BASIC
fracture mechanics

RNL Smith

BASIC fracture mechanics

including an introduction to fatigue

R N L Smith, BSc, MSc, PhD, FIMA

Senior Lecturer
Applied and Computational Mathematics Group
RMCS (Cranfield)
Shrivenham
Swindon
Wilts

BUTTERWORTH
HEINEMANN

To
Barbara and Les

Butterworth-Heinemann Ltd
Halley Court, Jordan Hill, Oxford OX2 8EJ

PART OF REED INTERNATIONAL BOOKS

OXFORD LONDON BOSTON MUNICH
NEW DELHI SINGAPORE SYDNEY
TOKYO TORONTO WELLINGTON

First published 1991

British Library Cataloguing in Publication Data
Smith, R.N.L.
BASIC fracture mechanics.
I. Title
620.100285

ISBN 0 7506 1489 7

Library of Congress Cataloging in Publication Data
Smith, R. N. L.
Basic fracture mechanics : including an introduction to fatigue / R.N.L. Smith.
p. cm.
Includes index.
ISBN 0-7506-1489-7 : $9.95
1. Fracture mechanics—Data processing. 2. Materials—Fatigue—Data processing. I. Title.
TA409.S63 1991
620.1'126—dc20 91-7417
CIP

Printed and bound in Great Britain by
Biddles Ltd, Guildford and King's Lynn

Preface

Fracture mechanics is becoming increasingly important to engineering design as designers search for ways of producing ever lighter structures. Lightweight designs are clearly crucial in the aerospace industry but are becoming of much wider significance in the pursuit of greater efficiency in almost all branches of engineering. Unfortunately the high strength materials required are prone to develop cracks which may grow slowly under quite modest stresses until there is a sudden catastrophic failure.

The aim of this book is to introduce the reader to fracture and fatigue and to provide a series of programs which implement those methods which are both useful and simple. Many of the methods of fracture mechanics which are, in principle, quite simple, may in fact be obscured by the often tedious or complex hand calculations required. The topic is well suited to a computer based approach since repetitive calculations can be done very quickly and simply with just a few lines of BASIC. The reader has at the simplest level a series of 'black boxes' which allows the study of say the effects of crack length on failure stress, without an intimate understanding of the manipulation of algebraic formulae. Of course, all the formulae used in the programs are explained or derived in the text for more thorough readers.

This book uses programs written in BASIC because it is probably the most widely available and easily understood computer language. Almost any computer can be used ranging from large 'Mainframe' machines to the personal computer and even some 'pocket calculators'. BASIC is probably the most straight-forward computer language to learn and students can be operating 'hands on' writing very simple programs in their first study period. No special compiling, linking or editing commands need to be learnt before starting on the language itself, although compiled versions may be available on some machines.

One of the drawbacks of BASIC lies in its poor subroutine facilities in standard form (some improved versions are available) but the programs given here would be virtually of the same form in many

languages. This is because the programs are written not to demonstrate programming skill or for computational efficiency, but in order to calculate in a way that relates simply to the text. An experienced programmer could translate the programs into say, FORTRAN while reading the BASIC, and may well add more comments and spacing since these are minimised here.

Chapter 1 is, as is usual with this series, a brief review of the BASIC language, while chapter 2 revises some useful formulae in stress analysis and continues to discuss stress concentrations. This leads to the idea of infinite stress at the crack tip, the stress intensity factor and the energy required for crack growth. Chapter 3 reviews some methods of obtaining stress intensity factors including the use of results obtained using finite or boundary elements although the actual FEM and BEM methods are not given. Having thus far assumed purely linear elasticity, in chapter 4 we outline some models of plastic behaviour in the highly-stressed crack tip region which leads naturally to the discussion of fatigue growth in chapter 5. Also in chapter 5 we introduce the idea of second-order fracture parameters and consider their effects on the initiation and direction of crack growth. Chapter 6 introduces the ideas of fracture-conscious design and inspection which aims to avoid cracks growing to near-failure lengths. We discuss the problems of residual stresses, crack growth from notches and experimentally derived fatigue results. Failure assessment diagrams which aim to incorporate fracture mechanics and plastic collapse are briefly illustrated. Finally, we discuss the use of numerical results for three-dimensional fracture problems in the context of the elliptical crack.

