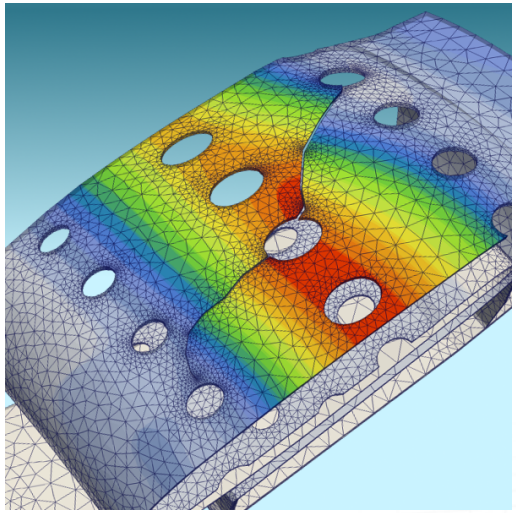
Z-cracks - 3D fracture mechanics simulation

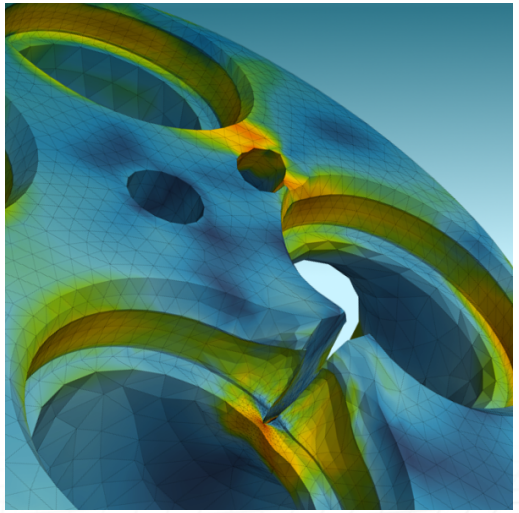
June 2, 2020

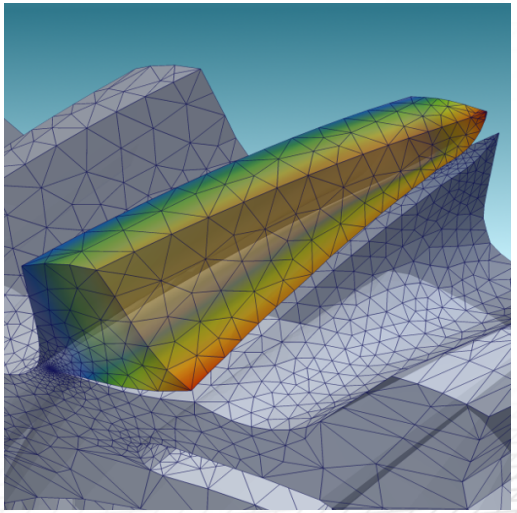
Z-set | Material & structure analysis suite

3D fracture mechanics simulation with Z-cracks

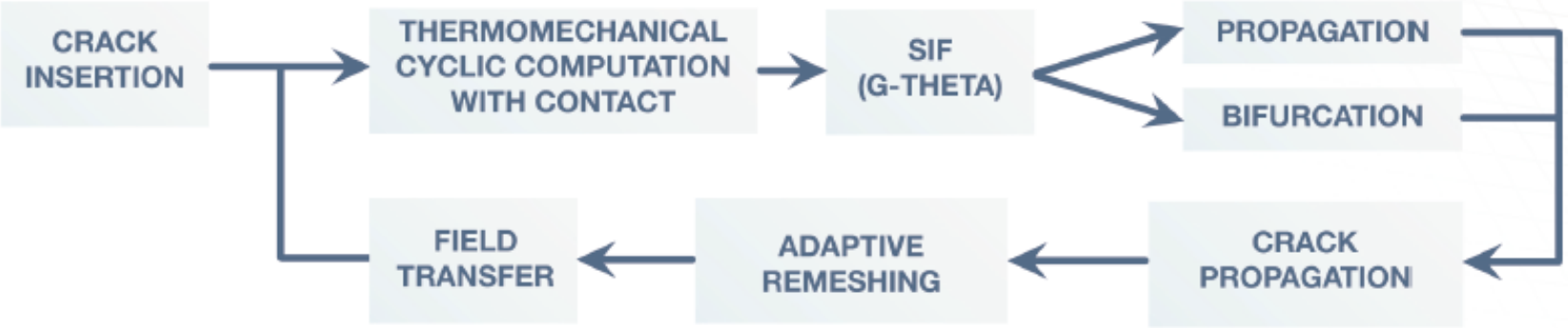
- insertion of an arbitrary number of cracks in an unstructured FE mesh
- static crack SIF computation
- crack propagation simulation :
  - ⋆ cyclic loading
  - ⋆ automatic remeshing
  - ⋆ propagation laws
  - ⋆ bifurcation criteria (out-of-plane propagation)
  - ⋆ material non-linearity and transfer of plasticity initial state during cycles



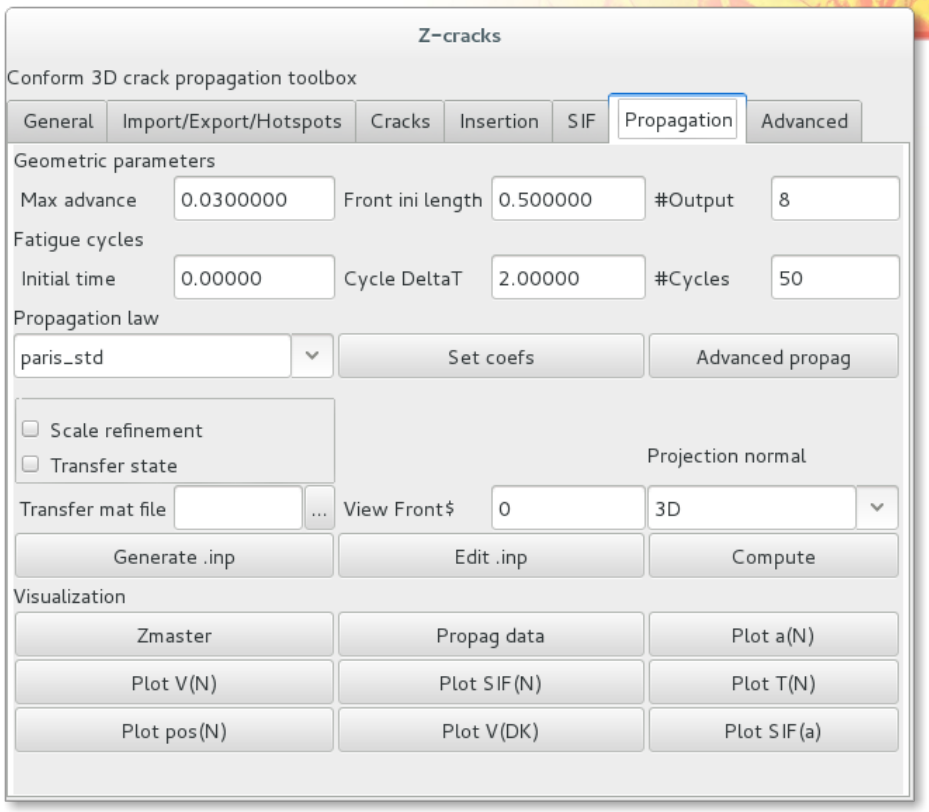




Z-cracks - operating principle



Z-cracks - Graphical User Interface



- interactive setup of 3D fracture mechanics simulations
- automatic generation of input files for different steps
- default settings for main parameters
- interactive post-processing/visualization
- storage of the current state of the simulation setup in a history file

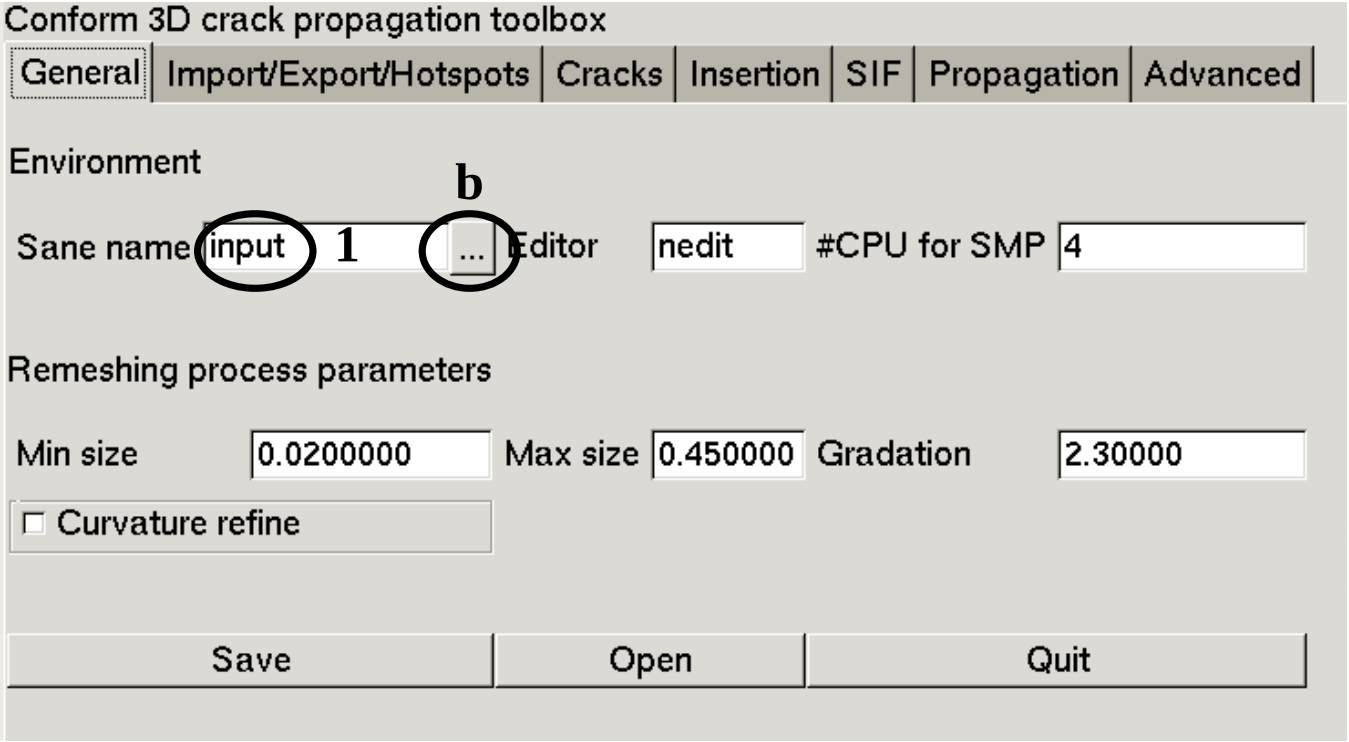
Import of a FE cyclic input file

Input for step 1 (import)

- input deck to do 1 cycle on the uncracked (sane) geometry
- must include preloading steps if any
- in the linear-elastic case the unloading steps are not necessary

Import GUI controls

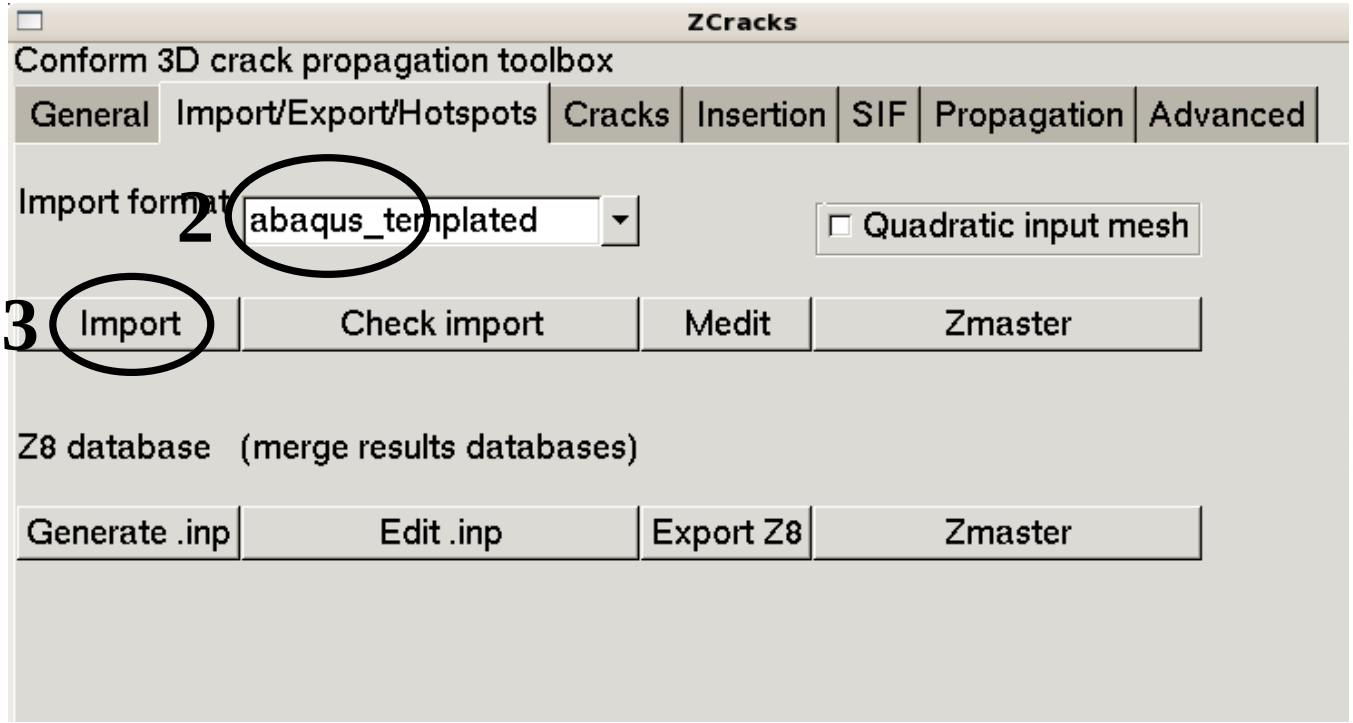
- 1. give the name of the FE input file



- button (**b**) can be used to browse for the input file

Import GUI controls

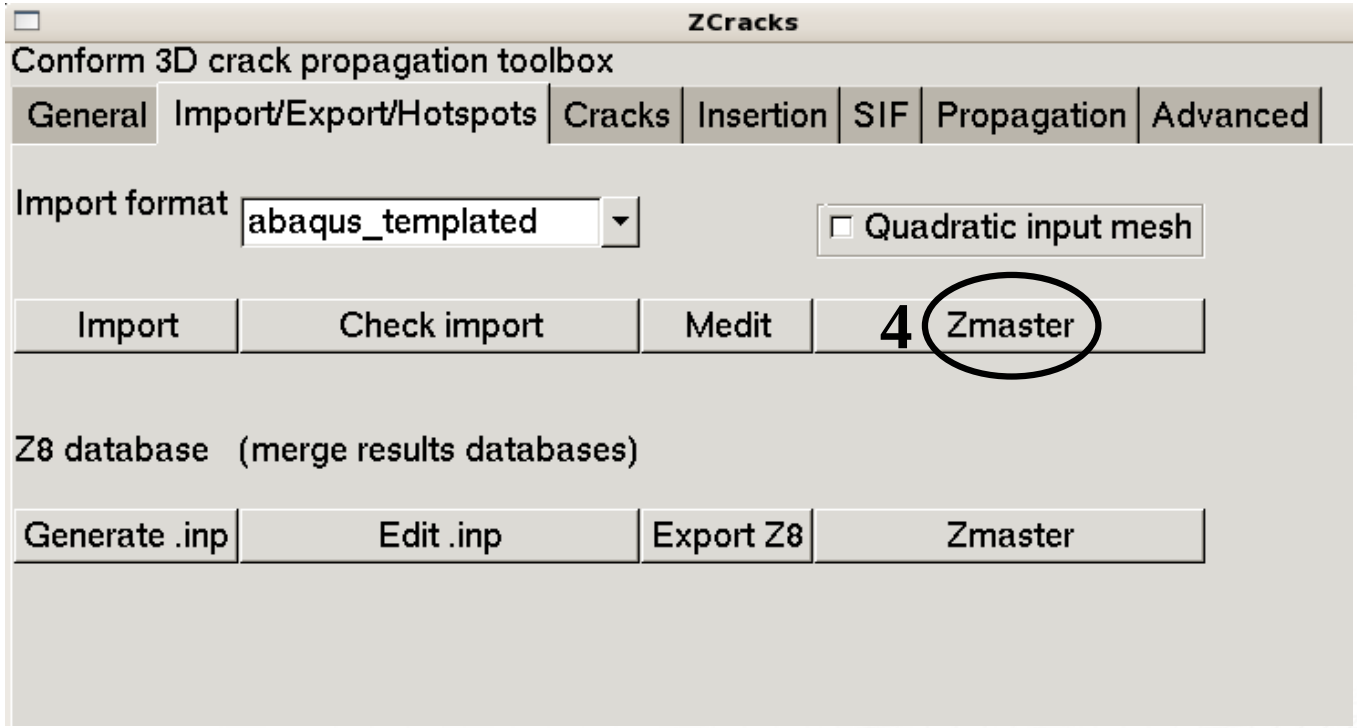
- 2. select an FE code (Abaqus, Ansys, Zebulon, Samcef)
- 3. click **Import**



Import results

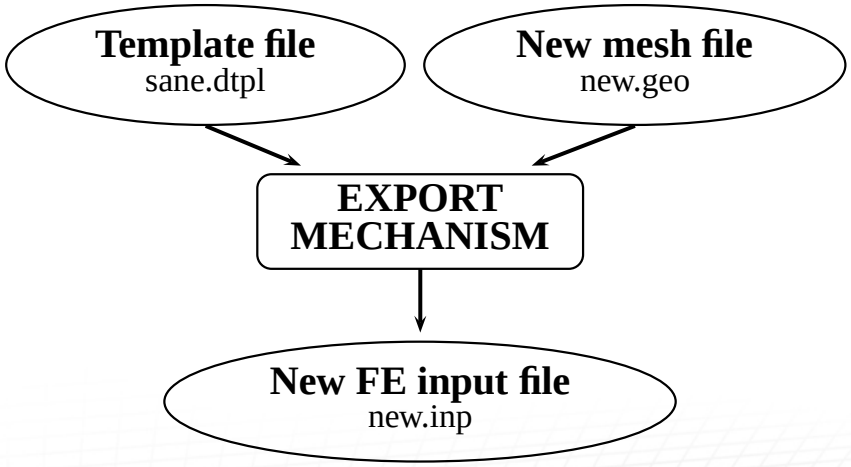
Import results:

- a template file *input.dtpl*
- a Zebulon mesh *input.geo*
- click on **Zmaster** (4) to view the imported mesh



Import template file

- ASCII file (may be modified by the user)
- created from the original FE file in order to:
  - ⋆ remove mesh objects (nodes, elements, group definitions)
  - ⋆ preserve bcs, loads and step definitions
  - ⋆ if needed, change bc/loads syntax for commands using **named** objects (nsets, bsets, surfaces preserved after remeshing)
- used to generate a new FE input file when the mesh is changed (crack insertion/propagation)



Step 2

Crack definition/meshing

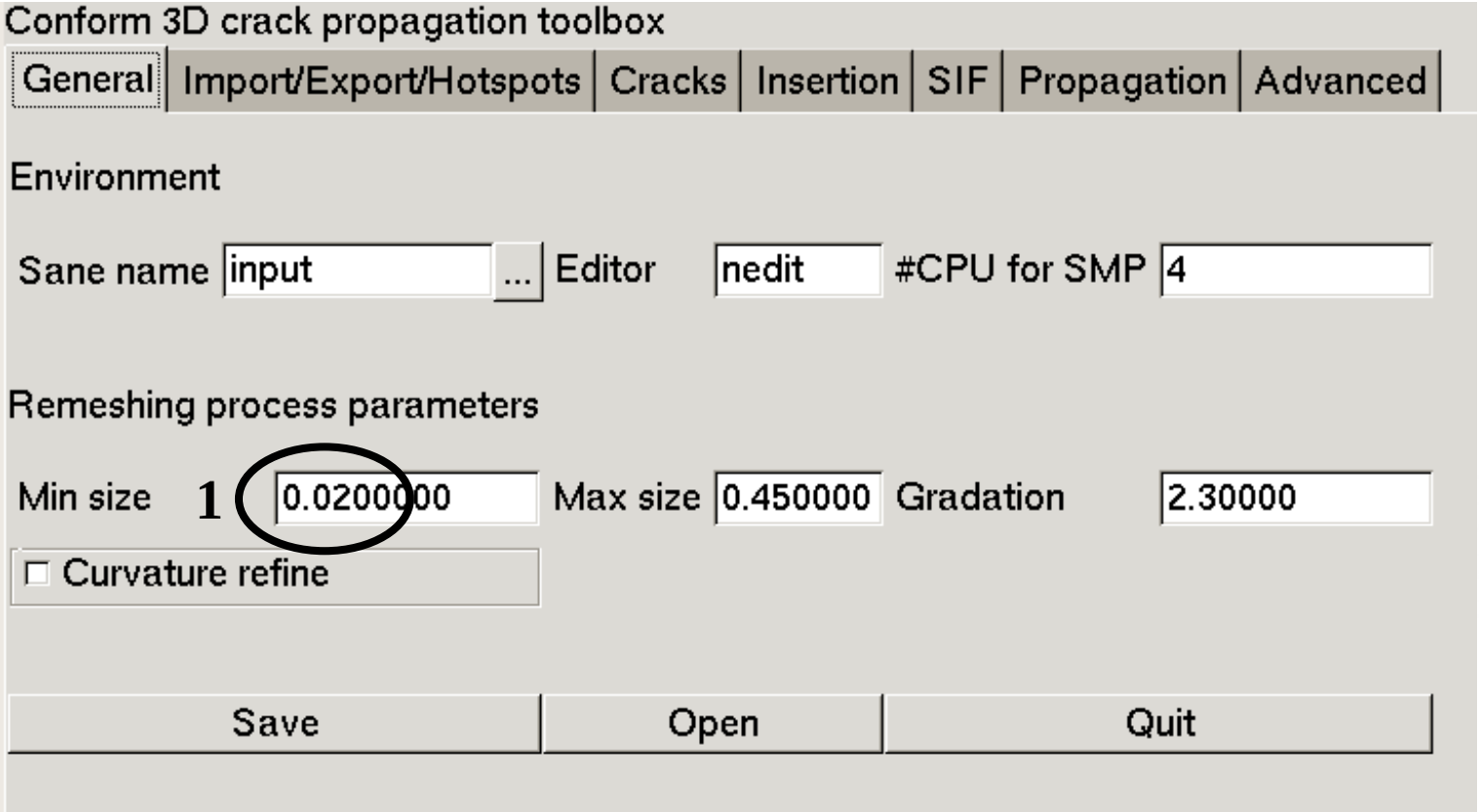
Input for step 2 (crack meshing)

Can be defined either by:

- the geometric definition of simple crack shapes (circles, ellipse)
  - ⋆ coordinates of the circle (ellipse) center
  - ⋆ vector normal to the crack plane
  - ⋆ crack size (circle radius)
  - ⋆ for an ellipse definition of an additional minor axis direction and radius
- or a mesh (Zebulon format) of the crack surface (2D or 3D shell elements)

