MECHANICAL JOINTS IN COMPOSITE LAMINATES - A COMPLEX STRESS FUNCTION BASED PIN LOADED HOLE APPROXIMATION

Peter Mark Bidgood

A dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg 1989

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg, South Africa. No claim is made as to the originality of any of the theory presented in this dissertation, although elements of originality may be found in methedology and approach.

This work has not been submitted before for any degree or examination by this author in any other University.

T.H. Badgood (Signature of Candidate)

16 th day of MAY 19 90

ABSTRACT

Classical laminate theory is a well known theory for obtaining the properties and stress distribution in a layered orthotropic laminate. This theory, however, only applies to laminates of infinite size,where disturbances in the stress field as may be caused by free edges,holes or cut-outs are not present.

Methods of calculating the stress distributions and behaviour of holes and pin-loaded holes in a composite laminate have been investigated.

This dissertation presents a computer program written in the "C" programming language as implemented on a personal computer. The theory is based upon the original work of Lekhnitski (1947) [Ref], as further developed and presented by De Jong. The theory is briefly presented. The method is an alternative to the more expensive method of finite element modelling and is derived from the solution of the governing differential equation by means of complex stress functions.

The program, (BHOLES), is a data generating module which generates the stress field in the vicinity of a hole or pin-loaded hole in a laminate specimen of arbitrary width to hole diameter ratio.

Alternative methods of presenting and analysing the generated data have been investigated but no direct comparison is made with experimental results.

The accuracy of the generated data is verified by several methods including correlation with data generated by an independently developed program. Indirect reference to test results obtained by De Jong [2] is used to indicate the effectiveness of the model.

ACKNOWLEDGEMENTS

The work completed in this dissertation would not have been possible without the financial support of Kentron (Pty) Ltd.

The cooperation of the University of the Witwatersrand laboratory staff in the manufacture of a prototype test jig was of fundamental importance as was the assistance given by Mr C Diamantakos in the manufacture of pre-impregnated composite panels and the supply of additional information.

A special word of thanks must be extended to Drs R.J. Fritz for his supervision and role in liaison between the author and Mr T.H. de Jong of Delft University of Technology during the initial stages of this project.

The work contained in this study is based largely upon the work of Mr T.H. de Jong and a word of gratitude must be extended to his willingness to supply much of the theory and notes which have been so well developed and presented by himself.

Finally a word of thanks to Dr R.J. Huston for his input in the form of discussions and papers on empirical methods and approaches and Miss Joanne Kirsten and Mrs Grace Proudfoot for the typing of the manuscript, all of the Division of Aeronautical and Systems Technology (Aerotek) of the Council for Scientific and Industrial Research.

LIST OF FIGURES

LIST OF SYMBOLS

C_{ij}	terms in the laminate compliance matrix.
S_{ij}	used interchangeably with C_{ij} in some analyses.
S_{ij_ϕ}	transformed laminate compliance terms.
σ_i	normal stresses.
ϵ_i	normal strains.
σ_{ij}	shear stress (same as τ_{ij}).
ϵ_{ij}	shear strain (same as γ_{ij}).
U	the Airy stress function ($U(x,y)$).
r	material angularity (defined in the theory).
a	material directionality (defined in the theory).
z	complex variable ($z = x + sy$).
x	coordinate in the solution plane.
y	coordinate in the solution plane.
s	complex term in $z = x + sy$.
$F_k(z_k)$	solution function for $U(x,y)$ in the complex plane.
$\phi_k(z_k)$	F_k differentiated with respect to z_k.
$\phi'_k(z_k)$	$\phi_k(z_k)$ differentiated with respect to z_k.
u	displacement in direction of x-coordinate.
v	displacement in direction of y-coordinate.
P_x	boundary load in direction of x-coordinate.
P_y	boundary load in direction of y-coordinate.
P_{xy}	shear boundary load.
ζ_k	substitution variable.
C_i	constants.
R_y	load resultant on pin in y-coordinate direction.
R_x	Load resultant on pin in x-coordinate direction.
P_r	radial force on hole edge due to pin (frictionless).
a_n	terms in sine series representing pressure distribution due to pin.
A_k	constants resulting from pin load.

NOMENCLATURE

1 General

i) The complex terms s_k and $s_{k\phi}$ are distinguishable from the material compliances S_{ij} and S_{ij_ϕ} by the single and double suffexes.

ii) Material compliances are initially referred to as C_{ij} at the end of Chapter 1 and in Chapter 2.1. The deeper analysis given in Chapters 2.2 and 2.3 uses the S_{ij} variable for material compliances.

iii) The terms v_k and u_k referred to in equations (2.21) are distingnisable from the displacements v and u by the suffex. Their expansion is given in equation (2.8).

2 References

References with an asterisk refer to references referred to by the author referenced.

CONTENTS LIST

CONTENTS LIST (continued)

CHAPTER 1
INTRODUCTION

1.1 Work Justification

Generally, two methods of joining two components are used in the fabrication of composite structures, viz. adhesive bonding, where the load transfer is predominantly by shear in the adhesive layer, and mechanical joints, where the load transfer is provided by fasteners such as rivets, bolts and pins.

Earlier investigations into the efficiency of a pin loaded hole in carbon/epoxy tab specimens [4] indicated that the efficiencies that can be expected from mechanical joining of such materials is lower than is typically found in joints in isotropic metals. This is especially true for the more ductile materials, such as many forms of aluminium, where deformation of the material can reduce the stress concentrations present in close proximity to the bolt hole(s). The implication of this is that the attractive high specific strength and stiffness, as well as generally good fatigue resistance of fibrous composites, are offset by their intolerance of stress raisers such as may be created by holes or cutouts.

Although the drilling of holes in a laminate may often be avoided by the selection of a fabrication process involving integrally cured components or by adhesive bonding techniques, this is not always possible nor desirable. Situations are often found where mechanical joints are not only the best solution to a particular structural integration process, but may often be the only method of component integration. This is the case where, for example, structural disassembly is required, where access panels are required, when the magnitude of the loads to be transferred are too large to allow even the best adhesives to be used, when fail-safe failure modes are mandatory, in combination with adhesive bonding to eliminate the need for costly bonding jigs and so forth.

The efficiency of a mechanical joint in a fibrous composite is strongly influenced by the laminate lay up, and it may therefore be concluded that such joints can be optimised [5], and should at the very least be understood, analysed and tested before implementation in any demanding structural application.

That the design of a mechanical joint in a composite laminate requires special attention is best put by Poon [6]:

> "... one of the more challenging aspects of mechanically fastened joints is that the well-established design procedures for metal joints, that are based on years of experience with isotropic and homogeneous materials, have yet to be changed in order to accommodate the anisotropic and non-homogeneous properties of composite materials".

1.2 Literature Survey

Papers on the subject of holes and mechanical joints in composites date back as far as the late sixties. The problems involved with this particular aspect are however numerous, and the many existing papers on the subject can therefore be considered to be limited.

An extensive literature survey is not offered in this dissertation. Instead some of the aspects that have been investigated by various researchers are briefly mentioned and the reader is instead referred to an earlier and more comprehensive survey done by the author [1]. This survey was intended to contain a more in depth investigation, with the intention that the reader may obtain an acceptable amount of background information to the general problem. Empirical methods as used by Hart-Smith [7], Collings [8], Matthews et al [9] and Opplinger, et al [10] are presented in a fair amount of detail in an attempt to distill a more universal methedology and to investigate general parametrics.

The more detailed aspects such as friction between mating surfaces and loss of bolt torque due to the visco elastic properties of resin-based composites are only briefly mentioned. Sandifer [11] found that there was no significant effect on the fatigue life of graphite/epoxy material when fretted against aluminium, titanium or graphite/epoxy of the same type. The effect of friction in a fraying plate surface can be included in the stress analysis if the clamping force is known. A typical idealisation of a bolted joint used to determine the contact pressure between plates is illustrated by Mathews, et al [12].

Poon [6] gives a reasonably detailed discussion on the basic methodologies used in fatigue life prediction. Also discussed in this report are effects such as galling (also discussed by Cole [6*],, and installation damage where remedies such as lubrication as used in the F18 and AV-813 are suggested.

Rosenfield [13] qualitatively examines the now well known graphite- aluminium corrosion problem by testing mechanically fastened joints fabricated from these materials with and without various forms of corrosion protection methods. An investigation into the corrosion characteristics of various metal fasteners in graphite/epoxy composites during exposure to a hostile environment such as salt spray and the protection offered by various protective coating systems is also presented in a paper from the Air Force Materials Laboratory, Ohio [14].

Static strength prediction using fracture mechanics principles is presented by Eisenmann [5], and this approach can be seen to be similar to the "characteristic dimension" approach used by Whitney and Nuismer [15].

.efining the failure load of a composite material joint is a problem in itself. Poon [6] notes that basing the definition on the maximum load that can be sustained by such joints is rather crude and inappropriate, since in this case failures occurred in some cases in bearing at a specimen width to hole diameter ratio of around 3, after which the sustained load continued to increase as the fibres piled up behind the pin. This observation was confirmed by the author [4] and by subsequent related undergraduate work [17]. The problem of failure load definition leads to the need to be able to predict the load at which first significant damage occurs such that specimens can be loaded to this predicted value whilst using acoustic emissions as a guide, and subsequent x-ray examination for verification of the extent of the damage. This method was successfully implemented by De Jong [2] where first significant damage loads occurred within twenty percent of those predicted. In the light of limitations of the theory used in the test cases and the limitations of the damage detection techniques, this is believed to be a significant step in the understanding of the behaviour of composite materials. The problems involved with failure theories should also not be forgotten.

Analytical techniques for predicting the stress field in the vicinity of a bolted joint include finite element techniques and complex stress function methods.

Various finite element codes have been used to calculate the stress field. Waszcak and Cruse [17] presented a two-dimensional finite element model as early as 1971 using an assumed cosine pressure distribution on the hole edge. This assumption has subsequently been shown to be incorrect. Chang et al [18*] investigated the same problem in a similar way and obtained improved correlations in failure strength predictions by applying the Yamada-Sun shear strength failure criterion in conjunction with a proposed failure hypothesis that predicts failure based on stresses at a characteristic distance away from the hole in order to minimise three dimensional effects. Argawal [19] used a NASTRAN code to determine the stress distribution around the fastener hole of a double shear bolt bearing specimen and Whitney-Nuismer average stress criterion for failure load.

Soni [20] used the same NASTRAN code and boundary conditions but adopted the Tsai-Wu tensor polynomial failure criterion. York, et al [21*] used the structural analysis package SAP V and the modified "point stress" failure criterion. Rowlands, et al [22] used a finite element model which included the effects of variations in friction,material properties, load distribution between bolts in series, end distance, bolt clearance and bolt spacing. An important advantage of this approach is the ability to take three dimensional effects into account. Three dimensional effects are of special importance when including bolt torque and edge effects. Rybicki and Schmueser [24] made one of the earliest attempts at analysing a curved boundary. Their model included effects of through the thickness stresses. Matthews, et al [23] describes the development of an element derived from a standard 20 noded isoparametric brick element. The effects of bolt tightning on through-the-thickness effects are discussed. The brick was incorporated as standard within the FINEL analysis package.

The method of complex functions has been applied by numerous researchers of which the most notable is De Jong [2]. Others such as Tung [25] have made useful contributions to this analytical technique. Tung presents a limiting procedure and a root modification scheme that give accurate results with ordinary machine precision for any plate material. This procedure has not been implemented in the analysis presented in this thesis, and consequently it can be expected that a solution cannot be obtained for the case when the characteristic roots are equal.

Besides the empirical, analytical and experimental aspects of the problem of bolted joints in composites, are the more practical aspects such as the effects of countersink angle (which may vary between 90 and 100 degrees), the use of solid versus hollow rivets, and rivet types, such as is discussed by Matthews, et al [26]. Some commercially available rivet systems are also discussed in the fuller review previously mentioned by the author. Practical aspects such as galvanic corrosion, galling, installation damage and pull through strength with reference to semi-tubular rivets, big foots, Cherry Buck rivets, stress-wave rivet systems, groove proportioned lock bolts, composite fasteners and self tapping screws are discussed by Cole, et al [27], and the determination of suitable safety factors for use when designing bolted joints in GRP are discussed by Matthews and Johnston [28].

In addition to the vast number of papers on the subject, numerous reviews are also available. Some of these are by Tseng-Hua Tsiang [29] and Goodwin and Mathews [30].

1.3 Objectives of Current Research

The objective of this research was to investigate the literature in order to extract enough information to be able to make a significant contribution to the design and analysis of advanced aircraft structures with particular reference to mechanical joints in composite materials. This has been achieved in the following manner:

(a) The accumulation and collation of as much information as possible relating to research done by other researchers on the subject.

(b) The extraction of information from relevant papers and the presentation of this information in a literature survey such that a reasonable understanding of most aspects and approaches currently used to design and analyse composite joints can be readily obtained by the reader. Since this phase was done whilst the author was working in an industry not yet possessing a well developed capability in this area, a suggested approach to the development of such expertise is presented.

(c) The development of an analytical model capable of generating the theoretical stress field in the vicinity of a pin loaded hole in order to make a tool available for deeper understanding of the mechanisms involved in load transfer. An attempt was made to develop a graphical representation of the relevant stresses in such a way as to enable the interpretation of results presented by other authors. A modest attempt is also made to predict the load at which first significant damage in the laminate occurs and its location by means of the concept of failure propensities based on the Tsai-Hill failure criterion.

1.4 Approach

Empirical methods have been investigated to a reasonable extent and results from such an approach can be very useful in the design environment, but the costs involved with developing the necessary infrastructure, the data collection process and the equipment itself are high. The various approaches which may be adopted for developing the necessary expertise in this area are discussed in Part 1, Literature survey. In summary it can be said that practical application is always the final test, that empirical data is necessary for rapid first order design procedures, but that the availability of an analytical model will offer the tool required for more in-depth understanding of the actual stress field and mechanisms of load transfer. In addition to this a detailed model may indicate directions required for optimised joint design and reveal trends or effects which cannot be obtained by other approaches.

Of the analytical methods used only two have been noted and considered to be worth pursuing. These are finite element models and complex stress function models. Of these two methods the finite element method at present has possibly been the most effective, since it is capable of including through-the-thickness effects such as may be ordinarily present or induced by bolt torque. The costs of building such a model can however be considered to be extremely high, due both to the high cost of the required software, and due to the time required to build and run a complex model which will have to include layering, anisotropy and contact or pressure modelling. The method of complex stress functions on the other hand can be implemented quite independently of any other specialised software, and as this thesis demonstrates, can be made effective on a personal computer resulting in greatly reduced costs.

The method is derived from the solution, (by means of analytic functions), of the differential equation obtained by substituting firstly the constitutive equations:

$$\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{16} \\ C_{21} & C_{22} & C_{26} \\ C_{61} & C_{62} & C_{66} \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} \tag{1.1}$$

and secondly the Airy stress function $U(x,y)$;

where $$\sigma_x = \frac{\partial^2 U}{\partial_y^2}; \quad \sigma_y = \frac{\partial^2 U}{\partial_x^2}; \quad \tau_{xy} = -\frac{\partial^2 U}{\partial_x \partial_y} \tag{1.2}$$

into the compatibility equation:

$$\frac{\partial^2 \epsilon_x}{\partial_y^2} + \frac{\partial^2 \epsilon_y}{\partial_x^2} = \frac{\partial^2 \gamma_{xy}}{\partial_x \partial_y} \tag{1.3}$$

The resulting 4th order differential equation is:

$$C_{22}\frac{\partial^4 U}{\partial_x^4} - 2C_{26}\frac{\partial^4 U}{\partial_x^3\,\partial_y} + (2C_{12} + C_{66})\frac{\partial^4 U}{\partial_x^2\,\partial_y^2} - 2C_{16}\frac{\partial^4 U}{\partial_x\,\partial_y^3} + C_{11}\frac{\partial^4 U}{\partial_y^4} = 0 \tag{1.4}$$

which can be simplified and solved using appropriate boundary conditions.

CHAPTER 2
THEORY

The solution of the differential equation is done by means of analytic functions. The solution involves much algebraic manipulation. A detailed analysis is avoided in this report for the sake of brevity. Instead, only the major steps in the derivation as developed by De Jong [3] are presented.

2.1 Complex Stress Functions

By taking the x-y coordinates as having an orientation with respect to the laminate such that these axes lie along the principal material axes, the terms C_{16} and C_{26} conveniently become zero. This results in a simplified form of equation (1.4):

$$\left[\frac{C_{22}}{C_{11}}\right] \frac{\partial^4 U}{\partial_x^4} + \left[\frac{2C_{12} + C_{66}}{C_{11}}\right] \frac{\partial^4 U}{\partial_x^2 \, \partial_y^2} + \frac{\partial^4 U}{\partial_y^4} = 0 \tag{2.1}$$

Now by defining two properties, directionality (r) and angularity (a),

where
$$r = \sqrt{\frac{C_{22}}{C_{11}}} = \sqrt{\frac{E_1}{E_2}} \tag{2.2}$$

$$a = \frac{C_{12} + \frac{C_{66}}{2}}{C_{11}} = \frac{E_1}{2G_{12}} - \mu_{12} \tag{2.3}$$

this equation becomes:

$$r^2 \frac{\partial^4 U}{\partial_x^4} + 2a \frac{\partial^4 U}{\partial_x^2 \, \partial_y^2} + \frac{\partial^4 U}{\partial_y^4} = 0 \tag{2.4}$$

It is of interest to note that by inserting isotropic values for r and a, this equation reduces to:

$$\Delta^2 U = 0$$

In order to solve this differential equation, $U(x,y)$ is assumed to have a solution function in the complex plane, i.e.

$$U(x,y) = f(x+sy) = F(z) \tag{2.5}$$

where $z = x+sy$
and s = complex number

It can be shown that:

$$U = F_1(x+s_1y) + \overline{F}_1(x+\bar{s}_1y) + F_2(x+s_2y) + \overline{F}_2(x+\bar{s}_2y) + C_1x + C_2y + C_3$$

and since the particular solution $U = C_1x + C_2y + C_3$ gives $\sigma_x = \sigma_y = \tau_{xy} = 0$, it can be ignored.

The resulting equation can be written as:

$$U = 2\mathrm{Re}\,[F_1(z_1) + F_2\,(z_2)]$$

and simplified to:

$$U = 2\mathrm{Re} \sum F_k(z_k) \qquad k = 1,2 \tag{2.6}$$

Letting $\frac{dF_1}{dz_1} = \phi_1(z_1)$ and $\frac{dF_2}{dz_2} = \phi_2(z_2)$, the stresses can be found by differentiation to be:

$$\sigma_x = 2\mathrm{Re} \sum s_k^2\, \phi_k'\,(z_k)$$

$$\sigma_y = 2\mathrm{Re} \sum \phi_k'\,(z_k)$$

$$\sigma_{xy} = -2\mathrm{Re} \sum s_k\, \phi_k'\,(z_k) \qquad k = 1,2 \tag{2.7}$$

where $\phi_1^{'} = \frac{d\phi_1}{dz_1}$ and $\phi_2^{'} = \frac{d\phi_2}{dz_2}$

and the displacements become:

$$u = 2\mathrm{Re}\sum u_k\, \phi_k + C_1 y + C_2$$

$$v = 2\mathrm{Re}\sum v_k\, \phi_k + C_1 x + C_4 \qquad (2.8)$$

where $u_k = C_{11}\, s_k^2 + C_{12}$ and $v_k = \frac{C_{22}}{s_k} + C_{12}\, s_k$

To solve the differential equation, a convention for the direction of integration along the boundaries is chosen as shown below. In this case it is defined positive to the left if facing the boundary from inside the region.

