Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria dei Materiali

Efficient computation of the strain energy density for the assessment of fracture and fatigue of welded structures

Candidato:
Mattia Pujatti
Matricola 622088

Relatori:
Prof. Alexander Düster
Prof. Paolo Lazzarin

Anno Accademico 2011–2012

Dedicated to the memory of my grandfather Lino.

Knowledge is of no value unless you put it into practice.

—Anton Čechov (1860–1904)

ABSTRACT

Nowadays, the most widespread approach to fatigue design is based on S-N curves. Although this approach works in a lot of practical situations, there are also many others in which it does not give enough accurate results: The most important exception are probably the welded joints, which are widely adopted for the connection of structural parts.

In recent years, many authors suggested to assess the fatigue life of welds on the basis of the local stress and strain fields in the most stressed zones, using the concepts of fracture mechanics. It was in this context that the SED criterion was formulated.

The purpose of this work is to investigate the numerical implementation of the SED criterion, and to further enhance its efficiency on the basis of some theoretical observations, as we are going to explain in details.

SOMMARIO

Al giorno d'oggi, l'approccio più diffuso alla progettazione a fatica è basato sulle curve S-N. Sebbene esso si riveli efficace in molte situazioni di interesse pratico, in molti altri casi esso non è in grado di dare risultati sufficientemente accurati: probabilmente, il caso più eclatante riguarda i giunti saldati, una soluzione ampiamente adottata per la connessione di elementi strutturali.

Negli ultimi anni, molti autori hanno suggerito di stimare la vita a fatica delle saldature sulla base dei campi locali di tensione e deformazione nelle zone maggiormente sollecitate, piuttosto che su un approccio in tensione nomale. È in questo contesto che il criterio SED, formulato sui concetti della meccanica della frattura, è stato proposto.

L'obbiettivo che questo lavoro si prefigge è di indagare l'implementazione numerica del criterio SED, e di migliorarne l'efficienza sulla base di alcune osservazioni teoriche, come verrà spiegato in dettaglio.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS

In order not to overload the reading, only the main symbols were reported. Although we have tried to avoid repetition as much as possible, some of them were unavoidable.

Latin alphabet

A	area
a	crack half length
C_{ijkl}	stiffness tensor
$\mathbb{C}$	set of complex numbers
E	Young's modulus
E'	effective Young's modulus $\left(= \frac{E}{1-\nu^2}\right)$
$e_{\mathcal{SED}}$, $\tilde{e}_{\mathcal{SED}}$	relative error
F_i	forces
$\{\mathbf{F}\}$	body force vector
G, G'	shear modulus
I	modulus of inertia, set of FE nodes
I^*	subset of FE enriched nodes
I_1, I_2	angular integrals of mode I and II
Im	imaginary part of a complex quantity
i	imaginary unit $\left(= \sqrt{-1}\,\right)$
K_1, K_2	NSIFs of mode I and II
ΔK_{1C}	critical NSIF of mode I
K_I	SIF of mode I
L^2	square-integrable functions space
m	number of Gaussian points
N_i	shape functions
n	number of subdivisions
n_i	normal unit vector components
P_n	Legendre polynomials

R	characteristic radius
$\mathbb{R}$	set of real numbers
Re	real part of a complex quantity
r, ϑ, z	cylindrical coordinates
r, ϑ, φ	spherical coordinates
S	stress ratio $\left(= \frac{\sigma_{\min}}{\sigma_{\max}}\right)$
$\mathcal{SED}$	local strain energy density
$\Delta\,\mathcal{SED}_C$	critical strain energy density
T_i	traction vector components
t_i	Gauss-Legendre abscissas
$\{\mathbf{T}\}$	traction vector
$\mathcal{U}(R)$	local strain energy
u, v, w	Cartesian displacements
$\{\mathbf{u}\}$	displacement vector
W	total strain energy density
w_i	Gauss-Legendre weights
x, y, z	Cartesian coordinates
$Z(z)$	Westergaard stress function
$Z^\star(z)$	primitive of Z $\left(= \int Z(z)\,dz\right)$
z	complex variable $\left(= x + i\,y \text{ or } r\,e^{i\vartheta}\right)$
$\bar{z}$	conjugate complex variable $\left(= x - i\,y \text{ or } r\,e^{-i\vartheta}\right)$

Greek alphabet

α	half notch opening angle
Γ_{12}	boundary
Γ_c	crack boundary
γ	supplementary angle of α $(= \pi - \alpha)$
$\Delta(\cdot)$	finite variation of a quantity
δ_{ij}	Kronecker symbol
$\delta(\cdot)$	first variation of a functional
ε_{ij}	strain tensor components
$\{\boldsymbol{\varepsilon}\}$	strain tensor

κ	Kolosov's constant
λ_1, λ_2	Williams' eigenvalues of mode I and II
ν	Poisson's ratio
ν'	effective Poisson's ratio $\left(= \frac{\nu}{1-\nu}\right)$
ξ, η	standard element coordinates
σ_{ij}	stress tensor components
$\tilde{\sigma}_{ij}^{(I)}$, $\tilde{\sigma}_{ij}^{(II)}$	angular functions of mode I and II
$\Delta\sigma_A$, $\Delta\sigma_D$	fatigue life at 2×10^6 and 5×10^6 cycles
$\{\boldsymbol{\sigma}\}$	stress tensor
Φ	Airy stress function
$\Phi(\boldsymbol{x})$	signed-distance function
$\boldsymbol{\psi}_{\text{crack}}(\boldsymbol{x})$	crack tip enrichment vector
$\boldsymbol{\psi}_{\text{notch}}^{(I)}(\boldsymbol{x})$, $\boldsymbol{\psi}_{\text{notch}}^{(II)}(\boldsymbol{x})$	notch tip enrichment vectors of mode I and II
Ω_{st}	standard element

Other symbols

∇^2	Laplacian operator or nabla squared

Acronyms

1-D, 2-D, 3-D	one-, two-, and three-dimensional
BC	boundary condition
DOF	degrees of freedom
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
NSIF	Notch Stress Intensity Factor
ODE	ordinary differential equation
PDE	partial differential equation
PU	partition of unity
SED	strain energy density
SIF	Stress Intensity Factor
XFEM	extended Finite Element Method

ACKNOWLEDGEMENTS

First of all, I would like to thank all the persons at the Department of Ship Structural Design and Analysis of the Technical University of Hamburg-Harburg, for the welcoming atmosphere I breathed when I was there. I would like especially to express my gratitude to my supervisor, Professor Alexander Düster, for his patient guidance and encouragement during all the period of my thesis. A person who deserves a special mention is Richard Szöke-Schuller, student in Mathematics in Science and Engineering at the Technical University of München, who kindly provided me his experience on Code_Aster. It was only thanks to his help that I could carry out successfully all the Finite Element simulations that I had scheduled. Many thanks go also to Bjarne, Claas, Dr. Höft, Martin, and Sonja, for all the nice talks we had together. I would like to thank Maedeh, Meysam, and Silvan, who I have the pleasure to call friends. A special thank goes to Meysam, not only for the moments of fun and relax we shared, but also for the help and the support he gave me in a countless number of occasions: Thank you, *compare*!

Outside the university, I am grateful to my flatmates. We had great moments together and I always felt at home with them. Receiving their calls and their messages when I discovered my disease was really heartwarming for me. For all of this and much more, thanks from the heart, Amanda, Bachir, Carlos, Denik, José, Martin, Meiyue, and Oscar. I would like to mention also Emilio, Elizabeth, Gerry, Sandra, Vadim, and Wendy, for their kindness and friendliness.

Mattia

RINGRAZIAMENTI

Il ringraziamento più sentito, in ambito accademico, va al Professor Lazzarin, per essersi gentilmente reso disponibile come Relatore Interno per la mia tesi. Ringrazio anche il Dottor Zappalorto, per gli innumerevoli consigli e suggerimenti che mi ha elargito prima, dopo e durante il periodo di tesi.

Ringrazio i miei genitori, Maurizio e Viviana, senza il cui supporto non avrei mai raggiunto questo traguardo. Ringrazio Luca e Simone, che pur conoscendo bene i difetti del loro fratello, si ostinano a volergli bene.

Ringrazio i miei amici di sempre, Albertone, Andrea, Baffo, Bezza, Chiara, Dejo, Gigi, Giulia, Ilijas, Manny, Maurella, Numan e Vox.

Ringrazio Jacopo e Luciano per i bei momenti che abbiamo trascorso assieme nella splendida Amburgo.

Ringrazio gli amici con cui ho condiviso gli anni dell'università, Ale, Boaz, Ferraz, Johnny, Moreno, Nicolò, Paolo, Riccardo, il Rosso, Zoc.

Grazie ai migliori coinquilini di sempre: Alberto, Annalisa, Olga e la piccola Luna!

Ma soprattutto grazie ad Elisa, per avermi fatto capire che in una coppia non esistono solo i momenti di idillio, ma anche quelli di dialogo e discussione; per tutto il sostegno e la comprensione che mi ha dato, per essermi stata vicina anche quando ero oggettivamente intrattabile; per essere stata non solo la mia ragazza, ma anche un'amica sincera, una confidente, una compagna di gioco; per l'amore e l'affetto che proviamo l'uno per l'altra; per tutto quello che abbiamo vissuto insieme e che insieme vivremo d'ora inanzi, per sempre.

Mattia

INTRODUCTION

Since its discover in the middle of the 19th century, fatigue has been a phenomenon extensively studied by engineers. Nowadays, all the norms on structural design present extensive sections dedicated to fatigue, that take in account of different aspects like variable amplitude and multiaxial loadings, stress concentration effects, corrosion, etc.

In the vast majority of the norms, the data are given in terms of nominal stresses, using the S-N curves. Although this approach works in a lot of practical situations, there are also many others in which it does not give enough accurate results: The most important exception are probably the welded joints, which are widely adopted for the connection of structural parts. To overcome this issue, the International Institute of Welding separates the joints on the basis of their structural details in different fatigue classes (called FAT classes) and assigns to each one a specific S-N curve [14]; a similar approach is followed also in the Eurocodes 3 and 9 [9, 10]. This strategy is obviously expensive and time-consuming, since the number of welds realized in the industrial practice is enormous.

In recent years, considering the substantially brittle behaviour of the welds, many authors suggested to assess their fatigue life on the basis of the local stress and strain fields in the most stressed zones [24]. Since the aim of fracture mechanics is to describe the perturbation in the local quantities induced by internal defects like cracks or flaws in a loaded structure, it was natural to employ it in this context. Nonetheless, it was not the first time that the concepts of fracture mechanics were applied to fatigue: In the 1960s, Paris *et al.* [22, 23] found that it was possible to obtain a good empirical correlation between the crack length and the range of the Stress Intensity Factor of mode I; Paris' law is now a standard in the design of aircraft components.

The biggest difference between these two «waves» of fracture mechanics is the enormously higher calculus capabilities of modern computers: If one time it was necessary to rely mainly (if not exclusively) on experiments, now the trend is to couple the powerful analytical models developed by fracture mechanics with the flexibility offered by numerical analysis. Although it is now possible to realize very sophisticated simulations, the computational costs are still a major concern. In fact, the short times often available in the industrial practice tend to favour rapid solutions, whose results have to be accurate and highly reliable. Therefore, there is still a great interest in finding easy ways to conduct robust analyses at low computational costs.

It is from this perspective that the SED criterion was formulated by Lazzarin and Zambardi [19]. The Authors focused on the fatigue crack *initiation*, thus neglecting the path that the crack is going to follow once it starts to propagate. Although this approach may seem limiting, it has the great advantage of requiring only a static structural analysis. This allows (i) to give a rigorous mathematical basis to the criterion and (ii) to implement it easily in the Finite Element codes. On the contrary, Paris' law requires an empirical connection between the crack length, which increases with time, and the Stress Intensity Factor, which is a static quantity, requiring ineluctably some data fitting procedures. As a consequence, a huge number of different crack propagation laws have been proposed in the years in literature; in some cases, also because of the scatter of the values measured experimentally, it was reached the almost paradoxical result that the same set of data was fitted by apparently contradictory laws, with no possibility to determine which one was the most correct [4]. In addition, the related numerical simulations take significantly longer computational times, since they require a dynamic analysis.

The main purpose of this work is to carry out the numerical implementation of the SED criterion, taking advantage of some recent theoretical observations to enhance its efficiency, as explained in details in chapter 4.

The document consist in five chapters and three appendices. The chapters are thus structured:

IN THE FIRST CHAPTER, some basic aspects of the theory of elasticity are recalled.

IN THE SECOND CHAPTER, the basic equations of the SED criterion are derived.

IN THE THIRD CHAPTER, the theory of the Finite Element Method is briefly discussed.

IN THE FOURTH CHAPTER, the numerical procedures adopted are described and the related results are commented.

IN THE FIFTH CHAPTER, the conclusions are reported and possible further research hints are proposed.

while for what concerns the appendices:

THE APPENDIX A describes briefly the main properties of the shape functions.

THE APPENDIX B reports all the Python scripts used for validating the algorithm written.

THE APPENDIX C reports all the command files used to run the Finite Element simulations.

1 PLANE ELASTICITY

1.1 BASIC RELATIONS

Let us start by recalling the stress-strain relations for a homogeneous, isotropic material as predicted by linear elasticity. In a Cartesian coordinate system defined by the x, y, and z axes, they are [25, p. 82]:

$$\begin{aligned}
\varepsilon_x &= \frac{1}{E}\left[\sigma_x - \nu\left(\sigma_y + \sigma_z\right)\right], &\quad \gamma_{xy} &= \frac{\tau_{xy}}{G} \\
\varepsilon_y &= \frac{1}{E}\left[\sigma_y - \nu\left(\sigma_x + \sigma_z\right)\right], &\quad \gamma_{yz} &= \frac{\tau_{yz}}{G} \\
\varepsilon_z &= \frac{1}{E}\left[\sigma_z - \nu\left(\sigma_x + \sigma_y\right)\right], &\quad \gamma_{xz} &= \frac{\tau_{xz}}{G}\,.
\end{aligned} \tag{1.1}$$

As an alternative, using the tensor notation, one can write [25, p. 82]:

$$\varepsilon_{ij} = \frac{1+\nu}{E}\,\sigma_{ij} - \frac{\nu}{E}\,\sigma_{kk}\,\delta_{ij} \tag{1.2}$$

where the tensor shear strains are half of the corresponding engineering strains and δ_{ij} is the Kronecker symbol:

$$\delta_{ij} := \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j\,. \end{cases} \tag{1.3}$$

The elastic behaviour of an isotropic material is completely described by two parameters. It is in fact possible to demonstrate that the shear modulus G, the YOUNG's modulus E and the POISSON's ratio ν are related by [30, pp. 8–9]:

$$G = \frac{E}{2\left(1+\nu\right)}\,. \tag{1.4}$$

The strain-displacement relations, according to the small deformation theory, are [25, p. 36]:

$$\begin{aligned}
\varepsilon_x &= \frac{\partial u}{\partial x}, & \varepsilon_y &= \frac{\partial v}{\partial y}, & \varepsilon_z &= \frac{\partial w}{\partial z} \\
\gamma_{xy} &= \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}, & \gamma_{xz} &= \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}, & \gamma_{yz} &= \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}
\end{aligned} \tag{1.5}$$

where u, v, and w are the displacements in the x, y, and z directions, respectively. In tensor notation, we write [25, p. 37]:

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right). \tag{1.6}$$

Since most of the three-dimensional elasticity problems are not easy to solve, it is quite common in the engineering practice to further simplify the equations just presented, as we are now going to explain.

1.2 PLANE STRAIN

This hypothesis is typical in the case of thick sections, for which the strains in the z direction are constrained and therefore considered negligible. Hence, it is possible to write [25, p. 136]:

$$\varepsilon_z = \gamma_{xz} = \gamma_{yz} = 0\,, \quad \sigma_z = \nu\,(\sigma_x + \sigma_y)\,. \tag{1.7}$$

It is important to notice that this assumption, in the most general case, leads to a triaxial stress condition, since σ_z can differ from zero. Under these hypotheses, the only non-trivial relations in the system (1.1) are:

$$\varepsilon_x = \frac{1+\nu}{E}\,(\sigma_x - \nu\,\sigma_y)\,,\; \varepsilon_y = \frac{1+\nu}{E}\,(\sigma_y - \nu\,\sigma_x)\,,\; \gamma_{xy} = \frac{\tau_{xy}}{G}\,. \tag{1.8}$$

1.3 PLANE STRESS

This hypothesis is applied to thin sections, where the absence of stresses at the edges acting in the thickness direction is extended inside the body. In other words, only in-plane stresses are admitted. Mathematically speaking, this means [25, p. 138]:

$$\sigma_z = \tau_{xz} = \tau_{yz} = 0\,, \quad \varepsilon_z = -\frac{\nu}{E}\,(\sigma_x + \sigma_y)\,. \tag{1.9}$$

The system of equations (1.1) then reduces to:

$$\varepsilon_x = \frac{1}{E}\,(\sigma_x - \nu\,\sigma_y)\,, \quad \varepsilon_y = \frac{1}{E}\,(\sigma_y - \nu\,\sigma_x)\,, \quad \gamma_{xy} = \frac{\tau_{xy}}{G}\,. \tag{1.10}$$

1.4 GENERALIZED PLANE ELASTICITY

By using the effective elastic constants E', ν' defined in Table 1.1, equations (1.8) and (1.10) can be rewritten as:

$$\varepsilon_x = \frac{1}{E'}\,(\sigma_x - \nu'\sigma_y)\,, \quad \varepsilon_y = \frac{1}{E'}\,(\sigma_y - \nu'\sigma_x)\,, \quad \gamma_{xy} = \frac{\tau_{xy}}{G'} \tag{1.11}$$

where the effective shear modulus G' coincides with G:

$$G' = \frac{E'}{2\,(1+\nu')} = \frac{E}{2\,(1+\nu)} = G\,. \tag{1.12}$$

Table 1.1. Definitions of the effective elastic constants E' and ν' [2, p. 38].

	Plane stress	Plane strain
E'	E	$\dfrac{E}{1-\nu^2}$
ν'	ν	$\dfrac{\nu}{1-\nu}$

The relations in (1.11) describe the *generalized plane elasticity* problem. They can be inverted so to give explicitly the dependence on the strains of the in-plane stresses σ_x, σ_y, and τ_{xy}, provided that $\nu < 0.5$, i.e. for every material which is subjected to a variation in volume because of the applied loads.[1] Equations (1.1) to (1.11) can be used also in a spherical (or cylindrical) coordinate system, upon substitution of the tern (x, y, z) with (r, ϑ, φ) (respectively (r, ϑ, z)). For the displacements, the symbols usually adopted are u_r, u_ϑ, and u_φ (respectively u_z).

Despite the fact that both are just an idealization of the real problems (usually halfway between one condition and the other), these approximations are widespread in the engineering practice and are the starting point of a very powerful mathematical formalism which will be described in details later on.

1.5 EQUILIBRIUM AND COMPATIBILITY EQUATIONS

1.5.1 Cartesian coordinates

Once we have defined the stress components acting on the body, we can derive the equilibrium equations in the planar case, which turn out to be [25, p. 136]:

$$\begin{cases} \dfrac{\partial \sigma_x}{\partial x} + \dfrac{\partial \tau_{xy}}{\partial y} + F_x = 0 \\ \dfrac{\partial \tau_{xy}}{\partial x} + \dfrac{\partial \sigma_y}{\partial x} + F_y = 0 \end{cases} \tag{1.13}$$

where F_x, F_y are the body forces (e.g. gravity). The system (1.13) consists of two equations in three unknowns, and cannot be solved without introducing another condition, which is the congruence of planar strains. From equation (1.5), for the planar case, the strains are thus related to the displacements:

$$\varepsilon_x = \frac{\partial u}{\partial x}, \quad \varepsilon_y = \frac{\partial v}{\partial y}, \quad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}. \tag{1.14}$$

1 Rubbers are nearly incompressible materials, with a Poisson's ratio very close to the limit value of 0.5 [25, pp. 84–85].

By calculating the mixed derivative of γ_{xy}:

$$\begin{aligned}\frac{\partial^2\gamma_{xy}}{\partial x\,\partial y} &= \frac{\partial}{\partial x}\left(\frac{\partial^2 u}{\partial y^2}\right) + \frac{\partial}{\partial y}\left(\frac{\partial^2 v}{\partial x^2}\right)\\ &= \frac{\partial^2\varepsilon_x}{\partial y^2} + \frac{\partial^2\varepsilon_y}{\partial x^2}\end{aligned} \tag{1.15}$$

we get the so-called *compatibility equation* [25, p. 137]. Then, by (i) switching from strains to stresses through equations (1.11), (ii) differentiating the first (respectively second) equation of equilibrium with respect to x (respectively y), and (iii) introducing it into equation (1.15), one obtains [25, pp. 137, 140]:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)(\sigma_x + \sigma_y) = -f(\nu)\left(\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}\right) \tag{1.16}$$

where $f(\nu)$ is a function of the Poisson's ratio:

$$f(\nu) \coloneqq \begin{cases} 1+\nu\,, & \text{plane stress}\\ \dfrac{1}{1-\nu}\,, & \text{plane strain}\,. \end{cases} \tag{1.17}$$

If we set $F_x = F_y = 0$ and we introduce the ∇^2 notation:

$$\nabla^2(\cdot) \coloneqq \frac{\partial^2(\cdot)}{\partial x^2} + \frac{\partial^2(\cdot)}{\partial y^2} \tag{1.18}$$

we can also write:

$$\nabla^2(\sigma_x + \sigma_y) = 0\,. \tag{1.19}$$

By noticing that the sum in brackets represents the first fundamental invariant of the stress tensor [25, p. 66], we can say that *in plane elasticity, in the absence of body forces, the first stress invariant is a solution of* LAPLACE*'s equation*.

1.5.2 Polar coordinates

In polar coordinates, the planar equilibrium is [25, p. 146]:

$$\begin{cases} \dfrac{\partial\sigma_r}{\partial r} + \dfrac{1}{r}\dfrac{\partial\tau_{r\vartheta}}{\partial\vartheta} + \dfrac{\sigma_r - \sigma_\vartheta}{r} + F_r = 0\\ \dfrac{\partial\tau_{r\vartheta}}{\partial r} + \dfrac{1}{r}\dfrac{\partial\sigma_\vartheta}{\partial\vartheta} + \dfrac{2\,\tau_{r\vartheta}}{r} + F_\vartheta = 0\,. \end{cases} \tag{1.20}$$

The strain-displacement relations are [25, p. 146]:

$$\varepsilon_r = \frac{\partial u_r}{\partial r}\,,\quad \varepsilon_\vartheta = \frac{1}{r}\left(u_r + \frac{\partial u_\vartheta}{\partial\vartheta}\right)\,,\quad \gamma_{r\vartheta} = \frac{1}{r}\frac{\partial u_r}{\partial\vartheta} + \frac{\partial u_\vartheta}{\partial r} - \frac{u_\vartheta}{r} \tag{1.21}$$

and the compatibility equations reads [6, p. 460]:

$$\frac{\partial}{\partial r}\left(r\frac{\partial \gamma_{r\vartheta}}{\partial \vartheta} - r^2\frac{\partial \varepsilon_\vartheta}{\partial r}\right) + r\,\frac{\partial \varepsilon_r}{\partial r} - \frac{\partial^2 \varepsilon_r}{\partial \vartheta^2} = 0\,. \tag{1.22}$$

Following the same procedure described for the Cartesian coordinate system, equation (1.22) becomes [25, p. 147]:

$$\nabla^2(\sigma_r + \sigma_\vartheta) = -f(\nu)\left(\frac{\partial F_r}{\partial r} + \frac{F_r}{r} + \frac{1}{r}\frac{\partial F_\vartheta}{\partial \vartheta}\right) \tag{1.23}$$

where $f(\nu)$ is still defined by equation (1.17) and ∇^2 is

$$\nabla^2(\cdot) \coloneqq \frac{\partial^2(\cdot)}{\partial r^2} + \frac{1}{r}\frac{\partial(\cdot)}{\partial r} + \frac{1}{r^2}\frac{\partial^2(\cdot)}{\partial \vartheta^2}\,. \tag{1.24}$$

1.6 AIRY STRESS FUNCTION

One of the most powerful tools available for the resolution of plane elasticity problems is the Airy stress function, denoted by the symbol Φ, whose definition is [25, p. 144]:[2]

$$\sigma_x = \frac{\partial^2\Phi}{\partial y^2}\,, \qquad \sigma_y = \frac{\partial^2\Phi}{\partial x^2}\,, \qquad \tau_{xy} = -\frac{\partial^2\Phi}{\partial x\,\partial y} \tag{1.25}$$

in Cartesian coordinates and

$$\sigma_r = \frac{1}{r}\frac{\partial\Phi}{\partial r} + \frac{1}{r^2}\frac{\partial^2\Phi}{\partial\vartheta^2}\,, \qquad \sigma_\vartheta = \frac{\partial^2\Phi}{\partial r^2}\,, \qquad \tau_{r\vartheta} = -\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial\Phi}{\partial\vartheta}\right) \tag{1.26}$$

in polar coordinates [25, p. 147]. It can be easily shown that Φ automatically satisfies the equilibrium equations (1.13) (respectively equations (1.20)) when no body forces are involved. The condition on the first invariant, expressed by equation (1.19) or (1.23), turns out to be [25, pp. 145, 147]:

$$\nabla^2\,\nabla^2\Phi = 0\,. \tag{1.27}$$

Equation (1.27) means that the Airy stress function is a *biharmonic* function. We remember that a function is said to be *harmonic* when it is a solution of Laplace's equation:

$$\nabla^2 u = 0 \quad \Leftrightarrow \quad u \text{ is harmonic}\,. \tag{1.28}$$

2 When the body forces are active, by assuming that exists a potential function V, such that $F_x = -\frac{\partial V}{\partial x}$ and $F_y = -\frac{\partial V}{\partial y}$, the Airy function can be defined as [25, p. 144]:

$$\sigma_x = \frac{\partial^2\Phi}{\partial y^2} + V\,, \qquad \sigma_y = \frac{\partial^2\Phi}{\partial x^2} + V\,, \qquad \tau_{xy} = -\frac{\partial^2\Phi}{\partial x\,\partial y}\,.$$

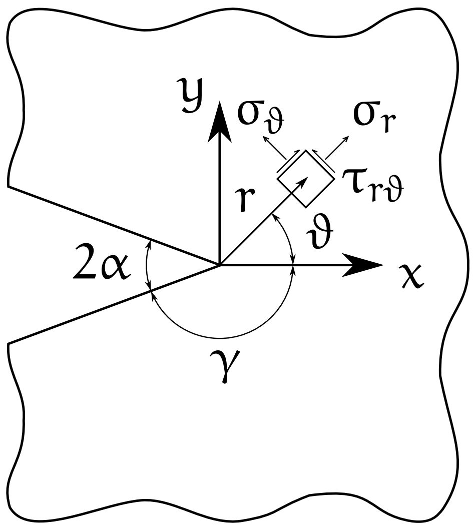


Figure 1.1. Configuration of the notch problem.

This important property is the basis of the method of complex variables, as will be explained in section 1.8.

1.7 WILLIAMS' EQUATIONS

In this section, we are going to describe Williams' treatise on sharp V-shaped notches [34], based on the Airy function formulation.

1.7.1 Stresses and displacements

Because of the configuration of the problem, it is suitable to adopt a polar coordinate system (see Figure 1.1). The biharmonic equation (1.27) then reads:

$$\left(\frac{\partial^2}{\partial r^2}+\frac{1}{r}\frac{\partial}{\partial r}+\frac{1}{r^2}\frac{\partial^2}{\partial \vartheta^2}\right)\left(\frac{\partial^2\Phi}{\partial r^2}+\frac{1}{r}\frac{\partial \Phi}{\partial r}+\frac{1}{r^2}\frac{\partial^2\Phi}{\partial \vartheta^2}\right)=0\,. \tag{1.29}$$

Exploiting the separation of variables, Williams assumed the following form for the stress function [34]:

$$\Phi(r,\vartheta)=r^{\lambda+1}\,F(\vartheta,\lambda) \tag{1.30}$$

which turns the previous PDE into an ODE which depends only on the ϑ angle:

$$\left[(\lambda-1)^2+\frac{\partial^2}{\partial\vartheta^2}\right]\left[(\lambda+1)^2+\frac{\partial^2}{\partial\vartheta^2}\right]F(\vartheta,\lambda)=0\,. \tag{1.31}$$

Its general solution is $F(\vartheta,\lambda) = e^{m\vartheta}$, where $m = m(\lambda)$ are the roots of the characteristic equation

$$\left[(\lambda-1)^2+m^2\right]\left[(\lambda+1)^2+m^2\right] = 0\,. \tag{1.32}$$

It follows that

$$m_k = \pm i\,(\lambda \pm 1)\,, \qquad \text{for } k = 1,\dots,4 \tag{1.33}$$

and $F(\vartheta,\lambda)$ is given by a linear combination of the elementary functions thus found:

$$F(\vartheta,\lambda) = \sum_{k=1}^{4} A_k\, F_k(\vartheta,\lambda) = \sum_{k=1}^{4} A_k\, e^{m_k\vartheta}\,. \tag{1.34}$$

Since the Airy function has to be real, by combining pairs of F_k and exploiting the well-known EULER formula $e^{i\vartheta} = \cos\vartheta + i\,\sin\vartheta$, it is possible to determine its final form [2, p. 145]:

$$\begin{aligned}\Phi(r,\vartheta) = r^{\lambda+1}\big[&A_1\cos(\lambda+1)\,\vartheta + A_2\cos(\lambda-1)\,\vartheta\\ &+ A_3\sin(\lambda+1)\,\vartheta + A_4\sin(\lambda-1)\,\vartheta\big].\end{aligned} \tag{1.35}$$

By using the definition (1.26) of the Airy stress function in polar coordinates, we can derive the stresses [2, p. 145]:

$$\begin{aligned}
\sigma_r &= r^{\lambda-1}\left[F''(\vartheta,\lambda) + (\lambda+1)\,F(\vartheta,\lambda)\right]\\
&= r^{\lambda-1}\big[-A_1\,\lambda\,(\lambda+1)\cos(\lambda+1)\vartheta - A_2\,\lambda\,(\lambda-3)\cos(\lambda-1)\vartheta\\
&\qquad - A_3\,\lambda\,(\lambda+1)\sin(\lambda+1)\vartheta - A_4\,\lambda\,(\lambda-3)\sin(\lambda-1)\vartheta\big],\\
\sigma_\vartheta &= r^{\lambda-1}\left[\lambda\,(\lambda+1)\,F(\vartheta,\lambda)\right]\\
&= r^{\lambda-1}\big[A_1\,\lambda\,(\lambda+1)\cos(\lambda+1)\vartheta + A_2\,\lambda\,(\lambda+1)\cos(\lambda-1)\vartheta\\
&\qquad + A_3\,\lambda\,(\lambda+1)\sin(\lambda+1)\vartheta + A_4\,\lambda\,(\lambda+1)\sin(\lambda-1)\vartheta\big],\\
\tau_{r\vartheta} &= -r^{\lambda-1}\left[\lambda\,F'(\vartheta,\lambda)\right]\\
&= r^{\lambda-1}\big[A_1\,\lambda\,(\lambda+1)\sin(\lambda+1)\vartheta + A_2\,\lambda\,(\lambda-1)\sin(\lambda-1)\vartheta\\
&\qquad - A_3\,\lambda\,(\lambda+1)\cos(\lambda+1)\vartheta - A_4\,\lambda\,(\lambda-1)\cos(\lambda-1)\big].
\end{aligned} \tag{1.36}$$

According to the original paper [34], the plane strain displacements are defined by the following relations:

$$\begin{aligned}
2G\,u_r &= r^\lambda\left[-(\lambda+1)\,F(\vartheta) + \frac{1}{1+\nu}\,G'(\vartheta)\right]\\
2G\,u_\vartheta &= r^\lambda\left[-F'(\vartheta) + \frac{\lambda-1}{1+\nu}\,G(\vartheta)\right]
\end{aligned} \tag{1.37}$$

where $G(\vartheta)$ is

$$G(\vartheta) = \frac{4}{\lambda-1}\left[A_2\sin(\lambda-1)\vartheta - A_4\cos(\lambda-1)\vartheta\right]. \tag{1.38}$$

By introducing $F(\vartheta)$, $G(\vartheta)$, and their first derivatives in the previous definitions, we obtain [2, p. 39]:

$$\begin{aligned} 2G\,u_r &= r^\lambda\big[-A_1(\lambda+1)\cos(\lambda+1)\vartheta + A_2(\kappa-\lambda)\cos(\lambda-1)\vartheta \\ &\qquad - A_3(\lambda+1)\sin(\lambda+1)\vartheta + A_4(\kappa-\lambda)\sin(\lambda-1)\vartheta\big] \\ 2G\,u_\vartheta &= r^\lambda\big[A_1(\lambda+1)\sin(\lambda+1)\vartheta + A_2(\kappa+\lambda)\sin(\lambda-1)\vartheta \\ &\qquad - A_3(\lambda+1)\cos(\lambda+1)\vartheta - A_4(\kappa+\lambda)\cos(\lambda-1)\vartheta\big] \end{aligned} \tag{1.39}$$

where κ is the Kolosov's constant [2, p. 151]:

$$\kappa := \begin{cases} \dfrac{3-\nu}{1+\nu}\,, & \text{plane stress} \\ 3-4\nu\,, & \text{plane strain}\,. \end{cases} \tag{1.40}$$

1.7.2 Singularity

Looking at the equations derived in the previous subsection, we notice that all the stress tensor components depend on a power of r: $\sigma_{ij} \sim r^{\lambda-1}$. Under certain conditions that we are going to define soon, the exponent of r is negative, i.e. the stresses go to infinity as r approaches zero: When a field shows this behaviour, it is called *singular*. The *singularity* — in this case, $\lambda - 1$ — is of great importance in structural engineering, since it describes the severity of the local stress field, and of the damage phenomena which are related to it. For a V-shaped sharp notch, the singularity depends on the prescribed boundary conditions, as we are now going to demonstrate. More generally, it can be determined also experimentally (for example using strain gauges) or numerically (for example with the Finite Element Method, by getting the slope of the stresses versus r in a log-log diagram, as explained in subsection 4.3.2).

The exponent $\lambda - 1$ can be determined by imposing the boundary conditions. Although in Williams' original article [34] the BCs are applied directly to $F(\vartheta, \lambda)$, we prefer to write them explicitly, using the trigonometric functions just derived. Under the hypothesis that both edges are *free*, i.e. that no stresses are applied, it must be:

$$\sigma_\vartheta(\pm\gamma) = \tau_{r\vartheta}(\pm\gamma) = 0 \qquad \Longrightarrow \qquad F(\pm\gamma) = F'(\pm\gamma) = 0\,. \tag{1.41}$$

We thus obtain a homogeneous system of four equations, where the matrix coefficients depend on the angle $\gamma = \pi - \alpha$:

$$\begin{bmatrix} (\lambda+1)\sin(\lambda+1)\gamma & (\lambda-1)\sin(\lambda-1)\gamma \\ (\lambda+1)\cos(\lambda+1)\gamma & (\lambda+1)\sin(\lambda-1)\gamma \end{bmatrix} \begin{Bmatrix} A_1 \\ A_2 \end{Bmatrix} = 0 \tag{1.42a}$$

$$\begin{bmatrix} (\lambda+1)\cos(\lambda+1)\gamma & (\lambda-1)\cos(\lambda-1)\gamma \\ (\lambda+1)\sin(\lambda+1)\gamma & (\lambda+1)\sin(\lambda-1)\gamma \end{bmatrix} \begin{Bmatrix} A_3 \\ A_4 \end{Bmatrix} = 0\,. \tag{1.42b}$$

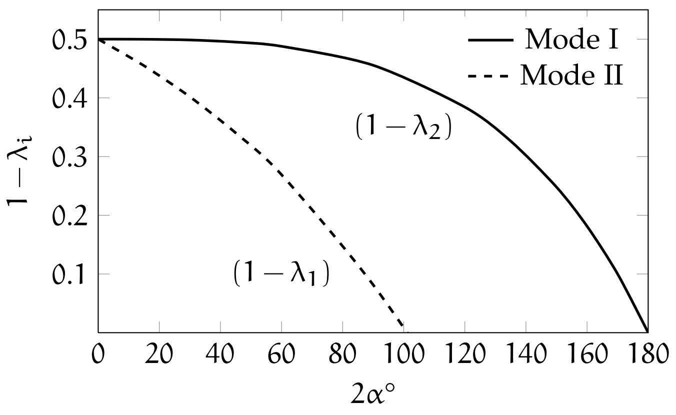


Figure 1.2. Williams' eigenvalues as a function of the notch opening angle [38, p. 26].

The coefficients were separated on the basis of the opening mode. In fact, A_1 and A_2 are related to mode I (*opening mode*), A_3 and A_4 to mode II (*sliding mode*): When a symmetric load (traction) is applied, only the first two coefficients are non-zero, vice versa when the plate is subjected to an antisymmetric load (pure shear) it follows that A_3, $A_4 \neq 0$.

The only non-trivial solution to the systems (1.42a) and (1.42b), according to ROUCHÉ-CAPELLI theorem, is obtained by imposing the determinant to be zero, i.e. by solving the following eigenvalue problem:

$$\begin{cases} \lambda_1 \sin(2\gamma) + \sin(2\lambda_1\gamma) = 0\,, & \text{for mode I} \\ \lambda_2 \sin(2\gamma) - \sin(2\lambda_2\gamma) = 0\,, & \text{for mode II}\,. \end{cases} \tag{1.43}$$

λ_1 and λ_2 are called Williams' eigenvalues of mode I and II, respectively. By solving numerically the transcendental equations in (1.43), it is possible to determine the stress singularities for the two modes.

Figure 1.2 reports the trends of the exponents $1-\lambda_{1,2}$ as a function of the notch opening angle 2α. From this chart, one can infer that:

- The eigenvalues $\lambda_{1,2}$ are always positive.[3]

- For both modes, the singularity tends to decrease as the opening angle 2α increases; it is always greater than or equal to -0.5.

3 This observation is explained mathematically with the boundedness of the local strain energy $\mathcal{U}(R)$ [2, p. 143]. If we write $\sigma_{ij} \sim r^a$, the energy related to a circle of radius R is

$$\mathcal{U}(R) = \frac{1}{2}\int_0^{2\pi}\int_0^R \sigma_{ij}\varepsilon_{ij}\, r\, dr d\vartheta = C\int_0^R r^{2a+1} dr$$

where C is a constant which depends on the elastic constants and the nature of the stress variation with ϑ. It follows that $a > -1$ for the integral to be bounded. In other words, singular stress fields are acceptable if and only if the exponent on the stress components exceeds -1.

