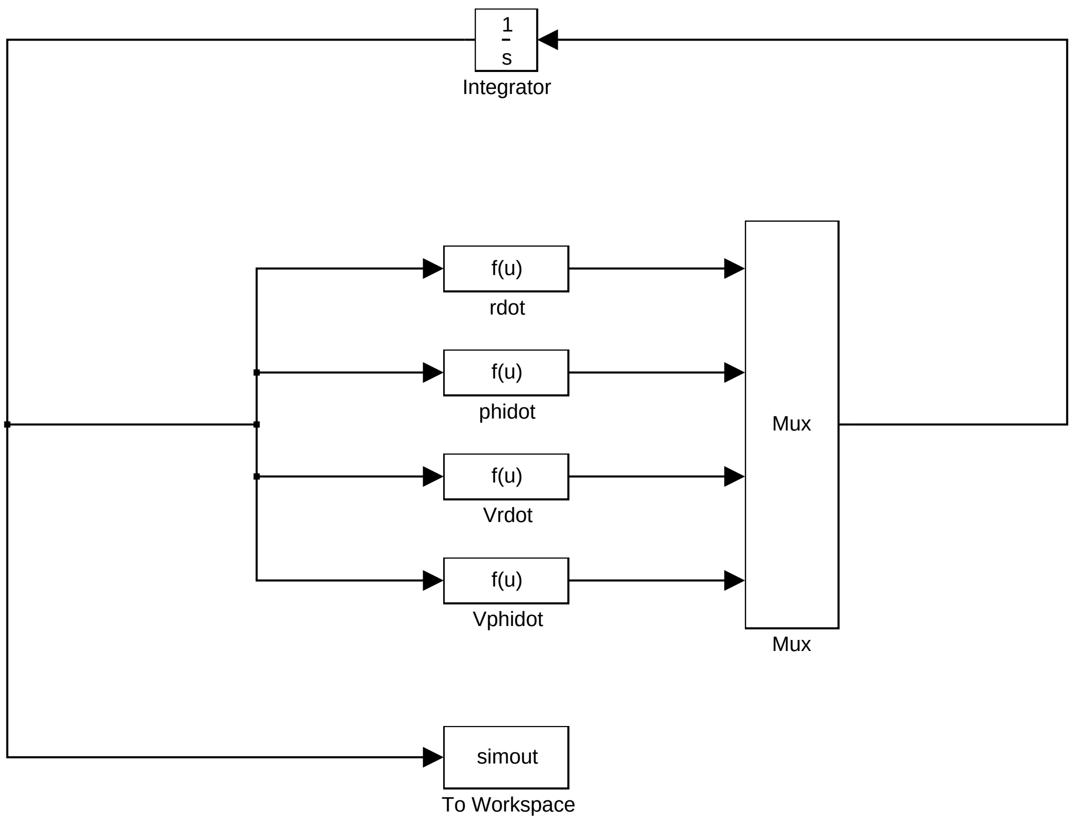


Figure 5.3: Schematic of the Simulink system simulating a meteorite in the Earth's gravitational field.

You can drag and select blocks using the left mouse button. To delete a block, select it and use the del key on your keyboard. To copy a block, drag it to the desired place using the right mouse button and release the right button. The integrator block can be flipped from left to right by selecting it and using the "Flip Block" option in the Format menu. The number of inputs of the Multiplexer block can be increased from three to four by double-clicking on the Mux block and adjusting the number of inputs. Blocks can be resized by selecting them and dragging a corner of the block to the desired size.

To connect the blocks by signal lines, connect the output side of a block to the input side of another block using the left mouse button. Sometimes a line must be built up from different interconnected line segments. Lines can also be selected and deleted. After a bent line has been drawn the corners can be dragged to the desired position. A little circle appears in the corner when the left mouse button is held. To connect a line somewhere in the middle of an existing line, use the right mouse button. Lines are only interconnected when a small black dot appears at the interconnection point.

In order to keep a good overview over the Simulink block diagram, certain blocks may be comprised in a subsystem. In the main diagram this subsystem becomes a "black box" with input signals and output signals. The way to proceed is as follows: select the blocks you want to comprise in a subsystem by dragging a box around them with your left mouse button. The selected blocks become highlighted. From the Edit menu select "Create Subsystem". A subsystem appears, which can be resized and repositioned if necessary.

After double-clicking on the subsystem a new window containing the original blocks of the subsystem is opened. In this window you will also see an input and output block, called "In1" and "Out1". Rename these blocks to "x" and "$\dot{x}$" respectively, according to the names of the input and output signals of the subsystem in the simulation. When clicking on the names a text box appears, and the names can be changed.

Figure 5.4a shows the main system and figure 5.4b the subsystem so far.

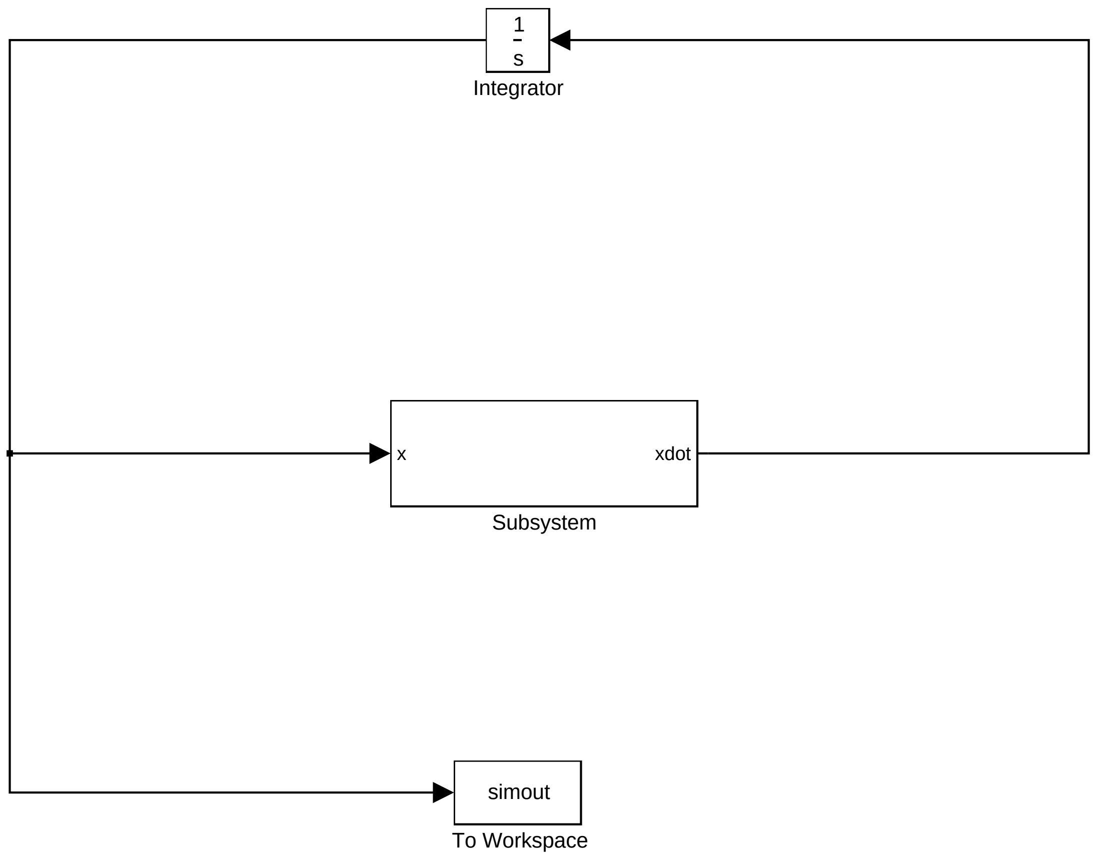


Figure 5.4a: The Simulink block diagram containg a subsystem.

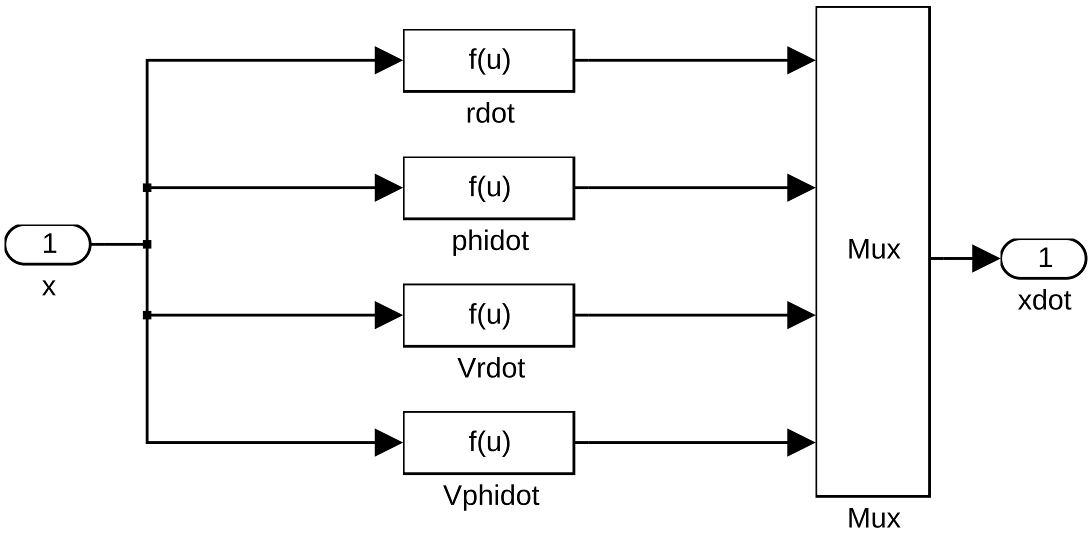


Figure 5.4b: The contents of the subsystem in figure 3a.

