XLVI CILAMCE
VITÓRIA - ES - BRAZIL - 2025
XLVI IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGENEERING

PyCrack: Finite element software for linear elastic fracture mechanics analysis

Silvia Corbani[1], Gabriel M. M.Santos[1], Leonardo O. Rodriguez[1], André M. B. Pereira[2]

[1]*Dept. de Estruturas, Universidade Federal do Rio de Janeiro*
Ilha do Fundão, 21945-970, Rio de Janeiro/RJ, Brazil
corbani@poli.ufrj.br, gabrielmm@poli.ufrj.br, leonardo_rodriguez@poli.ufrj.br
[2]*Instituto de Computação, Universidade Federal Fluminense*
Rua Passo da Pátria 156, 24210-240, Niterói/RJ, Brazil
andre@ic.uff.br

Abstract. Teaching linear elastic fracture mechanics faces challenges related to conceptual understanding, computational modeling, and the time required. The modeling process involves predicting discontinuities, crack mapping and mesh discretization, particularly in finite elements. The objective of this work is to develop a Python code to numerically analyze cracked components, based on the hypotheses of linear-elastic fracture mechanics with axial symmetry. The two-dimensional analysis is performed using quadratic tetrahedral finite elements. The methodology includes the automatic generation of the mesh mapping the elements around the crack tip, which are usually called rosette. The user of this software can choose whether to numerically analyze a rosette of 1 to 3 rings. The first ring is composed of Quarter Point elements. The calculation of the stress intensity factor (SIF) is done using the J-Integral methods and the displacement correlation method. A user interface is developed using the Tkinter library in the Python programming environment. The Python program helps students interested in the implementation to understand the process, and the Tkinter interface is important to familiarize with the data inputs of problems involving fracture mechanics. Validation examples are compared with the literature to verify the code.

Keywords: fracture mechanics, finite element, python.

1 Introduction

Fatigue crack growth represents an important cause of failure in engineering structures and mechanical components subjected to cyclic loading. The structural integrity of these systems is essential to ensure both functional performance and operational safety. In this context, the development of research aimed at accurately estimating the service life of structures becomes indispensable, not only to extend their usage but also to prevent catastrophic failures that may lead to significant human and financial losses. Therefore, the analysis of the behavior of structural elements containing defects plays a central role in the education of engineers and in the advancement of knowledge in the field.

Linear Elastic Fracture Mechanics (LEFM) provides a theoretical foundation for the analysis and design of cracked structures subjected to repeated loading. However, the practical application of these concepts, particularly through numerical simulations, still presents technical challenges, requiring specific strategies to accurately represent the stress field near the crack tip. Given the relatively recent adoption of LEFM in structural analysis, the implementation of algorithms in accessible programming languages, such as Python, significantly contributes to the dissemination and practical application of its concepts, as well as to the development of educational and computational tools aimed at structural engineering.

In this project, a program was developed to determine parameters of Linear Elastic Fracture Mechanics (LEFM), using a rectangular plate model discretized by a mesh of eight-node quadrilateral elements. The user provides the geometric data and material properties. From the displacement vector, calculated using the Finite Element Method (FEM), the J-integral and the stress intensity factor are obtained.

FEM was chosen due to its widespread use in 2D problem analysis and its inclusion in elective courses during engineering undergraduate programs.

For validation, the results obtained were compared with data available in the literature. The program helps users quickly understand the calculation process of the stress intensity factor, as well as the implementation of the

J-Integral and displacement correlation method.

2 J Integral

By considering the J-integral, solving the expression eq. (1) under the assumptions of a two-dimensional plane, nonlinear elastic solid, using a small-scale displacement gradient and neglecting body forces as shown in Rice [1]

$$J = \int_{\Gamma} \left[W n_1 - \sigma_{ij} n_i \frac{\partial u_j}{\partial x_1} \right] ds. \tag{1}$$

where, in the cracked component Γ an arbitrary path is defined around the crack tip, starting from the lower crack face and ending at the upper crack face, enclosing the tip to evaluate the J-integral (see Fig. 1). n_1 is the normal vector to the contour σ_{ij} is the stress, u_j is the displacement e ds is the it is the increment of arc length in Γ. W it is the strain energy density defined by eq. (2)

$$W\left(\varepsilon_{kl}\right) = \int_0^{\varepsilon_H} \sigma_{ij} d\varepsilon_{ij}. \tag{2}$$