Crack mesh GUI controls

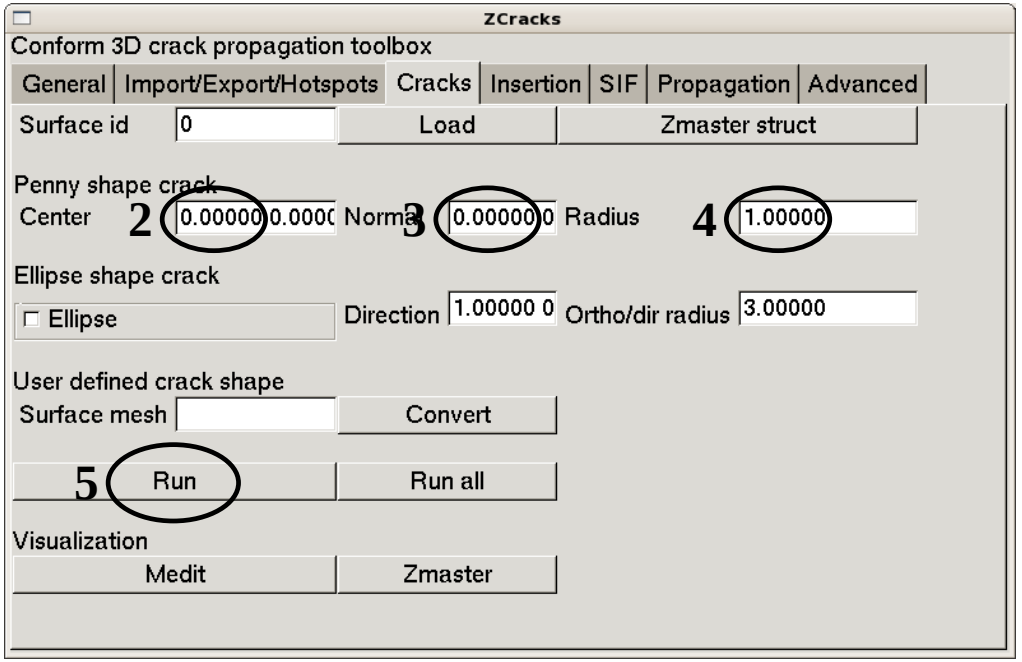
- 1. Size of elements at the crack tip



size in the same unit as the one used for the mesh

Crack mesh GUI controls

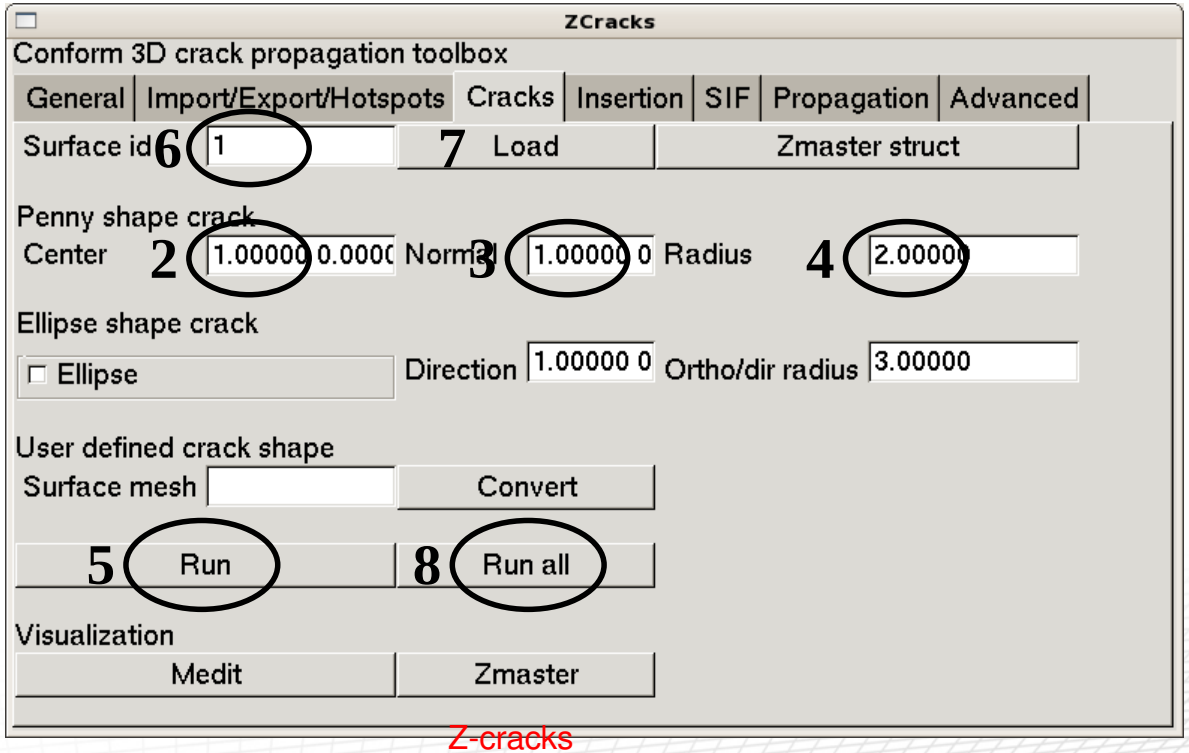
- 2. Crack (circle) center
- 3. Vector normal to the crack plane
- 4. Size (circle radius)
- 5. Click **Run** to mesh the crack



vectors (center, normal) are specified as 3 float values separated by a blank space a "." is mandatory in the definition of float values

Multiple cracks definition

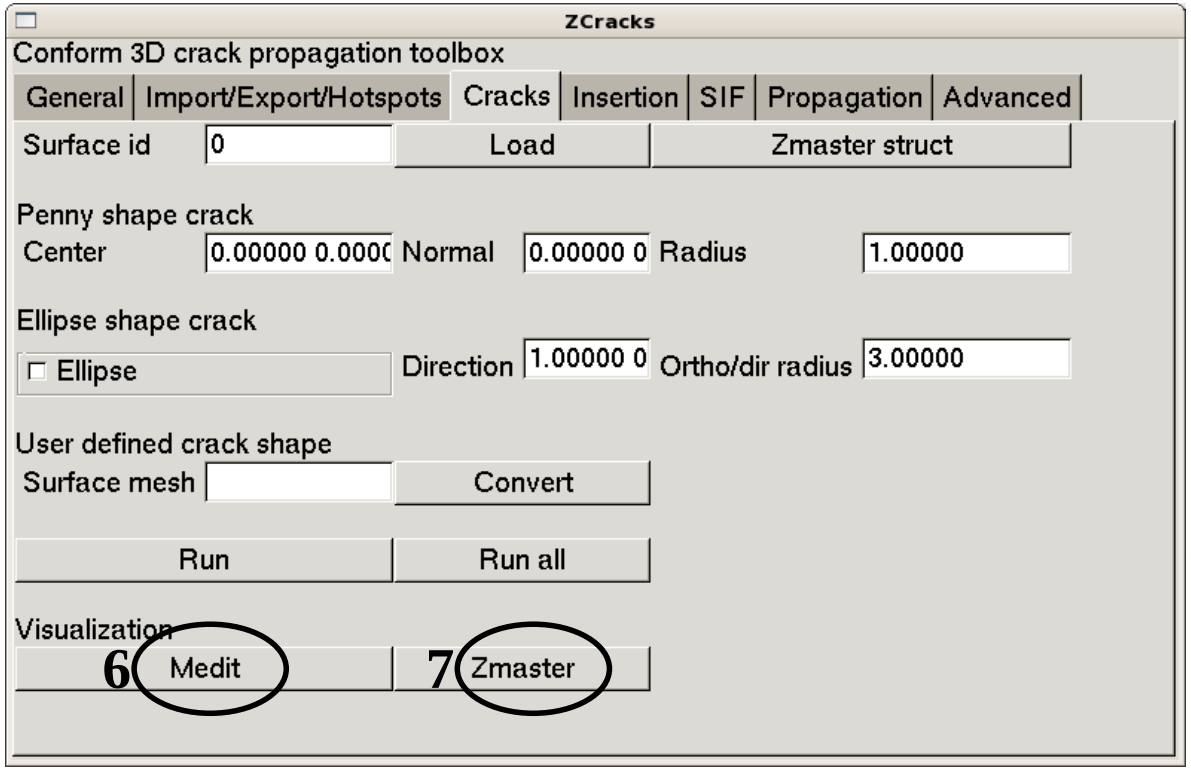
- to define an additional crack, just change the crack id in (6), and define the geometric parameters in (2),(3),(4) as in the previous case
- clicking on (7) loads the geometric definitions associated to the crack id specified in (6), thus allowing to change the values if needed
- (5) remesh only the crack specified in (6), while (8) mesh all cracks at once



Z-cracks

Crack mesh results

- a mesh (Zebulon format) of the crack(s) surface(s) *crack.geo*
- click **Medit** (6) or **Zmaster** (7) to view the crack mesh



size of triangular elements near the crack front should correspond to the one defined in the General Tab (1)

Step 3

Crack insertion in the sane mesh

Generalities: Distene remeshing tools

- For remeshing **Z-cracks** makes an heavy use of various software modules of MeshGems Suite distributed by the **DISTENE** company:
  - ⋆ **MeshGems-SurfOpt**: surface remeshing tool
  - ⋆ **MeshGems-Tetra**: 3D tetrahedral mesh generator taking as input a surface mesh representing the boundary of the object
  - ⋆ **MeshGems-Adapt**: adaptive surface and/or volume remeshing tool, that can handle both a boundary surface mesh and a 3D initial tet mesh at the same time
- those remeshing tools work directly from an input mesh (i.e. no CAO definition is needed), thus offering a great flexibility and an easy interface with arbitrary FE software
- they are automatically driven by **Z-cracks** when performing crack insertion/crack advance remeshing operations
- specific license keys are needed that should be obtained directly from the **Z-set** distributor

Generalities: algorithm

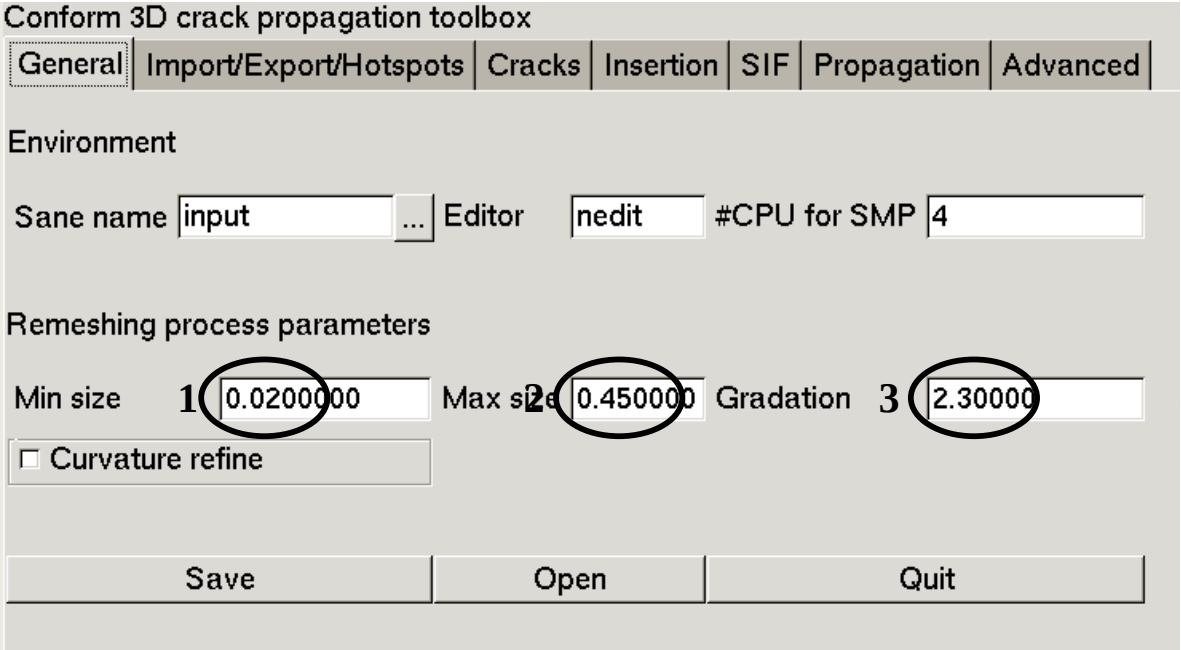
Crack insertion is an automatic 2-phase process that may be roughly summarized as:

- Phase 1: cutting of the original geometry by the crack mesh produced during step 2
  the result is an intermediate mesh with the discontinuity (crack surfaces) explicitly introduced in the geometry.
- Phase 2: remeshing of the previous to obtain a good quality mesh (respecting the size required near the crack front) needed to compute accurate values of SIF at the crack tip

Generalities: remeshing controls

Phase 2 remeshing is mainly controlled by 3 parameters:

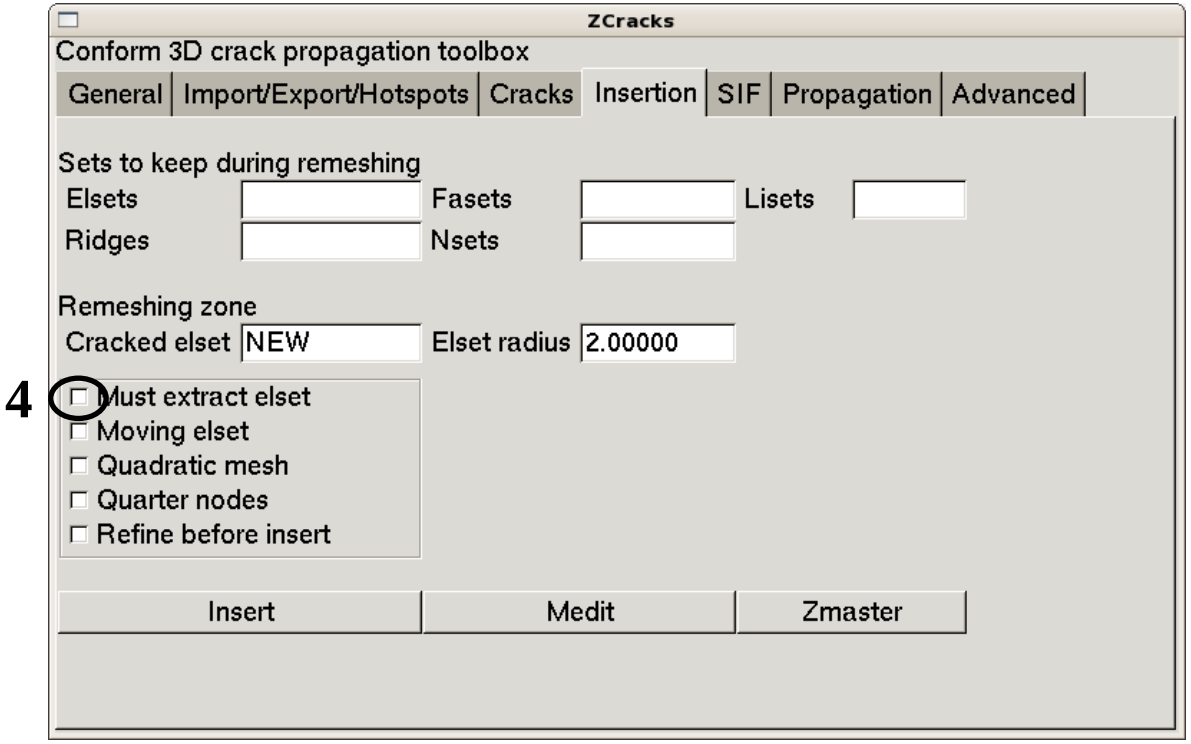
- the **Min size** value (1) of elements at the crack tip
- the **Max size** (2) of elements generated during remeshing
- the **Gradation** factor (3) or element size rate of increase when distance to the crack front is growing



the remesher only handles tet elements, such that quads in the initial geometry will be automatically converted to tets

Generalities: remeshing the whole structure

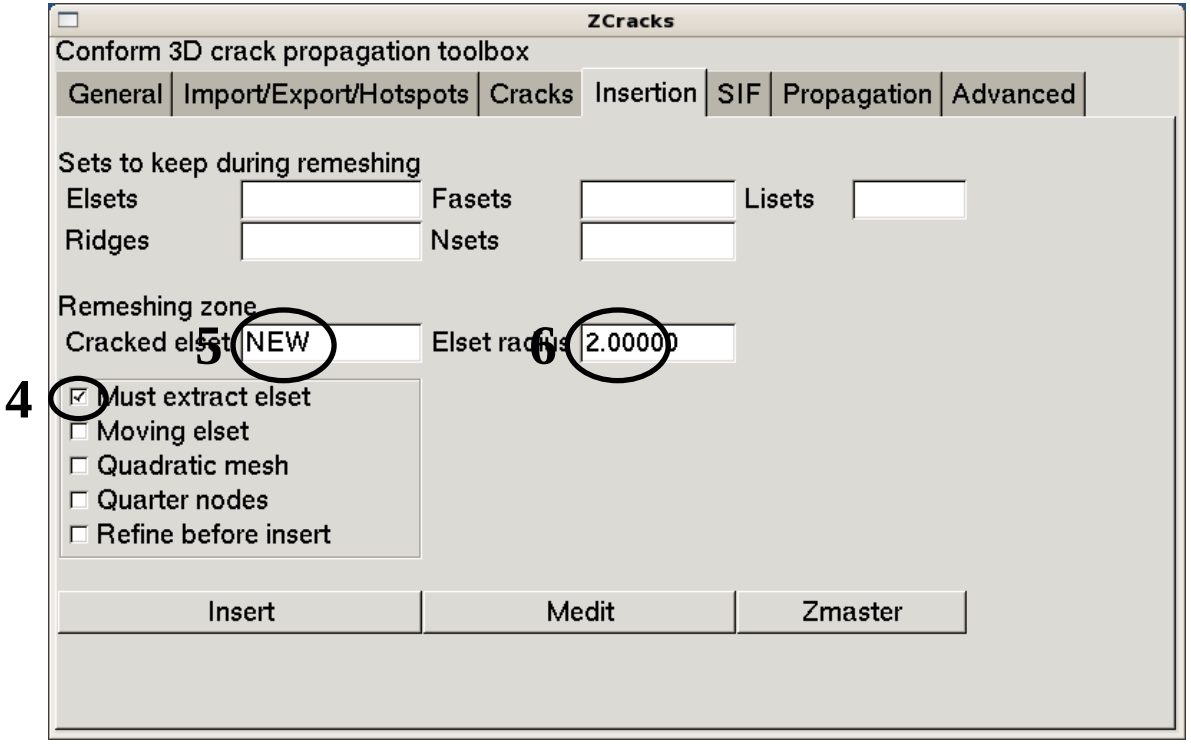
- leave unchecked the **Must extract elset** check-box (4)



- the output mesh contains only volume tet elements
- node and element positions/ids are not preserved

Generalities: remeshing a region around the crack tip

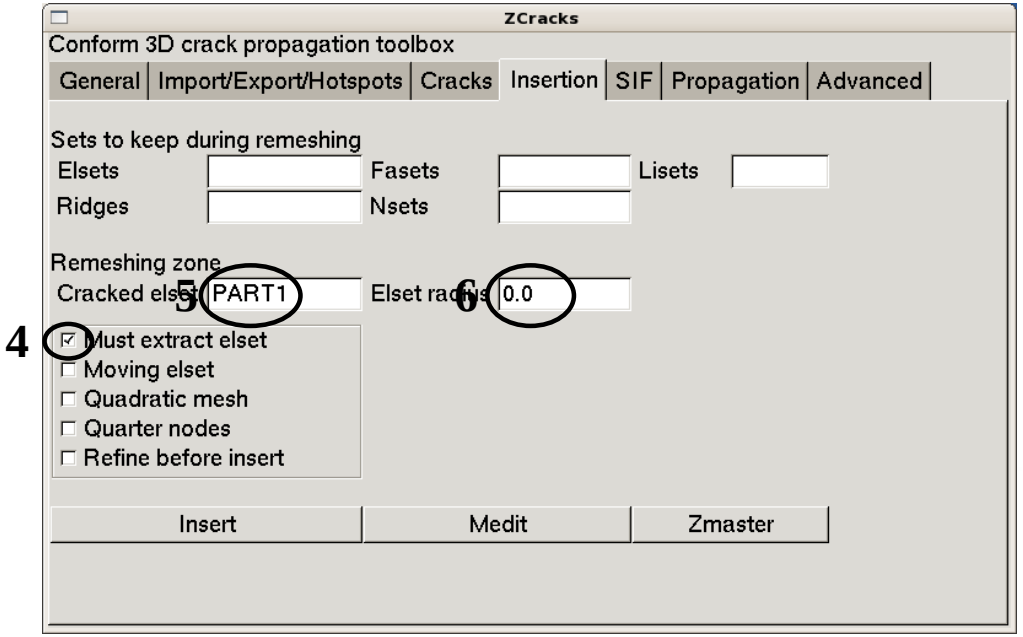
- select **Must extract elset** (4)
- set **NEW** as the name of the elset containing the remeshed region in the **Cracked elset** text-field
- give size of the region using the **Elset radius** (6) text-field



- node and element positions/ids outside the remeshed region are preserved (including non volume elements)

Generalities: remeshing a subset

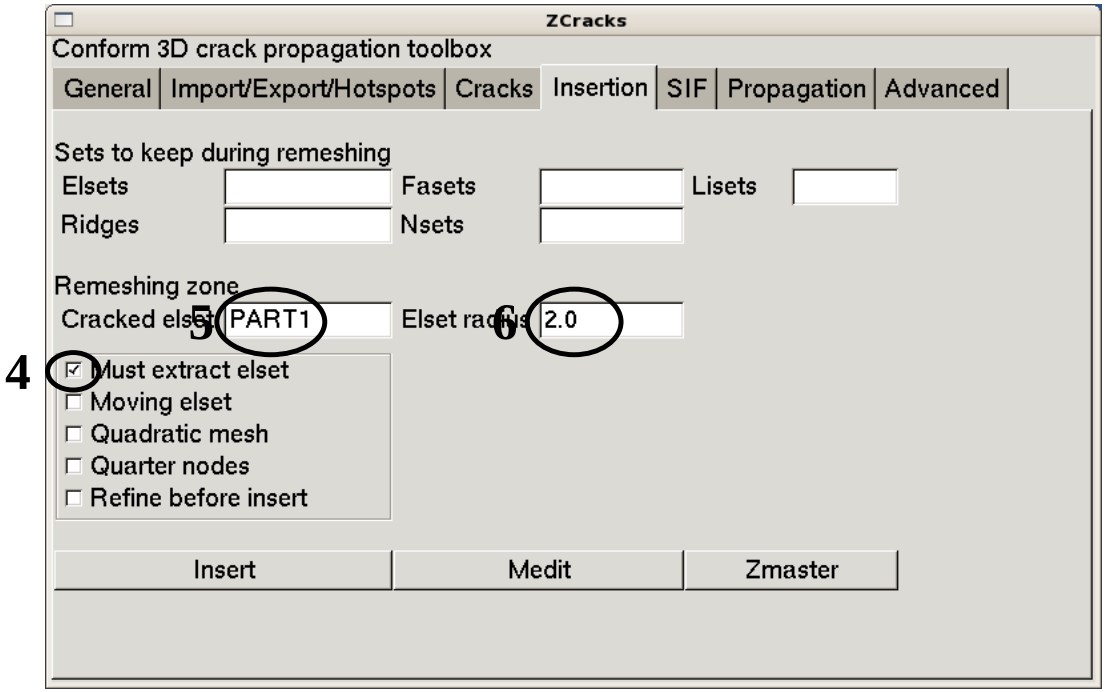
- select **Must extract elset** (4)
- set the name of an elset containing elements of the subset in the **Cracked elset** text-field
- set **0.0** in the **Elset radius** (6) text-field to imply that the whole subset is remeshed



- mandatory for contact between to avoid fusion of contacting nodes: set **Cracked elset** as one of the 2 contacting components (i.e. the one with the crack)

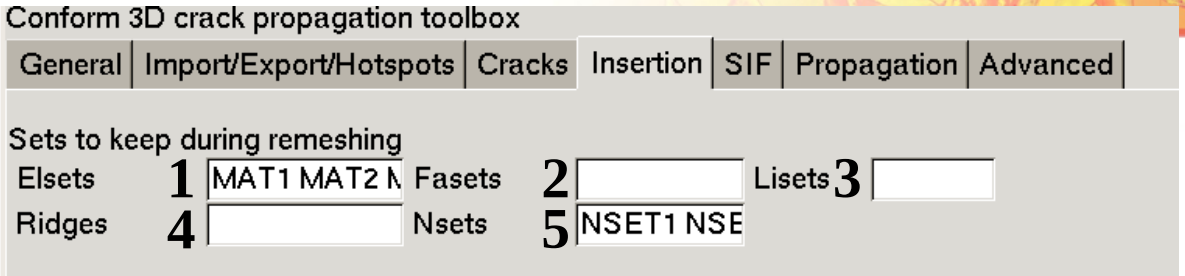
Generalities: remeshing a region inside a subset

- select **Must extract elset** (4)
- set the name of an elset containing elements of the subset in the **Cracked elset** text-field (5)
- set the size of a region around the crack tip using the **Elset radius** (6) text-field



- an elset named **NEW** will be created with the size specified inside the **PART1** elset