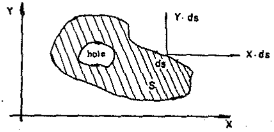


Figure 2.1 : Integration conventions.

When the loads on the boundary are known the following equations are derived:

$$2\mathrm{Re}\sum \phi_k\,(z_k) = -\int Y\,.\,ds + C$$

$$2\mathrm{Re}\sum s_k\, \phi_k\,(z_k) = \int X\,.\,ds + C \qquad k + 1,2 \qquad (2.9)$$

and when the displacements are used we have:

$$2\mathrm{Re} \sum u_k \, \phi_k \,(z_k) \;=\; u(s)$$

$$2\mathrm{Re} \sum v_k \, \phi_k \,(z_k) \;=\; v(s) \qquad\qquad k = 1,2 \qquad\qquad (2.10)$$

Thus, by applying either external loading or displacements or both along the boundary region, the functions ϕ_k can be obtained.

The functions ϕ_k' are represented by a Laurent series:

$$f(z) \;=\; \sum_{-\infty}^{\infty} n \, f_n \, z^n$$

without the presence of the positive terms. ϕ_k' can be written as:

$$\phi_k'(z_k) \;=\; g^{(k)} + A_k \, z_k^{-1} - \left[g_1^{(k)} \, z_k^{-2} + 2g_2^{(k)} \, z_k^{-3} + 3g_3^{(k)} \, z_k^{-4} \cdots \right]$$

and by integration:

$$\phi_k(z_k) \;=\; g^{(k)} \,.\, z_k + A_k \ln z_k + \sum_{n=1}^{\infty} g_n^{(k)} \, z_k^{-n} \; (+ \, \mathrm{Const}) \qquad\qquad (2.11)$$

It can then be reasoned that:

(a) The terms $g^{(k)} \, z_k$ represent the homogeneous stress field, and

(b) the $\sum_{n=1}^{\infty} g_n^{(k)} \, z_k^{-n}$ terms represent the effects of a hole, while

(c) the $A_k \ln z_k$ terms are related to any loading on the hole edge.

These terms can to some extent be separated during analysis. Also, it can be shown that this Laurent series converges for all values of $|z| \geq 1$, i.e. outside the unit circle in the complex plane.

2.2 Stresses Around Unloaded Holes

By considering an infinitely large plate with a hole of radius R = 1 and the centre of the axis system such that it coincides with the material principle axes, (as shown in Figure 2.2), the functions ϕ may be found.

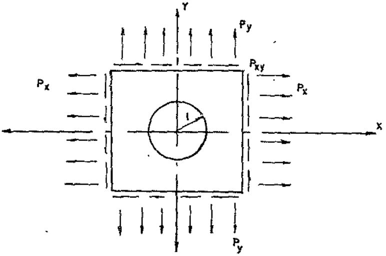


Figure 2.2 : Idealisation for a hole in an orthotropic plate.

The stress distribution may be found by evaluating the complex functions $\phi_k(z_k)$, with the general form:

$$\phi_k(z_k) = g^{(k)} \cdot z_k + \sum_{n=1}^{\infty} g_n^{(k)} \cdot z_k^{-n} \tag{2.12}$$

where the first term can easily be shown to represent the homogenous field by:

$$g^{(k)} = \frac{P_x - P_y s_\ell^2 - P_{xy}(s_k - s_\ell)}{2(s_k^2 - s_\ell^2)} \qquad \begin{matrix} k = 1,2 \\ \ell = 2,1 \end{matrix} \tag{2.13}$$

The perturbated stress field due to the presence of the hole may be evaluated using the

$$\sum_{n=1}^{\infty} g_n^{(k)} \cdot z_k^{-n}$$

term and this can then be superimposed onto the homogenous stress field. Thus from equation (2.9) and using the fact that at this particular boundary no external load has been applied, we have:

$$2\mathrm{Re}\left[\left[g^{(1)} \cdot z_1 + g^{(2)} \cdot z_2\right] + \sum_{n=1}^{\infty} g_n^{(1)} \cdot z_1^{-n} + \sum_{n=1}^{\infty} g_n^{(2)} \cdot z_2^{-n}\right] = 0$$

and

$$2\mathrm{Re}\left[\left[s_1 g^{(1)} \cdot z_1 + s_2 \cdot g^{(2)} \cdot z_2\right] + \sum_{n=1}^{\infty} s_1 \cdot g_n^{(1)} \cdot z_1^{-n} + \sum_{n=1}^{\infty} s_2 \cdot g_n^{(2)} \cdot z_2^{-n}\right] = 0 \tag{2.14}$$

or

$$2\mathrm{Re}\left[\sum_{n=1}^{\infty} g_n^{(1)} \cdot z_1^{-n} + \sum_{n=1}^{\infty} g_n^{(2)} \cdot z_2^{-n}\right] = -2\mathrm{Re}\left[g^{(1)} \cdot z_1 + g^{(2)} \cdot z_2\right]$$

and

$$2\mathrm{Re}\left[\sum_{n=1}^{\infty} s_1 \cdot g_n^{(1)} \cdot z_1^{-n} + \sum_{n=1}^{\infty} s_2 \cdot g_n^{(2)} \cdot z_2^{-n}\right] = -2\mathrm{Re}\left[s_1 g^{(1)} \cdot z_1 + s_2 g^{(2)} \cdot z_2\right] \tag{2.15}$$

Now by considering the stress field at infinity we can assume that the stress perturbations are zero and the external loading is as given in Figure 2.2, giving:

$$2\mathrm{Re}\left[g^{(1)} z_1 + g^{(2)} z_2\right] = -\int Y.ds = P_y.x - P_{xy}.y$$

and

$$2\mathrm{Re}\left[s_1 g^{(1)} z_1 + s_2 g^{(2)} z_2\right] = \int X.ds = P_x.y - P_{xy}.x \qquad (2.16)$$

which may be substituted into equation (2.15):

$$2\mathrm{Re}\left[\sum_{n=1}^{\infty} g_n^{(1)}.z_1^{-n} + \sum_{n=1}^{\infty} g_n^{(2)}.z_2^{-n}\right] = -P_y.x + P_{xy}.y$$

and

$$2\mathrm{Re}\left[\sum_{n=1}^{\infty} s_1 g_n^{(1)}.z_1^{-n} + \sum_{n=1}^{\infty} s_2 g_n^{(2)}.z_2^{-n}\right] = -P_x.y + P_{xy}.x \qquad (2.17)$$

In order to more easily solve the equations, new variables ζ_1 and ζ_2 are introduced where:

$$\zeta_k = \frac{s_k + \sqrt{s_k^2 - s_k^2 - 1}}{1 - is_k} \qquad (2.18)$$

and on the unit circle $\zeta_k = z$.

The resulting series after substitution becomes:

$$2\mathrm{Re}\left[\sum_{n=1}^{\infty} C_n^{(1)}.\zeta_1^{-n} + \sum_{n=1}^{\infty} C_n^{(2)}.\zeta_2^{-n}\right] = -P_y.y + P_{xy}.y$$

and

$$2\mathrm{Re}\left[\sum_{n=1}^{\infty} s_1 C_n^{(1)}.\zeta_1^{-n} + \sum_{n=1}^{\infty} s_2 C_n^{(2)}.\zeta_2^{-n}\right] = -P_x.y + P_{xy}.x \qquad (2.19)$$

It can, (by a fair amount of reasoning), be construed that

$$C_n^{(1)} = C_n^{(2)} = 0 \text{ for all } n \neq 1,$$

and then solving for C_1 and C_2 gives

$$\phi_k(z_k) = \frac{P_x - P_y s_\ell^2 - P_{xy}(s_k + s_\ell)}{2(s_k^2 - s_\ell^2)} \cdot z_k + \frac{-iP_x + s_\ell P_y + (1 - is_\ell)P_{xy}}{2(s_k - s_\ell)} \zeta_k^{-1}$$

$\phi_k(z_k)$ can be differentiated to give:

$$\phi_k'(z_k) = \frac{P_x - P_y s_\ell^2 - P_{xy}(s_k + s_\ell)}{2(s_k^2 - s_\ell^2)} + \frac{P_x + is_\ell P_y + (s_\ell + i)P_{xy}}{2(s_k - s_\ell)(s_k - i)} \left\{\frac{z_k}{z_k^1} - 1\right\} \quad (2.20)$$

which enables direct calculation of σ_x, σ_y and τ_{xy} as well as u and v. The solutions for the stresses become:

$$\sigma_x = P_x + 2\mathrm{Re} \sum \left[s_k^2 B_k \left\{\frac{z_k}{z_k^1} - 1\right\} \right]$$

$$\sigma_y = P_y + 2\mathrm{Re} \sum \left[B_k \left\{\frac{z_k}{z_k^1} - 1\right\} \right]$$

$$\tau_{xy} = P_{xy} - 2\mathrm{Re} \sum \left[s_k B_k \left\{\frac{z_k}{z_k^1} - 1\right\} \right]$$

where $$B_k = \frac{P_x + is_\ell P_y + (s_\ell + i)P_{xy}}{2(s_k - s_\ell)(s_k - i)}$$

and $$z_k^1 = \sqrt{z_k^2 - s_k^2 - 1}$$

2.3 Stresses Around Pin Loaded Holes

Forces applied to the edge of the hole are included in the analysis by means of the $A_k \ln \zeta_k$ term. Its presence in the formulation accounts for any loading that may be present on the hole edge. A_k may be solved from equations given in Ref. [3]. These equations are:

$$\sum_{k=1,2}^{\infty} \left\{ u_{k\phi} A_k - \overline{u_{k\phi}}\,\overline{A_k} \right\} = 0$$

$$\sum_{k=1,2}^{\infty} \left\{ v_{k\phi} A_k - \overline{v_{k\phi}}\,\overline{A_k} \right\} = 0$$

$$\sum_{k=1,2}^{\infty} \left\{ A_k - \overline{A_k} \right\} = \frac{R_y}{2\pi i}$$

$$\sum_{k=1,2}^{\infty} \left\{ s_{k\phi} A_k - \overline{s_{k\phi}}\,\overline{A_k} \right\} = \frac{-R_x}{2\pi i} \tag{2.21}$$

yielding

$$A_k = \left[R_y \left\{ s_{k\phi} \left[\overline{s_{\ell\phi}}\, s_{\ell\phi} + s_{\ell\phi}\, \overline{s_{k\phi}} + \overline{s_{\ell\phi}}\, \overline{s_{k\phi}} - \frac{s_{12\phi}}{s_{11\phi}} \right] - \frac{s_{26\phi}}{s_{11\phi}} \right\} \right.$$

$$\left. + R_x \left\{ s_{k\phi} \left[s_{\ell\phi} + \overline{s_{\ell\phi}} + \overline{s_{k\phi}} - \frac{s_{16\phi}}{s_{11\phi}} \right] - \frac{s_{26\phi}}{s_{11\phi}} \right\} \right]$$

$$\left\{ 2\pi i \left[s_{\ell\phi} - s_{k\phi} \right] \left[\overline{s_{\ell\phi}} - s_{k\phi} \right] \left[s_{k\phi} - \overline{s_{k\phi}} \right] \right\}$$

$$k = 1,2; \quad \ell = 2,1 \tag{2.22}$$

The original formulation of ϕ_k may be written as:

$$\phi_k(z_k) = A_k \ln \zeta_k + \phi_k^0(z_k) + B_k \qquad k = 1,2 \tag{2.23}$$

This may then be substituted into the boundary equations given in equation (2.9).

$$2\mathrm{Re} \sum_{k=1,2}^{\infty} \left\{ B_k + A_k \ln \sigma + \phi_k^0(z_k) \right\} = \int_0^s Y \cdot ds + K_1$$

and

$$2\mathrm{Re} \sum_{k=1,2}^{\infty} \left\{ s_{k\phi} B_k + s_{k\phi} A_k \ln \sigma + s_{k\phi} \phi_k^0(z_k) \right\} = -\int_0^s X \cdot ds + K_2 \tag{2.24}$$

(This is valid on the hole edge where $\zeta_1 = \zeta_2 = \sigma = 1$) and following from the third and fourth equations of (2.21):

$$2\mathrm{Re} \sum_{k=1,2}^{\infty} A_k \ln \sigma = \frac{R_Y \theta}{2\pi}$$

$$2\mathrm{Re} \sum_{k=1,2}^{\infty} s_{k\phi} A_k \ln \sigma = \frac{-R_X \theta}{2\pi} \tag{2.25}$$

Thus equations (2.24) become:

$$2\mathrm{Re} \sum_{k=1,2}^{\infty} \left\{ B_k + \phi_k^0(z_k) \right\} = \int_0^s Y \cdot ds + K_1 - \frac{R_Y \theta}{2\pi}$$

and

$$2\mathrm{Re} \sum_{k=1,2}^{\infty} \left\{ s_{k\phi} B_k + s_{k\phi} \phi_k^0(z_k) \right\} = -\int_0^s X \cdot ds + K_2 + \frac{R_X \theta}{2\pi} \tag{2.26}$$

Since no form for the pressure distribution is assumed beforehand, a general expression is needed which will accommodate any distribution that may result from the analysis. The chosen form is:

$$P_r = P_o \sum_{n=1,2,3}^{\infty} a_n \sin n\theta \qquad \text{for } 0 < \theta < \pi$$

$$P_r = 0 \qquad \text{for } \pi < \theta < 2\pi \tag{2.27}$$

or to have one expression which is valid on the whole circumference, (2.27) can be written as:

$$P_r = P_r \left\{ \frac{1}{2} + \frac{2}{\pi} \sum_{m=1,3}^{\infty} \frac{\sin\ m\theta}{m} \right\} = \begin{cases} P_r \text{ for } 0 < \theta < \pi \\ 0 \ \text{ for } \pi < \theta < 2\pi \end{cases}$$

resulting in:

$$P_r = P_o \left\{ \frac{1}{2} + \frac{2}{\pi} \sum_{m=1,3}^{\infty} \frac{\sin\ m\theta}{m} \right\} \sum_{n=1,2,3}^{\infty} a_n \sin n\theta \tag{2.28}$$

which can be converted to:

$$P_r = P_o \left[\frac{1}{2} \sum_{n=1,2,3}^{\infty} a_n \sin n\theta + \frac{1}{\pi} \left\{ \sum_{n=1,3}^{\infty} \frac{a_n}{n} + \sum_{m,n}^{*} a_n \left[\frac{1}{n-m} + \frac{1}{n+m} \right] \cos m\theta \right\} \right]$$

in which: $\displaystyle \sum_{m,n}^{*} = \sum_{n=1,3}^{\infty} \cdots \sum_{m=2,4}^{\infty} \cdots + \sum_{n=2,4}^{\infty} \cdots \sum_{m=1,3}^{\infty}$

This expression is continuous on the whole contour of the hole and obeys (2.27). The terms with odd values of n represent the symmetric part of the load on the edge, the terms with even n the asymmetric part.

Now by taking $X = P_r \cos\theta$ and $Y = P_r \sin\theta$ we can derive from (2.28):

$$\int_0^s X.ds = -f_1 + \frac{R_x\theta}{2\pi} + \frac{R_x}{4} + \frac{P_o}{4} \sum_{n=1,3}^{\infty} \left[\frac{a_n + a_{n+2}}{(n+1)}\right] \tag{2.29}$$

$$\int_0^s X.ds = +f_2 + \frac{R_y\theta}{2\pi} + K \tag{2.30}$$

where with a_1 chosen as unity:

$$R_x = \int_0^{2\pi} X.ds = P_o \left\{a_2 + \sum_{n=2,4}^{\infty} \frac{a_n + a_{n+2}}{(n+1)}\right\} \tag{2.31}$$

$$R_y = \int_0^{2\pi} Y.ds = (P_o/2) \cdot \pi \tag{2.32}$$

The functions f_1, f_2 and K are given in Appendix E.

Hence with (2.29) and (2.30) the boundary conditions become:

$$2\mathrm{Re} \sum_{k=1,2} \left\{B_k + \overset{o}{\phi}_k(z_k)\right\} = f_2 + K + K_1$$

and

$$2\mathrm{Re} \sum_{k=1,2} \left\{s_{k\phi} B_k + (s_{k\phi})\, \overset{o}{\phi}_k(z_k)\right\} = f_1 - \frac{R_x}{4} - \frac{P_o}{4} \sum_{n=1,3}^{\infty} \frac{a_n + a_{n+2}}{(n+1)} + K_2 \tag{2.33}$$

Now with equations (3.32) continuous on the whole contour a solution for the holomorphic functions ϕ_k can be obtained.

As previously mentioned, the terms $\overset{o}{\phi}_k(z_k)$ can be expressed as series of negative powers of z_k with unknown coefficients. Also the replacement of z_k by $\zeta_k = \sigma$ may be done on the hole edge.