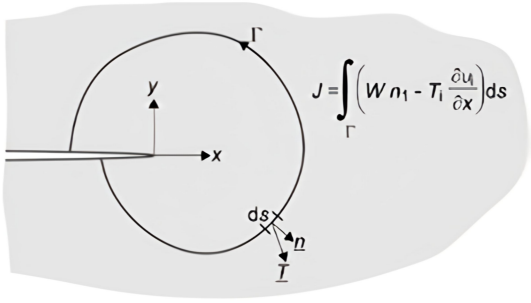


Figure 1. J-integral along the arbitrary contour Γ. Source: Janssen et al. [2]

For a through-thickness crack under tension in plane strain condition, J and G are equivalent, where G is the energy release rate. Thus, the J-integral is also the energy release rate Rice [1], in other words,

$$J\delta l = -\delta\pi \tag{3}$$

where δl is the virtual crack extension in the direction normal to the crack front and within the crack plane. The expressions for the area integral in a two-dimensional problem are described for an annular region of area A_I around the tip of a single crack as shown in Li et al. [3]. The boundary of the area A_I is called C. Thus,

$$G = J = -\int_C \left[\sigma_{ij} \frac{\partial u_i}{\partial x_1} - W\delta_{1j} \right] n_j ds = \int_C P_{1j} n_j ds \tag{4}$$

m is the normal vector to the contour C. The annular region can be defined by four (4) contours: C_1, C_2, C_3 e C_4. C_1 e C_2 are the outer and inner circular contours of this region, respectively. C_3 e C_4 are contours parallel to the upper face of the crack and the lower face of the crack, respectively. Since that n_j = $q_1.m_j$ = 0 (free surface without tension!) and m_j = 0 in C_3 e C_4, results that

$$G\Delta a = \Delta a \oint_C \left[\sigma_{ij}\frac{\partial u_i}{\partial x_1} - W\delta_{1j}\right] q_1 m_j ds = -\Delta a \oint_C P_{1j} q_1 m_j ds \tag{5}$$

where q_1 is a continuous interpolation function over the area A_I. q_1 is unity in C_1 and null in C_2. Δa can be interpreted as a virtual crack extension in the direction normal to the crack front. The contour adopted was C_1 reduced, corresponding to the crack tip.

Applying the divergence theorem

$$G\Delta a = -\Delta a \int_{A_I} \left[\frac{\partial}{\partial x_j}\left(P_{1j} q_1\right)\right] dA = -\Delta a \int_{A_I} \left[\frac{\partial P_{1j}}{\partial x_j} q_1 + P_{1j}\frac{\partial q_1}{\partial x_j}\right] dA \tag{6}$$

The direct calculation in Eshelby [4] and Eshelby [5] show that the expression reduces to equantion 7 as a area integral

$$G = -\int_{A_I} \left[P_{1j}\frac{\partial q_1}{\partial x_j}\right] dA = \int_{A_I} \left[\sigma_{ij}\frac{\partial u_i}{\partial x_1} - W\delta_{1j}\right] \frac{\partial q_1}{\partial x_j} dA \tag{7}$$

q_1 is interpreted as a mapping function between configurations of virtual crack length. For the two-dimensional case, the function q_1 is equal to one on the contour C_1 and vanishes on the contour C_2. Thus, the J-integral is evaluated around the crack tip.

3 Implementation strategy of the Python code for the 2D problem

The chosen strategy for writing the code includes a data input module for mesh generation and automatic creation of a quarter-point element rosette at the crack tip. The purpose of this work is to develop a concise program on LEFM that can be modeled in 2D. The program is written in the Python 3.8.5 language and is operated on a Intel ® Core™ i7-8565 CPU @ 2GHz, 64 bit, 200GB Computer.

A first consideration is that the program models only half of the specimen, with its lower edge representing the connection to the other half, where the crack will be inserted, as illustrated in Fig. 2. The program works with a standard mesh model to represent the structure, but requires user input to define the geometry (plate dimensions), material parameters (elastic modulus, Poisson's ratio), and also the crack characteristics such as length, number of rosette rings, and number of elements discretizing it.