I gratefully acknowledgement the help of a number of people who have read and criticised part or all of the manuscript. These include Mike Iremonger and Peter Smith, the series editors, Mike Edwards of Materials group RMCS and David Rooke of the Royal Aerospace Establishment at Farnborough. I must also thank my wife Lynne and son Ben for their forbearance !

R.N.L.S.

Contents

Chapter 1

Introduction to BASIC

1.1 The BASIC approach

The programs in this book are written in the BASIC programming language. BASIC, an acronym for Beginner's All-purpose Symbolic Instruction Code, was developed at Dartmouth College, USA as an easy-to-learn, general-purpose language. Originally intended for use on timesharing computer systems, it has gained widespread popularity as the main language associated with microcomputers. Not only is the language easy to learn but it is also particularly easy to use. Without complication a program can be written, typed in at the computer, run and corrected and run again if any errors are present. The main disadvantages of simple BASIC relate to its lack of structure (see Section 1.4) but this is not an important consideration for short programs such as those in the following chapters.

This book aims to help in the learning of BASIC by applying it to a relevant engineering subject. This aim can be met by the reader studying the examples, possibly copying them and then trying some of the problems. Although this book does not specifically teach the grammar of BASIC, a short description of the simple BASIC used is given in the next section.

1.2 The elements of BASIC

1.2.1 Mathematical expressions

One of the main objects of the example programs in this book is to evaluate the equations that arise in fracture mechanics. These equations contain numerical constants, variables (e.g. x) and functions (e.g. sine(x)). All numbers are treated identically whether they are integer (e.g. 36) or real (e.g. 36.1). An exponential form is used to

represent large or small numbers (3.61×10^6 is written $3.61E6$). Numeric variables are represented by a letter or a letter followed by a digit (e.g. E or E1). On many computers π is directly available to the user either as PI or as a π key. For generality π is always computed within the program as $4tan^{-1}(1.0)$ in this book. An operation, such as square root, can be done using an in-built function (SQR(X)). The argument in brackets (X) can be a number, a variable or a mathematical expression. For trigonometric functions (SIN(X), COS (X), etc.) the argument is interpreted as being measured in radians. Other functions include a natural logarithm and its exponential (LOG and EXP respectively), ABS which selects the absolute value of an argument and INT which selects the integer part of an argument.

Mathematical equations also contain operators such as plus and minus, etc. These operations have a hierarchy in that some are performed by the computer before others. In descending order of hierarchy the operators are

```
to the power of (^)
multiply (*) and divide (/)
add (+) and subtract (-)
```

Thus, for example, multiplication is done before addition. The computer works from left to right if the operators have the same hierarchy. Brackets can be used to override any of these operations. Hence (a+b)/3c becomes (A + B)/(3*C) or (A + B)/3/C.

1.2.2 Program structure and assignment

A BASIC program is a sequence of statements which define a procedure for the computer to follow. As it follows this procedure the computer allocates values to each of the variables. The values of some of these variables may be specified by data that is input to the program. Others are generated in the program using, for instance, the assignment statement. This has the form

line number [LET] variable = mathematical expression

where the word LET is usually optional and therefore omitted. For example one of the roots of a quadratic equation.

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

may be obtained from a statement such as

```
100 X1 = (-B + SQR(B^2-4*A*C))/(2*A)
```

It is important to realise that an assignment statement is not itself an equation. It is an instruction to give the variable on the left-hand side the numeric value of the expression on the right-hand side. Thus it is possible to have a statement

```
50 X = X+1
```

which increases the value of X by 1.