Therefore the boundary conditions given in equations (3.31) can be expressed as follows:

$$2\mathrm{Re} \sum_{k=1,2} \left\{ B_k + \phi_k^o(\sigma) \right\} = f_2 + K + K_1$$

and

$$2\mathrm{Re} \sum_{k=1,2} \left\{ s_{k\phi} B_k + (s_{k\phi})\, \phi_k^o(\sigma) \right\} = f_1 - \frac{R_x}{4} - \frac{P_o}{4} \sum_{n=1,3}^{\infty} \frac{a_n + a_{n+2}}{(n+1)} + K_2 \qquad (2.34)$$

The constants on the right hand sides of the above equation determine the constants B_k and the translation of the plate as a rigid body. These equations may now be simplified to:

$$2\mathrm{Re} \sum_{k=1,2} \phi_k^o(\sigma) = f_2$$

$$2\mathrm{Re} \sum_{k=1,2} s_{k\phi}\, \phi_k^o(\sigma) = f_1 \qquad (2.35)$$

which may be combined to give:

$$(s_{\ell\phi} - s_{k\phi})\, \phi_k^o(\sigma) + (s_{\ell\phi} - \overline{s_{k\phi}})\, \overline{\phi_k^o(\sigma)} + s_{\ell\phi} - \overline{s_{\ell\phi}}\, \overline{\phi_k^o(\sigma)} = s_{\ell\phi} f_2 - f_1 \qquad (2.36)$$

Now $\phi_k^o(\sigma)$, continuous on the edge of the hole, is the boundary value of the function $\phi_k^o(\zeta_k)$ on the edge of the hole. Also $\phi_k^o(\infty) = 0$ since the plate is infinite and at infinity the stresses must be zero. Now by considering Cauchy's integral we can write:

$$\frac{1}{2\pi i} \oint \frac{\phi_k^o(\sigma)}{\sigma - \zeta_k} \cdot d\sigma = \phi_k^o(\zeta_k)$$

$$\frac{1}{2\pi i} \oint \frac{\overline{\phi_k^o(\sigma)}}{\sigma - \zeta_k} \cdot d\sigma = 0 \qquad (2.37)$$

and applying (2.37) to (2.36) results in:

$$-(s_{\ell\phi} - s_{k\phi})\, \phi_k^o(\zeta_k) = \frac{s_\ell}{2\pi i} \oint \frac{f_2}{\sigma - \zeta_k} \cdot d\sigma - \frac{1}{2\pi i} \oint \frac{f_1}{\sigma - \zeta_k} \cdot d\sigma \tag{2.38}$$

The integral determination of the terms on the right hand sides of this equation are presented in Ref. [3]. The resulting equation is:

$$\phi_k^o(\zeta_k) = \frac{P_0}{2\pi i\,(s_{\ell\phi} - s_{k\phi})} \left\{ \sum_{m,n} \frac{2n\ a_n(m^2 - n^2 + 1 + 2is_{\ell\phi}m)}{N_{m,n}} \cdot \zeta_k^{-m} - \frac{\pi i}{4} \sum_{n=0,1,2}^{\infty} \frac{a_n(1+is_{\ell\phi}) + a_{n+2}(1-2is_{\ell\phi})}{n+1}\, \zeta_k^{-n-1} \right\} \tag{2.39}$$

where $\displaystyle \sum_{m,n} = \sum_{m=1,3}^{\infty} \sum_{n=1,3}^{\infty} + \sum_{m=2,4}^{\infty} \sum_{n=2,4}^{\infty}$

$$a_0 = 0$$

$$N_{m,n} = m\{(m+1)^2 - n^2\}\{(m-1)^2 - n^2\} \qquad k = 1,2 \quad \ell = 3-k$$

Except for the coefficients a_n, the complex stress functions $\phi_k(z_k)$ are completely determined. It now only remains to determine these constants by applying displacements on the edge of the hole, (resulting from some load), in equation (2.10).

By making use of equation (2.39) and equation (2.10) the following equations may be derived:

$$u = \sum_{k=1,2}^{\infty} \left\{ u_{k\phi}\, \phi_k^o(\sigma) + \overline{u_{k\phi}}\, \overline{\phi_k^o(\sigma)} \right\}$$

$$= \frac{P_o}{4} \left\{ \sum_{n=0,1,2}^{\infty} F_3^{(n)} \sin(n+1)\theta - \frac{8}{\pi} \sum_{m,n} F_1^{(m,n)} \cos m\theta \right.$$

$$\left. + \sum_{n=0,1,2}^{\infty} F_3^{*(n)} \cos(n+1)\theta + \frac{8}{\pi} \sum_{m,n} F_1^{*(m,n)} \sin m\theta \right\} \tag{2.40}$$

$$v = \sum_{k=1,2}^{\infty} \left\{ v_{k\phi}\, \phi_k^o(\sigma) + \overline{v_{k\phi}}\, \overline{\phi_k^o(\sigma)} \right\}$$

$$= \frac{P_o}{4} \left\{ \sum_{n=0,1,2}^{\infty} F_4^{(n)} \cos(n+1)\theta + \frac{8}{\pi} \sum_{m,n} F_2^{(m,n)} \sin m\theta \right.$$

$$\left. - \sum_{n=0,1,2}^{\infty} F_4^{*(n)} \sin(n+1)\theta - \frac{8}{\pi} \sum_{m,n} F_2^{*(m,n)} \cos m\theta \right\} \tag{2.41}$$

where the terms F_α^β and $F_\alpha^{*\beta}$ are all given in full in Appendix E.

A final condition of displacement on the hole edge may be obtained by considering displacements in the radial and tangential directions. It can easily be derived that this further condition may be expressed as:

$$u \cos\theta + (v - v_i) \sin\theta = 0 \tag{2.42}$$

Some of the series appearing in the displacement formulae (2.40) and (2.41) may be replaced by analytic expressions that are only valid for $0 < \theta < \pi$. The resulting expressions may be substituted into equation (2.42). This results in:

$$a_{10} + \sum_{n=2,3,4}^{\infty} a_{n0} a_n = 0 \tag{2.43}$$

where $a_{1\theta}$ and $a_{n\theta}$ are known coefficients.

With $$C_8 = C_5 \cos\theta - C_4 \sin\theta$$

$$C_9 = C_4 \cos\theta - C_7 \sin\theta$$

$$C_{10} = C_8 \cos\theta - C_9 \sin\theta$$

and $$C_{11} = -C_8 \cos\theta - C_9 \sin\theta$$

These become:

$$a_{1\theta} = \frac{C_8}{2} \sin 2\theta + \frac{C_9}{2} \cos 2\theta + C_6\left[\theta - \frac{\pi}{2}\right] \cos\theta - \frac{C_7}{2} \sin\theta$$

$$+ \frac{8}{\pi} \sum_{m=1,3}^{\infty} \frac{2C_9 \sin(m\theta) - C_8 m \cos(m\theta) - 2C_7 (-1)^{\frac{m+1}{2}} \sin\theta}{m^2(m-2)(m+2)}$$

and for even n:

$$a_{n\theta} = \frac{2}{n^2-1}\left[C_{10}\cdot n . \sin(n\theta) + C_{11} \cos(n\theta) + 2C_6 \sin(n\theta)\right.$$

$$\left. - \frac{C_6 \cdot n}{\pi}\left\{\frac{4}{n^2-1} \cos\theta - (2\theta-\pi)\sin\theta\right\} - C_4 (-1)^{\frac{n}{2}} \sin\theta\right]$$

$$+ \frac{8}{\pi} \sum_{m=2,4}^{\infty} \left[\frac{2C_9 mn . \sin(m\theta) - C_8 n(m^2-n^2+1)\cos(m\theta) - C_4 n(m^2-n^2+1)\cdot(-1)^{\frac{m}{2}} \sin\theta}{N_{m,n}}\right]$$

and for odd values of n:

$$a_{n\theta} = \frac{2}{n^2-1}\left[C_{10}\cdot n \cdot \sin(n\theta) + C_{11}\cos(n\theta) + 2C_6\sin(n\theta)\right.$$

$$\left. - \left\{2C_6(-1)^{\frac{n-1}{2}} + C_7 n(-1)^{\frac{n-1}{2}}\right\}\sin\theta\right]$$

$$+\frac{8}{\pi}\sum_{m=1,3}^{\infty}\left\{\frac{2C_9 mn\cdot\sin(m\theta) - C_8 n(m^2-n^2+1)\cos(m\theta) - 2mn\cdot C_7(-1)^{\frac{m+1}{2}}}{N_{m,n}}\sin\theta\right\}$$

The terms a_n may now be solved from:

$$[a_{n\phi}]\{a_n\} = \{-a_{1\phi}\}$$

In summary, then

- the stresses around an infinite plate with a hole and loaded pin are described by:

$$\sigma_x = 2\mathrm{Re}\sum s_k^2\,\phi_k'(z_k)$$

$$\sigma_y = 2\mathrm{Re}\sum \phi_k'(z_k)$$

$$\sigma_{xy} = -2\mathrm{Re}\sum s_k\,\phi_k'(z_k)$$

where

- $$\phi_k(z_k) = g^{(k)}\cdot z_k + A_k \ln z_k + \sum_{n=1}^{\infty} g_n^{(k)}\cdot z_k^{-n}$$

Using the substitution

$$\zeta_k = \frac{z_k + \sqrt{z_k^2 - s_k^2 - 1}}{1 - is_k}$$

this may be written as

$$\phi_k(z_k) = A_k \ln \zeta_k + \phi_k^o(z_k) + B_k$$

and $$\phi_k'(z_k) = \frac{a_k}{\zeta_k}\frac{\partial\zeta_k}{\partial z_k} + \phi_k'^o(z_k)$$

where, from above

$$\frac{\partial\zeta_k}{\partial z_k} = \frac{\zeta_k}{\sqrt{z_k^2 - s_{k\phi}^2 - 1}}$$

- The term A_k given in equation (2.22) may be reworked into the following term:

$$A_k = \frac{P_0}{2\pi i(s_{\ell\phi} - s_{k\phi})}\left\{\frac{1}{2}\left[R_x + s_{\ell\phi} \cdot R_y\right]\right.$$

$$\left. + S_{11} \cdot A \cdot \left[R_x + s_{k\phi} \cdot R_y\right]\left[\overline{s}_{\ell\phi} - \overline{s}_{\ell\phi}\right]\left[\overline{s}_{k\phi} - \overline{s}_{\ell\phi}\right]\right\}$$

where $$A = \frac{C_6}{4S_{11}^2 s_k s_\ell (s_k + s_\ell)^2}$$

- Forcing z_k to the hole edge where $\zeta = z = \sigma = 1$, enables a boundary value to be inserted into the Cauchy integral given in equation (2.37) to yield the function for $\phi_k^o(\zeta_k)$ given in equation (2.39). $\phi_k^{o'}(\zeta_k)$ is then easily derived and the stress may be evaluated anywhere in the plate.

2.4 Superposition of Stress States

Due to the form of equation (2.11), it can be shown that the terms representing the homogeneous stress field and the effects of a pin loaded hole may be calculated separately and the final stress field obtained by superposition.

This technique is useful when including practical issues such as finite specimen width and by-pass loads.

2.5 Computerisation of the Mathematical Model

Although it is not the purpose of this thesis to examine computational software or hardware, it was believed necessary to include a brief discussion due to the rapid developments occurring in microprocessors and software. It is felt that this aspect should be taken into consideration when evaluating the analysis technique presented in this thesis. The development of parallel iterative solution algorithms and the associated hardware for the finite element method are relevant to comparisons of computational efficiency. The software language OCCUM, although considered to have the attribute of simplicity (Davis [31]) for expressing many of the requirements of concurrency software, is still a long way from being a competitive tool for the solution of finite element equations. In fact, the advantages which can be gained from concurrency are equally applicable to the technique presented in this thesis, and are more easily applied.

Another factor to be considered is the availability of FEM software having the capability of handling layered and orthotropic elements and the additional complications of model complexity due to the contact problem.

It was with some of these factors in mind that a decision was made to implement the method of complex stress functions as presented by De Jong on a medium performance personal computer. Although the manipulation of a computational process involving complex variables would have been far simpler on a main frame computer using FORTRAN with complex variable capability, this would to some extent detract from one of the main attractions of the method, viz. portability.

Numerous languages are now available on personal computers, and the features of high-level languages which support good software design methods, as well as the need to improve the likelihood of software correctness and reliability are discussed in length by Davies [31]. With the possibility of using the method presented in this paper in a modular approach to the design of a data-base-centered modular design of a more expansive composite design and analysis software system, the"C" language was chosen. Although this is a high level language, it has the advantage of being able to compile and run software models with multiple data segments, each 64k in size and up to 1 Mb for code on a personal computer having 640k ram and a hard or floppy disc.

The program BHOLES has subsequently been translated into pascal and implemented on an APOLLO domain 3000 system with improved graphics interfacing. It should be noted that a BASIC preprocessor was used in the original version for this thesis as well as a simple graphics program, also in BASIC. These are not discussed. Appendix L contains the main section of the C program. In order to illustrate the coding a simple flowchart is given here.

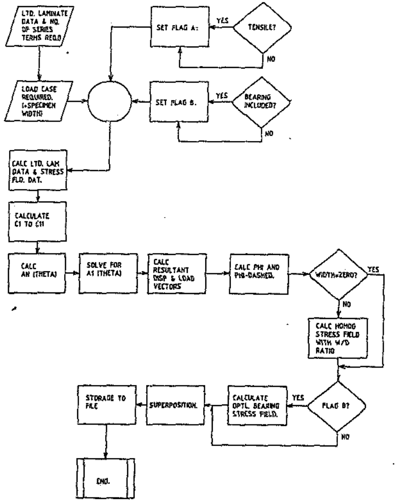
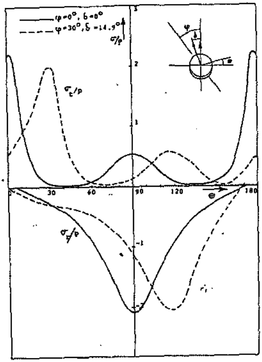


Figure 2.5 : Simplified flowchart for basic model.

2.6 Graphical Representation of the Stress State

The generally accepted method of representation of the stresses in the region of a hole or a pin-loaded hole in a composite laminate is a plot of radial and tangential stresses around the edge of the hole. It is however believed by this author that this method which may be applicable for isotropic studies is not applicable for directional multi-component materials. A typical representation of the stresses in a laminate as presented by De Jong [2] is given in Figure 2.6.1 for reference. By considering a more detailed stress distribution as described in Chapter 2.7, it is believed three graphs will not complicate matters much more than two. To this end, instead of using tangential and radial stresses, stresses parallel to the fibre direction, perpendicular to the fibre direction and shear in the matrix are used. Further, the distribution of these stresses in the whole region surrounding the hole may be beneficial in obtaining a better understanding of the mechanisms of load transfer and lead to theories of optimal lay up rules suitable for designers. The presentation of the stress distribution around a pin-loaded hole or hole is therefore represented by means of a three dimensional plot as illustrated in Figure 3.1.2(a) for a unidirectional carbon laminate with a pin load applied at 15° to the direction of the fibres. One further plot is provided for completeness. This is a plot of failure propensity where, in this thesis, the failure propensity is based on the Tsai-Hill failure criterion and is used to locate the position of the onset of first damage, as well as critical load distributions. A set of four, (three-dimensional), plots per layer are thus used to describe the stress state. A complete set of graphs as obtained on an APOLLO are shown in Figure 2.6.2. It should be noted that for a laminate, the production of four graphs per layer may seem excessive,but it must be kept in mind that such a system will enable the study of layer orientation effects on individual layer stress distributions,and not only may this enable optimum laminate design but will enable further understanding of the mechanisms of load transfer in multi-layered orthotropic systems.



THE STRESS DISTRIBUTION AROUND A PIN-LOADED HOLE IN A UNIDIRECTIONAL C.F.R.P. LAMINATE.

Figure 2.6.1 : Two-dimensional stress plot (De Jong [3])

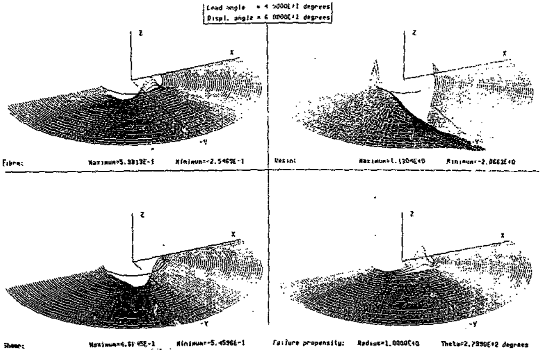


Figure 2.6.2 : Apollo domain graphics output.

2.7 Failure Criteria

2.7.1 The Tsai–Hill failure criterion

A multitude of failure criteria have been proposed by many researchers. Some of these are simple while others are complicated. A discussion on the various failure criterion is avoided. In order to verify results with those presented by De Jong the Tsai–Hill failure criteria has been applied in thi hesis. Failure propensities are therefore also based on this criteria.

While attempting to correlate results with those published by De Jong, it was found that this failure criteria resulted in close correlation for single layered materials but not for multilayered materials. It was discovered that De Jong applied this criteria to multilayered materials using test values of strength obtained from tests on these multilayered laminates. Using these values of strength in the principle directions quoted by De Jong, good correlation was found. Using the single layer test values applied to individual layers, and stresses obtained by applying classical laminate theory to the laminate in order to get the individual layer stresses, resulted in large errors. It was therefore believed to be necessary to include a brief investigation into the possible reasons for this discrepancy so as to avoid possible erroneous application of this failure criterion.

The basic form of the failure criterion is:

$$\frac{\sigma_1^2}{\overline{\sigma}_1^2}+\frac{\sigma_2^2}{\overline{\sigma}_2^2}-\frac{\sigma_1}{\overline{\sigma}_1}\frac{\sigma_2}{\overline{\sigma}_2}+\frac{\sigma_3^2}{\overline{\sigma}_3^2} = 1 \tag{2.44}$$

The envelope formed by this criteria is illustrated in Figure 2.7.1. Only the upper half is shown to improve clarity. The differing lobe sizes due to the differing ultimate strengths should be noted.

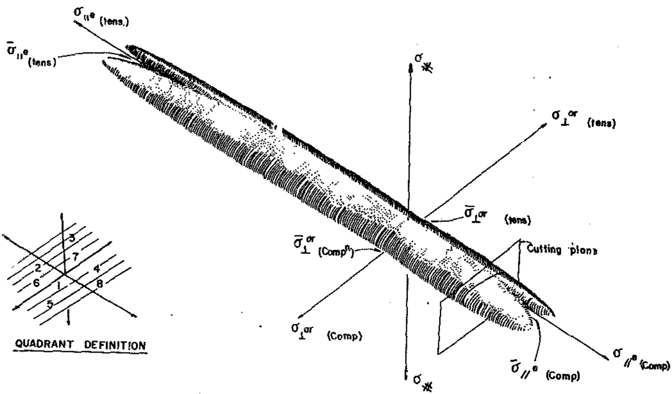


Figure 2.7.1 : Visualisation of Tsai-Hill failure envelope.

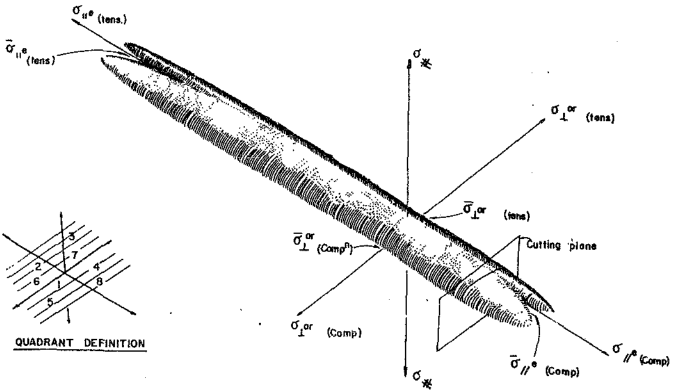


Figure 2.7.1 : Visualisation of Tsai-Hill failure envelope.

In order to determine the probable mode of failure, it was necessary to extract more information from the criteria than just a "yes/no" type of failure prediction. By considering a cross section through the envelope, as illustrated in Figure 2.7.2, the sensitivity of a lamina to either the stress parallel to the fibre direction (σ_1), perpendicular to the fibre direction (σ_2) or due to shear (σ_3) may be investigated. Figure 2.7.2 shows this cross-section in detail. Again only the top half of the envelope is shown.