- The term $1-\lambda_2$ decreases rapidly and becomes negative for $2\alpha \geqslant 102.6°$ [2, p. 148]. For greater opening angles, mode II is no more singular, that is $\sigma_{ij}^{(II)}$ go to zero as $r \to 0$.

- The term $1-\lambda_1$ decreases more slowly and does not differ significantly from 0.5 for angles smaller than 50°. Furthermore, is always greater than zero.

- When $2\alpha = 0°$, the singularity is the same for both mode I and II ($1-\lambda_1 = 1-\lambda_2 = 0.5$).

From an engineering point of view, this means that mode I is more severe than mode II. In particular, the case $2\alpha = 0°$ is the worst case possible, since both modes are singular with the lowest exponent.

The stress field determined by Williams for a sharp V-shaped notch is then the following:

$$\begin{Bmatrix} \sigma_r^{(I)} \\ \sigma_\vartheta^{(I)} \\ \tau_{r\vartheta}^{(I)} \end{Bmatrix} = \lambda_1 r^{\lambda_1-1} \left\{ A_1 \begin{bmatrix} -(\lambda_1+1)\cos(\lambda_1+1)\vartheta \\ (\lambda_1+1)\cos(\lambda_1+1)\vartheta \\ (\lambda_1+1)\sin(\lambda_1+1)\vartheta \end{bmatrix} + A_2 \begin{bmatrix} -(\lambda_1-3)\cos(\lambda_1-1)\vartheta \\ (\lambda_1+1)\cos(\lambda_1-1)\vartheta \\ (\lambda_1-1)\sin(\lambda_1-1)\vartheta \end{bmatrix} \right\} \quad (1.44a)$$

$$\begin{Bmatrix} \sigma_r^{(II)} \\ \sigma_\vartheta^{(II)} \\ \tau_{r\vartheta}^{(II)} \end{Bmatrix} = \lambda_2 r^{\lambda_2-1} \left\{ A_3 \begin{bmatrix} -(\lambda_2+1)\sin(\lambda_2+1)\vartheta \\ (\lambda_2+1)\sin(\lambda_2+1)\vartheta \\ -(\lambda_2+1)\cos(\lambda_2+1)\vartheta \end{bmatrix} + A_4 \begin{bmatrix} -(\lambda_2-3)\sin(\lambda_2-1)\vartheta \\ (\lambda_2+1)\sin(\lambda_2-1)\vartheta \\ (\lambda_2-1)\cos(\lambda_2-1)\vartheta \end{bmatrix} \right\} \quad (1.44b)$$

while the displacements are:

$$\begin{Bmatrix} u_r^{(I)} \\ u_\vartheta^{(I)} \end{Bmatrix} = \frac{r^{\lambda_1}}{2G} \left\{ A_1 \begin{bmatrix} -(\lambda_1+1)\cos(\lambda_1+1)\vartheta \\ -(\lambda_1+1)\sin(\lambda_1+1)\vartheta \end{bmatrix} + A_2 \begin{bmatrix} (\kappa-\lambda_1)\cos(\lambda_1-1)\vartheta \\ (\kappa+\lambda_1)\sin(\lambda_1-1)\vartheta \end{bmatrix} \right\} \quad (1.45a)$$

$$\begin{Bmatrix} u_r^{(II)} \\ u_\vartheta^{(II)} \end{Bmatrix} = \frac{r^{\lambda_2}}{2G} \left\{ A_3 \begin{bmatrix} -(\lambda_2+1)\sin(\lambda_2+1)\vartheta \\ (\lambda_2+1)\cos(\lambda_2+1)\vartheta \end{bmatrix} + A_4 \begin{bmatrix} (\kappa-\lambda_2)\sin(\lambda_2-1)\vartheta \\ -(\kappa+\lambda_2)\cos(\lambda_2-1)\vartheta \end{bmatrix} \right\} \quad (1.45b)$$

where the superscripts are referring to mode I and II, respectively.

1.7.3 Alternative notation

The stress and displacement fields derived in the previous subsection are defined except for two constants, for both modes. Introducing the quantities [17]:

$$\chi_i = \frac{\sin(\lambda_i - 1)\gamma}{\sin(\lambda_i + 1)\gamma}, \quad \text{for } i = 1, 2 \tag{1.46}$$

into the first (respectively second) raw of the system in (1.42a) (respectively (1.42b)), we get the relations:

$$A_1 = -\chi_1 \frac{\lambda_1 - 1}{\lambda_1 + 1} A_4, \quad A_2 = -\chi_2 A_4 . \tag{1.47}$$

Using these definitions, the stresses turn out to be:

$$\begin{Bmatrix} \sigma_r^{(I)} \\ \sigma_\vartheta^{(I)} \\ \tau_{r\vartheta}^{(I)} \end{Bmatrix} = \lambda_1 A_2 r^{\lambda_1 - 1} \left\{ \begin{bmatrix} -(\lambda_1 - 3)\cos(\lambda_1 - 1)\vartheta \\ (\lambda_1 + 1)\cos(\lambda_1 - 1)\vartheta \\ (\lambda_1 - 1)\sin(\lambda_1 - 1)\vartheta \end{bmatrix} + \chi_1 (\lambda_1 - 1) \begin{bmatrix} \cos(\lambda_1 + 1)\vartheta \\ -\cos(\lambda_1 + 1)\vartheta \\ -\sin(\lambda_1 + 1)\vartheta \end{bmatrix} \right\} \tag{1.48a}$$

$$\begin{Bmatrix} \sigma_r^{(II)} \\ \sigma_\vartheta^{(II)} \\ \tau_{r\vartheta}^{(II)} \end{Bmatrix} = \lambda_2 A_4 r^{\lambda_2 - 1} \left\{ \begin{bmatrix} -(\lambda_2 - 3)\sin(\lambda_2 - 1)\vartheta \\ (\lambda_2 + 1)\sin(\lambda_2 - 1)\vartheta \\ (\lambda_2 - 1)\cos(\lambda_2 - 1)\vartheta \end{bmatrix} + \chi_2 (\lambda_2 + 1) \begin{bmatrix} \sin(\lambda_2 + 1)\vartheta \\ -\sin(\lambda_2 + 1)\vartheta \\ \cos(\lambda_2 + 1)\vartheta \end{bmatrix} \right\} \tag{1.48b}$$

while the displacements become:

$$\begin{Bmatrix} u_r^{(I)} \\ u_\vartheta^{(I)} \end{Bmatrix} = \frac{A_2 r^{\lambda_1}}{2G} \left\{ \begin{bmatrix} (\kappa - \lambda_1)\cos(\lambda_1 - 1)\vartheta \\ (\kappa + \lambda_1)\sin(\lambda_1 - 1)\vartheta \end{bmatrix} + \chi_1 (\lambda_1 - 1) \begin{bmatrix} \cos(\lambda_1 + 1)\vartheta \\ \sin(\lambda_1 + 1)\vartheta \end{bmatrix} \right\} \tag{1.49a}$$

$$\begin{Bmatrix} u_r^{(II)} \\ u_\vartheta^{(II)} \end{Bmatrix} = \frac{A_4 r^{\lambda_2}}{2G} \left\{ \begin{bmatrix} (\kappa - \lambda_2)\sin(\lambda_2 - 1)\vartheta \\ -(\kappa + \lambda_2)\cos(\lambda_2 - 1)\vartheta \end{bmatrix} + \chi_2 (\lambda_2 + 1) \begin{bmatrix} \sin(\lambda_2 + 1)\vartheta \\ -\cos(\lambda_2 + 1)\vartheta \end{bmatrix} \right\} . \tag{1.49b}$$

Some values of $\lambda_{1,2}$ and $\chi_{1,2}$ are reported in Table 1.2.

Table 1.2. Some values of $\lambda_{1,2}$ and $\chi_{1,2}$ [18].

2α (deg)	γ/π (rad)	λ_1	λ_2	χ_1	χ_2
0	1	0.5000	0.5000	1.000	1.000
15	23/24	0.5002	0.5453	1.017	0.981
30	11/12	0.5014	0.5982	1.071	0.921
45	7/8	0.5050	0.6597	1.166	0.814
60	5/6	0.5122	0.7309	1.312	0.658
90	3/4	0.5445	0.9085	1.841	0.219
120	2/3	0.6157	1.1489	3.004	−0.314
135	5/8	0.6736	1.3021	4.152	−0.569
150	7/12	0.7520	1.4858	6.357	−0.787
160	5/9	0.8187	1.6305	9.536	−0.898
170	19/36	0.9000	1.7989	18.913	−0.972

The stress field in proximity of the notch tip can be written also in terms of the *Notch Stress Intensity Factors* (NSIFs), whose definitions according to Gross and Mendelson are [13]:

$$K_1 = \lim_{r\to 0^+} \sqrt{2\pi}\, r^{1-\lambda_1}\, \sigma_\vartheta^{(I)}(\vartheta = 0)\,, \tag{1.50a}$$

$$K_2 = \lim_{r\to 0^+} \sqrt{2\pi}\, r^{1-\lambda_2}\, \tau_{r\vartheta}^{(II)}(\vartheta = 0)\,. \tag{1.50b}$$

These quantities depend both on the opening mode, through a stress component related to the mode considered, and the notch opening angle, through a Williams' eigenvalue; the eigenvalues determine also their units: $[K_{1,2}] = \text{MPa}\,\text{mm}^{1-\lambda_{1,2}}$. This fact has important practical consequences, as we are going to explain later on.

By using the definitions (1.50a) and (1.50b), the stresses can be written as [17]:

$$\begin{Bmatrix} \sigma_r^{(I)} \\ \sigma_\vartheta^{(I)} \\ \tau_{r\vartheta}^{(I)} \end{Bmatrix} = \frac{K_1\, r^{\lambda_1 - 1}}{\sqrt{2\pi}\,[(\lambda_1+1) - \chi_1\,(\lambda_1 - 1)]} \left\{ \begin{bmatrix} -(\lambda_1 - 3)\cos(\lambda_1 - 1)\vartheta \\ (\lambda_1+1)\cos(\lambda_1 - 1)\vartheta \\ (\lambda_1 - 1)\sin(\lambda_1 - 1)\vartheta \end{bmatrix} + \chi_1\,(\lambda_1 - 1) \begin{bmatrix} -\cos(\lambda_1+1)\vartheta \\ \cos(\lambda_1+1)\vartheta \\ \sin(\lambda_1+1)\vartheta \end{bmatrix} \right\} \tag{1.51a}$$

$$\begin{Bmatrix} \sigma_r^{(II)} \\ \sigma_\vartheta^{(II)} \\ \tau_{r\vartheta}^{(II)} \end{Bmatrix} = \frac{K_2\, r^{\lambda_2 - 1}}{\sqrt{2\pi}\,[(\lambda_2-1) + \chi_2\,(\lambda_2 + 1)]} \left\{ \begin{bmatrix} -(\lambda_2 - 3)\sin(\lambda_2 - 1)\vartheta \\ (\lambda_2+1)\sin(\lambda_2 - 1)\vartheta \\ (\lambda_2 - 1)\cos(\lambda_2 - 1)\vartheta \end{bmatrix} + \chi_2\,(\lambda_2 + 1) \begin{bmatrix} -\sin(\lambda_2+1)\vartheta \\ \sin(\lambda_2+1)\vartheta \\ -\cos(\lambda_2+1)\vartheta \end{bmatrix} \right\} \tag{1.51b}$$

and the displacements as:

$$\begin{Bmatrix} u_r^{(I)} \\ u_\vartheta^{(I)} \end{Bmatrix} = \frac{1}{2G} \frac{K_1 \, r^{\lambda_1}}{\sqrt{2\pi}\,[(\lambda_1+1)-\chi_1\,(\lambda_1-1)]} \Bigg\{ \begin{bmatrix} (\kappa-\lambda_1)\cos(\lambda_1-1)\vartheta \\ (\kappa+\lambda_1)\sin(\lambda_1-1)\vartheta \end{bmatrix} + \chi_1\,(\lambda_1-1) \begin{bmatrix} \cos(\lambda_1+1)\vartheta \\ \sin(\lambda_1+1)\vartheta \end{bmatrix} \Bigg\} \tag{1.52a}$$

$$\begin{Bmatrix} u_r^{(II)} \\ u_\vartheta^{(II)} \end{Bmatrix} = \frac{1}{2G} \frac{K_2 \, r^{\lambda_2}}{\sqrt{2\pi}\,[(\lambda_2-1)+\chi_2\,(\lambda_2+1)]} \Bigg\{ \begin{bmatrix} (\kappa-\lambda_2)\sin(\lambda_2-1)\vartheta \\ -(\kappa+\lambda_2)\cos(\lambda_2-1)\vartheta \end{bmatrix} + \chi_2\,(\lambda_2+1) \begin{bmatrix} \sin(\lambda_2+1)\vartheta \\ -\cos(\lambda_2+1)\vartheta \end{bmatrix} \Bigg\}. \tag{1.52b}$$

The definitions reported in equations (1.50a) and (1.50b) introduce two parameters which are very useful for engineering analyses. K_1 and K_2 do not have a closed form, but can be computed with great accuracy using a Finite Element code, and can be exploited to formulate failure criteria (see [18] for an application to welded joints).

We conclude the section with an observation: By setting a notch opening angle $2\alpha = 0°$, the stress and displacement fields of a crack are obtained, as Williams himself demonstrated in a later paper [35]. Because of the great practical relevance of these equations, first obtained by Westergaard following a different approach, they will be explicitly derived in subsection 1.8.4.[4]

1.8 METHOD OF COMPLEX VARIABLES

One of the major contributions to the mathematical theory of elasticity in the 20th century is related to the names of Kolosov and Muskhelishvili. Starting from the Airy stress function, they developed an original and extremely powerful method to solve the problems of plane elasticity through the use of complex variables. Without claiming to be exhaustive, we are going to describe the salient points of their theory, which will be then used for our purposes. The main reference for this section is [21, pp. 105–115].

1.8.1 Some definitions

We define a *complex variable* z and its *complex conjugate* $\bar{z}$ as:

$$z = x + iy, \quad \bar{z} = x - iy \tag{1.53}$$

4 It is interesting to notice that Westergaard's equations (1.92) are derived considering a *central* crack, while for $2\alpha > 0°$ Williams' equations necessarily describe the local field associated to an *edge* notch. The two systems coincide when $2\alpha = 0°$ because the boundary conditions on the stresses are applied at infinity.

where x (the real part) and y (the imaginary part) can be obtained through the expressions:

$$\begin{cases} x = \operatorname{Re} z = \dfrac{z+\bar{z}}{2} \\ y = \operatorname{Im} z = \dfrac{z-\bar{z}}{2}\,. \end{cases} \tag{1.54}$$

The *complex derivative* of a function $f(z)$ in a point $z_0 \in A$ ($A \subseteq \mathbb{C}$) is the limit of the difference quotient as z approaches z_0, just like in the real case. Using formulas:

$$f'(z_0) \coloneqq \lim_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0}\,. \tag{1.55}$$

If the limit thus defined exists, f is said to be a *holomorphic function*: These kind of functions has the property of analyticity, that is, the function is equal to its Taylor series in a neighbourhood of each point in its domain ($f \in C^\infty$).

By applying the chain rule, it is easy to determine the first order partial derivatives:

$$\begin{cases} \dfrac{\partial f(z)}{\partial x} = \dfrac{df(z)}{dz} = f'(z) \\ \dfrac{\partial f(z)}{\partial y} = i\,\dfrac{df(z)}{dz} = i\,f'(z)\,. \end{cases} \tag{1.56}$$

1.8.2 Cauchy-Riemann conditions

Let us suppose to have a complex function of the form:

$$f(z) = u(x, y) + i\,v(x, y)\,. \tag{1.57}$$

Its partial derivatives are easily obtained:

$$\begin{cases} \dfrac{\partial f(z)}{\partial x} = \dfrac{\partial u(x,y)}{\partial x} + i\,\dfrac{\partial v(x,y)}{\partial x} \\ \dfrac{\partial f(z)}{\partial y} = \dfrac{\partial u(x,y)}{\partial y} + i\,\dfrac{\partial v(x,y)}{\partial y}\,. \end{cases} \tag{1.58}$$

By defining $h \coloneqq z - z_0$ ($h \in \mathbb{C}$), it is possible to rewrite equation (1.55) as:

$$f'(z_0) = \lim_{\substack{h \to 0 \\ h \in \mathbb{C}}} \frac{f(z_0 + h) - f(z_0)}{h}\,. \tag{1.59}$$

If the limit exists, whether calculating it along the real axis or the imaginary axis must give the same result. Considering the x axis, we have:

$$\lim_{\substack{h \to 0 \\ h \in \mathbb{R}}} \frac{f(z_0 + h) - f(z_0)}{h} = \frac{\partial f}{\partial x}(z_0) \tag{1.60a}$$

while along the y axis, it is:

$$\lim_{\substack{h\to 0\\ h\in\mathbb{R}}} \frac{f(z_0+i\,h)-f(z_0)}{i\,h} = \frac{1}{i}\frac{\partial f}{\partial y}(z_0)\,. \tag{1.60b}$$

For what we have just said, it must be:

$$i\,\frac{\partial f}{\partial x}(z_0) = \frac{\partial f}{\partial y}(z_0) \tag{1.61}$$

or, in terms of u and v:

$$-\frac{\partial v}{\partial x} + i\,\frac{\partial u}{\partial x} = \frac{\partial u}{\partial y} + i\,\frac{\partial v}{\partial y}\,. \tag{1.62}$$

The only way for the derivative to be independent of the direction chosen to compute the limit is that real and imaginary parts in the two cases coincide:

$$\begin{cases} \dfrac{\partial u}{\partial x} = \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} = -\dfrac{\partial v}{\partial x}\,. \end{cases} \tag{1.63}$$

These two conditions are called CAUCHY-RIEMANN conditions after their discoverers. Calculating the mixed derivatives of u (respectively v) and summing them, thanks to SCHWARZ's theorem, one finds that

$$\nabla^2 u = \nabla^2 v = 0\,. \tag{1.64}$$

In words, *the real and imaginary parts of a holomorphic function are solutions of Laplace's equation*. They are therefore called *harmonic conjugates*.

1.8.3 Complex representation of stresses

In section 1.6, we demonstrated that a planar stress condition can be expressed in terms of the Airy stress function Φ, which automatically satisfies the equilibrium conditions. In the absence of body forces, Φ satisfies equation (1.27), here recalled:

$$\nabla^2\nabla^2\Phi = 0\,. \tag{1.27, rep.}$$

Writing $\nabla^2\Phi = P$, it follows that $\nabla^2 P = 0$, i.e. P is a harmonic function. It is therefore possible to define a function Q which is the harmonic conjugate of P, and a holomorphic function $f(z)$, such that $P = \operatorname{Re} f(z)$ and $Q = \operatorname{Im} f(z)$.

By integrating, one gets the function $\Psi(z)$:

$$\Psi(z) = \frac{1}{4}\int f(z)\,dz = p + i\,q \tag{1.65}$$

which is again a holomorphic function. It follows from Cauchy-Riemann conditions that:

$$\begin{cases} \dfrac{\partial p}{\partial x} = \dfrac{\partial q}{\partial y} = \dfrac{P}{4} \\ \dfrac{\partial p}{\partial y} = -\dfrac{\partial q}{\partial x} = -\dfrac{Q}{4}\,. \end{cases} \tag{1.66}$$

Now, let us define the function $p_1 := \Phi - px - qy$. For p_1 to be harmonic, the quantity

$$\begin{aligned} \nabla^2 p_1 &= \nabla^2\Phi - \left[\frac{\partial^2}{\partial x^2}(px+qy) + \frac{\partial^2}{\partial y^2}(px+qy)\right] \\ &= \quad P \quad - \left[x\,\nabla^2 p + y\,\nabla^2 q + 2\left(\frac{\partial p}{\partial x} + \frac{\partial q}{\partial y}\right)\right] \end{aligned} \tag{1.67}$$

must be zero. Because of the equalities in the first raw of system (1.66), P and the term in brackets erase each other. The previous condition then reads:

$$\begin{cases} x\,\nabla^2 p = 0 \\ y\,\nabla^2 q = 0\,. \end{cases} \tag{1.68}$$

Both equalities hold for every x and y, because p and q are solutions of Laplace's equation. Since as we demonstrated p_1 is harmonic, it is possible to define a new function χ:

$$\chi := p_1 + i\,q_1 \tag{1.69}$$

such that q_1 is the harmonic conjugate of p_1. If we now combine Ψ and χ in the following way:

$$H(z) := \bar{z}\,\Psi(z) + \chi(z) \tag{1.70}$$

we obtain the fundamental relation between these complex quantities and the Airy stress function:

$$\begin{aligned} 2\,\Phi &= 2\,\mathrm{Re}\{H(z)\} = H(z) + \overline{H(z)} \\ &= \bar{z}\,\Psi(z) + \chi(z) + z\,\overline{\Psi(z)} + \overline{\chi(z)}\,. \end{aligned} \tag{1.71}$$

By deriving equation (1.71) with respect to x and y, we obtain:

$$\begin{aligned} 2\,\frac{\partial\Phi}{\partial x} &= \quad \bar{z}\,\Psi'(z) + \Psi(z) + \chi'(z) + z\,\overline{\Psi'(z)} + \overline{\Psi(z)} + \overline{\chi'(z)} \\ 2\,\frac{\partial\Phi}{\partial y} &= i\left[\bar{z}\,\Psi'(z) - \Psi(z) + \chi'(z) - z\,\overline{\Psi'(z)} + \overline{\Psi(z)} - \overline{\chi'(z)}\right] \end{aligned} \tag{1.72}$$

or, equivalently:

$$\frac{\partial\Phi}{\partial x} + i\,\frac{\partial\Phi}{\partial y} = \Psi(z) + z\,\overline{\Psi'(z)} + \overline{\chi'(z)} \tag{1.73}$$

By deriving equation (1.73) with respect to x and y, and multiplying by the imaginary unit i the second expression, we find:

$$\begin{aligned}\frac{\partial^2\Phi}{\partial x^2}+i\,\frac{\partial^2\Phi}{\partial x\,\partial y} &= \Psi'(z)+\overline{\Psi'(z)}+z\,\overline{\Psi''(z)}+\overline{\chi''(z)}\\ -\frac{\partial^2\Phi}{\partial y^2}+i\,\frac{\partial^2\Phi}{\partial x\,\partial y} &= -\Psi'(z)-\overline{\Psi'(z)}+z\,\overline{\Psi''(z)}+\overline{\chi''(z)}\,.\end{aligned} \tag{1.74}$$

By summation and subtraction of the equations thus found, we obtain the so-called *fundamental stress combinations* [25, p. 268]:

$$\begin{cases}\sigma_x+\sigma_y = 2\left[\Psi'(z)+\overline{\Psi'(z)}\right] = 4\,\mathrm{Re}\,\Psi'(z)\\ \sigma_y-\sigma_x+2i\,\tau_{xy} = 2\left[\bar{z}\,\Psi''(z)+\overline{\chi''(z)}\right].\end{cases} \tag{1.75}$$

Although we do not describe explicitly the procedure to derive such relation, it can demonstrated that the planar displacements are subject to the condition [25, p. 267]:

$$2G\,(u+i\,v)=\kappa\,\Psi(z)-z\,\overline{\Psi'(z)}-\overline{\chi'(z)} \tag{1.76}$$

where κ is the Kolosov's constant defined in subsection 1.7.1.

Following Muskhelishvili's procedure, the final step is to define a new complex function:

$$\varphi(z) \coloneqq \chi'(z) \tag{1.77}$$

so that the planar stresses become:

$$\begin{cases}\sigma_x+\sigma_y = 2\left[\Psi'(z)+\overline{\Psi'(z)}\right] = 4\,\mathrm{Re}\,\Psi'(z)\\ \sigma_y-\sigma_x+2i\,\tau_{xy} = 2\left[\bar{z}\,\Psi''(z)+\overline{\varphi'(z)}\right]\end{cases} \tag{1.78}$$

and the displacement field is:

$$2G\,(u+i\,v)=\kappa\,\Psi(z)-z\,\overline{\Psi'(z)}-\overline{\varphi(z)}\,. \tag{1.79}$$

We therefore conclude that, according to the method of complex variables, *the exact stresses and displacements in plane elasticity can be completely determined once that two proper complex functions $\Psi(z)$, $\varphi(z)$ are defined.*

1.8.4 Westergaard's equations

We will now use the method of complex variables to obtain the well-known Westergaard's equations for a central crack in an infinite plate, subjected to mode I. The complex functions used in this problem are the following [31, p. 26]:

$$\Psi'(z) = \tfrac{1}{2}\, Z(z), \quad \varphi'(z) = -\tfrac{1}{2}\, z\, Z'(z)\,. \tag{1.80}$$

The relations in (1.78) then become:

$$\begin{cases} \sigma_x + \sigma_y = Z(z) + \overline{Z(z)} = 2\,\mathrm{Re}\, Z(z) \\ \sigma_y - \sigma_x + 2i\,\tau_{xy} = (\bar{z} - z)\, Z'(z) = 2y\left[\mathrm{Im}\, Z'(z) - i\,\mathrm{Re}\, Z'(z)\right]. \end{cases} \tag{1.81}$$

With a simple integration by parts, it is found that [31, p. 26]

$$\varphi(z) = \tfrac{1}{2}\, Z^{\star}(z) - \tfrac{1}{2}\, z\, Z(z) \tag{1.82}$$

where $Z^{\star}(z) := \int Z(z)\, dz$. Hence, equation (1.79) turns out to be:

$$\begin{aligned} 2G\,(u + i\,v) = {} & \tfrac{1}{2}(\kappa - 1)\,\mathrm{Re}\, Z^{\star}(z) - y\,\mathrm{Im}\, Z(z) \\ & + i\left[\tfrac{1}{2}(\kappa + 1)\,\mathrm{Im}\, Z^{\star}(z) - y\,\mathrm{Re}\, Z(z)\right]. \end{aligned} \tag{1.83}$$

In order to determine explicitly stresses and displacements, it is necessary to define $Z(z)$. This stress function is called *Westergaard function* after its discoverer, and in this case assumes the form [33]:

$$Z(z) = \frac{\sigma \cdot z}{\sqrt{z^2 - a^2}} \tag{1.84}$$

where σ is the tensile stress acting at an infinite distance from the crack and a is the crack half length. Consequently, $Z^{\star}(z)$ reads:

$$Z^{\star}(z) = \sigma\sqrt{z^2 - a^2}\,. \tag{1.85}$$

By observing Figure 1.3, adopting the polar form for complex quantities, one can define the following relations:

$$z = r\, e^{i\vartheta}, \quad \sqrt{z^2 - a^2} = \sqrt{r_1 r_2}\; e^{i\bar{\vartheta}} \tag{1.86}$$

where $\bar{\vartheta} := \frac{1}{2}(\vartheta_1 + \vartheta_2)$. Therefore, the stresses turn out to be:

$$\begin{cases} \sigma_x + \sigma_y = \dfrac{\sigma\, r}{\sqrt{r_1 r_2}} \cos(\vartheta - \bar{\vartheta}) \\ \sigma_y - \sigma_x + 2i\,\tau_{xy} = 2\,\dfrac{\sigma\, a^2}{(r_1 r_2)^{3/2}}\, r_1 \sin\vartheta_1 \left[\sin(3\bar{\vartheta}) + i\,\cos(3\bar{\vartheta})\right] \end{cases} \tag{1.87}$$

while the displacements are:

$$\begin{aligned} 2G\,(u + i\,v) = {} & \tfrac{1}{2}(\kappa - 1)\,\sigma\sqrt{r_1 r_2}\cos\bar{\vartheta} - r_1 \sin\vartheta_1 \frac{\sigma\, r}{\sqrt{r_1 r_2}} \sin(\vartheta - \bar{\vartheta}) \\ & + i\left[\tfrac{1}{2}(\kappa + 1)\,\sigma\sqrt{r_1 r_2}\sin\bar{\vartheta} - r_1 \sin\vartheta_1 \frac{\sigma\, r}{\sqrt{r_1 r_2}} \cos(\vartheta - \bar{\vartheta})\right]. \end{aligned} \tag{1.88}$$

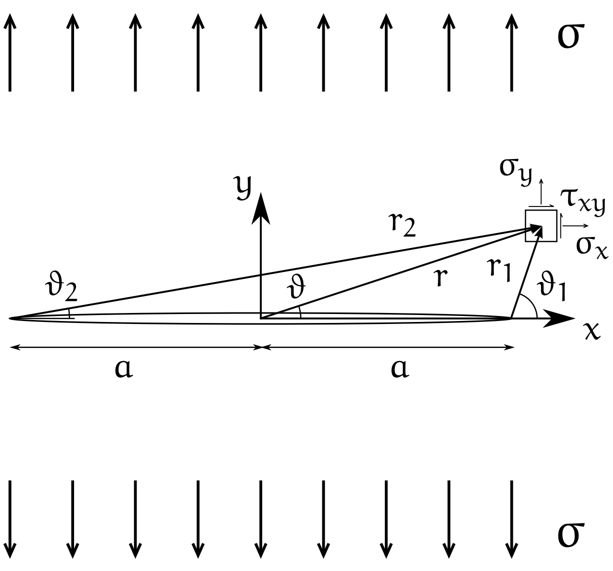


Figure 1.3. Configuration of the crack problem.

If we want just to determine the asymptotic fields, i.e. the ones in proximity of the crack tip, we can introduce the following approximations:

$$\begin{aligned} r &\approx a, & \vartheta &\approx 0 \\ r_2 &\approx 2a, & \vartheta_2 &\approx 0\,. \end{aligned} \tag{1.89}$$

The shear component τ_{xy} is the imaginary part of the second equation:

$$\tau_{xy} \approx \frac{\sigma\sqrt{a}}{\sqrt{2\,r_1}} \sin\frac{\vartheta_1}{2} \cos\frac{\vartheta_1}{2} \cos\frac{3\vartheta_1}{2} \tag{1.90}$$

while the normal stresses are obtained using the relations:

$$\begin{cases} \sigma_x + \sigma_y \approx 2\,\dfrac{\sigma\sqrt{a}}{\sqrt{2\,r_1}} \cos\dfrac{\vartheta_1}{2} \\ \sigma_y - \sigma_x \approx \dfrac{\sigma\sqrt{a}}{\sqrt{2\,r_1}} \sin\dfrac{\vartheta_1}{2} \cos\dfrac{\vartheta_1}{2} \sin\dfrac{3\vartheta_1}{2}\,. \end{cases} \tag{1.91}$$

We conclude that the solution is

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{\sigma\sqrt{a}}{\sqrt{2\,r}} \begin{Bmatrix} \cos\frac{\vartheta}{2}\left[1-\sin\frac{\vartheta}{2}\sin\frac{3\vartheta}{2}\right] \\ \cos\frac{\vartheta}{2}\left[1+\sin\frac{\vartheta}{2}\sin\frac{3\vartheta}{2}\right] \\ \sin\frac{\vartheta}{2}\cos\frac{\vartheta}{2}\cos\frac{3\vartheta}{2} \end{Bmatrix} \tag{1.92}$$

for the stresses and

$$\begin{aligned} 2G\,(u+i\,v) \approx \sigma\sqrt{a}\sqrt{\frac{r_1}{2}} \Bigg\{ &(\kappa-1)\,\cos\frac{\vartheta_1}{2} + \sin\vartheta_1 \sin\frac{\vartheta_1}{2} \\ &+ i\left[(\kappa+1)\,\sin\frac{\vartheta_1}{2} - \sin\vartheta_1\cos\frac{\vartheta_1}{2}\right]\Bigg\} \end{aligned} \tag{1.93}$$

for the displacements. Exploiting again the trigonometric relation $\sin\vartheta = 2\sin\frac{\vartheta}{2}\cos\frac{\vartheta}{2}$, the latter can be also rewritten as:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \frac{\sigma\sqrt{a}}{2G}\sqrt{\frac{r}{2}} \begin{Bmatrix} \cos\frac{\vartheta}{2}\left[\kappa - 1 + 2\sin^2\frac{\vartheta}{2}\right] \\ \sin\frac{\vartheta}{2}\left[\kappa + 1 - 2\cos^2\frac{\vartheta}{2}\right] \end{Bmatrix}. \tag{1.94}$$

The subscript was omitted, since the coordinate system was moved with a rigid translation to the crack tip.

These are the original equations derived by Westergaard [33]. Irwin modified them further by introducing the concept of the *Stress Intensity Factor* (SIF), which reads [15]:

$$K_I = \lim_{r\to 0^+} \sqrt{2\pi r}\,\sigma_y(\vartheta = 0)\,. \tag{1.95}$$

As previously stated, the SIF was then generalized to the notches by other Authors [13]. For the crack problem, K_I has a closed form. In fact, introducing σ_y as given by equation (1.92) in the previous definition, one obtains:

$$K_I = \sigma\sqrt{\pi a}\,. \tag{1.96}$$

Equation (1.96) relates the local field parameter K_I to the nominal stress σ and the crack length a. The stress field then becomes:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{K_I}{\sqrt{2\pi r}} \begin{Bmatrix} \cos\frac{\vartheta}{2}\left[1 - \sin\frac{\vartheta}{2}\sin\frac{3\vartheta}{2}\right] \\ \cos\frac{\vartheta}{2}\left[1 + \sin\frac{\vartheta}{2}\sin\frac{3\vartheta}{2}\right] \\ \sin\frac{\vartheta}{2}\cos\frac{\vartheta}{2}\cos\frac{3\vartheta}{2} \end{Bmatrix} \tag{1.97}$$

while the displacements are:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \frac{K_I}{2G}\sqrt{\frac{r}{2}} \begin{Bmatrix} \cos\frac{\vartheta}{2}\left[\kappa - 1 + 2\sin^2\frac{\vartheta}{2}\right] \\ \sin\frac{\vartheta}{2}\left[\kappa + 1 - 2\cos^2\frac{\vartheta}{2}\right] \end{Bmatrix}. \tag{1.98}$$

It is worth noticing that the second equation in (1.94) allows an alternative definition of the SIF [1, p. 559]:

$$K_I = \lim_{r\to 0^+} \sqrt{\frac{2\pi}{r}\frac{E'}{4}}\, v(\vartheta = \pi) \tag{1.99}$$

where E' is the effective Young's modulus, defined in Table 1.1. The displacement-based definition of K_I is extremely useful for its numerical estimation, since $v \sim \sqrt{r}$ as $r \to 0$ and is therefore more easy to compute than the stresses, which are singular near the crack tip.

2 THE SED CRITERION

2.1 INTRODUCTION

Now that the necessary theoretical background has been introduced, we can describe the SED criterion as formulated by Lazzarin and Zambardi [19]. For some aspects, it can be seen as an evolution of a previous criterion, based on the evaluation of the Notch Stress Intensity Factors [18]. The reasons for a change are twofold [16]:

- The NSIFs' dimensions depend on the notch opening angle, as shown in subsection 1.7.3. It is therefore not possible to compare them directly when non-similar geometries are considered.
- The volume dominated by the singular stress field decreases with the thickness. When low thicknesses are considered (for example the metal sheets extensively used in the automotive industry, whose thickness is less than 1 mm), it is necessary to take in account also non-singular terms, which cannot be predicted by Williams' asymptotic solution.

As the name suggests, the SED criterion is based on the evaluation of the strain energy density. The use of this quantity allows to overcome both limits of the NSIFs, since (i) it has always the dimensions of $\mathrm{N\,mm/mm^3}$ and (ii) can be computed numerically by summing the contributions of both singular and non-singular terms.

The idea that the quantity controlling the failure of a solid is the strain energy density was first suggested by Beltrami [12, p. 196]. Instead of considering the strain energy density of the entire structure, in the SED criterion this quantity is computed locally, in the zones which are subject to singularities or strong gradients, and averaged on a volume that depends on the material used, according to the concept of *control volume* first proposed by Neuber and retrieved by Peterson [27, p. 197]. This volume is defined by a characteristic radius, whose order of magnitude is usually 0.1 to 1 mm.

2.2 BASIC EQUATIONS

In the principal coordinate system, where all the shear stress components are zero, the strain energy density is [30, p. 148]:

$$W = \frac{1}{2E}\left[\sigma_1^2 + \sigma_2^2 + \sigma_3^2 - 2\nu\left(\sigma_1\,\sigma_2 + \sigma_2\,\sigma_3 + \sigma_1\,\sigma_3\right)\right]. \tag{2.1}$$

If we consider any non-principal polar coordinate system, the SED turns out to be [30, p. 148]:

$$W = \frac{1}{2E}\left[\sigma_r^2 + \sigma_\vartheta^2 + \sigma_z^2 - 2\nu\left(\sigma_r\,\sigma_\vartheta + \sigma_r\,\sigma_z + \sigma_\vartheta\,\sigma_z\right) + 2\left(1+\nu\right)\tau_{r\vartheta}^2\right]. \tag{2.2}$$

Since we are working under the generalized plane elasticity hypothesis, we can exploit the effective elastic constants reported in Table 1.1 to rewrite the strain energy density in a more handy form:

$$W = \frac{1}{2E'}\left[\sigma_r^2 + \sigma_\vartheta^2 - 2\nu'\sigma_r\,\sigma_\vartheta + 2\left(1+\nu'\right)\tau_{r\vartheta}^2\right]. \tag{2.3}$$

On the basis of the superposition principle, the singular stress field due to the V-shaped notch can be thus expressed (see Figure 2.1):

$$\begin{Bmatrix}\sigma_r\\ \sigma_\vartheta\\ \tau_{r\vartheta}\end{Bmatrix} = K_1 r^{\lambda_1-1}\begin{Bmatrix}\tilde{\sigma}_r^{(I)}\\ \tilde{\sigma}_\vartheta^{(I)}\\ \tilde{\tau}_{r\vartheta}^{(I)}\end{Bmatrix} + K_2 r^{\lambda_2-1}\begin{Bmatrix}\tilde{\sigma}_r^{(II)}\\ \tilde{\sigma}_\vartheta^{(II)}\\ \tilde{\tau}_{r\vartheta}^{(II)}\end{Bmatrix}. \tag{2.4}$$

This form highlights the most relevant parameters for the stresses, that are the NSIFs and the singular terms $r^{\lambda_{1,2}-1}$; the trigonometric terms are collected into the angular functions $\tilde{\sigma}_r$, $\tilde{\sigma}_\vartheta$, and $\tilde{\tau}_{r\vartheta}$.