Each variable can have only one value at any time unless it is subscripted (see section 1.2.7).

Note that all BASIC statements (i.e. all the program lines) are numbered. This defines the order in which they are executed.

1.2.3 Input

For 'interactive conversational' programs the user specifies variables by 'run-time' input of their values. The statement has the form

line number INPUT variable [,variable 2,...]

e.g.

```
20 INPUT A, B, C
```

When the program is run the computer prints ? as it reaches this statement and waits for the user to type values for the variables, e.g.

```
? 5, 10, 15
```

which makes A = 5, B = 10 and C = 15 in the above example.

Most computers will also allow a string in quotes as part of the INPUT statement. This string is printed at run-time and can thus be used to clarify which parameter value is required. The statement

```
20 INPUT "WHAT ARE A,B,C";A, B, C
```

prints the query

```
WHAT ARE A,B,C ?
```

when the program runs and the values are then typed in as above.

There is an alternative form of data input which is useful if there are a lot of data or if the data are not to be changed by the user. For this type of data specification there is a statement of the form

line number READ variable 1 [,variable 2,...]

e.g.

```
20 READ A,B,C
```

with an associated statement (or number of statements) of the form

line number DATA number 1 [,number 2.,,,]

e.g.

```
1 DATA 5,10,15.6
```

or

```
1 DATA 5
2 DATA 10
3 DATA 15.6
```

DATA statements can be placed anywhere in a program - it is often convenient to place them at the beginning or the end so they can be easily changed.

When using built-in data it is sometimes necessary to read the data from their start more than once during a single program run. This is done using the statement

line number RESTORE

1.2.4 Output

Output of data and the results of calculations, etc. is done using a statement of the form

line number PRINT list

This list may contain variables or expressions, e.g.

```
200 PRINT A,B,C,A*B/C
```

text enclosed in quotes, e.g.

```
10 PRINT "INPUT A,B,C IN MM";
```

or mixed text and variables, e.g.

```
300 PRINT "STRESS IS";S;"N/MM^2"
```

The items in the list are separated by commas or semi-colons. Commas give tabulation in columns, each about 15 spaces wide. A semi-colon suppresses this spacing and if it is placed at the end of a list it suppresses the line feed. If the list is left unfilled a blank line is printed.

PRINT statements may also be used in association with both 'run-time' input (to indicate what input is required) and READ/DATA statements (because otherwise the program user has no record of the data).

1.2.5 Conditional statements

It is often necessary to enable a program to take some action, if, and only if, some condition is fulfilled. This is done with a statement of the form

line number IF expression 1 (operator) expression 2 THEN line number

where the possible conditional operators are

```
=  equals
<> not equal to
<  less than
<= less than or equal to
>  greater than
>= greater or equal to
```

For example a program could contain the following statements if it is to stop when a zero value of A is input, i.e.

```
20 INPUT A
30 IF A <> 0 THEN 50
40 STOP
50 ...
```

Note the statement

line number STOP

which stops the run of a program.

1.2.6 Loops

There are several means by which a program can repeat some of its procedure; the self-repeating sequence of program statement is called a loop. The simplest such statement is

line number GO TO line number

This can be used, for instance, with the above conditional statement example so that the program continues to request values of A until the user inputs zero.

The most common means of performing loops is with a starting statement of the form

line number FOR variable = expression 1 TO expression 2 [STEP expression 3]

where the STEP is assumed to be unity if omitted. The finish of the loop is signified by a statement

line number NEXT variable

where the same variable is used in both FOR and NEXT statements. Its value should not be changed in the intervening lines.

