Fatigue analysis - local geometry optimization

Anna Lundkvist (anlu0210@student.umu.se)

July 9, 2018

Abstract

The cause of fatigue failure is repeated loads, that cause cracks to appear and grow even if the loads are far below the static load that would make a structure fail. High-cycle fatigue, which this project will focus on, is characterized by linear elastic stress and only fails after a large amount of loading cycles. While fatigue is the most common cause of failure in structures, it is not feasible to calculate fatigue damage analytically. The aim of this project was to develop, implement and test a workflow that unifies the wide range of physical scales and transient features that are relevant to fatigue analysis of complex dynamic machinery. The workflow should take both system-level and local aspects into account. The goal was to address both the global and local while still keeping practical feasibility and simulation performance in mind. The resulting unified fatigue analysis method was then used on several test cases and illustrated from a local geometry optimization perspective.

The workflow contains the following steps: First the model is to be simulated in order to get the load history. Then the finite element method (FEM) is used to make a submodel of the component that is to be analyzed. The submodel is subjected to forces and moments, and then the stress is extracted from the areas of interest in the model. Thus, a linear relation for the stress can be calculated. The stress history is calculated by putting the load history into the stress relation. Using established fatigue analysis methods like rainflow counting and the Palmgren-Miner rule the fatigue life is then calculated.

This project only had its focus on the FEM submodel part of the fatigue workflow. The geometry of the submodel should then be able to be optimized for the longest fatigue life.

The workflow was tested on several test cases. The first one was a simple upside-down T-shaped component that was made by welding the parts together. The same component but with no weld, as if it had been moulded, was the next case. The final case was a component from the base of a crane. Two things were analyzed on this case. The first was the fatigue life around a hole in which a shaft was welded. The other was optimizing to find the best position for a hook to be welded onto the surface on the component.

Advisor at Algoryx: Peter Norlindh
peter.norlindh@algoryx.se
Examinator: Martin Servin
martin.servin@umu.se

Master's Thesis in Engineering Physics

Contents

1 Introduction **3**

2 Theory **4**
2.1 Stress . . . 4
2.1.1 Saint-Venant's principle . . . 4
2.2 Fatigue . . . 4
2.3 Current way of fatigue analysis . . . 4

3 Fatigue Analysis Building Blocks **6**
3.1 Stress amplitude and mean stress . . . 6
3.2 The S-N curve and the Basquin relation . . . 6
3.3 Palmgren-Miner rule . . . 7
3.4 Rainflow counting . . . 7
3.5 The hot spot method . . . 8

4 Tools **9**
4.1 Code_Aster . . . 9
4.2 WAFO . . . 9
4.3 SpaceClaim Algoryx Momentum . . . 9

5 Method of unified fatigue analysis **10**
5.1 Overview . . . 10
5.2 Assumptions . . . 11
5.3 Loads from dynamic system level simulation . . . 11
5.4 From loads to stress . . . 12
5.4.1 The locked interface . . . 13
5.5 From stress to weld model . . . 13
5.6 From weld model to rainflow counting . . . 13
5.7 From rainflow counting to S-N curve . . . 14
5.8 From S-N curve to Miner-Palmgren rule . . . 14

6 Test cases **15**
6.1 Case 1 . . . 15
6.2 Case 1.5 . . . 16
6.3 Case 2 . . . 17

7 Results **19**
7.1 Case 1 . . . 19
7.2 Case 1.5 . . . 19
7.3 Case 2 . . . 20

8 Discussion **23**
8.1 Case 1 24
8.2 Case 1.5 24
8.3 Case 2 25

9 Conclusion **26**

10 References **27**

A Appendix **28**

B Appendix **30**

1 Introduction

Fatigue is the most common cause of failure in mechanical structures. A structure may break when subjected to a repeated load far below what it would be able to bear in the case of a static load. When designing a structure, it is often not possible to make it so that it will never break from fatigue. Instead the designers aim to construct the structure so that it will not break during a sufficient long time, called its design life[1].

Unfortunately, there is no analytical way of calculating fatigue. Many assumptions have to be made, and small differences in the material and in the surface affect the life time. The stress that is caused by the load that the structure is subjected to causes damage, but that stress depends on the geometry of the component as well as residual stresses from welds or other things from its construction. The loading that the structure is subjected to may also not be exactly what it is designed against. For example, winds may cause a crane stress. The use not being what the designer intended will also change the life time. Corrosion is another thing that will make a component more susceptible to fatigue failure.

When designing against fatigue, the use of standardized calculation procedures is common[1]. In order to get more accurate calculations, it is important that the load that the structure is subjected is as accurate as possible.

The problem with getting a good fatigue analysis today is that engineers hardly ever are able to simulate the mechanical system in a way that gives the loadings with high resolution at the local level. Instead they have to rely on loading cases that are not necessarily representative for the loads the structure will be subjected to in use. But they have no general work sequence they can get load history for. Even if they try to use conservative values that are supposed to be an high estimate of the damage it is still possible that they are too low.

The aim of this project is to create a workflow to calculate fatigue, and then optimize with respect to local geometry. The workflow should be a procedure to use for components where the user will check areas that they think are likely to break due to fatigue. Then the point with the shortest life time will be found.

On the construct that one wishes to do a fatigue analysis on, the part that should be most susceptible to fatigue should be chosen. A submodel of that part will be made in order to use the finite element method (FEM) to calculate the stress when the sub model is subjected to unit forces. Then a linear relation will be calculated between the load and the stress. By simulating the construct, the load history can be produced and the stress history is then calculated with the linear stress relation. At that point established fatigue analysis methods can be used. That way, the stress that the component is subjected to will be more realistic compared when using loading cases, which will lead to more accurate results.

This project will only focus on the FEM calculations and the stress model, as well as the local geometry.

2 Theory

2.1 Stress

Stress is the force per area. If the stress is in the direction of the surface normal it is called normal stress. Stress that acts in the surface's tangent plane is called skew stress.

The stress components at a point in Carthesian coordinates is

$$\sigma_{ij} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix}. \tag{1}$$

The diagonal components are the normal stresses, and the non-diagonal are the skew stresses. Since every point is in equilibrium there is no net torque, and $\sigma_{ij} = \sigma_{ji}$[2].

2.1.1 Saint-Venant's principle

Saint-Venant's principle says that if a boundary condition is changed to a statically equivalent one the only difference is locally. Farther away from the boundary the solution should be approximately the same in the static case as it is in the original one. Two statically equivalent loads should cause the same stress sufficiently far away from the loads. What constitutes as sufficiently far is not specified, but depends on the case.[3]

2.2 Fatigue

Fatigue is the failure caused by cyclic loads. A repeated load can cause failure even if it is far below the static load that the structure can bear. Fatigue tests are carried out to find fatigue properties of materials and structures. Data for many materials and structures have been published.

Fatigue is divided into two areas, high-cycle fatigue (HCF) and low-cycle fatigue (LCF). Low-cycle fatigue happens when there is high stress amplitudes or stress concentrations and much plastic deformation occurs. The fatigue life is then considerable shorter. HCF does not have as dramatic plastic deformation, and the fatigue life is then longer[4]. This project will not consider LCF, and will instead focus on HCF.

Stress concentrations happens at the surface because of inhomogeneous stress distribution due to geometric factors. Therefore crack initiation happens at the surface of a material. The surface of a structure will thus affect its fatigue life[1].

Fatigue life is the number of cycles to failure. Under a certain stress amplitude, a specimen may never fail, no matter how many cycles it is subjected to. This stress amplitude is called the fatigue limit. However, damage to the surface, such as corrosion, will lower this limit, so it cannot be relied on. Some metals and metals alloys do not have a fatigue limit[4].