The only remaining action is to fill in the four function blocks. The input of each function block is the signal x, which is a four element time-dependent vector. In all function blocks the only valid input variable name is u. The first element of the vector x is automatically renamed in u[1], the second element in u[2], the third element in u[3] and the fourth element in u[4]. Remember that u is now defined as:

$$u = \begin{bmatrix} u[1] \\ u[2] \\ u[3] \\ u[4] \end{bmatrix} = \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} = \begin{bmatrix} r \\ \varphi \\ V_r \\ V_\varphi \end{bmatrix}$$

The first function block must determine $\dot{r}$ from u[1], u[2], u[3] and u[4], the second function must determine $\dot{\varphi}$ from the same parameters, the third V_r and the fourth V_φ.

Find the expressions for the function blocks and enter them by double-clicking each function block. Be careful because the difference between ( ) and [ ] is hardly noticeable in the general expression block.

5.2.3. Running the simulation from a Matlab m-file

Simulink works in close co-operation with Matlab. Although it is not strictly necessary we will run the Simulink simulation from a Matlab m-file. This m-file is very uncomplicated, it starts with the standard commands to clear the Matlab workspace, to close all figures and to clear the Matlab command window.

The simulation will not run until we defined all constants and the initial conditions. The only required constant is μ, but for convenience the radius R of the Earth is also defined. The initial conditions are r0 (distance from the centre of gravity of the Earth), phi0 (angular position of the meteorite), Vr0 (radial velocity) and Vphi0 (tangential velocity). The variables r0, phi0, Vr0 and Vphi0 are defined in the m-file but must also be entered as the vector [r0 phi0 Vr0 Vphi0] in the integrator block in the simulink model. The trajectory of the meteorite is largely dependent on the initial condition. Running the simulation from the m-file has the advantage that the initial conditions can be readjusted all the time and the simulation can be rerun easily from the Matlab command window using the new initial conditions.

The simulation can be started using the `sim` command from Matlab. Type `help sim` in the Matlab command window to learn more about the `sim` command. In our example we use the command line:

```
[T,X] = sim('meteor', [Tstart Tfinal]);
```

In this case the arguments of `sim` are the Simulink model name "meteor" and the integration interval. The output of the simulation appears in the Matlab workspace in the variables T and X. The variable T is a vector containing the points in time where the state variables where determined. The variable X is a matrix containing the states of the system at each point in time contained by the vector T.

The results of the simulation, i.e. the trajectory of the meteorite, can be extracted from the vector X. The first column of X contains the distance r of the meteorite to the centre of gravity of the Earth as a function of time. The second column of X contains the angle θ, the third column the radial velocity V_r and the fourth column the tangential velocity V_θ. On top of the polar co-ordinates r and θ, the Cartesian co-ordinates x and y are calculated in the variables xco and yco. Finally the results can be plotted in different figures by giving the subsequent plotting commands in the m-file.

Running the m-file by entering its name in the Matlab command window will initialise the simulation, run the simulation and plot the results on the screen. The simulation can be run with different initial conditions by adjusting the variables in the m-file and executing the m-file again. In this way different trajectories can be investigated.

To make the simulation more realistic, it would be nice to automatically stop the simulation when the meteorite hits the Earth's surface. This can be accomplished by adding a function block and the "Stop Simulation" block from the *sinks* menu. The expression in the function block can be a Boolean expression that checks whether the distance of the meteorite to the Earth's centre of gravity tends to be equal or less than the radius of the Earth R. When the output of this function (0 for false and 1 for true) is lead to the Stop Simulation block, the simulation will stop as soon as the meteorite hits the Earth's surface. Add these blocks to your simulation and check if it works.

Below you will find the listing of the m-file, which is used to run the simulation.

```
% M-file meteormat.m
% Simulation of a meteorite in the Earth's gravitational field.
% Assumptions: plane motion, no atmospheric friction, no earth rotation.

clear;
close all;
clc;

% Definition of constants

G  = 6.67259e-11;        % Universal gravity constant [m^3/(kg sec^2)]
M  = 5.976e+24;          % Earth's mass [kg]
mu = G*M;                % Product of G and M [m^3/sec^2]
R  = 6370000;            % Earth's radius [m]

% Initial conditions

r0    = 3*R;             % [m]
phi0  = 0*(pi/180);      % [rad]
Vr0   = 0*1000;          % [m/sec]
Vphi0 = 6.38*1000;       % [m/sec]

% Run the simulink system

Tstart = 0;              % [sec]
Tfinal = 5000000;        % [sec]
[T,X] = sim('meteor', [Tstart Tfinal]);

% Results

r    = X(:,1)/1000;      % [km]
phi  = X(:,2)*(180/pi); % [deg]
Vr   = X(:,3)/1000;      % [km/sec]
Vphi = X(:,4)/1000;      % [km/sec]
xco  = r.*cos(X(:,2));   % [km]
yco  = r.*sin(X(:,2));   % [km]

% Plot the results

figure(1)
plot(phi,r);
xlabel('phi [deg]');
ylabel('r [km]');

figure(2)
plot(T/3600,r);
xlabel('time [hrs]');
ylabel('r [km]');

figure(3)
plot(xco,yco);
axis equal;
hold on;
plot([0,0],'k*');
xlabel('x distance [km]');
ylabel('y distance [km]');
```

5.2.4. Simulation results

The results of the simulation are plotted in figures 5.5, 5.6 and 5.7. The figures show the trajectory of a meteorite when the initial conditions are as follows:

$r_0 = 3R$;
$\varphi_0 = 0$;
$V_{r,0} = 0$;
$V_{\varphi,0} = 6.38$ km/sec

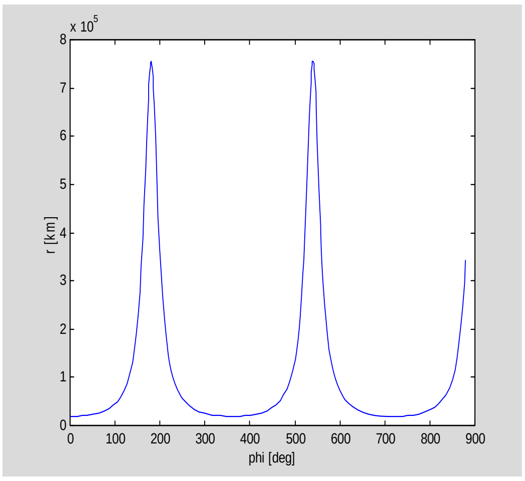


Figure 5.5: The trajectory of the meteorite in polar co-ordinates.

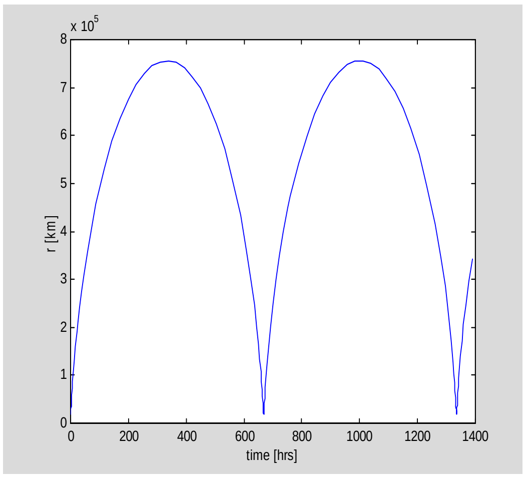


Figure 5.6: The distance of the meteorite to the Earth in time.

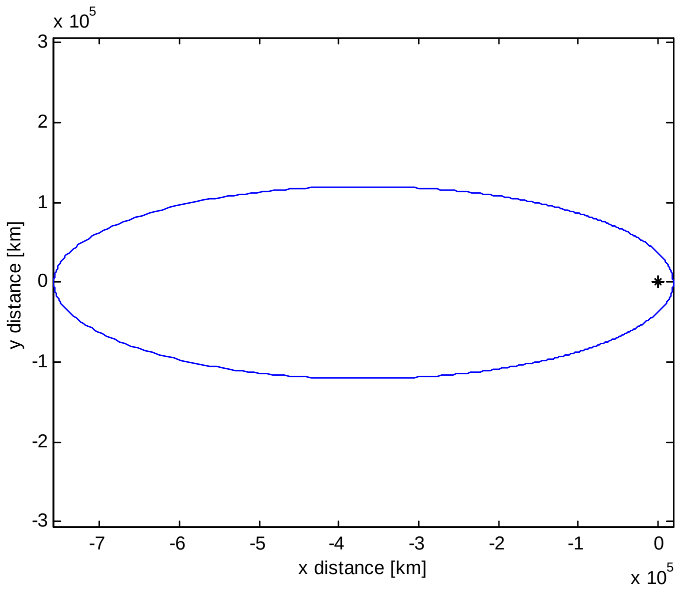


Figure 5.7: The trajectory of the meteorite in Cartesian co-ordinates.