Using these equations, it is possible to determine the contributions of mode I, mode II, and mixed mode to the SED:

$$\begin{aligned}
W_1(r,\vartheta) &= \frac{K_1^2\, r^{2(\lambda_1-1)}}{2E'}\Big[\tilde{\sigma}_r^{(I)^2} + \tilde{\sigma}_\vartheta^{(I)^2} - 2\nu'\tilde{\sigma}_r^{(I)^2}\,\tilde{\sigma}_\vartheta^{(I)^2}\\
&\quad + 2\left(1+\nu'\right)\tilde{\tau}_{r\vartheta}^{(I)^2}\Big],\\
W_2(r,\vartheta) &= \frac{K_2^2\, r^{2(\lambda_2-1)}}{2E'}\Big[\tilde{\sigma}_r^{(II)^2} + \tilde{\sigma}_\vartheta^{(II)^2} - 2\nu'\tilde{\sigma}_r^{(II)^2}\,\tilde{\sigma}_\vartheta^{(II)^2}\\
&\quad + 2\left(1+\nu'\right)\tilde{\tau}_{r\vartheta}^{(II)^2}\Big],\\
W_{12}(r,\vartheta) &= \frac{K_1 K_2\, r^{\lambda_1+\lambda_2-2}}{E'}\Big[\tilde{\sigma}_r^{(I)}\,\tilde{\sigma}_r^{(II)} + \tilde{\sigma}_\vartheta^{(I)}\,\tilde{\sigma}_\vartheta^{(II)}\\
&\quad - 2\nu'\left(\tilde{\sigma}_r^{(I)}\,\tilde{\sigma}_\vartheta^{(II)} + \tilde{\sigma}_\vartheta^{(I)}\,\tilde{\sigma}_r^{(II)}\right) + 2\left(1+\nu'\right)\tilde{\tau}_{r\vartheta}^{(I)}\,\tilde{\tau}_{r\vartheta}^{(II)}\Big].
\end{aligned} \tag{2.5}$$

In order to get the local strain energy, one has to integrate the components thus found over the area A:

$$\mathcal{U}(R) = \int_A W\,dA = \int_0^R\int_{-\gamma}^{+\gamma}\left[W_1(r,\vartheta) + W_2(r,\vartheta) + W_{12}(r,\vartheta)\right] r\,dr d\vartheta\,. \tag{2.6}$$

Since the term W_{12} is a combination of the two modes, and since they are symmetric respect to the notch bisector, its integral is zero. Therefore, the local strain energy turns out to be:

$$\mathcal{U}(R) = \frac{1}{E}\left[\frac{I_1(\gamma)}{4\,\lambda_1}\,K_1^2\,R^{2\lambda_1} + \frac{I_2(\gamma)}{4\,\lambda_2}\,K_2^2\,R^{2\lambda_2}\right]. \tag{2.7}$$

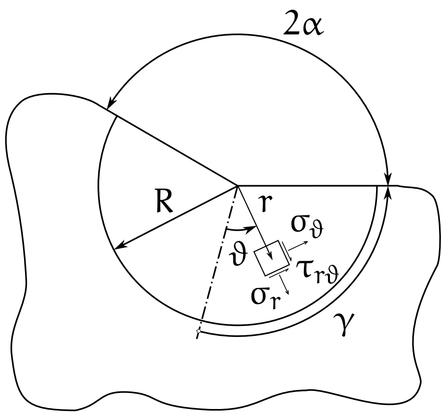


Figure 2.1. Polar stress components for an element inside the control volume [19].

where I_1 and I_2 are:

$$\begin{aligned} I_1 &= \int_{-\gamma}^{+\gamma} \left[\tilde{\sigma}_r^{(I)^2} + \tilde{\sigma}_\vartheta^{(I)^2} - 2\nu' \tilde{\sigma}_r^{(I)^2} \tilde{\sigma}_\vartheta^{(I)^2} + 2\,(1+\nu')\, \tilde{\tau}_{r\vartheta}^{(I)^2} \right] r \, dr d\vartheta \,, \\ I_2 &= \int_{-\gamma}^{+\gamma} \left[\tilde{\sigma}_r^{(II)^2} + \tilde{\sigma}_\vartheta^{(II)^2} - 2\nu' \tilde{\sigma}_r^{(II)^2} \tilde{\sigma}_\vartheta^{(II)^2} + 2\,(1+\nu')\, \tilde{\tau}_{r\vartheta}^{(II)^2} \right] r \, dr d\vartheta \,. \end{aligned} \tag{2.8}$$

These integrals depend both on the notch opening angle and the Poisson's ratio. They are reported in Table 2.1 for some characteristic angles, assuming $\nu = 0.3$ (which is a typical value for structural steels).

The local strain energy density is obtained by averaging $\mathcal{U}(R)$ on the area of integration:

$$\mathcal{SED} = \frac{\mathcal{U}(R)}{\gamma R^2} = \frac{1}{E} \left[e_1(2\alpha)\, K_1^2 \, R^{2(\lambda_1 - 1)} + e_2(2\alpha)\, K_2^2 \, R^{2(\lambda_2 - 1)} \right] \tag{2.9}$$

where $e_i(2\alpha) = \frac{I_i(\gamma)}{4\lambda_i \gamma}$, for $i = 1, 2$. The expression thus obtained has general validity and relates $\mathcal{SED}$ to the notch geometry and the radius R, which is thought to be a property of the material as welded.

It is interesting to point out some considerations:

- The left-hand side of equation (2.9) plays the same role of the equivalent stress defined in the classical failure criteria (Tresca, von Mises, etc.): In fact, this quantity can be easily computed with a simple tensile test, allowing to gain information about the quantities on the right-hand side, which may refer to complex loading conditions.

- Under simple stress conditions, $\mathcal{SED}$ can be directly related to the nominal stresses, that are traditionally used in machine design; the energetic approach allows to relate them with fracture mechanics parameters such as the NSIFs, thus building a connection between the two design procedures.

Table 2.1. Some values of the integrals I_1 and I_2 [19].

2α	γ/π	Plane stress		Plane strain	
(deg)	(rad)	$I_1(\gamma)$	$I_2(\gamma)$	$I_1(\gamma)$	$I_2(\gamma)$
0	1	1.0250	2.3250	0.8450	2.1450
15	23/24	1.0216	2.1608	0.8431	2.0087
30	11/12	1.0108	2.0091	0.8366	1.8810
45	7/8	0.9918	1.8688	0.8247	1.7610
60	5/6	0.9642	1.7385	0.8066	1.6479
90	3/4	0.8826	1.5018	0.7504	1.4379
120	2/3	0.7701	1.2887	0.6687	1.2437
135	5/8	0.7058	1.1883	0.6201	1.1505
150	7/12	0.6386	1.0908	0.5678	1.0590
160	5/9	0.5930	1.0269	0.5315	0.9986
170	19/36	0.5481	0.9635	0.4957	0.9383

- Equation (2.9) was derived under the linear elastic hypothesis, i.e. neglecting the plasticity effects that occur in the proximity of the notch tip when ductile materials are involved (the so-called *small scale yielding* condition). A key point of the SED criterion is that, due to (i) the alterations induced locally by the process of joining and (ii) the experimental evidences of elastic behaviour in high cycle fatigue of metals, it is legitimate to assume a brittle behaviour for the material, and therefore to use the relation previously derived.[1]

2.3 FORMULATION OF THE CRITERION

After these preliminaries, we can formulate the failure hypothesis:

> *According to the* SED criterion, *the fatigue failure of a welded joint weakened by a V-shaped sharp notch occurs when the strain energy density averaged over a material-dependent volume reaches a critical value.*

Speaking with formulas, the safety condition is:

$$\Delta\mathcal{SED} \leqslant \Delta\mathcal{SED}_C \tag{2.10}$$

where the subscript C indicates the critical value of a quantity (i.e. the one that induces the failure initiation) and the symbol Δ is used to highlight that only ranges of the quantities are considered.[2]

1 Since the only requirement in terms of material is a linear elastic behaviour until rupture, the criterion has more general validity and can be applied to other situations, such as the assessment of static strength for purely brittle materials [19].

2 In the classical approach of mechanical design, the fatigue behaviour is described in terms of stress range $\Delta\sigma = \sigma_{max} - \sigma_{min}$ and stress ratio $S = \frac{\sigma_{min}}{\sigma_{max}}$ (see e.g. [27, pp. 59–62]).

In order to use the criterion, we have to determine the characteristic radius R, which can be obtained for a particular (and possibly well-documented) case. The Authors' original choice fell on the mode I-dominated fatigue failure of a 135°-notched welded joint, due to the big amount of experimental data available in literature for this configuration [19]. Equation (2.9) then becomes:

$$\Delta\,\mathcal{SED} = \frac{I_1(\gamma)}{4\,\lambda_1\gamma}\,\frac{\Delta K_1^2}{E}\,R^{2(\lambda_1-1)}\,. \tag{2.11}$$

A key point in the arguments of the Authors is the following [19]: While R is a characteristic quantity for a *welded* material, the critical strain energy density is thought to be a property of the *non-welded* metal. Hence, by considering a fatigue tensile test of non-welded metal sheets, for which the assumption of uniform stress field is plausible, the critical strain energy density reads:

$$\Delta\,\mathcal{SED}_C \approx \frac{\Delta\sigma_A^2}{2E} \tag{2.12}$$

where the subscript A indicates the category of the structural details, i.e. its allowed fatigue life at 2×10^6 cycles, as Eurocode 3 states [9]. Upon substitution of $\Delta\,\mathcal{SED}$ with $\Delta\,\mathcal{SED}_C$, we get the critical NSIF:

$$\Delta K_{1C} = \sqrt{\frac{2\lambda_1\gamma}{I_1(\gamma)}}\,\Delta\sigma_A\,R^{1-\lambda_1} = f_1(2\alpha)\,\Delta\sigma_A\,R^{1-\lambda_1} \tag{2.13}$$

where f_1 is a function of the opening angle. Therefore, the expression for the radius R is the following:

$$R = \left(\frac{\Delta K_{1C}}{f_1(2\alpha)\,\Delta\sigma_A}\right)^{\frac{1}{1-\lambda_1}}\,. \tag{2.14}$$

With (i) a fatigue life $\Delta\sigma_A = 160\,\text{MPa}$ for $S = 0$, as reported by Eurocode 3 [9], and (ii) a critical NSIF $\Delta K_{1C} = 214\,\text{MPa}\,\text{mm}^{0.326}$ for a probability of survivance P.S. $= 97.7\%$, equation (2.14) gives $R = 0.265\,\text{mm}$ [19]. In some recent papers [16, 20], in order to determine more accurately the influence of the welding process, the fatigue tensile test was conducted on a butt ground welded joint, mechanically polished to remove any stress concentration effect. Moreover, the number of cycles was increased to 5×10^6, which according to Eurocode 3 has to be considered the fatigue limit of metals under constant amplitude load histories [9]. The new data are $\Delta\sigma_D = 155\,\text{MPa}$ at $S = 0$ and $\Delta K_{1C} = 211\,\text{MPa}\,\text{mm}^{0.326}$ for a P.S. $= 97.7\%$, and the radius predicted by equation (2.14) is $R = 0.28\,\text{mm}$.

For our analyses, unless otherwise specified, we sat $R = 0.3\,\text{mm}$, so to allow the comparison with some values of $\mathcal{SED}$ previously computed [8].

3 NUMERICAL ANALYSIS

3.1 INTRODUCTION

We already said in the introduction that most of present-day fracture mechanics-based failure criteria are dealing more or less markedly with numerical analysis. The main reason is that this branch of mathematics presents itself as a practical and reliable way to compute the local quantities which, according to fracture mechanics, are governing the structural damage. One of the most widespread techniques adopted by numerical fracture mechanics to compute rapidly and accurately such quantities is certainly the Finite Element Analysis (FEA), whose main concepts are now briefly discussed.

3.2 THE FINITE ELEMENT METHOD

The Finite Element Analysis is a tool extensively used in structural engineering for design purposes. Without claiming to be exhaustive, we are going to outline briefly the fundamental concepts at the basis of the Finite Element Method (FEM).

3.2.1 Differential formulation

The Finite Element Method is an extremely powerful technique that allows to obtain approximate solutions of mathematical models described by partial differential equations on continuous domains. In continuum mechanics, an important class of problems can be expressed in terms of elliptic PDEs, whose general formulation on a two-dimensional domain is [3, p. 105]:

$$A(x,y)\,\frac{\partial^2 u}{\partial x^2} + 2B(x,y)\,\frac{\partial^2 u}{\partial x\,\partial y} + C\,(x,y)\frac{\partial^2 u}{\partial y^2} = \varphi\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) \tag{3.1}$$

where $B^2 - AC < 0$. For example, as pointed out in subsection 1.5.1, the elastostatic problem is governed by a set of three linear partial differential equations and the prescribed boundary conditions.

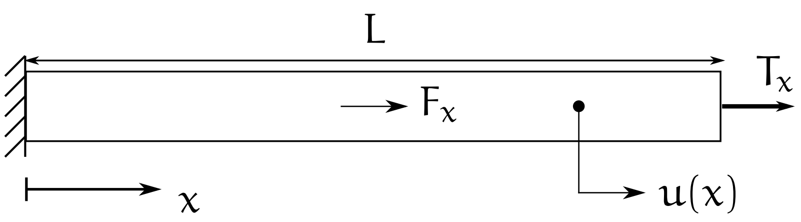


Figure 3.1. One-dimensional bar subjected to a body load F_x and an end stress T_x (*adapted from* [3, p. 109]).

3.2.2 Variational formulation

The problem (3.1) can be expressed in an alternative form, on the basis of the physics which governs it. In this case, instead of solving directly a differential equation, we seek an expression for the total potential associated to the physical system and we impose its stationarity. In mathematical terms, the condition of stationarity of a functional $F(v(x), v'(x), \dots, v^{(p)}(x))$ is expressed through its *first variation*, thus defined [3, p. 111]:

$$\delta F = \lim_{\varepsilon \to 0} \frac{F[v + \varepsilon\eta, v' + \varepsilon\eta', \dots, v^{(p)} + \varepsilon\eta^{(p)}] - F[v, v', \dots, v^{(p)}]}{\varepsilon} \tag{3.2}$$

where both $v(x)$ and $\eta(x)$ depend on x, while ε is a constant. Let us suppose $\eta(x)$ to be an arbitrary but sufficiently smooth function which is zero at the essential boundary conditions. We call it a *variation* in v and we write $\eta(x) = \delta v(x)$. We then notice that, under these hypothesis, equation (3.2) reads [3, p. 111]:

$$\delta F = \frac{\partial F}{\partial v}\delta v + \frac{\partial F}{\partial(dv/dx)}\delta\left(\frac{dv}{dx}\right) + \dots + \frac{\partial F}{\partial(d^p v/dx^p)}\delta\left(\frac{d^p v}{dx^p}\right) \tag{3.3}$$

that is, the variational operator $\delta(\cdot)$ acts like the differential operator with respect to the variables v, dv/dx, …, $d^p v/dx^p$.

That said, indicating the total potential energy with Π, we can equivalently express the equilibrium condition through the equation:

$$\delta\Pi(u) = 0 \tag{3.4}$$

which is called *variational formulation*, while Π is the *functional* of the problem. The condition (3.4) must be coupled with the essential or Dirichlet boundary conditions, that specify the values that the solution assumes at the boundary of the domain. Comparing equations (3.1) and (3.4), one may think that the adoption of one method respect to the other could lead to different results. With the next example we want to show that the two formulations are, in all respects, identical (see the example in [3, pp. 112–113] and following).

Let us consider the static response of the one-dimensional elastic bar shown in Figure 3.1. By truncating to the first order the term $\sigma A\big|_{x+dx}$, the equilibrium of the forces in the x direction of a typical differential element reads (see Figure 3.2):

$$\sigma A\big|_x + A \left.\frac{d\sigma}{dx}\right|_x dx + F_x\, dx - \sigma A\big|_x = 0\,. \tag{3.5}$$

Introducing the constitutive relation:

$$\sigma = E\,\frac{du}{dx} \tag{3.6}$$

we can write the differential formulation of the problem in its entirety [3, p. 124]:

$$EA\,\frac{d^2u}{dx^2} + F_x = 0 \qquad \text{in the bar} \tag{3.7a}$$

$$u\big|_{x=0} = 0, \qquad EA\left.\frac{du}{dx}\right|_{x=L} = T_x\,. \tag{3.7b}$$

The functional associated to this problem is [3, p. 125]:

$$\Pi(u) = \int_0^L \frac{1}{2}\,EA\left(\frac{du}{dx}\right)^2 dx - \int_0^L u\,F_x\,dx - u_L T_x \tag{3.8}$$

where $u_L := u\big|_{x=L}$ and $u_0 := u\big|_{x=0} = 0$. By imposing the condition (3.4), we get:

$$\delta\Pi(u) = \int_0^L \left(EA\,\frac{du}{dx}\right)\delta\left(\frac{du}{dx}\right) dx - \int_0^L \delta u\,F_x\,dx - \delta u_L T_x = 0\,. \tag{3.9}$$

Integrating by parts and using the equality $\delta\left(\frac{du}{dx}\right) = \frac{d}{dx}\delta u$, we obtain the equation:

$$\underbrace{-\int_0^L \left(EA\,\frac{d^2u}{dx^2} + F_x\right)\delta u\,dx}_{①} + \underbrace{\left[EA\left.\frac{du}{dx}\right|_{x=L} - T_x\right]\delta u_L}_{②} \underbrace{- EA\left.\frac{du}{dx}\right|_{x=0}\delta u_0}_{③} = 0\,. \tag{3.10}$$

Since there cannot be variations on the prescribed boundary conditions, it must be $\delta u_0 = 0$, and term ③ disappears. Considering now term ②, we notice that δu_L is completely arbitrary. Therefore, we can

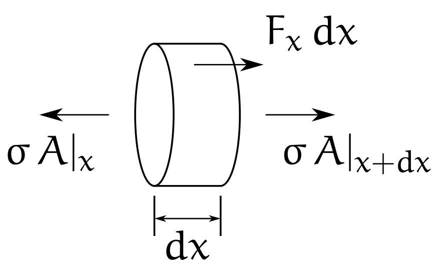


Figure 3.2. Equilibrium of a typical differential element of the bar.

assume δu to be zero in all the domain except at $x = L$. Since the condition is to hold for any δu, it must be:

$$EA \left.\frac{du}{dx}\right|_{x=L} = T_x \tag{3.11}$$

which is the second of the equations in (3.7b), corresponding to the natural or NEUMANN boundary condition. Conversely, the argument that $\delta u \neq 0$ everywhere except at $x = L$ requires term ① to be zero:

$$EA\,\frac{d^2u}{dx^2} + F_x = 0 \tag{3.12}$$

thus demonstrating that the two approaches lead to the same result. It is worth noticing that in the variational approach the natural boundary conditions are automatically satisfied.

3.2.3 Weak formulation

In subsection 3.2.2, we showed that a differential problem (which governs the mathematical model of a physical phenomenon) can be expressed equivalently with the variational approach. We are now going to investigate further on the variational formulation, and check if it can be expressed in a more useful — that is, easily implementable — way. The procedure followed here is described in [3, pp. 126–127].

The basic idea is to consider the variation δu as a *test function* v that satisfies the essential BCs. Equation (3.9) then reads:

$$\int_0^L \frac{dv}{dx}\,EA\,\frac{du}{dx}\,dx = \int_0^L F_x\,v\,dx + T_x\,v\big|_{x=L} = 0 \tag{3.13}$$

which can be enunciated in the following way:

> For u to be *the* solution of the problem, the left-hand side of equation (3.13) must be equal to the right-hand side for *any* arbitrary test function v that is continuous and satisfies the prescribed essential boundary conditions.

If we denote by L^2 the space of square-integrable functions on a certain domain Ω:

$$L^2(\Omega) := \left\{ f \,\middle|\, f \in \Omega, \int_\Omega |f|^2 \, d\Omega < \infty \right\} \tag{3.14}$$

and with V the function space such that

$$V(L) = \left\{ v \,\middle|\, v \in L^2(L), \frac{dv}{dx} \in L^2(L), \left. v\right|_{x=0} = 0 \right\} \tag{3.15}$$

we can express the previous statement in the form:

$$\text{Find } u \in V \text{ such that} \quad B(u,v) = F(v), \quad \forall v \in V$$

where the left-hand side

$$B(u,v) := \int_0^L \frac{dv}{dx} EA \frac{du}{dx} \, dx \tag{3.16}$$

is the *bilinear form* and the right-hand side

$$F(v) := \int_0^L F_x \, v \, dx + T_x \left. v\right|_{x=L} \tag{3.17}$$

is the *linear functional* of the problem. This approach is called *weak formulation* and is the basis of the Galerkin method, which we are now going to discuss. It should be noted that equation (3.15) corresponds to the condition of finite energy for a mechanical system [28, p. 34].

3.2.4 Galerkin method

The GALERKIN method pertains to a class of methods for the numerical resolution of differential equations called *weighted residuals methods*. The basic assumption is that the approximate solution u_n can be written as a linear combination of a set of linearly independent trial functions, that is [3, p. 118]:

$$u_n = \sum_{i=1}^{n} a_i N_i \tag{3.18}$$

where N_i is the i-th function and a_i the corresponding coefficient to be determined. Using the notation introduced in the previous subsection, we can also state the problem in the following way [3, p. 127]:

$$\text{Find } u_n \in V_n \text{ such that} \quad B(u_n,v_n) = F(v_n), \quad \forall v_n \in V_n$$

having defined V_n as

$$V_n(\Omega) = \left\{ v_n \,\middle|\, v_n \in L^2(\Omega), \frac{dv_n}{dx} \in L^2(\Omega), \left. v_n\right|_{S_u} = 0 \right\} \tag{3.19}$$

where S_u is the surface area on which zero displacement is prescribed. In the Galerkin method, the coefficients a_i are sought by imposing the orthogonality (called *Galerkin orthogonality*) between the error $e := u - u_n$ and the trial function v_n [28, p. 43]:

$$B(e, v_n) = 0\,. \tag{3.20}$$

Such condition is obviously satisfied when the exact solution is found ($u \equiv u_n$).

3.2.5 Principle of virtual displacements

It is interesting to specialize the previous statements for a particular yet important class of problems, the elastostatics problems, because of the physical meaning that the weak formulation assumes [3, pp. 157–158]. In three dimensions, using Einstein notation, the problem is given by the equilibrium condition:

$$\sigma_{ij,j} + F_i = 0 \tag{3.21}$$

that must be coupled with the natural (force) boundary conditions

$$\sigma_{ij} n_j = T_i \quad \text{on } S_f \tag{3.22a}$$

and the essential (displacement) boundary conditions

$$u_i = \tilde{u}_i \quad \text{on } S_u \tag{3.22b}$$

where $S = S_f \cup S_u$, $S_f \cap S_u = 0$. Let us consider *any* arbitrary chosen continuous displacement $\bar{u}_i$ that satisfies

$$\bar{u}_i = 0 \quad \text{on } S_u\,. \tag{3.23}$$

Equation (3.21) must hold also in this case:

$$(\sigma_{ij,j} + F_i)\,\bar{u}_i = 0 \tag{3.24}$$

and the equality is preserved also upon integration:

$$\int_V (\sigma_{ij,j} + F_i)\,\bar{u}_i \,dV = 0\,. \tag{3.25}$$

Using the product rule

$$(\sigma_{ij}\bar{u}_i)_{,j} = \sigma_{ij,j}\bar{u}_i + \sigma_{ij}\bar{u}_{i,j} \tag{3.26}$$

and applying the divergence theorem

$$\int_V (\sigma_{ij}\bar{u}_i)_{,j} \,dV = \int_S (\sigma_{ij}\bar{u}_i)\, n_j \,dS \tag{3.27}$$

we obtain:

$$\int_V (-\sigma_{ij}\bar{u}_{i,j} + F_i\bar{u}_i)\, dV + \int_S (\sigma_{ij}\bar{u}_i)\, n_j\, dS \tag{3.28}$$

that, in light of the boundary conditions (3.22a) and (3.22b), becomes:

$$\int_V (-\sigma_{ij}\bar{u}_{i,j} + F_i\bar{u}_i)\, dV + \int_{S_f} T_i\check{\bar{u}}_i\, dS = 0 \tag{3.29}$$

where $\check{\bar{u}}_i \coloneqq \bar{u}_i\big|_{S_f}$. At this point, we only have (i) to exploit the symmetry of the stress tensor ($\sigma_{ij} = \sigma_{ji}$) so to write

$$\sigma_{ij}\bar{u}_{i,j} = \sigma_{ij}\left[\tfrac{1}{2}(\bar{u}_{i,j} + \bar{u}_{j,i})\right] = \sigma_{ij}\bar{\varepsilon}_{ij} \tag{3.30}$$

and (ii) to introduce the constitutive equation

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} \tag{3.31}$$

to get the expression

$$\int_V C_{ijkl}\varepsilon_{kl}\bar{\varepsilon}_{ij}\, dV = \int_V F_i\bar{u}_i\, dV + \int_{S_f} T_i\check{\bar{u}}_i\, dS \tag{3.32}$$

which is the enunciation of the principle of virtual displacements for a linear elastic material.[1] In words,

> For u to be *the* solution of the problem, the left-hand side of equation (3.32) (the internal virtual work) must be equal to the right-hand side (the external virtual work) for *any* virtual displacement $\bar{u}$ that is continuous and satisfies the prescribed boundary conditions.

We have thus demonstrated that the principle of virtual displacements is the emanation of the weak formulation for linear elastostatic problems.

The principle fulfils all the fundamentals requirements of continuum mechanics [3, pp. 160–161]:

1. *Equilibrium* clearly holds, since the principle was derived starting from equation (3.21).

2. *Compatibility* holds because the displacement field is continuous and satisfies the prescribed essential boundary conditions.

3. The *constitutive law* holds because the stresses are calculated from the strains, at their time evaluated from the displacement field through derivation.

1 The validity of the principle is not limited to linear elasticity. Introducing a different constitutive law at point (ii), it could be possible to apply it to inelastic materials, as well [37, p. 55].

As a concluding remark, we point out that equation (3.32) could be obtained by imposing the stationarity of the following total potential [3, p. 160]:

$$\Pi(\{\mathbf{u}\}) = \frac{1}{2}\int_V \{\varepsilon\}^T[\mathbf{C}]\{\varepsilon\}\,dV - \int_V \{\mathbf{u}\}^T\{\mathbf{F}\}\,dV - \int_{S_f} \{\check{\mathbf{u}}\}^T\{\mathbf{T}\}\,dS \qquad (3.33)$$

confirming again the equivalence between the differential, variational, and weak formulation.

3.2.6 Finite Element equations

As we said in subsection 3.2.1, a large class of physical problems can be expressed in terms of differential equations, whose solution is sought onto a certain domain. When complicated domains are considered, it is not generally possible to obtain a closed-form solution, and numerical approximation becomes necessary. The basic idea of the Finite Element Method is to subdivide the domain into a grid of elements, called *mesh*, onto which the Galerkin method is applied. In this subsection, we are going to derive the basic matrix equations which govern the Finite Element Method, on the basis of the theoretical concepts previously described. Since we are dealing with two-dimensional problems, the formulation will be derived for this particular case, although the validity of the method is more general. The main reference for this subsection is [37, pp. 49–66].

Once again, our starting point are the equilibrium equations, defined in subsection 1.5.1 in a Cartesian coordinate system, and reported here for convenience:

$$\begin{cases} \dfrac{\partial \sigma_x}{\partial x} + \dfrac{\partial \tau_{xy}}{\partial y} + F_x = 0 \\ \dfrac{\partial \tau_{xy}}{\partial x} + \dfrac{\partial \sigma_y}{\partial x} + F_y = 0\,. \end{cases} \qquad (1.13,\ \text{rep.})$$

Let us seek a way to write them in a matrix form, which is more easy to handle numerically. If we define the *differential operator matrix* $[\mathbf{D}]$ as follows:

$$[\mathbf{D}] = \begin{bmatrix} \dfrac{\partial}{\partial x} & 0 & \dfrac{\partial}{\partial y} \\ 0 & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \qquad (3.34)$$

and we collect the stresses and the body forces in two vectors, respectively $\{\boldsymbol{\sigma}\} = \{\sigma_x, \sigma_y, \tau_{xy}\}^T$ and $\{\mathbf{F}\} = \{F_x, F_y\}^T$, the equations in (1.13) become:

$$[\mathbf{D}]\{\boldsymbol{\sigma}\} + \{\mathbf{F}\} = 0\,. \qquad (3.35)$$

The stress-strain relations in matrix form are:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{12} & E_{22} & E_{23} \\ E_{13} & E_{23} & E_{33} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{3.36}$$

where the coefficients E_{ij} are obtained by inverting the relations (1.11). If we collect them in a matrix [E] (which is called *elasticity matrix*), we can express the previous relation in the vectorial form:

$$\{\boldsymbol{\sigma}\} = [\mathbf{E}]\{\boldsymbol{\varepsilon}\} \tag{3.37}$$

where $\{\boldsymbol{\varepsilon}\} = \{\varepsilon_x, \varepsilon_y, \gamma_{xy}\}^T$ is the strain vector. We just need to recall the strain-displacement relations:

$$\varepsilon_x = \frac{\partial u}{\partial x}, \quad \varepsilon_y = \frac{\partial v}{\partial y}, \quad \gamma_{xy} = \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \tag{1.14, rep.}$$

which in vectorial form become

$$\{\boldsymbol{\varepsilon}\} = [\mathbf{D}]^T\{\mathbf{u}\} \tag{3.38}$$

where $\{\mathbf{u}\} = \{u, v\}^T$ is the displacement vector, to reformulate equation (1.13) as:

$$[\mathbf{D}]([\mathbf{E}][\mathbf{D}]^T\{\mathbf{u}\}) + \{\mathbf{F}\} = 0\,. \tag{3.39}$$

The boundary conditions read:

$$\begin{cases} \{\mathbf{u}\} = \{\tilde{\mathbf{u}}\} & \text{on } S_u \\ [\mathbf{L}]\{\boldsymbol{\sigma}\} = \{\mathbf{T}\} & \text{on } S_f \end{cases} \tag{3.40}$$

where

$$[\mathbf{L}] = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{3.41}$$

is the matrix collecting the components of the outer normal unit vector to the boundary surface S_f and $\{\mathbf{T}\} = \{T_x, T_y\}^T$ is the traction vector.

According to what said in subsection 3.2.4, we approximate the displacement vector $\{\mathbf{u}\}$ as follows:

$$\{\mathbf{u}\} = \begin{Bmatrix} N_1(x,y)\,u_1 + \cdots + N_n(x,y)\,u_n \\ N_1(x,y)\,v_1 + \cdots + N_n(x,y)\,v_n \end{Bmatrix} \tag{3.42}$$

where u_i, v_i are the displacements at the nodes $i = 1, \ldots, n$. N_i are interpolating functions called *shape functions*. They are described in some details in Appendix A; in order to continue our discussion, it is enough to remark their fundamental property:

$$N_i(x,y) = \begin{cases} 1, & \text{at node } i \\ 0, & \text{otherwise}\,. \end{cases} \tag{3.43}$$

If we define a *shape function matrix* $[\mathbf{N}]$:

$$[\mathbf{N}] = \begin{bmatrix} N_1 & 0 & \dots & N_n & 0 \\ 0 & N_1 & \dots & 0 & N_n \end{bmatrix} \tag{3.44}$$

and a nodal displacement vector $\{\mathbf{u}\}_n = \{u_1, v_1, \dots, u_n, v_n\}^T$, we can rewrite equation (3.42) in the form:

$$\{\mathbf{u}\} = [\mathbf{N}]\{\mathbf{u}\}_n \,. \tag{3.45}$$

The target of the Finite Element Method is to compute the vector $\{\mathbf{u}\}_n$, whose components are called *Degrees Of Freedom* (DOF).

Introducing a virtual displacement vector $\{\mathbf{v}\}$, thus defined:

$$\{\mathbf{v}\} = [\mathbf{N}]\{\mathbf{v}\}_n \tag{3.46}$$

and following the procedure described in subsection 3.2.5, we obtain:

$$\int_S ([\mathbf{D}]^T\{\mathbf{v}\})^T\{\boldsymbol{\sigma}\}\,dS = \int_S \{\mathbf{v}\}^T\{\mathbf{F}\}\,dS + \int_{l_f} \{\mathbf{v}\}^T\{\mathbf{T}\}\,dl \tag{3.47}$$

which is the principle of virtual displacements in two dimensions.

By defining the matrix $[\mathbf{M}]$ such that:

$$[\mathbf{M}] = [\mathbf{D}]^T[\mathbf{N}] \tag{3.48}$$

equation (3.47) can be rewritten as:

$$\{\mathbf{v}\}_n^T \int_S [\mathbf{M}]^T\{\boldsymbol{\sigma}\}\,dS = \{\mathbf{v}\}_n^T \int_S [\mathbf{N}]^T\{\mathbf{F}\}\,dS + \{\mathbf{v}\}_n^T \int_{l_f} [\mathbf{N}]^T\{\mathbf{T}\}\,dl \tag{3.49}$$

where vector $\mathbf{v}_n$ is a constant and can be simplified. Using equations (3.37) and (3.38) and introducing the relation

$$[\mathbf{D}]^T\{\mathbf{u}\} = [\mathbf{M}]\{\mathbf{u}\}_n \tag{3.50}$$

we finally get

$$\int_S [\mathbf{M}]^T[\mathbf{E}][\mathbf{M}]\{\mathbf{u}\}_n\,dS = \int_S [\mathbf{N}]^T\{\mathbf{F}\}\,dS + \int_{l_f} [\mathbf{N}]^T\{\mathbf{T}\}\,dl \,. \tag{3.51}$$

Since vector $\{\mathbf{u}\}_n$ is a constant, it can be placed outside the integral. If we denote by $[\mathbf{K}]$ the remaining integral:

$$[\mathbf{K}] = \int_S [\mathbf{M}]^T[\mathbf{E}][\mathbf{M}]\,dS \tag{3.52}$$

and by $\{\mathbf{F}\}_n$ the right-hand side:

$$\{\mathbf{F}\}_n = \int_S [\mathbf{N}]^T\{\mathbf{F}\}\,dS + \int_{l_f} [\mathbf{N}]^T\{\mathbf{T}\}\,dl \tag{3.53}$$

we obtain the fundamental expression of equilibrium of the Finite Element Method:

$$[\mathbf{K}]\{\mathbf{u}\}_n = \{\mathbf{F}\}_n \, . \tag{3.54}$$

$[\mathbf{K}]$ is called the *stiffness matrix*, while $\{\mathbf{F}\}_n$ is the nodal forces vector.

3.2.7 Standard element transformations

Let us define the *standard element* as follows:

$$\Omega_{st} := \{(\xi,\eta) \mid -1 \leqslant \xi \leqslant 1, \, -1 \leqslant \eta \leqslant 1\} \, . \tag{3.55}$$

The shape functions are interpolating functions that allow to map any two-dimensional element to the standard element, which is a square. If we consider a four-node quadrilateral element, the mapping reads:

$$\begin{cases} x(\xi,\eta) = \displaystyle\sum_{i=1}^{4} x_i N_i(\xi,\eta) \\ y(\xi,\eta) = \displaystyle\sum_{i=1}^{4} y_i N_i(\xi,\eta) \end{cases} \tag{3.56}$$

where (x_i, y_i) are the coordinates of the nodal displacements. The shape functions N_i in equation (3.56) are the following:[2]

$$\begin{aligned} N_1 &= \tfrac{1}{4}(1-\xi)(1-\eta), \quad N_2 = \tfrac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \tfrac{1}{4}(1+\xi)(1+\eta), \quad N_4 = \tfrac{1}{4}(1-\xi)(1-\eta) \, . \end{aligned} \tag{3.57}$$

Following the isoparametric approach, the same shape functions are used also to map the displacements:

$$\begin{cases} u(\xi,\eta) = \displaystyle\sum_{i=1}^{4} u_i N_i(\xi,\eta) \\ v(\xi,\eta) = \displaystyle\sum_{i=1}^{4} v_i N_i(\xi,\eta) \, . \end{cases} \tag{3.58}$$

The change of variables thus introduced would require to rewrite all the expressions derived in the previous subsection in terms of integrals of ξ, η defined onto the standard element. Without deriving the equations explicitly, we just point out that the transformation involves the Jacobian matrix $[\mathbf{J}]$:

$$\begin{Bmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{Bmatrix} = [\mathbf{J}] \begin{Bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{Bmatrix} \tag{3.59}$$

2 For further information about how the shape functions can be built, refer to Appendix A.

where

$$[\mathbf{J}] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \sum_{k=1}^{n} \begin{bmatrix} \dfrac{\partial N_k}{\partial \xi} x_k & \dfrac{\partial N_k}{\partial \xi} y_k \\ \dfrac{\partial N_k}{\partial \eta} x_k & \dfrac{\partial N_k}{\partial \eta} y_k \end{bmatrix} \tag{3.60}$$

from which follows that

$$\begin{Bmatrix} \dfrac{\partial N_k}{\partial x} \\ \dfrac{\partial N_k}{\partial y} \end{Bmatrix} = [\mathbf{J}]^{-1} \begin{Bmatrix} \dfrac{\partial N_k}{\partial \xi} \\ \dfrac{\partial N_k}{\partial \eta} \end{Bmatrix}. \tag{3.61}$$

By denoting $[\overline{\mathbf{M}}] = [\mathbf{M}(\boldsymbol{\xi}, \boldsymbol{\eta})]$, we can write the stiffness matrix of a single element as:

$$[\mathbf{K}]_e = \int_e [\overline{\mathbf{M}}]_e^T [\mathbf{E}][\overline{\mathbf{M}}]_e \, dS_e \,. \tag{3.62}$$

Since $dS_e = \det[\mathbf{J}] \, d\xi d\eta$, we have

$$[\mathbf{K}]_e = \int_{\Omega_{st}} [\overline{\mathbf{M}}]_e^T [\mathbf{E}][\overline{\mathbf{M}}]_e \det[\mathbf{J}] \, d\xi d\eta \tag{3.63}$$

and the global stiffness matrix reads

$$[\mathbf{K}] = \sum_e [\mathbf{K}]_e = \sum_e \int_{\Omega_{st}} [\overline{\mathbf{M}}]_e^T [\mathbf{E}][\overline{\mathbf{M}}]_e \det[\mathbf{J}] \, d\xi d\eta \,. \tag{3.64}$$

3.3 THE EXTENDED FINITE ELEMENT METHOD

In the standard Finite Element Method, the convergence to a *smooth* solution is achieved with a progressive mesh refinement. An *a priori* error estimate is given by [28, p. 193]:

$$\|u_{EX} - u_{FE}\|_E \approx \frac{k}{N^\beta} \tag{3.65}$$

where N in the number of degrees of freedom, k and β are two constants, and $\|u\|_E := \sqrt{\frac{1}{2} B(u,u)}$ is the *energy norm* [28, p. 42].