2.3 Current way of fatigue analysis

The usual way to do a high-cycle fatigue analysis currently is shown in table 1.

Table 1 – An overview of a usual fatigue analysis workflow[5].

Step	
1	Find category that most closely corresponds to the detail of the case
2	Find loading case that corresponds most closely to the real loading
3	Use factors to account for dynamic and other effects
4	Calculate fatigue life using the given values with a S-N curve

3 Fatigue Analysis Building Blocks

The building blocks in the fatigue analysis are the following

- σ_m
- σ_a
- S-N curve
- Miner-Palmgren rule
- Rainflow counting
- Weld model
- Loads from a dynamic, system level simulation
- Converting the loads to stress

These building blocks will be explained below.

3.1 Stress amplitude and mean stress

The stress amplitude, σ_a, is defined as half the stress range $\Delta\sigma$:

$$\sigma_a = \frac{\sigma_{max} - \sigma_{min}}{2} = \frac{\Delta\sigma}{2}. \tag{2}$$

The mean stress is defined as [4]

$$\sigma_m = \frac{\sigma_{max} + \sigma_{min}}{2}. \tag{3}$$

3.2 The S-N curve and the Basquin relation

A Wöhler curve, or S-N curve is obtained by doing fatigue tests at different stress levels. Plotting the stress amplitude σ_a versus the fatigue life N_f on a logarithmic scale gives a linear relation for a large range of N_f. This linear relation is called the Basquin relation and can be written as

$$\sigma_a = \sigma'_f(2N_f)^b, \tag{4}$$

where σ'_f is the fatigue strength coefficient and b is the Basquin exponent. The Basquin relation is only valid when the mean stress, σ_m, is zero. But a modification was made by Morrow in 1968 to account for mean stress effects:

$$\sigma_a = (\sigma'_f - \sigma_m)(2N_f)^b. \tag{5}$$

From this expression the fatigue life can be calculated[4].

3.3 Palmgren-Miner rule

The Palmgren-Miner rule is a simple model for fatigue damage accumulation. The fatigue damage is calculated as

$$D = \sum \frac{n_i}{N_{fi}}, \tag{6}$$

where n_i is number of cycles with the ith stress amplitude and N_{fi} is the fatigue life for that stress amplitude. When $D = 1$ failure occurs.

The peak stress in a notch may be larger than the yield stress, the stress at which the material deforms plastically. If so plasticity will occur in the root of the notch and that notch root plasticity can give compressive residual stress, which makes the damage of the following loading sequences milder. As such, the order of the loading sequences does matter to fatigue damage, but is not taken in account by the Palmgren-Miner rule.

According to the Palmgren-Miner rule, the stress below the fatigue limit is non-damaging, since N_{fi} is infinite, and as such $n_i/N_{fi} = 0$. But while stress amplitudes below the fatigue limit cannot create a growing microcrack under constant loading, if the specimen has already been subjected to stress over the fatigue limit, the crack can grow under this stress.

The Palmgren-Miner rule was created to calculate the damage of loading with varying amplitude, but does not have a physical basis to it. Fatigue damage is dependent on many variables, and cannot be described accurately with a single parameter. But despite its weaknesses, the Palmgren-Miner rule has been found to work reasonably well and it is widely used when calculating fatigue[1].

3.4 Rainflow counting

In order to analyze variable amplitude fatigue, cycle counting methods can be used to turn the stress history into simple stress reversals. That way, the data obtained from constant amplitude fatigue can be used to analyze the stress history. The rainflow counting method is an example of a cycle counting method.

The idea of rainflow counting is to take the stress history and turn it 90° with the start of time at the top. Imagine that stress history as a pagoda roof, with rain running down it. Start at a valley and let the water flow downwards. If the next valley is deeper, stop at the peak. Otherwise continue down to the next valley. When a rainflow stops, start a new one from the valley below the one where the last rainflow started. If a water flow encounters another from above, it stops. Go through all valleys in that manner. The same is done for the peaks. That way, all the stress history should be counted in the rainflow. Then half-cycles are counted, and if the stress ranges and mean stress match, they are put together to whole cycles[4]. Finding the amplitude and mean of these cycles is easily done by checking the value the cycle starts and where it ends. Then all information that is needed to use the Palmgren-Miner rule is present.

In fig. 1 an example of rainflow counting is shown for a simple stress history.

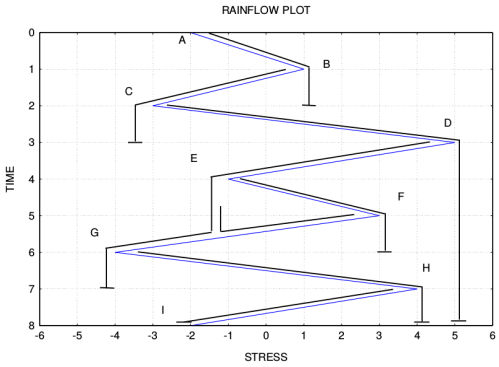


Figure 1 – An example of rainflow counting done on a simple stress history.[6]

There are of course algorithms that can perform rainflow counting on the stress history data.

3.5 The hot spot method

A hot spot is a critical point where a crack is assumed to grow. In a welded structure, the hot spot is at the weld toe. The hot spot stress is defined as the geometrical stress at the hot spot. At the hot spot, there is usually an non-linear stress distribution caused by the geometry of the weld. This area is estimated to be within a radius of $0.4t$ and $0.3t$, where t is thickness of the structure. The geometrical stress does not include the non-linear stress peak, by definition. Getting the geometrical stress in the hot spot can be done by extrapolation. The equation for quadratic extrapolation is

$$\sigma_{hs} = 2.52\sigma_{0.4t} - 2.24\sigma_{0.9t} + 0.72\sigma_{1.4t}, \tag{7}$$

where σ_x is the surface stress x mm away from the weld toe[7].

4 Tools

The tools used to do the fatigue analysis will be presented here.

4.1 Code_Aster

Code_Aster is a free FEM software, made by EDF R&D. It is available on Unix and Linux, and the command language is in Python[8].

Salome-Meca is a platform that can be used with Code_Aster. Using Salome-Meca's Geometry module, the geometry can be defined either through tools of the platform or by importing a file describing it. In the Mesh module, the mesh can be computed, using a number of algorithms. Parameters such as maximum and minimum size of elements can be change in order to get a mesh suitable to one's needs. It is also possible to set local sizes on areas and edges of the geometry. Groups of nodes, or faces on the mesh, can be defined from the geometry, or by the user selecting them. These groups are needed to extract values or set a force on them.

Eficas Command File Editor and Syntax Analyser can be used to generate command files for Code_Aster with correct syntax. Writing other Python code, for example when post-processing, must be done outside Eficas[8].

In the command file, the model for the case is chosen. All cases in this project use the 3D model. However, moments cannot be added directly to 3D models in Code_Aster. To get moment onto the 3D model it can be added to a free node that one connects to the group of nodes on the face on the 3D model.

In the Aster module study cases are made, by choosing a command file and a mesh. In the Aster module the interface ASTK can be run. From ASTK it is possible to use the data from a previous run as the base for a new one, which makes it convenient to use for post-processing in a separate run. This is useful when the original solution takes a long time.

The results can be inspected in the module ParaViS.

4.2 WAFO

WAFO stands for Wave Analysis for Fatigue and Oceanography, and it is a Python package for statistical analysis and simulation of random waves and loads. It contains functions for rainflow counting[9].