The finite element mesh is generated automatically by the code, with element numbering starting from the element at the bottom-left corner and proceeding to the top-right corner, as schematically shown in Fig. 3 (a) and (b). The maximum number of elements is an input parameter of the code, defined as the total number of elements in each direction. Although structural components are often discretized using triangular elements, this code employs quadratic quadrilateral elements, as the focus is on providing a simple implementation to facilitate student understanding.

To obtain more accurate results around the crack, the quarter-point element was implemented only in the elements surrounding it for a linear elastic approach. The program maps the node representing the crack tip and its adjacent nodes, moving them closer (to a distance equal to a quarter of the element edge) to the mapped node. To identify this node, a structure was created to find the elements surrounding the crack tip. The element to the right is given by the number of elements along the Y-axis multiplied by the number of elements along the crack face, and the one to the left is the same number of elements in Y multiplied by the number of elements along the crack face minus one. The nodes to be moved are node 5 of each element and the intermediate shared node between them (node 6 of the left element or node 8 of the right element), see Fig. 4. After mapping these nodes, it is only necessary to assign them values in the function that generates the coordinate vectors to position them correctly in a quarter of element as illustrated in Fig. 5.

Next, the program uses a function to generate the coordinate matrix and the degrees of freedom matrix based on input data and data automatically calculated by the code. These matrices are stored and accessed as needed.

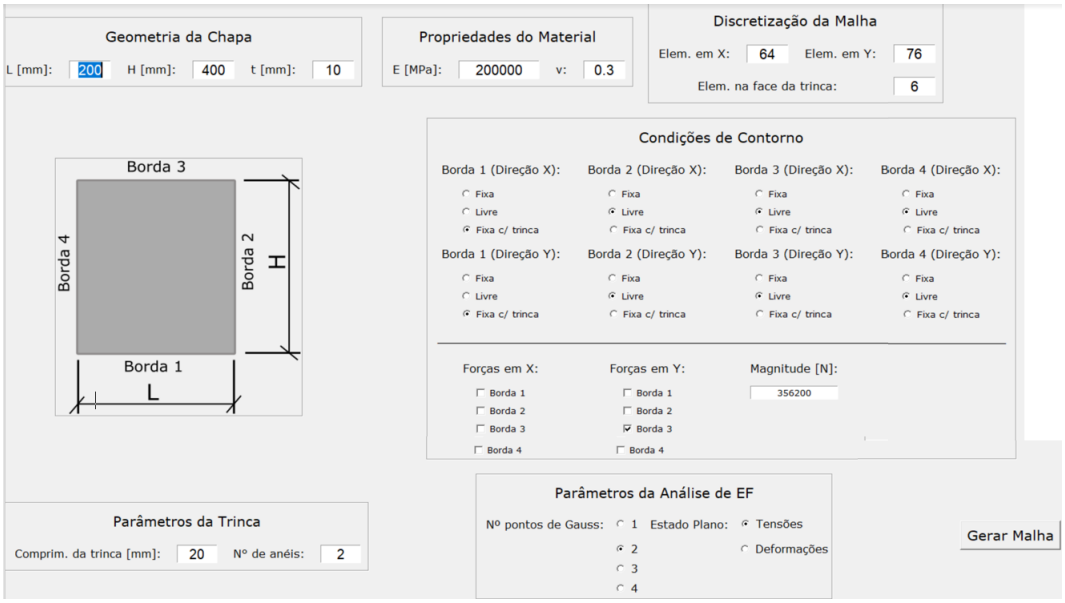


Figure 2. Data input window.

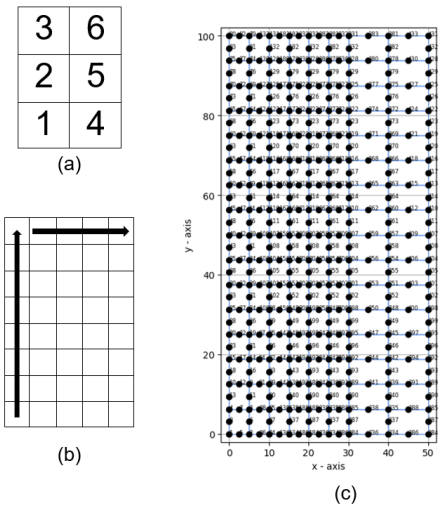


Figure 3. Mesh generation strategy: (a) example of numbering for 6 elements; (b) schematic indication of numbering; (c) mesh generated by the code.