As seen in section 1.7, there are also many situations of practical interest where the solution presents high gradients or even singularities. The non-smoothness can drastically decrease the convergence rate of the FEM, and therefore increase dramatically the computational cost of the resolution; sometimes it can even lead to incorrect results [29]. In the standard FEM, the way to overcome this issue is to refine the mesh in proximity of these sources of discontinuities: In terms of error adaptivity, this technique is known as *h*-FEM. More recently, other techniques were developed, such as *p*-FEM, where the degree of the polynomial approximation space is increased, keeping the mesh fixed [29], or the eXtended Finite Element Method (XFEM). In the latter, the polynomial approximation space is enriched with special functions that take in account of the kind of discontinuity analysed; non-smooth solutions can be thus modelled independently of the mesh. In the following two subsections, we are briefly reviewing the XFEM, taking as a reference [11].

3.3.1 Description of interfaces

Since the XFEM does not involve mesh refinement, it is necessary to define a strategy to describe an interface within the domain. This target is achieved with the concept of *level set function*. A level set function is any continuous function $\Phi(\mathbf{x})$, $x \in \Omega$, that is negative in one subdomain and positive in the other. The closed interface Γ_{12} corresponds to the zero-level of this function:

$$\Gamma_{12} = \left\{ \mathbf{x} \,|\, \Phi(\mathbf{x}) = 0 \right\} . \tag{3.66}$$

A particularly useful function pertaining to this class is the *signed-distance function*, thus defined:

$$\Phi(\mathbf{x}) = \pm \min_{\mathbf{x}^\star \in \Gamma_{12}} \|\mathbf{x} - \mathbf{x}^\star\| \quad \forall \mathbf{x} \in \Omega \tag{3.67}$$

where $\|\cdot\|$ denotes the Euclidean norm. The signed-distance function is sketched in Figure 3.3. For discretized domains, the values of the level set function are stored at the nodes ($\Phi_i = \Phi(\mathbf{x}_i)$), and $\Phi(\mathbf{x})$ is interpolated using the standard FE shape functions $N_i(\mathbf{x})$:

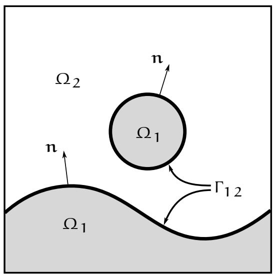
$$\Phi^n(\mathbf{x}) = \sum_{i \in I} N_i(\mathbf{x})\, \Phi_i \tag{3.68}$$

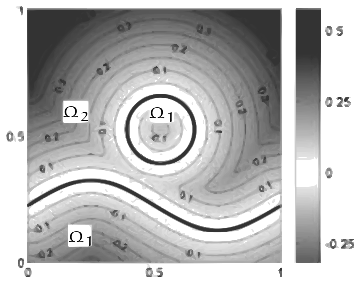
where I is the set of all nodes in Ω.

Until now, we tacitly assumed that the domain $\Omega \in \mathbb{R}^d$ was divided by the interface Γ_{12} into two different regions Ω_1 and Ω_2 such that $\Omega = \Omega_1 \cup \Omega_2$ and $\Omega_1 \cap \Omega_2 = \Gamma_{12}$, i.e. that Γ_{12} was a closed interface. Open interfaces, like cracks, dislocations, and shear bands, usually end inside the domain Ω. For cracks, it is necessary to introduce another level set function $\gamma(\mathbf{x})$ which defines the position of the crack tip. The crack is given by:

$$\Gamma_c = \left\{ \mathbf{x} \,|\, \Phi(\mathbf{x}) = 0 \text{ and } \gamma(\mathbf{x}) \leqslant 0 \right\} \tag{3.69}$$

where $\Phi(\mathbf{x})$ is the same signed-distance function described above, now tangentially extended from the crack tip to the entire domain (so to define a closed interface). $\gamma(\mathbf{x})$ — which is not necessarily a signed-distance function — is constructed such that it is orthogonal to Γ_c at the crack tip (see Figure 3.4).

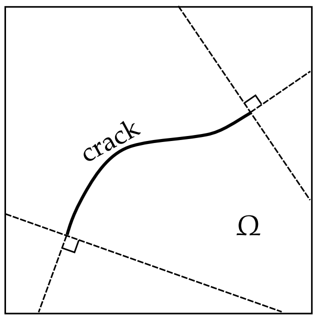




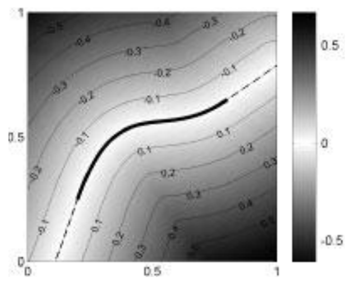
(a) The domain Ω decomposed into Ω_1 and Ω_2.

(b) The signed-distance function $\Phi(x)$.

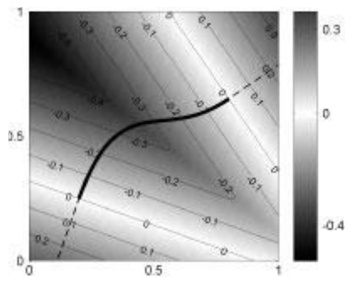
Figure 3.3. An example of the signed-distance function [11].



(a)



(b)



(c)

Figure 3.4. Definition of a crack with the XFEM: (a) The domain Ω with a crack; (b) the signed-distance function $\Phi(x)$ for the description of the crack path; (c) the second level set function $\gamma(x)$ for defining the crack tips [11].

3.3.2 Structure of the XFEM

Let us consider a domain $\Omega \in \mathbb{R}^d$, discretized in n elements, where a function $u(x)$, $x \in \Omega$, is defined. The *global* enrichment of the approximation $u_n(x)$ reads:

$$u_n(x) = \underbrace{\sum_{i\in I} N_i(x)\, u_i}_{\text{Standard FE approx.}} + \underbrace{\sum_{i\in I} N_i^\star(x) \cdot \psi(x)\, a_i}_{\text{Enrichment term}} \qquad (3.70)$$

where I is the set of all the nodes in the domain. Both N_i and $N_i^\star$ are standard FE shape functions, that not necessarily coincide, just like the coefficients u_i are the same used in the standard FEM. In addition, the enrichment term brings other nodal unknowns a_i. $\psi(x)$ is the *enrichment function*, that incorporates the special knowledge about the discontinuity in the approximation space. The product $N_i^\star(x) \cdot \psi(x)$ has the same support of the standard FE shape function and leads to the sparsity of the discrete equations.

A fundamental property of the functions $N_i^\star$ is the ability to build a Partition of Unity (PU) over the domain Ω, that means

$$\sum_{i\in I} N_i^\star(x) = 1\,. \qquad (3.71)$$

As a consequence, the approximation (3.70) can reproduce exactly *any* enrichment function in Ω. Since this kind of approximations generally does not have the Kronecker-δ property, it follows that $u_h(x_i) \neq u_i$, thus complicating the imposition of the essential boundary conditions and making more difficult to interpret the results. In order to recover the δ property, the approximation is shifted:

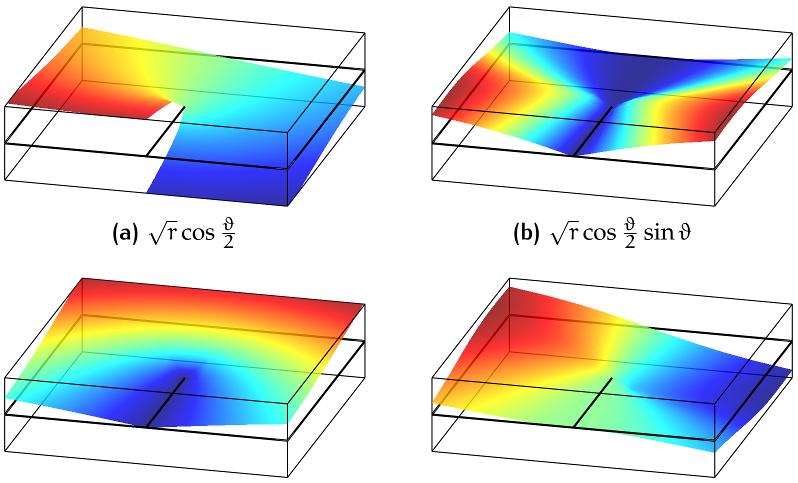
$$u_n(x) = \sum_{i\in I} N_i(x)\, u_i + \sum_{i\in I} N_i^\star(x) \cdot \big[\psi(x) - \psi(x_i)\big]\, a_i\,. \qquad (3.72)$$

It is possible to demonstrate that the shifting does not affect the ability of reproducing exactly any enrichment function $\psi(x)$.

A global enrichment is computationally demanding because the number of enriched degrees of freedom is proportional to the number of nodes in Ω. Since discontinuities and high gradients involve local phenomena, in many cases it is sufficient to enrich a nodal subset $I^\star \subset I$. The approximation then becomes:

$$u_n(x) = \sum_{i\in I} N_i(x)\, u_i + \sum_{i\in I^\star} N_i^\star(x) \cdot \big[\psi(x) - \psi(x_i)\big]\, a_i\,. \qquad (3.73)$$

In local enrichments, three categories of elements can be defined: The element is (i) a standard FE if *none* of the element nodes are enriched, (ii) a reproducing element if *all* element nodes are enriched, or (iii) a blending element if *some* of the element nodes are enriched. The


(a) $\sqrt{r}\cos\frac{\vartheta}{2}$ (b) $\sqrt{r}\cos\frac{\vartheta}{2}\sin\vartheta$

(c) $\sqrt{r}\sin\frac{\vartheta}{2}$ (d) $\sqrt{r}\sin\frac{\vartheta}{2}\sin\vartheta$

Figure 3.5. Crack tip enrichment functions for brittle materials [11].

presence of blending elements is problematic, since although there the functions $N_i^\star(x)$ are non-zero, they do not build a PU. As a consequence, (i) the enrichment function cannot be reproduced exactly, and (ii), additional, parasitic terms are added to the approximation, which badly affect the convergence properties of the method. Some techniques were developed to avoid the drawbacks due to the presence of such elements: The interested reader is recommended to consult the reference [11].

For cracks in brittle materials, that is our case of interest, it was suggested to use the following enrichment function vector [5]:

$$\boldsymbol{\psi}_{\text{crack}}(\mathbf{x}) = \left\{\sqrt{r}\cos\tfrac{\vartheta}{2}, \sqrt{r}\cos\tfrac{\vartheta}{2}\sin\vartheta, \sqrt{r}\sin\tfrac{\vartheta}{2}, \sqrt{r}\sin\tfrac{\vartheta}{2}\sin\vartheta\right\} \quad (3.74)$$

which spans the displacement field predicted by Westergaard (see again equation (1.94) in subsection 1.8.4), for mode I and II; its components are represented graphically in Figure 3.5. The definition (3.74) can be further generalized on the basis of the displacements derived by Williams (equations (1.45a) and (1.45b) in subsection 1.7.2):

$$\begin{aligned}\boldsymbol{\psi}^{(i)}_{\text{notch}}(\mathbf{x}) = \big\{& r^{\lambda_i}\cos(\lambda_i - 1)\vartheta, r^{\lambda_i}\cos(\lambda_i + 1)\vartheta, \\ & r^{\lambda_i}\sin(\lambda_i - 1)\vartheta, r^{\lambda_i}\sin(\lambda_i + 1)\vartheta\big\}, \quad \text{for } i = 1, 2\,. \end{aligned} \quad (3.75)$$

3.4 NUMERICAL QUADRATURE

Inside a Finite Element code, the integral formulations described in subsection 3.2.6 are solved numerically. It is worth spending some words on numerical integration (also called numerical quadrature),

Table 3.1. Exact values of Gauss-Legendre abscissas and weights, for a number of integration points up to 5 [32].

n	t_i	w_i
1	0	2
2	$\pm 1/\sqrt{3}$	1
3	0	$8/9$
	$\pm\sqrt{3/5}$	$5/9$
4	$\pm\sqrt{(3-2\sqrt{6/5})/7}$	$(18+\sqrt{30})/36$
	$\pm\sqrt{(3+2\sqrt{6/5})/7}$	$(18-\sqrt{30})/36$
5	0	$128/225$
	$\pm\frac{1}{3}\sqrt{5-2\sqrt{10/7}}$	$(322+13\sqrt{70})/900$
	$\pm\frac{1}{3}\sqrt{5+2\sqrt{10/7}}$	$(322-13\sqrt{70})/900$

since the same technique is going to be implemented in the algorithm for the computation of the local strain energy density. The reference for this section, unless otherwise stated, is [28, pp. 321–322].

A *quadrature rule* is an approximation of the definite integral of a function as a weighted sum of the function values at specific points of the domain. On the conventional domain of integration $[-1, +1]$, it takes the form:

$$\int_{-1}^{+1} f(x)\,dx \approx \sum_{i=1}^{n} w_i\, f(t_i)\,. \tag{3.76}$$

In the Gauss-Legendre quadrature, the weights are calculated with the Legendre polynomials $P_n(x)$:[3]

$$w_i = \frac{2}{(1-t_i^2)[P_n'(t_i)]^2} \tag{3.77}$$

where the evaluation point t_i is the i-th root of P_n. If n evaluation points are used, the rule yields to the exact result (up to round-off errors) for polynomials of degree $2n-1$. With a simple change of variables, every interval $[a, b]$ can be traced back to $[-1, +1]$:

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{b-a}{2}\int_{-1}^{+1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) dx \\ &\approx \frac{b-a}{2}\sum_{i=1}^{n} w_i\, f\left(\frac{1-t_i}{2}a + \frac{1+t_i}{2}b\right).\end{aligned} \tag{3.78}$$

Some exact values of t_i and w_i are reported in table 3.1.

3 See Appendix A for the definition of Legendre polynomials.

4 NUMERICAL PROCEDURES

In the previous chapter, we briefly outlined the main theoretical aspects of numerical analysis which were useful for our purposes. We are now using that concepts to build our numerical procedure.

In chapter 2, we explained as the SED criterion can be employed to assess the fatigue life of welded joints. For what concerns the numerical implementation of the criterion, two important observations have to be made:

- In a recent paper, Lazzarin *et al.* [16] showed as an accurate evaluation of the local strain energy density can be achieved with meshes much coarser than the ones necessary for the evaluation of other singular field parameters, such as the Notch Stress Intensity Factors.

- Applying GREEN's theorem to the elastic energy

  $$\mathcal{U}(R) = \frac{1}{2}\, b \int_0^R \int_{\vartheta_a}^{\vartheta_b} \sigma_{ij}\, \varepsilon_{ij}\, r \, dr d\vartheta \tag{4.1}$$

  where b is the constant thickness, Yosibash *et al.* [36] were able to express it as a contour integral:

  $$\mathcal{U}(R) = \frac{1}{2}\, b \int_{\vartheta_a}^{\vartheta_b} [\sigma_{ij} n_j u_i]_{r=R}\, R \, d\vartheta \tag{4.2}$$

  whose evaluation requires significant less computational effort. From now on, we are referring in the text to equations (4.1) and (4.2) as the 2-D and 1-D integral formulation, respectively.

That said, our aims can be thus summarized:

1. Implementation of an algorithm for calculating the SED, able to interface with the FE code that computes the input quantities (stresses and displacements or stresses and strains).

2. Computation of the local SED for a cracked and notched plate with the 1-D and 2-D formulations, using the standard FEM.

3. Computation of the local SED for a cracked plate with the 2-D formulation, using the extended FEM.

4. Comparison of the efficiency of the 2-D and 1-D integral formulation in the two cases.

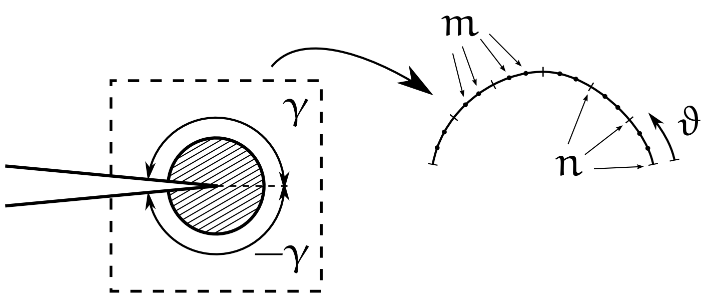


Figure 4.1. Schematic illustration of the integration procedure.

4.1 ALGORITHM FOR THE SED

Before introducing the algorithm, we would like to spend some words on its conception. Let us consider the control volume (an area in two dimensions) represented by the shaded region in Figure 4.1. Since the domain is symmetric, the angular interval is 2γ. The integration is realized by splitting the arc in n subintervals, and defining m Gaussian points inside each subinterval. The stresses and displacements are extrapolated at the $n \times m$ integration points from the Finite Element code. Considering the (k)-th iteration, we can describe the procedure as follows: Firstly, the traction vectors $T_i = \sigma_{ij} n_j$ are calculated; then, the local strain energy is computed:

$$\mathcal{U}^{(k)}(R) = \frac{1}{2} T_i \, u_i \, w^{(k)} \, R \, \Delta\vartheta^{(k)} \tag{4.3}$$

where $\Delta\vartheta^{(k)} = \frac{1}{2}(\vartheta^{(k+1)} - \vartheta^{(k)})$, and summed up with the value obtained at the previous iteration ($\mathcal{U}^{(k)}(R) = \mathcal{U}^{(k)}(R) + \mathcal{U}^{(k-1)}(R)$). Once the for loop is concluded, the last value of $\mathcal{U}(R)$ is divided by the area $A = \gamma R^2$ to release $\mathcal{SED}$.

All the steps necessary to compute the strain energy density are reported in the Algorithm 4.1, in guise of a pseudocode; the main operations are commented.

Algorithm 4.1. Pseudocode for the computation of $\mathcal{SED}$.

```
READ $n$, $m$        # Subdivisions and Gaussian points for each subdivision
READ $x_c$, $y_c$    # Coordinates of the centre [mm]
READ $R$             # Radius of the arc [mm]
READ $\gamma$        # Half angular interval [rad]
$\vartheta_a = -\gamma$
$\vartheta_b = +\gamma$
$A = \frac{1}{2}(\vartheta_b - \vartheta_a) R^2$
$\mathcal{U}(R) = 0$
FOR $i = 1, \ldots, n$:
        $\vartheta_1 = \vartheta_a + \frac{i-1}{n}\,(\vartheta_b - \vartheta_a)$
        $\vartheta_2 = \vartheta_a + \frac{i}{n}\,(\vartheta_b - \vartheta_a)$
        $d\vartheta = \frac{1}{2}\,(\vartheta_2 - \vartheta_1)$
        FOR $j = 1, \ldots, m$:
            READ t      # Gauss-Legendre abscissa
            $\vartheta = \frac{1}{2}\,(1-t)\,\vartheta_1 + \frac{1}{2}\,(1+t)\,\vartheta_2$    # Curvilinear abscissa [rad]
            $n_x = \cos\vartheta$
            $n_y = \sin\vartheta$
            $x = x_c + R\,\cos\vartheta$
            $y = y_c + R\,\sin\vartheta$
            GET $\sigma_x$, $\sigma_y$, $\tau_{xy}$              # From the Finite Element code
            GET $u_x$, $u_y$                   # From the Finite Element code
            $T_x = \sigma_x\,n_x + \tau_{xy}\,n_y$
            $T_y = \tau_{xy}\,n_x + \sigma_y\,n_y$
            READ $w$      # Gauss-Legendre weight
            $\mathcal{U}(R) = \mathcal{U}(R) + \frac{1}{2}(t_x\,u_x + t_y\,u_y)\,w\,R\,d\vartheta$
$\mathcal{SED} = \mathcal{U}(R)/A$
PRINT $\mathcal{SED}$
```

4.2 VALIDATION OF THE ALGORITHM

The Algorithm 4.1 was validated at two different levels:

1. Firstly, a numerical comparison between the closed-form 2-D integral and the contour integral built combining the analytical stresses and displacements was conducted using Python (see Appendix B for the scripts).

2. Secondly, the Python code was coupled with the FE code, which computed the stress tensor $\{\boldsymbol{\sigma}\}$ and the displacement vector $\{\mathbf{u}\}$ (see Appendix C for the command files).

This two-step check made it possible to detect bugs of the algorithm and distinguish whether the errors were due to the FE code or the post-processing quadrature of the integral.

The Finite Element analyses were conducted with the open source, freeware code Code_Aster, written by Électricité de France. The user can interact with the code in two ways:

- At a higher level, by using the native language of the code, which is the most common approach;

- At a deeper level, by modifying directly the FORTRAN subroutines.

Table 4.1. Values of the elastic constants used in the numerical analyses.

Quantity	Units	Value
E	MPa	210 000
ν		0.3
G	MPa	80 770

For our purposes, it was enough to work at the first level, since the strain energy density was computed in post-processing.

For the calculations, we used the elastic constants of a structural steel, reported in Table 4.1. The reliability of the output data was measured by computing the relative error $e_{\mathcal{SED}}$, thus defined:

$$e_{\mathcal{SED}} := \left| \frac{\mathcal{SED}_{\text{FEM}} - \mathcal{SED}_{\text{th}}}{\mathcal{SED}_{\text{th}}} \right| \tag{4.4}$$

where $\mathcal{SED}_{\text{FEM}}$ and $\mathcal{SED}_{\text{th}}$ are respectively the Finite Element and the theoretical solution. In some cases, we used also another definition of the relative error:

$$\tilde{e}_{\mathcal{SED}} := \left| \frac{\mathcal{SED}_{\text{FEM}} - \mathcal{SED}_{p\text{-FEM}}}{\mathcal{SED}_{p\text{-FEM}}} \right| \tag{4.5}$$

where $\mathcal{SED}_{p\text{-FEM}}$ is the value of the local strain energy density computed by a p-FEM code [7].

4.2.1 Plate subjected to a constant stress

The first test case is a square plate of side h with unit thickness, constrained as shown in Figure 4.2 and subjected to a constant stress σ. The stress field is therefore simply:

$$\sigma_x = \sigma, \quad \sigma_y = \tau_{xy} = 0\,. \tag{4.6}$$

By introducing the only non-zero stress component into the stress-strain relations (1.8) for plane strain, we get:

$$\begin{cases} \varepsilon_x = \dfrac{1-\nu^2}{E}\,\sigma \\ \varepsilon_y = -\dfrac{\nu\,(1+\nu)}{E}\,\sigma \\ \gamma_{xy} = 0 \end{cases} \tag{4.7}$$

while the displacements are obtained upon integration:

$$\begin{cases} u = \displaystyle\int \varepsilon_x \,dx = \dfrac{1-\nu^2}{E}\,\sigma \cdot x + f_1(y) \\ v = \displaystyle\int \varepsilon_y \,dy = -\dfrac{\nu\,(1+\nu)}{E}\,\sigma \cdot y + f_2(x)\,. \end{cases} \tag{4.8}$$

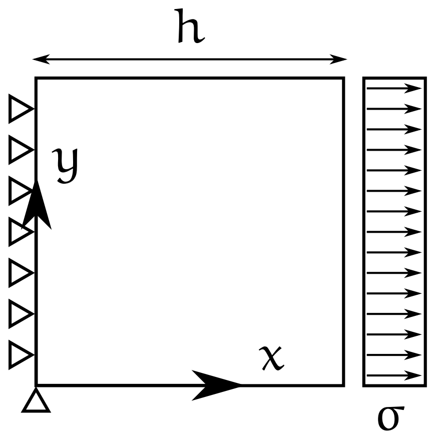


Figure 4.2. Plate subjected to a constant tensile stress.

Applying the essential boundary conditions, it results:

$$\begin{cases} u\big|_{x=0} = 0 \implies f_1(y) = 0 \\ v\big|_{x=0,\,y=0} = 0 \implies f_2(0) = 0\,. \end{cases} \tag{4.9}$$

By recalling the compatibility equation

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = f_2'(x) = 0 \tag{4.10}$$

we conclude that $f_2(x) = c = f_2(0) = 0$. Hence, the displacements are:

$$\begin{cases} u = \dfrac{1-\nu^2}{E}\,\sigma \cdot x \\ v = -\dfrac{\nu\,(1+\nu)}{E}\,\sigma \cdot y\,. \end{cases} \tag{4.11}$$

The strain energy of a closed circle with radius R is:

$$\begin{aligned} \mathcal{U}(R) &= \frac{1}{2}\int_A (\sigma_x \varepsilon_x + \sigma_y \varepsilon_y + \tau_{xy}\gamma_{xy})\, dA \\ &= (1-\nu^2)\,\frac{\sigma^2}{2E}\int_A dA \\ &= (1-\nu^2)\,\frac{\sigma^2}{2E}\int_{-\pi}^{+\pi}\int_0^R r\,dr\,d\vartheta \\ &= (1-\nu^2)\,\frac{\sigma^2}{2E}\,\pi R^2 \end{aligned} \tag{4.12}$$

and the strain energy density is therefore:

$$\mathcal{SED} = \frac{\mathcal{U}(R)}{\pi R^2} = (1-\nu^2)\,\frac{\sigma^2}{2E} \tag{4.13}$$

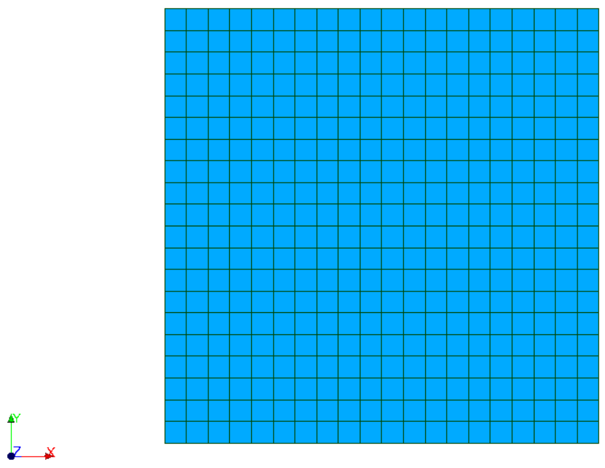


Figure 4.3. Finite Element model of the plate.

independent of the radius and constant over the entire plate. Assuming $\sigma = 100\,\mathrm{MPa}$, it results $\mathcal{SED} \equiv W = 0.021\overline{6}\,\mathrm{N\,mm/mm^3}$.

First check:

Because of the easiness of the model, we expect the Python script to converge rapidly to the exact solution. In fact, 50 samplings in random locations of the plate with a number of subdivisions of the circumference n equal to 3 and one Gaussian point for each subdivision ($m = 1$) have lead to a relative error always lower than $1.5 \times 10^{-12}\%$.

Second check:

The FE model is represented in Figure 4.3 and consists in a plate of side $h = 100\,\mathrm{mm}$ subdivided in 400 quadratic elements of $5 \times 5\,\mathrm{mm}$. The total number of nodes is 441. Also in this case, the convergence was very fast: With $n = 3$, $m = 1$ the final error was always less than $1.5 \times 10^{-10}\%$.

4.2.2 Plate subjected to a linear stress

The second test case we are considering is slightly more complex than the previous one: The plate is now subjected to a linear tensile stress, which goes from 0 to σ, as shown in Figure 4.4. The stress field is easily determined:

$$\sigma_x = \sigma\left(1 - \frac{y}{h}\right), \quad \sigma_y = \tau_{xy} = 0\,. \tag{4.14}$$

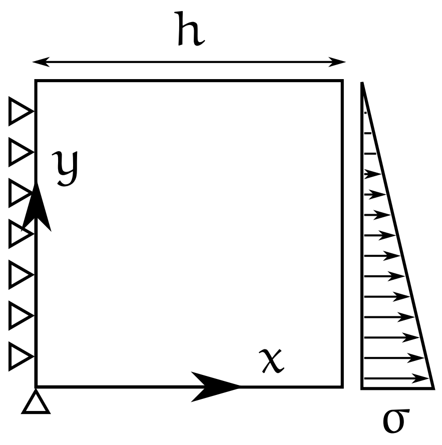


Figure 4.4. Plate subjected to a linear tensile stress.

From this, using again the relations (1.8) for plane strain, the following strain field is derived:

$$\begin{cases} \varepsilon_x = \dfrac{1-\nu^2}{E}\,\sigma\left(1-\dfrac{y}{h}\right) \\ \varepsilon_y = -\dfrac{\nu(1+\nu)}{E}\,\sigma\left(1-\dfrac{y}{h}\right) \\ \gamma_{xy} = 0\,. \end{cases} \tag{4.15}$$

The planar displacements are defined except for two functions, f_1 and f_2, which depend respectively on y and x:

$$\begin{cases} u = \displaystyle\int \varepsilon_x\,dx = \dfrac{1-\nu^2}{E}\,\sigma\left(1-\dfrac{y}{h}\right)x + f_1(y) \\ \nu = \displaystyle\int \varepsilon_y\,dy = -\dfrac{\nu(1+\nu)}{E}\,\sigma\left(1-\dfrac{y}{2h}\right)y + f_2(x)\,. \end{cases} \tag{4.16}$$

After applying the essential boundary conditions:

$$\begin{cases} u\big|_{x=0} = 0 \implies f_1(y) = 0 \\ \nu\big|_{x=0,\,y=0} = 0 \implies f_2(0) = 0 \end{cases} \tag{4.17}$$

— which correspond to the left edge constrained in the x direction and the lower left corner constrained in both directions, — f_1 is completely determined, while f_2 is still unknown. Using the compatibility equation:

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial \nu}{\partial x} = -\frac{1-\nu^2}{E}\,\sigma\,\frac{x}{h} + f_2'(x) = 0 \tag{4.18}$$

we obtain

$$f_2(x) = \frac{1-\nu^2}{E}\,\frac{\sigma}{2h}\,x^2 + c \tag{4.19}$$

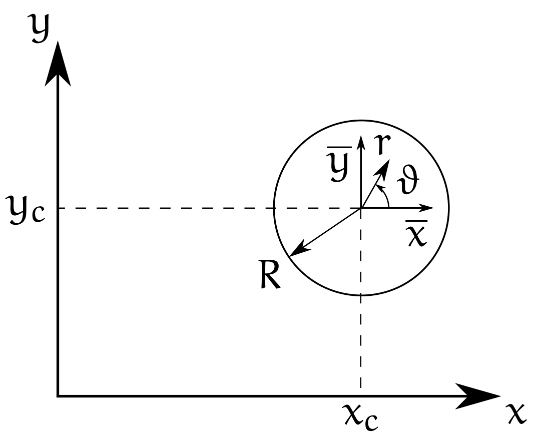


Figure 4.5. Definition of the local coordinate system.

where obviously $f_2(0) = c = 0$. So, the displacement field:

$$\begin{cases} u = \dfrac{1-\nu^2}{E}\,\sigma\left(1-\dfrac{y}{h}\right)x \\ \nu = \dfrac{1-\nu^2}{E}\,\sigma\left[\dfrac{x^2}{2h} - \dfrac{\nu}{1-\nu}\left(1-\dfrac{y}{2h}\right)y\right]. \end{cases} \tag{4.20}$$

It should be noted that ν shows a parabolic dependence on both coordinates x and y.

The calculation of the strain energy on a circle is less immediate than the previous case, because now the applied stress varies with y. In order to take in account of this fact, it is necessary to define a local Cartesian coordinate system $(\overline{x}, \overline{y})$, related to the global one by the following relations:

$$\begin{cases} x = \overline{x} + x_c \\ y = \overline{y} + y_c \end{cases} \tag{4.21}$$

where (x_c, y_c) are the coordinates of the center of the circle, as can be guessed by looking at Figure 4.5. The strain energy is then:

$$\begin{aligned} \mathcal{U}(R) &= \frac{1}{2}\int_A (\sigma_x\varepsilon_x + \sigma_y\varepsilon_y + \tau_{xy}\gamma_{xy})\,dA \\ &= \frac{1-\nu^2}{2E}\,\sigma^2\int_A \left(1-\frac{y}{h}\right)^2 dA \\ &= \frac{1-\nu^2}{2E}\,\sigma^2\int_A \left[1-\left(\frac{\overline{y}+y_c}{h}\right)\right]^2 dA\,. \end{aligned} \tag{4.22}$$

In order to compute the integral more easily, it is convenient to switch to polar coordinates:

$$\begin{cases} \overline{x} = r\cos\vartheta \\ \overline{y} = r\sin\vartheta\,. \end{cases} \tag{4.23}$$

Since the Jacobian determinant of the transformation is

$$\det[J] = \begin{vmatrix} \dfrac{\partial \overline{x}}{\partial r} & \dfrac{\partial \overline{x}}{\partial \vartheta} \\ \dfrac{\partial \overline{y}}{\partial r} & \dfrac{\partial \overline{y}}{\partial \vartheta} \end{vmatrix} = \begin{vmatrix} \cos\vartheta & -r\sin\vartheta \\ \sin\vartheta & r\cos\vartheta \end{vmatrix} = r \tag{4.24}$$

the strain energy reads

$$\begin{aligned} \mathcal{U}(R) &= \frac{1-\nu^2}{2E}\,\sigma^2 \int_A \left[1 - \left(\frac{r\sin\vartheta + y_c}{h}\right)\right]^2 r\,d\vartheta dr \\ &= \frac{1-\nu^2}{2E}\,\sigma^2 \int_0^R \int_0^{2\pi} \left[1 - \left(\frac{r\sin\vartheta + y_c}{h}\right)\right]^2 r\,d\vartheta dr \\ &= \frac{1-\nu^2}{2E}\left(\frac{\sigma}{h}\right)^2 \pi R^2 \left[(h-y_c)^2 + \left(\frac{R}{2}\right)^2\right]. \end{aligned} \tag{4.25}$$

Hence, the local strain energy density is:

$$\mathcal{SED} = \frac{\mathcal{U}(R)}{\pi R^2} = \frac{1-\nu^2}{2E}\left(\frac{\sigma}{h}\right)^2 \left[(h-y_c)^2 + \left(\frac{R}{2}\right)^2\right] \tag{4.26}$$

and depends on both the radius R and the ordinate of the center y_c. We have thus derived all the analytical expressions that we need to set our numerical problem.

First check:

As one can expect, the convergence of the Python script is not as fast as in the previous test: With $n = 4$ and $m = 2$, we still found an error of 1–2% for a couple of samplings. Only increasing m of another unity led to an error $e_{\mathcal{SED}} < 0.1\%$ everywhere.

Second check:

The FE model used is the same of the previous example. In this case, using a number of subdivisions $n = 4$ and 2 Gaussian points for each subdivision, the relative error $e_{\mathcal{SED}}$ was always lower than 0.5%.

4.2.3 Beam subjected to an end load

The last test case we are going to consider is a two-dimensional beam with unit thickness subjected to an end load F [30, pp. 35–38]. The problem is shown schematically in Figure 4.6. Unlike the other two cases, we are now working under the plane stress hypothesis.

From the beam theory, we expect only two components of the stress tensor to be active: (i) a non-zero tensile stress σ_x, induced by the bending, which depends on both x and y, and (ii), a shear stress τ_{xy}, which results from the superposition of the parabolic stress on the

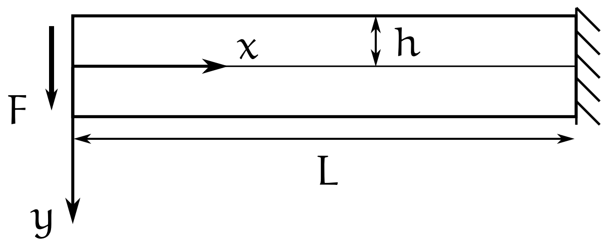


Figure 4.6. Beam subjected to an end load.

pure shear condition.

Having recourse to the stress function method, as explained in section 1.6, we can then hypothesize the following Airy function:

$$\Phi = Axy^3 + Bxy\,. \tag{4.27}$$

By applying its definition (1.25), we get:

$$\sigma_x = 6Axy, \quad \sigma_y = 0, \quad \tau_{xy} = -B - 3Ay^2\,. \tag{4.28}$$

In order to determine the constants A and B, we have to impose two boundary conditions. Firstly, the shear stresses must vanish at the free edges, that is:

$$\tau_{xy}\big|_{y=\pm h} = 0 \tag{4.29}$$

which implies $A = -\frac{B}{3h^2}$. Then, by imposing the equilibrium between the sum of the shearing forces distributed at the edge and F:

$$-\int_{-h}^{+h} \tau_{xy}\, dy = F \tag{4.30}$$

one obtains $B = \frac{3}{4}\frac{F}{h}$, and therefore $A = -\frac{F}{4h^3}$. The stress field is now completely determined:

$$\sigma_x = -\frac{3}{2}\frac{F}{h^3}xy, \quad \sigma_y = 0, \quad \tau_{xy} = -\frac{3}{4}\frac{F}{h}\left[1 - \left(\frac{y}{h}\right)^2\right]. \tag{4.31}$$

Introducing the moment of inertia $I = \frac{2}{3}\,bh^3$, we can write:

$$\sigma_x = -\frac{F\,xy}{I}, \quad \sigma_y = 0, \quad \tau_{xy} = -\frac{F}{2\,I}(h^2 - y^2)\,. \tag{4.32}$$

The strain field follows from equations (1.10):

$$\begin{cases} \varepsilon_x = -\dfrac{F\,xy}{EI} \\ \varepsilon_y = \nu\,\dfrac{F\,xy}{EI} \\ \gamma_{xy} = -\dfrac{F}{2GI}(h^2 - y^2) \end{cases} \tag{4.33}$$

while the displacements are obtained upon integration:

$$\begin{cases} u = \int \varepsilon_x \, dx = -\dfrac{F x^2 y}{2EI} + f_1(y) \\ v = \int \varepsilon_y \, dy = \nu \dfrac{F x y^2}{2EI} + f_2(x) \, . \end{cases} \tag{4.34}$$

The compatibility equation reads:

$$\underbrace{-\frac{F x^2}{2EI} + f_1'(y)}_{\frac{\partial u}{\partial y}} + \underbrace{\nu \frac{F y^2}{2EI} + f_2'(x)}_{\frac{\partial v}{\partial x}} = \underbrace{-\frac{F}{2GI}(h^2 - y^2)}_{\gamma_{xy}} \, . \tag{4.35}$$

If we make the following definitions:

$$\begin{aligned} F(x) &:= -\frac{F x^2}{2EI} + f_2'(x) \\ G(y) &:= \nu \frac{F y^2}{2EI} - \frac{F y^2}{2GI} + f_1'(y) \\ C &:= -\frac{F h^2}{2GI} \end{aligned} \tag{4.36}$$

equation (4.35) becomes

$$F(x) + G(y) = C \tag{4.37}$$

which means that the functions F, G have to be constant. Otherwise, in fact, we could vary one coordinate keeping the other fixed, and the equality would be violated.