The version of NumPy that WAFO uses is a newer version than the one Code_Aster 2016 uses. In order to get WAFO work in a script run by Code_Aster 2016 the version of NumPy has to be replaced by the version used by WAFO.

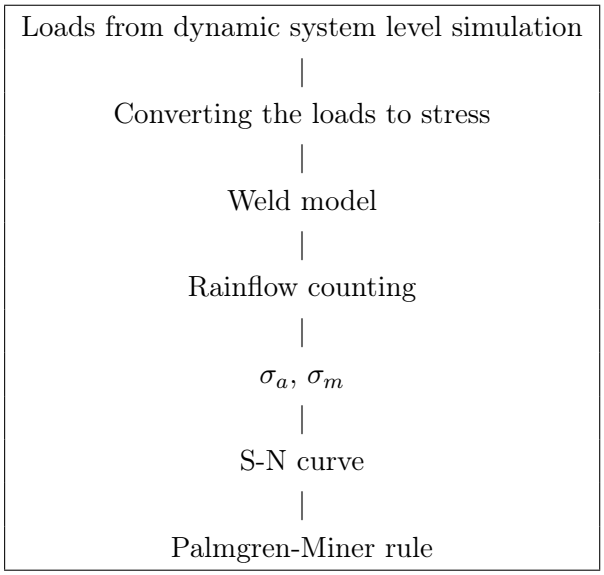
4.3 SpaceClaim Algoryx Momentum

SpaceClaim is a 3D modeling CAD software. It is available on Windows. It has the add-on Algoryx Momentum, a multibody dynamics tool. With Momentum, it is possible to simulate the models in SpaceClaim. Momentum will not be directly used in this project, but data from Momentum simulations will be used.

5 Method of unified fatigue analysis

Here the way of unifying the fatigue analysis building blocks into a workflow will be presented. The order is first presented in table 2 and after an overview the path between the building blocks will then be explained.

Table 2 – The order of the fatigue analysis building blocks.



5.1 Overview

The fatigue analysis workflow is summarized in table 3. Step 2 through 5 are the new parts in this workflow, while the steps after them are general fatigue analysis steps that are also used elsewhere.

In order to use this workflow some things are necessary. First, a way to use multibody dynamics, to simulate a moving mechanical systems. Second, a FEM tool which lets the user subject the model to many different loading cases. Third, a rainflow counting algorithm. In appendix B the code for the process from step 5 and below is available for the same case.

Table 3 – An overview of the fatigue analysis workflow.

Step	
1	Simulate model for load history
2	Make sub model with FEM
3	Lock one interface, subject others to unit forces and moments
4	Calculate linear relation between stress and load
5	Put load history in stress relation to get stress history
6	Use rainflow counting to get stress amplitude, mean stress and number of cycles
7	Find properties σ'_f and b
8	Use the Basquin relation to calculate the fatigue life of the cycles
9	Calculate damage from number of cycles and their respective fatigue life with Palmgren-Miner rule
10	Calculate total fatigue life

5.2 Assumptions

There are several assumptions made in the calculation of the fatigue life.

It is assumed that the sub model is large enough for Saint-Venant's principle to be in effect, that the point that is analyzed is far enough from the boundary that the change in the boundary condition will not matter. However it is also assumed that the sub model is small enough that it can be looked at as a static model.

The fatigue is assumed to be high-cycle fatigue.

The The load-stress relation is assumed to be linear.

Both the rainflow counting and the Palmgren-Miner rule assumes that the order of the stress does not matter for the fatigue life.

Overall it is assumed that the established fatigue methods as the Palmgren-Miner rule, Rainflow counting, the S-N curve and so on give correct results.

The superposition principle is used, which is possible since the stress is linear.

5.3 Loads from dynamic system level simulation

First a model is made in Algoryx Momentum to simulate the appropriate loadings. From this simulation, the load history is extracted.

5.4 From loads to stress

The method to get the stress from the load was the most important step in this project, as it is the new one that created in order for the workflow to work, while the other steps are established methods. The approach was devised by Peter Norlindh at Algoryx Simulation AB.

The idea is that when analyzing high-cycle fatigue go from a dynamic system to a quasi-static one. Using a submodel of a size such that the dynamic effects can be neglected, as well as Saint-Venant's principle and the superposition principle, arbitrary combinations of loads can be applied on an arbitrary number of mechanical interfaces in order to calculate the stress resulting from those loads. In practice, this means converting the load history to stress by making a submodel with FEM and calculating a linear relation between the stress and the load.

By taking a submodel of the original model and then making a linear relation of the stress to put the load history into, the size of the calculations that have to be made and the data that has to be handled are dramatically reduced. Extracting the stress history directly from a complete model would be a much too complicated task, but by using a submodel the problem lessens considerably. It will still be a too long and complex procedure to project the load history onto the submodel to get the stress history, and even that may not even be possible considering that the load history can be very long and complicated. It is a very quick process to instead put the load history into the linear relation to get the stress history. This means that the method makes it possible to apply an arbitrary combination of loads and get the stress tensor corresponding to them almost instantaneously. This is what makes the fatigue analysis possible in practice.

This process is as follows: In Code_Aster, a submodel of the original model is made. This is a static model, unlike the one in Algoryx Momentum. The places where the submodel connects with the rest of the model are called mechanical interfaces and are where the submodel is subjected to the loads. One of the mechanical interfaces are locked in all degrees of freedom. The other interfaces are subjected to unit forces, one in each direction, x, y and z, as well as all three moments, m_x, m_y and m_z. The moments are around the x-, y-, and z-axis, respectively. These loads are applied evenly over the area of the mechanical interface. The loads are applied separately. Then the stress components resulting from the different loads are extracted from the places on the component where the stress should be concentrated, like notches or welds. From this data, a linear relation between the loads and the stress is calculated. That linear relation is:

$$\sigma_{ij} = L_{kl} C_{klij} \tag{8}$$

where k stands for the mechanical interfaces and $l = 1, ..., 6$ and correspond to the loads f_x, f_y, f_z, m_x, m_y and m_z, in that order. The number of mechanical interfaces depends on the case. The equation describes the stress for one point (node) on the sub model. The first $6k$ rows thus correspond to the first node, with the first six rows corresponding to the first interface. The first row is then the stress components caused by the unit force f_x the first interface is subjected to, and the the seventh is the stress components caused

by f_x on the second interface.

The load history is formatted so that each mechanical interface has its own file with loads. In it, each row is a time step, and the columns are the load components. By putting the load history in the linear relation corresponding to the extrapolation points, the stress is calculated. The stress is summed up in its components. For example, σ_{xx} on one node with loads from different interfaces are summed up, but the six stress components are separate. If there are n mechanical interfaces then the normal stress in the x direction is

$$\sigma_{xx} = \sum_{k=1}^{n} \sum_{l=1}^{6} L_{kl} C_{klxx}. \tag{9}$$

The Code Aster script writes a file with the constants for the linear relation between the load and the stress. In appendix A the code for the process of calculating the stress relation and saving the constants can be found. This is for the test case that is described in section 6.3.

5.4.1 The locked interface

When calculating the linear relation of the load and the stress, one interface is held still by constraints in all degrees of freedom. The other interfaces are subjected to unit forces and moments in all directions, one at a time.

It is not necessary to subject the locked interface to loads in order to get the stresses those loads would incur. The reason for this is that when the other interfaces are subjected to all possible loads, there will be an opposing force on the locked interface that holds it in place. Therefore, those loads will also affect the locked interface. The opposing force holding the interface in place will also cause stress. The stress that would happen if the locked interface was being affected directly by a load, with another interface being locked in its stead, will already have appeared, so there is no need to have separate cases to cover those loads.