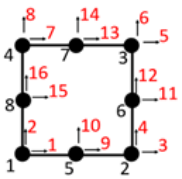


Figure 4. Quadratic quadrilateral elements: 8 nodes.

With the obtained coordinates, it is possible to plot the undeformed mesh, which corresponds exactly to the next step of the code: representing the modeled mesh with node labeling and scale in millimeters.

Next, the finite element analysis is performed, where the constitutive matrix and the stiffness matrix are generated to calculate the displacements related to each element. To compute these matrices, the program uses specific functions, such as the function that generates the stiffness matrix for each element.

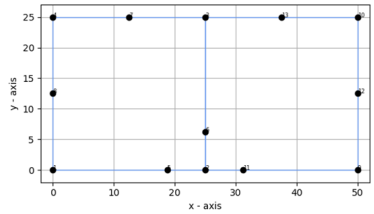


Figure 5. Quarter Point element around the crack tip.

After this process, the displacement and stress vectors are obtained. The program plots the mesh again, now in its deformed configuration, indicated by dashed red lines.

At the end of the finite element analysis, with the displacement and stress values available, it is possible to calculate the J-integral. The stress intensity factor K and the numerical geometry function are determined from the J-integral. The closed domain used for integration, which covers data from one or more elements around the crack tip, can correspond to 1, 2, or 3 rings, as illustrated in Fig. 6.

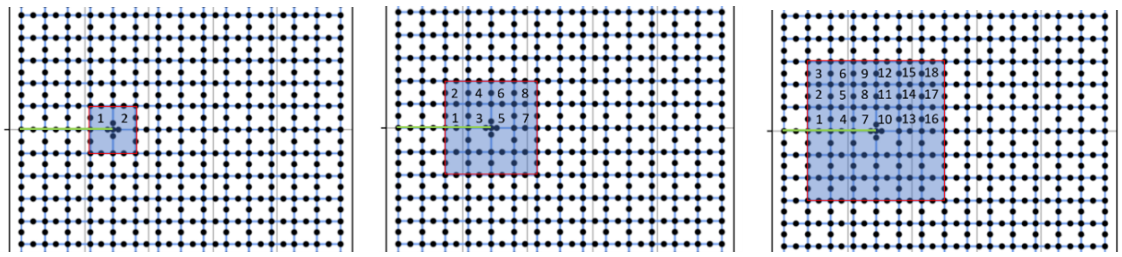


Figure 6. J-integral elements: one ring; two rings; three rings.

After exiting the function that calculates these values, the analytical geometry function is computed, and with this value, it is possible to obtain the error between the calculated and analytical values. Finally, the program returns the output screen of the data illustrated in Fig. 7.

For consideration of the displacement correlation method, all nodes belonging to the crack faces are mapped. Based on the displacements obtained from the finite element analysis, it is possible to calculate the stress intensity factor. In the Python code, both options are available in addition to the analytical solution of the problem. This allows students, even with limited knowledge of finite elements, to carry out the exercises proposed during the course. Furthermore, if they need to use a more complex fracture mechanics program, they are able to understand the purpose of configuring certain input data and how it will impact their results.

The code was implemented by undergraduate students participating in scientific initiation. Besides applying the programming language they learned at the beginning of their course, they were introduced to programming details used in a finite element problem and how to integrate finite element analysis with fracture mechanics. Subsequently, the code was used in three graduate-level classes of the academic program in civil engineering. With this code, students do not need prior familiarity with finite element software and are able to understand the key data involved in a numerical analysis. To date, the program has been downloaded by 164 people. The program is available in Zenodo repository Santos et al. [6] (see Fig. 8).

The PyCrack program helps students interested in implementation by providing a clear understanding of the process, while the Tkinter interface plays an important role in learning about fracture mechanics problems.

4 Conclusions

This work described a Python code developed to solve crack problems. Although the displacement correlation method was not detailed, it is also implemented in the code, in addition to the J-integral method. The TKinter interface helps the students to understand the problem involving fracture mechanics. Finally, this work is inspired by the teaching approach of Professor Luiz Fernando Martha.