By introducing two new constants c_1 and c_2, it is possible to write the following conditions on the functions f_1, f_2:

$$\begin{aligned} f_1'(y) &= -\nu \frac{F y^2}{2EI} + \frac{F y^2}{2GI} + c_2 \\ f_2'(x) &= \frac{F x^2}{2EI} + c_1 \end{aligned} \tag{4.38}$$

which upon integration release

$$\begin{aligned} f_1(y) &= -\nu \frac{F y^3}{6EI} + \frac{F y^3}{6GI} + c_2 \, y + c_3 \\ f_2(x) &= \frac{F x^3}{6EI} + c_1 \, x + c_4 \, . \end{aligned} \tag{4.39}$$

The displacement field is therefore:

$$\begin{cases} u = -\dfrac{F x^2 y}{2EI} - \nu \dfrac{F y^3}{6EI} + \dfrac{F y^3}{6GI} + c_2 \, y + c_3 \\ v = \nu \dfrac{F x y^2}{2EI} + \dfrac{F x^3}{6EI} + c_1 \, x + c_4 \, . \end{cases} \tag{4.40}$$

In order to determine the constants c_1 to c_4, we need to impose four BCs. From the conditions

$$\left[u,\, v,\, \frac{\partial v}{\partial x}\right]_{x=L,\, y=0} = 0 \tag{4.41}$$

we obtain $c_1 = -\frac{F L^2}{2EI}$, $c_3 = 0$, and $c_4 = \frac{F L^3}{3EI}$. The last constant can be derived using the compatibility equation:

$$c_2 = C - c_1 = \frac{F L^2}{2EI} - \frac{F h^2}{2GI} \tag{4.42}$$

The displacements are now completely defined:

$$\begin{cases} u = -\dfrac{F x^2 y}{2EI} - \nu \dfrac{F y^3}{6EI} + \dfrac{F y^3}{6GI} + \left(\dfrac{F L^2}{2EI} - \dfrac{F h^2}{2GI}\right) y \\ v = \nu \dfrac{F x y^2}{2EI} + \dfrac{F x^3}{6EI} - \dfrac{F L^2 x}{2EI} + \dfrac{F L^3}{3EI}\,. \end{cases} \tag{4.43}$$

It is interesting to notice that

$$v\big|_{y=0} = \frac{F L^3}{6EI}\left[2 - 3\,\frac{x}{L} + \left(\frac{x}{L}\right)^3\right] \tag{4.44}$$

is the deflection of the neutral axis predicted by the Euler-Bernoulli beam theory, which demonstrates the consistency of our hypotheses.

After the displacements, we calculate the strain energy related to a circle with radius R:

$$\begin{aligned} \mathcal{U}(R) &= \frac{1}{2}\int_A (\sigma_x \varepsilon_x + \sigma_y \varepsilon_y + \tau_{xy}\gamma_{xy})\, dA \\ &= \frac{1}{2}\int_A \left[\frac{F xy}{I}\frac{F xy}{EI} + \frac{F}{2I}(h^2 - y^2)\frac{F}{2GI}(h^2 - y^2)\right] dA \\ &= \frac{1}{2}\left(\frac{F}{I}\right)^2 \int_A \left[\frac{x^2 y^2}{E} + \frac{(h^2 - y^2)^2}{4G}\right] dA\,. \end{aligned} \tag{4.45}$$

By following the procedure described in the previous example, which defines firstly a local coordinate system $(\overline{x}, \overline{y})$, and then a polar coordinate system (r, ϑ) with the same origin, the strain energy turns out to be:

$$\begin{aligned} \mathcal{U}(R) &= \frac{1}{2}\left(\frac{F}{I}\right)^2 \int_0^R \int_0^{2\pi} \left\{\frac{1}{E}(r\cos\vartheta + x_c)^2 (r\sin\vartheta + y_c)^2 \right. \\ &\qquad\qquad \left. + \frac{1}{4G}\left[h^2 - (r\sin\vartheta + y_c)^2\right]^2\right\} r\, d\vartheta dr \\ &= \frac{1}{48}\left(\frac{F}{I}\right)^2 \pi R^2 \left\{\frac{1}{E}(R^4 + 6\,(x_c^2 + y_c^2)\, R^2 + 24\, x_c^2\, y_c^2) \right. \\ &\quad \left. + \frac{1}{G}\left[R^4 + 3\,(3\,y_c^2 - h^2)\, R^2 + 6\,(h^2 - y_c^2)^2\right]\right\} \end{aligned} \tag{4.46}$$

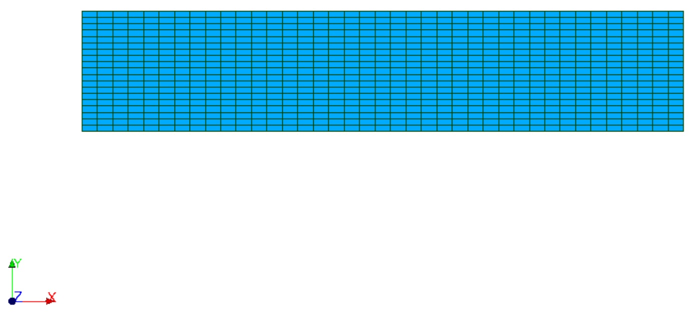


Figure 4.7. Finite Element model of the beam.

Dividing by the area, we obtain the local strain energy density:

$$\mathcal{SED} = \frac{\mathcal{U}(R)}{\pi R^2} = \frac{1}{48}\left(\frac{F}{I}\right)^2 \left\{ \frac{1}{E}(R^4 + 6\,(x_c^2 + y_c^2)\,R^2 + 24\,x_c^2\,y_c^2) + \frac{1}{G}\left[R^4 + 3\,(3\,y_c^2 - h^2)\,R^2 + 6\,(h^2 - y_c^2)^2\right] \right\} \tag{4.47}$$

which depends on the coordinates x_c, y_c of the center of the circle and on its radius R.

First check:

Since the dependence on the coordinates for both the stresses and the displacements is not linear, we are expecting the solution to converge more slowly. In agreement with this prediction, the Python script required at least 5 subdivisions and 4 Gaussian points to ensure a relative error $e_{\mathcal{SED}}$ on the SED always lower than 0.1%.

Second check:

The FE model of the beam is represented in Figure 4.7. Its dimensions are $L = 100\,\text{mm}$, $h = 10\,\text{mm}$. The model consists in 741 elements of approximately $1 \times 2.6\,\text{mm}$, for a total of 800 nodes.

This time, the calculation of the strain energy density was more problematic. More precisely, the accuracy was usually comparable with the previous cases, but there were always a limited number of points where the convergence was not reached, even when increasing significantly the fineness of the mesh and the integration points.

This observation was explained with the presence of shear forces. In fact, the accuracy in the evaluation of the shear components of the stress tensor depends on the assumptions made in the formulation of the elements, and is commonly less good in the proximity of the boundary conditions or in regions where the shear contribution is significant. According to this interpretation, all the problematic points were located either close to the edges or to the neutral axis.[1] When these points were ignored, setting $n = m = 5$, it always resulted $e_{\mathcal{SED}} < 0.5\%$.

4.3 APPLICATION OF THE ALGORITHM

After validating the algorithm with the previous test cases, we want to use it in configurations where only the asymptotic solution is known. When the theory is not enough powerful to give us a comparison value, we are using as a reference the results obtained with a *p*-FEM code [7]. For our computations, unless specified, we are considering a radius R of the control volume equal to $0.3\,\text{mm}$, for the reasons outlined in section 2.3.

4.3.1 Cracked plate

The first application of the Algorithm 4.1 is the classical Fracture Mechanics problem discussed in subsection 1.8.4: A (theoretically) infinite plate weakened by a central crack, as shown in Figure 4.8. Equations (1.92) and (1.94) allow us to estimate the asymptotic stress and displacement fields, but they lose rapidly their validity when we move away from the crack tip. The region of K_I dominance depends on the crack size and the geometry of the plate, but is usually less than $1\,\text{mm}$ [26, p. 51]. Outside this region, Westergaard's solution should be expanded introducing more terms; alternatively, one can estimate the stresses and the displacements with other techniques, like the boundary collocation method or the Finite Element Method, as we are doing.

Let us derive the strain energy density near the crack tip, as predicted by linear elastic fracture mechanics. As we stated several times, the two-dimensional strain energy reads:

$$\mathcal{U}(R) = \frac{1}{2}\int_A (\sigma_x \varepsilon_x + \sigma_y \varepsilon_y + \tau_{xy}\gamma_{xy})\, dA\,. \tag{4.48}$$

1 The *element shear locking* should not be a source of error, since quadrilateral elements were employed [3, pp. 403–408].

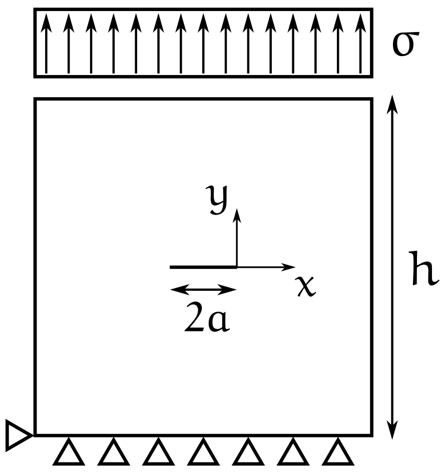


Figure 4.8. Plate weakened by a central crack subjected to a constant tensile stress.

Using the stress-strain equations (1.11) under the plane strain hypothesis, $\mathcal{U}(R)$ becomes:

$$\mathcal{U}(R) = \frac{1+\nu}{2E} \int_A \left[\sigma_x^2 + \sigma_y^2 - \nu \left(\sigma_x + \sigma_y\right)^2 + 2\,\tau_{xy}^2\right] dA\,. \tag{4.49}$$

Introducing equations (1.97) derived in subsection 1.8.4, the integral turns out to be:

$$\begin{aligned}\mathcal{U}(R) &= \frac{(1+\nu)\,K_I^2}{4\pi\,E} \int_0^R \int_{-\pi}^{+\pi} \frac{1}{r} \Big[2\cos^2\tfrac{\vartheta}{2}\left(1+\sin^2\tfrac{\vartheta}{2}\sin^2\tfrac{3\vartheta}{2}\right) \\ &\quad + 2\cos^2\tfrac{\vartheta}{2}\sin^2\tfrac{\vartheta}{2}\cos^2\tfrac{3\vartheta}{2} - 4\nu\cos^2\tfrac{\vartheta}{2}\Big]\, r\, d\vartheta dr \\ &= \frac{(1+\nu)\,K_I^2}{2\pi\,E}\, R \int_{-\pi}^{+\pi} \cos^2\tfrac{\vartheta}{2}\Big[1-2\nu \\ &\quad + \sin^2\tfrac{\vartheta}{2}\left(\sin^2\tfrac{3\vartheta}{2} + \cos^2\tfrac{3\vartheta}{2}\right)\Big]\, d\vartheta\end{aligned} \tag{4.50}$$

whose solution is [8]:

$$\mathcal{U}(R) = \frac{(1+\nu)(5-8\nu)}{8E}\, K_I^2\, R\,. \tag{4.51}$$

The local strain energy density is obtained dividing $\mathcal{U}(R)$ by the area:

$$\mathcal{SED} = \frac{\mathcal{U}(R)}{\pi R^2} = \frac{(1+\nu)(5-8\nu)}{8\pi\,R}\,\frac{K_I^2}{E}\,. \tag{4.52}$$

With $\sigma = 100\,\text{MPa}$ and $2a = 20\,\text{mm}$, the Stress Intensity Factor of mode I turns out to be $K_I = 560.50\,\text{MPa}\sqrt{\text{mm}}$; for the Finite Element analyses, the side h was fixed at $200\,\text{mm}$. The theoretical local strain energy density for $R = 0.3$, 0.5, 1.0, and $2.0\,\text{mm}$ is reported in Table 4.2, together with the corresponding values predicted by the p-FEM code ($\mathcal{SED}_{p\text{-FEM}}$).

Table 4.2. Local strain energy density of a cracked plate for different radii.

R (mm)	$\mathcal{SED}_{\text{th}}$ (N mm/mm^3)	$\mathcal{SED}_{p\text{-FEM}}$ (N mm/mm^3)
0.3	0.670 635	0.675 051
0.5	0.402 381	0.408 650
1.0	0.201 190	0.210 754
2.0	0.100 595	0.112 903

The target of the computation is twofold:

- To determine the influence of the singularity-dominated zone on the convergence of the algorithm, fixing the fineness of the mesh and calculating $\mathcal{SED}$ for different radii.
- To analyse the influence of gradually coarser meshes on the accuracy of the computation, for the case $R = 0.3$ mm.

Also in this case, a Python script was written (see Appendix B). Since the computation is based on the analytical expressions for $\{\boldsymbol{\sigma}\}$ and $\{\mathbf{u}\}$ derived in subsection 1.8.4, the convergence is very fast: Setting $n = 3$ and $m = 1$ allows to get a relative error $e_{\mathcal{SED}} \sim 10^{-14}\%$, for every radius considered. This demonstrates the consistency between the 2-D integral formulation and the numerically computed contour integral, but ignores totally the effect of non-singular terms.

When the FE computation is involved, we expect the solution to converge more slowly: From subsection 1.7.2 we know in fact that cracks induce the strongest singularity possible in elasticity problems, and in section 3.3 we said that such singularity can drastically affect the efficiency of the standard Finite Element Method.

Let us start with the first problem. Thanks to the symmetry of the geometry and the loads, it was possible to analyse only to one fourth of the plate: The mesh consists of 716 quadratic elements for a total of 1505 nodes (see Figure 4.9). The radii investigated are $R = 0.3$, 0.5, 1.0, and 2.0 mm. The subdivisions n are 3, 5, 10, 20, and 40; m goes from 1 to 3.

By looking at the results reported in graphical form in figures 4.10 to 4.13, we can highlight some common aspects:

- The error with respect to the theoretical solution tends to increase with higher radii, while the agreement with the p-FEM solution is always good: This means that non-singular terms are becoming predominant.[2] The case $R = 0.3$ mm is completely K_I-dominated, while for R equal to 0.5 mm one can already notice a slightly higher error (about 1%) in $e_{\mathcal{SED}}$ which is not observed

2 The only case in which $\tilde{e}_{\mathcal{SED}}$ is higher than 1% is for $R = 0.3$ mm. This can be explained with the little difference (0.66%) between $\mathcal{SED}_{\text{th}}$ and $\mathcal{SED}_{p\text{-FEM}}$, which adds to the actual error.

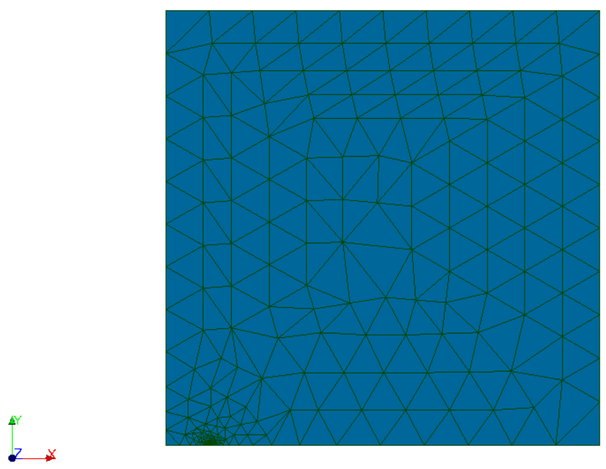
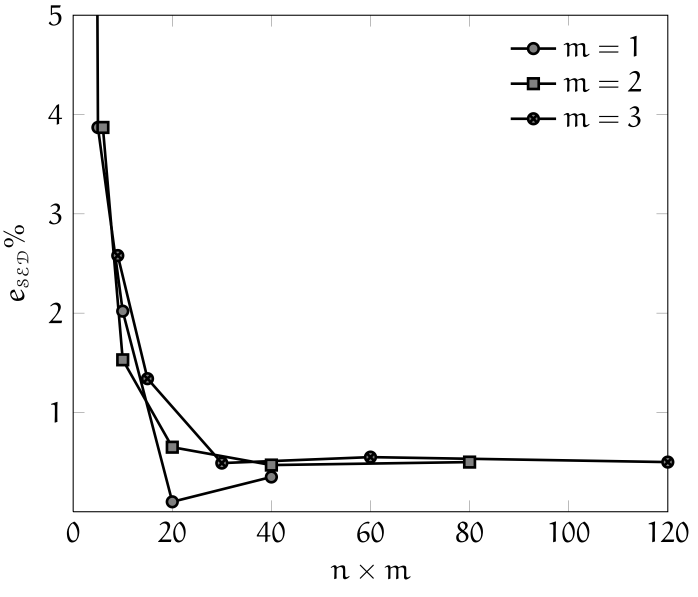


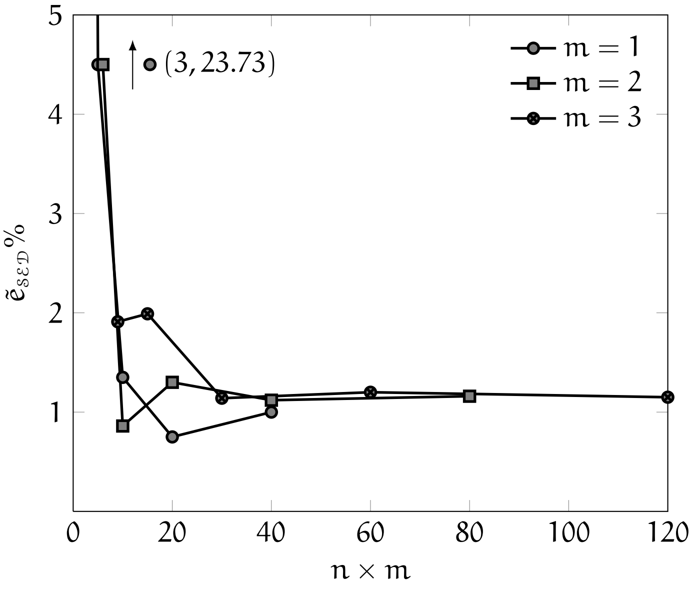
Figure 4.9. Finite Element model of the cracked plate.

when $\tilde{e}_{\mathcal{SED}}$ is considered. For $R = 1.0$ and $2.0\,\text{mm}$, the plots of $e_{\mathcal{SED}}$ and $\tilde{e}_{\mathcal{SED}}$ are almost identical, but translated of a constant quantity due to non-singular terms (whose contribution on $\mathcal{SED}$ is of 3.4 and 11.9%, respectively). This means that these terms are computed exactly with few integration points, and the most significant source of error comes from the singular terms.

- The convergence is quite fast. With a number of integration points equal to 10, the relative error is lower than 1%, except for the case $R = 0.3\,\text{mm}$, where the closeness to the singularity requires $n \times m$ to be slightly higher (between 15 and 20).
- The minimum error is in the neighbourhood of $n \times m = 20$, with slightly better results when $m = 1$. Increasing the number of integration points to 50 or more allows to stabilize the error to values which are a bit higher, although still very small.

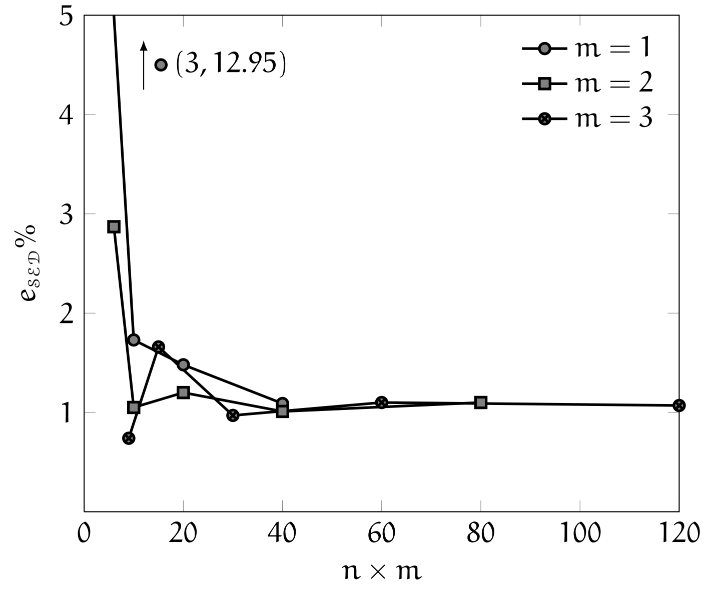


(a) $e_{\mathcal{SED}}$

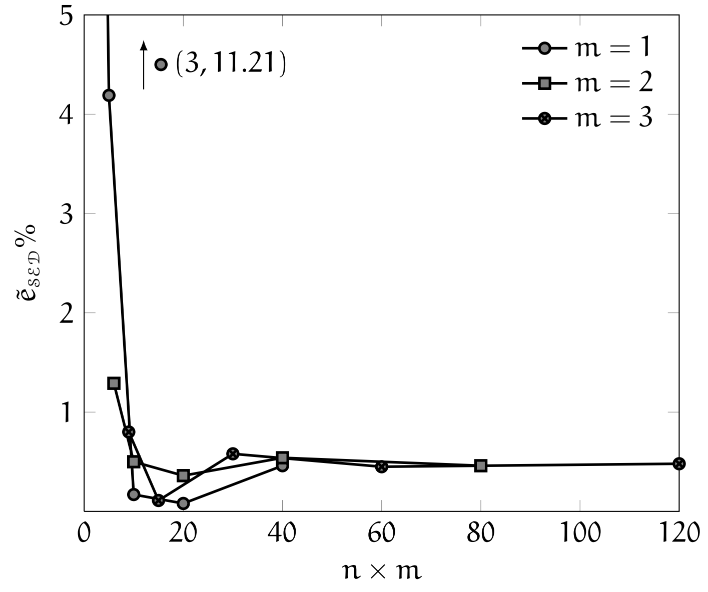


(b) $\tilde{e}_{\mathcal{SED}}$

Figure 4.10. Trend of the relative error of $\mathcal{SED}$ as the number of integration points increases, for $R = 0.3\,\text{mm}$.

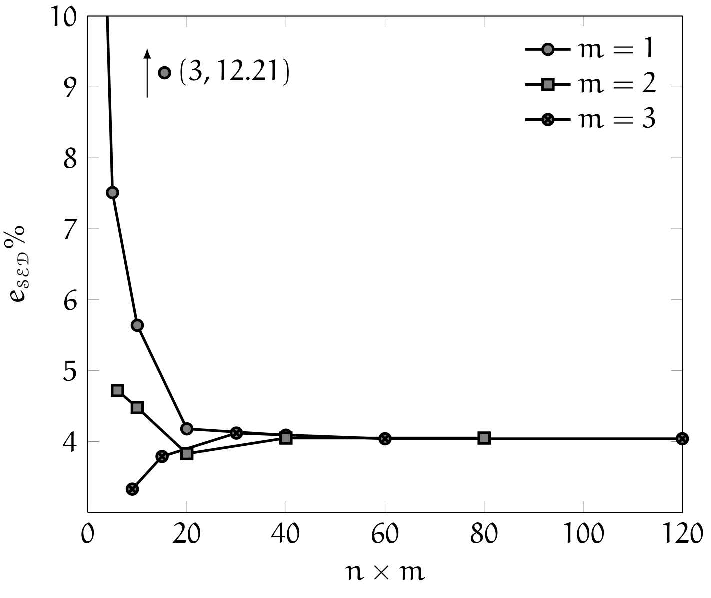


(a) $e_{\mathcal{SED}}$

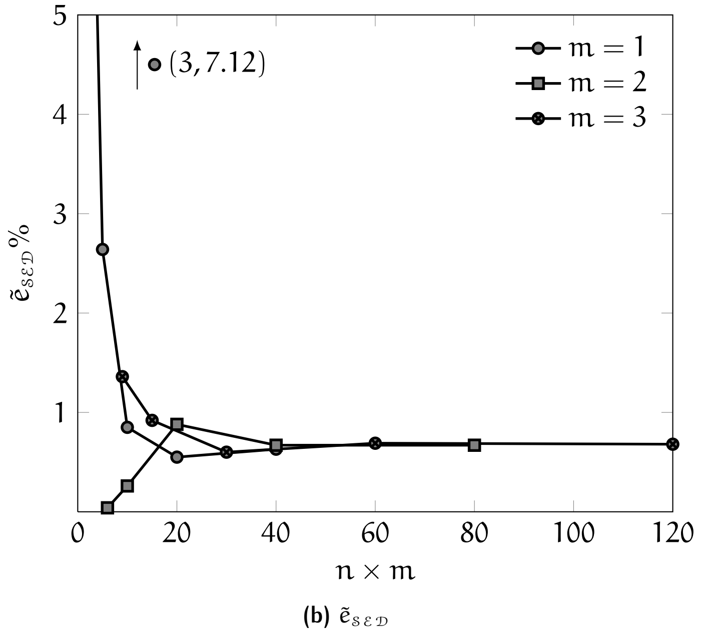


(b) $\tilde{e}_{\mathcal{SED}}$

Figure 4.11. Trend of the relative error of $\mathcal{SED}$ as the number of integration points increases, for $R = 0.5\,\text{mm}$.

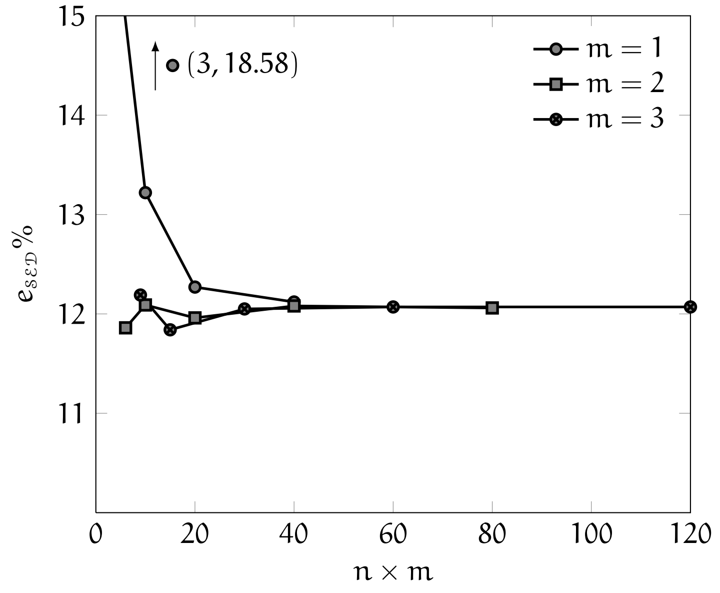


(a) $e_{\mathcal{SED}}$

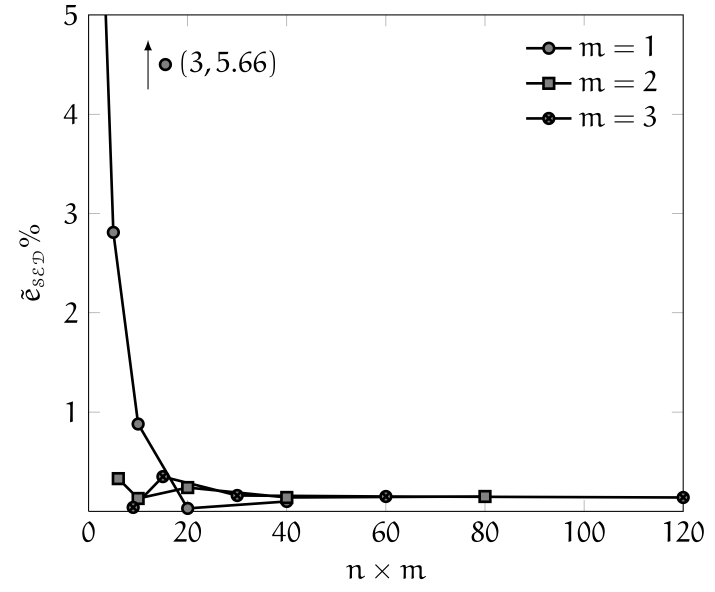


(b) $\tilde{e}_{\mathcal{SED}}$

Figure 4.12. Trend of the relative error of $\mathcal{SED}$ as the number of integration points increases, for $R = 1.0\,\text{mm}$.



(a) $e_{\mathcal{SED}}$



(b) $\tilde{e}_{\mathcal{SED}}$

Figure 4.13. Trend of the relative error of $\mathcal{SED}$ as the number of integration points increases, for $R = 2.0\,\text{mm}$.

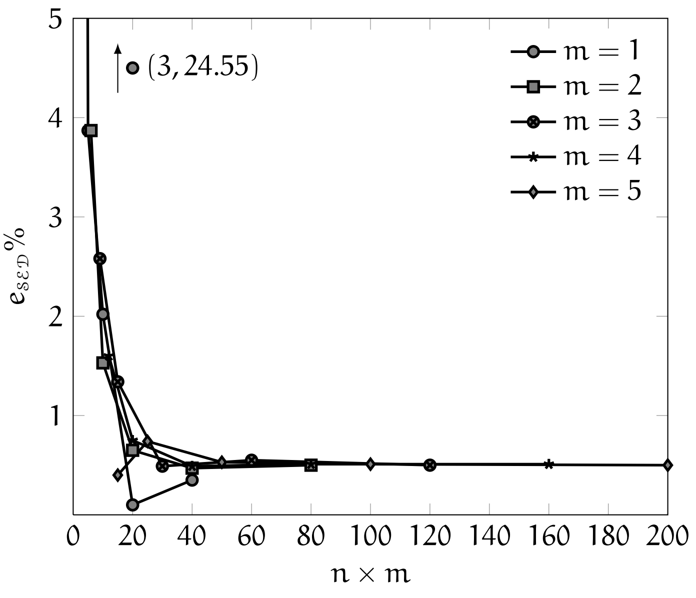
Table 4.3. Meshes used for the analysis of the cracked plate.

Mesh	Elements	Nodes
1	716	1505
2	596	1263
3	487	1042

Our next aim is to determine the influence of the mesh on the accuracy of the computation. To do so, we fix the radius at 0.3 mm and we calculate $\mathcal{SED}$ with gradually coarser meshes. The characteristics of the meshes adopted are reported in Table 4.3; in all the analyses, quadratic elements were employed. The subdivisions chosen are the same of the previous analysis ($n = 3$, 5, 10, 20, and 40), while m goes from 1 to 5. Since the closed-form solution gives an accurate prediction of $\mathcal{SED}$, we are considering only the relative error $e_{\mathcal{SED}}$.

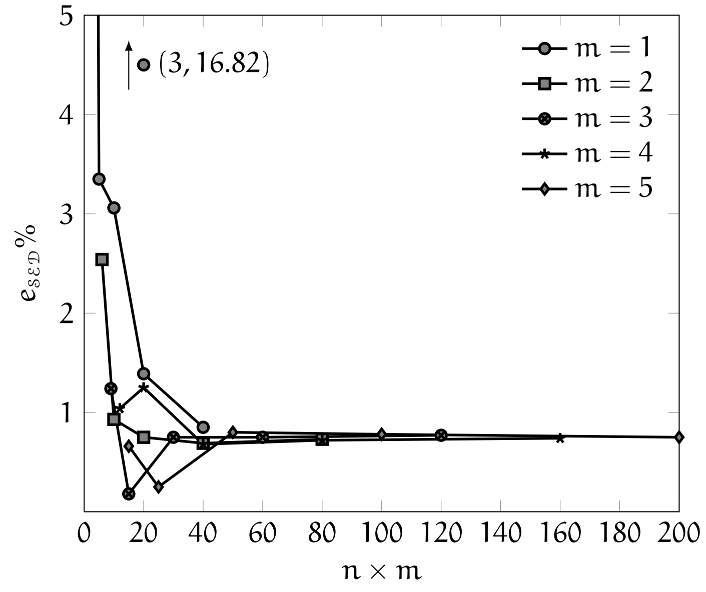
We can summarize the following results (see figures 4.14a to 4.14c):

- With meshes 1 and 2, the convergence is reached quite rapidly; 20 integration points are enough to get a relative error lower than 1%, and better results are obtained when m is between 2 and 4. For $n \times m \geqslant 50$, the error does not vary significantly.
- With mesh 3, the same trend is observed, although $e_{\mathcal{SED}}$ is always higher than 2%. Hence, the mesh is not enough fine to give the same accuracy in the results.

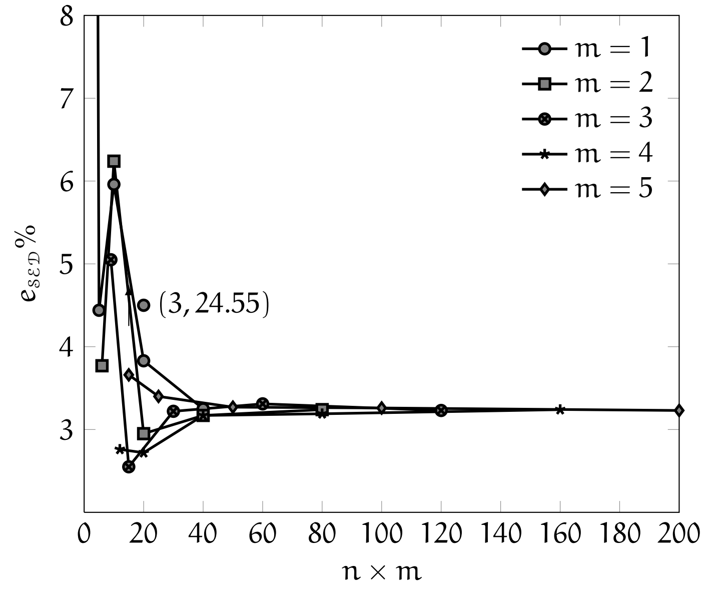


(a) Mesh 1.

Figure 4.14. Trend of the relative error of $\mathcal{SED}$ for a cracked plate, with different meshes. *(cont.)*



(b) Mesh 2.



(c) Mesh 3.

Figure 4.14. Trend of the relative error of $\mathcal{SED}$ for a cracked plate, with different meshes.

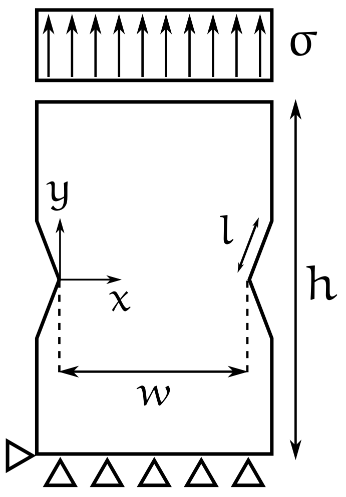


Figure 4.15. Plate weakened by a double 135° sharp V-shaped notch subjected to a constant tensile stress.

4.3.2 Notched plate

After the cracked plate, we use the Algorithm 4.1 to compute the strain energy density of a plate weakened by a double sharp V-shaped notch with an opening angle of 135° (Figure 4.15). The height h is 50 mm, the width of the net section is $w = 40$ mm, and the length of the re-entrant corner's edge is $l = 5$ mm.

Since the singularity exponent is higher than -0.5, it may be that the K_1-dominance region is smaller than the one of the crack. It is therefore necessary to compare $\mathcal{SED}_{\text{th}}$ with $\mathcal{SED}_{p\text{-FEM}}$ to check how much they differ one from the other.

The theoretical strain energy density over a control volume with radius R for a mode I-loaded V-shaped notch is given by the first term in the right-hand side of equation (2.9):

$$\mathcal{SED}_1 = \frac{1}{E}\left[e_1\, K_1^2\, R^{2(\lambda_1-1)}\right] \tag{4.53}$$

where e_1 and λ_1 can be obtained using the data in tables 1.2 and 2.1.

As stated in subsection 1.7.3, the NSIFs do not have a closed-form solution, and their evaluation necessarily requires to use a Finite Element code or other numerical strategies.

In our case, exploiting the symmetry of the geometry and the loads, the analysis was conducted on one fourth of the plate. The mesh consisted in 100 915 elements and 203 950 nodes, and was therefore much more fine than the ones used for computing the local strain energy density.

The procedure followed can be summarized in the following steps:

1. Firstly, the *plateau* region for the NSIFs was determined. This is the most delicate step, since this region cannot include neither the stresses at the nodes very close to the tip, which are not accurately computed by the FEM, nor the ones too far from it, because of the increasing significance of non-singular terms. On the basis of the singularity of σ_ϑ in correspondence of the notch bisector, we identified this zone with the range from 0.01 to $0.3\,\text{mm}$, where the singularity exponent resulted to be -0.3277 (see Figure 4.16); this value differs for less than 0.4% from the one that can be calculated by solving Williams' eigenvalue problem ($1-\lambda_1 = -0.3264$).

2. Secondly, we computed $K_{1,\text{FEM}}$ at each nodal point. The definition of this quantity is similar to the one given in (1.50a), but without the limit:

$$K_{1,\text{FEM}} = \sqrt{2\pi}\, r_{(i)}^{1-\lambda_1}\, \sigma_{\vartheta,(i)}(\vartheta = 0) \tag{4.54}$$

   where (i) represents the node considered. It is important that $K_{1,\text{FEM}}$ does not vary significantly in the selected range: By looking at Figure 4.17, we see that this condition was satisfied.

3. Finally, the estimate of K_1 was obtained by averaging $K_{1,\text{FEM}}$ calculated at each node of the range:

$$K_1 \approx \frac{1}{N}\sum_{i=1}^{N} K_{1,\text{FEM}}^{(i)} = \frac{1}{N}\sum_{i=1}^{N} \sqrt{2\pi}\, r_{(i)}^{1-\lambda_1}\, \sigma_{\vartheta,(i)}(\vartheta = 0) \tag{4.55}$$

   where N is the number of nodes inside the plateau.

In this way, we obtained $K_1 = 379.56\,\text{MPa}\,\text{mm}^{0.326}$; together with $\lambda_1 = 0.6736$ and $e_1 = 0.1172$, it results $\mathcal{SED}_1 = 0.176\,460\,\text{N}\,\text{mm}/\text{mm}^3$, which is almost identical to the value of $0.176\,347\,\text{N}\,\text{mm}/\text{mm}^3$ predicted by the *p*-FEM code. It is therefore completely legitimate to use the theoretical strain energy density as a reference value.