Referring to Figure 2.7.2, the two points A and B represent two stress states with the same parallel stress (i.e. stress states occurring in the same cutting plane). If one were now to calculate a reserve factor based on extending the line 0A to a_3 and using the definition of reserve factor equal to $a_3$0 divided by A0, it can be seen that a point B could also be located having the same reserve factor given by $b_3$0 divided by B0.

Assume now that the σ_2 stresses remain fixed as well. Any change in the stress state can now only be achieved by varying the third. The variable stress can now be used to calculate a reserve factor with respect to that stress only. In this case the use of the σ_3 stress indicates a larger failure propensity at point A than at point B.

Using this approach it is possible to extract five reserve factors which will more clearly define the stress state of a layer. These are given below:

- *The Tsai-Hill absolute value plus a quadrant indicator.*
- The vectored reserve factor.
- The parallel stress reserve factor.
- The perpendicular stress reserve factor.
- The shear stress reserve factor.

where the "vectored" reserve factor is defined as that obtained by maintaining a constant ratio of parallel, shear and perpendicular stresses.

This approach has been applied to a [90/±45] carbon laminate. Appendix B gives *details of results for this laminate. These results are summarised in* Table 2.7.1.

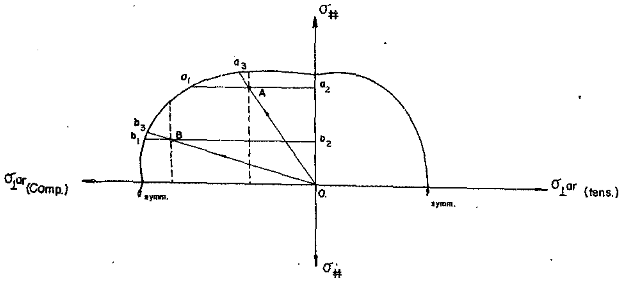


Figure 2.7.2 : Section through Tsai-Hill failure envelope.

Table 2.7.1 : Strength predictions for a [90/±45]s laminate.

PROPERTY	BASIC LAYER PROPERTIES (TEST)	LAMINATE PROPERTIES (TEST)	CALCULATED LAMINATE FPF AND MODULI
E_x	9.11 GPa	23.62 GPa	24.79 GPa
E_y	156.39 GPa	80.03 GPa	88.42 GPa
G_{xy}	5.35 GPa	16.55 GPa	22.67 GPa
ν_{xy}	0.0198	0.1650	0.19787
σ_1(Tens)	64 MPa	212 MPa	202 MPa
σ_1(Comp)	212 MPa	-286 MPa	-233 MPa
σ_2 (Tens)	1600 MPa	1072 MPa	665 MPa
σ_2 (Comp)	1042 MPa	-805 MPa	-665 MPa
σ_3	70 MPa	217 MPa	-300+ MPa

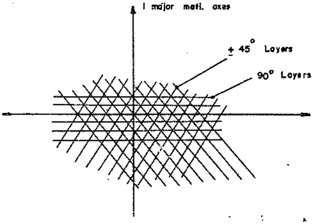


Figure 2.7.3 : Axes convention used by De Jong [2].

CHAPTER 3
MODEL VERIFICATION

Verification of the computer model was done in four ways. These were:

- Correlation with expectations on a physical level.
- Correlation of intermediate results with similar results by De Jong.
- Correlation of failure load and locations for a few laminates with De Jong.
- Test result correlation - De Jong [2].

Each of these will be discussed in turn.

3.1 Verification by Physical Interpretation

One major aim of this thesis was to develop a model capable of describing the mechanisms of load transfer that occurs when a pin bears against the edge of a hole in a composite laminate. In order to justify the stress fields produced by this model a unidirectional laminate with a pin-load aligned in the direction of the fibres is considered. Figure 3.1.1 shows a simplified representation of such a situation. The diagram also shows expected parallel, transverse and shear stresses. These are easily seen to correlate with the graphical program outputs shown in Figure 3.1.2(a) and 3.1.2(b).

For the case where the load angle is no longer in the same direction as the fibres, no detailed discussion is given. It should however be noted that the load and displacement angles are no longer coincident,that the maximum compressive stresses are smaller and occur at an angle to the load direction. Figure 3.1.2 shows the case for a displacement angle of 30 degrees with the fibre direction.

Appendix (A) contains numerous plots for various cases of loading. In these plots the angle of displacement is varied, the ratio of specimen width to hole diameter is varied and in some cases pure tensile or compressive fields are represented or superimposed onto the stress field due to a pin loaded hole in a plate.

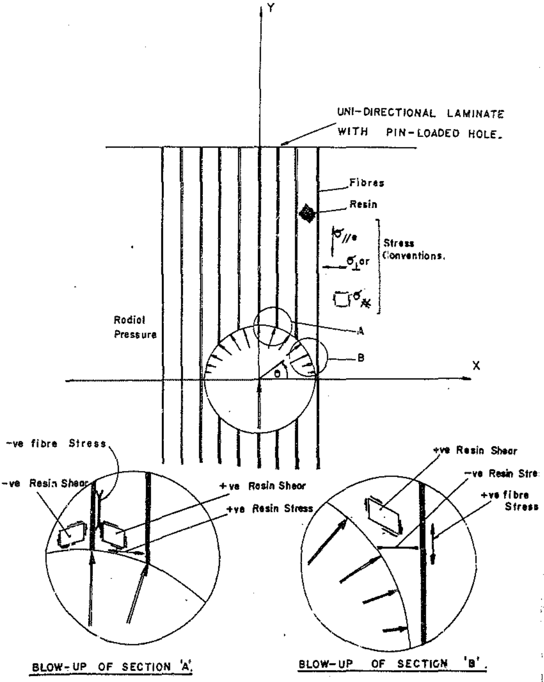


Figure 3.1.1 : Justification of results on physical expectation.

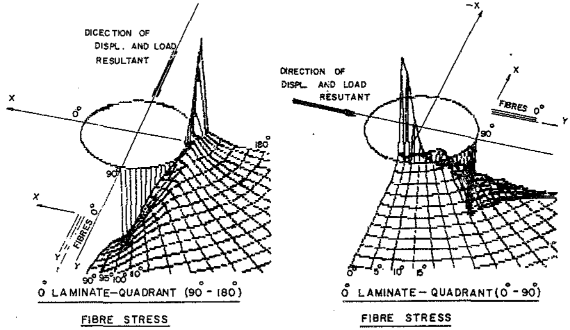


Figure 3.1.2(a) : Three dimensional plots.

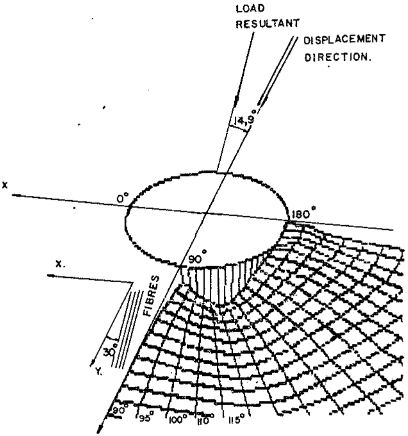


Figure 3.1.2(b) : 30° laminate - quadrant (90° to 180°) fibre stress.

It should be noted that the last plot in this series is for a [±45°] laminate and the stresses can no longer be treated as stresses in the fibre direction, perpendicular to the fibre direction and shear. The failure propensity diagram is also not valid. Further work is required in this area to reduce the laminate stresses to individual layer stresses. This entails the development of a more detailed post processor.

3.2 Intermediate Result Correlation

The most intermediate result is the production of the a_n terms described by equation (2.42). A qualitative estimate of the accuracy of solution can be made by examining the terms for any symmetrical displacement case where all even terms produced should ideally be zero in order that symmetry of the pressure distribution is maintained. Results for a unidirectional carbon laminate obtained from program BHOLES is compared in Table 3.1 with results published by De Jong [2].

Table 3.1 : Correlation of intermediate results.

BHOLES		DE JONG	
UD = (0 deg)		UD = (0 deg)	
$a_n[1]$	+1.000e-000	a(1)	0.100d+01
$a_n[2]$	+2.625e-011	a(2)	-0.111d-05
$a_n[3]$	-3.823e-001	a(3)	-0.382d+00
$a_n[4]$	+4.271e-012	a(4)	0.467d-07
$a_n[5]$	+1.429e-001	a(5)	0.142d+00
$a_n[6]$	+1.173e-011	a(6)	-0.159d-05
$a_n[7]$	-6.377e-002	a(7)	-0.637d-01
$a_n[8]$	+9.817e-012	a(8)	-0.268d-05
$a_n[9]$	+1.094e-002	a(9)	0.109d-01
$a_n[10]$	-1.200e-011	a(10)	-0.393d-05
$a_n[11]$	-1.585e-002	a(11)	-0.158d-01
$a_n[12]$	+1.252e-011	a(12)	-0.460d-05
$a_n[13]$	-4.667e-003	a(13)	-0.466d-02
$a_n[14]$	+1.284e-011	a(14)	-0.468d-05
$a_n[15]$	-6.581e-003	a(15)	-0.657d-02
$a_n[16]$	+1.191e-011	a(16)	-0.407d-05
$a_n[17]$	-3.601e-003	a(17)	-0.359d-02
$a_n[18]$	+9.738e-012	a(18)	-0.295d-05
$a_n[19]$	-2.434e-003	a(19)	-0.243d-02
$a_n[20]$	+6.267e-012	a(20)	-0.161d-05
$a_n[21]$	-9.774e-004	a(21)	-0.974d-03
$a_n[22]$	+2.155e-012	a(22)	-0.478d-06
$a_n[23]$	-2.599e-004	a(23)	-0.257d-03

3.3 Correlation of Failure Load Predictions

(i) Case 1: UD laminate with 0 deg displacement angle

Input Data:

Lamina Properties :	E11 = 9.11 GPa.
	E22 = 156.39GPa
	v12 = 0.0198
	G12 = 5.35 GPa.
Lamina Compliancies:	S11 = 1.097695E-10
	S22 = 6.394271E-12
	S12 = -2.173436E-12
Displacement Angle:	0 degrees
Directionality:	0.2413541
Angularity:	0.08316018
Width to hole diameter ratio:	infinite
No. of terms in approximations:	50

Output Data:

(a) Failure propensity diagram gives failure location at 90 degrees.

(b) Extracted stresses for Pb = 123,5 MPa are

σ_1 = -254.40 MPa

σ_2 = -70.15 MPa

σ_3 = 0.00 MPa

(c) Vectored reserve factors:

Tsai-Hill value = 0.993

Vectored reserve factor = 1.003

Reserve factor (σ_1) = 3.534

Reserve factor (σ_2) = 1.003

Failure is therefore by splitting at 90 degrees and a load of 123.5 MPa.
Failure value calculated by De Jong - 123 MPa.

(ii) **Case 2: UD carbon laminate with a 30 degree pin displacement angle.**

Input Data:
Same as for case (i) but displacement angle = 30 degrees.

For this case detailed results analysis is not given. Refer instead to Chapter 2.7.1 on extension of the Tsai-Hill failure criterion. The results are presented to demonstrate the fact that failure occurs at neither the location of maximum shear, fibre or resin stresses, but at some combination of them. This location is however dependent on the particular failure criterion used to locate it.

Results:

(a) *Failure at 105°*

(b) Vectored reserve factors:

Tsai-Hill value	=	1.007
Reserve factor (σ_1)	=	1.808
Reserve factor (σ_2)	=	1.020
Reserve factor (σ_3)	=	1.004

Failure is at a load of 119 MPa.
Failure value calculated by De Jong = 118 MPa.

The method of calculating this location and value is somewhat more laborious than locating it by means of a propensity diagram.

Table 3.2 shows the relevant values and vectored-reserve-factor based deduction of the failure modes. The resulting load resultant angle is approximately half of the displacement angle with a value of 14,9 degrees.

The data in Table 3.2 is presented in Figure 3.3.

Table 3.2 : Stresses and reserve factors for unit load = 1 N for UD carbon.

ANGLE	RESIN (Stress Pa)	FIBRES (Pa)	SHEAR (Pa)	RF RESIN	RF FIBRES	RF SHEAR	RF TOTAL	MODE
80	-1.63 e^{-1}	-2.387 e^{-1}	-2.153 e^{-1}	6410	885	324	307.7	Shear
85	-1.37 e^{-1}	-3.168 e^{-1}	-2.462 e^{-1}	7576	885	431	397.7	Shear
90	-9.73 e^{-2}	-4.148 e^{-1}	-2.753 e^{-1}	10720	510	254	228.8	Shear
95	-3.86 e^{-2}	-5.363 e^{-1}	-2.966 e^{-1}	27030	395	236	202.8	Shear
100	3.92 e^{-2}	-6.781 e^{-1}	-3.007 e^{-1}	40820	312	232	186.9	Shear
105	1.297 e^{-1}	-8.289 e^{-1}	-2.769 e^{-1}	12330	255	253	180.8	Shear/Buckling
110	2.181 e^{-1}	-9.656 e^{-1}	-2.155 e^{-1}	7353	219	325	183.8	Buckling
115	2.81 e^{-1}	-1.052	-1.173 e^{-1}	5682	201	404	182.8	Buckling
120	2.96 e^{-1}	-1.055	2.766 e^{-6}	5405	200.8	25 e^{9}	204.5	Buckling
125	2.58 e^{-1}	-9.708 e^{-1}	1.082 e^{-1}	6211	218	645	210.1	Buckling
130	1.83 e^{-1}	-8.209 e^{-1}	1.828 e^{-1}	8772	258	383	216.5	Buckling

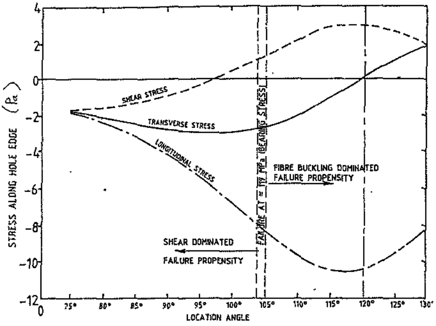


Figure 3.3 : Plot for $\phi = 30$ degrees {UD carbon}. Stresses at hole edge for load angle $\phi = 30$ degrees.

3.4 Test Results (De Jong)

De Jong [2] ran tests in verification of the analytical results of a model which included pin flexibility, friction and clearance. The tests involved accurate measuring of the elastic response of the test specimens as well as the acoustic emissions.

These results will not be discussed in detail here except to comment that no significant damage is reported to have occurred until loads in excess of 80 percent of the theoretically predicted failure load. This is attributed to, amongst other things, experimental error and to some extent the failure criteria used.

It should be noted that the experimental results appear to correlate very well with theory for elastic response and that bolt clearance has a significant effect on the results.

CHAPTER 4
DISCUSSION OF RESULTS

A literature survey on mechanical joints in composite laminates shows that much work has been done in this area of advanced composite structures.

Although empirical methods have been developed and may be a useful method for obtaining parametric information or even as the basis for a design methedology, they are expensive, (due to the large number of variables), and do not provide an insight into the mechanisms of load transfer. They are therefore limited in the amount of understanding that can be obtained from them (due to the large number of unknowns).

Methods of obtaining the stress distribution, elastic response, and failure load (and mode) include finite element models. This method was however not deeply investigated due to the fact that the amount of work required to build a model, and the amount of effort required to alter a lay-up or width to diameter ratio, is large when compared to the less detailed model afforded by complex stress functions, (the main loss in detail occurring in the lack of three dimensional effects).

The method of complex stress functions has been used to develop a model for a hole or a pin-loaded hole in a composite laminate with variable width to diameter ratios and load angles. The program BHOLES has been shown by correlation with an independently developed model to generate accurate data. Indications are that a higher degree of computational accuracy has been achieved in a program capable of running on a personal computer than were achieved on a main frame using Fortran IV, thus also adding an element of portability.

Theoretical strength predictions have not been correlated directly with test results. This is mainly due to the high cost of the required test equipment and testing. Although BHOLES does not include the effects of pin flexibility or friction, indications of the effectiveness of the method can be indirectly obtained by referring to results produced by De Jong [2]. Test results showed that extensive damage was not detected until a load of around 80 percent of the theoretical failure load was applied. The elastic response predictions appear to correlate well for pin clearances of 0 to 2 percent.

Some experimental work has been done on the effects of bolt torque on the bearing strength of glass fibre laminates (3G) and indications are that increases in joint efficiency of as much as 20 percent can be achieved.

CHAPTER 5
CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER WORK

Finite element modelling of bolted joints is probably the most effective method of studying the behaviour of bolted joints in composite laminates due to the fact that three dimensional effects can be included in the model. This means that bolt torque and edge effects can also be investigated. However, the work required by this method and the complications involved in modelling an orthotropic layered system with bearing loads make it an expensive and highly specialised tool which should only be applied at a later stage for deeper investigations.

Complex stress functions offer an efficient and apparently effective analytical/ computational tool for investigating the behaviour of pin-loaded joints. Also the spin-offs from this approach are more extensive.

For example:

- Graphical design data may be produced efficiently. These may be corrected for three dimensional effects using data obtained from limited testing.

- The method may also be used to analyse square, elliptical and triangular cut-outs in Laminates using conformal transformations.

- Exact solutions for specific situations may be derived which are simple enough to be implemented on a programmable calculator.

The importance of being able to design and analyse mechanical joints in composite laminates cannot be ignored. Computational techniques for investigating the behaviour of such joints for purposes of understanding, preliminary concept evaluation and the production of design data, is made particularly attractive by the rapid growth in computer technology. The high cost of obtaining relevant data due to the vast number of unknowns makes a computational approach even more attractive.

In the course of this research numerous related areas of importance were isolated. Some of these are listed below for reference:

- Effective failure criteria and an efficient materials data base linked to a statistical data analyser are required.

- *Highly loaded advanced composite structures* are often limited by the effectiveness of joining methods in general.

- The nature of advanced composite structures requires effective analysis. This analysis can be achieved by the implementation of situation dependent computational units which are capable of operating interactively with an effective data-base.

- Mechanical joining techniques may be effective in the integration process and as such may be assisted by bonding.

Much work remains to be done on joints in composite materials in general, on mechanical joints, and the complex stress function approach to the computation of the behaviour of advanced composites. The model presented in this dissertation is by no means complete and much scope for further development exists. Some aspects still requiring attention are listed below:

- The present model must be upgraded to include pin flexibility, pin friction, and shear in the homogeneous stress field.

- The approach can be extended to rows of pin-loaded holes, thus allowing investigation of effective bolt pitch.

- The exact solutions for specific situations should be investigated for general applicability.

- Application of the method to other cut-out shapes should be investigated.

- The model at present requires a fair amount of pre- and post-processing. This aspect has yet to be fully developed.

- Drivers for the data generation unit can be developed which are aimed at the generation of design data.

- Probably most importantly of all, accurate test procedures must be developed to provide information on the accuracy of the model, and the role of three dimensional effects.