In figure 5.7 the Matlab command `axis equal` is given *after* the plot command in order to get the same scales along the axes. This command will make a circle look like a circle and not an ellipse.

The simulation may be repeated with different initial conditions.

5.2.5. The simulation exercise

The goal of the exercise is to investigate the behaviour of the meteorite in the gravitational field of the Earth. Answer the following questions using the simulation of the trajectory of a meteorite in the gravitational field of the Earth:

Suppose someone would like to launch a meteorite from the Northpole of an Earth-like planet without atmospheric friction. He/she builds a launching platform with an altitude of 10 meters above the surface of the planet. The meteorite will be launched horizontally, i.e. the radial velocity is zero and the tangential velocity component is non-zero.

a. What is the required tangential velocity of the meteorite in order to complete a 360° orbit around the planet? Find this velocity component "experimentally" using your own simulation. It is convenient to let the simulation stop when either the meteorite hits the surface of the planet ($r \leq R$), or the meteorite has completed a 360° orbit around the planet ($\varphi > \varphi_0 + 2\pi$).

The meteorite will now be launched vertically from the surface of the planet, i.e. the radial velocity component is non-zero and the tangential velocity component is zero. This is the way Jules Verne once imagined space travel.

b. What is the minimum required radial velocity in order to send the meteorite into space and never let it return to the planet? Assume that "never" means "not within $1*10^{48}$ seconds".

c. What happens if the meteorite is launched in an angle of 45° with the planet's surface, with the same velocities as found under part b. and c.?

Johannes Keppler

The sixteenth century astronomer Tycho Brahe has made many accurate observations of the motion of the planets known in those days. From these observations Johannes Keppler (1571 – 1630) found that there were five planets moving in elliptical orbits with the sun at one focus. Consequently, the Earth was a planet just like others and in contrast to the general opinion, not the centre of the universe.

He was the first to discover the laws of the planets which later were named after him. The three laws of Keppler are formulated as follows:

1. Each planet describes an elliptical trajectory around the sun, with the sun in the focus of the ellipse (1609).
2. The swept area by the planet is constant along the trajectory (1609).
3. The squares of the orbital periods are proportional to the third powers of the longest axes of the elliptical trajectories (1619).

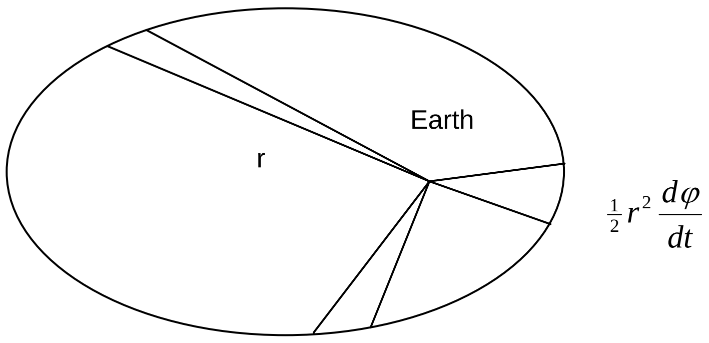


Figure 5.8: The second law of Keppler states that the swept area in a time interval dt remains constant along the trajectory of the meteorite.

d. Do the simulations confirm the first law of Keppler "experimentally"?

e. To check the second law of Keppler, make a plot of the swept area per unit of time as a function of time. Do the simulations confirm the second law of Keppler?

f. To check the third law of Keppler "experimentally", make your simulation such that it stops after completing one orbit around the Earth. Determine the square of the period of the orbit. Next, calculate the length of the longest axis of the elliptical trajectory by determining the minimum and the maximum distance of the meteorite to the centre of mass of the Earth. Repeat this for orbits with different initial conditions and check whether the squares of the periods of the different orbits are proportional to the third powers of the longest axes of the elliptical orbits.

After Keppler had found the three laws from the optical observations of Tycho Brahe, it took another three quarters of a century before Newton explained the kinematic laws of Keppler.

Of course, our simulations are based upon the mechanical laws of Newton and we used the simulations to "experimentally" check Keppler's laws.

5.3. Simulink exercise: symmetric motion of a rigid airship

In this exercise the flight dynamics of an airship are investigated. Only the symmetric motion is considered, so there are three degrees of freedom. A Simulink model is built to investigate the stability of the ship and to improve its design. Some simplifying assumptions will be made, however the simulation is quite realistic.

Reference frames
A right-handed orthogonal body fixed reference frame XYZ is used, with the origin in the intersection of the lines of symmetry of the airship. The X- and Z-axes are lines of symmetry, the X-axis is directed forward and the Z-axis is directed downward in normal upright flight.

A right-handed orthogonal earth fixed (geodetic) reference frame $X_gY_gZ_g$ with the origin in the intersection of the lines of symmetry of the airship. The X_g -axis remains horizontal, positive in the direction of the motion and the Z_g -axis vertical, positive downwards.

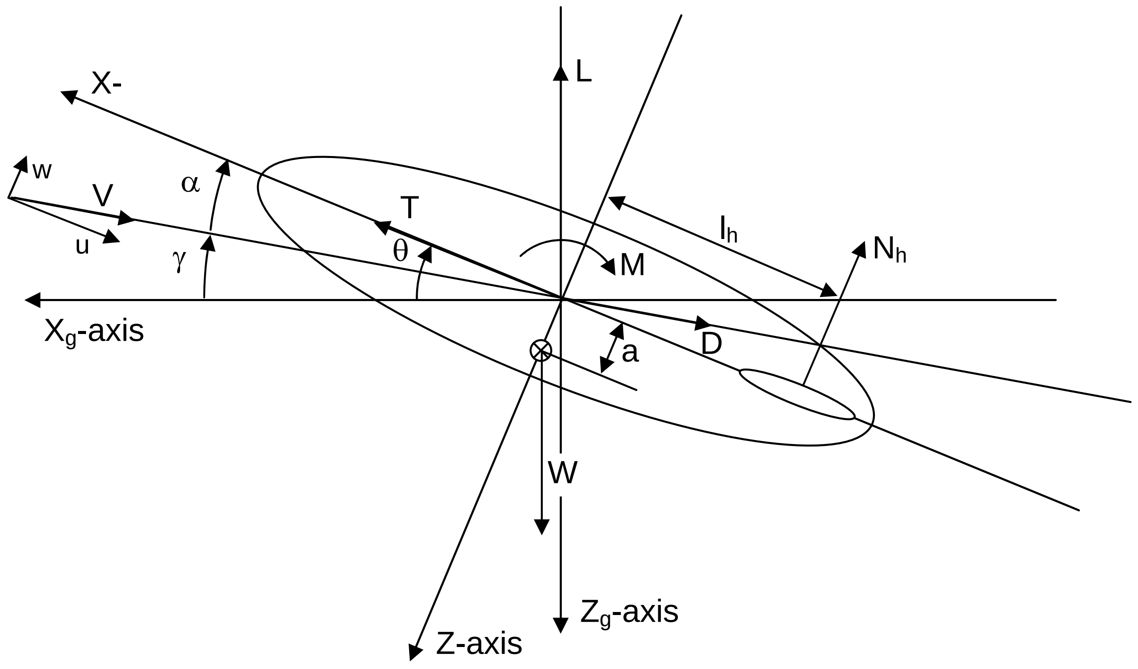


Figure 5.9: *Forces and moments acting on a rigid airship in symmetric flight.*

Forces
The airship is filled with helium and the upward force L is according to Archimedes' law.

$$L = Volume \cdot (\rho - \rho_{helium}) \cdot g$$

The airship is equipped with a horizontal stabiliser to improve its flight characteristics. The stabiliser generates a force N_h depending on the angle of attack α and the dynamic pressure. N_h is assumed to be very small relative to the upward force L.

$N_h = C_{N_{h_\alpha}} \cdot \alpha \cdot \frac{1}{2}\rho V^2 S_h$, $N_h << L$

The propelling force T of the airship is assumed to be constant and independent of the airspeed V. The line of action of T is directed along the positive X-axis.

$T = T_0$

Due to friction with the surrounding atmosphere, a drag force D exists, which is directed parallel to the undisturbed incoming airflow.

$D = C_{D_0} \frac{1}{2}\rho V^2 S$

The centre of mass is assumed to be somewhere along the Z-axis, a distance *a* from the intersection of the lines of symmetry. The weight W of the airship is assumed to be constant during flight.