5.5 From stress to weld model

The weld model used is the hot spot method. The extrapolation points used by the hot spot method are known, since the geometry of the component is known. The nodes closest to the points can then be found.

From the stress components the hot spot stress is extrapolated, using eq. 7. If the component does not have a weld, it is not necessary to use the hot spot method. Then the stress history on the nodes is used directly.

5.6 From weld model to rainflow counting

Using rainflow counting on the stress history from the hot spot method gives the number of cycles, stress amplitude, σ_a, and mean stress, σ_m. In fig. 2 the data that one gets

from a rainflow counting is shown in a histogram. The rainflow counting is done for each stress component, instead of putting all the stress components into one stress variable.

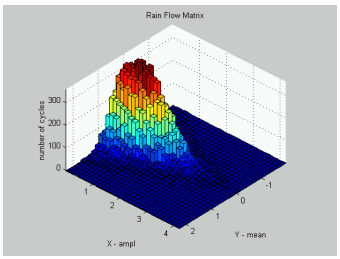


Figure 2 – The data that is gotten from a rainflow counting shown in a histogram[10].

Below is an example of using WAFO on the stress components.

```
for i in range(no_stress_comp):
        t_sigma = np.column_stack((t,sigma[:,i]))
        ts1 = wo.mat2timeseries(t_sigma)
        try:
                sig_tp = ts1.turning_points(h=-1, wavetype='astm')
        except ValueError:
                sig_tp = None
        try:
                sig_cp = sig_tp.cycle_astm()
        except AttributeError:
                sig_cp = None
```

The *sig_cp* variable then contains the stress amplitude, the mean stress and the number of cycles.

5.7 From rainflow counting to S-N curve

The stress amplitude and mean stress is then used to calculate the fatigue life for the respective cycles, using the Basquin relation from the S-N curve.

5.8 From S-N curve to Miner-Palmgren rule

Finally, the Palmgren-Miner rule is used to calculate the total fatigue life of the component. The point which has the shortest fatigue life is saved, with the value of the fatigue life.

6 Test cases

6.1 Case 1

The first case is a component shaped like an upside down T. The vertical part is welded to the horizontal bottom part. In fig. 3 the component is shown, with the coordinate system. The lines on the components shows where the sub model is.

The component can only move horizontally and is subjected to a loading $F(x) = 5 \cdot 10^6 \sin(50\pi t) + 3 \cdot 10^6 \sin(37\pi t)$, where t is the time.

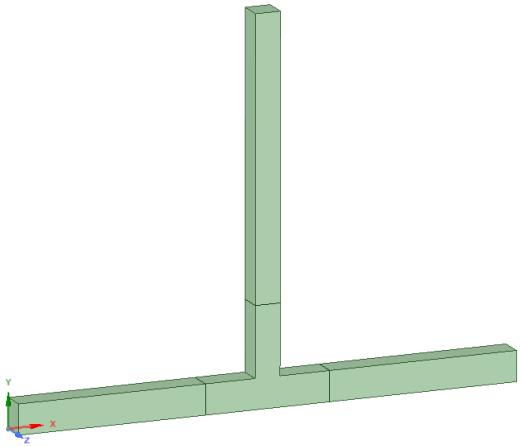


Figure 3 – The T-shaped component.

The sub model is shown in fig. 4. The top part is held still with all degrees of freedom locked, while the left and right mechanical interfaces are subjected to unit forces and moments in different directions.

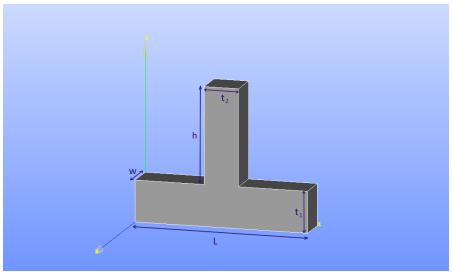


Figure 4 – The sub model of the T shaped component.

The table 4 shows the measurements for the sub model.

Table 4 – The measurements of the sub model of the component.

Measurement	[mm]
L	29.0
h	14.0
t_1	6.0
t_2	5.0
w	5.0

Table 5 shows the material properties for component.

Table 5 – This table contains the material properties of the component.

Property	Value
b	-0.13
σ'_f	$756 \cdot 10^6$
ν	0.29
E	$2.05 \cdot 10^{11}$ Pa

Since the component is symmetric, it is only necessary to analyze one of the welds, on one of the sides of the component. As such, the left side has been chosen to be analyzed. Both weld toes of that weld will be analyzed, the vertical as well as the horizontal one, with the hot spot method. The welds will be analyzed at points over their whole lengths and the ones with the shortest fatigue life will be found.

6.2 Case 1.5

This case is similar to case 1, but this component is not welded. Instead it has been moulded. The places where the welds where are now rounded, with radius $r = 3$ mm. Otherwise all is the same as in case 1. But since there is no weld to be modeled, the hot spot method will not be used to calculate the fatigue life. Instead the stress can be calculated directly in the area where the component is most likely to break. The sub model is shown in fig. 5.

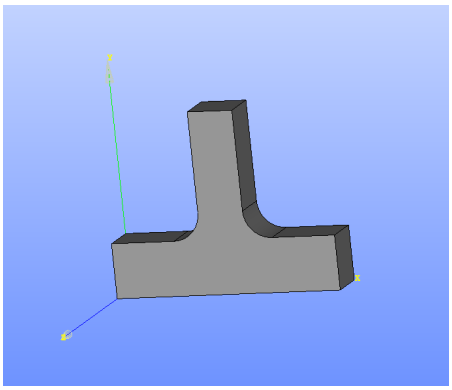


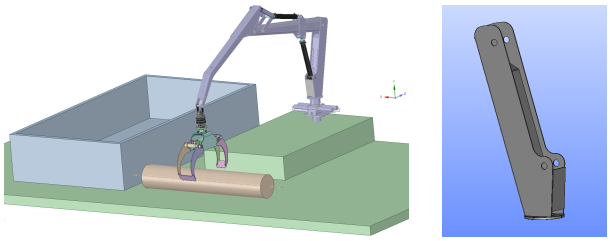
Figure 5 – The sub model of the T-shaped component that was not welded.

Again, the model is symmetric, so only the left side will be analyzed.

6.3 Case 2

The second case is a model of a crane, taken from GrabCAD[11]. The grapple[12] and the rotator[13] is from other files from GrabCAD and have been added to the crane.

The sub model was exported from Space Claim as a stp-file and imported into Code_Aster. In fig. 6 crane and the sub model is shown.



(a) The complete crane. **(b)** The submodel of the crane.

Figure 6 – The model and the submodel of the crane.

Table 6 shows the material properties for crane sub model. The work sequence is the crane picking up a tree trunk and placing it into a box.

Table 6 – This table contains the material properties of the crane submodel.

Property	Value
ρ	7870 kg/m^3
ν	0.29
E	$2.05 \cdot 10^{11}$ Pa
b	-0.13
σ'_f	$168 \cdot 10^6$

First the fatigue life around the upper hole was considered. In the hole a shaft is welded. It is around the shaft that the crane arm moves, so the sub model is completely static. The hot spot method was used to consider the boundary around the hole.

But another analysis was also made on the crane component. The best placement was found for a hook to be used to move the crane. By optimizing over a line on the surface of the component, $y = kz + m$, going from $p_1 = [0.23452, 1.7447, -2.2979]$ to $p_2 = [0.23452, 0.81952, -2.56391]$. Thus, x was constant on the surface. On that line the spot which had the longest fatigue life was found. That point should be the best placement for the hook, so that the stresses made by welding a hook in place would do as little damage as possible. The hook was not added to the geometry when the optimization took place.