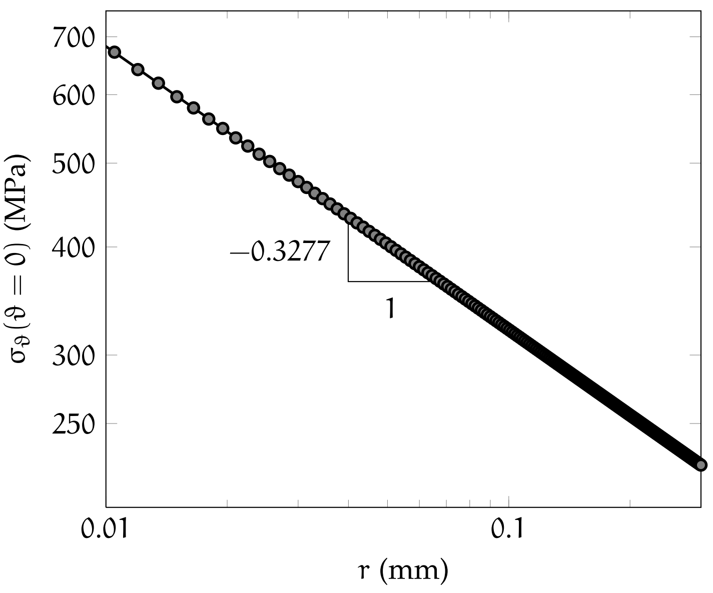


Figure 4.16. Determination of the plateau by the singularity of $\sigma_\vartheta(\vartheta = 0)$.

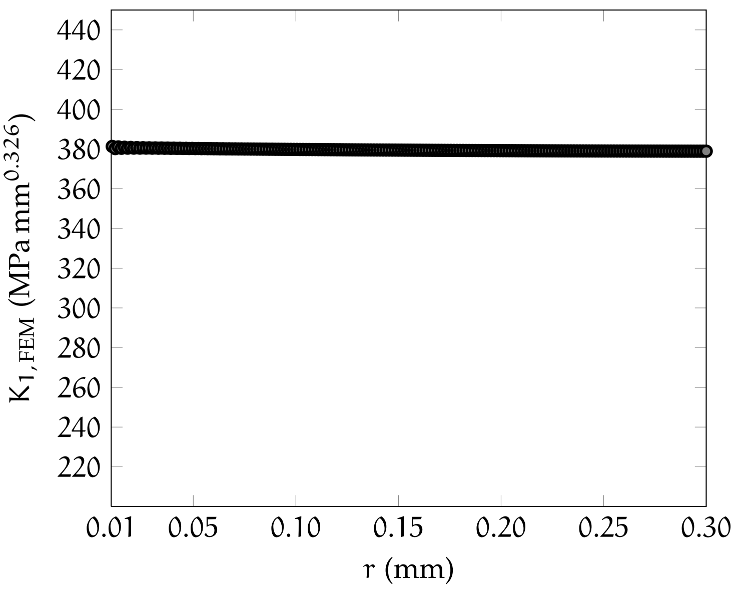


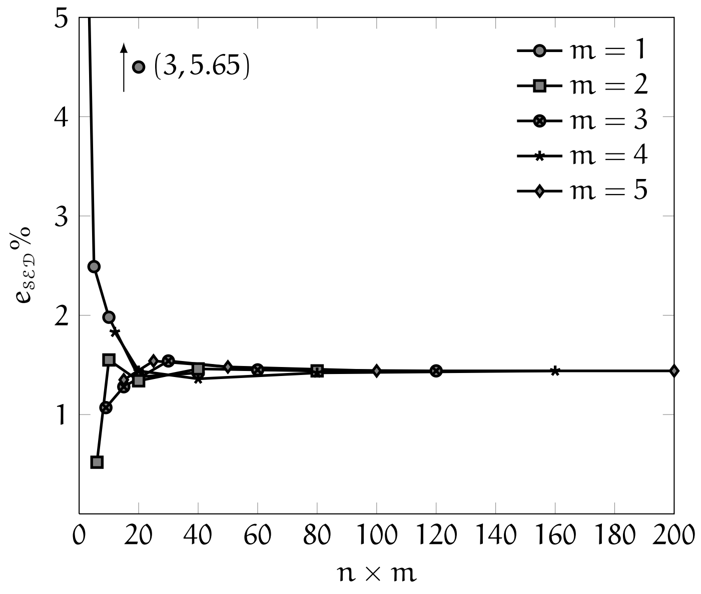
Figure 4.17. Trend of $K_{1,\mathrm{FEM}}$ inside the plateau zone.

Table 4.4. Meshes used for the analysis of the notched plate.

Mesh	Elements	Nodes
1	674	1423
2	539	1152
3	428	923

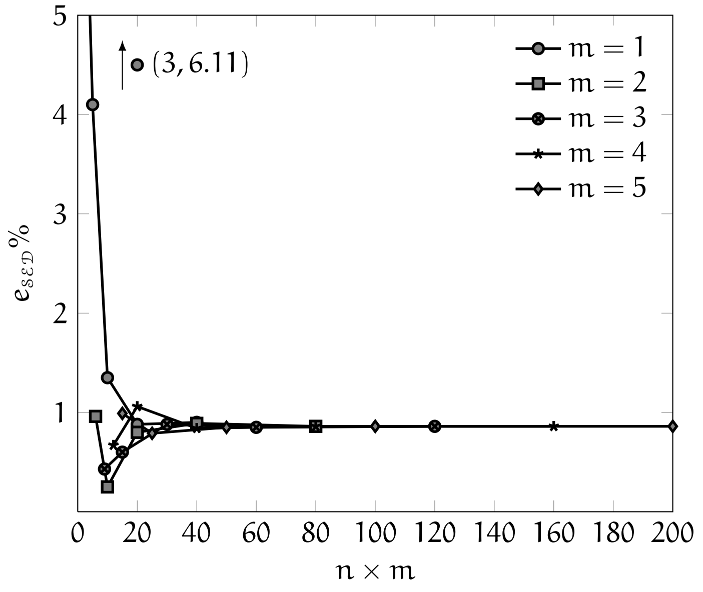
As in the previous case, we want to determine the influence of the element size on the results. For this reason, three different meshes were constructed, using quadratic elements. The number of elements and nodes for each mesh is reported in Table 4.4. Looking at the trends of the error reported in figures 4.18a to 4.18c, we deduce that:

- The relative error is subjected to a slight decrease when coarser meshes are adopted. This may be explained with the fact that (i) the stresses at the source of a singularity increase when the mesh is locally refined and (ii) the singularity induced by a notch is weaker than the one induced by a crack: Therefore, not too fine meshes allow to compute satisfactorily the stresses at a certain distance from the tip, and at the same time are less affected from the error originated at the tip, which gets redistributed to the neighbouring nodes.
- The best results are obtained with meshes 2 and 3, when 20 integration points are used and $m = 1$ or 2. The error stabilizes when $n \times m \geqslant 50$.

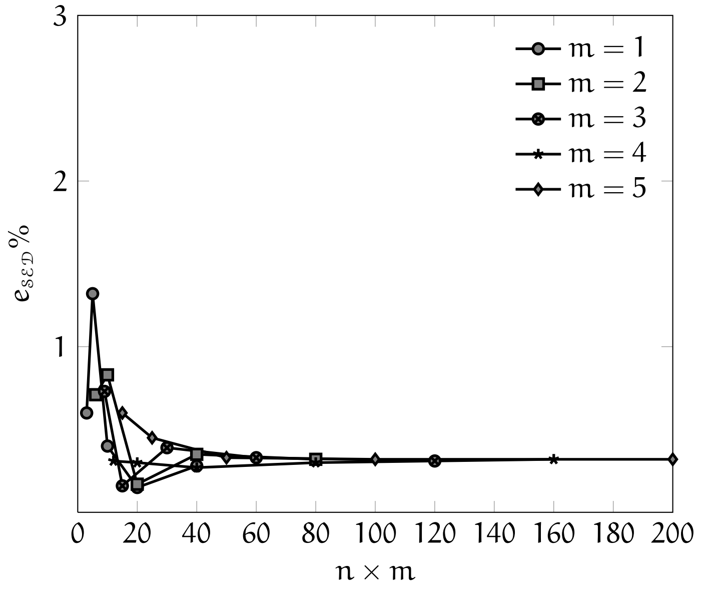


(a) Mesh 1.

Figure 4.18. Trend of the relative error of $\mathcal{SED}$ for a notched plate, with different meshes. *(cont.)*



(b) Mesh 2.



(c) Mesh 3.

Figure 4.18. Trend of the relative error of $\mathcal{SED}$ for a notched plate, with different meshes.

4.4 COMPARISON OF THE FORMULATIONS

Once determined the local strain energy density of a cracked and notched plate, for a radius $R = 0.3\,\text{mm}$, we want to compare the numerically efficiency of the 1-D integral formulation, reported in equation (4.2), with respect to the 2-D one, given by equation (4.1).

4.4.1 Cracked plate

For the case of the crack, $\mathcal{SED}$ was computed both with the standard FEM and the extended XFEM. The comparison was realized analysing the relative error $e_{\mathcal{SED}}$ as the number of degrees of freedom increases. For the XFEM analyses, the DOF were estimated directly from the size of the stiffness matrix.

From the comparison shown in Figure 4.19, one can infer that:

- The computation of the contour integral is much more efficient than the one of the 2-D integral. About 3000 degrees of freedom are enough to get a relative error lower than 1%, while the double integral formulation requires at least 10^5 DOF.
- The coupling of 5 Gaussian points with 40 subdivisions gives better results with coarser meshes, while $m = 1$ and $n = 20$ is slightly more efficient when 3000 DOF are employed.
- Although neither of the simulations based on the 2-D integral formulation allow to lower the error to less than 1%, the extended FEM is more advantageous than the standard FEM. In fact, (i) the XFEM requires less DOF to reach the same error ($e_{\mathcal{SED}} = 2.64\%$ for 299 304 DOF against $e_{\mathcal{SED}} = 2.70\%$ for 635 518 DOF with standard FEM) and (ii) the decreasing trend with the XFEM starts at $\sim 10^4$, while with the standard FEM it increases of more than 1% in the last simulation, thus demonstrating that the convergence is not yet stable.

4.4.2 Notched plate

The considerations made for the previous case are still valid, except for two things:

- In the last three simulations, the relative error is subjected to minor variations. This means that the convergence is probably reached, and a further decrease of $e_{\mathcal{SED}}$ should not be expected when finer meshes are constructed.
- For the 1-D integral formulation, the error increases of approximately 1% when the mesh is locally refined. A possible explanation for this observation was given in subsection 4.3.2.

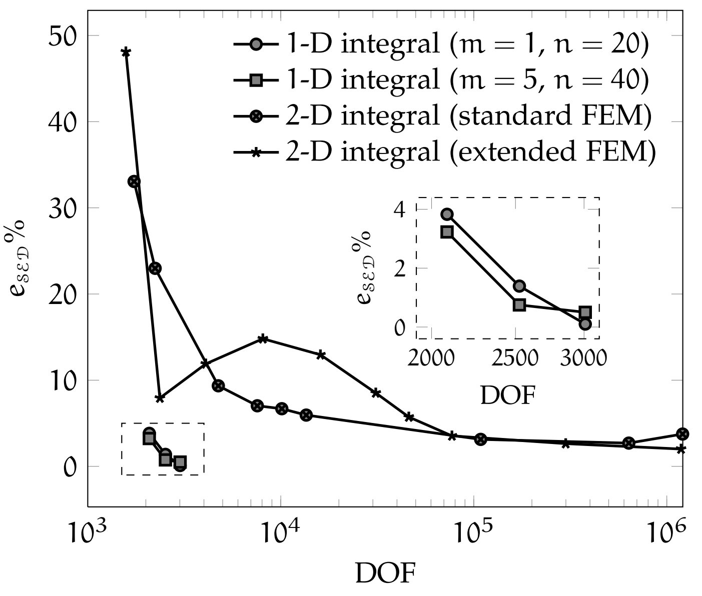


Figure 4.19. Comparison of the numerical efficiency of 1-D and 2-D integrals, for the cracked plate. In the smaller chart, a magnification of the curves inside the dashed box.

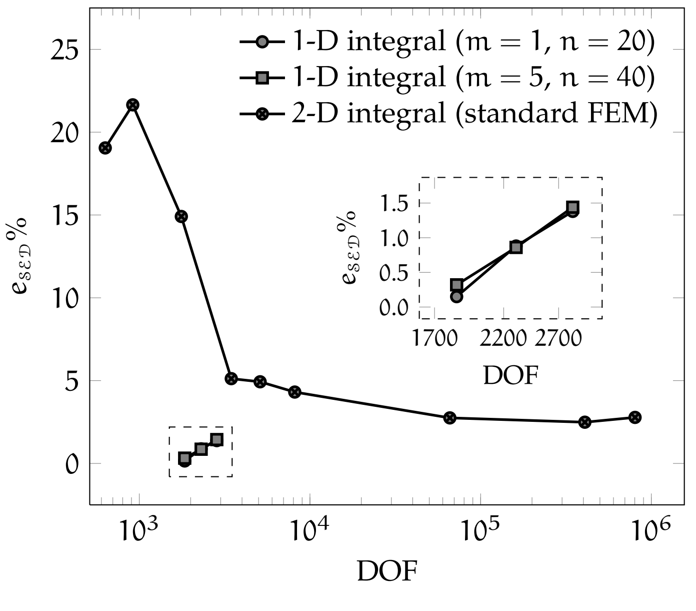


Figure 4.20. Comparison of the numerical efficiency of 1-D and 2-D integrals, for the notched plate. In the smaller chart, a magnification of the curves inside the dashed box.

5 CONCLUSIONS

The purpose of this work was to improve the numerical efficiency of the computation of the local strain energy density in presence of elastic singularities. The average of such quantity on a material dependent-volume, according to the SED criterion, can be used to assess the fatigue life of welded joints.

In order to achieve this target, a twofold approach was followed:

- On one side, an extensive study on the theory of singularities in elasticity was conducted; in particular, the well-known solutions of Westergaard and Williams were derived and analysed in view of their numerical implementation.
- On the other side, a numerical procedure that allowed to perform the computation of the two-dimensional strain energy density on a finite volume, based on a contour integral formulation, was realized and implemented inside the code used for the Finite Element analyses.

The resulting algorithm was checked in three different test cases, for which the analytical expressions of stresses and displacements were derived. Three Python scripts were written, in order to compare the theoretical strain energy density with the one obtained with the algorithm. Once verified that the results were matching, the algorithm was coupled with the commands of the Finite Element code, so to switch from the exact stresses and displacements to the approximated ones. All the checks were then repeated, confirming the previous trend.

At this point, the combination of the algorithm with the Finite Element code was applied to two different configurations of practical interest: A plate weakened respectively by a central crack and a 135° V-shaped edge notch. For the case of the crack, the local strain energy density was computed for different radii, so to determine the contributions of singular and non-singular terms to the error. It was thus noticed that the Finite Element Method allows to compute easily non-singular terms, and that the main source of error is therefore due to the singularity. The analysis of the influence of the mesh on the accuracy of the numerical solution demonstrated that the algorithm is not very sensible to the size of the local elements. The same result was confirmed when the notched plate was considered, although a slight increase of the error for finer meshes was observed; in this case, the calculation of the theoretical value for comparison required to estimate numerically the Notch Stress Intensity Factor of mode I. In both cases, the influence of the number of integration points was also taken in account, leading to the same conclusion in terms of the best combination of number of subdivisions and Gaussian points.

Finally, the comparison of the contour integral and double integral formulation highlighted the better efficiency of the first. In fact, the contour integral formulation (i) showed a faster convergence, (ii) required a number of degrees of freedom about three orders of magnitude lower than the one based on the double integral and (iii) led to a lower final error. For the case of the cracked plate, the double integral was computed both with the standard and the extended FEM: The latter was more advantageous than the first, because it converged more stably and with greater accuracy.

This approach demonstrated thus to be flexible, efficient, and reliable:

- It is *flexible*, because the algorithm was adapted to different configurations with only minor changes;
- It is *efficient*, since it requested a narrow number of integration points to get the convergence;
- It is *reliable*, since the final error with respect to the reference solution (theoretical or numerical, depending on the case) was always almost negligible.

We conclude this work with some suggestions for the possible further research in this topic:

GENERALIZING THE XFEM: Because of some limitations of the Finite Element code adopted, it was not possible to implement the enrichment functions for the case of the notch. Although the singularity in this case is less severe than the one induced by a crack, this could lead to better results, especially in view of three-dimensional simulations.

COMBINING XFEM AND CONTOUR INTEGRAL: Another improvement could be the combination of the extended FEM with the contour integral formulation proposed in this work; this may require to modify directly the Finite Element code used, since for the moment it allows to use the XFEM only for the computation of double integrals.

SWITCHING TO 3-D: It is well known that the efficiency of the Finite Element Method in three dimensions is not as good as in two dimensions. Using Green's theorem to switch from a volume integral to a a surface integral could probably improve significantly the convergence of the method.

A SHAPE FUNCTIONS

In this appendix, we are going to describe briefly some properties of the shape functions. For the sake of simplicity, we are referring to the *p*-dimensional space $\mathcal{S}^p(I_{st})$, where $I_{st} = \{\xi | -1 \leqslant \xi \leqslant +1\}$. The definitions can be easily extended to higher dimensions using the space product.

A.1 LAGRANGE SHAPE FUNCTIONS

The first shape functions that we describe are LAGRANGE polynomials, defined as:

$$N_i(\xi) = \prod_{\substack{k=1 \\ k \neq i}}^{p+1} \frac{\xi - \xi_k}{\xi_i - \xi_k}, \quad \text{for } i = 1, 2, \ldots, p+1 \tag{A.1}$$

These polynomials have the KRONECKER-δ *property*:

$$N_i(\xi_j) = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j\,. \end{cases} \tag{A.2}$$

Another fundamental property of these polynomials is the ability to build a partition of unity over the domain $\mathcal{S}^p(I_{st})$:

$$\sum_{i=1}^{p+1} N_i(\xi) = 1\,. \tag{A.3}$$

Thanks to the simplicity of their construction, Lagrange shape functions are implemented in every Finite Element code.

A.2 HIERARCHIC SHAPE FUNCTIONS

The increase of order of a Lagrange shape function is usually achieved by adding mid-side nodes within the elements, thus switching from linear to quadratic elements. A different approach is to build a high-order shape function by adding high-order terms. This procedure leads to the formulation of the so-called *hierarchic shape functions*. This name comes from the fact that the low-order components are not affected by the introduction of new higher order terms, contrary to Lagrange shape functions [37, p. 70].

One of the methods used to build hierarchic shape functions is based on the Legendre polynomials. The Legendre polynomial of order p is:

$$P_p(\xi) = \frac{1}{2^p p!} \frac{d^p}{d\xi^p}[(\xi^2 - 1)^n] \,. \tag{A.4}$$

Given the first two polynomials, respectively $P_0(\xi) = 1$ and $P_1(\xi) = \xi$, we can introduce an alternative definition, based on the recursive formula

$$(p+1)\,P_{p+1}(\xi) = (2p+1)\,\xi\,P_p(\xi) - p\,P_{p-1}(\xi) \,. \tag{A.5}$$

The corresponding shape functions are obtained upon integration:

$$N_i(\xi) = \sqrt{\frac{2i-3}{2}} \int_{-1}^{\xi} P_{i-2}(t)\,dt \,, \quad \text{for } i = 3, 4, \ldots, p+1 \,. \tag{A.6}$$

The hierarchical shape functions are orthogonal, that is:

$$\int_{-1}^{+1} \frac{dN_i}{d\xi}\frac{dN_j}{d\xi}\,d\xi = \delta_{ij} \,, \quad \text{for } i,j > 3 \tag{A.7}$$

which is an extremely useful property for Finite Elements, since it allows to reduce significantly the non-zero components of the $[\mathbf{B}]$ matrix (see subsection 3.2.6). The first five shape functions are here reported [37, pp. 72–73]:

$$\begin{aligned}
N_1(\xi) &= \frac{1}{2}(1-\xi) \,, \\
N_2(\xi) &= \frac{1}{2}(1+\xi) \,, \\
N_3(\xi) &= \frac{\sqrt{3}}{2\sqrt{2}}(\xi^2 - 1) \,, \\
N_4(\xi) &= \frac{\sqrt{5}}{2\sqrt{2}}\,\xi\,(\xi^2 - 1) \,, \\
N_5(\xi) &= \frac{\sqrt{7}}{8\sqrt{2}}(5\,\xi^4 - 6\,\xi^2 + 1) \,.
\end{aligned} \tag{A.8}$$

It is interesting to notice that for $i \geqslant 3$ they become zero at the extrema of the interval:

$$N_i(-1) = N_i(+1) = 0 \,. \tag{A.9}$$

B PYTHON SCRIPTS

In this appendix, we report all the Python scripts used for validating the Algorithm 4.1; the script for the cracked plate is also included, since in this case the asymptotical stresses and displacements are known (see subsection 1.8.4). In order to save some space, we omitted to write the Gauss-Legendre abscissas and weights.

B.1 PLATE_CNST_SED.PY

Algorithm B.1. Computation of $\mathcal{SED}$ for a plate subjected to a constant tensile stress (subsection 4.2.1).

```
f = open('py_plate_cnst_sed.dat','w')

import math
import random

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
   ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
   ⋮
}

print >> f, '=============================================================\n'
print >> f, '    SCRIPT FOR THE COMPUTATION OF THE LOCAL STRAIN ENERGY\n\
      DENSITY OF A STEEL PLATE SUBJECTED TO A CONSTANT\n\
         TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n'
print >> f, '=============================================================\n'

print >> f, '\
                _____________________\n\
              >|                     | --->\n\
              >|                     | --->\n\
              >|                     | --->\n\
              >|                     | --->\n\
    DX = 0    >|                     | --->    S0 = 100 MPa\n\
              >|                     | --->\n\
              >|                     | --->\n\
              >|                     | --->\n\
              >|_____________________| --->\n\
                ^\n\
             DY = 0\n\n'

# Input for the values R, q, n, m

print >> f, 'INPUT VALUES:\n\n'

R = input('Enter the radius of the circles onto which compute the SED: ')

print >> f, 'Radius of the circles: R =', R, '\n'

q = input('Enter the number of random points: ')

print >> f, 'Number of random points: q =', q, '\n'

n = input('Enter the number of subdivisions for each circumference: ')

print >> f, 'Number of subdivisions for each circumference: n =', n, '\n'

m = input('Enter the number of Gaussian points for each subdivision: ')

print >> f, 'Number of Gaussian points for each subdivision: m =', m, '\n'

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young modulus of steel [MPa]
NU = 0.3        # Poisson ratio of steel []

# Geometry

h = 100.0       # Length of the plate's edge [mm]

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
A = 0.5 * (theta_b - theta_a) * R ** 2

for k in range(q):

    # Definition of the point coordinates

    x_c = random.uniform(R, h - R)
    y_c = random.uniform(R, h - R)

    # Definition of the initial values

    SE = 0.0
    SED = 0.0
    p = 0.0

    print >> f, '==================================================\n'
    print >> f, '                    CIRCLE', k + 1, '\n'
    print >> f, '==================================================\n'

    for i in range(1, n + 1):

        a = 0

        print >> f, '----------------------------------------\n'
        print >> f, '             SUBDIVISION', i, '\n'
        print >> f, '----------------------------------------\n'

        # Definition of the angular quantities

        theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
        theta_2 = theta_a + i * (theta_b - theta_a) / n
        dtheta = 0.5 * (theta_2 - theta_1)

        print >> f, 'theta_1 =', theta_1, '\n'
        print >> f, 'theta_2 =', theta_2, '\n'
        print >> f, 'dtheta =', dtheta, '\n'

        for j in range(m):

            print >> f, '------------------------\n'
            print >> f, '      ITERATION', j + 1, '\n'
            print >> f, '------------------------\n'

            # Calculation of the desired quantities

            # Theta angle

            t = T.get(m)[m - j - 1]
            theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2

            # Normals

            n_x = math.cos(theta)
            n_y = math.sin(theta)

            # Point coordinates

            x = x_c + R * math.cos(theta)
            y = y_c + R * math.sin(theta)

            # Stresses

            S_xx = S0
            S_yy = 0.0
            S_xy = 0.0

            # Displacements

            u_x =  (1.0 - NU ** 2) * S0 * x / E
            u_y = -NU * (1.0 + NU) * S0 * y / E

            # Traction vectors

            T_x = S_xx * n_x + S_xy * n_y
            T_y = S_xy * n_x + S_yy * n_y

            # Strain energy

            SE = 0.5 * (T_x * u_x + T_y * u_y) * R * dtheta * W.get(m)[a]

            # Strain energy density

            SED += SE / A

            # Perimeter

            p += R * dtheta * W.get(m)[a]

            print >> f, 'Gaussian coordinate t =', t, '\n'
            print >> f, 'theta =', theta, '\n'

            print >> f, 'n_x =', n_x, '\n'
            print >> f, 'n_y =', n_y, '\n'

            print >> f, 'x =', x, '\n'
            print >> f, 'y =', y, '\n'

            print >> f, 'S_xx =', S_xx, '\n'
            print >> f, 'S_yy =', S_yy, '\n'
            print >> f, 'S_xy =', S_xy, '\n'

            print >> f, 'u_x =', u_x, '\n'
            print >> f, 'u_y =', u_y, '\n'

            print >> f, 'T_x =', T_x, '\n'
            print >> f, 'T_y =', T_y, '\n'

            print >> f, 'Strain Energy =', SE, '\n'
            print >> f, 'Strain Energy Density =', SED, '\n'

            print >> f, 'Perimeter =', p, '\n'

            a += 1

        # Definition of the theoretical value for the SED

        REF = 0.5 * (1.0 - NU ** 2) * S0 ** 2 / E

        # Printing of the final values

    print >> f, '\n===================================================\n'
    print >> f, '              RESULTS FOR CIRCLE', k + 1
    print >> f, '\n===================================================\n'

    print >> f, '  x_c =', x_c, ',     y_c =', y_c, '\n'

    print >> f, '         Computed SED =', SED, '\n'
    print >> f, '      Theoretical SED =', REF, '\n'
    print >> f, '     Percentual error =', abs(SED / REF - 1.0) * 100.0, '%\n'

    print >> f, '   Length of the path =', p

    print >> f, '\n===================================================\n\n\n'

f.close()
```

B.2 PLATE_LNR_SED.PY

Algorithm B.2. Computation of $\mathcal{SED}$ for a plate subjected to a linear tensile stress (subsection 4.2.2).

```
f = open('py_plate_lnr_sed.dat','w')

import math
import random

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

print >> f, '=============================================================\n'
print >> f, '    SCRIPT FOR THE COMPUTATION OF THE LOCAL STRAIN ENERGY\n\
       DENSITY OF A STEEL PLATE SUBJECTED TO A LINEAR\n\
          TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n'
print >> f, '=============================================================\n'

print >> f, '                                      SMIN = 0 MPa\n\
                _____________________\n\
              >|                     | \\\n\
              >|                     | >\\\n\
              >|                     | ->\\\n\
              >|                     | -->\\\n\
    DX = 0    >|                     | --->\\\n\
              >|                     | ---->\\\n\
              >|                     | ----->\\\n\
              >|                     | ------>\\\n\
              >|_____________________| ------->\\\n\
                ^\n\
             DY = 0                   SMAX = 100 MPa\n\n'

# Input for the values R, q, n, m

print >> f, 'INPUT VALUES:\n\n'

R = input('Enter the radius of the circles onto which compute the SED: ')

print >> f, 'Radius of the circles: R =', R, '\n'

q = input('Enter the number of random points: ')

print >> f, 'Number of random points: q =', q, '\n'

n = input('Enter the number of subdivisions for each circumference: ')

print >> f, 'Number of subdivisions for each circumference: n =', n, '\n'

m = input('Enter the number of Gaussian points for each subdivision: ')

print >> f, 'Number of Gaussian points for each subdivision: m =', m, '\n'

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young modulus of steel [MPa]
NU = 0.3        # Poisson ratio of steel []

# Geometry

h = 100.0       # Length of the plate's edge [mm]

# Boundary conditions

SM = 100.0      # Maximum applied tensile stress [MPa]

```

```
# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
A = 0.5 * (theta_b - theta_a) * R ** 2

for k in range(q):

    # Definition of the point coordinates

    x_c = random.uniform(R, h - R)
    y_c = random.uniform(R, h - R)

    # Definition of the initial values

    SE = 0.0
    SED = 0.0
    p = 0.0

    print >> f, '==================================================\n'
    print >> f, '                    CIRCLE', k + 1, '\n'
    print >> f, '==================================================\n'

    for i in range(1, n + 1):

        a = 0

        print >> f, '----------------------------------------\n'
        print >> f, '             SUBDIVISION', i, '\n'
        print >> f, '----------------------------------------\n'

        # Definition of the angular quantities

        theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
        theta_2 = theta_a + i * (theta_b - theta_a) / n
        dtheta = 0.5 * (theta_2 - theta_1)

        print >> f, 'theta_1 =', theta_1, '\n'
        print >> f, 'theta_2 =', theta_2, '\n'
        print >> f, 'dtheta =', dtheta, '\n'

        for j in range(m):

            print >> f, '------------------------\n'
            print >> f, '      ITERATION', j + 1, '\n'
            print >> f, '------------------------\n'

            # Calculation of the desired quantities

            # Theta angle

            t = T.get(m)[m - j - 1]
            theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2

            # Normals

            n_x = math.cos(theta)
            n_y = math.sin(theta)

            # Point coordinates

            x = x_c + R * math.cos(theta)
            y = y_c + R * math.sin(theta)

            # Stresses

            S_xx = SM * (1.0 - y / h)
            S_yy = 0.0
            S_xy = 0.0

            # Displacements

            u_x  = (1.0 - NU ** 2) * SM * (1.0 - y / h) * x / E

            u_y  = 0.5 * x ** 2 / h
            u_y -= NU * (1.0 - 0.5 * y / h) * y / (1.0 - NU)
            u_y *= (1.0 - NU ** 2) * SM / E

            # Traction vectors

            T_x = S_xx * n_x + S_xy * n_y
            T_y = S_xy * n_x + S_yy * n_y

            # Strain energy

            SE = 0.5 * (T_x * u_x + T_y * u_y) * R * dtheta * W.get(m)[a]

            # Strain energy density

            SED += SE / A

            # Perimeter

            p += R * dtheta * W.get(m)[a]

            print >> f, 'Gaussian coordinate t =', t, '\n'
            print >> f, 'theta =', theta, '\n'

            print >> f, 'n_x =', n_x, '\n'
            print >> f, 'n_y =', n_y, '\n'

            print >> f, 'x =', x, '\n'
            print >> f, 'y =', y, '\n'

            print >> f, 'S_xx =', S_xx, '\n'
            print >> f, 'S_yy =', S_yy, '\n'
            print >> f, 'S_xy =', S_xy, '\n'

            print >> f, 'u_x =', u_x, '\n'
            print >> f, 'u_y =', u_y, '\n'

            print >> f, 'T_x =', T_x, '\n'
            print >> f, 'T_y =', T_y, '\n'

            print >> f, 'Strain Energy =', SE, '\n'
            print >> f, 'Strain Energy Density =', SED, '\n'

            print >> f, 'Perimeter =', p, '\n'

            a += 1

        # Definition of the theoretical value for the SED

    REF  = 0.25 * R ** 2 + (h - y_c) ** 2
    REF *= 0.5 * (1.0 - NU ** 2) * (SM / h) ** 2 / E

        # Printing of the final values

    print >> f, '\n==================================================\n'
    print >> f, '              RESULTS FOR CIRCLE', k + 1
    print >> f, '\n==================================================\n'
    print >> f, '  x_c =', x_c, ',     y_c =', y_c, '\n'

    print >> f, '         Computed SED =', SED, '\n'
    print >> f, '      Theoretical SED =', REF, '\n'
    print >> f, '     Percentual error =', abs(SED / REF - 1.0) * 100.0, '%\n'

    print >> f, '   Length of the path =', p

    print >> f, '\n===================================================\n\n\n'

f.close()
```

B.3 BEAM_END_SED.PY

Algorithm B.3. Computation of $\mathcal{SED}$ for a beam subjected to an end load (subsection 4.2.3).

```
f = open('py_beam_end_sed.dat','w')

import math
import random

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

print >> f, '==========================================================\
=============\n'
print >> f, '     SCRIPT FOR THE COMPUTATION OF THE LOCAL STRAIN ENERGY \
DENSITY\n\
   OF A STEEL BEAM SUBJECTED TO A END LOAD THROUGH A CONTOUR INTEGRAL\n'
print >> f, '==========================================================\
=============\n'

print >> f, '\
              ______________________________\n\
             |                              |/\n\
       ||    |                              |/\n\
       ||    |                              |/\n\
       ||    |                              |/     DX = 0,\n\
       ||    |                              |/\n\
      _||_   |                              |/     DY = 0\n\
      \  /   |                              |/\n\
       \/    |                              |/\n\
             |______________________________|/\n\
   F = 100 N\n'

# Input for the values R, q, n, m
```

```

print >> f, 'INPUT VALUES:\n\n'

R = input('Enter the radius of the circles onto which compute the SED: ')

print >> f, 'Radius of the circles: R =', R, '\n'

q = input('Enter the number of random points: ')

print >> f, 'Number of random points: q =', q, '\n'

n = input('Enter the number of subdivisions for each circumference: ')

print >> f, 'Number of subdivisions for each circumference: n =', n, '\n'

m = input('Enter the number of Gaussian points for each subdivision: ')

print >> f, 'Number of Gaussian points for each subdivision: m =', m, '\n'

# Definition of some parameters of the problem

# Material

E  = 210000.0                # Young modulus of steel [MPa]
NU = 0.3                     # Poisson ratio of steel []
G  = 0.5 * E / (1.0 + NU)    # Shear modulus of steel [MPa]

# Geometry

b = 1.0                      # Thickness of the beam [mm]
L = 100.0                    # Length of the beam [mm]
h = 10.0                     # Height of half beam [mm]
I = 2.0 * b * h ** 3 / 3.0   # Moment of inertia [mm ^ 4]

# Boundary conditions

F = 100.0                    # Applied end load [N]

# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
A = 0.5 * (theta_b - theta_a) * R ** 2

for k in range(q):

    # Definition of the point coordinates

    x_c = random.uniform(R, L - R)
    y_c = random.uniform(-h + R, h - R)

    # Definition of the initial values

    SE = 0.0
    SED = 0.0
    p = 0.0

    print >> f, '==================================================\n'
    print >> f, '                    CIRCLE', k + 1, '\n'
    print >> f, '==================================================\n'

    for i in range(1, n + 1):

        a = 0

        print >> f, '----------------------------------------\n'
        print >> f, '            SUBDIVISION', i, '\n'
        print >> f, '--------------------------------------\n'

        # Definition of the angular quantities

        theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
        theta_2 = theta_a + i * (theta_b - theta_a) / n
        dtheta = 0.5 * (theta_2 - theta_1)

        print >> f, 'theta_1 =', theta_1, '\n'
        print >> f, 'theta_2 =', theta_2, '\n'
        print >> f, 'dtheta =', dtheta, '\n'

        for j in range(m):

            print >> f, '-----------------------\n'
            print >> f, '      ITERATION', j + 1, '\n'
            print >> f, '-----------------------\n'

            # Calculation of the desired quantities

            # Theta angle

            t = T.get(m)[m - j - 1]
            theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2

            # Normals

            n_x = math.cos(theta)
            n_y = math.sin(theta)

            # Point coordinates

            x = x_c + R * math.cos(theta)
            y = y_c + R * math.sin(theta)

            # Stresses

            S_xx = F * x * y / I
            S_yy = 0.0
            S_xy = 0.5 * F * (h ** 2 - y ** 2) / I

            # Displacements

            u_x  = 0.5 * F * x ** 2 * y / (E * I)
            u_x += NU * F * y ** 3 / (6.0 * E * I)
            u_x -= F * y ** 3 / (6.0 * G * I)
            u_x -= 0.5 * F * (L ** 2 / E - h ** 2 / G) * y / I

            u_y  = - 0.5 * NU * x * y ** 2
            u_y -= x ** 3 / 6.0 - 0.5 * L ** 2 * x
            u_y -= L ** 3 / 3.0
            u_y *= F / (E * I)

            # Traction vectors

            T_x = S_xx * n_x + S_xy * n_y
            T_y = S_xy * n_x + S_yy * n_y

            # Strain energy

            SE = 0.5 * (T_x * u_x + T_y * u_y) * R * dtheta * W.get(m)[a]

            # Strain energy density

            SED += SE / A

            # Perimeter

            p += R * dtheta * W.get(m)[a]

            print >> f, 'Gaussian coordinate t =', t, '\n'
            print >> f, 'theta =', theta, '\n'

            print >> f, 'n_x =', n_x, '\n'
            print >> f, 'n_y =', n_y, '\n'

            print >> f, 'x =', x, '\n'
            print >> f, 'y =', y, '\n'

            print >> f, 'S_xx =', S_xx, '\n'
            print >> f, 'S_yy =', S_yy, '\n'
            print >> f, 'S_xy =', S_xy, '\n'

            print >> f, 'u_x =', u_x, '\n'
            print >> f, 'u_y =', u_y, '\n'

            print >> f, 'T_x =', T_x, '\n'
            print >> f, 'T_y =', T_y, '\n'

            print >> f, 'Strain Energy =', SE, '\n'
            print >> f, 'Strain Energy Density =', SED, '\n'

            print >> f, 'Perimeter =', p, '\n'

            a += 1

        # Definition of the theoretical value for the SED

    REF1  = (6.0 * (x_c ** 2 + y_c ** 2) + R ** 2) * R ** 2
    REF1 += 24.0 * x_c ** 2 * y_c ** 2
    REF1 /= E

    REF2  = (3.0 * (3.0 * y_c ** 2 - h ** 2) + 0.75 * R ** 2) * R ** 2
    REF2 += 6.0 * (h ** 2 - y_c ** 2) ** 2
    REF2 /= G

    REF  = REF1 + REF2
    REF *= (F / I) ** 2 / 48.0

        # Printing of the final values

    print >> f, '\n==================================================\n'
    print >> f, '              RESULTS FOR CIRCLE', k + 1
    print >> f, '\n==================================================\n'

    print >> f, '  x_c =', x_c, ',     y_c =', y_c, '\n'

    print >> f, '         Computed SED =', SED, '\n'
    print >> f, '      Theoretical SED =', REF, '\n'
    print >> f, '     Percentual error =', abs(SED / REF - 1.0) * 100.0, '%\n'

    print >> f, '   Length of the path =', p

    print >> f, '\n==================================================\n\n\n'

f.close()
```

B.4 PLATE_CRACK_SED.PY

Algorithm B.4. Computation of $\mathcal{SED}$ for a cracked plate subjected to a constant tensile stress (subsection 4.3.1).

```
f = open('py_plate_crack_sed.dat','w')

import math
import random

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

print >> f, '=============================================================\n'
print >> f, '    SCRIPT FOR THE COMPUTATION OF THE LOCAL STRAIN ENERGY\n\
       DENSITY OF A CRACKED STEEL PLATE SUBJECTED TO A\n\
     CONSTANT TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n'
print >> f, '=============================================================\n'

print >> f, '\
                ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^    S0 = 100 MPa\n\
                | | | | | | | | | | |\n\
                _____________________\n\
               |                     |\n\
               |                     |\n\
               |                     |\n\
               |                     |\n\
               |        _____        |\n\
               |                     |\n\
               |                     |\n\
               |                     |\n\
       DX = 0 >|_____________________|\n\
                ^^^^^^^^^^^^^^^^^^^^^\n\
                       DY = 0\n\n'

# Input for the values R, n, m

print >> f, 'INPUT VALUES:\n\n'

R = input('Enter the radius of the circle onto which compute the SED: ')

print >> f, 'Radius of the circle: R =', R, '\n'

n = input('Enter the number of subdivisions: ')

print >> f, 'Number of subdivisions: n =', n, '\n'

m = input('Enter the number of Gaussian points for each subdivision: ')

print >> f, 'Number of Gaussian points for each subdivision: m =', m, '\n'
```

```

# Definition of some parameters of the problem

# Material

E  = 210000.0                # Young modulus of steel [MPa]
NU = 0.3                     # Poisson ratio of steel []
G  = 0.5 * E / (1.0 + NU)    # Shear modulus of steel [MPa]

# Geometry

h = 100.0       # Length of the plate’s edge [mm]
c = 10.0        # Half crack length [mm]

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
A = 0.5 * (theta_b - theta_a) * R ** 2
K_I = S0 * math.sqrt(math.pi * c)

# Definition of the point coordinates

x_c = 0.5 * h + c
y_c = 0.5 * h

# Definition of the initial values

SE = 0.0
SED = 0.0
p = 0.0

for i in range(1, n + 1):

    a = 0

    print >> f, ’---------------------------------------\n’
    print >> f, ’             SUBDIVISION’, i, ’\n’
    print >> f, ’---------------------------------------\n’

    # Definition of the angular quantities

    theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
    theta_2 = theta_a + i * (theta_b - theta_a) / n
    dtheta = 0.5 * (theta_2 - theta_1)

    print >> f, ’theta_1 =’, theta_1, ’\n’
    print >> f, ’theta_2 =’, theta_2, ’\n’
    print >> f, ’dtheta =’, dtheta, ’\n’

    for j in range(m):

        print >> f, ’-----------------------\n’
        print >> f, ’      ITERATION’, j + 1, ’\n’
        print >> f, ’-----------------------\n’

        # Calculation of the desired quantities

        # Theta angle

        t = T.get(m)[m - j - 1]
        theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2

        # Normals

        n_x = math.cos(theta)
        n_y = math.sin(theta)

        # Stresses

        S_xx  = math.cos(0.5 * theta) * (1.0 - math.sin(0.5 * theta) * \
math.sin(1.5 * theta))
        S_xx *= K_I / math.sqrt(2.0 * math.pi * R)

        S_yy  = math.cos(0.5 * theta) * (1.0 + math.sin(0.5 * theta) * \
math.sin(1.5 * theta))
        S_yy *= K_I / math.sqrt(2.0 * math.pi * R)

        S_xy  = math.sin(0.5 * theta) * math.cos(0.5 * theta) * \
math.cos(1.5 * theta)
        S_xy *= K_I / math.sqrt(2.0 * math.pi * R)

        # Displacements

        u_x  = math.cos(0.5 * theta) * (1.0 - 2.0 * NU + \
math.sin(0.5 * theta) ** 2)
        u_x *= K_I * math.sqrt(0.5 * R / math.pi) / G

        u_y  = math.sin(0.5 * theta) * (2.0 - 2.0 * NU - \
math.cos(0.5 * theta) ** 2)
        u_y *= K_I * math.sqrt(0.5 * R / math.pi) / G

        # Traction vectors

        T_x = S_xx * n_x + S_xy * n_y
        T_y = S_xy * n_x + S_yy * n_y

        # Strain energy

        SE = 0.5 * (T_x * u_x + T_y * u_y) * R * dtheta * W.get(m)[a]

        # Strain energy density

        SED += SE / A

        # Perimeter

        p += R * dtheta * W.get(m)[a]

        print >> f, 'Gaussian coordinate t =', t, '\n'
        print >> f, 'theta =', theta, '\n'

        print >> f, 'n_x =', n_x, '\n'
        print >> f, 'n_y =', n_y, '\n'

        print >> f, 'S_xx =', S_xx, '\n'
        print >> f, 'S_yy =', S_yy, '\n'
        print >> f, 'S_xy =', S_xy, '\n'

        print >> f, 'u_x =', u_x, '\n'
        print >> f, 'u_y =', u_y, '\n'

        print >> f, 'T_x =', T_x, '\n'
        print >> f, 'T_y =', T_y, '\n'

        print >> f, 'Strain Energy =', SE, '\n'
        print >> f, 'Strain Energy Density =', SED, '\n'

        print >> f, 'Perimeter =', p, '\n'

        a += 1

# Definition of the theoretical value for the SED

REF  = (1.0 + NU) * (5.0 - 8.0 * NU) * K_I ** 2
REF /= 8.0 * math.pi * R * E

# Printing of the final values

print >> f, '\n==================================================\n'
print >> f, '                      RESULTS'
print >> f, '\n==================================================\n'

print >> f, '          x_c =', x_c,',     y_c =', y_c, '\n'

print >> f, '         Computed SED =', SED, '\n'
print >> f, '      Theoretical SED =', REF, '\n'
print >> f, '     Percentual error =', abs(SED / REF - 1.0) * 100.0, '%\n'

print >> f, '   Length of the path =', p

print >> f, '\n==================================================\n\n\n'

f.close()
```

C COMMAND FILES

This appendix collects all the command files used in the Finite Element Analyses. As in the previous appendix, the Gauss-Legendre abscissas and weights were omitted.