Generalities: sets/components preservation



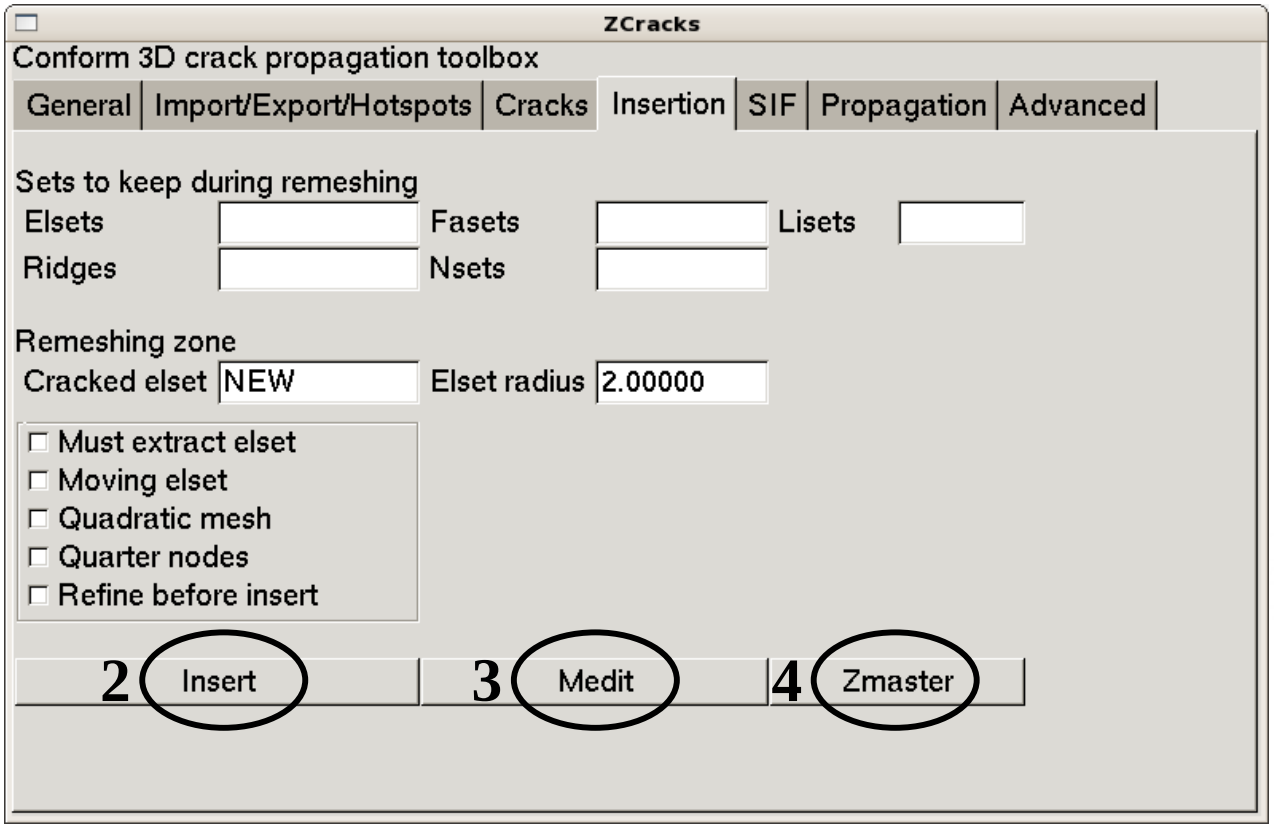
- Elsets (1) : elsets (in the remeshed subset) used to affect material properties should be declared here
- Fasets (2) : fasets are surface elements in the **Z7** mesh. Fasets are automatically created during import from surface loads/bcs and contact definitions found in the input file. Those are automatically preserved and there is no need to specify their names using (2). However additional surface definitions may be added here for preservation
- Nsets (5) : nsets are list of nodes used to apply boundary conditions. Those are automatically created in the output **Z-set** mesh obtained during import. Preservation is not automatic and pertinent nset names should be explicitly defined here to allow valid bcs definition after export

Generalities: guidelines and pitfalls

- prefer named components to apply loads and bcs, instead of node ids
- define an elset containing only supported (volume) elements and boundary conditions and remesh only a subset of the latter
- non-uniform surface loads are not supported
- nodal bcs and nset preservation in the remeshed part:
  - ⋆ should be used only for point load/bcs, because preservation of geometric positions of large collection of nodes result in too much constraints on remeshing and poor quality end result mesh
  - ⋆ ids are not preserved
  - ⋆ only surface nodes can be preserved
  - ⋆ **Z-cracks** attempts to transform nsets to surfaces automatically . Those are automatically preserved (without constraint on the remeshing), and translated back to nsets during export of new FE command files. This works fine for uniform bcs, but cannot be used in a submodeling procedure.

Crack insertion GUI controls

- 1. give the size of elements at the crack tip (**General tab**, same definition as for step 2)
- 2. click **Insert** to run crack insertion



- view the output cracked mesh using **Medit** (3) or **Zmaster** (4)

Crack-related components in the geo file

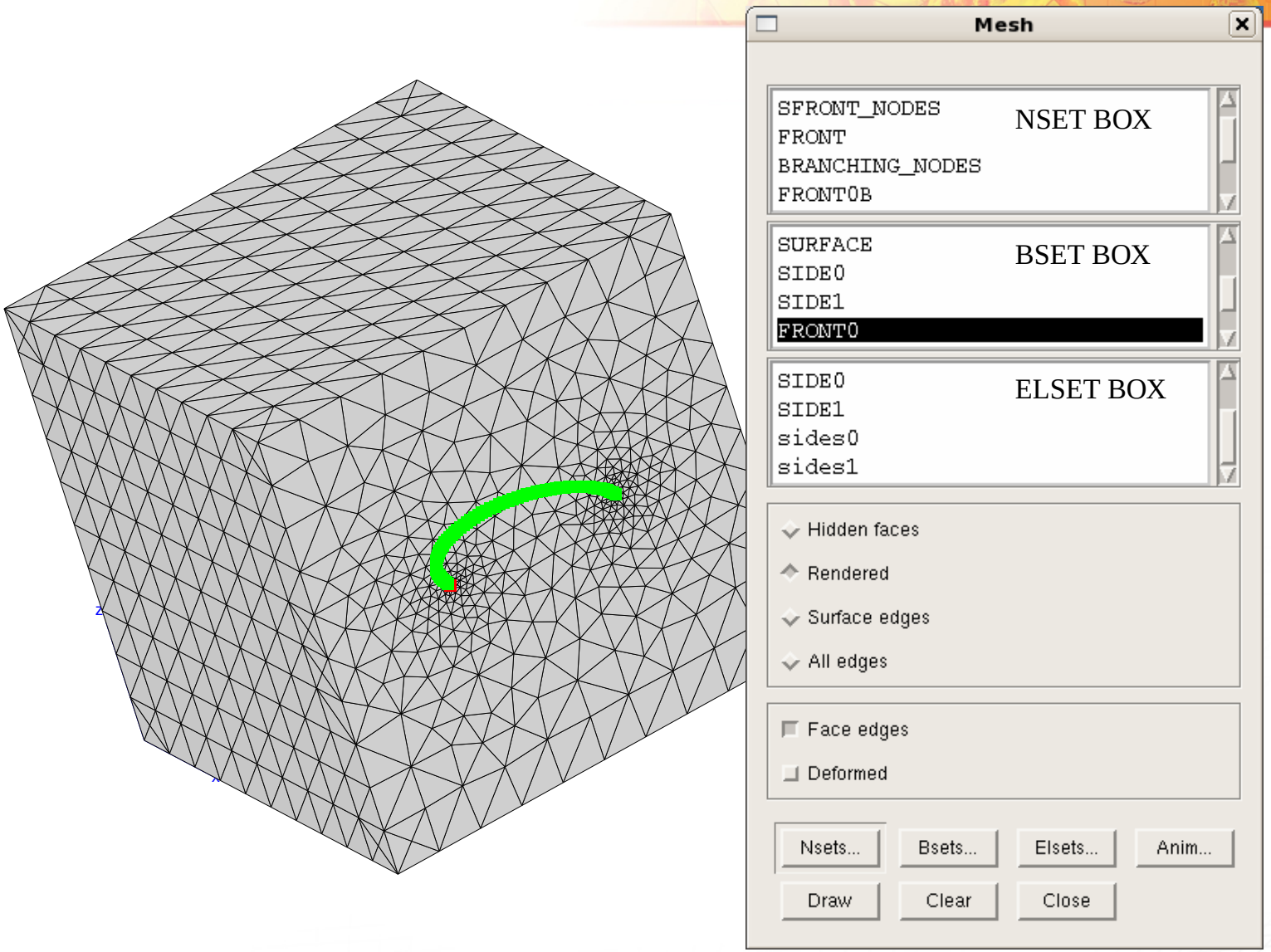
- insertion output is a **Z7** meshfile (**geo** file with name **cracked.geo**) that can be loaded in **Zmaster** for display
- besides mesh objects translated from the FE input file, various crack-related components are added that may be worth looking through to control the output:

⋆ nsets (group of nodes)
FRONT : nodes on crack front
lip : nodes on crack lips

⋆ bsets (surface or line elements)
FRONT0, **FRONT1** ... : individual crack front lines
SIDE0, **SIDE1** : surface elements on both sides lips

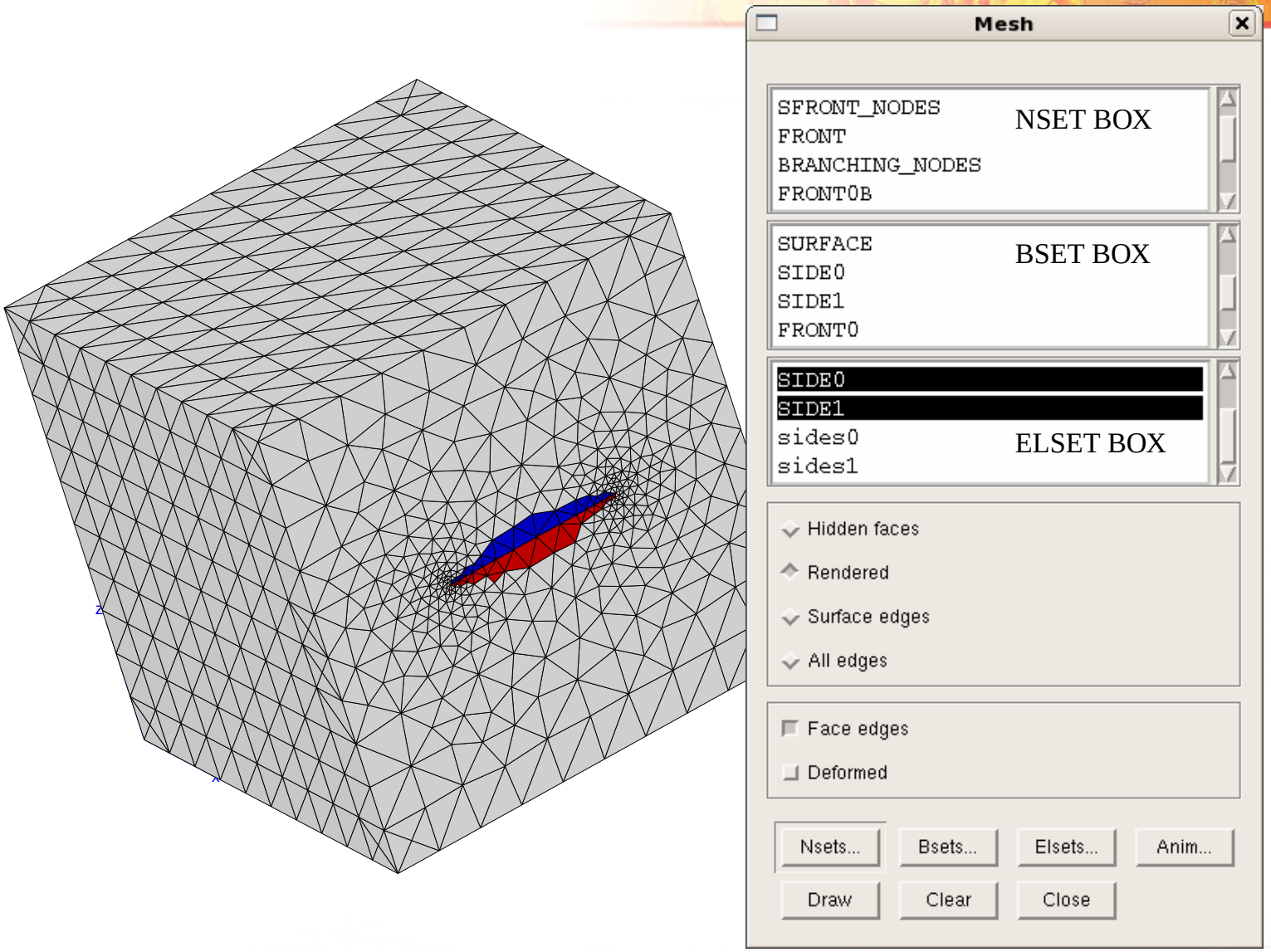
⋆ elsets (group of elements)
SIDE0, **SIDE1** : elements connected to lip nodes on both sides

Crack front



Display of the crack front line set using **Zmaster**

Lip elsets

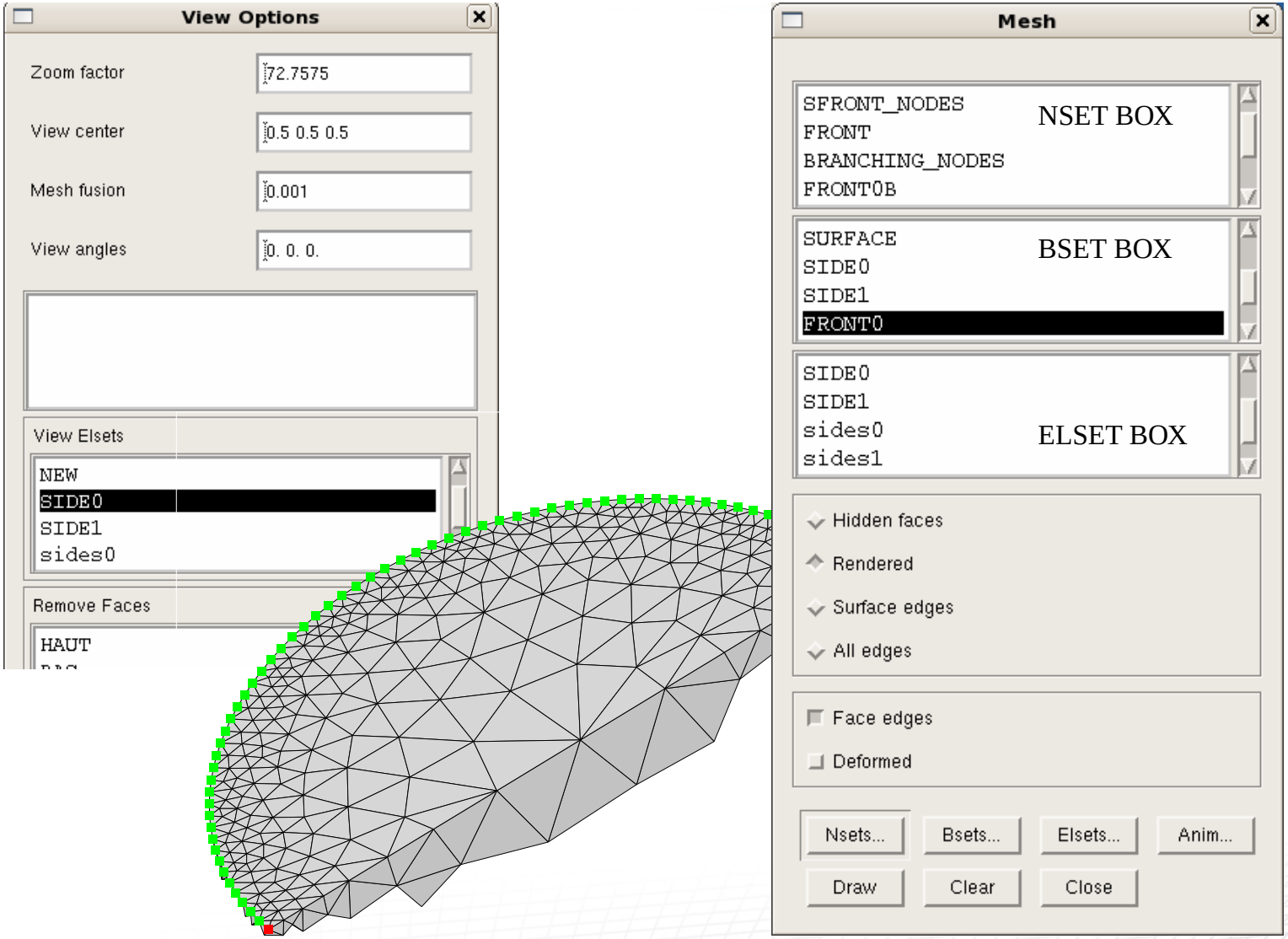


Display of lip side elsets using **Zmaster**

Lip elements elset selection

Selection of elset SIDE0
by Menu/View Options

Plot only those elements
with the Mesh command



SIF calculation

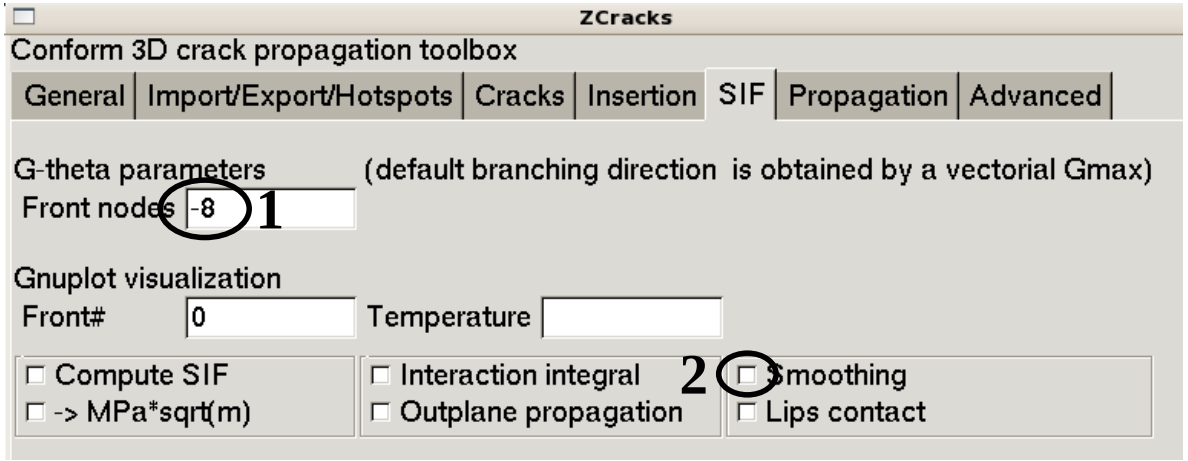
- automatic generation of FE input file to do 1 cycle on the cracked mesh
- cycle computation
- post-processing of FE results to get SIF values:
  - ⋆ energy release rate G (thermo-elasticity, plasticity)
  - ⋆ stress intensity factors K_I, K_{II}, K_{III} (thermo-elasticity)
  - ⋆ bifurcation angle α

Generalities: SIF calculation by the G-theta method

- integral over the whole domain of the potential energy derivative wrt a crack virtual extension field θ
- no integration box definitions as in other methods (Domain Integrals, Crack Tip Contour Integrals)
- can be used with tets unstructured meshes obtained after remeshing (Step 3)
- see theory for some details on the method and validation for results obtained on classical (semi-)analytical problems

Crack front discretization

The front is discretized by spline elements built on *np* control points (see theory) and SIF value (G, K_I ...) are evaluated at those points

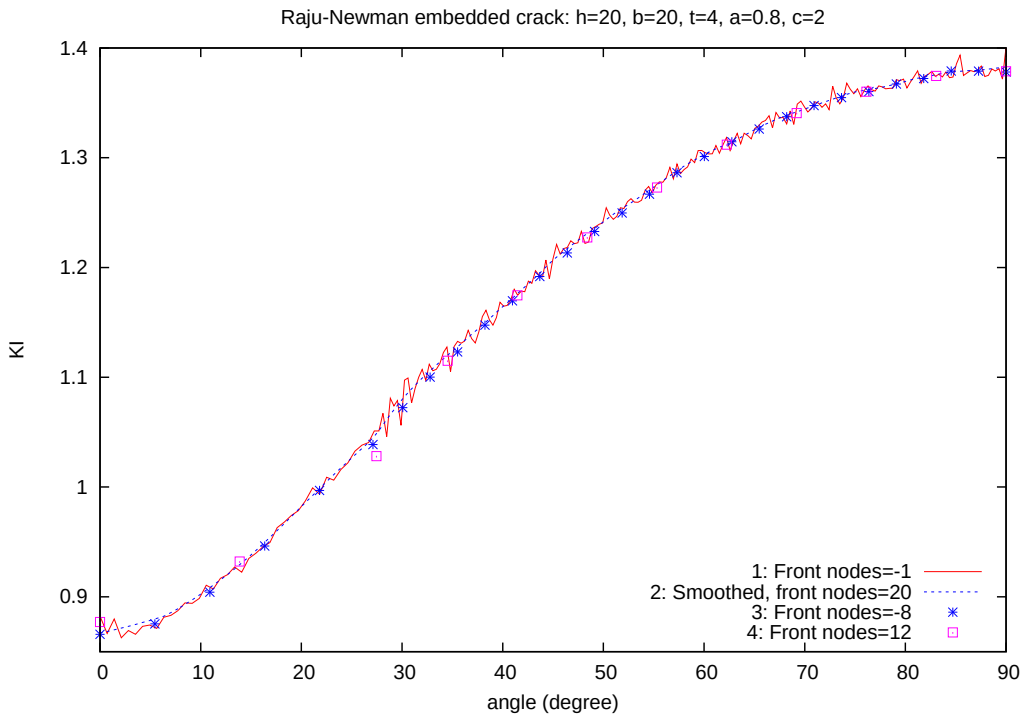


Several options may be used to define those control points:

- specify the exact number of points *np* on the front:
  set a positive *np* value in (1)
- one point each *nno* nodes on the front:
  set a negative ($-nno$) value in (1)
- one point at each node, but smooth the output by the LOESS method using subsets of size *ns*:
  select (2), and give a positive *ns* value in (1)

Crack front discretization: influence on the results

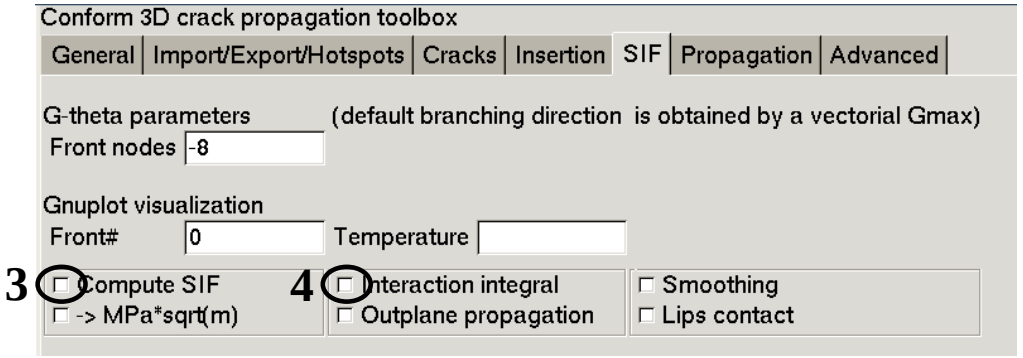
K_I evolutions on the crack-front in the case of an elliptic embedded crack (see Raju-Newman validation).



- oscillations when K_I is computed at each node (218 points) of the FE mesh (curve 1)
- result of the smoothing algorithm (curve 2)
- one point every 8 FE nodes (curve 3, $218/8 = 27$ control points)
- distribution of 12 points equally spaced (curve 4)

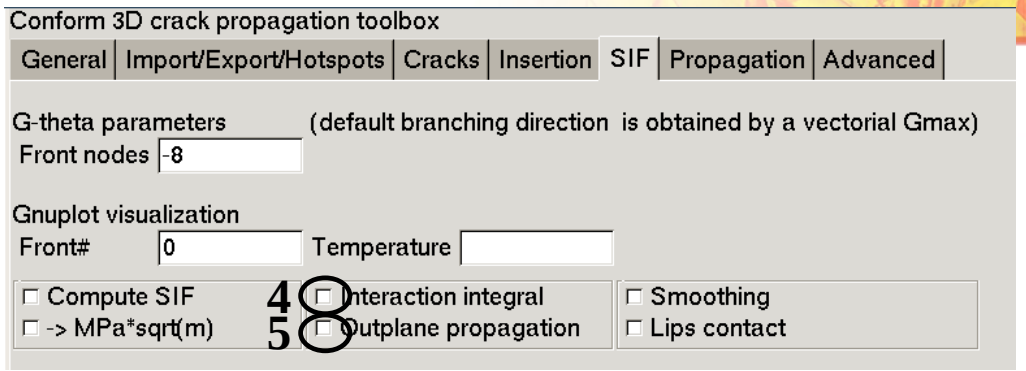
Choice of a particular SIF output quantity

Choice of the SIF output (G or $K_{I,II,II}$) has an influence on the propagation law and the bifurcation angle (Outplane propagation).