$W = mg$

Moments
The airship can be considered as a fuselage, placed under a non-zero angle of attack in an incompressible flow. The pressure distribution around the fuselage results in an aerodynamic moment that tends to increase the angle of attack. The magnitude of the moment acting on a slender fuselage of circular cross section in an incompressible, inviscid flow has been derived by Munk, ref. Lecture Notes Flight Dynamics page 146:

$M_\alpha = \rho V^2 \cdot \alpha \cdot (Volume\ of\ \ fuselage)$

In this simplified model of an airship it is assumed that the q-motion of the fuselage causes a moment about the Y-axis resisting the q-motion. This moment is given by:

$M_q = -1.1 \cdot C_{N_{h_\alpha}} \cdot S_h l_h^2 \cdot \frac{1}{2}\rho V \cdot q$

The normal force on the horizontal stabiliser causes a moment about the centre of mass, depending on the angle of attack α, the dynamic pressure ½ρV and the tail length l_h.

In case the centre of mass is not located in the origin of the reference frames but a distance *a* below it, upward force L, propelling force T and drag D cause moments about the centre of mass.

<u>Answer the following questions</u>:

a. Derive the symmetric equations of motion of the airship, use the general equations (3-18) from the Lecture Notes Flight Dynamics:

$$F_x = m(\dot{u} + qw - rv) \qquad M_x = I_x \dot{p} + (I_z - I_y)qr - J_{xz}(\dot{r} + pq)$$

$$F_y = m(\dot{v} + ru - pw) \qquad M_y = I_y \dot{q} + (I_x - I_z)rp + J_{xz}(p^2 - r^2)$$

$$F_z = m(\dot{w} + pv - qu) \qquad M_z = I_z \dot{r} + (I_y - I_x)pq - J_{xz}(\dot{p} - rq)$$

b. Since the system has three degrees of freedom it must have six state variables. Complete the set of equations found under part a. with three more first order differential equations, so as to describe the course in time of all six state variables. As a state vector choose $\bar{\mathbf{x}} = [u\ w\ q\ \theta\ x\ z]^T$. x and z give the position of the airship as a function of time.

c. Derive expressions for the output vector, defined as $\bar{\mathbf{y}} = [\alpha\ \gamma\ h\ V]^T$, given the state vector $\bar{\mathbf{x}}(\mathrm{t})$.

α	angle of attack
γ	flight path angle
h	height
V	resulting airspeed

d. Create a Simulink model and an m-file to run a simulation of the motion of the airship. The result of the model should be the state vector and the output vector as a function of time. For now, assume that the centre of mass is placed on the X-axis (a = 0) and the air density remains constant. Choose your own tailplane area S_h. Furthermore, use the following numerical values:

```
% Constants

g    = 9.80665;          % gravity constant [m/sec^2]
rho  = 1.2250;           % air density at sea level [kg/m^3]
rhom = 0.178;            % helium density in the airship[kg/m^3]

% Geometry of the airship

S    = 300;              % frontal area [m^2]
Sh   = ?;                % horizontal tailplane area [m^2]
Vol  = 30000;            % volume [m^3]
lh   = 50;               % tail length [m]
m    = 31300;            % mass of the airship [kg]
a    = 0;                % z-coordinate of centre of mass [m]
Iyy0 = 625*m;            % moment of inertia about the Y-axis
                         % [kgm^2]

% Forces and moments on the airship

T0   = 3000;             % thrust in [N]
CD0  = 0.025;            % drag coefficient [ ]
CNha = (2*pi*Sh)/(Sh+2); % derivative of normal force
                         coefficient on the horizontal tailplane
                         with respect to alpha [ ]
```

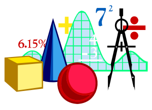
e. Is it possible to stabilise the airship with the horizontal stabiliser? What is the minimum required size of the horizontal stabiliser? Have a look at the simulation results on different time scales. What happens to the pitch angle in the long term?

f. In a well-designed airship the angle of attack remains within limits and is subject to damping. What force and/or moment is responsible for the damping in the angle of attack?

g. In order to increase passenger comfort, it is desirable to limit the pitch angle to a maximum of 15 degrees. Is it possible to design the airship such that this limitation is not exceeded by lowering the centre of gravity of the airship? What values of tailplane area S_h and distance *a* are optimal according to your opinion? Did the change in design increase or decrease the climb performance?

h. So far the air density ρ has been taken as a constant. In reality however, the density of the atmosphere decreases as the airship climbs. To get a more realistic simulation, implement the change of the air density with height in the standard atmosphere in your simulation. What are the results compared to part f.? What altitude does the airship reach? What is the influence on stability?

5.4. SIMULINK EXERCISE: SYMMETRIC MOTION OF AN AIRCRAFT



In this exercise the longitudinal flight dynamics of an aircraft are investigated. The influence of the location of the centre of gravity of the aircraft on the stability will become clear. As an example the Cessna Citation laboratory aircraft is used.

5.4.1. Reference frames

For the simulation two reference frames are important:

- A right-handed orthogonal body fixed reference frame F_B ($OX_BY_BZ_B$), with the origin in the centre of gravity of the aircraft. The positive X_B-axis is directed forward along the longitudinal axis and the positive Z_B-axis points downward in normal, upright flight.

- A right-handed orthogonal earth fixed (geodetic) reference frame F_E ($OX_EY_EZ_E$) with the origin in the centre of gravity of the aircraft. As the aircraft moves, the X_E -axis remains horizontal, positive in the direction of the initial motion and the Z_E -axis remains vertical, positive towards the centre of the earth.

These reference frames are similar to those used in the airship exercise in section 5.3.

5.4.2. Forces

The forces acting on the aircraft are the lift generated by the wing L_w and the horizontal stabiliser L_h, the aerodynamic drag D, engine thrust T_e and gravity force W.

In this exercise the resulting aerodynamic force R will be given in terms of the total normal force N acting along the negative Z_B axis and the total tangential force T acting along the positive X_B axis.

The relation between lift, drag, normal force and tangential force is given by:

$$\begin{aligned} N &= +L\cos(\alpha) + D\sin(\alpha) \\ T &= -L\sin(\alpha) + D\cos(\alpha) \end{aligned}$$

, where α is the angle of attack of the wing.

To reduce this example to its simplest form, while conserving the essence of symmetric aircraft motion, the following assumption is made:

$$\begin{aligned} N &= L \\ T &= D \end{aligned}$$

which is valid for small angles of attack only.

The total normal force N is given by the sum of the normal force on the wing N_w and on the horizontal stabiliser N_h:

$$N = N_w + N_h$$

The total normal force is expressed in its dimensionless coefficients as:

$$N = C_{N_w} \cdot \tfrac{1}{2}\rho V^2 S + C_{N_h} \cdot \tfrac{1}{2}\rho V^2 S \left(\frac{V_h}{V}\right)^2 \frac{S_h}{S}$$

$\tfrac{1}{2}\rho V^2$ is the dynamic pressure, *S* and *S_h* are the wing area and horizontal stabiliser area respectively and *V_h* is the true airspeed at the horizontal stabiliser.

In the same way the total tangential force is expressed in its dimensionless coefficient as:

$$T = C_T \tfrac{1}{2}\rho V^2 S$$

The engine thrust T_e is assumed to remain constant throughout the manoeuvres:

$$T_e = constant$$

5.4.3. Moments

The moments acting on the aircraft are the aerodynamic moments about the aerodynamic centre of the wing $M_{ac,w}$ and the horizontal stabiliser $M_{ac,h}$ and the moments about the centre of gravity of the aircraft caused by the normal force *N*, tangential force *T* and engine thrust T_e.