A loop is used if, for example, N sets of data have to be READ and their reciprocals printed, e.g.

```
10 READ N
20 PRINT "NUMBER","RECIPROCAL"
30 FOR I=1 TO N
40 READ A
50 PRINT A,1/A
60 NEXT I
```

Loops can also be used to generate data. Consider the simple temperature conversion program

```
10 PRINT "CENTIGRADE","FAHRENHEIT"
20 FOR C = 0 TO 100 STEP 5
30 PRINT C,9*C/5+32
40 NEXT C
```

1.2.7 Subscripted variables

It is sometimes very beneficial to allow a single variable to have a number of different values during a single program run. For instance, if a program contains data for several materials it is convenient for their densities to be called R(1), R(2), R(3), etc. instead of R1, R2, R3, etc. It is then possible for a single statement to perform calculations for all the materials, e.g.

```
50 FOR I = 1 TO N
60 M(I) = V*R(I)
70 NEXT I
```

which determines the mass M(I) for each of the N materials from the volume (V) of the body.

A non-subscripted variable has a single value associated with it and if a subscript variable is used it is necessary to provide space for all the values. This is done with a dimensioning statement of the form.

line number DIM variable 1 (integer 1) [,variable 2 (integer 2), ...]

e.g.

```
20 DIM R(50),M(50)
```

which allows up to 50 values of R and M. The DIM statement must occur before the subscripted variables are first used.

On some computers it is possible to use a dimension statement of a different form, e.g.

```
20 DIM R(N),M(N)
```

where the value of N has been previously defined. This form, when available, has the advantage of not wasting space.

1.2.8 Subroutines

Sometimes a sequence of statements needs to be accessed more than once in the same program. Instead of merely repeating these statements it is better to put them in a subroutine. The program then contains statements of the form

line number GOSUB line number

When the program reaches this statement it branches (i.e. transfers control) to the second line number. The sequence of statements starting with this second line number ends with a statement

line number RETURN

and the program returns control to the statement immediately after the GOSUB call.

Subroutines can be placed anywhere in the program but it is usually convenient to position them at the end, separate from the main program statements.

Another reason for using a subroutine occurs when a procedure is written which is required in more than one program. It is often desirable to use less common variable names (e.g. X9 instead of X) in such subroutines. This minimises the possibility of the same variable name being used with a different meaning in separate parts of a program.

1.2.9 Other statements

(1) Explanatory remarks or headings which are not to be output can be inserted into a program using

line number REM comment

Any statement beginning with the word REM is ignored by the computer. On some computers it is possible to include remarks on the same line as other statements.

(2) Non-numeric data (e.g. words) can be handled by string variables. A string is a series of characters within quotes, e.g. 'STRESS' and a string variable is a letter followed by a $ sign, e.g. S$. They are particularly valuable when printed headings need to be varied.

(3) Multiple branching can be done with statements of the form

line number ON expression THEN line number 1 [,line number 2,...]

and

line number ON expression GOSUB line number 1 [, line number 2,...]

When a program reaches one of these statements it branches to line number 1 if the integer value of the expression is 1, to line number 2 if the expression is 2, and so on. An error message is printed if the expression gives a value less than 1 or greater than the number of referenced line numbers.

(4) Functions (other than those built into the language such as SIN(X)) can be created as defined functions using a DEF statement. For example

```
10 DEF FNA(X) = X^3 + X^2 + X + 1
```

defines a cubic function which can be recalled later in the program as FNA (variable) where the value of this variable is substituted for X. A defined function is useful if an algebraic expression is to be evaluated several times in a program.

1.3 Checking Programs

Most computers give a clear indication if there are grammatical (syntax) errors in a BASIC program. Program statements can be modified by retyping them correctly or by using special editing procedures. The majority of syntax errors are easy to locate but if a variable has been used with two (or more) different meanings in separate parts of the program some mystifying errors can result.

It is not sufficient for the program to be just grammatically correct. It must give the correct answers. A program should therefore be checked either by using data which give a known solution or by hand calculation. If the program is to be used with a wide range of data or by users other than the program writer, it is necessary to check that all parts of it function. It is also important to ensure that the program does not give incorrect but plausible answers when 'nonsense' data are input. It is quite difficult to make programs completely 'userproof' and they become lengthy in so doing. The programs in this