- default (i.e. when (3) and (4) are left unchecked) is to compute the strain energy release rate G (the only valid measure when material non-linearities are included).
- when (3) is selected the propagation law should be given in terms of K_I (MPa.sqrt(m)) instead of G (MPa.m)
- in the latter case, when (4) is unchecked, K_I is computed directly from G using the classical plane-strain equation $K_I = \sqrt{\frac{EG}{1-\nu^2}}$ (use with caution at points close to the free surface)
- when both (3) and (4) are selected the G-theta method is used to compute directly the $K_{I,II,II}$ values (see theory)

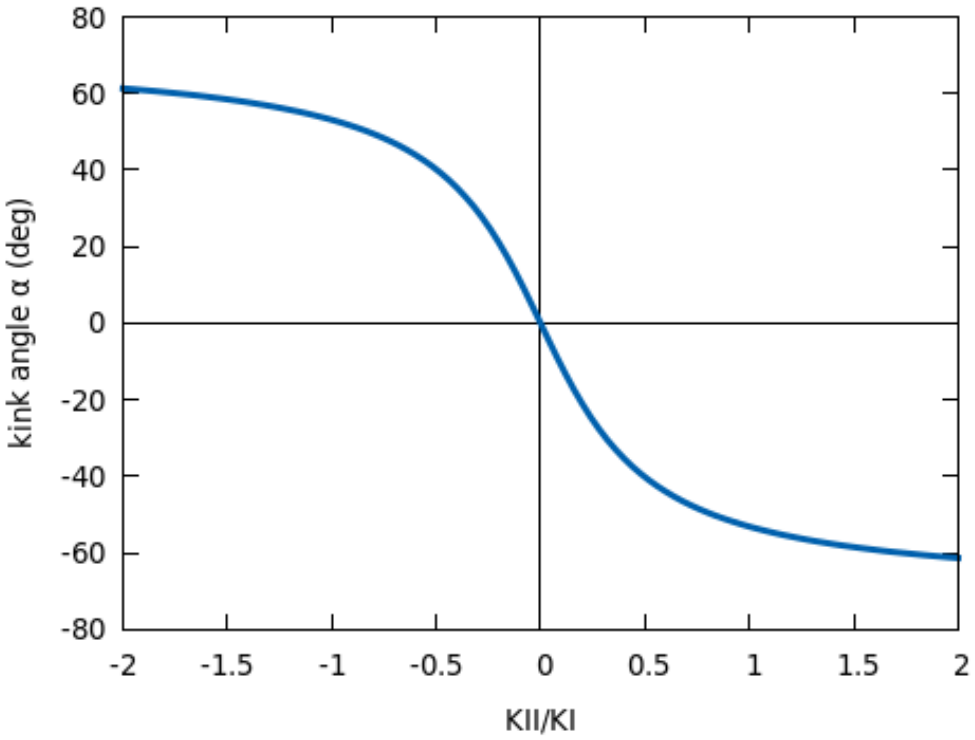
Out-of-plane propagation and bifurcation angle α (1/2)



- when (5) is checked-out a bifurcation angle is evaluated at each point of the crack front, using the fracture mechanics quantities computed by the G-theta method. The criterion used then depends of the choice made in (4):
- if (4) is left unchecked a default G_{max} criterion is used to calculate α. A G_{II} value is then computed in a direction normal to the crack plane, allowing to define the angle as $\alpha = atan(G_{II}/G)$
- when (4) is selected, K_{II} given by the **Interaction integral** method is used to compute a bifurcation angle maximizing the $\sigma_{\theta\theta}$ principal stress according to the following equation:

$$\alpha = 2\ atan\left(\frac{K_I/K_{II} + \sqrt{(K_I/K_{II})^2 + 8}}{4}\right) sign(K_{II})$$

Out-of-plane propagation and bifurcation angle α (2/2)



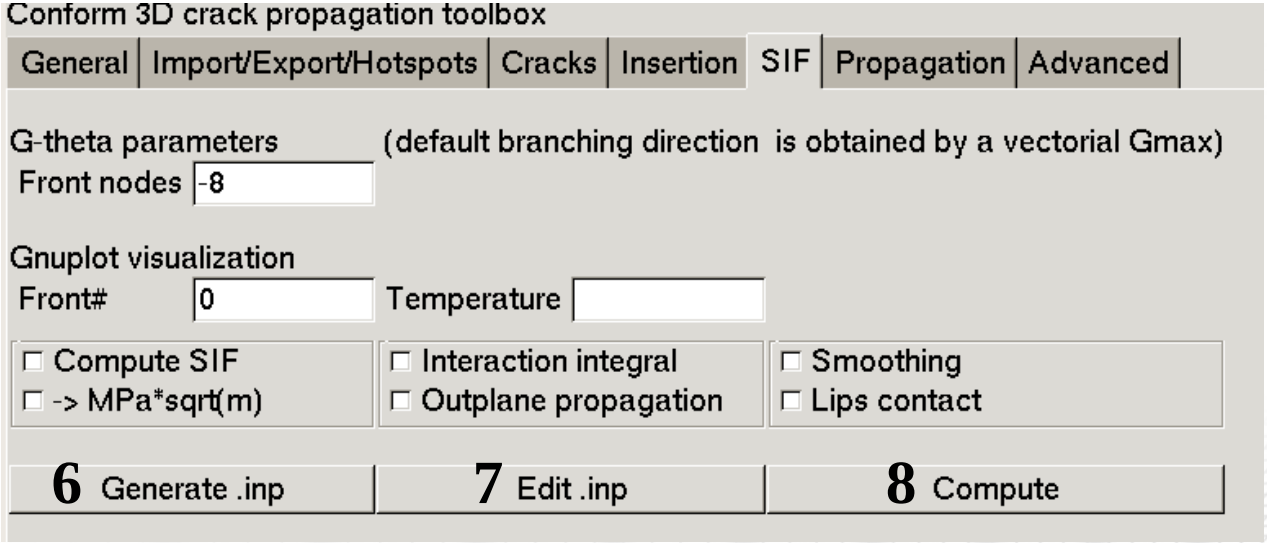
Crack propagation direction α as a function of the K_{II}/K_I ratio according to the $\sigma_{\theta\theta max}$ model.

SIF extraction GUI controls

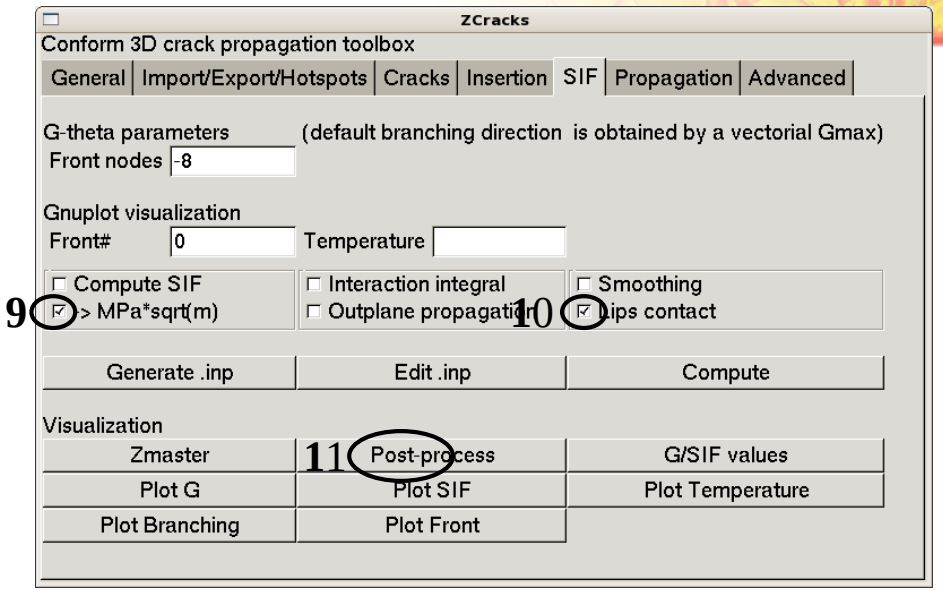
- fill-out various G-theta parameters/options (see SIF generalities)
- generate a computation script by clicking on (6). A file named **cracked_SIF.z7p** is written with **Zprogram** command line interpreter instructions (C/C++ syntax).
- run the script by pressing (8). Note that, once generated, the script can also be launched from a shell window using command:

```
$ Zrun -zp cracked_SIF.z7P
```

- (7) allows to edit the script from the GUI (see reference)



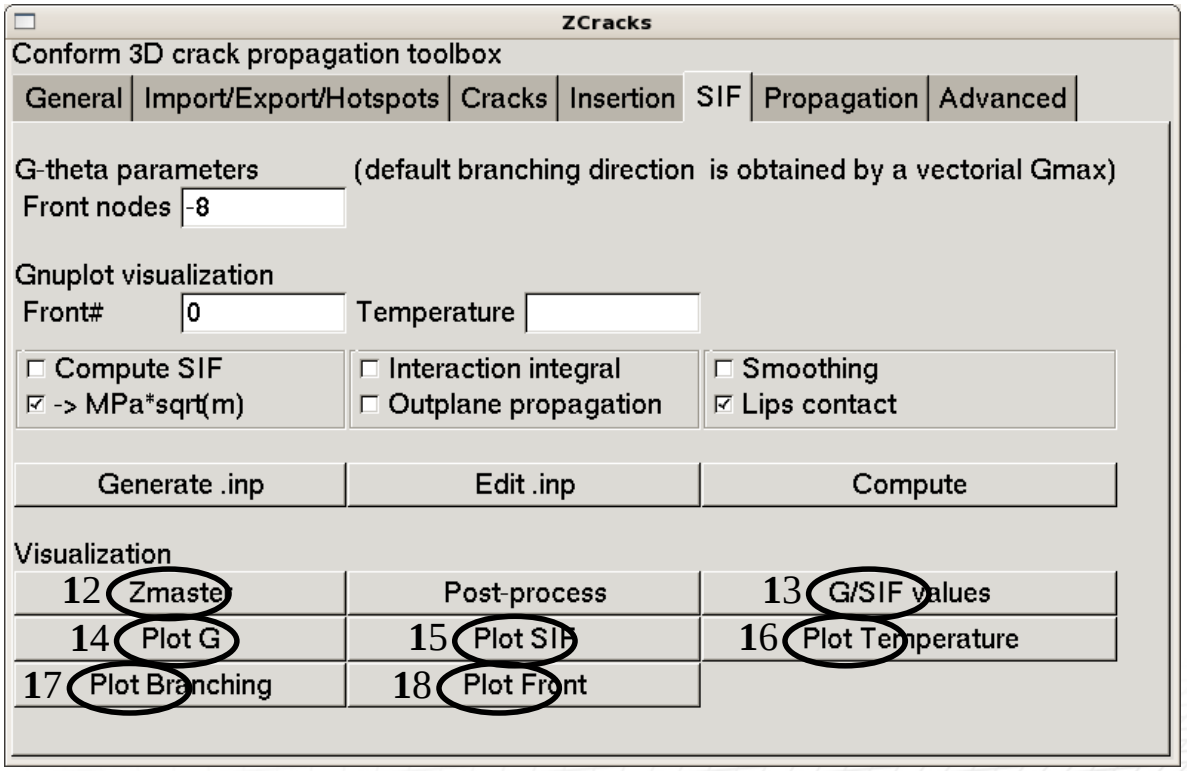
SIF extraction GUI controls



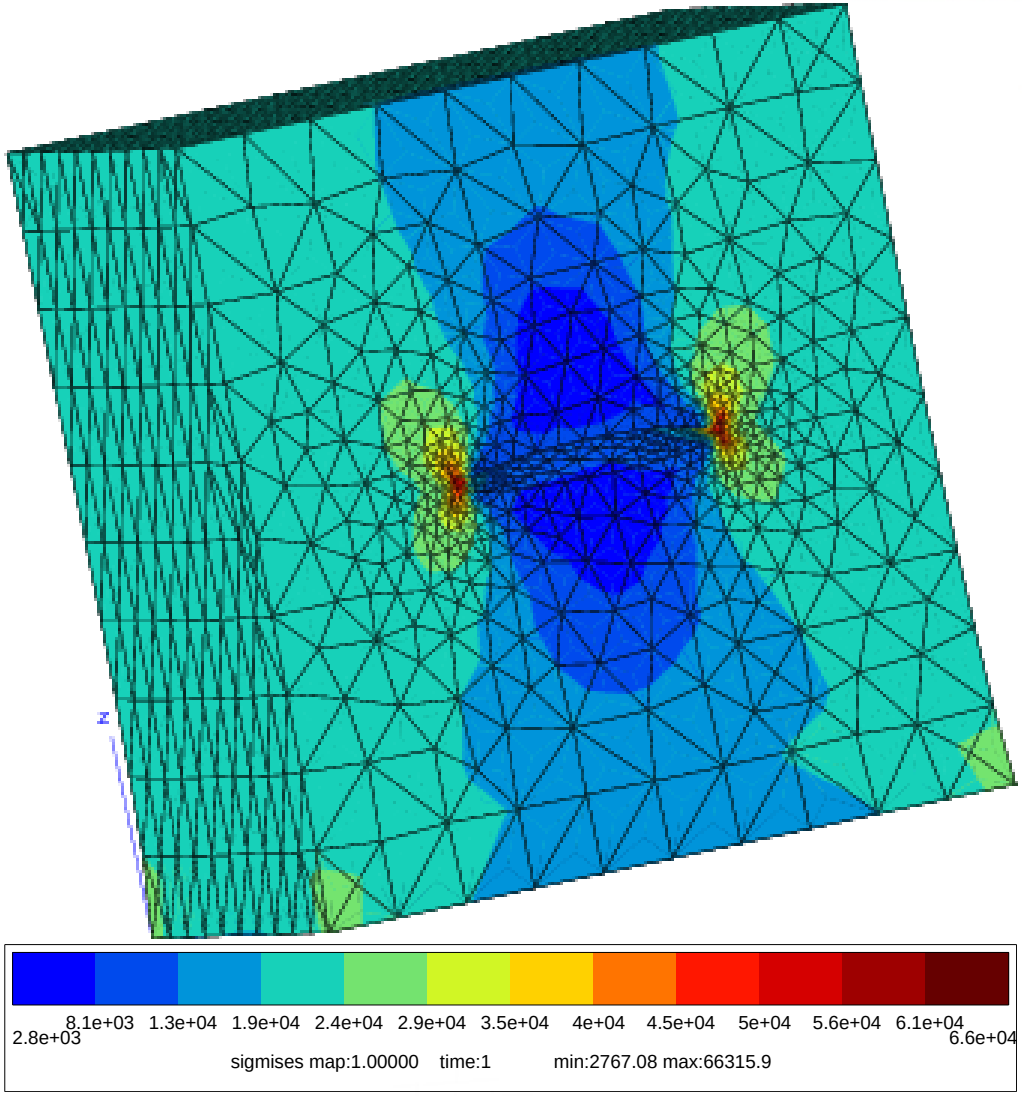
- select (9) to obtain SIF values in MPa.sqrt(m) when units in the FE problem are given in (MPa,mm)
- when (10) is selected, contact between the crack lips is automatically added to the FE input files (useful for negative loading ratios)
- once calculation of a cycle has been performed, (11) can be used to run again the SIF extraction post-processing, for example after having changed the various options (discretization, G/K selection, bifurcation angle criterion).

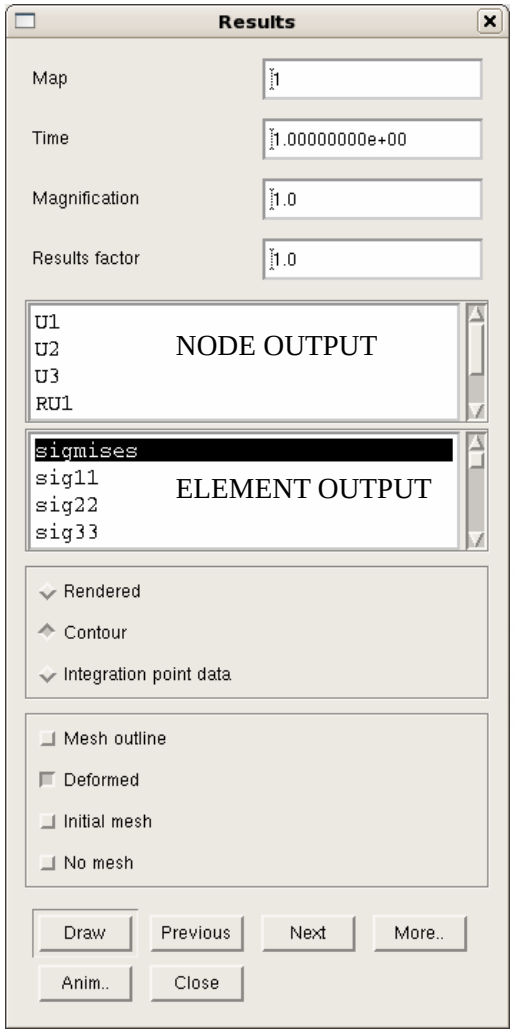
SIF results visualization

- results of 1 cycle applied to the cracked mesh are named **cracked_SIF.***
- can be loaded in the native viewers (**Abaqus CAE** or **Ansys post**) or directly in **Zmaster** using (12)
- SIF post-processing generates ASCII files (see reference) that can be displayed by various plot commands (14) to (18)



Display of cracked results file

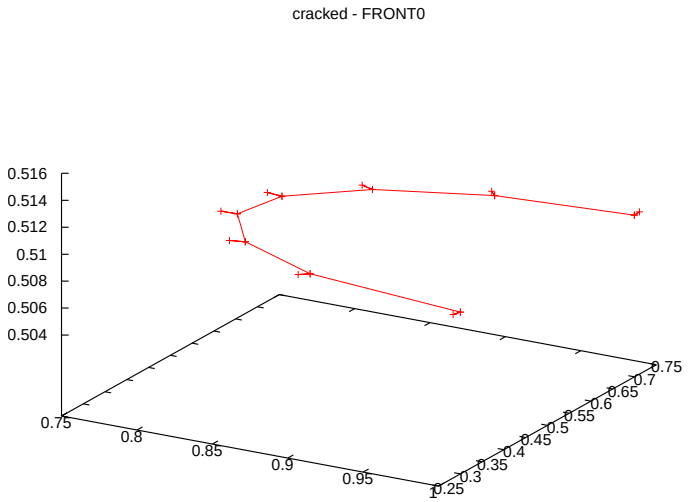




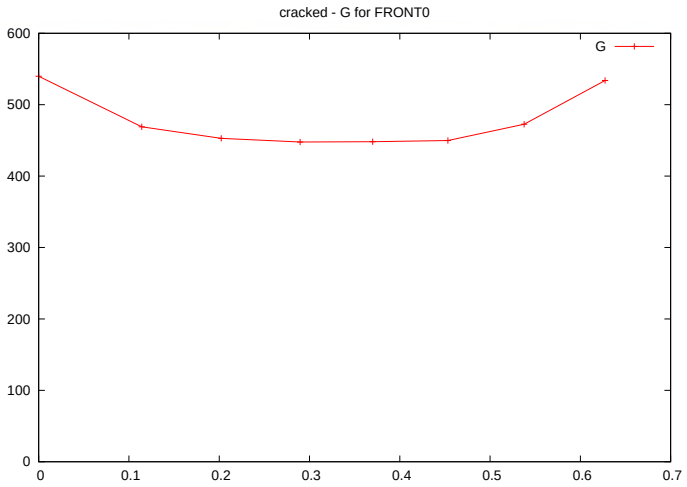
Isocontours with **Zmaster**

SIF plots

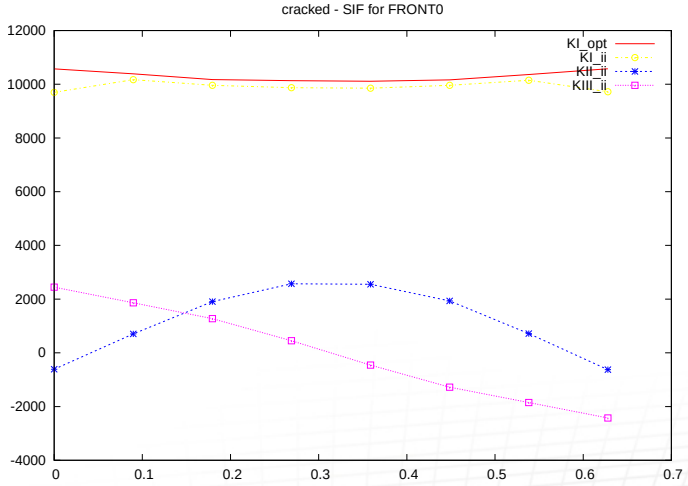
Plot front (18)



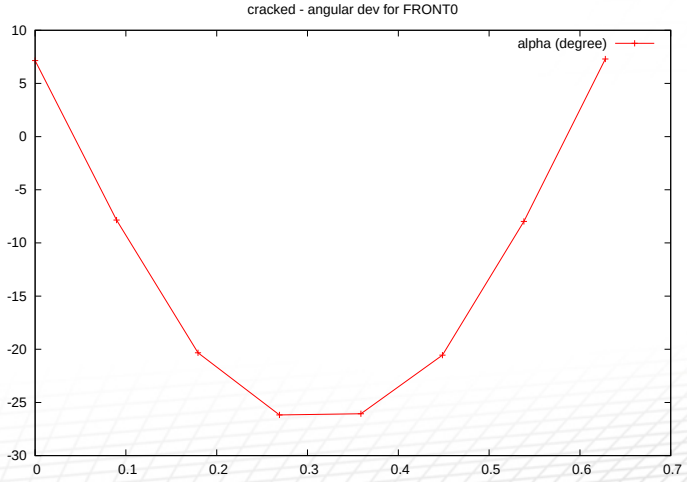
Plot G (14)



Plot SIF (15)



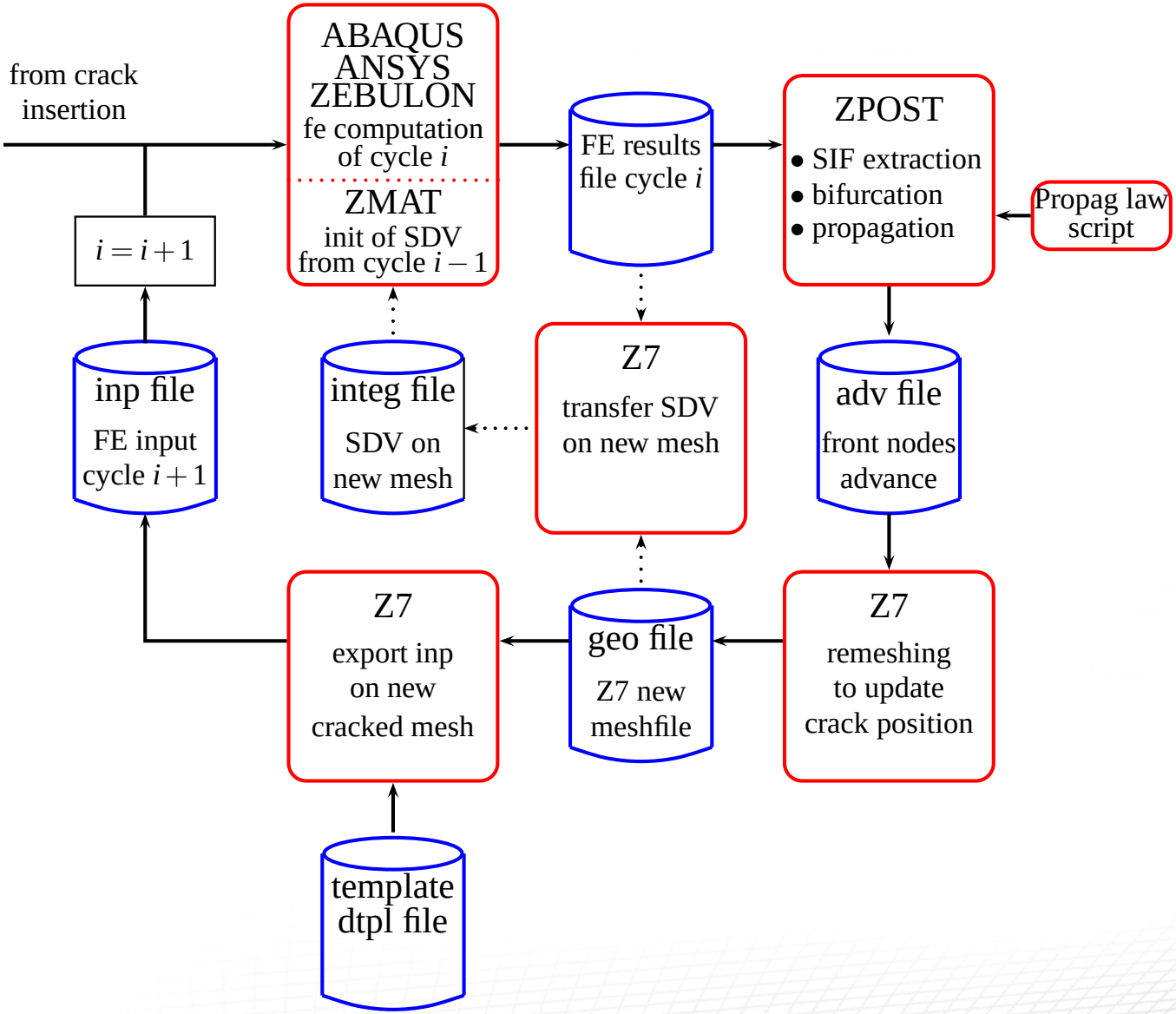
Plot Branching (17)