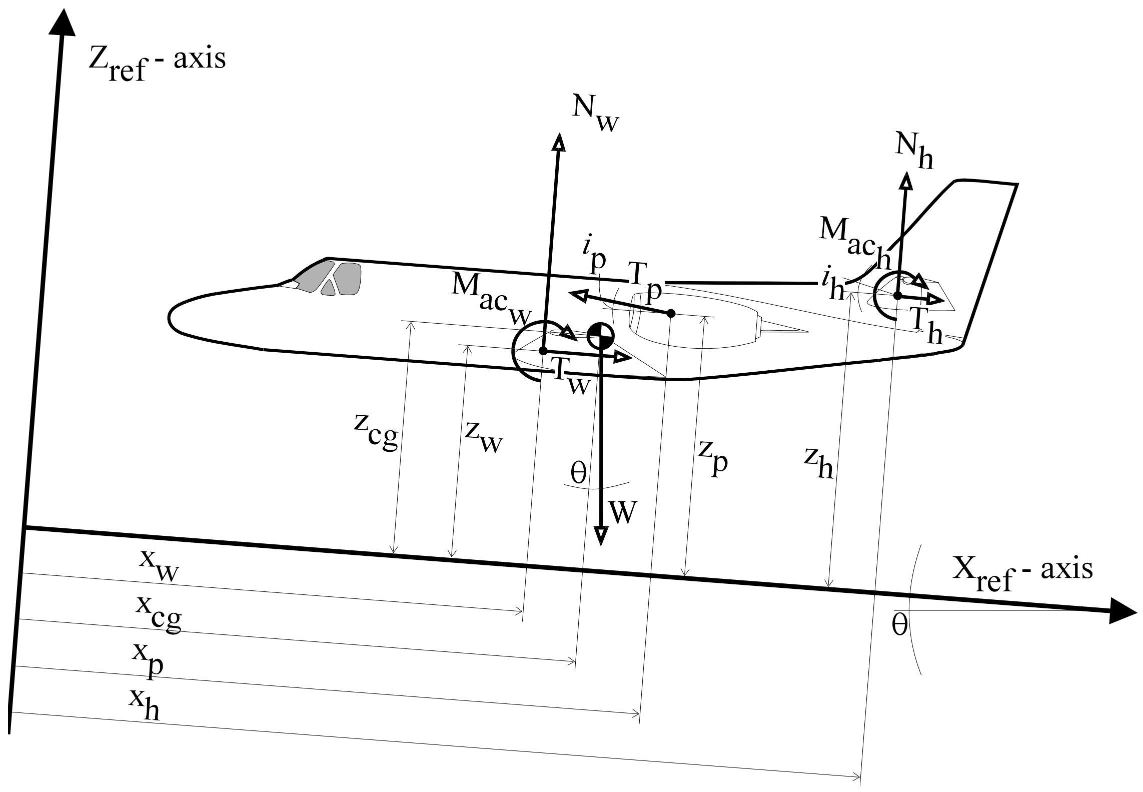


Figure 5.10: Forces and moments acting on the aircraft in symmetric flight

Figure 5.10 can be used as a reference, keeping in mind the simplifying assumptions that the aerodynamic centre of both wing and horizontal stabiliser are fixed points, and:

$$z_w = z_p = z_{cg}$$
$$T_h = 0$$
$$M_{ac,h} = 0$$
$$i_p = 0$$

The total aerodynamic moment is defined as:

$$M = C_m \tfrac{1}{2} \rho V^2 S \bar{c}$$

$\bar{c}$ is the mean aerodynamic chord of the wing.

5.4.4. Force and moment coefficients

The normal force coefficients of the wing and horizontal stabiliser

$$C_{N_w} = C_{N_{w_\alpha}} (\alpha - \alpha_0)$$
$$C_{N_h} = C_{N_{h_\alpha}} \alpha_h + C_{N_{h_\delta}} \delta_e$$

$\alpha_0 = \alpha |_{C_N = 0}$
α_h is the angle of attack at the horizontal stabiliser
δ_e is the deflection of the elevator, positive downwards
$C_{N_{w_\alpha}}$ and $C_{N_{h_\alpha}}$ are the derivatives of C_{N_w} and C_{N_h} with respect to angle of attack.

$$C_{N_{w_\alpha}} = \frac{2\pi A_w}{A_w + 2}$$

$$C_{N_{h_\alpha}} = \frac{2\pi A_h}{A_h + 2}$$

A_w

A_w and A_h are the aspect ratios of horizontal stabiliser and wing respectively.

For a more detailed description refer to the Lecture Notes "Flight Dynamics", for example section 6.4.

The angle of attack at the horizontal stabiliser is not the same as the angle of attack at the wing, due to the downwash caused by the wing.

$$\alpha_h = (\alpha - \alpha_0)\left(1 - \frac{d\varepsilon}{d\alpha}\right) + (\alpha_0 + i_h)$$

ε is the downwash angle, see also section 5.2.5. of the "Flight Dynamics" Lecture Notes.

$$\frac{d\varepsilon}{d\alpha} = \frac{4}{A_w+2}$$

i_h is the angle of incidence of the horizontal stabiliser relative to the X_B-axis, refer to figure 5.10 or figure 5.11.

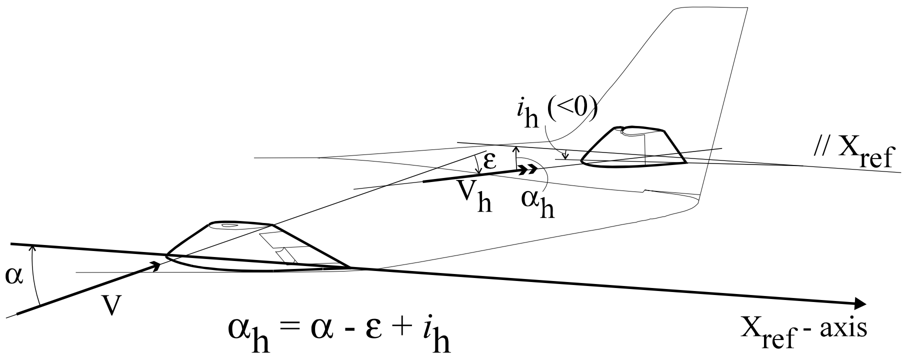


Figure 5.11. Definition of the local stabiliser angle of attack, α_h

The tangential force coefficient

$$T = C_T \tfrac{1}{2}\rho V^2 S = D = C_D \tfrac{1}{2}\rho V^2 S$$

$$C_D = C_{D_0} + \frac{C_{L_w}^2}{\pi A e}$$

$$C_{L_w} = C_{N_w} = C_{N_{w\alpha}}\left(\alpha - \alpha_0\right) = \frac{2\pi A_w}{A_w+2}\left(\alpha - \alpha_0\right)$$

$$C_T = C_{D_0} + \frac{4\pi A_w}{\left(A_w+2\right)^2 e}\left(\alpha - \alpha_0\right)^2$$

The moment coefficients

$$C_m = C_{m_0} + C_{m_\alpha}\left(\alpha - \alpha_0\right) + C_{m_\delta}\delta_e$$

in which:

$$C_{m_0} = C_{m_{\text{a.c.}}} - C_{N_{h_\alpha}}\left(\alpha_0 + i_h\right)\left(\frac{V_h}{V}\right)^2 \frac{S_h \ell_h}{S\,\overline{c}}$$

is constant; with $\ell_h = x_h - x_{cg}$

$$C_{m_\alpha} = C_{N_{w\alpha}} \frac{x_{\mathrm{cg}} - x_{\mathrm{w}}}{\overline{c}} - C_{N_{h\alpha}} \left(1 - \frac{\mathrm{d}\varepsilon}{\mathrm{d}\alpha}\right)\left(\frac{V_h}{V}\right)^2 \frac{S_h \ell_h}{S\,\overline{c}}$$

is the *static longitudinal stability.*

$$C_{m_\delta} = -C_{N_{h\delta}} \left(\frac{V_h}{V}\right)^2 \frac{S_h \ell_h}{S\,\overline{c}}$$

is called the *elevator effectiveness*.

5.4.5. Questions

Answer the following questions:

1. Derive the symmetric equations of motion of the aircraft by simplifying the general equations of motion (3-18) from the Lecture Notes "Flight Dynamics":

$$
\begin{aligned}
F_x &= m(\dot{u} + qw - rv) & M_x &= I_x\dot{p} + (I_z - I_y)qr - J_{xz}(\dot{r} + pq) \\
F_y &= m(\dot{v} + ru - pw) & M_y &= I_y\dot{q} + (I_x - I_z)rp + J_{xz}(p^2 - r^2) \\
F_z &= m(\dot{w} + pv - qu) & M_z &= I_z\dot{r} + (I_y - I_x)pq - J_{xz}(\dot{p} - rq)
\end{aligned}
$$

   Substitute forces and moments (N_w, N_h, T, M_{ac}, T_e, W) into the equations.

2. Since the system has three degrees of freedom (two translations and one rotation) it must have six state variables. Complete the set of equations previously found with three more first order differential equations, so as to describe the course in time of all six state variables. As a state vector choose $\bar{\mathbf{x}} = [u\ w\ q\ \theta\ x\ z]^T$. The state variables x and z give the position of the airship in the geodetic reference frame as a function of time.

3. Derive expressions for the output vector, defined as $\bar{\mathbf{y}} = [\alpha\ \ \gamma\ \ h\ \ V\ \alpha_h]^T$, given the state vector $\bar{\mathbf{x}}(\mathrm{t})$.

α	angle of attack
γ	flight path angle
h	height
V	resulting true airspeed
α_h	angle of attack at the horizontal stabiliser

4. Create a Simulink model and an m-file to run a simulation of the motion of the aircraft. The result of the model should be the state vector $\bar{\mathbf{x}}$, the output vector $\bar{\mathbf{y}}$ and the forces and moments N_w, N_h, T and M as a function of time. Show the structure of your model in a few graphs or drawings.