The two outer points of the hook were analyzed using the hot spot method, under the assumption that the hook was 75 mm across. The shortest lifetime of the two points was saved. The placement which had the longest fatigue life on its shortest point was found as the optimal placement.

7 Results

7.1 Case 1

In table 7 the shortest fatigue life as well as the coordinates of that point are shown. The fatigue life is given in how many times the component would survive the work sequence.

Table 7 – The shortest fatigue life and the coordinates for that point.

N_f [work sequences]	x [mm]	y [mm]	z [mm]
$2.561 \cdot 10^9$	12.0	7.0	4.0

In fig. 7 the point with the shortest fatigue life is marked.

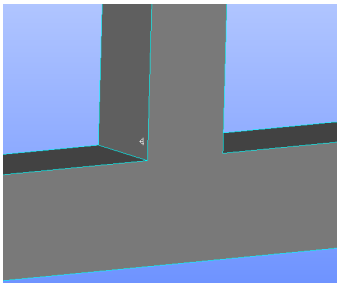


Figure 7 – The point which has the shortest fatigue life on the weld toe.

An example of the stress on the T-shaped component is shown in fig. 8, where the stress component σ_{yz} that is caused by the load f_z on the left interface is shown.

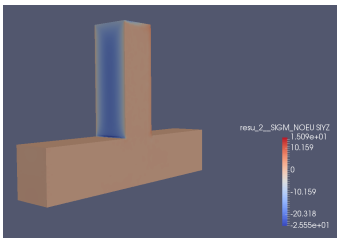


Figure 8 – The stress σ_{yz} from when the let interface is subjected to a unit load f_z.

7.2 Case 1.5

In table 8 the shortest fatigue life as well as the coordinates of that point are shown.

Table 8 – The shortest fatigue life and the coordinates for that point.

N_f [work sequences]	x [mm]	y [mm]	z [mm]
$6.539 \cdot 10^7$	12.0	9.0	3.429

In fig. 9 the point with the shortest fatigue life is marked.

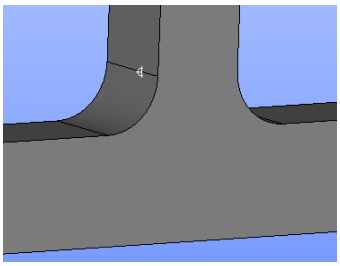


Figure 9 – The point with the shortest fatigue life.

An example of the stress on the component is shown in fig. 10, where the stress component σ_{xy} that is caused by the load f_x on the left interface is shown.

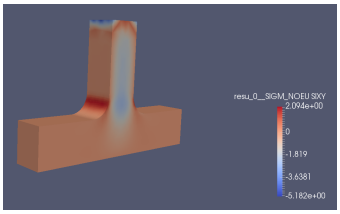


Figure 10 – The stress σ_{xy} from when the left interface is subjected to a unit load f_z.

7.3 Case 2

The results of the fatigue analysis around the hole with both work sequences can found in table 9. The work sequence where the crane is almost still is referred to loading case 1. The other, in which a tree trunk is lifted, is called work sequence 2.

Table 9 – The shortest fatigue life and the coordinates for that point.

N_f [work sequences]	x [m]	y [m]	z [m]
$4.107 \cdot 10^3$	0.23452	2.133874	-2.267016

In fig. 11, the component is shown with the failing point.

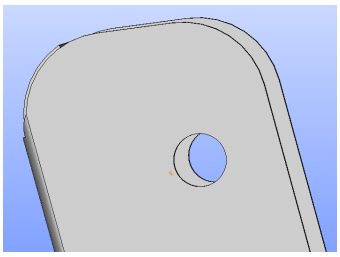


Figure 11 – The point with the shortest fatigue life around the hole.

Table 10 shows the optimal placement for the hook along the line, as well as the fatigue life for that placement.

Table 10 – The longest fatigue life and the coordinates for that point.

N_f [work sequences]	x [m]	y [m]	z [m]
$1.304 \cdot 10^8$	0.23452	1.7447	-2.2979

In fig. 12 the point for the placement of the hook's left corner is marked.

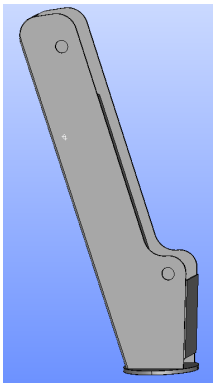


Figure 12 – The point where the hook's left corner should be placed for the longest fatigue life.

An example of the stress on the component is shown in fig. 13, where the stress component σ_{yy} that is caused by the load f_x on the upper interface is shown.

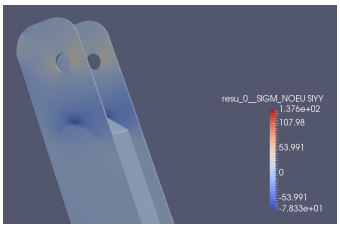


Figure 13 – The stress σ_{yy} from when the upper interface is subjected to a unit load f_z.

8 Discussion

The advantages of simulating the load history are obvious. It is possible to get the general loadings the component will be subjected to, instead of just a loading case that may or may not be representative of the general work sequence. Having a more realistic load will give a much more accurate result when doing a fatigue analysis. In order to use the load history from the simulation it is necessary to convert the load history to stress history, since it is impossible to get the stress directly from the simulation. There are two steps to that process. First there is the submodel, and secondly there is the linear relation. By taking a submodel the problem is made smaller since the submodel is smaller as well as much less complicated than the whole model as it is static. Even so, projecting the load history onto the submodel would be complicated and time-consuming considering how complicated the load history may be. The load history would need to be accurate to the real one, which can be very varied, and thus the simulation would need to go on for a long time to give a long load history. Therefore the linear relation is calculated in order to get a way to quickly convert the load history to stress. The relation also does not need to be calculated for the whole model. It is enough that it is done for the area around the weld toe, if there is a weld. This again reduces the amount of data that needs to be saved.

There are some known weaknesses of the methods used that could be considered: Palmgren-Miner is not based on a physical argument, but is a simplistic way to sum up the fatigue damage. It is difficult to avoid using it when doing fatigue analysis, since there are no physics-based way of calculating damage. However, a new way of calculating damage is far outside the scope of this project, which is supposed to use already known methods.

Rather than using the fatigue limit, the Basquin relation is also used for stresses below the limit. This way, the damage from stresses that should be non-damaging are taken into account, which is a good thing, since those stresses can cause damage if they come after stresses that have already caused a microcrack. Since the order of the stresses are not taken into account it is better to assume the stresses below the fatigue limit cause damage. That way, the actual damage will not be above the estimate, and the result should be conservative.

In the test cases the weld is not modeled. Instead the place where the weld should be is smooth as if component were just in one piece from the beginning. Since the geometry is supposed to be taken into account with the hot spot method, the geometry should match the true one as much as possible. So for a more accurate result, it may be a good idea to model the weld.

A possible improvement would be to streamline the process so that the geometry can be changed and the fatigue analysis for the change can run automatically, instead of spending time on meshing and renaming node groups. Currently every change in the geometry needs to be imported into Code_Aster and then meshed. Then mesh groups need to be defined in so that it is possible to subject the mechanical interfaces to loads and extract data from interesting parts. This is inconvenient for optimization.