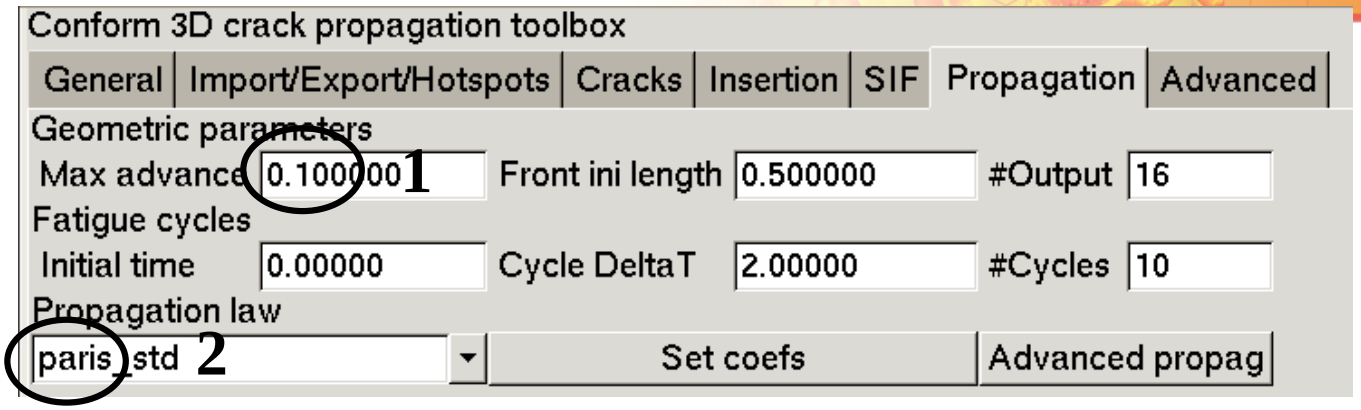
Fatigue crack propagation

Crack propagation loop

for $i = 1$ to n_c computed cycles



Generalities: required advance

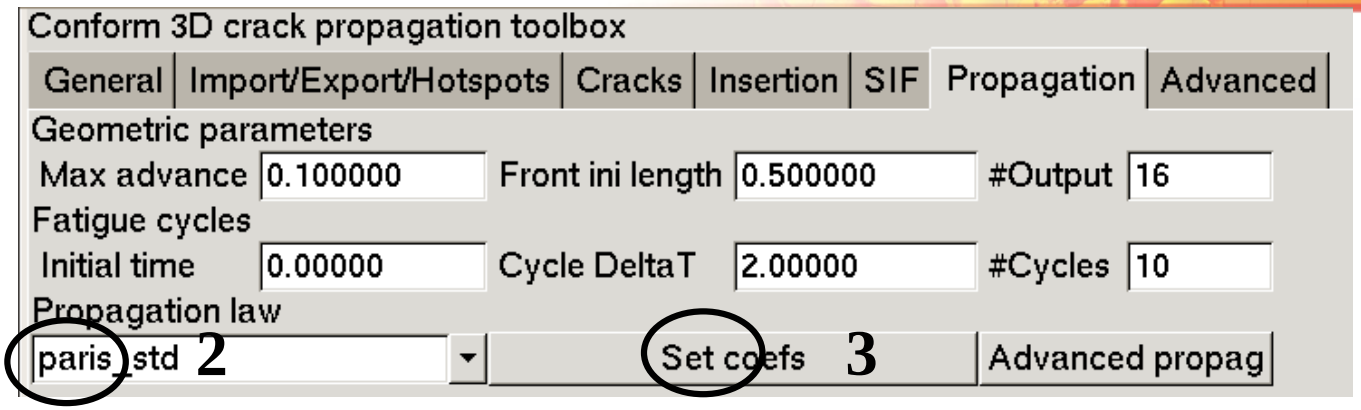


- to guarantee a significant advance at each *computed* cycle, remeshing is controlled by the **Max advance** parameter given in (1)
- the propagation law is then *inverted* to derive the *real* number of cycles (N_r) needed to reach the advance required. In case of a simple Paris law (2), N_r is then evaluated as:

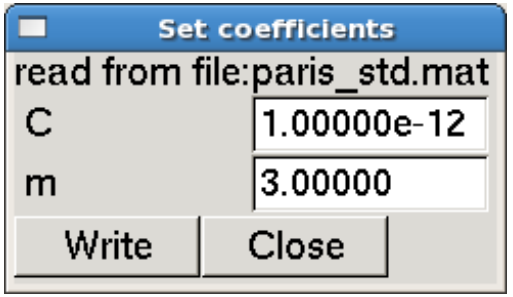
$$N_r = Max\ advance\ .\left[C\ \left(\Delta K_{max}\right)^m\right]^{-1}$$

where ΔK_{max} is the maximum value of the SIF amplitude ΔK (can be G or K_I) calculated over all crack front points

Generalities: propagation laws



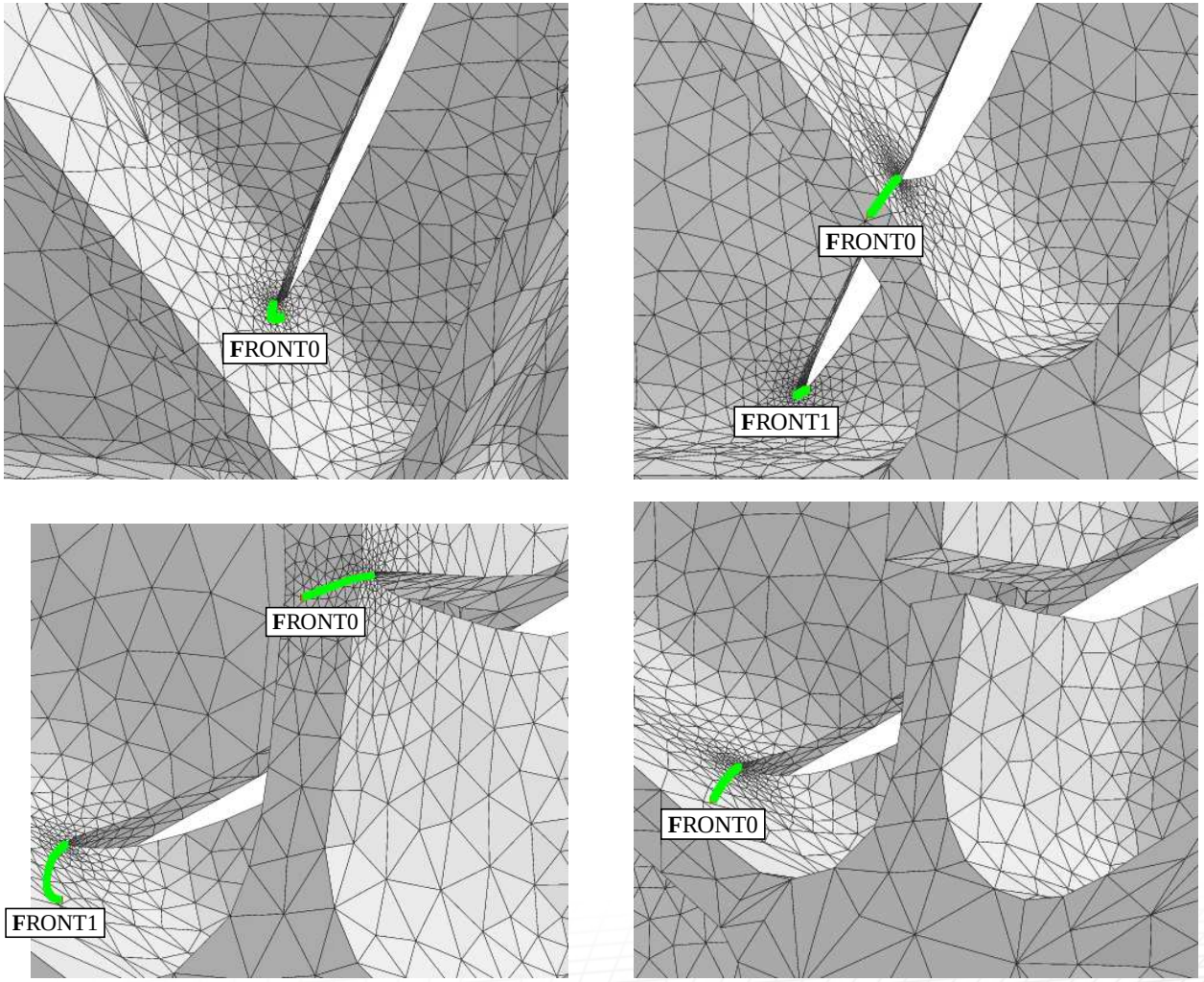
- use the combo box (2) to select a propagation law
- the **Set coefs** button (3) opens a dialog allowing to enter values for coefficients corresponding to the model selected:



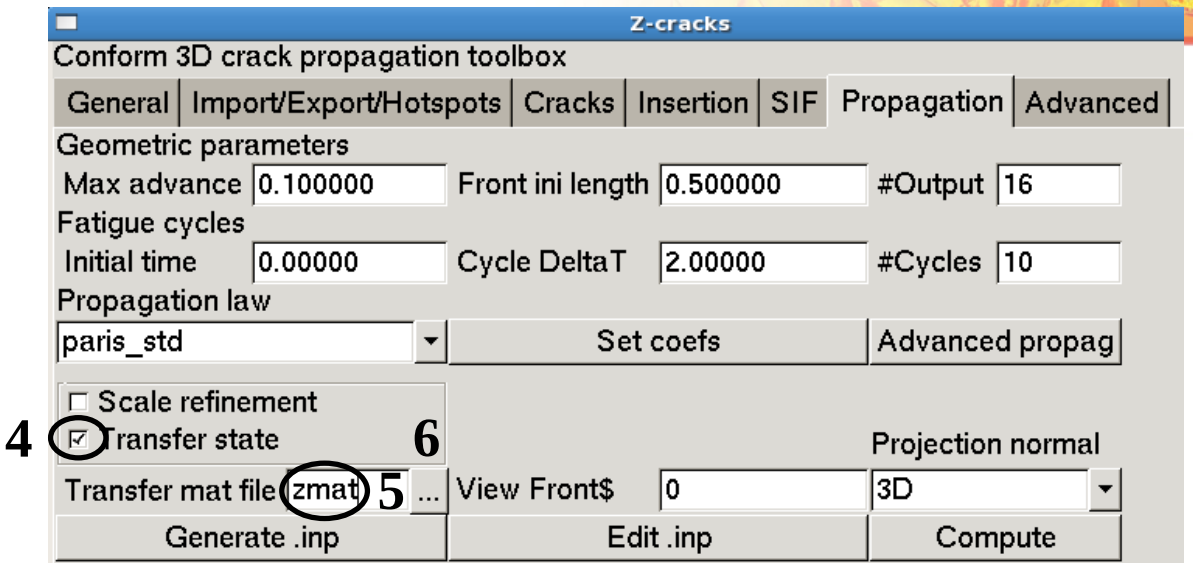
- models available are described in the Propagation laws section

Management of crack multiple fronts

In addition to multiple cracks definition at Insertion, **Z-cracks** handles new crack fronts that are created when crossing obstacles ... and may eventually disappear afterwards

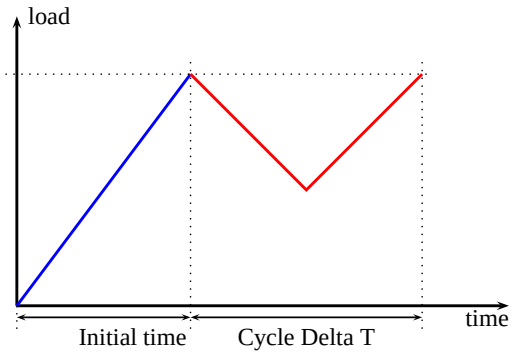


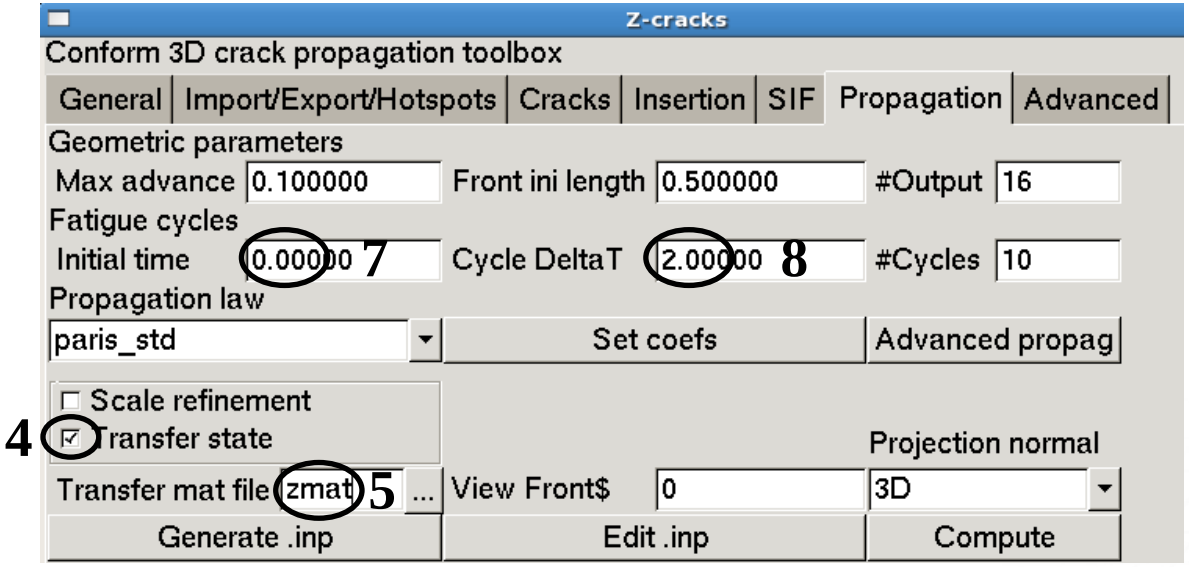
Transfer of plasticity state from the previous cycle



- material internal variables obtained at the end of a cycle can be used to initialize the next cycle (see propagation loop)
- this mechanism is implemented in the **Z-mat** user material subroutines available for **Abaqus** and **Ansys**, such that the use of **Z-mat** in the FE calculation is mandatory to allow this advanced option
- to activate the SDV transfer capability select (4) and give the name of the **Z-mat** file in (5) (or use the browse button (6) to select one)

Definition of a preloading step before cycling





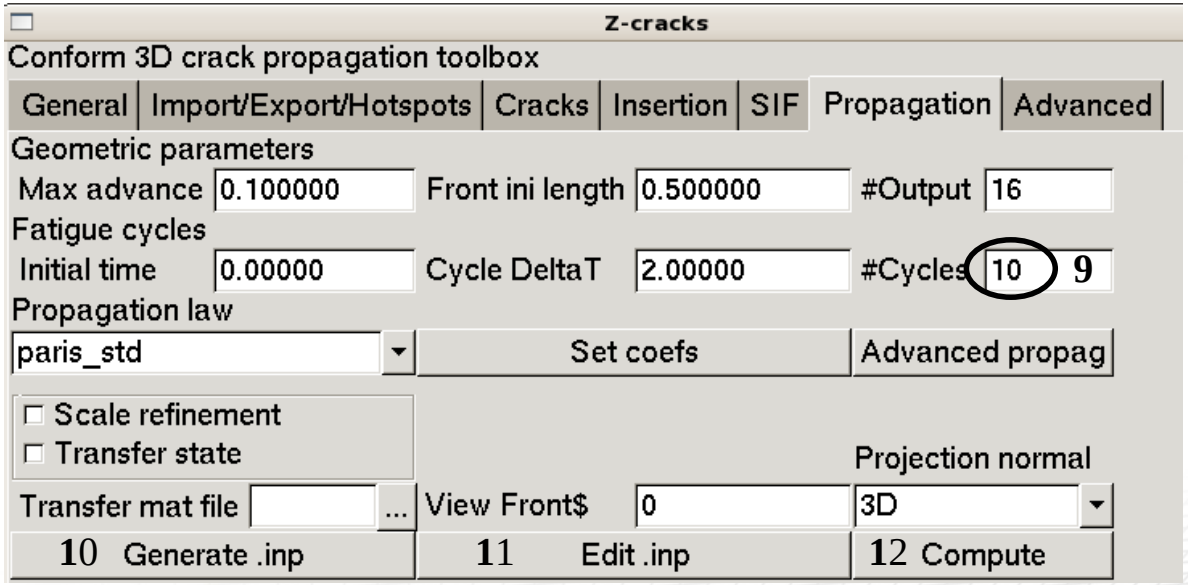
- a preloading step can be defined by setting **Initial time** to a non-zero value in (7)
- corresponding results are then skipped to compute ΔK
- in addition, if the **Transfer state** option (4) is activated, preloading increments are automatically removed from the FE input decks generated at all cycles except the first one

Crack propagation GUI controls

- enter the number of cycles to compute in (9)
- generate a computation script by clicking on (10). A file named **cracked_PROPAG.z7p** is written with **Zprogram** command line interpreter instructions (C/C++ syntax).
- run the script by pressing (12). Note that, once generated, the script can be launched from a shell window using command:

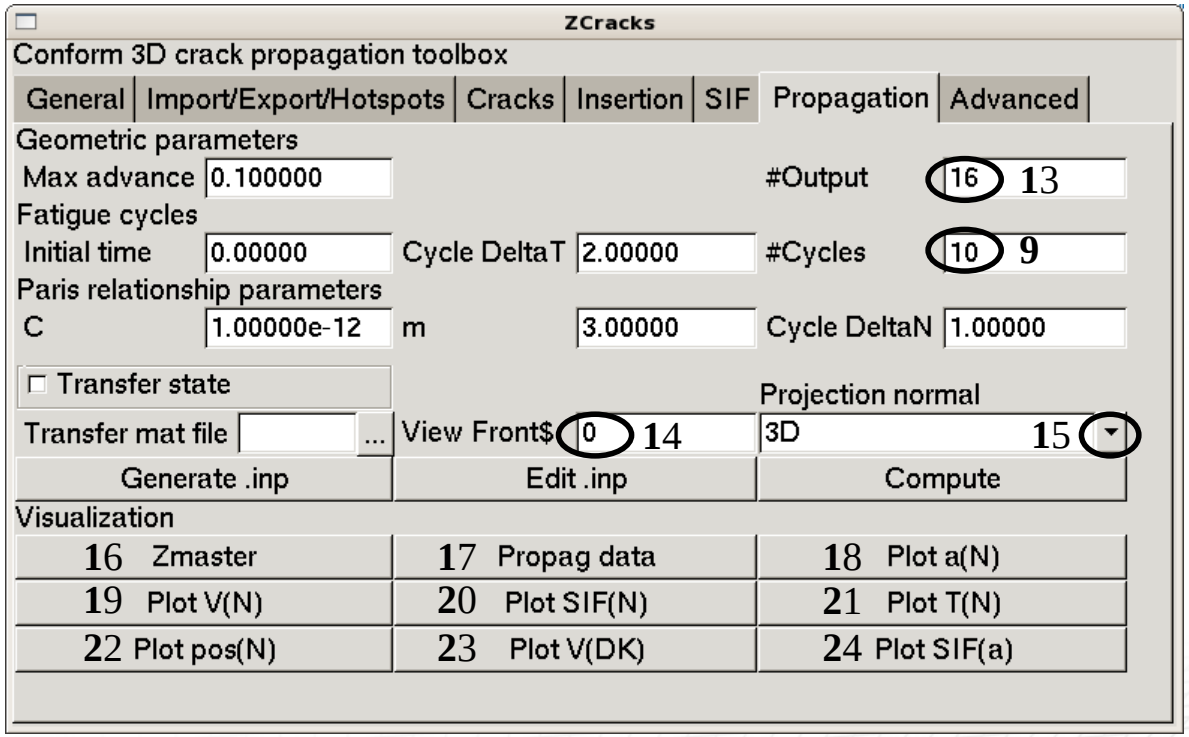
```
$ Zrun -zp cracked_PROPAG.z7P
```

- (11) allows to edit the script from the GUI (see reference)



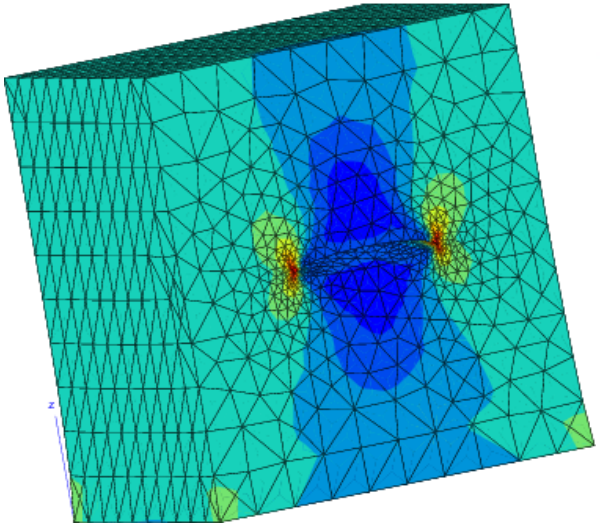
Propagation results visualization

- propagation FE results files are named **cracked_PROPAG*i*.*** with *i* the cycle number
- those files can be loaded in native viewers or in **Zmaster** using (12). In this case the cycle loaded is given by (9)
- a **cracked_PROPAG.zck** file is maintained during cycles in order to draw SIF evolutions with commands (18)-(24) (see reference)