C.1 PLATE_CNST_SED_1D.COMM

Algorithm C.1. Finite Element computation of $\mathcal{SED}$ through a contour integral for a plate subjected to a constant tensile stress (subsection 4.2.1).

```
# File PLATE_CNST_SED_1D.COMM
# Computes the local strain energy density in random
# points for a plate subjected to a constant tensile
# stress through a contour integral
# Utilizes the MACR_LIGN_COUPE command

DEBUT(PAR_LOT='NON');

import math
import random as rnd
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_cnst_sed_1d.dat')
f = open(exportfile,'w')

f.write('===========================================================\
=========\n')
f.write('===========================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
        DENSITY IN RANDOM POINTS FOR A PLATE SUBJECTED TO A\n\
        CONSTANT TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n')
f.write('===========================================================\
=========\n')
f.write('===========================================================\
=========\n\n')

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young’s modulus of steel [MPa]
NU = 0.3        # Poisson’s ratio of steel []

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Input for the values R, q, n, m

f.write(’INPUT VALUES:\n\n’)

R = input(’Enter the radius of the circles onto which compute the SED: ’)

f.write(’Radius of the circles: R = ’ + ’{0:2.2f}’.format(R) + ’\n\n’)

q = input(’Enter the number of random points: ’)

f.write(’Number of random points: q = ’ + str(q) + ’\n\n’)

n = input(’Enter the number of subdivisions for each circumference: ’)

f.write(’Number of subdivisions for each circumference: n = ’ + \
str(n) + ’\n\n’)

m = input(’Enter the number of Gaussian points for each subdivision: ’)

f.write(’Number of Gaussian points for each subdivision: m = ’ + \
str(m) + ’\n\n’)
f.write(’=======================================================\
=========\n\n’)

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT=’MED’,);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=(’Edge_1’,’Edge_2’,),),);

# Application of the plane strain conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT=’OUI’,
                         PHENOMENE=’MECANIQUE’,
                         MODELISATION=’D_PLAN’,),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA='Edge_1',
                                 DX=0.0,),
                              _F(GROUP_NO='Vertex_1',
                                 DY=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    PRES_REP=_F(GROUP_MA='Edge_2',
                                PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
b = 0

# Definition of the empty arrays

C_X = []
C_Y = []
STRESS = [None] * q * n * m
DISPL = [None] * q * n * m
n_x = [None] * q * n * m
n_y = [None] * q * n * m

for k in range(q):

    a = -1

    # Definition of the coordinates of the points

    x_c = rnd.uniform(R, 100.0 - R)
    y_c = rnd.uniform(R, 100.0 - R)
    x_0 = x_c + R * math.cos(theta_a)
    y_0 = y_c + R * math.sin(theta_a)

    # Appending the coordinates to the corresponding vectors

    C_X.append(x_c)
    C_Y.append(y_c)

    # Interpolation of the desired quantities onto the path

    for i in range(1, n + 1):

        theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
        theta_2 = theta_a + i * (theta_b - theta_a) / n
        dtheta = 0.5 * (theta_2 - theta_1)

        for j in range(m):

            t = T.get(m)[m - j - 1]
            theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2
            n_x[i + j + k + a + b] = math.cos(theta)
            n_y[i + j + k + a + b] = math.sin(theta)
            x_1 = x_c + R * math.cos(theta)
            y_1 = y_c + R * math.sin(theta)

            # Stresses

            STR=MACR_LIGN_COUPE(RESULTAT=RESU,
                                NOM_CHAM='SIGM_NOEU',
                                LIGN_COUPE=_F(INTITULE='STRESSES',
                                              TYPE='SEGMENT',
                                              NB_POINTS=2,
                                              COOR_ORIG=(x_0,y_0),
                                              COOR_EXTR=(x_1,y_1),),);

            # Displacements

            DIS=MACR_LIGN_COUPE(RESULTAT=RESU,
                                NOM_CHAM='DEPL',
                                LIGN_COUPE=_F(INTITULE='DISPLACEMENTS',
                                              TYPE='SEGMENT',
                                              NB_POINTS=2,
                                              COOR_ORIG=(x_0,y_0),
                                              COOR_EXTR=(x_1,y_1),),);

            # Definition of the tables from the concepts

            STRESS[i + j + k + a + b] = STR.EXTR_TABLE()
            DISPL[i + j + k + a + b] = DIS.EXTR_TABLE()

            # Destruction of the concepts

            DETRUIRE(CONCEPT=(_F(NOM=STR),
                             _F(NOM=DIS),),);

            x_0 = x_1
            y_0 = y_1

        a += m - 1

    b += n * m - 1

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Python script for SED calculation

# Definition of the initial values and constants

A = 0.5 * (theta_b - theta_a) * R ** 2
a = 0
b = 0

# Definition of the empty arrays

SED = []
per = []

for s in range(q):

    # Definition of the initial values for the given point

    SEth = 0.0
    SE = 0.0
    p = 0.0

    for i in range(n * m):

        # Definition of the arrays from the tables

        coor_x = STRESS[i + s + b].values()['COOR_X']
        coor_y = STRESS[i + s + b].values()['COOR_Y']
        S_xx = STRESS[i + s + b].values()['SIXX']
        S_yy = STRESS[i + s + b].values()['SIYY']
        S_xy = STRESS[i + s + b].values()['SIXY']
        u_x = DISPL[i + s + b].values()['DX']
        u_y = DISPL[i + s + b].values()['DY']

        k = len(S_xx) - 1
        l = len(u_x) - 1

        # Calculation of the theoretical quantities

        # Displacements

        u_xth = (1.0 - NU ** 2) * S0 * coor_x[k] / E
        u_yth = -NU * (1.0 + NU) * S0 * coor_y[k] / E

        # Strain energy

        SEth += 0.5 * S0 * n_x[i + s + b] * u_xth * R * dtheta * W.get(m)[a]

                # Calculation of the FE quantities

        # Traction vectors

        T_x = S_xx[k] * n_x[i + s + b] + S_xy[k] * n_y[i + s + b]
        T_y = S_xy[k] * n_x[i + s + b] + S_yy[k] * n_y[i + s + b]

        # Strain energy

        SE += 0.5 * (T_x * u_x[l] + T_y * u_y[l]) * R * dtheta * W.get(m)[a]

        # Perimeter

        p += R * dtheta * W.get(m)[a]

        f.write('\n==================================================\n')
        f.write('            Iteration ' + str(i + 1) + ' for circle ' + \
str(s + 1) + ':')
        f.write('\n==================================================\n\n')

        f.write('Coordinates:      x = ' + '{0:3.10f}'.format(coor_x[k]) + '\n')
        f.write('                  y = ' + '{0:3.10f}'.format(coor_y[k]) + \
'\n\n')

        f.write('Stresses:       Sxx = ' + '{0:3.2f}'.format(S_xx[k]) + '\n')
        f.write('                Syy = ' + '{0:3.2f}'.format(S_yy[k]) + '\n')
        f.write('                Sxy = ' + '{0:3.2f}'.format(S_xy[k]) + '\n\n')

        f.write('Displacements:   Ux = ' + '{0:2.10e}'.format(u_x[l]) + '\n')
        f.write('                 Uy = ' + '{0:2.10e}'.format(u_y[l]) + '\n\n')

        f.write('Normal vector:   nx = ' + '{0:1.10f}'.format(n_x[i + s + b]) + \
'\n')
        f.write('                 ny = ' + '{0:1.10f}'.format(n_y[i + s + b]) + \
'\n\n')

        f.write('Traction vector: Tx = ' + '{0:1.10f}'.format(T_x) + '\n')
        f.write('                 Ty = ' + '{0:1.10f}'.format(T_y) + '\n\n')

        f.write('Strain energy:   SE = ' + '{0:2.10e}'.format(SE) + '\n')
        f.write('SED:            SED = ' + '{0:2.10e}'.format(SE / A) + '\n')

        f.write('\n==================================================\n\n')

        if a == m - 1:
            a = 0
        else:
            a += 1

    # Appending the results to the corresponding vectors

    SED.append(SE/A)
    per.append(p)

    b += n * m - 1

# Printing of the final values

# Definition of the theoretical value for the SED

REF = 0.5 * (1.0 - NU ** 2) * S0 ** 2 / E

for i in range(q):

    f.write('\n==================================================\n')
    f.write('              RESULTS FOR CIRCLE ' + str(i + 1))
    f.write('\n==================================================\n\n')

    f.write(' Coordinates of the center: x_c = ' + '{0:3.10f}'.format(C_X[i]) + \
'\n')
    f.write('                            y_c = ' + '{0:3.10f}'.format(C_Y[i]) + \
'\n\n')

    f.write('         Computed SED = ' + '{0:2.10e}'.format(SED[i]) + '\n')
    f.write('      Theoretical SED = ' + '{0:2.10e}'.format(REF) + '\n')
    f.write('     Percentual error = ' + '{0:4.2e}'.format((abs(SED[i] / REF - \
1.0) * 100.0)) + '%\n\n')
    f.write('   Length of the path = ' + '{0:1.10f}'.format(per[i]) + '\n\n')

    f.write('==================================================\n')

f.close()

FIN();
```

C.2 PLATE_LNR_SED_1D.COMM

Algorithm C.2. Finite Element computation of $\mathcal{SED}$ through a contour integral for a plate subjected to a linear tensile stress (subsection 4.2.2).

```
# File PLATE_LNR_SED_1D.COMM
# Computes the local strain energy density in random
# points for a plate subjected to a linear tensile
# stress through a contour integral
# Utilizes the MACR_LIGN_COUPE command

DEBUT(PAR_LOT='NON');

import math
import random as rnd
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_lnr_sed_1d.dat')
f = open(exportfile,'w')

f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
        DENSITY IN RANDOM POINTS FOR A PLATE SUBJECTED TO A\n\
         LINEAR TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n')
f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n\n')

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young's modulus of steel [MPa]
NU = 0.3        # Poisson's ratio of steel []

# Geometry

h = 100.0       # Length of the plate's edge [mm]

```

```
# Boundary conditions

SM = 100.0      # Maximum applied tensile stress [MPa]

# Input for the values R, q, n, m

f.write('INPUT VALUES:\n\n')

R = input('Enter the radius of the circles onto which compute the SED: ')

f.write('Radius of the circles: R = ' + '{0:2.2f}'.format(R) + '\n\n')

q = input('Enter the number of random points: ')

f.write('Number of random points: q = ' + str(q) + '\n\n')

n = input('Enter the number of subdivisions for each circumference: ')

f.write('Number of subdivisions for each circumference: n = ' + \
str(n) + '\n\n')

m = input('Enter the number of Gaussian points for each subdivision: ')

f.write('Number of Gaussian points for each subdivision: m = ' + \
str(m) + '\n\n')
f.write('=========================================================\
=========\n\n')

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT='MED',);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=('Edge_1','Edge_2',),),);

# Application of the plane strain conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA='Edge_1',
                                 DX=0.0,),
                              _F(GROUP_NO='Vertex_1',
                                 DY=0.0,),),);

# Application of the external loads

```

```
SX=FORMULE(NOM_PARA='Y',VALE='(-SM * (1.0 - Y / h))');

LOAD=AFFE_CHAR_MECA_F(MODELE=MODE,
                      PRES_REP=_F(GROUP_MA='Edge_2',
                                  PRES=SX,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
b = 0

# Definition of the empty arrays

C_X = []
C_Y = []
STRESS = [None] * q * n * m
DISPL = [None] * q * n * m
n_x = [None] * q * n * m
n_y = [None] * q * n * m

for k in range(q):

    a = -1

    # Definition of the coordinates of the points

    x_c = rnd.uniform(R, 100.0 - R)
    y_c = rnd.uniform(R, 100.0 - R)
    x_0 = x_c + R * math.cos(theta_a)
    y_0 = y_c + R * math.sin(theta_a)

    # Appending the coordinates to the corresponding vectors

    C_X.append(x_c)
    C_Y.append(y_c)

    # Interpolation of the desired quantities onto the path

    for i in range(1, n + 1):

        theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
        theta_2 = theta_a + i * (theta_b - theta_a) / n
        dtheta = 0.5 * (theta_2 - theta_1)

        for j in range(m):

            t = T.get(m)[m - j - 1]
            theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2
            n_x[i + j + k + a + b] = math.cos(theta)
            n_y[i + j + k + a + b] = math.sin(theta)
            x_1 = x_c + R * math.cos(theta)
            y_1 = y_c + R * math.sin(theta)

            # Stresses

            STR=MACR_LIGN_COUPE(RESULTAT=RESU,
                                NOM_CHAM='SIGM_NOEU',
                                LIGN_COUPE=_F(INTITULE='STRESSES',
                                              TYPE='SEGMENT',
                                              NB_POINTS=2,
                                              COOR_ORIG=(x_0,y_0),
                                              COOR_EXTR=(x_1,y_1),),);

            # Displacements

            DIS=MACR_LIGN_COUPE(RESULTAT=RESU,
                                NOM_CHAM='DEPL',
                                LIGN_COUPE=_F(INTITULE='DISPLACEMENTS',
                                              TYPE='SEGMENT',
                                              NB_POINTS=2,
                                              COOR_ORIG=(x_0,y_0),
                                              COOR_EXTR=(x_1,y_1),),);

            # Definition of the tables from the concepts

            STRESS[i + j + k + a + b] = STR.EXTR_TABLE()
            DISPL[i + j + k + a + b] = DIS.EXTR_TABLE()

            # Destruction of the concepts

            DETRUIRE(CONCEPT=(_F(NOM=STR),
                             _F(NOM=DIS),),);

            x_0 = x_1
            y_0 = y_1

        a += m - 1

    b += n * m - 1

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Python script for SED calculation

# Definition of the initial values and constants

A = 0.5 * (theta_b - theta_a) * R ** 2
a = 0
b = 0

# Definition of the empty arrays

SED = []
per = []
```

```

for s in range(q):

    # Definition of the initial values for the given point

    SEth = 0.0
    SE = 0.0
    p = 0.0

    for i in range(n * m):

        # Definition of the arrays from the tables

        coor_x = STRESS[i + s + b].values()['COOR_X']
        coor_y = STRESS[i + s + b].values()['COOR_Y']
        S_xx = STRESS[i + s + b].values()['SIXX']
        S_yy = STRESS[i + s + b].values()['SIYY']
        S_xy = STRESS[i + s + b].values()['SIXY']
        u_x = DISPL[i + s + b].values()['DX']
        u_y = DISPL[i + s + b].values()['DY']

        k = len(S_xx) - 1
        l = len(u_x) - 1

        # Calculation of the theoretical quantities

        # Stresses

        S_xxth = SM * (1.0 - coor_y[k] / h)
        S_yyth = 0.0
        S_xyth = 0.0

        # Displacements

        u_xth  = (1.0 - NU ** 2) * SM * (1.0 - coor_y[k] / h) * coor_x[k] / E

        u_yth  = 0.5 * coor_x[k] ** 2 / h
        u_yth -= NU * (1.0 - 0.5 * coor_y[k] / h) * coor_y[k] / (1.0 - NU)
        u_yth *= (1.0 - NU ** 2) * SM / E

        # Traction vectors

        T_xth = S_xxth * n_x[i + s + b] + S_xyth * n_y[i + s + b]
        T_yth = S_xyth * n_x[i + s + b] + S_yyth * n_y[i + s + b]

        # Strain energy

        SEth += 0.5 * (T_xth * u_xth + T_yth * u_yth) * R * dtheta * W.get(m)[a]

                # Calculation of the FE quantities

        # Traction vectors

        T_x = S_xx[k] * n_x[i + s + b] + S_xy[k] * n_y[i + s + b]
        T_y = S_xy[k] * n_x[i + s + b] + S_yy[k] * n_y[i + s + b]

        # Strain energy

        SE += 0.5 * (T_x * u_x[l] + T_y * u_y[l]) * R * dtheta * W.get(m)[a]

        # Perimeter

        p += R * dtheta * W.get(m)[a]

        f.write('\n=================================================\n')
        f.write('            Iteration ' + str(i + 1) + ' for circle ' + \
str(s + 1) + ’:’)
        f.write(’\n==================================================\n\n’)

        f.write(’Coordinates:      x = ’ + ’{0:3.10f}’.format(coor_x[k]) + ’\n’)
        f.write(’                  y = ’ + ’{0:3.10f}’.format(coor_y[k]) + \
’\n\n’)

        f.write(’Stresses:       Sxx = ’ + ’{0:3.2f}’.format(S_xx[k]) + ’\n’)
        f.write(’                Syy = ’ + ’{0:3.2f}’.format(S_yy[k]) + ’\n’)
        f.write(’                Sxy = ’ + ’{0:3.2f}’.format(S_xy[k]) + ’\n\n’)

        f.write(’Displacements:   Ux = ’ + ’{0:2.10e}’.format(u_x[l]) + ’\n’)
        f.write(’                 Uy = ’ + ’{0:2.10e}’.format(u_y[l]) + ’\n\n’)

        f.write(’Normal vector:   nx = ’ + ’{0:1.10f}’.format(n_x[i + s + b]) + \
’\n’)
        f.write(’                 ny = ’ + ’{0:1.10f}’.format(n_y[i + s + b]) + \
’\n\n’)

        f.write(’Traction vector: Tx = ’ + ’{0:1.10f}’.format(T_x) + ’\n’)
        f.write(’                 Ty = ’ + ’{0:1.10f}’.format(T_y) + ’\n\n’)

        f.write(’Strain energy:   SE = ’ + ’{0:2.10e}’.format(SE) + ’\n’)
        f.write(’SED:            SED = ’ + ’{0:2.10e}’.format(SE / A) + ’\n’)

        f.write(’\n==================================================\n\n’)

        if a == m - 1:
            a = 0
        else:
            a += 1

    # Appending the results to the corresponding vectors

    SED.append(SE/A)
    per.append(p)

    b += n * m - 1

# Printing of the final values

for i in range(q):

    # Definition of the theoretical value for the SED

    REF  = 0.25 * R ** 2 + (h - C_Y[i]) ** 2
    REF *= 0.5 * (SM / h) ** 2 * (1.0 - NU ** 2) / E

    f.write(’\n==================================================\n’)
    f.write(’              RESULTS FOR CIRCLE ’ + str(i + 1))
    f.write(’\n==================================================\n\n’)

    f.write(’ Coordinates of the center: x_c = ’ + ’{0:3.10f}’.format(C_X[i]) + \
’\n’)
    f.write(’                            y_c = ’ + ’{0:3.10f}’.format(C_Y[i]) + \
’\n\n’)

    f.write(’         Computed SED = ’ + ’{0:2.10e}’.format(SED[i]) + ’\n’)
    f.write(’      Theoretical SED = ’ + ’{0:2.10e}’.format(REF) + ’\n’)
    f.write(’     Percentual error = ’ + ’{0:4.2e}’.format((abs(SED[i] / REF - \
1.0) * 100.0)) + ’%\n\n’)
    f.write(’   Length of the path = ’ + ’{0:1.10f}’.format(per[i]) + ’\n\n’)

    f.write(’==================================================\n’)

f.close()
```

```

FIN();
```

C.3 BEAM_END_SED_1D.COMM

Algorithm C.3. Finite Element computation of $\mathcal{SED}$ through a contour integral for a beam subjected to an end load (subsection 4.2.3).

```
# File BEAM_END_SED_1D.COMM
# Computes the local strain energy density in random
# points for a two-dimensional beam subjected to an
# end load through a contour integral
# Utilizes the MACR_LIGN_COUPE command

DEBUT(PAR_LOT='NON');

import math
import random as rnd
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_beam_end_sed_1d.dat')
f = open(exportfile,'w')

f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
        DENSITY IN RANDOM POINTS FOR A TWO-DIMENSIONAL BEAM\n\
        SUBJECTED TO AN END LOAD THROUGH A CONTOUR INTEGRAL\n')
f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n\n')

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

# Definition of some parameters of the problem

# Material

E  = 210000.0               # Young's modulus of steel [MPa]
NU = 0.3                    # Poisson's ratio of steel []
```

```
G  = 0.5 * E / (1.0 + NU)   # Shear modulus of steel [MPa]

# Geometry

b = 1.0                     # Thickness of the beam [mm]
L = 100.0                   # Length of the beam [mm]
h = 10.0                    # Height of half beam [mm]
I = 2.0 * b * h ** 3 / 3.0  # Moment of inertia [mm ^ 4]

# Boundary conditions

F = 100.0                   # Applied end load [N]

# Input for the values R, q, n, m

f.write('INPUT VALUES:\n\n')

R = input('Enter the radius of the circles onto which compute the SED: ')

f.write('Radius of the circles: R = ' + '{0:2.2f}'.format(R) + '\n\n')

q = input('Enter the number of random points: ')

f.write('Number of random points: q = ' + str(q) + '\n\n')

n = input('Enter the number of subdivisions for each circumference: ')

f.write('Number of subdivisions for each circumference: n = ' + \
str(n) + '\n\n')

m = input('Enter the number of Gaussian points for each subdivision: ')

f.write('Number of Gaussian points for each subdivision: m = ' + \
str(m) + '\n\n')
f.write('=========================================================\
=========\n\n')

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT='MED',);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=('Edge_1','Edge_2',),),);

# Application of the plane stress conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='C_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints
```

```

CONST=AFFE_CHAR_MECA(MODELE=MODE,
                     DDL_IMPO=(_F(GROUP_MA='Edge_1',
                                  DX=0.0,
                                  DY=0.0,),),);

# Application of the external loads

FY = -0.5 * F / (b * h)

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    FORCE_CONTOUR=_F(GROUP_MA='Edge_2',
                                     FY=FY,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=CONST,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Definition of the initial values and constants

theta_a = -math.pi
theta_b = math.pi
b = 0

# Definition of the empty arrays

C_X = []
C_Y = []
STRESS = [None] * q * n * m
DISPL = [None] * q * n * m
n_x = [None] * q * n * m
n_y = [None] * q * n * m

for k in range(q):

    a = -1

    # Definition of the coordinates of the points

    x_c = rnd.uniform(0.15 * L + R, 0.9 * L - R)
    y_c = rnd.uniform(-h + R, h - R)
    x_0 = x_c + R * math.cos(theta_a)
    y_0 = y_c + R * math.sin(theta_a)

    # Appending the coordinates to the corresponding vectors

    C_X.append(x_c)
    C_Y.append(y_c)

    # Interpolation of the desired quantities onto the path

    for i in range(1, n + 1):

        theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
        theta_2 = theta_a + i * (theta_b - theta_a) / n
        dtheta = 0.5 * (theta_2 - theta_1)

        for j in range(m):

            t = T.get(m)[m - j - 1]
            theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2
            n_x[i + j + k + a + b] = math.cos(theta)
            n_y[i + j + k + a + b] = math.sin(theta)
            x_1 = x_c + R * math.cos(theta)
            y_1 = y_c + R * math.sin(theta)

            # Stresses

            STR=MACR_LIGN_COUPE(RESULTAT=RESU,
                                NOM_CHAM='SIGM_NOEU',
                                LIGN_COUPE=_F(INTITULE='STRESSES',
                                              TYPE='SEGMENT',
                                              NB_POINTS=2,
                                              COOR_ORIG=(x_0,y_0),
                                              COOR_EXTR=(x_1,y_1),),);

            # Displacements

            DIS=MACR_LIGN_COUPE(RESULTAT=RESU,
                                NOM_CHAM='DEPL',
                                LIGN_COUPE=_F(INTITULE='DISPLACEMENTS',
                                              TYPE='SEGMENT',
                                              NB_POINTS=2,
                                              COOR_ORIG=(x_0,y_0),
                                              COOR_EXTR=(x_1,y_1),),);

            # Definition of the tables from the concepts

            STRESS[i + j + k + a + b] = STR.EXTR_TABLE()
            DISPL[i + j + k + a + b] = DIS.EXTR_TABLE()

            # Destruction of the concepts

            DETRUIRE(CONCEPT=(_F(NOM=STR),
                              _F(NOM=DIS),),);

            x_0 = x_1
            y_0 = y_1

        a += m - 1

    b += n * m - 1

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Python script for SED calculation

# Definition of the initial values and constants

```

```
A = 0.5 * (theta_b - theta_a) * R ** 2
a = 0
b = 0

# Definition of the empty arrays

SED = []
per = []

for s in range(q):

    # Definition of the initial values for the given point

    SEth = 0.0
    SE = 0.0
    p = 0.0

    for i in range(n * m):

        # Definition of the arrays from the tables

        coor_x = STRESS[i + s + b].values()['COOR_X']
        coor_y = STRESS[i + s + b].values()['COOR_Y']
        S_xx = STRESS[i + s + b].values()['SIXX']
        S_yy = STRESS[i + s + b].values()['SIYY']
        S_xy = STRESS[i + s + b].values()['SIXY']
        u_x = DISPL[i + s + b].values()['DX']
        u_y = DISPL[i + s + b].values()['DY']

        k = len(S_xx) - 1
        l = len(u_x) - 1

        # Calculation of the theoretical quantities

        # Stresses

        S_xxth = F * coor_x[k] * coor_y[k] / I
        S_yyth = 0.0
        S_xyth = 0.5 * F * (h ** 2 - coor_y[k] ** 2) / I

        # Displacements

        u_xth  = 0.5 * F * coor_x[k] ** 2 * coor_y[k] / (E * I)
        u_xth += NU * F * coor_y[k] ** 3 / (6.0 * E * I)
        u_xth -= F * coor_y[k] ** 3 / (6.0 * G * I)
        u_xth -= 0.5 * F * (L ** 2 / E - h ** 2 / G) * coor_y[k] / I

        u_yth  = -0.5 * NU * coor_x[k] * coor_y[k] ** 2
        u_yth -= coor_x[k] ** 3 / 6.0 - 0.5 * L ** 2 * coor_x[k]
        u_yth -= L ** 3 / 3.0
        u_yth *= F / (E * I)

        # Traction vectors

        T_xth = S_xxth * n_x[i + s + b] + S_xyth * n_y[i + s + b]
        T_yth = S_xyth * n_x[i + s + b] + S_yyth * n_y[i + s + b]

        # Strain energy

        SEth += 0.5 * (T_xth * u_xth + T_yth * u_yth) * R * dtheta * W.get(m)[a]

        # Calculation of the FE quantities

        # Traction vectors

        T_x = S_xx[k] * n_x[i + s + b] + S_xy[k] * n_y[i + s + b]
        T_y = S_xy[k] * n_x[i + s + b] + S_yy[k] * n_y[i + s + b]

        # Strain energy

        SE += 0.5 * (T_x * u_x[l] + T_y * u_y[l]) * R * dtheta * W.get(m)[a]

        # Perimeter

        p += R * dtheta * W.get(m)[a]

        f.write('\n==================================================\n')
        f.write('            Iteration ' + str(i + 1) + ' for circle ' + \
str(s + 1) + ':')
        f.write('\n==================================================\n\n')

        f.write('Coordinates:      x = ' + '{0:3.10f}'.format(coor_x[k]) + '\n')
        f.write('                  y = ' + '{0:3.10f}'.format(coor_y[k]) + \
'\n\n')

        f.write('Stresses:       Sxx = ' + '{0:3.2f}'.format(S_xx[k]) + '\n')
        f.write('                Syy = ' + '{0:3.2f}'.format(S_yy[k]) + '\n')
        f.write('                Sxy = ' + '{0:3.2f}'.format(S_xy[k]) + '\n\n')

        f.write('Displacements:   Ux = ' + '{0:2.10e}'.format(u_x[l]) + '\n')
        f.write('                 Uy = ' + '{0:2.10e}'.format(u_y[l]) + '\n\n')

        f.write('Normal vector:   nx = ' + '{0:1.10f}'.format(n_x[i + s + b]) + \
'\n')
        f.write('                 ny = ' + '{0:1.10f}'.format(n_y[i + s + b]) + \
'\n\n')

        f.write('Traction vector: Tx = ' + '{0:1.10f}'.format(T_x) + '\n')
        f.write('                 Ty = ' + '{0:1.10f}'.format(T_y) + '\n\n')

        f.write('Strain energy:   SE = ' + '{0:2.10e}'.format(SE) + '\n')
        f.write('SED:            SED = ' + '{0:2.10e}'.format(SE / A) + '\n')

        f.write('\n==================================================\n\n')

        if a == m - 1:
            a = 0
        else:
            a += 1

    # Appending the results to the corresponding vectors

    SED.append(SE/A)
    per.append(p)

    b += n * m - 1

# Printing of the final values

for i in range(q):

    # Definition of the theoretical value for the SED

    REF1  = (6.0 * (C_X[i] ** 2 + C_Y[i] ** 2) + R ** 2) * R ** 2
    REF1 += 24.0 * C_X[i] ** 2 * C_Y[i] ** 2
    REF1 /= E

    REF2  = 3.0 * (3.0 * C_Y[i] ** 2 - h ** 2) + R ** 2
    REF2 *= R ** 2
    REF2 += 6.0 * (h ** 2 - C_Y[i] ** 2) ** 2
    REF2 /= G

    REF  = REF1 + REF2
    REF *= (F / I) ** 2
    REF /= 48.0

    f.write('\n================================================\n')
    f.write('              RESULTS FOR CIRCLE ' + str(i + 1))
    f.write('\n================================================\n\n')

    f.write(' Coordinates of the center: x_c = ' + '{0:3.10f}'.format(C_X[i]) + \
'\n')
    f.write('                            y_c = ' + '{0:3.10f}'.format(C_Y[i]) + \
'\n\n')

    f.write('         Computed SED = ' + '{0:2.10e}'.format(SED[i]) + '\n')
    f.write('      Theoretical SED = ' + '{0:2.10e}'.format(REF) + '\n')
    f.write('     Percentual error = ' + '{0:4.2e}'.format((abs(SED[i] / REF - \
1.0) * 100.0)) + '%\n\n')
    f.write('   Length of the path = ' + '{0:1.10f}'.format(per[i]) + '\n\n')

    f.write('================================================\n')

f.close()

FIN();
```

C.4 PLATE_CRACK_SED_1D.COMM

Algorithm C.4. Finite Element computation of $\mathcal{SED}$ through a contour integral for a cracked plate.

```
# File PLATE_CRACK_SED_1D.COMM
# Computes the local strain energy density
# for a cracked plate subjected to a constant
# tensile stress through a contour integral
# Utilizes the MACR_LIGN_COUPE command

DEBUT(PAR_LOT='NON');

import math
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_crack_sed_1d.dat')
f = open(exportfile,'w')

f.write('=======================================================\
=========\n')
f.write('=======================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
        DENSITY FOR A CRACKED PLATE SUBJECTED TO A CONSTANT\n\
             TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n')
f.write('=======================================================\
=========\n')
f.write('=======================================================\
=========\n\n')

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young’s modulus of steel [MPa]
NU = 0.3        # Poisson’s ratio of steel []

# Geometry

c = 10.0        # Half crack length [mm]

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Input for the values R, n, m

f.write(’INPUT VALUES:\n\n’)

R = input(’Enter the radius of the circle onto which compute the SED: ’)

f.write(’Radius of the circles: R = ’ + ’{0:2.2f}’.format(R) + ’\n\n’)

n = input(’Enter the number of subdivisions for each circumference: ’)

f.write(’Number of subdivisions for each circumference: n = ’ + \
str(n) + ’\n\n’)

m = input(’Enter the number of Gaussian points for each subdivision: ’)

f.write(’Number of Gaussian points for each subdivision: m = ’ + \
str(m) + ’\n\n’)
f.write(’=========================================================\
=========\n\n’)

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT=’MED’,);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=(’Edge_1’,’Edge_2’,’Edge_3’,),),);

# Application of the plane strain conditions