Someone who does not have much knowledge of fatigue will have a hard time using the work flow. The method needs an engineer who can find the areas which should be analyzed. They need to first choose the right sub model, and then the right point on the sub model. The whole sub model would take far too long to check. This means that it is up to the engineer to have an understanding of where the weak points are, or to check similar constructs and where they usually fail. However, this is not much of a weakness. If someone is to work with analyzing fatigue, it is reasonable to expect that they have such an understanding.

It is possible to do an analysis on all the nodes of a larger area than what is done in the test cases. It is the calculation of the linear relation that takes time. When calculating the relation stress data from the nodes are extracted for all the different load cases. But over a larger area this would take a lot of time so it would not be feasible to analyze too large an area.

8.1 Case 1

In case 1, the component breaks in the weld toe on its vertical part. This is as expected, since the vertical part is slightly thinner than the horizontal one. The upper part is not supported by anything, so the stress should be higher on its base than it is on the horizontal part. However, the point where it fails is asymmetrical, which is unexpected considering the component's symmetric geometry and the even distribution of the load on the interface. This might be due to nothing more than slight variations over the weld toe.

There is some inaccuracies in the solution, since the nodes where the stress were taken from to extrapolate the hot spot stress is not exactly where the extrapolation points are. If the mesh is too coarse, then the closest node to the extrapolation point can be quite a distance away. Here the distance should not be large enough to affect the result much. In a real case, the best thing to do would be do make another version with a finer mesh and compare the result to make sure the mesh is not affecting the results. However, since this is just a test case to try the method, it is not necessary.

The size of the sub model may also affect the results. The stresses from the locked interface are very high close to it, so if the sub model has been chosen too small, then the stresses from the locked interface may interfere. When the hot spot method is used, extra caution is needed to make sure the extrapolation points do not end up to close to the locked interface. This is because of the high stresses that occurs when the component is held in place.

8.2 Case 1.5

This is the only test case when the hot spot method is not used. Since no extrapolation points are needed, there is no error from not having the node slightly off where it should be. But that does not mean that this case is more accurate than the first one. By taking the hot spot stress, the hot spot method attempts to take the geometry in consideration.

The fatigue life is shorter for this case than it is in case 1. This seems counterintuitive, since a weld is supposed to be a weak point. The most probable reason for this result is that the hot spot method uses the hot spot stress to estimate the geometric stress, and as such does not take into account the total stress. In case 1.5, no estimation of the stress behavior has been made. Instead the calculation is made directly after the stress in the points that have been deemed to be likely to fail.

Since the hot spot method is not used and no certain point need to be found to extrapolate from, the mesh is not especially fine. This can also be a reason why for the difference in result for case 1 and case 1.5.

8.3 Case 2

If the results of the fatigue life of the placement of the hook is compared with the fatigue life around the hole, it is obvious that it will break first around the hole. But it is more interesting to look at the cases separately to use the hook as an example on how to try to optimize the geometry, rather than dismiss the placement because it will break anyway in another spot.

Optimizing the placement of the hook is done without adding the hook to the geometry. Not having the geometry of the hook when calculating the stress may give an inaccurate result, especially since the hot spot method is supposed to take the geometry in consideration. However, the hook itself is not likely to have added much stress to the component. The place where the stress on the surface without the hook is largest should also be the place where the hook would do the most damage. The actual fatigue life may change, but it is likely not a large difference, unless one checks for when the crane is lifted with the hook.

The fatigue life is the only thing taken into account when optimizing the placement for the hook. Where it is most practical to have the hook, and where it would be easiest to move the crane is not, but that would of course be needed when doing this for a real case. It would not matter how long it would hold, if it would be impossible to use because of an impractical placement.

9 Conclusion

This fatigue analysis workflow has some rough edges to be filed off, but should give an realistic stress history that gives a more accurate fatigue life than commonly used ways of calculating fatigue.

In order to use this workflow to optimize the geometry more work is needed since it is not a very streamlined procedure in the present. However, at the present it works and produces reasonable results.

10 References

[1] Schijve, J., *Fatigue of structures and materials*, 2nd ed. Springer, 2009.

[2] Sundström, B., *Handbok och formelsamling i hållfasthetslära*, 2nd ed. Södertälje: Fingraf AB, 1999.

[3] Reddy, J. N., *An introduction to continuum mechanics*. Cambridge University Press, 2008.

[4] Suresh, S., *Fatigue of materials*, 2nd ed. Cambridge University Press, 2004.

[5] European Committee for Standardization, *Eurocode 3: Design of steel structures - part1-9: Fatigue*, Brussels, Belgium, 2005.

[6] Irvine, T., *Rainflow cycle counting in fatigue analysis*, 2011-08-26. [Online]. Available: https://pdfs.semanticscholar.org/e866/3ef9235a09845b0eb065531b036ee207c89b.pdf.

[7] Eriksson, Å. and Lignell, A.-M., *Svetsutvärdering med fem*. Industrilitteratur AB, 2002.

[8] EDF R&D, *Code_Aster - Analysis of Structures and Thermomechanics for Studies & Research*, EDF, Clamart, France, 2000. [Online]. Available: https://www.code-aster.org/UPLOAD/DOC/Presentation/plaquette_aster_en.pdf.

[9] WAFO-group, *WAFO - a matlab toolbox for analysis of random waves and loads - a tutorial*, ISBN XXXX, Math. Stat., Center for Math. Sci., Lund Univ., Lund, Sweden, 2000. [Online]. Available: http://www.maths.lth.se/matstat/wafo.

[10] Nieslony, A., *Grainflow counting algorithm*, 2017-12-19. [Online]. Available: https://se.mathworks.com/matlabcentral/fileexchange/3026-rainflow-counting-algorithm?.

[11] Doxsyde, *Guindaste - STEP/IGES, STL - 3D CAD model*, 2017-12-14. [Online]. Available: https://grabcad.com/library/guindaste.

[12] Per, *Grapple - STL, STEP/IGES - 3D CAD model*, 2017-12-14. [Online]. Available: https://grabcad.com/library/grapple.

[13] Marcelo, *Rotator - STEP/IGES - 3D CAD model*, 2017-12-14. [Online]. Available: https://grabcad.com/library/guindaste.