Acknowledgements. The authors acknowledge the granting of scholarships awarded under the 2018 and 2020

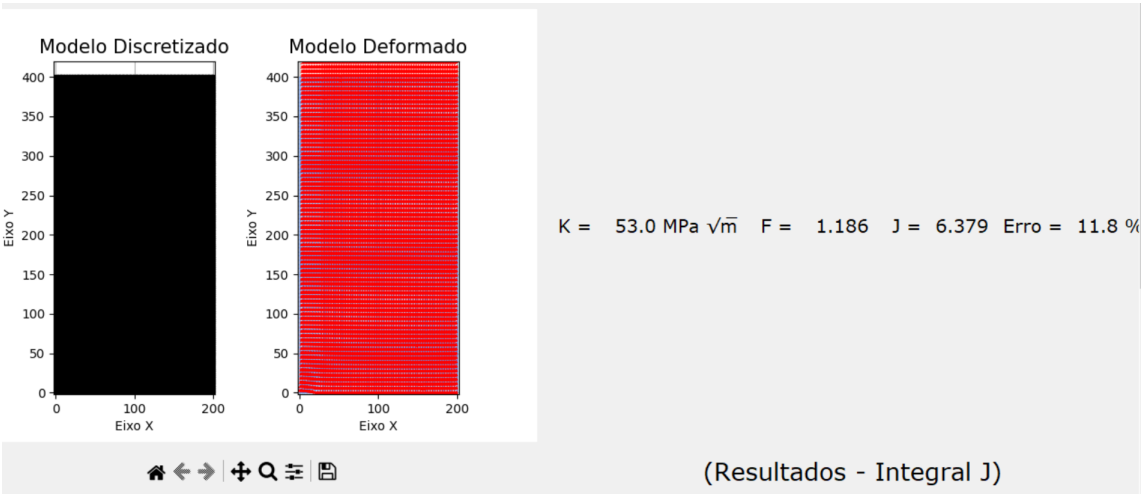


Figure 7. Data output window.

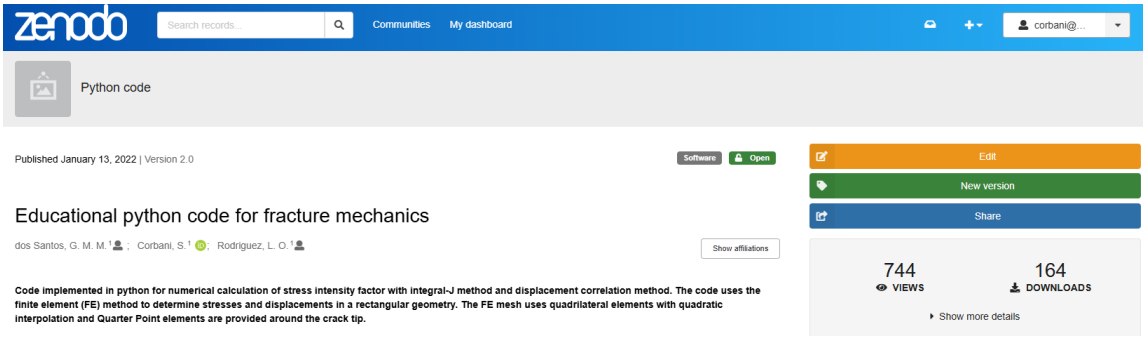


Figure 8. Zenodo window.

CEG/CEPG Programa Institucional de Bolsas de Iniciação Científica, funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (PIBIC-CNPq), as well as the counterpart funding provided by Universidade Federal do Rio de Janeiro (PIBIC-UFRJ).

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] J. R. Rice. A path independent integral and the approximate analysis of strain concentration by notches and cracks. *Journal of Applied Mechanics*, vol. 35, n. 2, pp. 379–386, 1968.
[2] M. Janssen, J. Zuidema, and R. Wanhill. *Fracture Mechanics: Fundamentals and Applications*. CRC Press, London, 2nd edition edition, 2006.
[3] F. Li, C. Shih, and A. Needleman. A comparison of methods for calculating energy release rates. *Engineering Fracture Mechanics*, vol. 21, n. 2, pp. 405–421, 1985.
[4] J. Eshelby. The continuum theory of lattice defects. *Solid State Physics*, vol. 3, pp. 79–144, 1956.
[5] J. D. Eshelby. The elastic energy-momentum tensor. *Journal of Elasticity*, vol. 5, pp. 321–335, 1975.
[6] G. M. M. Santos, S. Corbani, and L. O. Rodriguez. Educational python code for fracture mechanics (2.0), 2022.