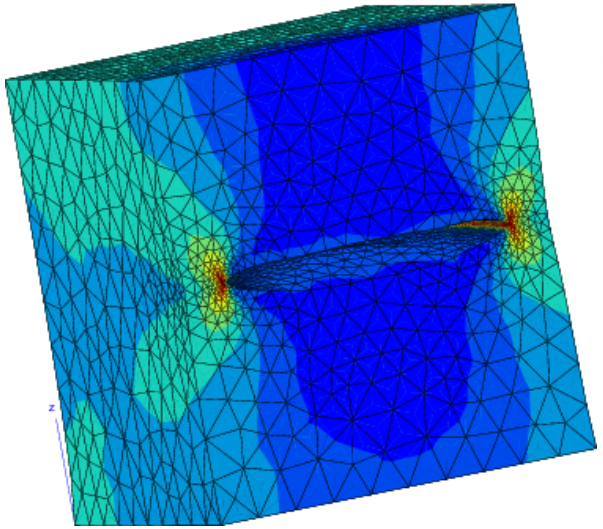


Isocontours on cyclic FE results files

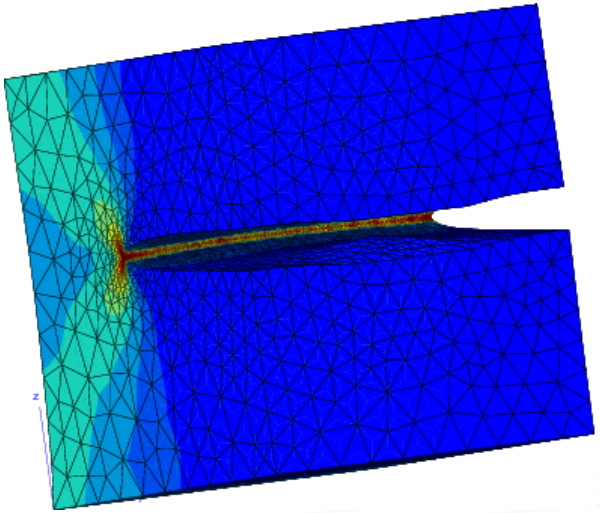
cracked_PROPAG1



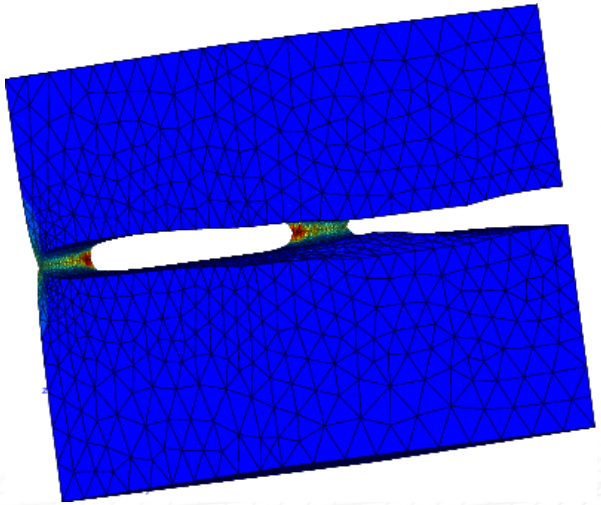
cracked_PROPAG5



cracked_PROPAG15

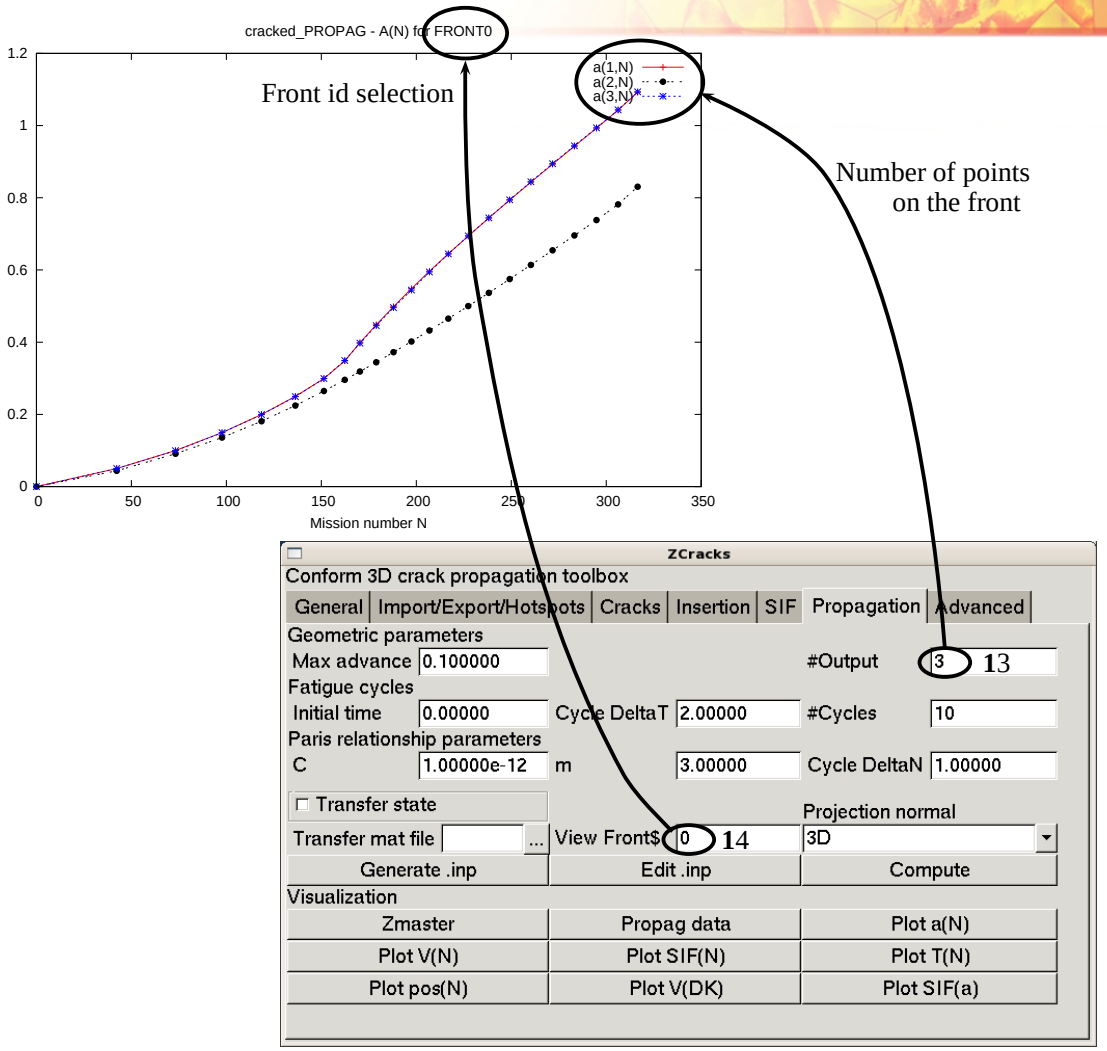


cracked_PROPAG23



Zmaster isocontours drawn from the GUI

Propagation plots definition

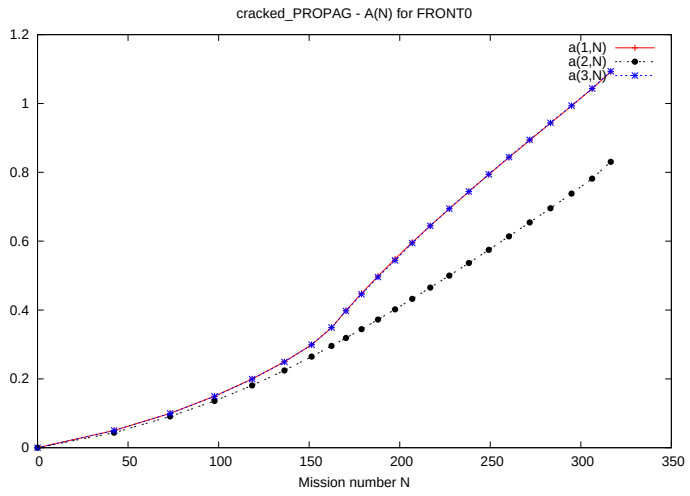


- caution, changing the number of points (13) between restarts may confuse plot interpretation

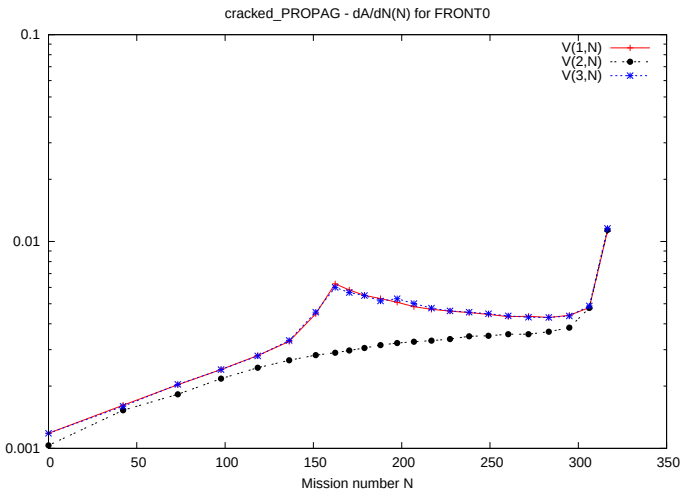
Propagation plots

x axis: *real* number of cycles N_r (see required advance)

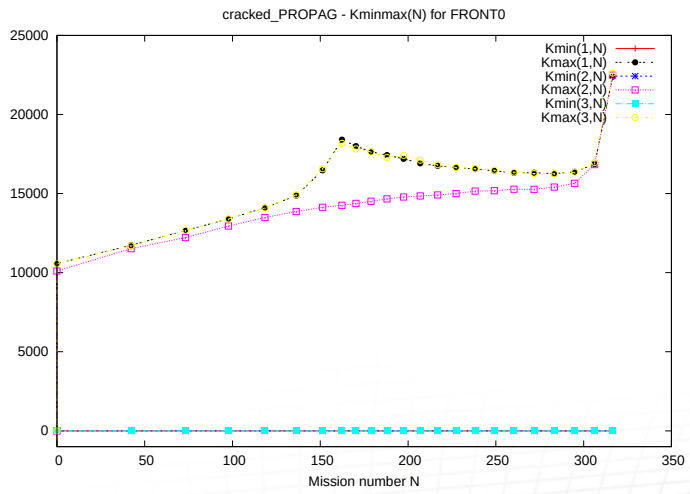
Plot a(N) (18)



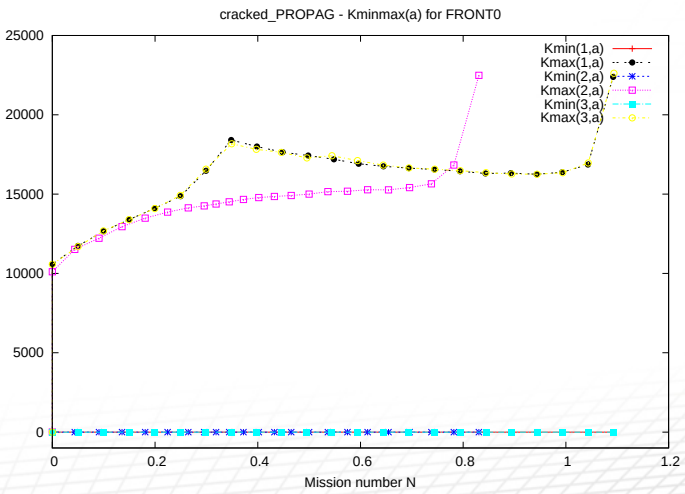
Plot V(N) (19)



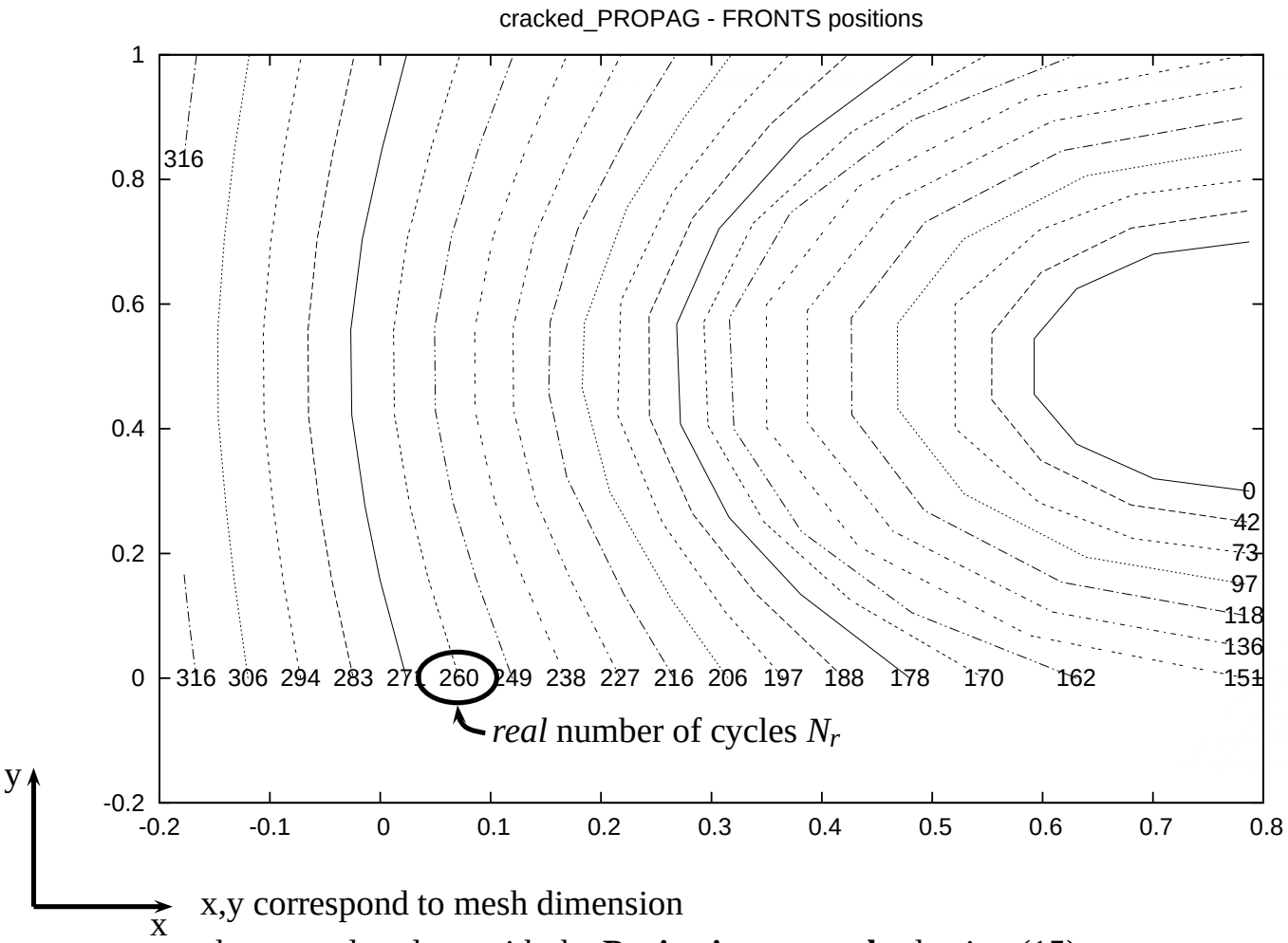
Plot SIF(N) (20)



Plot SIF(a) (24)



Front advance plots

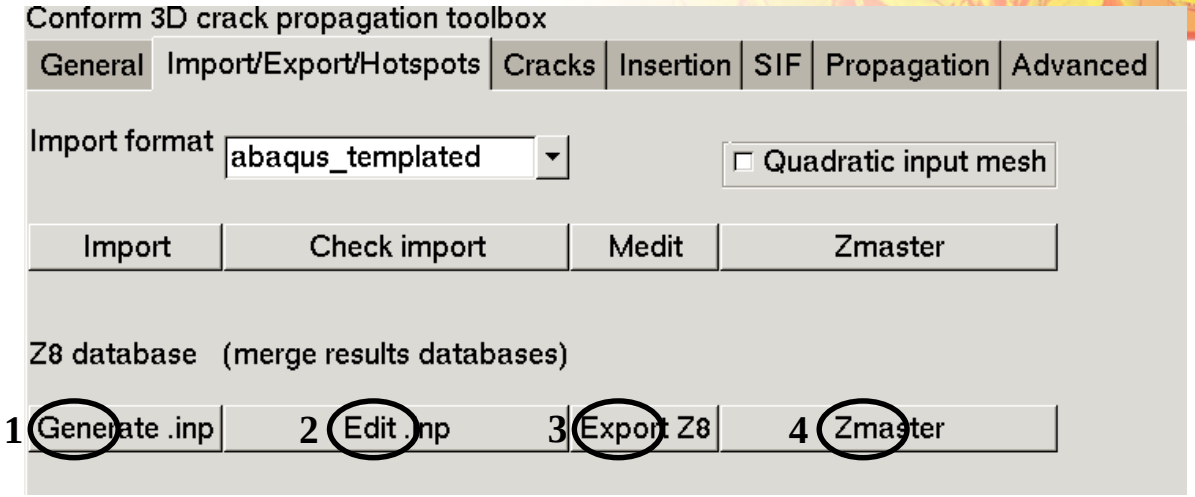


x,y correspond to mesh dimension

choose a plot plane with the **Projection normal** selection (15)

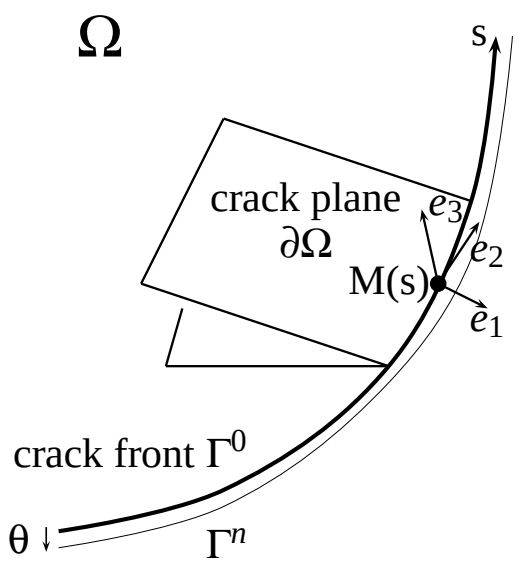
- plot all fronts during propagation cycles

Export to a Z8 database



- the **Export Z8** command (3) of the **Import/Export Tab** allows to collect separate cyclic results file (**cracked_PROPAG1**, **cracked_PROPAG2**...) in a single **Z8** database, an output format that supports remeshing
- this database can then be loaded in **Zmaster** (4) to draw animations of crack propagation results
- (1) generates the (default) input command for **Export Z8**. To cut down disk storage, editing the input with (2) allows to add ***frequency** options, and select which increments/cycles should be stored in the database

G-theta method theory (1/2)



Transformations F^n of domain Ω to Ω^n due purely to crack propagation

$$F^n \ : \ M \rightarrow M \ + \ \eta \, \theta(M)$$

θ : crack extension virtual field that modifies only the position of the crack front Γ_0

$\theta \in \Theta \ = \ \{\mu \ such \ that \ \mu.e_3 = 0\}$ (tangent to the crack plane)

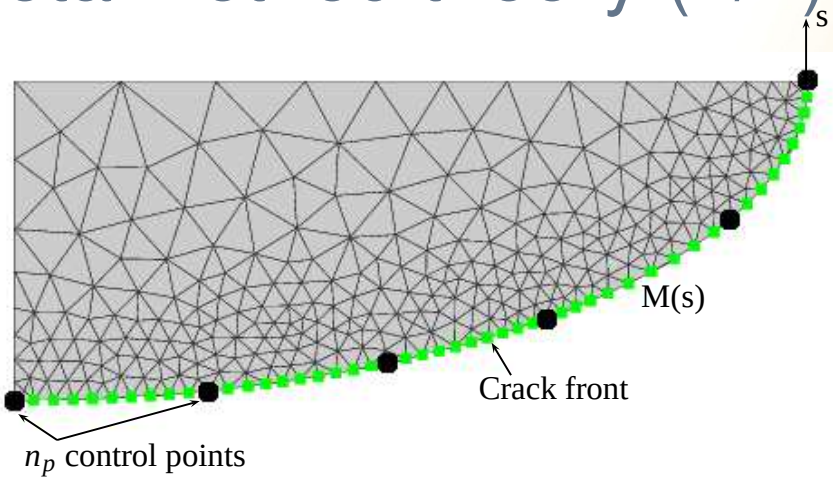
The stress energy release rate $G(\theta)$ for crack extension θ is given by the Lagrangian derivative of potential energy W:

$$G(\theta) \ = \ -\frac{\partial W}{\partial \eta}$$

For thermo-elasticity the right-hand side reduces to :

$$\frac{\partial W}{\partial \eta} \ = \ \int_\Omega \left[\frac{1}{2} \left(\sigma : \left(\epsilon - \epsilon^{th} \right) \right) \nabla \theta \ - \ \sigma : \left(\nabla u \nabla \theta \right) \right] d\Omega$$

G-theta method theory (2/2)



The left-hand side is obtained by integration on the crack front Γ^0:

$$G(\theta) \;=\; \int_{\Gamma_0} G(s)\;\theta(s)\;e_1(s)\;ds$$

Discretization by n_p control points with shape functions $N_j(s)$:

$$G(s) \;=\; \sum_{j=1}^{n_p} G_j\;N_j(s) \quad , \quad G(\theta) \;=\; \sum_{j=1}^{n_p} G_j \int_{\Gamma_0} \theta(s)\;N_j(s)\;ds\;,\;\forall\theta\in\Theta$$

For θ^i $(i=1,n_p)$ virtual fields $\theta^i(s).e_1(s) \;=\; N_i(s)$:

$$\begin{aligned} G(\theta^i) \quad &= \quad \sum_{j=1}^{n_p} G_j \int_{\Gamma_0} N_i(s)\;N_j(s)ds \qquad i=1,n_p \\ &= \quad \int_{\Omega} \left[\frac{1}{2}\left(\sigma:\left(\epsilon-\epsilon^{th}\right)\right)\nabla\theta^i \;-\; \sigma:\left(\nabla u\nabla\theta^i\right)\right] d\Omega \end{aligned}$$

a system with n_p unknowns G_j $(j=1,n_p)$

SIF extraction using the G-theta method (1/2)

Introducing the same discretization of a G^v value on the crack front Γ^0 as the one defined for G previously:

$$G^v(s) \;=\; \sum_{j=1}^{n_p} G_j^v \; N_j(s)$$

it is possible to evaluate an interaction integral using any virtual displacement field v in combination of the FEA obtained displacement u in the following formulation:

$$\sum_{j=1}^{n_p} G_j^v \int_{\Gamma_0} N_i(s)\; N_j(s) ds \;=$$

$$\int_{\Omega} \left[\frac{1}{2} \left(\sigma(u) : (\epsilon(v)) \right) \nabla\theta^i \;-\; \sigma(v) : \left(\nabla U . \nabla\theta^i \right) \right] d\Omega$$

SIF extraction using the G-theta method (2/2)

Introducing, in the previous equation, any pure mode I, II or III Westergaard displacement solutions $v^{I,II,III}$, defined in the crack front vicinity, allows to compute associated $G^{v,I,II,III}$ values. The following Irwin formula for a given isotropic linear elastic behavior, leads to each associated SIF $K_j^{I,II,III}$ along the front discretization:

$$\sum_{j=1}^{n_p} G_j^{v,I,II,III} \int_{\Gamma_0} N_i(s)\, N_j(s) ds \; =$$

$$\sum_{j=1}^{n_p} \frac{1-\nu^2}{E} \left(K_j^I\, K_j^{v,I} + K_j^{II}\, K_j^{v,II} \right) + \frac{1}{2\mu} K_j^{III}\, K_j^{v,III} \int_{\Gamma_0} N_i(s)\, N_j(s) ds$$

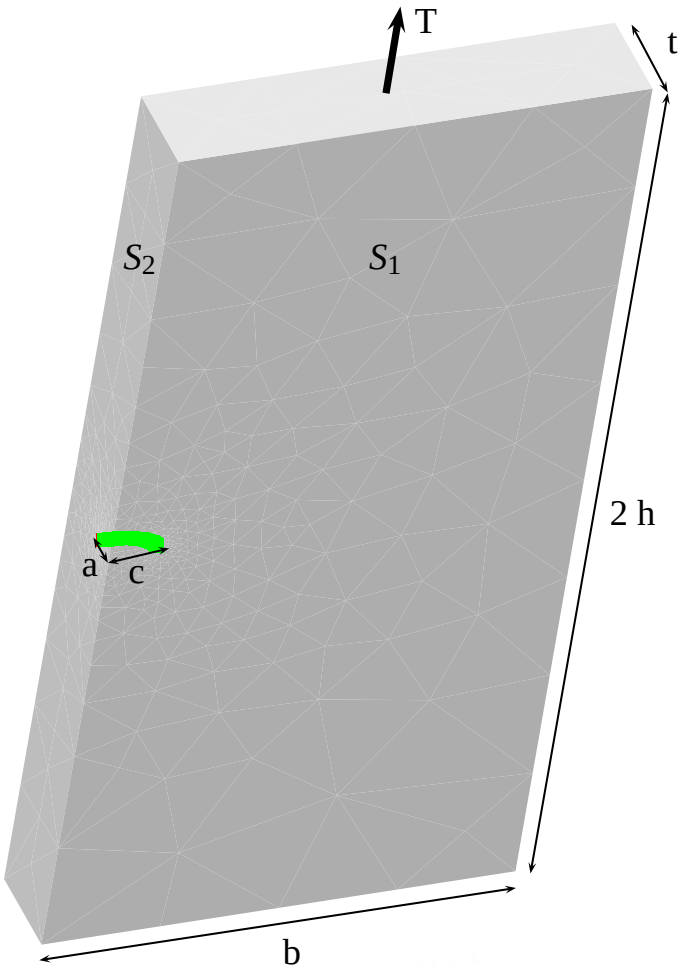
G-theta formulation in plasticity

A domain invariant integral can be computed in case of elastic-plastic material behavior. Such an approach evaluates the potential energy that would be released during an infinitesimal advance of the crack front considering a pure elastic evolution of the material submitted to a fixed inelastic residual stress field (generated by thermo-visco-plasticity). The integrated quantity G_p is defined by the following equation:

$$G_p(\theta^i) = \int_\Omega \left[\frac{1}{2} \left(\sigma : (\epsilon - \epsilon^{ae}) \right) \nabla \theta^i - \sigma : \left(\nabla u \nabla \theta^i \right) - \sigma : \nabla \epsilon^{ae} . \theta^i \right] d\Omega$$

with: $\epsilon^{ae} = \epsilon - \epsilon^e = \epsilon^{th} + \epsilon^p$

Validation: Comparison with the Raju-Newman solution



- embedded crack: symmetry on the S_1, S_2 faces
- surface crack: symmetry on the S_2 face
- corner crack: no symmetry bcs