REFERENCES

[1] Bidgood, P.M. : Mechanical joints in composites - Literature survey, summary and notes. (Unpublished).

[2] De Jong, T. and Klang, E.C. : *Pinned connections in composite* materials - Theory and experiment, Delft University of Technology, Report No. LR-445, Jan 1985.

[3] de Jong, T. and Vuil, H.A. : *Stresses* around pin-loaded holes in elastically orthotropic plates with arbitrary load direction, Delft University of Technology, Report No. LR-333.

[4] *Bidgood, P.M. : Mechanical joints* in composites, Research project for BSc Eng (Mech(aero)), University of the Witwatersrand, South Africa, Dec. 1984.

[5] Eisenman, J.R. and Loenhardt, J.L. : Improving composite bolted joint *efficiency* by laminate tailoring, ASTM, STP 749, 1989, pp. 117-130.

[6] Poon, C. : Literature review on the design of composite mechanically fastened joints, National Aeronautical Establishment, Ottawa, February 1986.

[7] Hart-Smith, L.J. : Mechanically-fastened joints for advanced composites - phenomenological considerations and simple analyses, Structures Subdivision, Douglas Aircraft Company, McDonnel Douglass Corporation - in Proceedings of Fourth Conference on Advanced Composites in Structural Design, 1979.

[8] Collings, T.A. : The use of bolted connections as a means of joining carbon fibre reinforced plastics, Structures Department, Royal Aircraft Establishment, Farnborough. Report No. C229/77, in Designing with Fibre Reinforced Materials. (I Mech E Conf. Publications, 1977-1979).

[9] Matthews, F.L. (et al) : The bolt bearing strength of glass/carbon hybrid composites, in Composites, Butterworth and Co (Publishers) Ltd, July 1982.

[10] Oplinger, D.W. : On the structural behaviour of mechanically fastened joints in composite structures, Army Materials and Mechanics Research Centre, Watertown, Massachusetts.

[11]* Sandifer, J.P. : Fretting fatigue on graphite/epoxy composites, AIAA, Paper No. 77-418, 1977.

[12] Goodwin, E.W., Mattews, F.L. and Kilty, R.F. : Strength of multi-bolt joints in GRP, in Composites, Butterworth and Co (Publishers) Ltd, July 1982.

[13] Rosenfield, M.S. : The effect of a corrosive environment on the strength and life of graphite-epoxy mechanically fastened joints, Naval Air Development Centre, Warminster, Pa.

[14] Air Force Materials Laboratory : Corrosion behaviour of metal fasteners in graphite-epoxy composites, Air Force Systems Command, Wright-Patterson AFB, Ohio, Febr. 1975.

[15]* Whitney, J.M. and Nuismer, R.J. : Stress fracture criteria for laminated composites containing stress concentrations, Journal of Composite Materials, Vol. 8, July 1974, pp. 253-265.

[16] Furguson, G. : Tests to establish the effect of bolt torque on bolted joints in FRP, Undergraduate Research, University of the Witwatersrand, South Africa. (Unpublished Report).

[17] Waszcak, J.P. and Cruise, T.A. : Failure mode and strength predictions of anisotropic bolt bearing specimens, Department of Mechanical Engineering, Carnegie-Mellon University, Pittsburgh, Pennyslvania in J. Composite *materials, Vol. 5, July 1971, p. 421.*

[18]* Chang, F.K., Scott, R.A. and Springer, G.S. : Design of composite laminates containing a pin-loaded hole, Journal of Composite Materials, Vol. 18, 1984, *pp. 279-289.*

[19]* Agarwal, B.L. : Behaviour of multifastener bolted joints in composite materials, AIAA-80-0307, 1980.

[20] Soni, S.R. : Failure analysis of composite laminates with a fastener hole, Joining of Composite Materials, ASTM, STP 749, K.T. Kedward, Ed., American Society for Testing and Materials, 1981, pp. 145-164.

25 x

[21]* York, J.L. et al : Analysis of the net tension failure mode in composite bolted joints, Journal of Reinforced Plastics and Composites, Vol. 1, April 1982, pp. 141-152.

[22] Rowlands et al : Single- and multiple-bolted joints in orthotropic materials, in Composites, Butterworth and Co (Publishers) Ltd, July 1982.

[23] Mathews, F.L. et al : Stress distribution around a single bolt in fibre-reinforced plastic, in Composites, Butterworth and Co (Publishers) Ltd, July 1982, pp. 316-321.

[24]* Rybicki, E.F. and Schmueser, D.W. : Effect of stacking sequence and lay-up angle on free edge stresses around a hole in laminated plate under tension, in J. Composite Materials, Vol. 12, No. 4, July 1978, p. 300.

[25] Tung, T.K. : On the computation of stresses around a single bolt in fibre-reinforced plastic, in J. Composite Materials, Vol. 21, Butterworth and Co (Publishers) Ltd, Febr. 1987.

[26] Mathews, F.L. and Leong, W.K. : The strength of riveted joints in CFRP, Department of Aeronautics, Imperial College of Science and Technology, London, England, in ICCM 3, Vol. 2, pp. 1241-1246.

[27] Cole, R.T. (et al) : Fasteners for composite structures, in Composites, Butterworth and Co (Publishers) Ltd, July 1982, pp. 233-240.

[28] Johnson, M. and Mathews, F.L. : Determination of safety factors for use when designing bolted joints in grp, in Composites, IPC Business Press, April 1979, pp. 73-76.

[29] Tseng-Hua Tsiang : Survey of bolted-joint technology in composite laminates, Massachusetts Institute of Technology, Cambridge, MA 02139, in Composite Technology Review, Vol. 6, No. 2, Summer 1984.

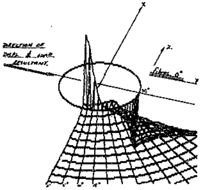
[30] Goodwin, E.W. and Mathews, F.L. : A review of the strength of joints in fibre-reinforced plastics, in Composites, Butterworth and Co (Publishers) Ltd, July 1982, pp. 155-160.

[32] Davies, A.C. : Features of high-level languages for microprocessors, in Microprocessors and Microsystems, Vol. 11, No. 2, March 1987, pp. 77-87.

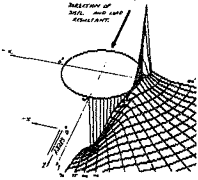
25

APPENDIX A
THREE-DIMENSIONAL STRESS PLOTS

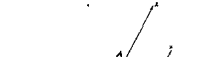
25X



0° LAMINATE – QUADRANT (0–90°)
FIBRE STRESS

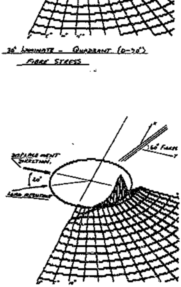


0° LAMINATE – QUADRANT (90°–180°)
FIBRE STRESS

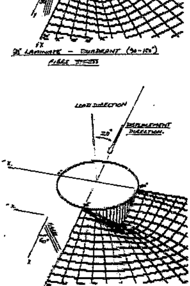


30° LAMINATE – QUADRANT (0–90°)
FIBRE STRESS

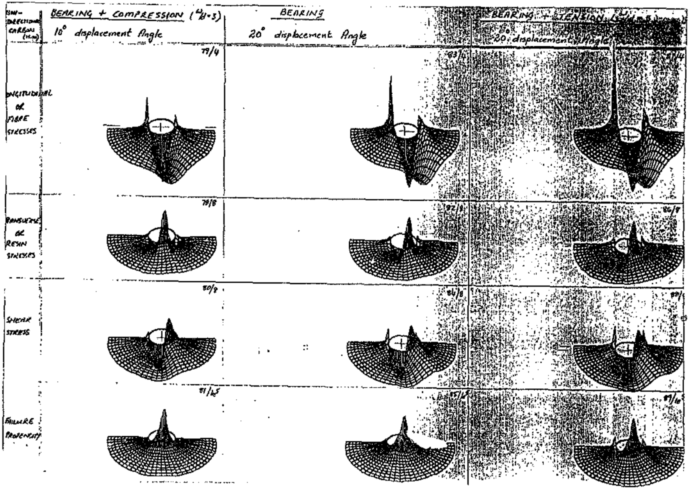
30° LAMINATE – QUADRANT (90–180°)
FIBRE STRESS

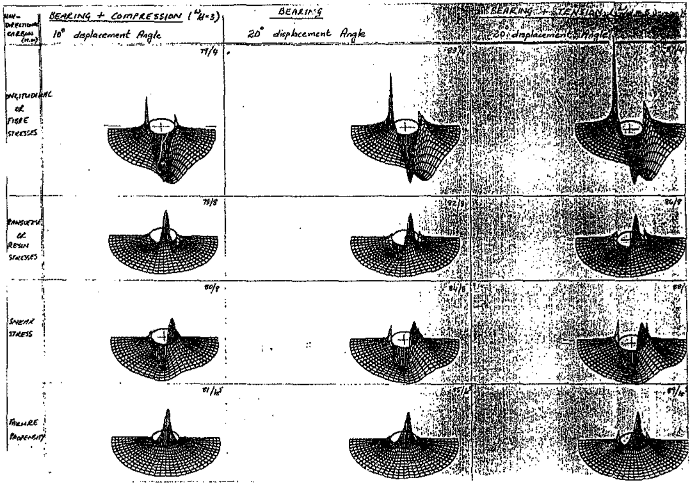


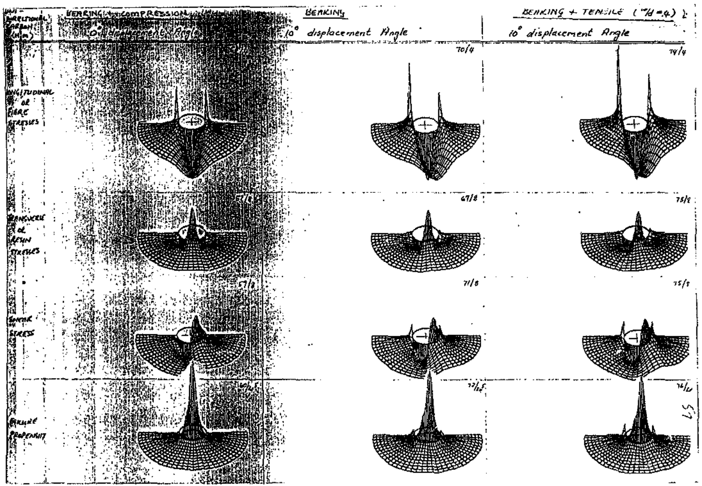
60° LAMINATE – QUADRANT (0–90°)
FIBRE STRESS

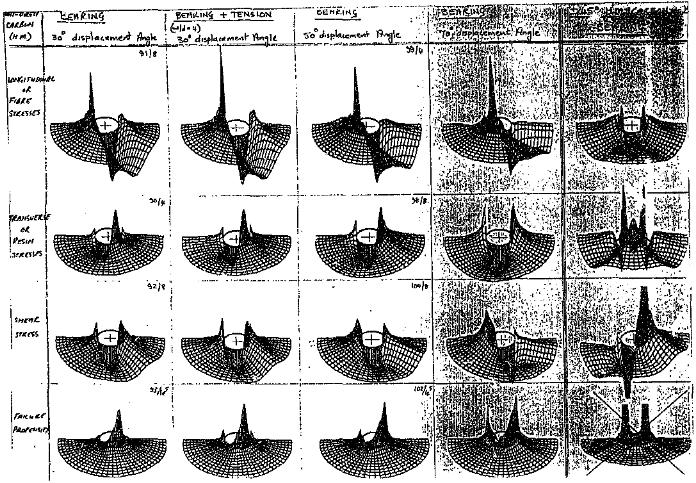


60° LAMINATE – QUADRANT (90–180°)
FIBRE STRESS


BEARING + COMPRESSION
10° displacement Angle
BEARING
20° displacement Angle
BEARING + TENSION
20° displacement Angle
SHEAR STRESS
FAILURE PROPENSITY
79/4
79/8
80/8
83/4
82/8
84/8
87/4
86/8


UNI-DIRECTIONAL CARBON (H.M)
BEARING + COMPRESSION (W/H=3)
10° displacement Angle
BEARING
20° displacement Angle
BEARING + TENSION
20 displacement Angle
LONGITUDINAL OR FIBRE STRESSES
TRANSVERSE OR RESIN STRESSES
SHEAR STRESS
FAILURE PROPENSITY
79/4
75/8
80/8
92/8
86/8
84/8


BEARING + COMPRESSION
0° displacement Angle
BEARING
10° displacement Angle
BEARING + TENSILE
10° displacement Angle
70/4
74/4
67/8
75/8
71/8
75/8
LONGITUDINAL OR FIBRE STRESSES
SHEAR STRESS


CARBON
(HM)
BEARING
30° displacement Angle
BEARING + TENSION
(w/d=4)
30° displacement Angle
BEARING
50° displacement Angle
BEARING
70° displacement Angle
LONGITUDINAL
OR
FIBRE
STRESSES
TRANSVERSE
OR
RESIN
STRESSES
SHEAR
STRESS
FAILURE
PROPENSITY

APPENDIX B
STRESS VECTORED RESERVE FACTORS USED IN TSAI-HILL FAILURE CRITERIA [90/±45]s.

Failure Criteria Results

Reserve Factors for individual layers based on the Tsai-Hill failure criterion for a -[90/±45]s. laminate:

1 **Load = +202 MPa in σ_1 direction:**

90° layers:	Tsai-Hill value	= 0.994 (Quadrant 8)
	RF (vect)	= 1.003
	RF (σ_2)	= 1.003
	RF (σ_1)	= 3.984
	RF (σ_3)	= ∞
±45° layers:	Tsai-Hill value	= 0.852 (Quadrant 5)
	RF (vect)	= 1.083
	RF (σ_2)	= 1.22
	RF (σ_1)	= 2.132
	RF (σ_3)	= 1.124

2 **Load – 233 MPa in σ_1 direction:**

90° layers:	Tsai-Hill value	= 0.108 (Quadrant 6)
	RF (vect)	= 3.04
	RF (σ_2)	= 2.83
	RF (A_1)	= 6.85
	RF (A_3)	= ∞
±45° layers:	Tsai-Hill value	= 0.999 (Quadrant 3)
	RF (vect)	= 1.00
	RF (σ_2)	= 1.69
	RF (σ_1)	= 1.002
	RF (σ_3)	= 1.00

3 **Load = +665 MPa in σ_2 direction:**

90° layers:	Tsai-Hill value	= 0.459 (Quadrant 6)
	RF (vect)	= 1.475
	RF (σ_2)	= 9.52
	RF (σ_1)	= 1.445
	RF (σ_3)	= ∞
±45° layers:	Tsai-Hill value	= 0.998 (Quadrant 1)
	RF (vect)	= 1.001
	RF (σ_2)	= 1.032
	RF (σ_1)	= 1.38
	RF (σ_3)	= 1.001

4 **Load = −665 MPa in σ_2 direction:**

90° layers:	Tsai-Hill value	= 0.968 (Quadrant 8)
	RF (vect)	= 1.016
	RF (σ_2)	= 2.016
	RF (σ_1)	= 1.015
	RF (σ_3)	= ∞
±45° layers:	Tsai-Hill value	= 0.986 (Quadrant 7)
	RF (vect)	= 1.001
	RF (σ_2)	= 3.195
	RF (σ_1)	= 1.253
	RF (σ_3)	= 1.0017

APPENDIX C
LISTING OF C PROGRAM FOR CALCULATING STRESS DATA

MODULE FILE LIST-BHOLES.PRJ: List of included files for manual application.

date: 15-03-'98

Author: P.M.Bidgood.

```
/*------MAIN EXEC. FILE-- BHOLES.PRJ-------------------------------------------------------*/
/*****************************************************************************************/


c:\cfiles\bholes\bhol
c:\cfiles\bholes\ccalc
c:\cfiles\bholes\inp
c:\cfiles\bholes\matman

/****************************************************************************************/
```

MODULE BHOL.C - OUTPUTS: Stress field data generation.
Resultant pin loads.

PLANNED: Strain info.