A Appendix

```
# Fatigue analysis on sub model of crane component

DEBUT();

import numpy as np
import wafo.objects as wo

# Initialize parameters
no_load_cases = 12
resu = [None]*no_load_cases
tab = [None]*no_load_cases

# First left interface, all load cases, then right interface
interface = ['h1','h1','h1','h1','h1','h1',
                'h2', 'h2', 'h2','h2', 'h2', 'h2']
# The node which transfers momentum to 3D model (not used in cases fx, fy, fz)
node = ['none', 'none', 'none', 'node1', 'node1', 'node1',
                'none', 'none', 'none', 'node2', 'node2', 'node2']

# Read mesh file
mesh0=LIRE_MAILLAGE(FORMAT='MED',);
# Make mesh groups to stand alone nodes
mesh=CREA_MAILLAGE(MAILLAGE=mesh0,
                    CREA_POI1=(_F(NOM_GROUP_MA='node1',
                                  GROUP_NO='node1',),
                               _F(NOM_GROUP_MA='node2',
                                  GROUP_NO='node2',),),);

# Assign model
model=AFFE_MODELE(MAILLAGE=mesh,
                  AFFE=(_F(TOUT='OUI',
                           PHENOMENE='MECANIQUE',
                           MODELISATION='3D',),
                        _F(GROUP_MA=('node1','node2',),
                           PHENOMENE='MECANIQUE',
                           MODELISATION='DIS_TR',),),);

# Define material
mat=DEFI_MATERIAU(ELAS=_F(E=6.895E+10,
                          NU=0.33,
                          RHO=2705.0,),);

# Apply material
ap_mat=AFFE_MATERIAU(MAILLAGE=mesh,
                     AFFE=_F(GROUP_MA='Solid',
                             MATER=mat,),);

# Boundary condition
bc=AFFE_CHAR_MECA(MODELE=model,
                  DDL_IMPO=_F(GROUP_MA='bottom',
                              DX=0.0,
                              DY=0.0,
                              DZ=0.0,),);
# Loop over the load cases
for i in range(no_load_cases):

        # Define boundary conditions and load for different load cases
        if i == 0 or i == 6:
                load=AFFE_CHAR_MECA(MODELE=model,
```

```
60                          FORCE_FACE=_F(GROUP_MA=interface[i],
61                                        FX=1.0,),);
62          elif i == 1 or i == 7:
63                  load=AFFE_CHAR_MECA(MODELE=model,
64                          FORCE_FACE=_F(GROUP_MA=interface[i],
65                                        FY=1.0,),);
66          elif i == 2 or i == 8:
67                  load=AFFE_CHAR_MECA(MODELE=model,
68                          FORCE_FACE=_F(GROUP_MA=interface[i],
69                                        FZ=1.0,),);
70          elif i == 3 or i == 9:
71                  load=AFFE_CHAR_MECA(MODELE=model,
72                              LIAISON_SOLIDE=_F(GROUP_NO=(interface[i],node[i],),),
73                              FORCE_NODALE=_F(GROUP_NO=node[i],
74                                   MX=1.0,),);
75          elif i == 4 or i == 10:
76                  load=AFFE_CHAR_MECA(MODELE=model,
77                              LIAISON_SOLIDE=_F(GROUP_NO=(interface[i],node[i],),),
78                              FORCE_NODALE=_F(GROUP_NO=node[i],
79                                   MY=1.0,),);
80          elif i == 5 or i == 11:
81                  load=AFFE_CHAR_MECA(MODELE=model,
82                              LIAISON_SOLIDE=_F(GROUP_NO=(interface[i],node[i],),),
83                              FORCE_NODALE=_F(GROUP_NO=node[i],
84                                   MZ=1.0,),);
85
86          # Apply characteristics to the two nodes
87          cara=AFFE_CARA_ELEM(MODELE=model,
88                       DISCRET=_F(CARA='K_TR_D_N',
89                              GROUP_MA=('node1','node2',),
90                              VALE=(0.1,0.1,0.1,0.1,0.1,0.1,),),);
91
92          # Solve the problem
93          resu[i]=MECA_STATIQUE(MODELE=model,
94                     CHAM_MATER=ap_mat,
95                     CARA_ELEM=cara,
96                     EXCIT=(_F(CHARGE=bc,),
97                            _F(CHARGE=load,),),);
98
99          # Calculate stress
100         resu[i]=CALC_CHAMP(reuse =resu[i],
101                         RESULTAT=resu[i],
102                         CONTRAINTE='SIGM_NOEU',
103                         FORCE='FORC_NODA',);
104
105         # Write result to med file, in case you want to look at it in ParaViS
106         IMPR_RESU(FORMAT='MED',
107           RESU=_F(RESULTAT=resu[i],
108                   INFO_MAILLAGE='OUI',
109                   TOUT='OUI',
110                   GROUP_MA='Solid',),);
111
112
113         # Destroy old parameters
114         if i < no_load_cases-1:
115                 DETRUIRE(CONCEPT = _F (NOM = load,) ,INFO=1, );
116                 DETRUIRE(CONCEPT = _F (NOM = cara,) ,INFO=1, );
117
118
119 FIN();
```