   It will be assumed that the air density remains constant. Furthermore, use the following numerical values:

```
% Constants
g    = 9.80665;          % gravity constant [m/sec^2]
rho  = 1.225;            % air density at sea level [kg/m^3]

% Geometry of the aircraft
S    = 30;               % wing area [m^2]
Sh   = 5;                % horizontal tailplane area [m^2]
A    = 8.28;             % wing aspect ratio []
Ah   = 6;                % horizontal stabiliser aspect ratio []
c    = 2.02;             % wing mean aerodynamic chord [m]
lh   = 6;                % tail length [m]
ih   = -2.2*(pi/180);    % stabiliser angle of incidence [rad]
```

```
% Inertia of the aircraft
m        = 5000;         % mass of the aircraft [kg]
Iyy      = 12000;        % moment of inertia about Y-axis[kgm^2]

% Aerodynamic constants
CD0      = 0.08;         % AoA independent drag coefficient []
e        = 0.8;          % Oswald factor []
alfa0    = -1*(pi/180);  % zero lift angle of attack [rad]
Vh_V     = 1;            % airspeed ratio stabiliser-wing []
Cmac     = -0.1;         % moment coefficient about the a.c. []
CNhd     = 1.735;        % derivative of CNh to elevator angle[]
xcg_xw_c = 0.175;        % dimensionless distance c.g. to a.c.
                         % of the wing []

% Constant forces and moments on the aircraft
T0       = 17348;        % engine thrust [N]
W        = m*g;          % aircraft weight [N]
```

In order to evaluate the flight dynamics of the aircraft in this configuration, it needs to be trimmed for equilibrium. With the following initial conditions the aircraft should be trimmed for a horizontal flight with an airspeed of approximately 100 m/sec.

```
u0  = 100.123;            % initial condition for u
w0  = 3.8458;             % initial condition for w
q0  = 0;                  % initial condition for q
th0 = 2.1997*pi/180;      % initial condition for theta
x0  = 0;                  % initial condition for x position
z0  = -1000;              % initial condition for z position
```

The aircraft will only fly straight if the elevator deflection is such that moment equilibrium exists. The required elevator deflection can be easily derived. Refer also to section 5.2.7 of the Lecture Notes of "Flight Dynamics".

$$\delta_{e_0} = -\frac{1}{C_{m_{\delta_e}}}\left\{C_{m_0} + C_{m_\alpha}\left(\alpha - \alpha_0\right)\right\}$$

In this formula α is the *initial* angle of attack at $t = 0$ and δ_{e_0} is a *constant*. The initial angle of attack can be calculated from u_0 and w_0 as given above. Calculate δ_{e_0} in this way and keep it constant in the simulation.

<u>Tips for building a successful simulation</u>:

- Read chapter 5 at home before starting.
- Follow the problem solving methodology given in section 3.4.
- Define all constants (given or calculated) in a Matlab m-file. State variables, output variables, forces and moments will be calculated by the Simulink model. Constants defined in the Matlab workspace can be freely used in the Simulink model. Run the Simulink model from the m-file. (Give the m-file and Simulink model different names!).
- Start building the Simulink model on a piece of paper. Design the model in terms of black boxes. Determine what the input and output of each black box should be and what the interconnection structure between the black boxes is. Fill in the black

boxes later. Remember that a good preparation will save you a lot of time in finding errors in a later stage.
- Use at least 5 black boxes, one for each of the following items: calculation of the states (equations of motion), calculation of the output, calculation of the force and moment coefficients, calculation of the dynamic pressure and calculation of the forces and moments. Whenever convenient make use of the "Create subsystem" command from the Simulink Edit menu.
- Output of the m-file should be at least 15 different plots, 6 for the states, 5 for the outputs and 4 for the forces and moments. Plot all angles in degrees, not in radians. Note: all calculations with angles must be done in radians, not degrees.
- If your model doesn't run, first check for syntax errors. Once these are solved and the simulation still doesn't run, try a very short simulation time to see what is going on. Use all available plots to check if forces and moments acting on the aircraft are as expected. This will help you find and eliminate problems.
- Remember the simulation looses its validity at large angles of attack. At an angle of attack larger than 10°, the aircraft will inevitably stall. The angle of attack is an important output variable to check whether the aircraft equilibrium is stable or not.

5. 10 seconds after starting the simulation give a step input of minus 1 degree on the elevator. Use the "step" block from the "Sources" library and set the step time to 10 seconds, the initial value to δ_{e_0} and the final value to δ_{e_0} *- 1*π/180* . Check the responses at two different timescales: [0,200 sec] and [0,30 sec].

   a. Give a plot of airspeed *V* against time and a plot of γ against time on the interval [0,200 sec].
   b. Has a new equilibrium been established after 200 seconds? If so, what is the new equilibrium airspeed *V* and what is the new equilibrium flight path angle γ?
   c. What is the characteristic motion called that can be seen from the plots?
   d. Give a plot of pitch rate *q* against time and a plot of angle of attack α against time on the interval [0,30 sec].
   e. What is the characteristic motion called that can be seen from the plots? Describe this short term behaviour you see from the plots in your own words.
   f. Take a look at the plot of flight path angle against time and zoom in on the area where γ starts to deviate from zero. Use the zoom function available in each figure. Does the aircraft start to climb right after the step was applied on the elevator? Clearly explain what happens.

6. The stability of the aircraft depends on the location of the centre of gravity. The simulation enables us to experiment with this. First remove the step input from the model. Move the centre of gravity 7.5 % of the chord length rearward and run the simulation for 500 seconds. Check the responses. Since we moved the centre of gravity, the aircraft is no longer trimmed for a horizontal flight. Look at the flight path angle and adjust the elevator deflection such that the aircraft flies level within at least plus or minus 0.05 degrees. Then use the steady-state values you find around t = 500 sec as the new initial conditions for u_0, w_0, and θ_0. These steady-state values can be found "by hand" using the zoom function in the plots of *u*, *w*, and θ. Of course q_0 always remains zero.

   a. With this new centre of gravity location plot the altitude h against time. Run the simulation for 200 seconds. State the location of the centre of gravity in the plot.
   b. What is the magnitude of the normal force on the horizontal stabiliser in this situation? Give a plot of N_h and state the location of the centre of gravity in the plot.

c. Move the centre of gravity even more rearward, trim the aircraft for horizontal flight and plot the altitude h against time. Run the simulation for 200 seconds. State the location of the centre of gravity in the plot.
d. What is the magnitude of the normal force on the horizontal stabiliser in this situation? Give a plot of N_h and state the location of the centre of gravity in the plot. Has the tail load increased or decreased compared to the previous situation?
e. What is the advantage of moving the centre of gravity rearward with regard to the tail load? Is it possible to save fuel this way? Explain.
f. What is the critical location of the centre of gravity? To find the location keep an eye on the angle of attack.

7. There are many ways to improve the simulation and make it much more realistic. As an example we will take the effect of the pitch rate *q* on the aerodynamic moment into consideration. This influence can be expressed by the following stability derivative:

$$C_{m_q} = -1.1 \cdot C_{N_{h\alpha}} \left(\frac{V_h}{V}\right)^2 \left(\frac{S_h}{S}\right)\left(\frac{\ell_h}{\bar{c}}\right)^2$$

Calculate C_{m_q} in your m-file and add the resulting aerodynamic moment in your simulink model. Show what happened in a few characteristic plots. Explain.

6. THE FLIGHT TEST

6.1. INTRODUCTION

A substantial part of the ae3-303P experimental program consists of a test flight with the Cessna Citation II laboratory aircraft. In some years and for some students the time between the flight and hand-in of the report is rather short. However, it is very well possible to create an m-file that is able to process the flight data long before the flight has taken place. Once the flight data becomes available it can be processed very quickly. For a good procedure to prepare an m-file, refer to chapter 3. In this chapter we will prepare the flight by making a Matlab program that determines the elevator trim and elevator control force curves from measured flight test data and by making linear simulations of the Cessna Citation symmetric and asymmetric flight dynamics. Refer to part II of the ae3-303P manual for more detailed information on the test flight.

6.2. EXERCISE 1: ELEVATOR TRIM AND ELEVATOR CONTROL FORCE CURVES

Goal of the exercise:

- To produce a Matlab generated plot of the aircraft weight as a function of time
- To produce a Matlab generated plot of the position of the centre of gravity in % of the mean aerodynamic chord as a function of time
- To determine the elevator trim curve $\delta_e^* - \tilde{V}_e$ from flight test data
- To determine the elevator control force curve $F_e^* - \tilde{V}_e$ from flight test data

See also sections 2.1 and 4.4 of the manual ae3-303P – Part II.

It is possible to program m-files to process data long before the actual data is available from an experiment. In this way the flight test may be prepared beforehand. In this case we will create an m-file that will process the measurement data from the flight test. Refer to chapter 3 of the manual for a structured manner to handle this problem.

One of the purposes of the flight test is to determine the elevator trim and the elevator control force curves. In order to determine these curves a number of measurements will be taken during stationary flight conditions. The variables that are measured in flight are:

a. elapsed time from the moment the FTIS has started up, in minutes and seconds
b. the pressure altitude, in feet
c. the indicated airspeed, in knots
d. the angle of attack, in degrees
e. the elevator deflection angle, in degrees
f. the elevator trim deflection angle, in degrees
g. the elevator control force, in Newton
h. the fuel flow of the left engine, in pounds per hour
i. the fuel flow of the right engine, in pounds per hour
j. the amount of fuel used from the moment the elapsed time was set to zero, in pounds
k. the total air temperature, in degrees Celsius

The mentioned variables will be collected on the Post-Flight Data Sheet after the flight.