Comparison with Z-cracks (n_p points on the crack front)

- max difference:

$$e_{max} = \max_{i=1,n_p} \left\{ \frac{|K_{Ij} - K_{Ij}^{raju}|}{K_{Ij}^{raju}} \right\}$$

- average: $\bar{e} = \frac{e_{max}}{n_p}$

Embedded elliptical crack

$$h = 20\ ,\ b = 20\ ,\ t = 4\ ,\ c = 2\ ,\ a = 0.8$$

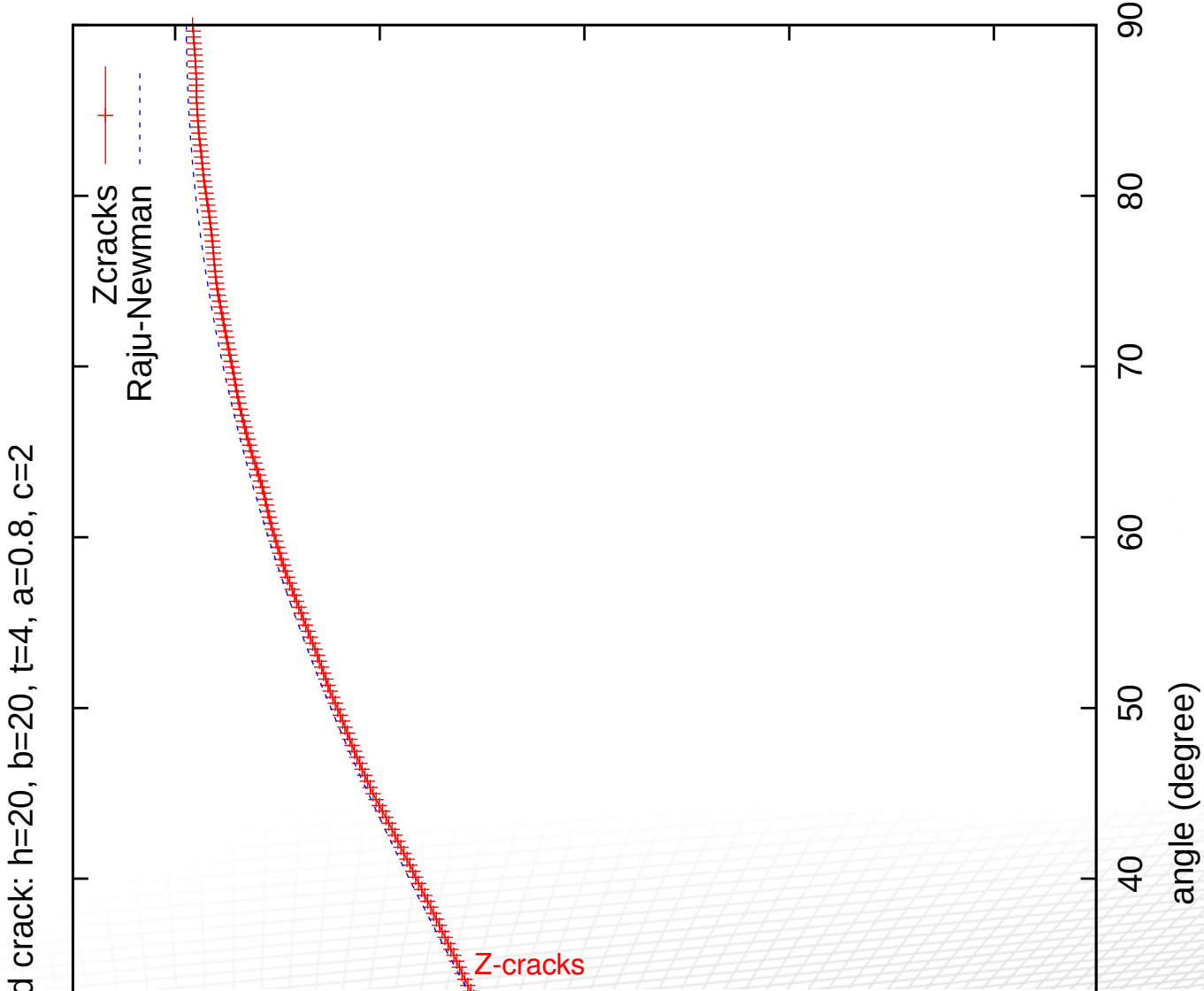
quadratic mesh						linear mesh				
Min s	Grad	Qn	#dof	e_{max}%	$\bar{e}$ %	Min s	Grad	#dof	e_{max}%	$\bar{e}$ %
5e-3	1.8	yes	480216	1.04	0.52	1e-2	1.8	32883	8.18	4.91
1e-2	1.8	yes	245355	3.18	0.67	1e-2	1.2	152907	4.46	2.40
2e-2	1.8	yes	125187	3.43	0.89	5e-3	1.2	313950	2.89	2.21
4e-2	1.8	yes	63921	4.00	1.64	2e-3	1.2	737631	2.67	2.31
1e-2	2.5	yes	166989	3.08	0.91					
1e-2	1.2	yes	1196433	???	???					
1e-2	1.8	no	245355	2.97	0.67					

- quadratic eles with large gradation value (≈ 2) offers the best cost/accuracy
- small gradation values (< 1.5) needed for linear meshes
- quarter node option significantly improves accuracy for quadratic meshes (caution: use only with linear elastic material)
- difference with Raju-Newman solution: 1% with quadratic and 3% with linear eles

Embedded elliptical crack

results with a quadratic mesh:

- `min_size`=0.01 , `gradation`=1.8
- e_{max}=1.22% , $\bar{e}$=0.54%



Surface elliptical crack

$$h = 20\ ,\ b = 20\ ,\ t = 4\ ,\ c = 2\ ,\ a = 0.8$$

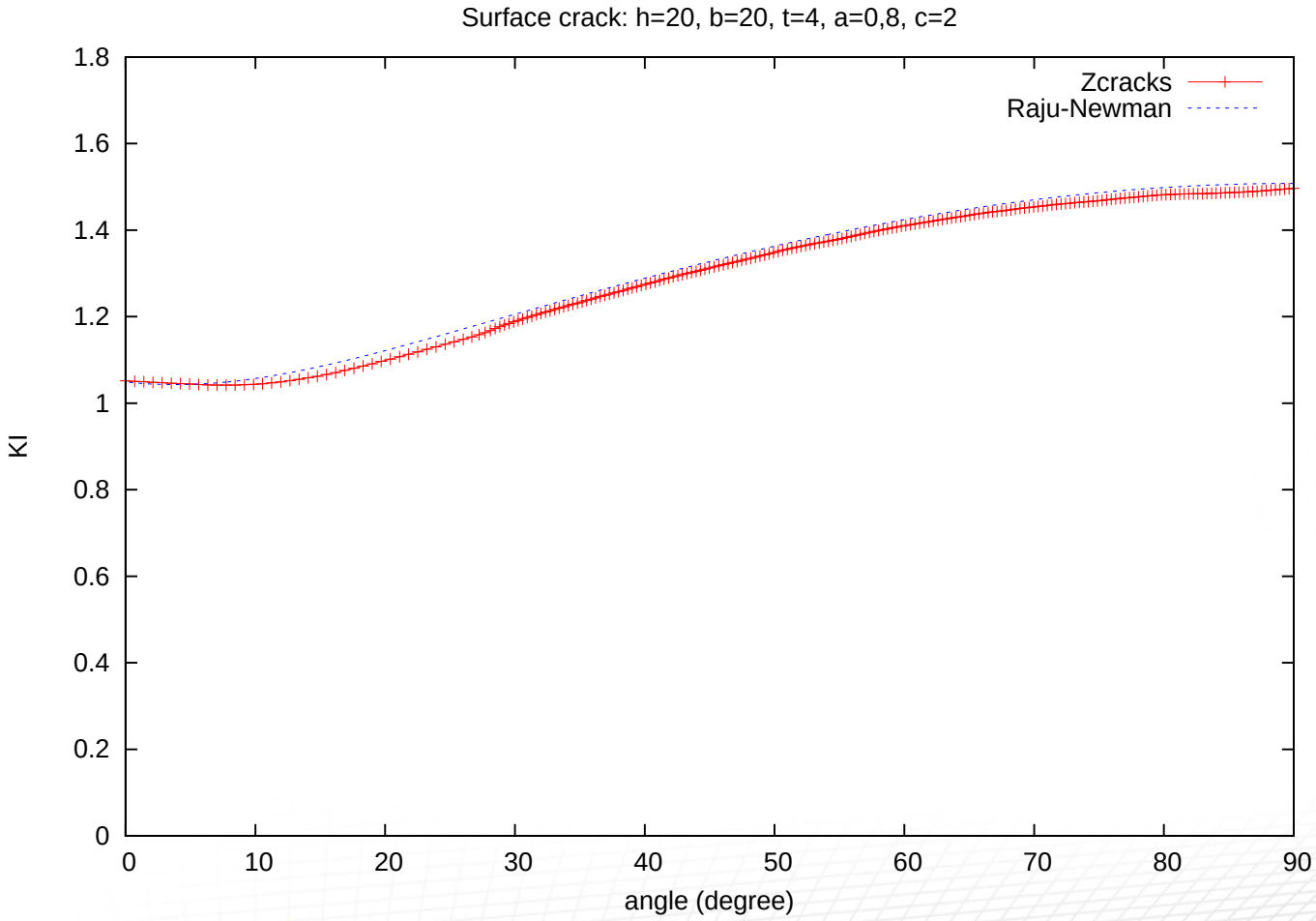
quadratic mesh						linear mesh				
Min s	Grad	Qn	#dof	e_{max}%	$\bar{e}$ %	Min s	Grad	#dof	e_{max}%	$\bar{e}$ %
5e-3	1.8	yes	480216	3.05	2.05	1e-2	1.8	32883	10.06	7.76
1e-2	1.8	yes	245355	3.40	2.19	1e-2	1.2	152907	5.89	4.37
2e-2	1.8	yes	125187	5.48	2.42	5e-3	1.2	313950	5.32	4.14
4e-2	1.8	yes	63921	6.29	3.10	2e-3	1.2	737631	5.31	4.26
1e-2	2.5	yes	166989	3.76	2.43					
1e-2	1.2	yes	1196433	???	???					

- same kind of results as in the embedded case
- difference with Raju-Newman solution: 2% with quadratic and 4% with linear eles

Surface elliptical crack

results with a quadratic mesh:

- `min_size`=0.01 , `gradation`=1.8
- e_{max}=2% , $\bar{e}$=1.19%



Corner quarter-elliptical crack

$$h = 20\ ,\ b = 20\ ,\ t = 4\ ,\ c = 2\ ,\ a = 0.8$$

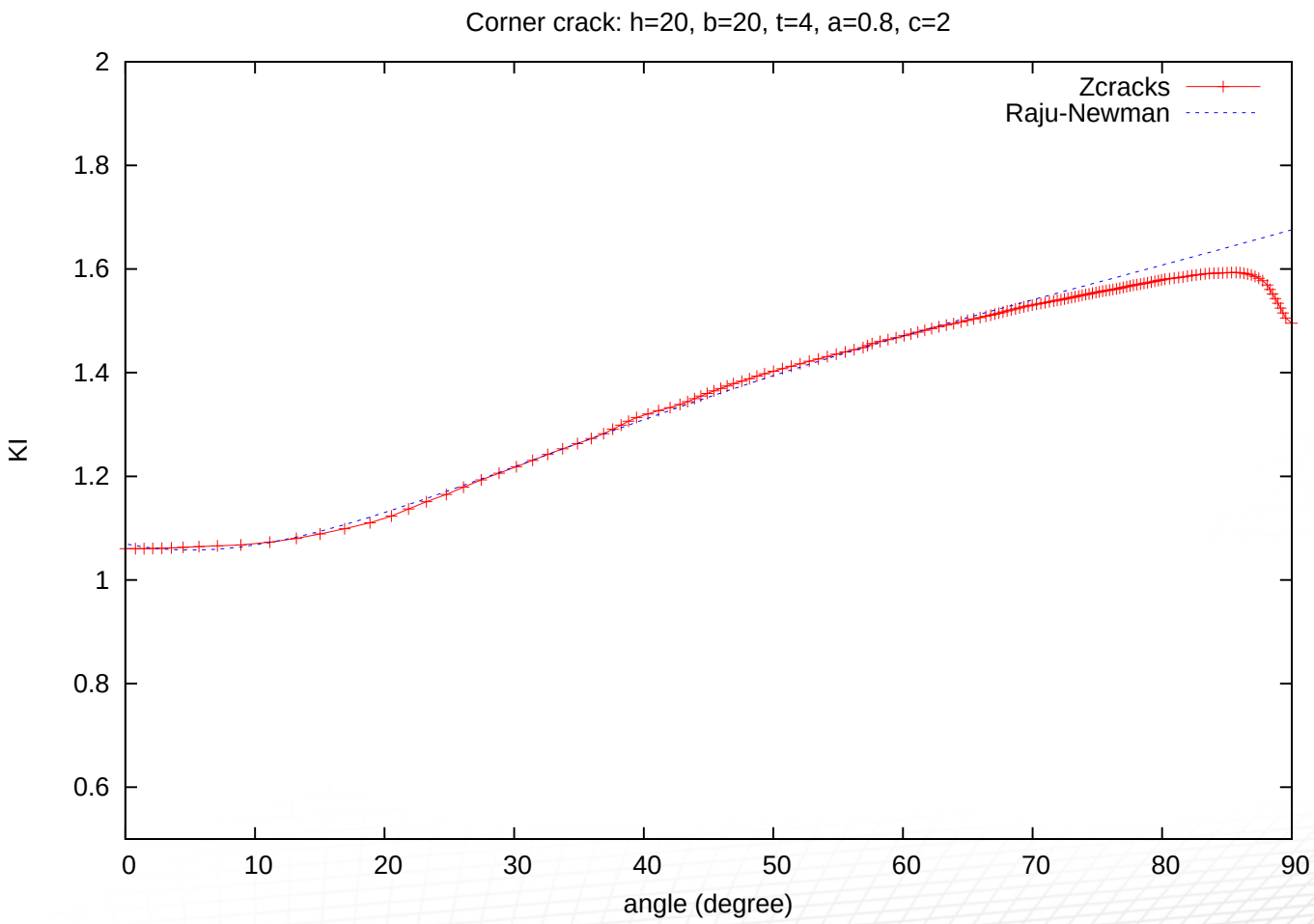
quadratic mesh						linear mesh				
Min s	Grad	Qn	#dof	e_{max}%	$\bar{e}$ %	Min s	Grad	#dof	e_{max}%	$\bar{e}$ %
5e-3	1.8	yes	480216	6.53	1.53	1e-2	1.8	32883	9.98	5.49
1e-2	1.8	yes	245355	5.01	1.40	1e-2	1.2	152907	7.17	2.01
2e-2	1.8	yes	125187	3.76	1.26	5e-3	1.2	313950	8.67	1.89
4e-2	1.8	yes	63921	3.43	1.17	2e-3	1.2	737631	13.34	2.31
1e-2	2.5	yes	166989	4.67	1.24					
1e-2	1.2	yes	1196433	???	???					

- same kind of results as in the previous case, except at the free surface where the Z-cracks solution deviates from the Raju-Newman one
- ???

Corner quarter-elliptical crack

results with a quadratic mesh:

- `min_size`=0.01 , `gradation`=1.8
- e_{max}=6% , $\bar{e}$=1.07%



Propagation laws

A wide variety of propagation models are available in Z-cracks, including:

- standard LEFM models (Paris, Elber, Forman, Walker)
- CRACKOXFLU advanced model (Kruch, Chaboche)
- an API allowing the user to define its own model in a **Zprogram** script

Standard LEFM propagation laws

- notations

  K_{max}, K_{min} : min, max values of stress intensity factor K on the cycle
  $\Delta K = K_{max} - K_{min}$
  $R = K_{min}/K_{max}$ (loading factor)
  $\langle f \rangle = f$ if $f > 0$ else $\langle f \rangle = 0$ (positive part)

- model definition and required coefficients

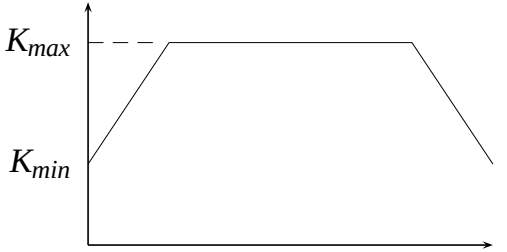
model	**equation**	**coefficients**
paris	$da = C\ (\Delta K)^m\ dN$	C, m
elber	$da = C\ \langle K_{max} - K_{op} \rangle^m\ dN$ $K_{op} = \langle K_{max} - (A + BR)\ \Delta K \rangle$	C, m, A, B
forman	$da = \frac{C\ \langle K_{max} - K_{th} \rangle^m}{(1-R)\langle K_c - K_{max} \rangle}\ dN$	C, m, K_{th}, K_c
walker	$da = C \left(\frac{\langle \Delta K - K_{th} \rangle}{(1-R)^{(1-\lambda)}} \right)^m\ dN$	C, m, K_{th}, λ

CRACKOXFLU propagation law (Kruch, Chaboche)

Phenomenological model for crack propagation under complex cyclic conditions including:

- fatigue with closure effect due to plasticity
- creep damage under tensile open crack condition
- environment-driven embrittlement effects increasing the fatigue propagation rate
- interaction of overloads with both the fatigue and creep crack growth
- influence of anisothermal loadings on crack growth rate

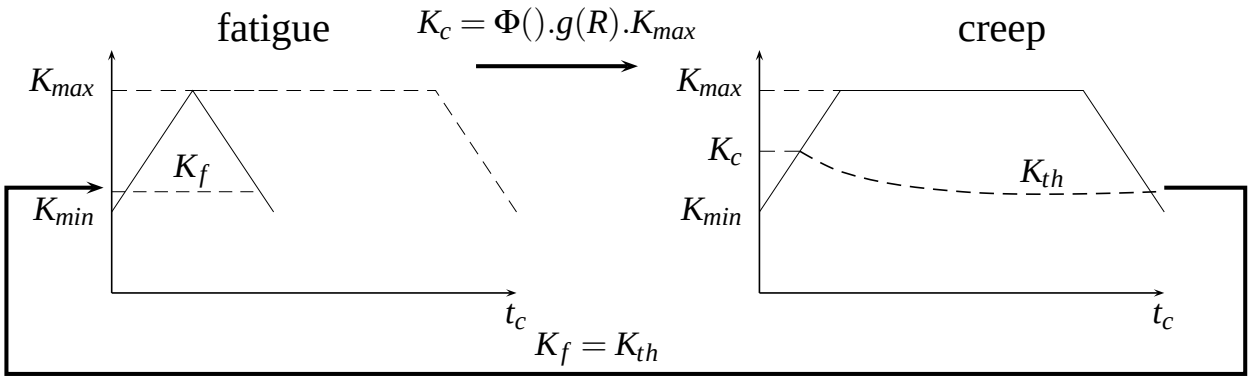
CRACKOXFLU: basic ingredients



$$\frac{da}{dN} = \frac{da_f}{dN} + \frac{da_c}{dN}$$

2 cracking mechanisms with threshold evolutions

- $\Phi()$ function accounts for overloads during the cycle
- $g(R)$ accounts for cycle loading ratio R



- loading ratio $R = \frac{K_{min}}{K_{max}}$
- toughness K_{Ic}
- fatigue threshold K_f
- Forman fatigue law :

$$\frac{da_f}{dN} = \frac{C_f \ \langle K_{max} - K_f \rangle^{\eta_f}}{(1-R)\langle K_{Ic} - K_{max} \rangle}$$

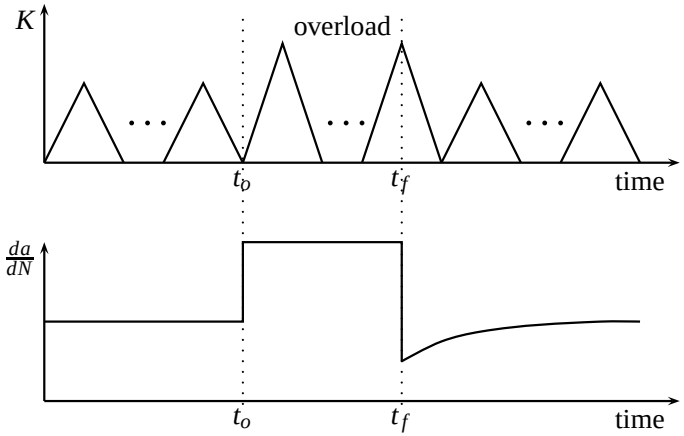
- creep threshold K_c
- creep damage integration on the cycle :

$$\frac{da_c}{dN} = \int_0^{t_c} C_c(T(t)) \ \langle K(t) - K_c \rangle^{\eta_c(T(t))}$$

- threshold relaxation :

$$dK_{th} = -A \left(\frac{K_{th}}{K} \right)^{\omega} K \, dt$$

CRACKOXFLU: influence of overloads



- crack growth rate increase during an overload
- rate decrease following the overload
- progressive vanishing of this effect until crack growth rate recovers its initial value

- definition of an *effective* $K_{Meq} = \Phi\, K_{max}$ used in the threshold calculation
- Φ depends on an estimation of current crack tip plasticity to account for overloads: $\Phi \geq 1$, $\Phi = 1$ wo overload

size of plasticity at the crack tip: $\rho = \frac{1}{2\pi}\left(\frac{K_{Meq}}{\sigma_y}\right)^2$ (1)

if $\left(K_{Meq}(i) \geq K_{Meq}(i-1)\right)$ increase of ρ using (1) , $\Phi = 1$

else

plasticity doesn't evolve but ρ is decreased by fatigue advance

$\rho(i) = \langle\, \rho(i-1) - da_f \,\rangle$

(1) inverted to get K_{Meq} : $K_{Meq}(i) = \sigma_y \sqrt{2\pi\rho(i)}$, $\Phi = \frac{K_{Meq}(i)}{K_{max}(i)} > 1$

("i" index of current cycle in the applied loading)

CRACKOXFLU: influence of environment

crack tip embrittlement by oxidation increasing the fatigue propagation rate

- integration of an oxide penetration length *lp* during the cycle

$$lp \;=\; \int_0^{t_c} \frac{1}{4}\,\alpha(T(t))\;t^{-\frac{3}{4}}\;dt$$

$\alpha(T)$ material coefficient depending on temperature

- decrease of *lp* by an amount corresponding to crack advance

$$lp \;=\; \langle\; lp - da_f - da_c \;\rangle$$

- local toughness K_c dependence on *lp*

$$K_c(lp) \;=\; K_{co}\;\left(1 - u - u\;exp\left(\frac{czm}{lp}\right)\right) \quad , \quad lp > 0$$

K_{co} toughness of a completely embrittled material, u, czm material coefficients

- use of $K_c(lp)$ in the fatigue law

$$\frac{da_f}{dN} \;=\; \frac{C_f\;\langle K_{max} - K_f\rangle^{\eta_f}}{(1-R)\,\langle K_c(lp) - K_{max}\rangle}$$

CRACKOXFLU: anisothermal effects

$K(t)\ ,\ t = 0, t_c$: K values during cycle of period t_c
$T(t)\ ,\ t = 0, t_c$: temperature values