B Appendix

```
# Fatigue analysis on submodel of crane component
#
# Reads the load history from files crane_load1_moving.txt, crane_load2_moving.txt
# from the two side mechanical interfaces and the time from crane_times_moving.txt.
#
# Calculates a linear relation between the load and the stress and saves the
# constants  for the boundary around the upper hole as well as the coordinates
# for the nodes
#
# Uses the hot spot method to calculate fatigue life at weld on the hole's boundary.
# Writes fatigue life to a file, results.txt.


# Function to extract stress components for nodes from table
def extractStress(table):
        table0 = table.EXTR_TABLE()
        pytab = table0.values()
        # Extract stress coefficients
        sxx=pytab['SIXX']
        syy=pytab['SIYY']
        szz=pytab['SIZZ']
        sxy=pytab['SIXY']
        sxz=pytab['SIXZ']
        syz=pytab['SIYZ']

        return sxx, syy, szz, sxy, sxz, syz


# Function to extract coordinates for nodes from table
def extractCoordinates(table):
        table0 = table.EXTR_TABLE()
        pytab = table0.values()

        # Extract coordinates for nodes
        x=pytab['COOR_X']
        y=pytab['COOR_Y']
        z=pytab['COOR_Z']

        no_nodes = len(x)
        coord=np.zeros((3, no_nodes))
        coord[:][0]=x
        coord[:][1]=y
        coord[:][2]=z
        coord=np.transpose(coord)

        return coord


# Function to format stress
def writeStressToInterface(sxx, syy, szz, sxy, sxz, syz, i):
        # left interface
        i1 = [[sxx[0][i], syy[0][i], szz[0][i], sxy[0][i], sxz[0][i], syz[0][i]],
                [sxx[1][i], syy[1][i], szz[1][i], sxy[1][i], sxz[1][i], syz[1][i]],
                [sxx[2][i], syy[2][i], szz[2][i], sxy[2][i], sxz[2][i], syz[2][i]],
                [sxx[3][i], syy[3][i], szz[3][i], sxy[3][i], sxz[3][i], syz[3][i]],
                [sxx[4][i], syy[4][i], szz[4][i], sxy[4][i], sxz[4][i], syz[4][i]],
                [sxx[5][i], syy[5][i], szz[5][i], sxy[5][i], sxz[5][i], syz[5][i]]]

        # right interface
```

```
60          i2 = [[sxx[6][i], syy[6][i], szz[6][i], sxy[6][i], sxz[6][i], syz[6][i]],
61                  [sxx[7][i], syy[7][i], szz[7][i], sxy[7][i], sxz[7][i], syz[7][i]],
62                  [sxx[8][i], syy[8][i], szz[8][i], sxy[8][i], sxz[8][i], syz[8][i]],
63                  [sxx[9][i], syy[9][i], szz[9][i], sxy[9][i], sxz[9][i], syz[9][i]],
64                  [sxx[10][i],syy[10][i],szz[10][i],sxy[10][i],sxz[10][i],syz[10][i]],
65                  [sxx[11][i],syy[11][i],szz[11][i],sxy[11][i],sxz[11][i],syz[11][i]]]
66
67          return i1,i2
68
69
70  # Function to write node coordinates to file
71  def writeNodeCoordtoFile(coord, filename):
72
73          filename = '/home/anna/Code_Aster/New_crane/' + filename
74          f = open(filename,'w+')
75          np.savetxt(f, coord)
76          f.close()
77
78          return
79
80
81  # Function to write stress coefficients C to file
82  def writeCtoFile(sxx, syy, szz, sxy, sxz, syz, filename):
83
84          no_loads = 6          # Number of loads on one interface
85          no_stress_comp = 6
86          no_nodes = len(sxx[0])
87          i1 = np.zeros(shape=[no_nodes,no_loads,no_stress_comp])
88          i2 = np.zeros(shape=[no_nodes,no_loads,no_stress_comp])
89          filename = '/home/anna/Code_Aster/New_crane/' + filename
90          f = open(filename,'w+')
91          # Loop over nodes
92          for i in range(no_nodes):
93                  # Format the stress coefficients
94                  i1[i], i2[i] = writeStressToInterface(sxx,syy,szz,sxy,sxz,syz,i)
95                  # Write stress coefficients to file
96                  np.savetxt(f, i1[i])
97                  np.savetxt(f, i2[i])
98          f.close()
99
100         return i1, i2
101
102
103 # Function to calculate stress from load history for one node
104 def getStress1node(i1_node, i2_node, load_left, load_right):
105         no_stress_comp = 6
106         no_timesteps = len(load_left)
107         sigma_ij = np.zeros(shape=[no_timesteps,no_stress_comp])
108         # Loop over timesteps
109         for j in range(no_timesteps):
110                 load_l = load_left[j]
111                 load_r = load_right[j]
112                 for i in range(no_stress_comp):
113                         sigma_ij[j] += load_l[i]*np.array(i1_node[i])
114                         sigma_ij[j] += load_r[i]*np.array(i2_node[i])
115         return sigma_ij
116
117
118 # Function that uses rainflow counting to calculate fatigue life.
119 # It calculates the damages from the stress components seperately.
120 def rainflow(t, sigma):
121         # Define constants
```

```
122         b = -0.13
123         sig_f = 756E6
124         no_stress_comp = 6
125 
126         D =  np.zeros(shape=[no_stress_comp, 1])
127         # Calculate damage from each stress component
128         for i in range(no_stress_comp):
129                 t_sigma = np.column_stack((t,sigma[:,i]))
130                 # Turn the array into a time series
131                 ts1 = wo.mat2timeseries(t_sigma)
132                 # Try to find turning points
133                 try:
134                         sig_tp = ts1.turning_points(h=-1, wavetype='astm')
135                 except ValueError:
136                         sig_tp = None
137                 # Try to find stress amplitude, mean stress and number of cycles
138                 try:
139                             sig_cp = sig_tp.cycle_astm()
140                 except AttributeError:
141                             sig_cp = None
142                 # Calculate fatigue life for cycles and sum damage
143                 if sig_cp is not None:
144                         N_f = 0.5*(sig_cp[:,0]/np.abs(sig_f - sig_cp[:,1]))**(1/b)
145                         D[i] = np.sum(sig_cp[:,2]/N_f)
146         # Calculate total fatigue life for node
147         N_f_tot = 1/np.sum(D)
148 
149         return N_f_tot
150 
151 
152 # Function to use the hot spot method in order to calculate fatigue life on the weld
153 # around a hole on the component.
154 # The hot spot method extrapolates the stress on a point on the the toe of the weld
155 # from three points, 0.4t, 0.9t and 1.4t away from it.
156 def calculateFatigue(i1, i2, coord, load_left, load_right, t, mid_circle, radius):
157 
158         no_points = 20        # Number of point whose fatigue life are to be checked
159         N_f = np.zeros(shape=[no_points,1])        # For saving fatigue life
160         points = np.zeros(shape=[no_points,3])        # The points were the fatigue life is checked
161         no_timesteps = len(t)
162         no_stress_comp = 6
163         sigma_ij = np.zeros(shape=[3,no_timesteps, no_stress_comp])
164         x = mid_circle[0]        # x is constant
165         dphi = 2*np.pi/(no_points-1)
166 
167         # Loop over points on hole boundary
168         for i in range(no_points):
169                 phi = i * dphi
170                 # Loop over extrapolation points
171                 for j in range(3):
172                         # Extrapolation points
173                         y = mid_circle[1] + radius[j]*np.sin(phi)
174                         z = mid_circle[2] + radius[j]*np.cos(phi)
175                         # Find closest node
176                         p = np.array([x, y, z])
177                         diff = np.abs(coord-p)
178                         node = np.sum(diff, axis=1).argmin()
179                         # Calculate stress for that node
180                         sigma_ij[j] = getStress1node(i1[node],i2[node],load_left,load_right)
181 
182                 # Calculate the hot spot stress and, using it, the fatigue life
183                 sigma_hs = 2.52*sigma_ij[0] - 2.24*sigma_ij[1] + 0.72*sigma_ij[2]
```

```
184                 #sigma_hs = 1.67*sigma_ij[0] - 0.67*sigma_ij[1]
185                 N_f[i] = rainflow(t, sigma_hs)
186                 points[i][0] = mid_circle[1] + 33.5E-3*np.sin(phi)
187                 points[i][1] = mid_circle[2] + 33.5E-3*np.cos(phi)
188
189         return N_f, points
190
191
192 # Write the results to a file
193 def writeResultsToFile(N_f, points):
194
195         f_res = open('/home/anna/Code_Aster/New_crane/results_hs2_moving.txt','w+')
196         f_res.write('# Fatigue life, y, z'+'\n')
197         for i in range(len(N_f)):
198                 f_res.write('%e ' % (N_f[i]))
199                 f_res.write('%e %e\n' % (points[i][0], points[i][1]))
200         f_res.close()
201
202         return
203
204
205 ########################################
206 # Start Code_Aster
207 ########################################
208
209 POURSUITE(PAR_LOT='NON',);
210
211 import numpy as np
212 import wafo.objects as wo
213
214 # Initialize parameters
215 no_load_cases = 12
216 sxx = [None]*no_load_cases
217 syy = [None]*no_load_cases
218 szz = [None]*no_load_cases
219 sxy = [None]*no_load_cases
220 sxz = [None]*no_load_cases
221 syz = [None]*no_load_cases
222
223 # Loop over the load cases
224 for i in range(no_load_cases):
225
226         # Tables with values from hole 1
227         # Stress components
228         tab1=POST_RELEVE_T(ACTION=_F(OPERATION='EXTRACTION',
229                                      INTITULE='h1',
230                                      RESULTAT=resu[i],
231                                      NOM_CHAM='SIGM_NOEU',
232                                      GROUP_NO=('extval',),
233                                      TOUT_CMP='OUI',),);
234
235         # Write table to file
236         IMPR_TABLE(TABLE=tab1,
237                    FORMAT='TABLEAU',);
238
239
240         # Extracting values from tables
241         sxx[i], syy[i], szz[i], sxy[i], sxz[i], syz[i]=extractStress(tab1)
242         coord = extractCoordinates(tab1)
243
244         # Destroy old parameters
245         if i < no_load_cases-1:
```

```
246                 DETRUIRE(CONCEPT = _F (NOM = tab1,) ,INFO=1, );
247
248 ########################################
249 # Post-processing
250 ########################################
251
252 # Write coordinates and coefficients to file
253 writeNodeCoordtoFile(coord, 'nodes_hs.txt')
254 i1,i2 = writeCtoFile(sxx, syy, szz, sxy, sxz, syz, 'C_hs.txt')
255
256 # Read load history from file
257 load_left = np.loadtxt('/home/anna/Code_Aster/New_crane/crane_load1_moving.txt')
258 load_right = np.loadtxt('/home/anna/Code_Aster/New_crane/crane_load2_moving.txt')
259
260 # Read time from file
261 t = np.loadtxt('/home/anna/Code_Aster/New_crane/crane_times_moving.txt')
262
263 # Calculate fatigue on weld around hole on component
264 mid_circle = [0.23452, 2.14794, -2.29742]        # coord for origin of hole
265 radius = [0.036676, 0.040646, 0.044616]        # radius for extrapolation points
266 N_f, points = calculateFatigue(i1,i2,coord,load_left,load_right,t,mid_circle,radius)
267
268 # Write result to file
269 writeResultsToFile(N_f, points)
270
271 FIN();
```