In order to derive accurate results, the weight and balance condition of the aircraft must be known. Therefore the aircraft empty weight, the weights of the occupants, the fuel onboard before starting the engines and the fuel used after engine start and before start up of the FTIS must also be input to the m-file. The weight and balance condition of the aircraft can be deduced from the Technical Weighing Report ae3-303P and the Post-Flight Data Sheet.

Finally the engine thrust of both engines must be specified for each stationary flight condition. The engine thrust can be calculated using the program VL-208.exe.

Summarising, the following input data is required:

```
% Input of stationary measurements

ET_sec    =    % [sec]                    Fe_N      =    % [N]
hp_ft     =    % [ft]                     FFl_lbshr =    % [lbs/hr]
IAS_kts   =    % [kts]                    FFr_lbshr =    % [lbs/hr]
alpha_deg =    % [deg]                    Fused_lbs =    % [lbs]
de_deg    =    % [deg]                    Tt_C      =    % [deg C]
detr_deg  =    % [deg]

% Input of weight and balance data

mpf    =    % mass of the pilot flying [kg]
mpnf   =    % mass of the pilot not flying [kg]
mco    =    % mass of the experiment co-ordinator [kg]
mobs1L =    % mass of the observer 1L [kg]
mobs1R =    % mass of the observer 1R [kg]
mobs2L =    % mass of the observer 2L [kg]
mobs2R =    % mass of the observer 2R [kg]
mobs3L =    % mass of the observer 3L [kg]
mobs3R =    % mass of the observer 3R [kg]

BEW_lbs =   % Aircraft Basic Empty Weight from W&B form [lbs]
FOB_lbs =   % Fuel On Board before starting engines [lbs]
SUF_lbs =   % Start Up Fuel [lbs]

% Input of the calculated thrust

Tl   =   % left hand engine thrust [N]
Tr   =   % right hand engine thrust [N]
Ts   =   % total standardised engine thrust [N]
```

The exercise is to create an m-file that processes this input and draws the elevator trim curve and the elevator control force curve. For the time being, assume that the number of stationary measurements is n. Use the measurements from flight 3 of November 19, 1998 as given on the post flight data sheet.

The first step is to convert the measured variables to SI units. It is important to find variable names that are logic and indicate clearly what the variable contains and in what units. An option is to assume that a variable is in SI units when no underscore is added to the variable. In case a variable is not in SI units, an underscore may indicate in what unity the variable is given.

Throughout the m-file a lot of constants and conversion factors will have to be used. It is wise to collect these constants in a separate section of the m-file. When you want to look up or change a constant you know where to find it. Also, someone that has not created the m-file can become familiar with it more easily.

After the definition of constants and the conversion of all measured variables to SI units, the air data calculations can be executed: the static air pressure p, the Mach number M, the static air temperature T, the static air temperature according to ISA T_{ISA}, the difference between the actual and the ISA temperature ΔT_{ISA}, the air density ρ, the speed of sound a, the true air speed V_t and the equivalent airspeed V_e.

Next, all weight calculations can be performed, resulting in the actual aircraft weight in Newton at every measurement point. The non standard aircraft weight can be reduced to obtain the reduced equivalent airspeed.

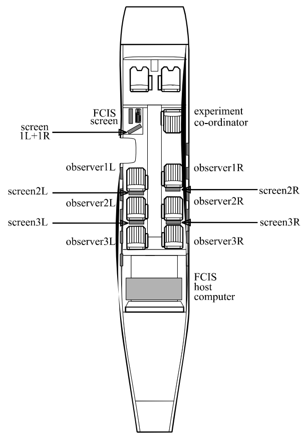
The non-standard engine thrust can be reduced to obtain the reduced elevator deflection using the guidelines in section 4.1.1 of the ae3-303P Manual Part II "Drawing the reduced elevator trim curve".

Interpolate the static unbalance calibration graph to find $F_{e_{unb}}$. From this value and the measured control force, determine $F_{e_{aer}}$ in each measurement point. The aerodynamic control force component can then be reduced using the chosen standard aircraft weight. Using the guidelines from section 4.1.2. of the ae3-303P Manual Part II "Drawing the reduced elevator control force curve", the control force curve can be plotted using Matlab.

Example Post-Flight Data Sheet ae3-303P

date of flight: 19/11/98	T/O time: 14:50
flight number: 3	LND time: 16:05

Weights

	name	mass [kg]	
pilot 1		71	
pilot 2		72	
co-ordinator		74	
observer 1L		69	
observer 1R		87	
observer 2L		78	
observer 2R		72	
observer 3L		84	
observer 3R		81	

block fuel	4480 lbs
start-up fuel	35 lbs

Stationary measurements

Aircraft configuration	clean
Number of measurements	9

nr.	time	ET[*]	h_p (ft)	IAS (kts)	α (deg)	δ_e (deg)	δe_{tr} (deg)	F_e (N)	FF_l (lbs/hr)	FF_r (lbs/hr)	Fuel (lbs)	TAT (°C)
1	15:02	12:00	7960	160	3.9	-0.7	2.2	-2	411	420	278	-9.0
2	15:04	13:40	8300	148	5.2	-1.4	2.2	-26	404	414	300	-10.0
3	15:05	15:00	8550	138	6.1	-1.7	2.2	-33	403	411	316	-11.0
4	15:06	16:35	8800	126	7.6	-2.4	2.2	-44	396	407	336	-11.8
5	15:08	18:15	9040	114	9.5	-3.3	2.2	-58	393	401	360	-12.7
6	15:10	20:20	8090	171	3.2	-0.3	2.2	18	412	422	385	-8.5
7	15:12	21:44	7780	181	2.6	-0.1	2.2	39	418	426	406	-7.8
8	15:14	24:17	6730	190	2.1	+0.2	2.2	76	432	439	442	-6.2
9	15:15	25:30	6130	200	1.8	+0.4	2.2	96	441	447	462	-4.1
10												

[*] ET = Elapsed Time

Shift in center of gravity

name:	moved to position: btwn 3L and 3R
position: experiment co-ordinator	

	time	ET[*]	h_p (ft)	IAS (kts)	α (deg)	δ_e (deg)	δe_{tr} (deg)	F_e (N)	FF_l lbs/hr	FF_r lbs/hr	Fuel used (lbs)	TAT
before	15:19	28:06	8000	163	3.6	-0.6	2.2	-1	412	422	510	-9.0
after	15:20	29:30	7990	163	3.6	-0.2	2.2	30	412	422	527	-9.0

Registrations of manoeuvres

nr.	manoeuvre	time	remarks
1	Phugoid motion		
2	Short Period oscillation		
3	Dutch roll		
4	Dutch roll with yaw damper		
5	Aperiodic roll		
6	Spiral mode		
7	Parabolic flight		
8			
9			
10			

[*] ET = Elapsed Time

6.3. EXERCISE 2: LINEAR SIMULATIONS OF THE AIRCRAFT EIGENMOTIONS

Goal of the exercise:

- To investigate the recorded motion of the Cessna Citation laboratory aircraft during the flight test using Matlab.
- To implement the linear equations of motion in Matlab, for symmetric as well as for asymmetric motion of the Cessna Citation laboratory aircraft.
- To investigate the simulated motion of the Cessna Citation laboratory aircraft, especially the eigenmotions.
- To compare the simulated and the recorded time responses of the Cessna Citation laboratory aircraft

See also sections 2.2, 4.3 and 4.4 of the ae3-303P Flight Dynamics and Simulation Manual Part II.

Investigation of the recorded motions

All five eigenmotions of the Cessna Citation aircraft were recorded during the flight. In order to review these motions they should be plotted using Matlab. First identify the required data files using the times from the post flight data sheet. Consider renaming the data files, in order to find them more easily in the future. You can load the datafiles into the Matlab workspace using the `load` command. Extract the various columns of this matrix into vectors representing the recorded parameters. These vectors can be plotted as a function of time.

For each eigenmotion find the most characteristic parameters to review that motion and make plots. Explain what you see. Sometimes you will have to pre-condition the data, because we are not interested in the recovery manoeuvre after a demonstration of an eigenmotion. Especially for the aperiodic roll and the short period pitch motion this might be the case.