date: 15-03-'88

Author: P.M.Bidgood.

```
#include <c:\cfiles\bholes\decln.h>

void  main()
{

/*-STRUCTURE DEFINITION.                                                    */
  ----------------------

typedef struct              { double            re,im;        }complex;

typedef struct stress_node {double              grid_value;
                            struct stress_node  *next;        }stress_value;

/***************************************************************************/

/*-DECLARATIONS.                                                            */
  --------------

double                      *newdbl(void);
complex                     *newno(void);
stress_value                *newpoint(void);
void                        freeno(complex *comvarble);

complex                     *subt(complex *alpha,complex *beta);
complex                     *divc(complex *alpha,complex *beta);
complex                     *powc(complex *alpha,double power);
complex                      *add(complex *alpha,complex *beta);
complex                      *mul(complex *alpha,complex *beta);
complex                      *sqt(complex *alpha);
complex                     *skrt(complex *alpha);
complex                     *kube(complex *alpha);

void                        initialise2(double matrix2d[50][50]);
void                        initialise1(double vector1d[50] );
void                        solve(double A[][50],double Ai[],double An[],int NN);
void                        tran(complex *alpha,complex *beta);

double                      dinple(double dtrnsfdir);
double                      dinplf(double dtrnsfdif);
int                         iinp(int trnsfi);

const                       double pi=3.141592654;

/***************************************************************************/
```

```
/* VARIABLE DEFINITION.
   --------------------                                                  */
        int             NN,          MM,                flagA,      sign3,      sign4;
        int             k,           i,         j,      flagB,      sign,       sign2;
        int             l,           m,         n;

    double              phi;
    double              x,                      z;
    double              angty,                  dircty;

    double   Antheta[50][50],        k4,        k8,     sum,        Rx,         Delta;
    double       Aitheta[50],        k5,        k9,     nmn,        Ry,         po;
    double           An[50],         k6,        k10,    theta;
    double                           k7,        k11;

    double           Angle,      radius,    SigmaX,     SigmaXY;
    double               a,           b,    SigmaY,     modulus;

    double        Sigxtens,  Sigxcomp,  Sigytens,   Sigycomp,       Sigxy;
    double            Sigx,      Sigy,     width;

   complex            *S11,    *SPHI11,       *c1,      *s[2],      *phisin,      *one,    *two;
   complex            *S22,    *SPHI22,       *c2,   *sphi[2],      *phicos,   *negone,    *trr;
   complex            *S12,    *SPHI12,       *c3, *scjphi[2],                             *temp;
   complex            *S16,    *SPHI16,       *c6,         *A,          *RX;
   complex            *S26,    *SPHI26,    *Ak[2],        *RY,                       *ii;
   complex                                  *B[2],       *RYY;

   stress_value   *SigmaXX, *SigmaYY,*SigmaXXY,         *Tsai,    *firstXX,  *firstYY;
   stress_value       *new,                                    *firstTsai, *firstXXY;

   complex         *xcoord,    *Zk[2],    *sumc1,      *sumc,          *mc,       *nc;
   complex         *ycoord,  *Zeta[2],    *sumc2,      *nmnc,       *value,      *Anc;
   complex        *PHInk[2], *Root[2];


    FILE               *fp,         *fs,        *ft,        *fu,  *fopen();
    char            *data4,     *data5,     *data6,      *data7;

/*************************************************************************************/
```

```
/* -COMPLEX VARIABLE- MEM. ALLOC.                                         */
   ------------------------------

      S11 = newno();    SPHI11 = newno();      s[0] = newno();    sphi[0] = newno();
      S22 = newno();    SPHI22 = newno();      s[1] = newno();    sphi[1] = newno();
      S12 = newno();    SPHI12 = newno(); scjphi[0] = newno();  scjphi[1] = newno();
      S16 = newno();    SPHI16 = newno();         A = newno();
      S26 = newno();    SPHI26 = newno();     Ak[0] = newno();      Ak[1] = newno();
                                               B[0] = newno();       B[1] = newno();
       c1 = newno();        RX = newno();
       c2 = newno();        RY = newno();
       c3 = newno();       RYY = newno();
       c6 = newno();

      one = newno();    negone = newno();    phisin = newno();
      two = newno();        ii = newno();    phicos = newno();

    xcoord = newno();   Zk[0] = newno();     Zk[1] = newno();    sumc1 = newno();
    ycoord = newno(); Zeta[0] = newno();   Zeta[1] = newno();    sumc2 = newno();

        mc = newno();       nc = newno();       Anc = newno();     nmnc = newno();

   PHIok[0] = newno();  Root[0] = newno();     sumc = newno();
   PHIok[1] = newno();  Root[1] = newno();    value = newno();

/******************************************************************************/

/*-VARIABLE INITIALISATION.
   ------------------------
      [See decln. proc. for newno().]                                     */

         phi = 0;            width = 0;         flagA = 0;          angty = 0;  MM = 0;
                                                flagB = 0;          dircty = 0; NN = 0;

      one->re = 1;        two->re =  2;       ii->im = 1;      negone->im =  0;
      one->im = 0;        two->im =  0;       ii->re = 0;      negone->re = -1;

      Sigxtens = 0;       Sigycomp = 0;     Sigxy    = 0;
      Sigytens = 0;       Sigxcomp = 0;

       SigmaXX = 0;        SigmaYY = 0;     SigmaXXY = 0;        Tsai    = 0;

initialise2(Antheta);
initialise1(Altheta);
     initialise1(An);

/******************************************************************************/
```

```
/*-FILE DESIGNATION AND PURGE.                                                            */
  ---------------------------

            data4 = "a:\\data4.hol";
            data5 = "a:\\data5.hol";
            data6 = "a:\\data6.hol";
            data7 = "a:\\data7.hol";

            fp = fopen(data4,"w");
            fclose(fp);
            fs = fopen(data5,"w");
            fclose(fs);
            ft = fopen(data6,"w");
            fclose(ft);
            fu = fopen(data7,"w");
            fclose(fu);

/*************************************************************************************/

/*-READ IN INPUT VARIABLES.                                                               */
  ------------------------

printf("MATERIAL AND BEARING STRESS DATA:\n");
printf("__________________________________\n");
printf("\n            Input angularity:");                      angty   = dinplf(angty);
printf("        Input directionality:");                        dircty  = dinplf(dircty);
printf("\n        Input displmnt. angle:");                     phi     = dinplf(phi);
printf("\n        Input compliance S11 :");                     S11->re = dinple(S11->re);
printf("        Input compliance S22 :");                       S22->re = dinple(S22->re);
printf("        Input compliance S12 :");                       S12->re = dinple(S12->re);
printf("        Input compliance S16 :");                       S16->re = dinple(S16->re);
printf("        Input compliance S26 :");                       S26->re = dinple(S26->re);
printf("\nINPUT  No. of  points on hole edge  required :");     NN   = iinp(NN);
printf("\nINPUT Max. No. of terms reqd. in sine series :");     MM   = iinp(MM);
printf("\nINPUT Material strength in longitudinal tension:");Sigytens = dinple(Sigytens);
printf("INPUT Material strength in longitudinal comprsn:");  Sigycomp = dinple(Sigycomp);
printf("INPUT Material strength in transverse   tension:");  Sigxtens = dinple(Sigxtens);
printf("INPUT Material strength in transverse   comprsn:");  Sigxcomp = dinple(Sigxcomp);
printf("INPUT Material strength in   In-plane     shear:");     Sigxy = dinple(Sigxy);

printf("\nBYPASS LOADS AND FINITE ELEMENT MODIFICATION INPUTS:\n");
printf("____________________________________________________\n");
printf("\n    Input estimated effective width (w/d):--");       width = dinplf(width);
printf("  Input (1) for pure      tensile load\n");
printf("      or (2) for pure compressive load:-------");       flagA = iinp(flagA);
printf("\nInput (1) to include the bearing stress:\n");
printf("Input (2) to calculate without bearing :-----");        flagB = iinp(flagB);
printf("\nPlease be patient - I am working on the solution.!!\n\n\n");

phi=-1*phi*pi/180;

/*************************************************************************************/
```

```
/*-CALCULATE s[1] & s[2].
   -----------------------                                                          */

  if (dircty>angty)
                  { s[1]->re = sqrt((dircty-angty)/2);
                    s[1]->im = sqrt((dircty+angty)/2);
                    s[0]->re = -1 * s[1]->re;
                    s[0]->im = s[1]->im;}
            else
                  { s[1]->re = 0;
                    s[1]->im = sqrt((angty-dircty)/2) + sqrt((angty+dircty)/2);
                    s[0]->re = 0;
                    s[0]->im = sqrt((angty+dircty)/2) - sqrt((angty-dircty)/2);}


/*-ROTATION FUNCTIONS.
   ------------------                                                               */

          phisin->re = sin(phi) ; phisin->im = 0;
          phicos->re = cos(phi) ; phicos->im = 0;

/*-ROTATE s[i] TO GIVE sphi[i].
   ----------------------------                                                     */
trr = divc((subt(mul(s[0],phicos),phisin)),(add(mul(s[0],phisin),phicos))); tran(trr,sphi[0]);
trr = divc((subt(mul(s[1],phicos),phisin)),(add(mul(s[1],phisin),phicos))); tran(trr,sphi[1]);



/*-ROTATE sij TO GIVE sphiij.
   --------------------------                                                       */

trr     = mul(mul((subt((mul(skrt(s[0]),skrt(phisin))),(skrt(phicos)))),
             (subt((mul(skrt(s[1]),skrt(phisin))),(skrt(phicos))))),S11);       tran(trr,SPHI11);
trr     = mul((mul((subt((mul((skrt(s[0])),(skrt(phicos)))),(skrt(phisin)))),
             (subt((mul((skrt(s[1])),(skrt(phicos)))),(skrt(phisin)))))),S11); tran(trr,SPHI22);
trr     = add(mul(mul(mul(mul(add(one,skrt(s[0])),add(one,skrt(s[1]))),
                                   S11),skrt(phisin)),skrt(phicos)),S12);       tran(trr,SPHI12);
  temp = mul(mul(add(add(mul(mul(skrt(s[0]),skrt(s[1])),two),
                 skrt(s[0])),skrt(s[1])),kube(phisin)),phicos);
trr     = mul(subt(temp,mul(mul(add(add(
           two,skrt(s[0])),skrt(s[1])),kube(phicos)),phisin)),S11);            tran(trr,SPHI16);
  temp = mul(mul(add(add(mul(mul(skrt(s[0]),skrt(s[1])),two),skrt(s[0])),
                                         skrt(s[1])),kube(phicos)),phisin);
trr     = mul(subt(temp,mul(mul(add(add(two,skrt(s[0])),skrt(s[1])),
                                         kube(phisin)),phicos)),S11);          tran(trr,SPHI26);


/*-CALCULATION C1 TO C7.
   ---------------------                                                            */

  temp = mul(add(sphi[0],sphi[1]),SPHI11);
trr     = subt(temp,SPHI16);                                                    tran(trr,c1);
  temp = mul(mul(sphi[0],sphi[1]),SPHI11);
trr     = mul(subt(temp,SPHI12),ii);                                            tran(trr,c2);
  temp = divc(mul(add(sphi[0],sphi[1]),SPHI22),mul(sphi[0],sphi[1]));
trr     = subt(SPHI26,temp);                                                    tran(trr,c3);

                k4  =  +(c2->re);
                k5  =  +(c1->im);
                k6  =  -(c2->im);
                k7  =  +(c3->im);
```

```
for(i=2; i<=(NN+1); ++i){                theta = 0.191986218 + (2.757620218/(NN-1))*(i-2);

                                         k8  =  k5*cos(theta)-k4*sin(theta);
                                         k9  =  k4*cos(theta)-k7*sin(theta);
                                         k10 =  k8*cos(theta)-k9*sin(theta);
                                         k11 = -k8*sin(theta)-k9*cos(theta);

                                         sign2=1;
                                         sign3=1;
                                         sign4=1;

                     sum  =  0;
                     sign =  1;
                     j    = -1;
                              do { j+=2;
                                   x =j;
                                   if (sign== 1)  sign=-1;  else  sign=1;
                                   sum += (((2*k9*sin(x*theta))-(k8*x*cos(x*theta))
                                          -(2*k7*sin(theta)*sign))/((x*x)*(x-2)*(x+2)));
                                 }while (j<MM);

                              Altheta[i-2] = -j* ((k8*sin(2*theta)/2)
                                                 +(k9*cos(2*theta)/2)
                                                 +(k6*(theta-pi/2)*cos(theta))
                                                 -(k7*sin(theta)/2)
                                                 +(8*sum/pi));

 for (k=2; k<=NN+1; ++k){z=k;
                         if (sign2== 1) sign2=-1; else sign2=1)

                          if (sign2 == -1) {sum = 0;
                                              j = 0;
                                           sign = 1;
                                                   do {j+=2;
                                                       x=j;
                                                      if (sign== 1) sign=-1; else sign=1;
                                                      nmn = x*((x+1)*(x+1)-'z*z))
                                                            *((x-1)*(x-1)-(z*z));
                                                      sum += (((2*k9*x*z*sin(x*theta))
                                                            -(k8*z*(x*x-z*z+1)
                                                            *cos(x*theta))-k4*z
                                                            *(x*x-z*z+1)*sign
                                                            *sin(theta))/nmn);
                                                      }while (j<MM);

                                           if (sign3== 1) sign3=-1; else sign3= 1;
                                           Antheta[i-2][k-2] = (2/(z*z-1))
                                                            *((k10*z*sin(z*theta))
                                                            +(k11*cos(z*theta))
                                                            +(2*k6*sin(z*theta))
                                                            -(k6*z/pi)
                                                            *((4*cos(theta)/(z*z-1))
                                                            -(2*theta-pi)*sin(theta))
                                                            -(k4*sign3*sin(theta)))
                                                            +(8*sum/pi);
                                           }/*end if*/
```

```
else            {sum  = 0;
                 j    =-1;
                 sign = 1;
                        do {j+=2;
                            x =j;
                            if (sign== 1) sign=-1; else sign=1;
                            nmn = x*((x+1'*(x+1)-(z*z))
                                   *((x-1)*(x-1)-(z*z));
                            sum += (((2*k9*z*x*sin(x*theta))
                                   -(kB*z*(x*x-z*z+1)
                                   *cos(x*theta))
                                   -(2*x*z*k7*sign
                                   *sin(theta)))/nmn);
                            }while (j<MM);

                 if (sign4== 1) sign4=-1; else sign4=1;
                 Antheta[i-2][k-2] = (2/(z*z-1))
                                   *(k10*z*sin(z*theta)
                                   +(k11*cos(z*theta))
                                   +(2*k6*sin(z*theta))
                                   -sin(theta)*((2*k6*sign4)
                                   +(k7*z*sign4)))
                                   +(8*sum/pi);
                 }/*end else*/

   }/* of k loop */
}/* of i loop */


/*  SOLVE FOR An TERMS .
    ------------------------                                             */

solve(Antheta,Altheta,An,NN);
                                                 printf("\nAn[1]=+1.000000e-000\n");
                                                 for (i=0; i<NN; ++i)
                                                     printf("\nAn[%d]=%+e\n",i+2,An[i]);

/*******************************************************************************/
/**//*  CALCULATION OF Rx & Ry  RESULTANT FORCES.
        ****************************************                         *//**/

          printf("\nBASIC LOAD RESULTANTS:");
          printf("\n************************\n");

          po = 2/pi;sum=0;for (i=0; i<=(NN-3); i+=2){x=i; sum +=(An[i]+An[i+2])/(x+3) ;}
          Rx = po*(An[0]+sum);Ry = po*pi/2;printf("\nF · = %e   Rx = %e\n",Ry,Rx);
          Delta = atan(Rx/Ry)*180/pi;printf("\nDelta · %e degrees.\n",Delta);

          printf("\n                          ******************\n");

/*******************************************************************************/
```

```
/*-CALCULATION OF STRESS VALUES.
   ----------------------------                                          */

printf("BUSY CALCULATING STRESS FIELD :\n\n");


/*-CALCULATION OF Ak.
   ---------------                                                       */

  c6->re = k6; c6->im = 0;
  temp = divc(c6, mul(mul(mul(skrt(add(s[0],s[1])),s[0]),s[1]),skrt(S1)));
  A->re = temp->re/4;
  A->im = temp->im/4;

RX->re = Rx;  RX->im = 0;  scjphi[0]->re = sphi[0]->re;  scjphi[0]->im = sphi[0]->im * -1;
RY->re = Ry;  RY->im = 0;  scjphi[1]->re = sphi[1]->re;  scjphi[1]->im = sphi[1]->im * -1;

for (i=0; i<=1; ++i){   j = 1-i;

                        temp  = mul(mul(add(mul(RY,sphi[i]),RX),A),SPHI1);
                        temp  = mul( mul(subt(scjphi[j],sphi[j]),
                                 subt(scjphi[i],sphi[j])),temp);
                        temp  = add(temp,divc(add(mul(sphi[j],RY),RX),two));
                        temp  = divc(temp,mul(two,mul(ii,subt(sphi[j],sphi[i]))));
                        Ak[i]->re = temp->re / pi;
                        Ak[i]->im = temp->im / pi;
                      }


/* CALCULATION OF STRESS FIELD - data.
   -------------------------------                                       */

for (j=0; j<=36; ++j){  z = j;                   /* Angle variant  - j x 5 deg steps...*/
                                                    printf("\n Calculating at angle = %d",j*5);
                                                    printf("\n ***************************\n");
                                                    printf("\n & Radius : ");

  for (i=4; i<=4 ;++i){ x = i;                   /* Radius variant - i / 4 (r) steps...*/
                        SigmaX  = 0;
                        SigmaY  = 0;
                        SigmaXY = 0;

                        radius = x/4;
                        Angle  = z*5 *pi/180;

                        xcoord->re = radius * cos(Angle); xcoord->im = 0;
                        ycoord->re = radius * sin(Angle); ycoord->im = 0;


   /* Calculation of PHI(Zk) and PHI'(Zk).
      ------------------------------------                               */

   for (k=0; k<=1; ++k){ l=1-k;
                         temp = add(xcoord,mul(ycoord,sphi[k]));     tran(temp,Zk[k]);
                         temp = sqt(subt(subt(skrt(Zk[k]),
                                skrt(sphi[k])),one));                tran(temp,Root[k]);
```

```
/*Choice of correct root sign.
  ----------------------------                                          */

if (xcoord->re==0){ if (Root[k]->im>0) { if (ycoord>0) tran(temp,Root[k]);
                                         if (ycoord<0) {Root[k]->re = temp->re*-1;
                                                        Root[k]->im = temp->im*-1;}}
                    if (Root[k]->im<0) { if (ycoord<0) tran(temp,Root[k]);
                                          if (ycoord>0) {Root[k]->re = temp->re*-1;
                                                         Root[k]->im = temp->im*-1;}}}

if (xcoord->re<0){  if (Root[k]->re<0) { tran(temp,Root[k]);}
                    if (Root[k]->re>0) {Root[k]->re = temp->re*-1;
                                        Root[k]->im = temp->im*-1;}}

if (xcoord->re>0){  if (Root[k]->re>0) {  tran(temp,Root[k]);}
                    if (Root[k]->re<0) {Root[k]->re = temp->re*-1;
                                        Root[k]->im = temp->im*-1;}}


temp = divc(add(Zk[k],Root[k]),
       subt(one,mul(ii,sphi[k])));  tran(temp,Zeta[k]);

modulus=sqrt(Zeta[k]->re*Zeta[k]->re+Zeta[k]->im*Zeta[k]->im);

if ((j*5)>=90 || i!=4) {if (modulus<1){temp =divc(subt(Zk[k],Root[k]),
                                       subt(one,mul(ii,sphi[k])));tran(temp,Zeta[k]);
                                 Root[k]->re=-1*Root[k]->re;
                                 Root[k]->im=-1*Root[k]->im;}}


   /*Calcn. of PHIo'k_Zeta[k] in eqn. B.2
     ------------------------------------                               */

     m=-1; sumc1->re=0; sumc2->re=0; sumc1->im=0; sumc2->im=0;

     do { m+=2; n=-1;
     do { n+=2;

          mc->re = m ;
          nc->re = n ;
          mc->im = 0 ;
          nc->im = 0 ;
          if ((n-2)>=0) {Anc->re = An[n-2];
                         Anc->im = 0;       }
          if ((n-2)==-1) {Anc->re=1;
                          Anc->im=0;        }


          a = m ;
          b = n ;
          nmnc->re =  ((a+1)*(a+1)-b*b) * ((a-1)*(a-1)-b*b);
          nmnc->im = 0;
          temp = divc(divc(mul(mul(mul( add(subt(add(mul(mul(mul(two,ii),sphi[k]])
                 ,mc),one),skrt(nc)),skrt(mc)),two),nc),Anc),powc(Zeta[k],a)),nmnc);
          temp = add(sumc1,temp);
                 tran(temp,sumc1);

     }while (n<NN-1);
     }while (m<MM-2);
```

```
m=0;
      do { m+=2; n=0;
      do { n+=2;
            mc->re = m;
            nc->re = n;
            mc->im = 0;
            nc->im = 0;
            Anc->re = An[n-2];
            Anc->im = 0       ;

            a = m;
            b = n;

            nmnc->re =  ((a+1)*(a+1)-b*b) * ((a-1)*(a-1)-b*b);
            temp = divc(divc(mul(mul(mul( add(subt(add(mul(mul(mul(two,ii),sphi[1])
                   ,mc),one),skrt(nc)),skrt(mc)),two),nc),Anc),powc(Zeta[k],a)),nmnc);
            temp = add(sumc2,temp);
            tran(temp,sumc2);

      }while (n<NN-1);
      }while (m<MM-1);


      sumc->re=0;
      sumc->im=0;
      n=-1;
      do{ n+=1;
            if (n==1) {value->re=1;
                       value->im=0;       }
            else      {value->re=An[n-2];
                       value->im=0;       }
            if (n==0) {value->re=0;
                       value->im=0;       }

            nc->re=n;
            nc->im=0;
            Anc->re = An[n];
            Anc->im = 0     ;

            b=n;
            temp = divc(add(mul(Anc,subt(one,mul(sphi[1],ii))),
                   mul(add(one,mul(sphi[1],ii)),value)),powc(Zeta[k],(1+b)));
            temp = add(sumc,temp);
             tran(temp,sumc);
        } while (n<NN-1);

    temp = mul(sumc,ii);
    tran(temp,sumc);
    sumc->re = sumc->re * pi / 4 ;
    sumc->im = sumc->im * pi / 4 ;

    temp = divc(divc(subt(add(sumc1,sumc2),sumc),subt(sphi[k],sphi[1])),ii);

    PHIok[k]->re = temp->re * po / (2 * pi);
    PHIok[k]->im = temp->im * po / (2 * pi);

    temp = divc(add(PHIok[k],Ak[k]),Root[k]);
    tran(temp,PHIok[k]);

} /*end of k loop for calculation of phi-dashed, k=1 & 2. */
```

```
/*Stress calc. and superposition.
  ------------------------------                                      */

  /*Pure Comp. or Tens. with width.
    -------------------------------                                   */

  if (width!=0){ for (k=0;k<=1;++k) {l = 1-k;
                                      RYY->re  = RY->re/(2*width);
                                      RYY->im = 0;
                                      if(flagA==2) RYY->re = -1*RYY->re;
                                      temp = mul(mul(subt(sphi[k],sphi[l]),
                                             subt(sphi[k],ii)),two);
                                      temp = divc(mul(mul(sphi[l],RYY),ii),temp);
                                      tran(temp,B[k]);
                                      if(RYY->re<0)RYY->re=-1*RYY->re;
                                     }

                 for (k=0;k<=1;++k) {temp   = mul(mul(subt(divc(Zk[k],Root[k]),
                                                       one),B[k]),skrt(sphi[k]));
                                      SigmaX = SigmaX + 2*temp->re;
                                      temp   = mul(subt(divc(Zk[k],Root[k]),
                                                            one),B[k]);
                                      SigmaY = SigmaY + 2*temp->re;
                                      temp   = mul(mul(subt(divc(Zk[k],Root[k]),
                                                         one),B[k]),sphi[k]);
                                      SigmaXY= SigmaXY+-2*temp->re;
                                      }

                 SigmaY = (RYY->re) + SigmaY;
                }/*end if*/

 /* Calculation of infinite bearing. (opt).
    -------------------------------------                            */

  if (flagB==1) {temp = add(mul(PHIok[0],skrt(sphi[0])),mul(PHIok[1],skrt(sphi[1])));
                  SigmaX = SigmaX +temp->re*2;
                  temp = add(PHIok[0],PHIok[1]);
                  SigmaY = SigmaY +temp->re*2;
                  temp = add(mul(PHIok[0],sphi[0]),mul(PHIok[1],sphi[1]));
                  SigmaXY= SigmaXY +temp->re*-2;
                 }

/*Linked list data storage.
  ************************                                           */

 if (SigmaXX==0) {SigmaXX = newpoint();
                   firstXX = SigmaXX;
                   SigmaXX->grid_value = SigmaX*cos(phi)*cos(phi)
                                       + SigmaY*sin(phi)*sin(phi)
                                       - 2*cos(phi)*sin(phi)*SigmaXY;
                  }
           else  {new = newpoint();
                   SigmaXX->next = new;
                   SigmaXX = new;
                   SigmaXX->grid_value = SigmaX*cos(phi)*cos(phi)
                                       + SigmaY*sin(phi)*sin(phi)
                                       - 2*cos(phi)*sin(phi)*SigmaXY;
                   SigmaXX->next=0;
                  }
```

```
                     firstYY = SigmaYY;
                     SigmaYY->grid_value =  SigmaX*sin(phi)*sin(phi)
                                          + SigmaY*cos(phi)*cos(phi)
                                          + 2*cos(phi)*sin(phi)*SigmaXY;
                     }
              else   {new = newpoint();
                     SigmaYY->next = new;
                     SigmaYY = new;
                     SigmaYY->grid_value =  SigmaX*sin(phi)*sin(phi)
                                          + SigmaY*cos(phi)*cos(phi)
                                          + 2*cos(phi)*sin(phi)*SigmaXY;

                     SigmaYY->next=0;
                     }

      if (SigmaXXY==0) {SigmaXXY = newpoint();
                     firstXXY = SigmaXXY;
                     SigmaXXY->grid_value =  SigmaX*cos(phi)*sin(phi)
                                           - SigmaY*cos(phi)*sin(phi)
                                           + SigmaXY*(cos(phi)*cos(phi)
                                           - sin(phi)*sin(phi));
                     }
              else {new = newpoint();
                     SigmaXXY->next = new;
                     SigmaXXY = new;
                     SigmaXXY->grid_value =  SigmaX*cos(phi)*sin(phi)
                                           - SigmaY*cos(phi)*sin(phi)
                                           + SigmaXY*(cos(phi)*cos(phi)
                                           - sin(phi)*sin(phi));

                     SigmaXXY->next = 0;
                     }

      if (Tsai==0)      {Tsai = newpoint();
                     firstTsai = Tsai;
                     if (SigmaYY->grid_value<0) Sigy = Sigycomp;
                     else Sigy = Sigytens;
                     if (SigmaXX->grid_value<0) Sigx = Sigxcomp;
                     else Sigx = Sigxtens;
                     Tsai->grid_value = (SigmaYY->grid_value*SigmaYY->grid_value)
                                /(Sigy*Sigy)+(SigmaXX->grid_value*SigmaXX->grid_value)
                                /(Sigx*Sigx)+(SigmaXXY->grid_value*SigmaXXY->grid_value)
                                /(Sigxy*Sigxy)-(SigmaXX->grid_value*SigmaYY->grid_value)
                                /(Sigy*Sigx);
                     }
              else  {new = newpoint();
                     Tsai->next = new;
                     Tsai = new;
                     if (SigmaYY->grid_value<0) Sigy = Sigycomp;
                     else Sigy = Sigytens;
                     if (SigmaXX->grid_value<0) Sigx = Sigxcomp;
                     else Sigx = Sigxtens;
                     Tsai->grid_value = (SigmaYY->grid_value*SigmaYY->grid_value)
                                /(Sigy*Sigy)+(SigmaXX->grid_value*SigmaXX->grid_value)
                                /(Sigx*Sigx)+(SigmaXXY->grid_value*SigmaXXY->grid_value)
                                /(Sigxy*Sigxy)-(SigmaXX->grid_value*SigmaYY->grid_value)
                                /(Sigy*Sigx);
                     Tsai->next = 0;
                     }

   printf("%g, ",radius);

   }     /* End of i-loop (radius variant)*/      printf("\n *-*");
   }     /* End of j-loop (angle  variant)*/
```

```
/*******************************************************************************************/

/*STORAGE OF DATA TO DISC.                                                               */

        SigmaXX = firstXX;
                    fp = fopen(data4,"a");
                    do { fprintf(fp,"%15le\n",SigmaXX->grid_value);
                         SigmaXX = SigmaXX->next;} while (SigmaXX->next!=0);
                         fprintf(fp,"%15le\n",SigmaXX->grid_value);
                    fclose(fp);

        SigmaYY = firstYY;
                    fs = fopen(data5,"a");
                    do { fprintf(fs,"%15le\n",SigmaYY->grid_value);
                         SigmaYY = SigmaYY->next;} while (SigmaYY->next!=0);
                         fprintf(fs,"%15le\n",SigmaYY->grid_value);
                    fclose(fs);

        SigmaXXY = firstXXY;
                    ft = fopen(data6,"a");
                    do { fprintf(ft,"%15le\n",SigmaXXY->grid_value);
                         SigmaXXY = SigmaXXY->next;} while (SigmaXXY->next!=0);
                         fprintf(ft,"%15le\n",SigmaXXY->grid_value);
                    fclose(ft);

        Tsai = firstTsai;
                    fu = fopen(data7,"a");
                    do { fprintf(fu,"%15le\n",Tsai->grid_value);
                         Tsai = Tsai->next;} while (Tsai->next!=0);
                         fprintf(fu,"%15le\n",Tsai->grid_value);
                    fclose(fu);

printf("\n***** END OF MODULE. *****\n");

}/* END OF MODULE BEARING_HOLES. */

/*******************************************************************************************/
/*******************************************************************************************/
```