- normalization of K by $K_n(T)$

$$S(t) \ = \ \frac{K(t)}{K_n(T(t))} \quad , \quad t = 0, t_c$$

  toughness K_{Ic} may be a good choice for K_n

- fatigue law written using S instead of K

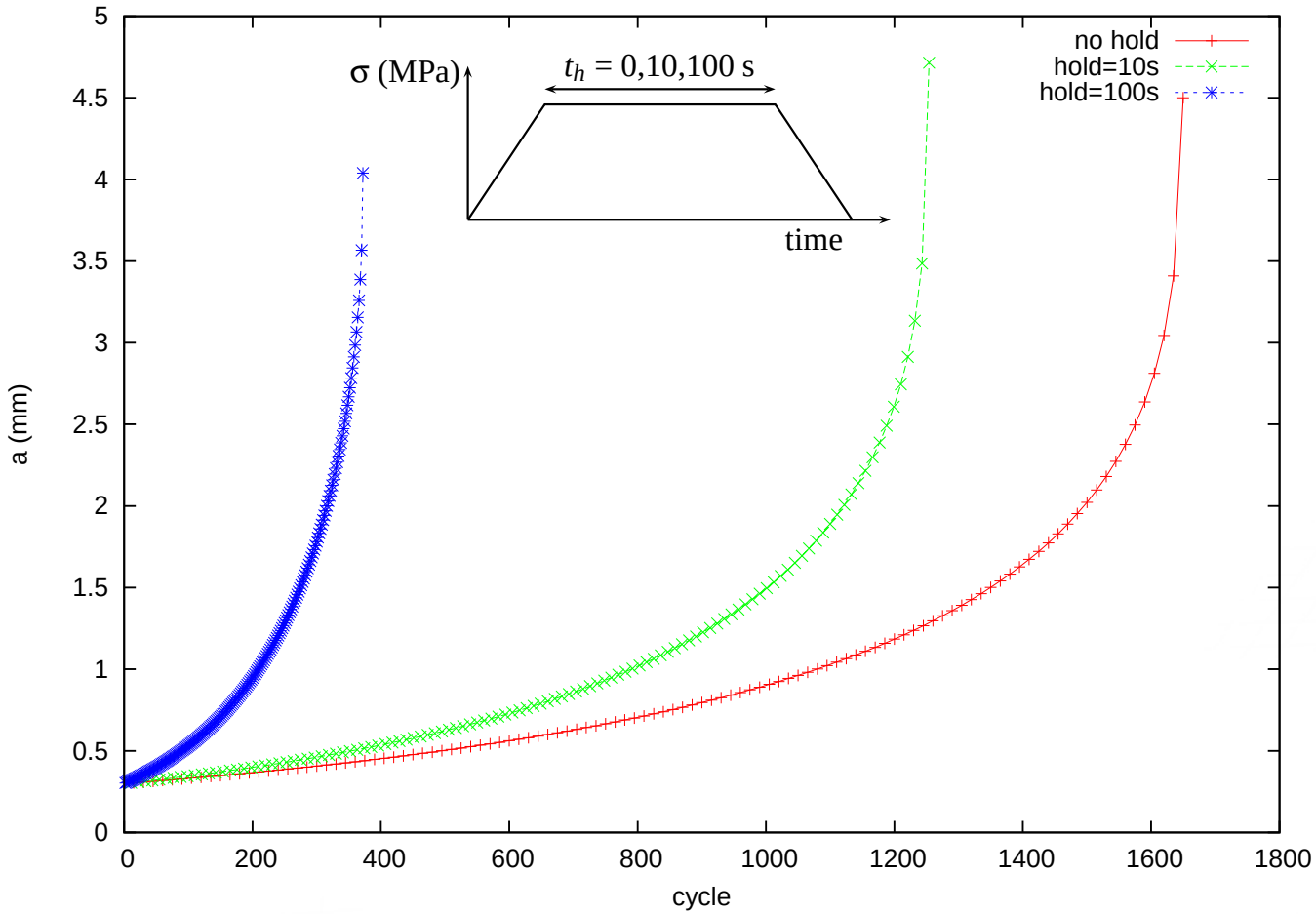
$$\frac{da_f}{dN} \ = \ \frac{C_f \ \langle S_{max} - S_f \rangle^{\eta_f}}{(1 - R) \ \langle 1 - S_{max} \rangle}$$

  with constant values for coefficients (C_f, η_f) calibrated on a master curve $\left(\frac{da}{dN} \ = f \ (\Delta S) \right)$ obtained from crack propagation tests at various temperatures

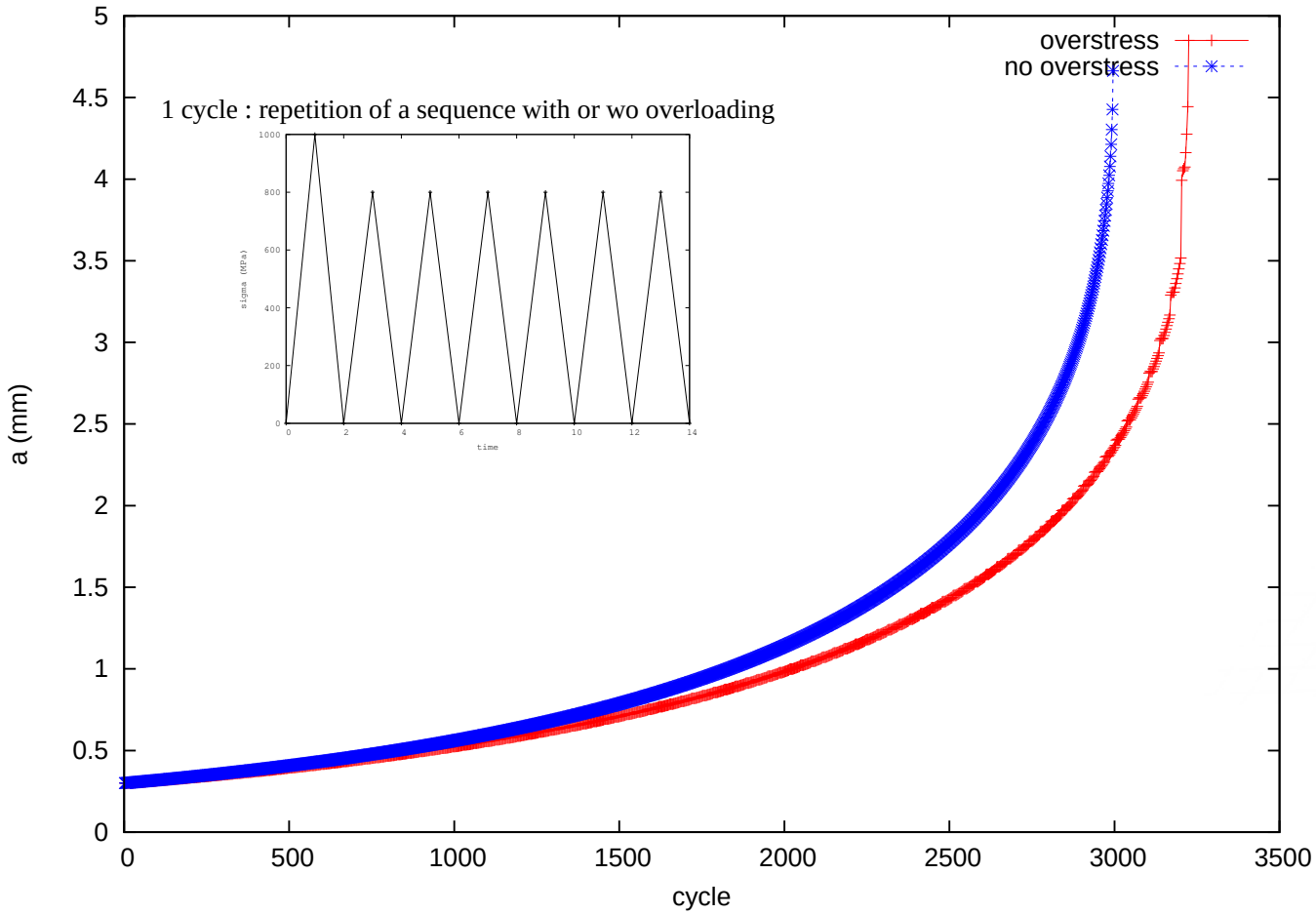
- creep law with coefficients depending on temperature

$$\frac{da_c}{dN} \ = \ \int_0^{t_c} C_c(T(t)) \ \langle K(t) - K_c \rangle^{\eta_c(T(t))}$$

CRACKOXFLU results: influence of holding time (creep)



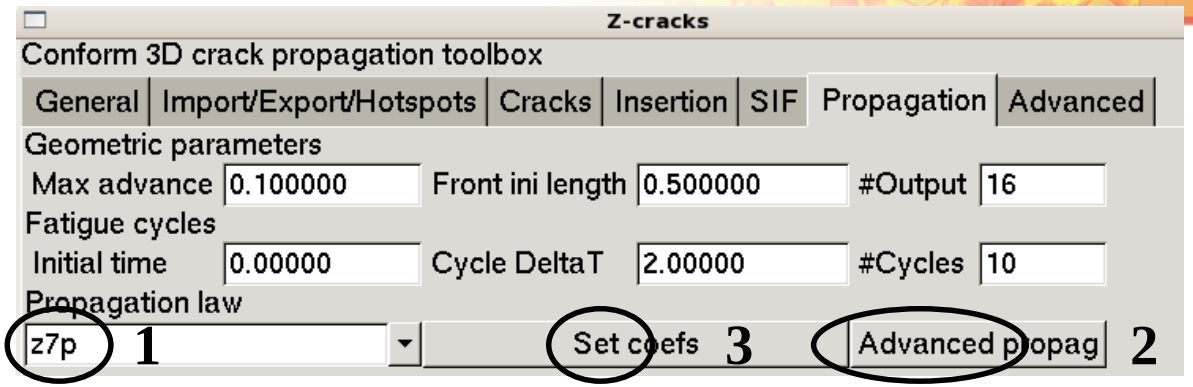
CRACKOXFLU results: influence of overloads



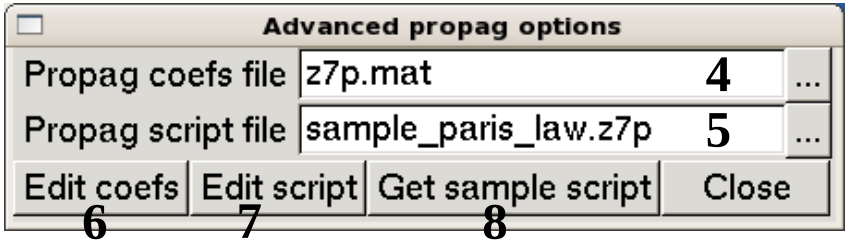
User propagation laws

A simple API is available in Z-cracks to bypass built-in models and allow the user to implement its own propagation law in a Zprogram script (interpreted script language).

User models GUI controls

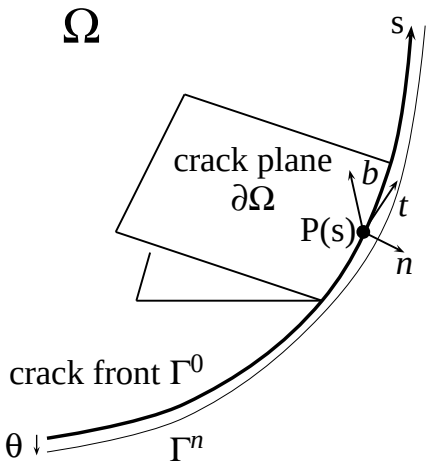


- select **z7p** for a propagation model defined in a user script using combo box **(1)**
- use the **Advanced propag** command to open a graphics dialog allowing to define the name of the script file **(5)**



- (7) allows to edit the script file selected
- with (8) a sample user script is fetched from the **Z-set** database. This script implements a basic Paris law that can be used as a basis to write new models
- (4) allows to define the name of a file with model coefficients expected in the script, while (7) edits the material file selected

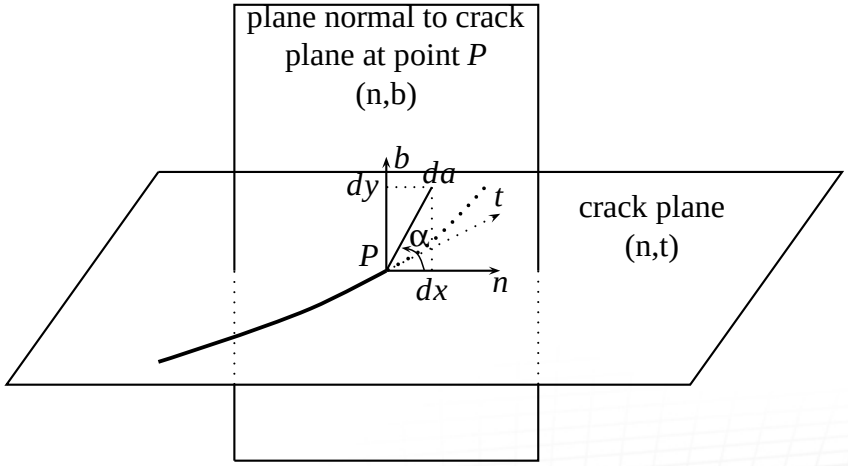
Definitions



for each point $P(s)$ along crack front Γ^0, and with the definition of a local frame (n, t, b) as represented on the figure:

- n normal to crack front at point $P(s)$ in the crack plane
- t tangent to crack front at point $P(s)$ in the crack plane
- b normal to the crack plane at point $P(s)$

and for the following definition of bifurcation angle α in plane (n, b), the user script should define:



- crack advance da in direction α
- projections (dx, dy) of da on local vectors (n, b):

$$dx = cos(\alpha).da$$
$$dy = sin(\alpha).da$$

Propagation script API

the following 2 functions are expected in the propagation script:

Declaration of coefficients

```
void declare_coefs()
{  coef_names.resize(5);
   coef_names[0]="some_coef";
   coef_names[1]="K";
   ...
   coef_names[4]="another_coef";
}
```

Crack advance calculation

```
double compute_propag(VECTOR& Gtimes, VECTOR& G,
  VECTOR& A, VECTOR& T, double& dx, double& dy)
{ double da;
  da = K()*...;
  dx = ...;
  dy = ...;
  return(da);
}
```

- `coef_names` is an array of strings. This array should be resized/filled with the corresponding names of coefficients to be read in the propagation material file. Coefs can then be referenced by their name in function `compute_propag()` using the `()` operator as shown above
- inputs of function `compute_propag()` are vectors whose size correspond to the number of increments stored in the FE results file for the current cycle:
  - ⋆ `Gtimes`, `T`: time and temperature values
  - ⋆ `G`: SIF values (G or K_I depending on the option selected) computed by the `G-theta` method for the current front point
  - ⋆ `A`: bifurcation angles computed by the `G-theta` post-processor
- outputs are the da, dx, dy scalar values of the crack advance defined in the previous slide

Propagation script example

An implementation of a simple Paris law using the user API is given hereafter for reference:

```
void declare_coefs()
{  coef_names.resize(2);
   coef_names[0]="C";
   coef_names[1]="m";
}

double compute_propag(VECTOR& Gtimes, VECTOR& G, VECTOR& A, VECTOR& T, double& dx, double& dy
{ int i;
  double Gmin, Gmax, Amax, Tmax, da;
  // Get min max
  Gmax = 0.0; Gmin = 1.e50; Amax = 0.0; Tmax = T[0];
  for(i=0;i<!Gtimes;i++) {
      // Do not take into account initial transferred value
      if((!Gtimes>2)&&(Gtimes[i]<=0.0)&&(i==0)) continue;
      if(G[i]>Gmax) { Gmax = G[i]; Amax = A[i]; Tmax = T[i]; }
      if(G[i]<Gmin)   Gmin = G[i];
  }
  if(Gmin<0.0)   Gmin = 0.0;
  if(!Gtimes==1) Gmin = 0.0;

  for(i=0;i<!coefficients;i++) coefficients[i].compute_value(Tmax);

  da = C()*(Gmax-Gmin)^m();
  // using the angle corresponding to Gmax to compute (dx,dy)
  dx = da*cos(Amax);
  dy = da*sin(Amax);
  return(da);
}
```

Zprogram scripts

- chains various operations involved in SIF computation and Crack Propagation procedures:
  generate input FE command to compute a cycle on the current cracked mesh, transfer fields on the cracked mesh when needed, calculate SIF and crack advance according to a given propagation law, remesh the structure according to the new crack position ...
- uses the command line interpreter built in the **Zrun** program (C/C++ syntax)
- can be modified/adapted by the user without compilation

SIF computation script: cracked_SIF.z7p

```
int main()
{ STRING cmd, fe_cmd;

  init_var();
  sane_name = "cube_abaqus";
  cracked_name = "cracked";
  thermal_field = "";
  format        = "abaqus_templated";
  smp           = 2;
  fe_cmd        = "Zmat -fg -mpi 2 ";
  if_contact    = 1;

  if(!thermal_field) {
     write_thermal_tranfer_inp(0);
     cmd = "Zrun -fe_transfer transfer_therm";
     system(cmd);
  }
  cmd = "Zrun -m EXPORT";
  system(cmd);
  cmd = fe_cmd+" "+cracked_name+"_SIF";
  system(cmd);
  cmd = "Zodb -pp -smp "+itoa(smp)+" "+cracked_name+"_GPP";
  system(cmd);
}
```

initializations from the GUI settings

mapping of thermal fields on the cracked mesh (if defined)

write FE input file to do 1 cycle on the cracked mesh

run FE calculation

SIF calculation from FE results

Propagation script: cracked_PROPAG.z7p (1/4)

```
int main()
{
  init_var();
```

```
smp            = 2;
cracked_name   = "cracked";
...
```

initializations from the GUI settings

```
start = 1;     // num of the first cycle computed
nb_cyc = 100;  // total number of cycles to compute
```

modify those values to do a restart

eg. set start=101 to calculate cycles 101-200

```
  // Write advance remeshing file, do not change during propag
  write_remesh_new(1);

  err = 0;
  for(cyc=start;cyc<(start+nb_cyc);cyc++) {
```

loop on propagation cycles

```
    cur_cyc = cyc;
    cout<<endl<<endl<<"-> Start of cycle: "<<cyc<<" --"<<endl<<endl; cflush();

    if(cyc==1) geof_name = cracked_name+".geo";
    else       geof_name = cracked_name+"_PROPAG"+itoa(cyc-1)+".geo";
```

```
    if(!thermal_field) {
      write_thermal_tranfer_inp(cyc-1);
      cmd = "Zrun -fe_transfer transfer_therm";
      err = system(cmd);
      if(err) {
        cout<<endl<<"Error when transferring thermal field to mesh: "<<geof_name<<" end"<<endl<<endl;
        break;
      }
    }
```

mapping of thermal fields on the new cracked mesh (if defined)

```
  ...
  }
}
```

SIF computation script: cracked_PROPAG.z7p (2/4)

```
int main()
{ ...

  for(cyc=start;cyc<(start+nb_cyc);cyc++) {
    ...
    if(if_transfert * (cyc>1) ) {
      write_sdv_tranfer_inp(cyc-1);
      cmd = "Zodb -fe_transfer -s ODB.AutoConvert 1 transfer_sdv";
      err = system(cmd);
      if(err) {
        cout<<endl<<"Error when transferring sdv to mesh: "<<geof_name<<" end"<<endl<<endl;
        break;
      }
      system("rm -f  REEQUILIBRIUM*");
      err = export_mesh_templated(cyc,1);
      if(err) {
        cout<<endl<<"Failed to generate the REEQUILIBRIUM "<<format<<" input file";
        break;
      }
      cmd = fe_cmd + " -sdv0 REMESHED REEQUILIBRIUM";
      cout<<" . reequilibrium at cycle "<<cyc<<" using command: "<<cmd<<endl<<endl; cflush();
      err = system(cmd);
      if(err) {
        cout<<endl<<"An error occured during reequilibrium at cycle: "<<cyc<<" end"<<endl<<endl;
        break;
      }
    }

    cout<<endl<<" . generation of "<<format<<" input file for cycle: "<<cyc<<endl<<endl; cflush();
    err = export_mesh_templated(cyc,0);
    if(err) {
      cout<<endl<<"Failed to generate the "<<format<<" input file at cycle: "<<cyc<<endl<<endl;
      break;
    }
    ...
  }
}
```

loop on propagation cycles

transfer SDV + re-equilibrium (if needed)

transfer SDV on the new mesh

run FE re-equilibrium increment

export FE input file for next cycle

SIF computation script: cracked_PROPAG.z7p (3/4)

```
int main()
{ ...

  for(cyc=start;cyc<(start+nb_cyc);cyc++) {
    ...
    cmd = fe_cmd;
    if(if_transfert * (cyc>1) ) cmd = cmd + " -oj REEQUILIBRIUM ";
    cmd = cmd + " " + cracked_name+"_PROPAG"+itoa(cyc);
    cout<<endl<<" . computation of cycle "<<cyc<<" using command: "<<cmd<<endl<<endl; cflush();
    err = system(cmd);
    if(err) {
       cout<<endl<<"An error occured during FE computation of cycle: "<<cyc<<" end"<<endl<<endl;
       break;
    }

    gpp_name = cracked_name+"_GPP_PROPAG"+itoa(cyc);
    write_gpp_templated(cyc);
    cout<<endl<<" . computation of SIF at cycle: "<<cyc<<endl<<endl; cflush();
    cmd = "Zodb -s ODB.AutoConvert 1 -pp -smp "+itoa(smp)+" "+gpp_name;
    err = system(cmd);
    if(err) {
       cout<<endl<<"An error occured when computing SIF at cycle: "<<cyc<<" end"<<endl<<endl;
       break;
    }
    ...
  }
}
```

loop on propagation cycles

run FE calulation of next cycle
(note, restart from re-equilibrium if needed)

calculate SIF from FE results

SIF computation script: cracked_PROPAG.z7p (4/4)

```
int main()
{ ...

  for(cyc=start;cyc<(start+nb_cyc);cyc++) {
     ...
     name = gpp_name+".ZCPOST";
     cout<<endl<<" . reading SIF in file: "<<name<<endl; cflush();
     err = read_zcpost(name,fronts,Gvalues);
     if(err) {
        cout<<endl<<"Invalid GPP results file: "<<name<<" ... end"<<endl<<endl;
        break;
     }
     if(fronts.size()==0) {
        cout<<endl<<"No more crack front: calculation is finished."<<endl<<endl;
        break;
     }
     compute_advance(fronts,Gvalues,max_h);
     // Write corresponding advance file
     adv_name = cracked_name+"_PROPAG"+itoa(cyc)+".adv";
     write_advance(adv_name,fronts);

     // Copy files with standard names needed by drive_crack_remesh
     STD_CP(gpp_name+".geo","TO_REMESH.geo");
     STD_CP(adv_name,cracked_name+"_PROPAG.adv");

     // Perform remeshing
     do_drive_remesh(max_quality);

     ascf.open("REMESHED.geo");
     if(!ascf.ok) {
        cout<<endl<<"Remeshing failed, no REMESHED.geo file ... end"<<endl<<endl;
        break;
     }
     STD_CP("REMESHED.geo",(cracked_name+"_PROPAG"+itoa(cyc)+".geo"));
  }
}
```

loop on propagation cycles

calculate advance from propagation law and write adv file (remeshing input)

remesh to account for new crack position

SIF output files

- SIF along the crack front(s) are written is ASCII files **FRONT0_SIF**, ..., **FRONT*n-1*_SIF** (for front number ***n***)
- those files are used by the **SIF tab** to draw curves with the **gnuplot** program
- for each n_p points in the front discretization / line in the file, SIF output is stored by column:

```
# s x y z G T angle KI KIi KIIi KIIIi Gii @ N=0 cycles
0.000000e+00 1.077127e-02 0.000000e+00 5.595189e-01 2.172077e+03 0.000000e+00 ...
...
1.664700e-01 -2.168404e-19 1.660996e-01 5.618385e-01 2.217370e+03 0.000000e+00 ...
```

1: curvilinear coordinate s of current point on the front
2-4: x, y, z coordinates
5: G value
6: T (temperature) if anisothermal (otherwise 0.0)
7: angle α if **Outplane propagation** 8: K_I (derived from G) when **Compute SIF** 9-11: K_I, K_{II}, K_{III} if **Interaction integral**

PROPAG output files

- the propagation script writes a file named **cracked_PROPAG.zck** containing the following info (one line for each computed cycle)
- caution, number of detected fronts can change from one cycle to the othermay be difficult to interpret

	# column
for c = 1 to n_c computed cycles	
n_r : cumulative number of *real* cycles to reach the required advance	1
dn_r : increment number of *real* cycles	2
c : index of current *computed* cycle	3
t : time at end of current *computed* cycle	4
$t = t_0 + c.T$, T cycle period , t_0 initial time entered in (8), (7) (see PROPAG tab)	
n_p : number of points stored on each front (field (9) of the PROPAG tab)	5
n_f : number of fronts detected during the current *computed* cycle	6
for f = 1 to n_f crack fronts	
for p = 1 to n_p stored points on front f	
12 values stored for point p starting at column $c_p = 6 + 12\left[n_p\,(f-1) + p - 1\right] + 1$	
(x, y, z) : coordinates of point	c_p
a : cumulative advance , $a = \sum_{i=1}^{c} da(i)$	$c_p + 3$
(u_x, u_y, u_z): displacement of point p due to advance at cycle c	$c_p + 4$
da : crack advance due to cycle c , $da = \sqrt{u_x^2 + u_y^2 + u_z^2}$	$c_p + 7$
K_{min}, K_{max} : min,max value of SIF during cycle c can be G or K_I depending on the SIF option selected	$c_p + 8$
T_{min}, T_{max} : min,max value of temperature on cycle c (0.0 if no temperature dependency)	$c_p + 10$