Aircraft linear equations of motion

The linear equations of motion derived in the course Flight Dynamics are based on a number of assumptions:

- the aircraft is represented as a single rigid body
- the mass of the aircraft does not change

The equations of motion are linearised, allowing only small deviations around an equilibrium state of the aircraft. The symmetric equations of motion as derived in "Flight Dynamics" are:

$$\begin{bmatrix} C_{X_u}-2\mu_c D_c & C_{X_\alpha} & C_{Z_0} & C_{X_q} \\ C_{Z_u} & C_{Z_\alpha}+\left(C_{Z_{\dot\alpha}}-2\mu_c\right)D_c & -C_{X_0} & C_{Z_q}+2\mu_c \\ 0 & 0 & -D_c & 1 \\ C_{m_u} & C_{m_\alpha}+C_{m_{\dot\alpha}}D_c & 0 & C_{m_q}-2\mu_c K_{yy} D_c \end{bmatrix} \cdot \begin{bmatrix} \hat{u} \\ \alpha \\ \theta \\ \frac{q\bar{c}}{V} \end{bmatrix} = \begin{bmatrix} -C_{X_\delta} \\ -C_{Z_\delta} \\ 0 \\ -C_{m_\delta} \end{bmatrix} \cdot \delta_e$$

In this equation D_c is a differential operator, $D_c = \frac{\bar{c}}{V}\frac{d}{dt}$ and $\hat{u}$ is the dimensionless speed, $\hat{u} = \frac{u}{V}$ with V the true airspeed of the aircraft of the stationary flight condition. The equation can be rewritten using the normal angular pitch rate q, instead of the dimensionless version $\frac{q\bar{c}}{V}$.

Implementing this substitution in the above equation yields the following equation:

$$C_1\dot{\overline{x}} + C_2\overline{x} + C_3\overline{u} = \overline{0}$$

Here $\overline{u}$ is the input vector and $\overline{x}$ is the state vector;

$$\overline{u} = \left[\delta_e\right]$$
$$\overline{x} = \left[\hat{u} \quad \alpha \quad \theta \quad q\right]^T$$

Derive the matrices *C₁*, *C₂* and *C₃* yourself. Rewrite the equation very carefully, because any mistake, however small, will lead to faulty results in the simulations.

The form of the equation we are aiming at is the *state space representation* of the system:

$$\dot{\overline{x}} = A\overline{x} + B\overline{u}$$
$$\overline{y} = C\overline{x} + D\overline{u}$$

With $\overline{x}$ the state vector, $\overline{u}$ the input vector and $\overline{y}$ the output vector. The system matrices A and B can be determined from:

$$A = -C_1^{-1}C_2$$
$$B = -C_1^{-1}C_3$$

In a similar manner the state-space equations can be derived for the asymmetric motions of the aircraft. Here the differential operator D_b is equal to $D_b = \frac{b}{V}\frac{d}{dt}$, with b equal to the span of the aircraft. The starting point for the derivation is the linearised equation describing the asymmetric motions of the aircraft:

$$\begin{bmatrix} C_{Y_\beta} + \left(C_{Y_{\dot\beta}} - 2\mu_b\right)D_b & C_L & C_{Y_p} & C_{Y_r} - 4\mu_b \\ 0 & -\frac{1}{2}D_b & 1 & 0 \\ C_{\ell_\beta} & 0 & C_{\ell_p} - 4\mu_b K_{xx} D_b & C_{\ell_r} - 4\mu_b K_{xz} D_b \\ C_{n_\beta} + C_{n_{\dot\beta}} D_b & 0 & C_{n_p} - 4\mu_b K_{xz} D_b & C_{n_r} - 4\mu_b K_{zz} D_b \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \varphi \\ \frac{pb}{2V} \\ \frac{rb}{2V} \end{bmatrix} =$$
$$= \begin{bmatrix} -C_{y_{\delta_a}} \\ 0 \\ -C_{\ell_{\delta_a}} \\ -C_{n_{\delta_a}} \end{bmatrix} \cdot \delta_a + \begin{bmatrix} -C_{y_{\delta_r}} \\ 0 \\ -C_{\ell_{\delta_r}} \\ -C_{n_{\delta_r}} \end{bmatrix} \cdot \delta_r$$

The equation can be rewritten using the normal angular roll and yaw rate *p* and *r*, instead of the dimensionless versions $\frac{pb}{2V}$ and $\frac{rb}{2V}$. Again, derive matrices C_1, C_2 and C_3 and implement in Matlab the state-space representation:

$$\dot{\overline{x}} = A\overline{x} + B\overline{u}$$
$$\overline{y} = C\overline{x} + D\overline{u}$$

There are a few ways to obtain simpler equations of motion, among others:

Motion described	**assumption**
short period pitching oscillation	$\hat{u}=0$
short period pitching oscillation	$\hat{u}=0$ and $\gamma=\theta-\alpha=0$
phugoid	$\dot{q}=0$ and $\alpha=0$
phugoid	$\dot{q}=0$ and $\dot{\alpha}=0$
aperiodic roll motion	$\beta=0$ and $r=0$
dutch roll motion	$\varphi=0$ and $p=0$

Compare the responses with the full equations of motion for the phugoid motion and another motion of your choice.

Simulated motion of the Cessna Citation II laboratory aircraft

After you have implemented the symmetric and asymmetric linearised equations of motion in Matlab, you can use them to simulate the eigenmotions of the Cessna Citation II aircraft. The required stability derivatives can be found in the ae3-303P directory on the computer network in the file *citlin.m*.

We can solve the equations of motion using Matlab. A very useful command is `lsim`. With this command it is possible to determine the motion of the aircraft following a specified initial condition or a specified input signal.

In the case of the short period pitch motion, a step input on the elevator was given. Of course, this input signal is not an exact mathematical step, however the actual input signal was recorded and we can use exactly the same input signal for the simulation. In the simulations you can replay this situation and compare the recorded motion with the simulated responses.

APPENDIX A: HOW TO USE THE MATLAB NOTEBOOK?

The Matlab Notebook offers the possibility to execute Matlab commands from within a Word document. After executing the command the output appears in the same document. It is also possible to give plot commands.

The document "Intromatlab.doc" in the directory ae3-303P on the computer network is written in the form of a Notebook document. It contains an introduction in the basics of Matlab. You can drag the document to your own workdirectory and open it from there. The document may be saved in this directory after you executed the Matlab commands.

Below you will find some instructions how to work with the Matlab Notebook.

Matlab commands must be entered using the input font. This font may be selected by changing text type in the upper left corner of the Word window from Normal to the **`input font`**. All Matlab commands are thus displayed in green colour.

To execute Matlab commands place the cursor on the command and press <Ctrl+Enter>. The output will appear in the document in the blue `output font`.

Of course Matlab commands can also be given directly in the Matlab command window. To switch from Word to Matlab, press <Alt + m>.

Each time a Notebook document is opened in Word, Matlab will be started up automatically. Since Matlab is installed on the server and not locally, the start-up of Matlab is rather slow.

In the toolbar in the Microsoft Word window the menu "Notebook" is added. From this menu it is possible to group a number of selected commands. Each command is called a *cell*. The "Group Cells" option allows the subsequent execution of a number of cells by pressing <Ctrl + Enter> only once. Cell markers indicate which cells are grouped together. The cell markers may be made invisible by selecting "Hide Cell Markers".

Cells may be ungrouped by selecting the commands in the cell and choosing "Ungroup Cells" from the "Notebook" menu.

To work with Matlab Notebook you should be aware of a few bugs in the program.

a. Sometimes the minus sign is not recognised properly. If you suspect this is the cause of an unexpected error message, delete the minus sign in the Matlab command and type it again. The problem should be solved.

b. Sometimes plots appear distorted in the document. If this happens, switch to Matlab by pressing <Alt + m> and type "figure(1)" in the Matlab command window. A Matlab window will be opened containing the plot. You can view the plot in Matlab and delete the plot in the Word document.

If you like to use Matlab Notebook for making exercises or writing the flight test report later on, you can use the "Empty M-book.doc" file in the ae3-303P directory.

APPENDIX B: WHAT IS NICE TO KNOW ABOUT MATLAB?

- The following commands and key combinations may be of help:

`quit`	exit Matlab
<Ctrl + q>	also terminates Matlab
<Ctrl + c>	ends a Matlab run
<Alt + m>	switches you from a Notebook document in Word to Matlab
`path`	displays the currently defined paths to different working directories
`dir`	displays the filenames of all files in the current working directory
`!`	opens a DOS window

- To export Matlab generated plots to a text document there are a few possibilities:

1. Use copy/paste to create a copy of the figure in the document. The figure can be resized to fit in the document when necessary. This method consumes a lot of computer memory, so when your document contains many pictures your text processor may become very slow.
2. Use the Matlab command `print` to save your m-file in a file format of your choice, for example an encapsulated postscript file (EPS-file). The following format may be used:

```
print drive:\path\filename -f1 –deps -tiff -r300
```

`-f1`	is a figure handle referring to the window containing the figure
`–deps`	indicates that the figure must be saved as an "eps" file in the given directory
`–tiff`	will create a tiff preview, which will show up on screen when `filename` is inserted as a picture in a Word document
`-r300`	will save `figure(1)` at a 300 dpi (dots per inch) resolution

When the document is printed with a postscript printer the figure will be printed with 300 dpi resolution.

- Be careful when working from a floppy disk when you have saved the file you are working on to hard disk as well. When you run the file, the order of the directories in your Matlab path determines whether the file is run from floppy or from harddisk. If you are not aware of this, things may get frustrating….

- Matlab offers various help-features. For every command typing `help <command>` will give a short description of how to use this specific command. Furthermore, there is a help menu available from the toolbar in the upper command window. It gives you an overview of available Matlab functions.