MODULE CCALC.C: Complex number routines for bhol.c.

date: 15 [illegible] 88.

Author: P.M.Bidgood.

```
/*--------------------------complex functions---------------------------*/
/*************************************************************************/

#include <math.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>

typedef struct
               {double re;
                double im;
                        }complex;

typedef struct stress_node
               {double grid_value;
                struct stress_node *next;}stress_value;

complex dummy0,dummy1,dummy2,dummy3,dummy4,dummy5,dummy6,dummy7,dummy8,dummy9;
complex dummy11,dummy12,dummy13,dummy14,dummy15,dummy16,dummy20,dummy21,dummy22,dummy23;
complex *init;

  /*-------------------------------------------------------------------*/
  /* Conversion(s) of pointer variable to complex space.*/
  /*----------------------------------------------*/

   complex *newno()
                 { init = ((complex *) malloc(sizeof(complex)));
                              init->re = 0;
                              init->im = 0;
                     return(init); }

   void freeno(complex *comvarble)
                              { free(comvarble); }

   double *newdbl()
                 { return ((double *) malloc(sizeof(double)));}

   stress_value *newpoint()
                 { return ((stress_value *) malloc(sizeof(stress_value)));}


/*--------------------------------------------------------------------*/
/*----------------TRANSFER FUNCTION.-----------------------------------*/

void tran(complex *alpha,complex *beta)

               {beta->re = alpha->re;
                beta->im = alpha->im;}


/*--------------------------------------------------------------------*/
```

```
              complex *subt(complex *alpha,complex *beta)

{  static int flag1;
   if (flag1==0) flag1=1;  else flag1=0;

               if (flag1==0)
              {  dummy1.re = alpha->re - beta->re;
                 dummy1.im = alpha->im - beta->im;
                 return(&dummy1);}

               else
              {  dummy2.re = alpha->re - beta->re;
                 dummy2.im = alpha->im - beta->im;
                 return(&dummy2);}
}
/*----------------------------------------------------------------------------*/
/*-----------------COMPLEX No. ADDITION.--------------------------------------*/

              complex *add(complex *alpha,complex *beta)

{ static int flag2;
  if (flag2==0) flag2=1;  else  flag2=0;

               if (flag2==0)
              {  dummy3.re = alpha->re + beta->re;
                 dummy3.im = alpha->im + beta->im;
                 return(&dummy3);}

               else
              {  dummy4.re = alpha->re + beta->re;
                 dummy4.im = alpha->im + beta->im;
                 return(&dummy4);}
}

/*----------------------------------------------------------------------------*/
/*-----------------COMPLEX No. MULTIPLICATION.--------------------------------*/

              complex *mul(complex *alpha,complex *beta)

{ static int flag3;
  if (flag3==3) flag3=0;  else flag3+=1;

               if (flag3==0)
              { dummy5.re = alpha->re * beta->re - alpha->im * beta->im;
                dummy5.im = alpha->re * beta->im + alpha->im * beta->re;
                return(&dummy5);}

               if (flag3==1)
              { dummy6.re = alpha->re * beta->re - alpha->im * beta->im;
                dummy6.im = alpha->re * beta->im + alpha->im * beta->re;
                return(&dummy6);}

               if (flag3==2)
              { dummy9.re = alpha->re * beta->re - alpha->im * beta->im;
                dummy9.im = alpha->re * beta->im + alpha->im * beta->re;
                return(&dummy9);}

               if (flag3==3)
              { dummy0.re = alpha->re * beta->re - alpha->im * beta->im;
                dummy0.im = alpha->re * beta->im + alpha->im * beta->re;
                return(&dummy0);}
                else
                {printf("Error!!!");return(&dummy0);}
}
/*----------------------------------------------------------------------------*/
```

```
                complex *divc(complex *alpha,complex *beta)

{static int flag4;
 if (flag4==0) flag4=1; else flag4=0;

                  if (flag4==0)
                 { dummy7.re = (alpha->re * beta->re + alpha->im * beta->im)/
                            (beta->re  * beta->re + beta->im  * beta->im);
                   dummy7.im = (alpha->im * beta->re - alpha->re * beta->im)/
                            (beta->re  * beta->re + beta->im  * beta->im);
                   return(&dummy7);}

                  else
                 { dummy8.re = (alpha->re * beta->re + alpha->im * beta->im)/
                            (beta->re  * beta->re + beta->im  * beta->im);
                   dummy8.im = (alpha->im * beta->re - alpha->re * beta->im)/
                            (beta->re  * beta->re + beta->im  * beta->im);
                   return(&dummy8);}

}

/*----------------------------------------------------------------------*/
/*----------------------------------------------------------------------*/

complex *skrt(complex *alpha) /* complex No squared.*/

{ static int flag6;
 if (flag6==3) flag6=0; else flag6+=1;

              if (flag6==0)
             { dummy11.re = (alpha->re)*(alpha->re) - (alpha->im)*(alpha->im);
               dummy11.im = 2*(alpha->re)*(alpha->im);
               return(&dummy11);}

              if (flag6==1)
             { dummy12.re = (alpha->re)*(alph[illegible]n) - (alpha->im)*(alpha->im);
               dummy12.im = 2*(alpha->re)*(al[illegible]m);
               return(&dummy12);}

              if (flag6==2)
             { dummy15.re = (alpha->re)*(alpha->re) - (alpha->im)*(alpha->im);
               dummy15.im = 2*(alpha->re)*(alpha->im);
               return(&dummy15);}

              if (flag6==3)
             { dummy16.re = (alpha->re)*(alpha->re) - (alpha->im)*(alpha->im);
               dummy16.im = 2*(alpha->re)*(alpha->im);
               return(&dummy16);}
               else {printf("ERROR(SKRT)!!");return(&dummy11);}
}
/*----------------------------------------------------------------------*/
```

```
complex *kube(complex *alpha)

{ static int flag7;
 if (flag7==0) flag7=1; else flag7=0;

              if (flag7==0)
             { dummy13.re =   alpha->re*(alpha->re*alpha->re - 3*alpha->im*alpha->im);
               dummy13.im =   alpha->im*(3*alpha->re*alpha->re - alpha->im*alpha->im);
               return(&dummy13);}

              else
             { dummy14.re =   alpha->re*(alpha->re*alpha->re - 3*alpha->im*alpha->im);
               dummy14.im =   alpha->im*(3*alpha->re*alpha->re - alpha->im*alpha->im);
               return(&dummy14);}

}
/*---------------------------------------------------------------------------------------*/

complex *sqt(complex *alpha)

{ static int flag8;
  if (flag8==0) flag8=1; else flag8=0;

              if (flag8==0)
             { if (alpha->re==0) dummy20.re=0;
               else dummy20.re = sqrt((alpha->re+sqrt(alpha->re*alpha->re
                                          +alpha->im*alpha->im))/2);
               if (alpha->im==0) dummy20.im=0;
               else dummy20.im = sqrt((sqrt(alpha->re*alpha->re
                                   +alpha->im*alpha->im)-alpha->re)/2);
               if (alpha->im<=0) dummy20.im = -1*dummy20.im;
               return(&dummy20);}

              else
             { if (alpha->re==0) dummy21.re=0;
               else dummy21.re = sqrt((alpha->re+sqrt(alpha->re*alpha->re
                                          +alpha->im*alpha->im))/2);
               if (alpha->im==0) dummy21.im=0;
               else dummy21.im = sqrt((sqrt(alpha->re*alpha->re
                                   +alpha->im*alpha->im)-alpha->re)/2);
               if (alpha->im<=0) dummy21.im = -1*dummy21.im;
               return(&dummy21);}
}
/*---------------------------------------------------------------------------------------
```

```
complex *cpow(complex *alpha,double power)

{ static int flag9;
    double phi,mod;

  if (flag9==0) flag9=1; else flag9=0;

               if (flag9==0)
              {phi = power*(atan2(alpha->im,alpha->re));
               mod = pow( (sqrt(alpha->re*alpha->re+alpha->im*alpha->im)),power);

             dummy22.re = mod * cos(phi);
             dummy22.im = mod * sin(phi);

             return(&dummy22);}

               else
              { phi = power*(atan2(alpha->im,alpha->re));
                mod = pow( (sqrt(alpha->re*alpha->re+alpha->im*alpha->im)),power);

             dummy23.re = mod * cos(phi);
             dummy23.im = mod * sin(phi);

              return(&dummy23);}

}

/*-----------------------------------------------------------------------------*/
```

MODULE INP.C: Input format for manual use of module bhoi.c.

date: 15-03-'88

Author: P.M.Bidgood.

```
/*  THE FOLLOWING IS FOR INPUT WHEN DIRECT MODULE APPLICATION IS REQUIRED.*/
/*--------------------------------------------------------------------------*/
/****************************************************************************/

#include <stdio.h>
#include <stdarg.h>

/*-------------------------------------------------------------------------------*/

double dinple(double dtrnsfdle)
                        {
                           scanf(" %le",&dtrnsfdle);
                           return(dtrnsfdle);         }

double  dinplf(double dtrnsfdlf)
                        {
                           scanf(" %lf",&dtrnsfdlf);
                           return(dtrnsfdlf);         }

int iinp(int trnsfi)
                        {
                          scanf("%d",&trnsfi);
                          return(trnsfi);        }

/*********************************************************************************/
```

MODULE MATMAN.C: Matrix manipulation file for bhol.c.

date: 15-03-'88.