```

```
MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA='Edge_1',
                                 DX=0.0,),
                              _F(GROUP_MA='Edge_2',
                                 DY=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    PRES_REP=_F(GROUP_MA='Edge_3',
                                PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Definition of the initial values and constants

theta_a = 0.0
theta_b = math.pi
a = -1

# Definition of the empty arrays

STRESS = [None] * n * m
DISPL = [None] * n * m
n_x = [None] * n * m
n_y = [None] * n * m

# Definition of the coordinates of the points

x_c = c
y_c = 0.0
x_0 = x_c + R * math.cos(theta_a)
```

```
y_0 = y_c + R * math.sin(theta_a)

# Interpolation of the desired quantities onto the path

for i in range(1, n + 1):

    theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
    theta_2 = theta_a + i * (theta_b - theta_a) / n
    dtheta = 0.5 * (theta_2 - theta_1)

    for j in range(m):

        t = T.get(m)[m - j - 1]
        theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2
        n_x[i + j + a] = math.cos(theta)
        n_y[i + j + a] = math.sin(theta)
        x_1 = x_c + R * math.cos(theta)
        y_1 = y_c + R * math.sin(theta)

        # Stresses

        STR=MACR_LIGN_COUPE(RESULTAT=RESU,
                            NOM_CHAM='SIGM_NOEU',
                            LIGN_COUPE=_F(INTITULE='STRESSES',
                                          TYPE='SEGMENT',
                                          NB_POINTS=2,
                                          COOR_ORIG=(x_0,y_0),
                                          COOR_EXTR=(x_1,y_1),),);
        # Displacements

        DIS=MACR_LIGN_COUPE(RESULTAT=RESU,
                            NOM_CHAM='DEPL',
                            LIGN_COUPE=_F(INTITULE='DISPLACEMENTS',
                                          TYPE='SEGMENT',
                                          NB_POINTS=2,
                                          COOR_ORIG=(x_0,y_0),
                                          COOR_EXTR=(x_1,y_1),),);

        # Definition of the tables from the concepts

        STRESS[i + j + a] = STR.EXTR_TABLE()
        DISPL[i + j + a] = DIS.EXTR_TABLE()

        # Destruction of the concepts

        DETRUIRE(CONCEPT=(_F(NOM=STR),
                          _F(NOM=DIS),),);

        x_0 = x_1
        y_0 = y_1

    a += m - 1

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Python script for SED calculation

# Definition of the initial values and constants

K_I = S0 * math.sqrt(math.pi * c)
A = 0.5 * (theta_b - theta_a) * R ** 2
a = 0
```

```

# Definition of the initial values for the given point

SED = 0.0
SE = 0.0
p = 0.0

for i in range(n * m):

    # Definition of the arrays from the tables

    coor_x = STRESS[i].values()[’COOR_X’]
    coor_y = STRESS[i].values()[’COOR_Y’]
    S_xx = STRESS[i].values()[’SIXX’]
    S_yy = STRESS[i].values()[’SIYY’]
    S_xy = STRESS[i].values()[’SIXY’]
    u_x = DISPL[i].values()[’DX’]
    u_y = DISPL[i].values()[’DY’]

    k = len(S_xx) - 1
    l = len(u_x) - 1

    # Calculation of the FE quantities

    # Traction vectors

    T_x = S_xx[k] * n_x[i] + S_xy[k] * n_y[i]
    T_y = S_xy[k] * n_x[i] + S_yy[k] * n_y[i]

    # Strain energy

    SE += 0.5 * (T_x * u_x[l] + T_y * u_y[l]) * R * dtheta * W.get(m)[a]

    # Perimeter

    p += R * dtheta * W.get(m)[a]

    f.write(’\n==================================================\n’)
    f.write(’               Iteration ’ + str(i + 1) + ’:’)
    f.write(’\n==================================================\n\n’)

    f.write(’Coordinates:      x = ’ + ’{0:3.10f}’.format(coor_x[k]) + ’\n’)
    f.write(’                  y = ’ + ’{0:3.10f}’.format(coor_y[k]) + ’\n\n’)

    f.write(’Stresses:       Sxx = ’ + ’{0:3.2f}’.format(S_xx[k]) + ’\n’)
    f.write(’                Syy = ’ + ’{0:3.2f}’.format(S_yy[k]) + ’\n’)
    f.write(’                Sxy = ’ + ’{0:3.2f}’.format(S_xy[k]) + ’\n\n’)

    f.write(’Displacements:   Ux = ’ + ’{0:2.10e}’.format(u_x[l]) + ’\n’)
    f.write(’                 Uy = ’ + ’{0:2.10e}’.format(u_y[l]) + ’\n\n’)

    f.write(’Normal vector:   nx = ’ + ’{0:1.10f}’.format(n_x[i]) + ’\n’)
    f.write(’                 ny = ’ + ’{0:1.10f}’.format(n_y[i]) + ’\n\n’)

    f.write(’Traction vector: Tx = ’ + ’{0:1.10f}’.format(T_x) + ’\n’)
    f.write(’                 Ty = ’ + ’{0:1.10f}’.format(T_y) + ’\n\n’)

    f.write(’Strain energy:   SE = ’ + ’{0:2.10e}’.format(SE) + ’\n’)
    f.write(’SED:            SED = ’ + ’{0:2.10e}’.format(SE / A) + ’\n’)

    f.write(’\n==================================================\n\n’)

    if a == m - 1:
        a = 0
    else:
        a += 1
```

```

SED = SE / A

# Printing of the final values

# Definition of the asymptotic value for the SED

REF  = (1.0 + NU) * (5.0 - 8.0 * NU) * K_I ** 2
REF /= 8.0 * math.pi * R * E

f.write('\n==================================================\n')
f.write('                    RESULTS')
f.write('\n==================================================\n\n')

f.write(' Coordinates of the center: x_c = ' + '{0:3.10f}'.format(x_c) + '\n')
f.write('                            y_c = ' + '{0:3.10f}'.format(y_c) + '\n\n')

f.write('         Computed SED = ' + '{0:2.10e}'.format(SED) + '\n')
f.write('      Theoretical SED = ' + '{0:2.10e}'.format(REF) + '\n')
f.write('     Percentual error = ' + '{0:4.2e}'.format((abs(SED / REF - \
1.0) * 100.0)) + '%\n\n')
f.write('   Length of the path = ' + '{0:1.10f}'.format(p) + '\n\n')

f.write('==================================================\n')

f.close()

FIN();
```

C.5 PLATE_NOTCH_SED_1D.COMM

Algorithm C.5. Finite Element computation of $\mathcal{SED}$ through a contour integral for a notched plate.

```
# File PLATE_NOTCH_SED_1D.COMM
# Computes the local strain energy density for
# a 135°-notched plate subjected to a constant
# tensile stress through a contour integral
# Utilizes the MACR_LIGN_COUPE command

DEBUT(PAR_LOT='NON');

import math
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_notch_sed_1d.dat')
f = open(exportfile,'w')

f.write('==========================================================\
=========\n')
f.write('==========================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
          DENSITY FOR A 135°-NOTCHED PLATE SUBJECTED TO A\n\
        CONSTANT TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n')
f.write('==========================================================\
=========\n')
f.write('==========================================================\
=========\n\n')
```

```

# Definition of the Gauss-Legendre abscissas

T = {
  1:[-0.0],
  2:[...],
  ⋮
}

# Definition of the Gauss-Legendre weights

W = {
  1:[2.0],
  2:[...],
  ⋮
}

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young’s modulus of steel [MPa]
NU = 0.3        # Poisson’s ratio of steel []

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Input for the values R, n, m

f.write(’INPUT VALUES:\n\n’)

R = input(’Enter the radius of the circle onto which compute the SED: ’)

f.write(’Radius of the circles: R = ’ + ’{0:2.2f}’.format(R) + ’\n\n’)

n = input(’Enter the number of subdivisions for each circumference: ’)

f.write(’Number of subdivisions for each circumference: n = ’ + \
str(n) + ’\n\n’)

m = input(’Enter the number of Gaussian points for each subdivision: ’)

f.write(’Number of Gaussian points for each subdivision: m = ’ + \
str(m) + ’\n\n’)
f.write(’==========================================================\
=========\n\n’)

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT=’MED’,);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=(’Edge_1’,’Edge_2’,),),);

# Application of the plane strain conditions

```

```
MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA='Edge_1',
                                 DY=0.0,),
                              _F(GROUP_NO='Vertex_1',
                                 DX=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    PRES_REP=_F(GROUP_MA='Edge_2',
                                PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Definition of the initial values and constants

theta_a = 0.0
theta_b = 5.0 * math.pi / 8.0
a = -1

# Definition of the empty arrays

STRESS = [None] * n * m
DISPL = [None] * n * m
n_x = [None] * n * m
n_y = [None] * n * m

# Definition of the coordinates of the points

x_c = 0.0
y_c = 0.0
x_0 = x_c + R * math.cos(theta_a)
```

```
y_0 = y_c + R * math.sin(theta_a)

# Interpolation of the desired quantities onto the path

for i in range(1, n + 1):

    theta_1 = theta_a + (i - 1) * (theta_b - theta_a) / n
    theta_2 = theta_a + i * (theta_b - theta_a) / n
    dtheta = 0.5 * (theta_2 - theta_1)

    for j in range(m):

        t = T.get(m)[m - j - 1]
        theta = 0.5 * (1.0 - t) * theta_1 + 0.5 * (1.0 + t) * theta_2
        n_x[i + j + a] = math.cos(theta)
        n_y[i + j + a] = math.sin(theta)
        x_1 = x_c + R * math.cos(theta)
        y_1 = y_c + R * math.sin(theta)

        # Stresses

        STR=MACR_LIGN_COUPE(RESULTAT=RESU,
                            NOM_CHAM='SIGM_NOEU',
                            LIGN_COUPE=_F(INTITULE='STRESSES',
                                          TYPE='SEGMENT',
                                          NB_POINTS=2,
                                          COOR_ORIG=(x_0,y_0),
                                          COOR_EXTR=(x_1,y_1),),);
        # Displacements

        DIS=MACR_LIGN_COUPE(RESULTAT=RESU,
                            NOM_CHAM='DEPL',
                            LIGN_COUPE=_F(INTITULE='DISPLACEMENTS',
                                          TYPE='SEGMENT',
                                          NB_POINTS=2,
                                          COOR_ORIG=(x_0,y_0),
                                          COOR_EXTR=(x_1,y_1),),);

        # Definition of the tables from the concepts

        STRESS[i + j + a] = STR.EXTR_TABLE()
        DISPL[i + j + a] = DIS.EXTR_TABLE()

        # Destruction of the concepts

        DETRUIRE(CONCEPT=(_F(NOM=STR),
                         _F(NOM=DIS),),);

        x_0 = x_1
        y_0 = y_1

    a += m - 1

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Python script for SED calculation

# Definition of the initial values and constants

A = 5.0 * math.pi * R ** 2 / 16.0
a = 0

```

```python
# Definition of the initial values for the given point

SED = 0.0
SE = 0.0
p = 0.0

for i in range(n * m):

    # Definition of the arrays from the tables

    coor_x = STRESS[i].values()['COOR_X']
    coor_y = STRESS[i].values()['COOR_Y']
    S_xx = STRESS[i].values()['SIXX']
    S_yy = STRESS[i].values()['SIYY']
    S_xy = STRESS[i].values()['SIXY']
    u_x = DISPL[i].values()['DX']
    u_y = DISPL[i].values()['DY']

    k = len(S_xx) - 1
    l = len(u_x) - 1

    # Calculation of the FE quantities

    # Traction vectors

    T_x = S_xx[k] * n_x[i] + S_xy[k] * n_y[i]
    T_y = S_xy[k] * n_x[i] + S_yy[k] * n_y[i]

    # Strain energy

    SE += 0.5 * (T_x * u_x[l] + T_y * u_y[l]) * R * dtheta * W.get(m)[a]

    # Perimeter

    p += R * dtheta * W.get(m)[a]

    f.write('\n==================================================\n')
    f.write('               Iteration ' + str(i + 1) + ':')
    f.write('\n==================================================\n\n')

    f.write('Coordinates:      x = ' + '{0:3.10f}'.format(coor_x[k]) + '\n')
    f.write('                  y = ' + '{0:3.10f}'.format(coor_y[k]) + '\n\n')

    f.write('Stresses:       Sxx = ' + '{0:3.2f}'.format(S_xx[k]) + '\n')
    f.write('                Syy = ' + '{0:3.2f}'.format(S_yy[k]) + '\n')
    f.write('                Sxy = ' + '{0:3.2f}'.format(S_xy[k]) + '\n\n')

    f.write('Displacements:   Ux = ' + '{0:2.10e}'.format(u_x[l]) + '\n')
    f.write('                 Uy = ' + '{0:2.10e}'.format(u_y[l]) + '\n\n')

    f.write('Normal vector:   nx = ' + '{0:1.10f}'.format(n_x[i]) + '\n')
    f.write('                 ny = ' + '{0:1.10f}'.format(n_y[i]) + '\n\n')

    f.write('Traction vector: Tx = ' + '{0:1.10f}'.format(T_x) + '\n')
    f.write('                 Ty = ' + '{0:1.10f}'.format(T_y) + '\n\n')

    f.write('Strain energy:   SE = ' + '{0:2.10e}'.format(SE) + '\n')
    f.write('SED:            SED = ' + '{0:2.10e}'.format(SE / A) + '\n')

    f.write('\n==================================================\n\n')

    if a == m - 1:
        a = 0
    else:
        a += 1

```

```
SED = SE / A

# Printing of the final values

f.write('\n==================================================\n')
f.write('                   RESULTS')
f.write('\n==================================================\n\n')

f.write(' Coordinates of the center: x_c = ' + '{0:3.10f}'.format(x_c) + '\n')
f.write('                            y_c = ' + '{0:3.10f}'.format(y_c) + '\n\n')

f.write('         Computed SED = ' + '{0:2.10e}'.format(SED) + '\n\n')
f.write('   Length of the path = ' + '{0:1.10f}'.format(p) + '\n\n')

f.write('==================================================\n')

f.close()

FIN();
```

C.6 PLATE_NOTCH_NSIF.COMM

Algorithm C.6. Finite Element computation of the mode I-NSIF for a notched plate.

```
# File PLATE_NOTCH_NSIF.COMM
# Computes the Notch Stress Intensity Factor
# of mode I for a 135°-notched plate
# subjected to a constant tensile stress
# Utilizes the POST_RELEVE_T command

DEBUT(PAR_LOT='NON');

import math
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_notch_nsif.dat')
f = open(exportfile,'w')

f.write('=========================================================\
=========\n\n')
f.write('=========================================================\
=========\n')
f.write('      FINITE ELEMENT COMPUTATION OF THE NOTCH STRESS \
INTENSITY\n\
          FACTOR FOR A 135°-NOTCHED PLATE SUBJECTED TO A\n\
        CONSTANT TENSILE STRESS THROUGH A CONTOUR INTEGRAL\n')
f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n\n')

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young's modulus of steel [MPa]
NU = 0.3        # Poisson's ratio of steel []

# Boundary conditions
```

```

S0 = 100.0      # Applied tensile stress [MPa]

# Definition of the notch tip coordinates

x_c = 0.0
y_c = 0.0

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT='MED',);

# Creation of the group of nodes

MAIL=DEFI_GROUP(reuse =MAIL,
                MAILLAGE=MAIL,
                CREA_GROUP_NO=_F(GROUP_MA='Edge_1',
                                 NOM='Bisector',),);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=('Edge_1','Edge_2','Edge_3',),),);

# Application of the plane strain conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA=('Edge_1','Edge_2',),
                                 DY=0.0,),
                              _F(GROUP_NO='Vertex_2',
                                 DX=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    PRES_REP=_F(GROUP_MA='Edge_3',
                                PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Extrapolation of the stresses along the bisector

STR=POST_RELEVE_T(ACTION=_F(OPERATION='EXTRACTION',
                            INTITULE='STRESSES',
                            RESULTAT=RESU,
                            NOM_CHAM='SIGM_NOEU',
                            GROUP_NO='Bisector',
                            TOUT_CMP='OUI',),);

# Definition of the table

STRESS = STR.EXTR_TABLE()

# Printing of the tables

IMPR_TABLE(TABLE=STR,);

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);


# Python script for the NSIF calculation

# Definition of the initial values and constants
lambda_1 = 0.6736

# Definition of the arrays from the tables

coor_x = STRESS.values()['COOR_X']
S_yy = STRESS.values()['SIYY']

k = len(S_yy) - 1

f.write('\n=================================================\n')
f.write('            Extrapolation of K_1')
f.write('\n=================================================\n\n')
f.write('         x              S_yy             K_1\n')

for i in range(k):

        K_1 = math.sqrt(2.0 * math.pi) * S_yy[i] * coor_x[i] ** (1.0 - lambda_1)

        f.write('       ' + '{0:1.3f}'.format(coor_x[i]) +  '           ' + \
'{0:3.2f}'.format(S_yy[i]) + '          ' + '{0:3.2f}'.format(K_1) + '\n')

f.close()

FIN();
```

C.7 PLATE_CRACK_SED_2D.COMM

Algorithm C.7. Finite Element computation of $\mathcal{SED}$ through a double integral for a cracked plate.

```
# File PLATE_CRACK_SED_2D.COMM
# Computes the local strain energy density
# for a cracked plate subjected to a constant
# tensile stress through a double integral
# Utilizes the DEFI_GROUP and POST_ELEM commands

DEBUT(PAR_LOT='NON');

import math
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_crack_sed_2d.dat')
f = open(exportfile,'w')

f.write('============================================================\
=========\n')
f.write('============================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
        DENSITY FOR A CRACKED PLATE SUBJECTED TO A CONSTANT\n\
             TENSILE STRESS THROUGH A DOUBLE INTEGRAL\n')
f.write('============================================================\
=========\n')
f.write('============================================================\
=========\n\n')

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young's modulus of steel [MPa]
NU = 0.3        # Poisson's ratio of steel []

# Geometry

c = 10.0        # Half crack length [mm]

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Definition of the crack tip coordinates

x_c = c
y_c = 0.0

# Input for the value R

f.write('INPUT VALUES:\n\n')

R = input('Enter the radius of the circle onto which compute the SED: ')

f.write('Radius of the circle: R = ' + '{0:2.2f}'.format(R) + '\n\n')

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT='MED',);

# Creation of the group of elements

MAIL=DEFI_GROUP(reuse =MAIL,
                MAILLAGE=MAIL,
                CREA_GROUP_MA=(_F(NOM='Circle',
                                  TYPE_MAILLE='2D',
                                  OPTION='SPHERE',
                                  POINT=(x_c,y_c),
                                  RAYON=R),),);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=('Edge_1','Edge_2',
                                             'Edge_3','Edge_4',),),);

# Application of the plane strain conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA='Edge_1',
                                 DX=0.0,),
                              _F(GROUP_MA=('Edge_2','Edge_3',),
                                 DY=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    PRES_REP=_F(GROUP_MA='Edge_4',
                                PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Calculation of the strain energy density

SED_CA=POST_ELEM(INTEGRALE=_F(GROUP_MA='Circle',
                   NOM_CHAM='ENEL_ELNO',
                   NOM_CMP='TOTALE',),
                   RESULTAT=RESU,);

# Printing of the table

IMPR_TABLE(TABLE=SED_CA,);

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Printing of the final values

# Definition of the asymptotic value for the SED

K_I = S0 * math.sqrt(math.pi * c)

REF  = (1.0 + NU) * (5.0 - 8.0 * NU) * K_I ** 2
REF /= 8.0 * math.pi * R * E

# Extraction of the values from the table

SED_TAB = SED_CA.EXTR_TABLE()

SED = SED_TAB.values()['MOYE_TOTALE']

f.write('\n==================================================\n')
f.write('                   RESULTS')
f.write('\n==================================================\n\n')

f.write(' Coordinates of the crack tip: x_c = ' + '{0:3.10f}'.format(x_c) + \
'\n')
f.write('                               y_c = ' + '{0:3.10f}'.format(y_c) + \
'\n\n')

f.write('         Computed SED = ' + '{0:2.5e}'.format(SED[0]) + '\n')
f.write('      Theoretical SED = ' + '{0:2.10e}'.format(REF) + '\n')
f.write('     Percentual error = ' + '{0:4.2e}'.format((abs(SED[0] / REF - \
1.0) * 100.0)) + '%\n\n')

f.write('==================================================\n')

f.close()

FIN();
```

C.8 PLATE_XCRACK_SED_2D.COMM

Algorithm C.8. Finite Element computation of $\mathcal{SED}$ through a double integral for a cracked plate with XFEM.

```
# File PLATE_XCRACK_SED_2D.COMM
# Computes the local strain energy density for
# a XFEM cracked plate subjected to a constant
# tensile stress through a double integral
# Utilizes the DEFI_GROUP and POST_ELEM commands

DEBUT(PAR_LOT='NON');

import math
import os

WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_xcrack_sed_2d.dat')
f = open(exportfile,'w')

f.write('==========================================================\
=========\n')
f.write('==========================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
          DENSITY FOR A XFEM CRACKED PLATE SUBJECTED TO A\n\
         CONSTANT TENSILE STRESS THROUGH A DOUBLE INTEGRAL\n')
f.write('==========================================================\
=========\n')
f.write('==========================================================\
=========\n\n')

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young's modulus of steel [MPa]
NU = 0.3        # Poisson's ratio of steel []

# Geometry

c = 10.0        # Half crack length [mm]

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Definition of the crack tip coordinates

x_c = c
y_c = 0.0

# Input for the value R

f.write('INPUT VALUES:\n\n')

R = input('Enter the radius of the circle onto which compute the SED: ')

f.write('Radius of the circle: R = ' + '{0:2.2f}'.format(R) + '\n\n')

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

```

```
MAIL=LIRE_MAILLAGE(FORMAT='MED',);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=('Edge_1','Edge_2',),),);

# Application of the plane strain conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Definition of the XFEM crack

CRACK=DEFI_FISS_XFEM(MODELE=MODE,
                     DEFI_FISS=_F(FORM_FISS='SEGMENT',
                                  PFON_ORIG=(-x_c,y_c,0.0,),
                                  PFON_EXTR=(x_c,y_c,0.0,),),);

# Introduction of the crack into the model

MODEX=MODI_MODELE_XFEM(MODELE_IN=MODE,
                       FISSURE=CRACK,);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

CONST=AFFE_CHAR_MECA(MODELE=MODEX,
                     LIAISON_XFEM='OUI',
                     DDL_IMPO=(_F(GROUP_NO=('Node_1','Node_2',),
                                 DX=0.0,),
                               _F(GROUP_NO=('Node_3','Node_4',),
                                 DY=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODEX,
                    LIAISON_XFEM='OUI',
                    PRES_REP=_F(GROUP_MA=('Edge_1','Edge_2',),
                                PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODEX,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=CONST,),
                          _F(CHARGE=LOAD,),),);

# Definition of the mesh in postprocessing

MA_XFEM=POST_MAIL_XFEM(MODELE=MODEX,);

# Creation of the groups of elements

MA_XFEM=DEFI_GROUP(reuse =MA_XFEM,
                   MAILLAGE=MA_XFEM,
                   CREA_GROUP_MA=(_F(NOM='Face_2',
                                     TYPE_MAILLE='2D',
                                     OPTION='SPHERE',
                                     POINT=(-x_c,y_c),
                                     RAYON=R),
                                  _F(NOM='Face_3',
                                     TYPE_MAILLE='2D',
                                     OPTION='SPHERE',
                                     POINT=(x_c,y_c),
                                     RAYON=R),),);

# Definition of the visualization model

MOD_VISU=AFFE_MODELE(MAILLAGE=MA_XFEM,
                     AFFE=_F(TOUT='OUI',
                             PHENOMENE='MECANIQUE',
                             MODELISATION='D_PLAN',),);

# Definition of the XFEM field

RES_XFEM=POST_CHAM_XFEM(MODELE_VISU=MOD_VISU,
                        RESULTAT=RESU,);

# Calculation of the XFEM nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RES_XFEM=CALC_ELEM(reuse =RES_XFEM,
                   RESULTAT=RES_XFEM,
                   OPTION=('SIGM_ELNO','SIEQ_ELNO','ETOT_ELNO'),);

# Calculation of the XFEM nodal solutions
# The nodal values from each element sharing
# that node are averaged

RES_XFEM=CALC_NO(reuse =RES_XFEM,
                 RESULTAT=RES_XFEM,
                 OPTION=('SIGM_NOEU','SIEQ_NOEU',),);

# Calculation of the strain energy density

# Left crack tip

SEL_CA=POST_ELEM(INTEGRALE=_F(GROUP_MA='Face_2',
                   NOM_CHAM='ETOT_ELNO',
                   NOM_CMP='TOTALE',),
                   RESULTAT=RES_XFEM,);

# Right crack tip

SER_CA=POST_ELEM(INTEGRALE=_F(GROUP_MA='Face_3',
                   NOM_CHAM='ETOT_ELNO',
                   NOM_CMP='TOTALE',),
                   RESULTAT=RES_XFEM,);

# Printing of the tables

IMPR_TABLE(TABLE=SEL_CA,);

IMPR_TABLE(TABLE=SER_CA,);

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          UNITE=80,
          RESU=_F(MAILLAGE=MA_XFEM,
                  RESULTAT=RES_XFEM,),);

```

```
# Printing of the final values

# Definition of the asymptotic value for the SED

K_I = S0 * math.sqrt(math.pi * c)

REF  = (1.0 + NU) * (5.0 - 8.0 * NU) * K_I ** 2
REF /= 8.0 * math.pi * R * E

# Extraction of the values from the tables

SEDL_TAB = SEDL_CA.EXTR_TABLE()
SEDR_TAB = SEDR_CA.EXTR_TABLE()

SEDL = SEDL_TAB.values()['MOYE_TOTALE']
SEDR = SEDR_TAB.values()['MOYE_TOTALE']

f.write('\n=================================================\n')
f.write('                 RESULTS')
f.write('\n=================================================\n\n')

f.write(' Coordinates of the crack tip: x_c = ' + '{0:3.10f}'.format(-x_c) + \
'\n')
f.write('                               y_c = ' + '{0:3.10f}'.format(y_c) + \
'\n\n')

f.write('         Computed SED = ' + '{0:2.5e}'.format(SEDL[0]) + '\n')
f.write('      Theoretical SED = ' + '{0:2.10e}'.format(REF) + '\n')
f.write('     Percentual error = ' + '{0:4.2e}'.format((abs(SEDL[0] / REF - \
1.0) * 100.0)) + '%\n\n')

f.write(' Coordinates of the crack tip: x_c = ' + '{0:3.10f}'.format(x_c) + \
'\n')
f.write('                               y_c = ' + '{0:3.10f}'.format(y_c) + \
'\n\n')

f.write('         Computed SED = ' + '{0:2.5e}'.format(SEDR[0]) + '\n')
f.write('      Theoretical SED = ' + '{0:2.10e}'.format(REF) + '\n')
f.write('     Percentual error = ' + '{0:4.2e}'.format((abs(SEDR[0] / REF - \
1.0) * 100.0)) + '%\n\n')

f.write('=================================================\n')

f.close()

FIN();
```

C.9 PLATE_NOTCH_SED_2D.COMM

Algorithm C.9. Finite Element computation of $\mathcal{SED}$ through a double integral for a notched plate.

```
# File PLATE_NOTCH_SED_2D.COMM
# Computes the local strain energy density for
# a 135°-notched plate subjected to a constant
# tensile stress through a double integral
# Utilizes the DEFI_GROUP and POST_ELEM commands

DEBUT(PAR_LOT='NON');

import os

```

```
WORKING_DIR = '...'

exportfile = os.path.join(WORKING_DIR,'fe_plate_notch_sed_2d.dat')
f = open(exportfile,'w')

f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n')
f.write('       FINITE ELEMENT COMPUTATION OF THE LOCAL STRAIN \
ENERGY\n\
          DENSITY FOR A 135°-NOTCHED PLATE SUBJECTED TO A\n\
         CONSTANT TENSILE STRESS THROUGH A DOUBLE INTEGRAL\n')
f.write('=========================================================\
=========\n')
f.write('=========================================================\
=========\n\n')

# Definition of some parameters of the problem

# Material

E  = 210000.0   # Young's modulus of steel [MPa]
NU = 0.3        # Poisson's ratio of steel []

# Boundary conditions

S0 = 100.0      # Applied tensile stress [MPa]

# Definition of the notch tip coordinates

x_c = 0.0
y_c = 0.0

# Input for the value R

f.write('INPUT VALUES:\n\n')

R = input('Enter the radius of the circle onto which compute the SED: ')

f.write('Radius of the circle: R = ' + '{0:2.2f}'.format(R) + '\n\n')

# Definition of the material

STEEL=DEFI_MATERIAU(ELAS=_F(E=E,
                            NU=NU,),);

# Reading of the mesh

MAIL=LIRE_MAILLAGE(FORMAT='MED',);

# Creation of the group of elements

MAIL=DEFI_GROUP(reuse =MAIL,
                MAILLAGE=MAIL,
                CREA_GROUP_MA=(_F(NOM='Circle',
                                 TYPE_MAILLE='2D',
                                 OPTION='SPHERE',
                                 POINT=(x_c,y_c),
                                 RAYON=R),),);

# Reorientation of the normals towards the outside

MAIL=MODI_MAILLAGE(reuse =MAIL,
                   MAILLAGE=MAIL,
                   ORIE_PEAU_2D=_F(GROUP_MA=('Edge_1','Edge_2','Edge_3',),),);
```

```

# Application of the plane strain conditions

MODE=AFFE_MODELE(MAILLAGE=MAIL,
                 AFFE=_F(TOUT='OUI',
                         PHENOMENE='MECANIQUE',
                         MODELISATION='D_PLAN',),);

# Application of the material properties to the domain

MATE=AFFE_MATERIAU(MAILLAGE=MAIL,
                   AFFE=_F(TOUT='OUI',
                           MATER=STEEL,),);

# Application of the constraints

SYMM=AFFE_CHAR_MECA(MODELE=MODE,
                    DDL_IMPO=(_F(GROUP_MA=('Edge_1','Edge_2',),
                                DY=0.0,),
                             _F(GROUP_NO='Vertex_2',
                                DX=0.0,),),);

# Application of the external loads

LOAD=AFFE_CHAR_MECA(MODELE=MODE,
                    PRES_REP=_F(GROUP_MA='Edge_3',
                               PRES=-S0,),);

# Definition of the linear elastic static model

RESU=MECA_STATIQUE(MODELE=MODE,
                   CHAM_MATER=MATE,
                   EXCIT=(_F(CHARGE=SYMM,),
                         _F(CHARGE=LOAD,),),);

# Calculation of the nodal solutions
# WARNING: For nodes shared between more than one
# element, the nodal values are calculated separately

RESU=CALC_ELEM(reuse =RESU,
               RESULTAT=RESU,
               OPTION=('SIGM_ELNO','SIEQ_ELNO','ENEL_ELNO',),);

# Calculation of the nodal solutions
# The nodal values from each element sharing
# that node are averaged

RESU=CALC_NO(reuse =RESU,
             RESULTAT=RESU,
             OPTION=('SIGM_NOEU','SIEQ_NOEU','ENEL_NOEU',),);

# Calculation of the strain energy density

SED_CA=POST_ELEM(INTEGRALE=_F(GROUP_MA='Circle',
                   NOM_CHAM='ENEL_ELNO',
                   NOM_CMP='TOTALE',),
                   RESULTAT=RESU,);

# Printing of the table

IMPR_TABLE(TABLE=SED_CA,);

# Saving the output in MED format

IMPR_RESU(FORMAT='MED',
          RESU=_F(MAILLAGE=MAIL,
                  RESULTAT=RESU,),);

# Printing of the final values

SED_TAB = SED_CA.EXTR_TABLE()

SED = SED_TAB.values()['MOYE_TOTALE']

f.write('\n==================================================\n')
f.write('                   RESULTS')
f.write('\n==================================================\n\n')

f.write(' Coordinates of the notch tip: x_c = ' + '{0:3.10f}'.format(x_c) + \
'\n')
f.write('                               y_c = ' + '{0:3.10f}'.format(y_c) + \
'\n\n')

f.write('         Computed SED = ' + '{0:2.5e}'.format(SED[0]) + '\n\n')

f.write('==================================================\n')

f.close()

FIN();
```

BIBLIOGRAPHY

[1] T.L. Anderson. *Fracture Mechanics: Fundamentals and Applications*. 3rd ed. CRC Press, 2004 (cit. on p. 20).

[2] J.R. Barber. *Elasticity*. 2nd ed. New York: Kluwer Academic Publishers, 2003 (cit. on pp. 3, 7–10).

[3] K.-J. Bathe. *Finite Element Procedures*. Prentice Hall, 1996 (cit. on pp. 27–34, 58).

[4] S.M. Beden, S. Abdullah, and A.K. Ariffin. "A review of fatigue crack propagation models for metallic components". In: *European Journal of Scientific Research* **28** (2009), pp. 364–397 (cit. on p. xx).

[5] T. Belytschko and T. Black. "Elastic crack growth in Finite Elements with minimal remeshing". In: *International Journal for Numerical Methods in Engineering* **45** (1999), pp. 601–620 (cit. on p. 42).

[6] A.P. Boresi, K.P. Chong, and J.D. Lee. *Elasticity in Engineering Mechanics*. 3rd ed. John Wiley and Sons, 2011 (cit. on p. 5).

[7] A. Düster, H. Bröker, and E. Rank. "The *p*-version of the finite element method for three-dimensional curved thin-walled structures". In: *International Journal for Numerical Methods in Engineering* **52** (2001), pp. 673–703 (cit. on pp. 48, 58).

[8] A. Düster, C. Fischer, and W. Fricke. "Beurteilung der Schwingfestigkeit von Schweißverbindungen aus Basis der lokalen Formänderungsenergiedichte". In: *Jahrbuch der Schiffbautechnischen Gesellschaft* **105** (2011) (cit. on pp. 25, 59).

[9] *Eurocode 3, Design of steel structures, Part 1-9: Fatigue*. EN 1993-1-9, Brussels, CEN. 2005 (cit. on pp. xix, 25).

[10] *Eurocode 9, Design of aluminium structures, Part 2: Structures susceptible to fatigue*. ENV 1999-2, Brussels, CEN. 2000 (cit. on p. xix).

[11] T.-P. Fries and T. Belytschko. "The extended/generalized finite element method: An overview of the method and its applications". In: *International Journal for Numerical Methods in Engineering* **84** (2010), pp. 253–304 (cit. on pp. 39, 40, 42).

[12] E.E. Gdoutos. *Fracture Mechanics, An Introduction*. 2nd ed. Springer, 2005 (cit. on p. 21).

[13] B. Gross and A. Mendelson. "Plane elastostatic analysis of V-notched plates". In: *International Journal of Fracture Mechanics* **8** (1972), pp. 267–276 (cit. on pp. 12, 20).

[14] A. Hobbacher, ed. *Recommendations for fatigue design of welded joints and components*. IIW Doc XIII-1965-03/XV-1127-03. 2006 (cit. on p. xix).

[15] G.R. Irwin. “Analysis of stresses and strains near the end of a crack traversing a plate”. In: *Journal of Applied Mechanics* **24** (1957), pp. 361–364 (cit. on p. 20).

[16] P. Lazzarin, F. Berto, F.J. Gomez, and M. Zappalorto. “Some advantages derived from the use of the strain energy density over a control volume in fatigue strength assessments of welded joints”. In: *International Journal of Fatigue* **30** (2008), pp. 1345–1357 (cit. on pp. 21, 25, 45).

[17] P. Lazzarin and R. Tovo. “A unified approach to the evaluation of linear elastic stress fields in the neighborhood of cracks and notches”. In: *International Journal of Fracture* **78** (1996), pp. 3–19 (cit. on pp. 11, 12).

[18] P. Lazzarin and R. Tovo. “A notch stress intensity factor approach to the stress analysis of welds”. In: *Fatigue and Fracture of Engineering Materials and Structures* **21** (1998), pp. 1089–1103 (cit. on pp. 12, 13, 21).

[19] P. Lazzarin and R. Zambardi. “A finite-volume-energy based approach to predict the static and fatigue behavior of components with sharp V-shaped notches”. In: *International Journal of Fracture* **112** (2001), pp. 275–298 (cit. on pp. xx, 21, 23–25).

[20] P. Livieri and P. Lazzarin. “Fatigue strength of steel and aluminium welded joints based on generalized stress intensity factors and local strain energy values”. In: *International Journal of Fracture* **133** (2005), pp. 247–276 (cit. on p. 25).

[21] N.I. Muskhelishvili. *Some basic problems in the mathematical theory of elasticity*. Noordhoff International Publishing, 1977 (cit. on p. 13).

[22] P.C. Paris and F. Ergodan. “A crytical analysis of crack propagation laws”. In: *Journal of Basic Engineering* **85** (1963), pp. 528–533 (cit. on p. xix).

[23] P.C. Paris, M.P. Gomez, and W.P. Anderson. “A rational analytic theory of fatigue”. In: *The Trend in Engineering* **13** (1961), pp. 9–14 (cit. on p. xix).

[24] D. Radaj, C.M. Sonsino, and W. Fricke. *Fatigue assessment of welded joints by local approaches*. 2nd ed. Woodhead Publishing Limited, 2006 (cit. on p. xix).

[25] M.H. Sadd. *Elasticity: Theory, Applications, and Numerics*. 2nd ed. Academic Press, 2009 (cit. on pp. 1–5, 17).

[26] A. Saxena. *Nonlinear Fracture Mechanics for Engineers*. CRC Press, 1998 (cit. on p. 58).

[27] R.I. Stephens, A. Fatemi, R.R. Stephens, and H.O. Fuchs. *Metal Fatigue in Engineering*. 2nd ed. John Wiley and Sons, 2000 (cit. on pp. 21, 24).

[28] B. Szabó and I. Babuška. *Introduction to Finite Element Analysis: Formulation, Verification and Validation*. John Wiley and Sons, 2011 (cit. on pp. 31, 32, 38, 43).

[29] B. Szabó, A. Düster, and E. Rank. "The *p*-version of the Finite Element Method". In: *Encyclopedia of Computational Mechanics*. Ed. by E. Stein, R. de Borst, and T. J.R. Hughes. Vol. 1. 2004, pp. 119–139 (cit. on pp. 38, 39).

[30] S.P. Timoshenko and J.N. Goodier. *Theory of Elasticity*. 2nd ed. McGraw-Hill, 1951 (cit. on pp. 1, 21, 22, 53).

[31] D.J. Unger. *Analytical Fracture Mechanics*. Academic Press, 1995 (cit. on p. 18).

[32] E.W. Weisstein. *Legendre-Gauss quadrature. From MathWorld — A Wolfram Web Resource*. URL: `http://mathworld.wolfram.com/Legendre-GaussQuadrature.html` (cit. on p. 43).

[33] H.M. Westergaard. "Bearing pressures and cracks". In: *Journal of Applied Mechanics* **6** (1939), pp. 49–53 (cit. on pp. 18, 20).

[34] M.L. Williams. "Stress singularities resulting from various boundary conditions in angular corners of plates in extension". In: *Journal of Applied Mechanics* **74** (1952), pp. 526–528 (cit. on pp. 6–8).

[35] M.L. Williams. "On the stress distribution at the base of a stationary crack". In: **24** (1957), pp. 109–114 (cit. on p. 13).

[36] Z. Yosibash, A. Bussiba, and I. Gilad. "Failure criteria for brittle elastic materials". In: *International Journal of Fracture* **125** (2004), pp. 307–333 (cit. on p. 45).

[37] E. Zahavi and D. Barlam. *Nonlinear Problems in Machine Design*. CRC Press, 2001 (cit. on pp. 33, 34, 78).

[38] M. Zappalorto. "Notch Mechanics under Elastic and Elastic-Plastic Conditions". PhD thesis. Università degli Studi di Padova, 2009 (cit. on p. 9).

INDEX

And remember...

Ghost Figure. Multiaxial fatigue crack propagated inside a viscoelastic material component (the author's slipper).

...Cracks are everywhere!