Author: P.M.Bidgood.

```
/*--------------------------MATRIX MANIPULATION FILE--------------------------*/
/******************************************************************************/
#include <math.h>

/*--INITIALISATION ROUTINES-*/

void initialise2(double matrix2d[50][50])
{                  int i,j;
                   for (i=0; i<=50; ++i) {
                    for (j=0; j<=50; ++j) matrix2d[i][j] = 0; }
}
void initialise1(double vector1d[50])
{                  int i;
                   for (i=0; i<=50; ++i) vector1d[i] = 0;
}
/*--SOLUTION OF EQUATIONS.-*/
void solve(double A[][50],double A1[];double An[],int NN)
{           int    i, j, k, imx;
            double    L, sum,  AInv[50][50];

sum = 0;
for  (j=0; j<NN; ++j)         {
 for (i=0; i<NN; ++i)         { AInv[i][j] = 0; }
                                AInv[j][j] = 1; }

for  (i=0; i<NN; ++i)         {L = A[i][0];
 for (j=1;j<NN;++j)           {if(fabs(A[i][j])>fabs(L))L=A[i][j]; }
 for (j=0; j<NN; ++j)         {A[i][j] = A[i][j]/fabs(L);
                               AInv[i][j] = AInv[i][j]/fabs(L); } }

for  (k=0;k<(NN-1);++k)       {imx = k;
                               L = A[k][k];
 for (i=(k+1);i<NN;++i)       {if (fabs(A[i][k])>fabs(L))
                                 { L = A[i][k];
                                   imx = i;      } }
 for (j=0; j<NN; ++j)         {L = A[k][j];
                               A[k][j] = A[imx][j];
                               A[imx][j] = L;
                               L = AInv[k][j];
                               AInv[k][j] = AInv[imx][j];
                               AInv[imx][j] = L;          }
  for ((i=(k+1)); i<NN; ++i)  {L = A[i][k]/A[k][k];
   for (j=0; j<NN; ++j)       {A[i][j] = A[i][j]-L*A[k][j];
                               AInv[i][j] = AInv[i][j] - L*AInv[k][j];} } }

for (k=(NN-1);k>=1;--k)       {
 for (i=0;i<=(k-1);++i)       {L = A[i][k]/A[k][k];
  for (j=0;j<=(NN-1);++j)     {A[i][j] = A[i][j]-L*A[k][j];
                               AInv[i][j] = AInv[i][j]-L*AInv[k][j]; } } }

for (i=0;i<=(NN-1);++i)       {L = A[i][i];
 for (j=0;j<=(NN-1);++j)      {AInv[i][j] = AInv[i][j]/L; } }

for (i=0;i<=(NN-1);++i)       {sum = 0;
 for (j=0;j<=(NN-1);++j)      {sum += AInv[i][j]*A1[j]; }
                               An[i]= sum;              }

 }
```

APPENDIX D
OVERVIEW OF PASCAL INTEGRATED PROGRAM

88

Comphole

February 25th, 1988

Mihaly Zsadanyi

Contents

INTRODUCTION

This report deals with the computer package designed for generating data for pin-loaded holes, and for obtaining a graphical display of the results. The package is designed to run on the APOLLO system, and is currently in the possession of P. Bidgood. The package was designed by Mihaly Zsadanyi, during a period of vacation work in January and February 1988.

PROGRAM SECTIONS

The package consists of 4 sections :-

1. the main program - PROGRAM Comphole
2. MODULE Bhol
3. MODULE Ccalc
4. MODULE Solve

We shall discuss each of the above individually.

MODULE BHOL

This module is used to calculate the stress fields. The data produced by this module can then be plotted by using the main program Comphole. Bhol is a direct translation of the C program Bhol.C, written by P. Bidgood. Module Bhol makes use of modules Ccalc and Solve (to be discussed later).

Module Bhol consists of the *following parts* :-

1. Initialisation

   Here, certain variables, as well as the matrices and vectors are initialise. Certain complex numbers, such as (0,1) and (1,0) are also defined.

2. Read-in

Here, the input parameters are read, either from the keyboard or from a user defined file. Note that if a file is used, the full pathname of the file must be given.

NB - ALL ANGLES ARE INPUT IN DEGREES
These are converted by the program into radians in the traditional way. The name of the output file is also to be read in. For a full discussion of input and output files, see later. All the output files are then opened.

3. Calculation of Antheta and Altheta.

First, the complex functions and compliances are rotated, according to the values of the variables 'dircty' and 'angty'. Then, constants c1-c3 and k4-k7 are calculated. PROCEDURE Calculate_Coefficients is then called, and here the vector Altheta and the matrix Antheta are calculated.
Note - the entries of Altheta are numbered from 0 to NN-1, while the NN x NN matrix Antheta has rows and columns numbered from 0 to NN-1 as well.

4. Solve the system of linear equations.

This is done by module Solve (see below).

5. Calculation of stress fields.

The 3 procedures Calculate_Rx_and_Ry, Calculate_Ak and Calculate_Stress_Fields do this job. They consist mainly of some long and intricate summations. When the load angle (delta) has been calculated, it is output, and so are the values for the stress fields.

MODULE CCALC

This module consists of several functions which perform arithmetic calculations on complex numbers. These functions are :-

1. FUNCTION Add - adds two complex numbers.

2. FUNCTION Subt - subtracts one complex number from another.

3. FUNCTION Mul - multiplies two complex numbers.

4. FUNCTION Conjugate - conjugates a complex number.

5. FUNCTION Divc - divides one complex number by another. To find (a,b) / (c,d) the following formula is used :-
   (a,b) / (c,d) = ((a,b) * (c,-d)) / (c * c + d * d)

6. FUNCTION Skrt - squares a complex number.

7. FUNCTION kube - cubes a complex number.

8. FUNCTION Sqt - finds a square root of a complex number. Since we can find two complex numbers which, when squared, yield the same result, the FUNCTION returns the square root with non-negative real part (except when finding the square root of (0.b) where b < 0).

9. FUNCTION powc - raises a complex number to a power using de Mouvre's formula :-
   (r(cosx+isinx))**n = r**n*(cos(n*x)+sin(n*x))
   Use is made of the FUNCTION Atan2, which returns the arctan of an angle in the correct quadrant.

MODULE SOLVE

Module Solve is used to solve a system of linear equations by Gaussian elimination using partial pivoting. PROCEDURE LU_Decompose converts the matrix to upper triangular form, and then PROCEDURE Back_Substitute does back substitution to find the solution.

PROGRAM COMPHOLE

The main program Comphole is used to coordinate the two main functions, which are :-

1. Plotting of data

2. Creation of data

The program is menu driven; the first menu being used to find out whether the user wants to plot data or to create data. If the data creation option is chosen, module Bhol takes over control, and returns when its operation is complete. See the section on module Bhol above for more details.

If the user decides to plot data instead, he can do one of two things :-

1. Plot one set of data.

2. Plot a succession of sets of data.

One plot

Here, the program reads in input from a set of 4 files, does the necessary calculations for 3D graphics, and then generates the plot on the screen. (See the section on input/output for more info about these files).

The first time a plot is drawn, the user is asked for the names of the input files. The plot is then drawn. On completion of the first plot, however, a new menu appears, which gives him the following options :-

1. Redraw the previous plot.

2. Draw a plot using different data (ie the user is asked for a new file name).

3. Change the tilt or rotate angles.

4. Decide whether the plot is to be output to a file for subsequent printing.

If the user decides to change the tilt or rotate angles, a new menu appears, which displays the current values of these angles. These can be changed, if desired. Once one of these angles is changed, the user can redraw the plot.

If the user decides to change the setting of the send-to-file option, a new menu will appear, displaying the current setting as being ON or OFF. If the user wants to set it to ON, he will be asked to give the name of the file to which the plot is to be sent.

A succession of plots

If the user wants to get a succession of plots, he must specify the names of the files from which the data for the plot is to be read. To avoid tedious typing, the program will read the data for plot number k from the class of files whose pathname is PNk. For example, if 3 plots are required, the data can be stored in 3 classes of files :-

1. //dfs/user/mike/test1
2. //dfs/user/mike/test2 and
3. //dfs/user/mike/test3

(For more information about 'classes of files' see the section on input/output below.)

The user will be asked to input the number of plots required. After one plot is complete, the user must press the space bar to get the next plot.

INPUT AND OUTPUT

When module Bhol generates output, it is stc
These 4 files have the same pathname, except
.data1 , .data2 , .data3 or .data4. For exam
output to have the name //dfs/user/mike/result.
in the files

1. //dfs/user/mike/results.data1
2. //dfs/user/mike/results.data2
3. //dfs/user/mike/results.data3 and
4. //dfs/user/mike/results.data4

Here, the 'class name' of the input files is

When the user wants a single plot, he will be
of the class of 4 files, so if he inputs '//df
will be read from the 4 files //dfs/user/mike/

If the user wants a succession of plots, one
of the user, and then the 4 files for plo
//dfs/user/mike/plotK.dataJ J = 1,2,3,4.

The input/output files will have the followi

1. Angle increment around the pin-hole (in d
2. Number of points per angle
3. displacement angle (in degrees)
4. load angle (in degrees)
5. the values of the stress fields.

INPUT AND OUTPUT

When module Bhol generates output, it is stored in a class of 4 files. These 4 files have the same pathname, except that they have an ending .data1 , .data2 , .data3 or .data4. For example, if the user wants the output to have the name //dfs/user/mike/results, the output will be stored in the files

1. //dfs/user/mike/results.data1

2. //dfs/user/mike/results.data2

3. //dfs/user/mike/results.data3 and

4. //dfs/user/mike/results.data4

Here, the 'class name' of the input files is '//dfs/user/mike/results'.

When the user wants a single plot, he will be asked to give the pathname of the class of 4 files, so if he inputs '//dfs/user/mike/plot' the data will be read from the 4 files //dfs/user/mike/plot.dataJ J = 1,2,3,4.

If the user wants a succession of plots, one class name will be asked of the user, and then the 4 files for plot number K will come from //dfs/user/mike/plotK.dataJ J = 1,2,3,4.

The input/output files will have the following data :-

1. Angle increment around the pin-hole (in degrees)

2. Number of points per angle

3. displacement angle (in degrees)

4. load angle (in degrees)

5. the values of the stress fields.

GENERATING A PLOT.

To generate a plot, the program does the following :-

1. The x and y coordinates are calculated. If a plot has been previously generated, and none of the following has changed -

   a. the angle increment

   b. the number of points per angle

   c. the rotate angle

   d. the tilt angle

   then the new x and y coordinates will be the same as the old x and y coordinates, so they need not be recomputed. However, if the user changes at least one of the 4 values above, then the x and y coordinates must be recomputed.

2. The z values (the values of the stress fields) are read in. The maximum and minimum z values are also found.

3. The x,y and z coordinates are transformed into a pair of screen coordinates sx and sy (since computer monitors are unfortunately only 2 dimensional). This is done by PROCEDURE Project. Note that there are 4 sets of z coordinates (read in from the 4 files), and thus 4 graphs will be plotted on one display simultaneously, one in each quarter of the screen.

4. The display is initialised, by using the gpr procedure gpr_$init. The display takes up the whole screen, since the mode that gpr is set to in the program is gpr_$borrow. The graphics routines borrow the whole screen from the display manager, and give it back when the graphics display is terminated.

5. The plot is drawn, according to the screen coordinates sx and sy.

6. The box at the top of the screen (which will contain the values of the load and displacement angles), as well as the border lines and the two centre lines are drawn, in PROCEDURE Draw_Box.

7. The axes for each of the 4 graphics are drawn. This is done by PROCEDURE Draw_Axes. Note that the Y axis is replaced by the -Y axis.

8. A semicircle of radius 1, starting at (-1,0,0), passing thru (0,-1,0) and ending at (1,0,0) is drawn by PROCEDURE Draw_Arc.

9. The text is drawn by PROCEDURE Display_Text. The text consists of :-

   a. the name of each graph.

b. the maximum and minimum z values of each graph, except for the 4th graph (failure propensity), where the R and THETA values at which the maximum occurs is displayed instead.

c. the load and displacement angles.

10. The message 'Press space bar' is put on the top right hand corner of the display, once plotting is complete. The program then waits until the space bar is pressed, before continuing.

11. Once the space bar has been press.', if the program is plotting a succession of plots, it will draw the next plot, otherwise the display will be terminated (gpr_$terminate) and the screen will be the same as before the plot started. Note that if the program is doing a succession of plots, then it will calculate the next plot's sx and sy coordinates while the user is viewing the previous plot. In this way, some time is saved. If the user presses the space bar before the message 'Press space bar' comes up, nothing happens - he will just have to press the space bar again at the right time.

25X1

SENDING GRAPHICS OUTPUT TO A FILE

If the user has set the send-to-file variable ON, then once the plot is drawn (and before the 'Press space bar' message appears), the output will be sent to the file that the user specified. This file can the be printed on a printer. (See later for details.) Note that the whole screen will not fit on the printer paper - it is just too wide. To overcome this problem, the display is output in two sections, first the left half and then the right half.

HOW TO COMPILE, BIND AND RUN THE PROGRAM

Compiling

Since the whole package consists of a main program and 3 modules, each of these have to be compiled. Suppose our 4 program sections are in the files Comphole.pas, Bhol.pas, Solve.pas and Ccalc.pas. Then we compile these 4 ogram sections by issuing the commands :-

1. pas comphole.pas -nwarn

2. pas bhol.pas -nwarn

3. pas solve.pas

4. pas ccalc.pas

The '-nwarn' is used just to suppress some pesky warnings that the compiles deems necessary to dump on us.

If all 4 program sections are compiled, and then a change is made to any one of them, only this section need be recompiled.

Binding

Once all parts of the program have been compiled, they must be bound by issuing the following command :-

bind comphole.bin bhol.bin solve.bin ccalc.bin -b run
The '-b run' bit means that the binary code of the whole, bound program will be sent to a file called 'run'

Running

Once the program sections have been bound, and the binary code sent to file 'run', issue the following command to run the package :-

```
run
```

PRINTING

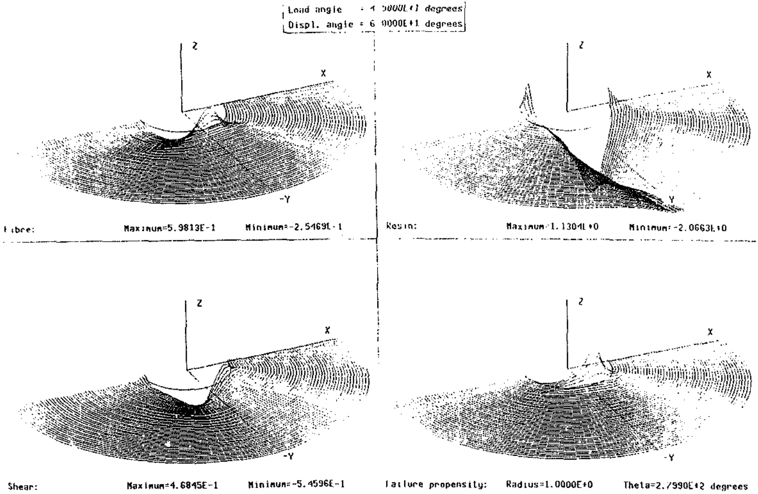
If the plot has been sent to output file 'plot.data', then the file can be printed by issuing the command :-

```
prf plot.data -plot
```

SOME CLOSING COMMENTS

The data creation process takes about 2/3 of the time that it takes the C program on the PC. For angle increments of 5 degrees, with 13 points per angle, with 22 points along the hole edge, and 30 terms in the sine series, the PASCAL program took about 20 minutes to calculate the stress fields, compared with 30 minutes for the C program on the PC. An advantage of using the APOLLO is that once the data creation process has begun, a new shell can be created, and the program run with different input, so many sets of data can be created simultaneously, and lots of time saved.

The calculations for the 3D graphics do not take very long - anything from 3 to 10 seconds, depending on how many users are using the system.


Load angle = 4.0000E+1 degrees
Displ. angle = 6.0000E+1 degrees
Z
X
-Y
Fibre: Maximum=5.9813E-1 Minimum=-2.5469E-1
Z
X
Y
Resin: Maximum=1.1304E+0 Minimum=-2.0663E+0
Z
X
-Y
Shear: Maximum=4.6845E-1 Minimum=-5.4596E-1
Z
X
-Y
Failure propensity: Radius=1.0000E+0 Theta=2.7990E+2 degrees

APPENDIX E

E.1 Function Terms used in Displacement Formulae (See Equations (2.39) and 2.40))

$$F_1^{(m,n)} = \frac{n\ a_n\{(m^2 - n^2 + 1)\ C_5 - 2_m\ C_6\}}{N_{m,n}}$$

$$F_1^{*(m,n)} = \frac{2n\ a_n C}{\{(m-1)^2 - n^2\}\{(m+1)^2 - n^2\}}$$

$$F_3^{(n)} = \frac{a_n(C_5 - C_6) + a_{n+2}\ (C_5 + C_6)}{n + 1}$$

$$F_3^{*(n)} = \frac{(C_4)\ (a_n - a_{n+2})}{n + 1}$$

$$F_2^{(m,n)} = \frac{n\ a_n\{(m^2 - n^2 + 1)\ C_6 - 2_m\ C_7\}}{N_{m,n}}$$

$$F_2^{*(m,n)} = \frac{n\ a_n(m^2 - n^2 + 1)\ C_4}{N_{m,n}}$$

$$F_4^{(n)} = \frac{a_n(C_6 - C_7) + a_{n+2}\ (C_6 + C_7)}{n + 1}$$

$$F_4^{*} = \frac{(C_4)\ (a_n + a_{n+2})}{n + 1}$$

E.2 Expansion of Terms Appearing in Equation (2.32)

$$f_1 = P_o \left[\frac{1}{4}\left\{ a_2 \cos\theta + \sum_{n=1,2,3,4}^{\infty} \frac{(a_n + a_{n+2})}{n+1} \cos(n+1)\theta \right\} - \frac{\sin 2\theta}{4\pi}\frac{R_x}{P_o} \right.$$

$$- \frac{\sin\theta}{\pi} \sum_{n=1,3,5}^{\infty} \frac{a_n}{n} - \frac{1}{2}\left[\sum_{n=1,3}^{\infty} \sum_{m=2,4}^{\infty} \sum_{n=2,4}^{\infty} \sum_{m=3,5}^{\infty} \right.$$

$$\left. \left. a_n \left[\frac{1}{n+m} + \frac{1}{n-m}\right] - \frac{1}{n-m}\left\{\frac{\sin(m-1)\theta}{(m-1)} + \frac{\sin(m-1)}{(m-1)}\theta\right\} \right] \right]$$

$$f_2 = P_o \left[\frac{1}{4}\left\{ a_2 \sin\theta + \sum_{n=1,2,3}^{\infty} \frac{(a_n + a_{n+2})}{n+1} \sin(n+1)\theta \right\} - \frac{\cos 2\theta}{4\pi}\frac{R_x}{P_o} \right.$$

$$- \frac{\cos\theta}{\pi} \sum_{n=1,3,5}^{\infty} \frac{a_n}{n} - \frac{1}{2\pi}\left[\sum_{n=1,3}^{\infty} \sum_{m=2,4}^{\infty} \sum_{n=2,4}^{\infty} \sum_{m=3,5}^{\infty} \right.$$

$$\left. \left. a_n \left[\frac{1}{n+m} + \frac{1}{n-m}\right] \left\{\frac{\cos(m-1)\theta}{(m-1)} - \frac{\cos(m-1)}{(m-1)}\theta\right\} \right] \right]$$

$$K = P_o \left[\frac{1}{\pi} \sum_{n=1,3}^{\infty} \frac{a_n}{n} - \frac{1}{2\pi}\left[\sum_{n=1,3}^{\infty} \sum_{m=2,4}^{\infty} \sum_{n=2,4}^{\infty} \sum_{m=3,5}^{\infty} \right. \right.$$

$$\left. \left. a_n \left[\frac{1}{n+m} + \frac{1}{n-m}\right] \left[\frac{1}{m-1} + \frac{1}{m+1}\right] \frac{R_x}{4\pi P_0} \right] \right]$$