






Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in: <http://oatao.univ-toulouse.fr/28245>

To cite this version:

Pantalé, Olivier  and Tize Mha, Pierre  and Tongne, Amèvi 
An efficient artificial neural network based non-linear flow law: towards the implementation into a radial return integration scheme.
(2021) In: Computational Science and AI in Industry (CSAI 2021),
7 June 2021 - 9 June 2021 (Trondheim, Norway). (Unpublished)

Any correspondence concerning this service should be sent
to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

An efficient Artificial Neural Network based non-linear flow law

Towards the implementation into a radial return integration scheme

Olivier Pantalé, Pierre Tize-Mha, Amèvi Tongne

*LGP, INP/ENIT, Université de Toulouse, 47 Av d'Azereix, Tarbes, France 65016
olivier.pantale@enit.fr*

Computational Sciences and AI in Industry
Virtual Conference

7-9 June 2021

- 1 Introduction and general purpose
- 2 Artificial Neural Network set-up
- 3 Training of the Artificial Neural Network and performance evaluation
- 4 Implementation of the Artificial Neural Network within Abaqus Explicit
- 5 Numerical results
- 6 Conclusions and perspectives

Identification of constitutive laws from experiments



Figure 1: Gleeble 3500 thermomechanical simulator

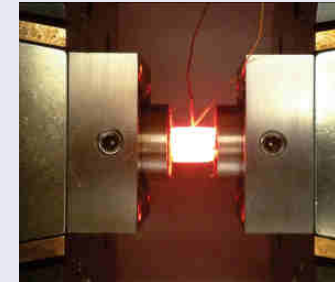


Figure 2: Specimen under compression at high temperature

- Realization of the experimental tests
- Mathematical formulation of a flow law
- Non-linear behavior depending on ε^P , $\dot{\varepsilon}^P$, T
- Identification of the parameters of the flow law
- Numerical implementation of the flow law in a FEM code
- Validation by reproducing numerically the experiments
- **Key idea** : Use an Artificial Neural Network to replace the mathematical formulation and perform simulations directly from experiments

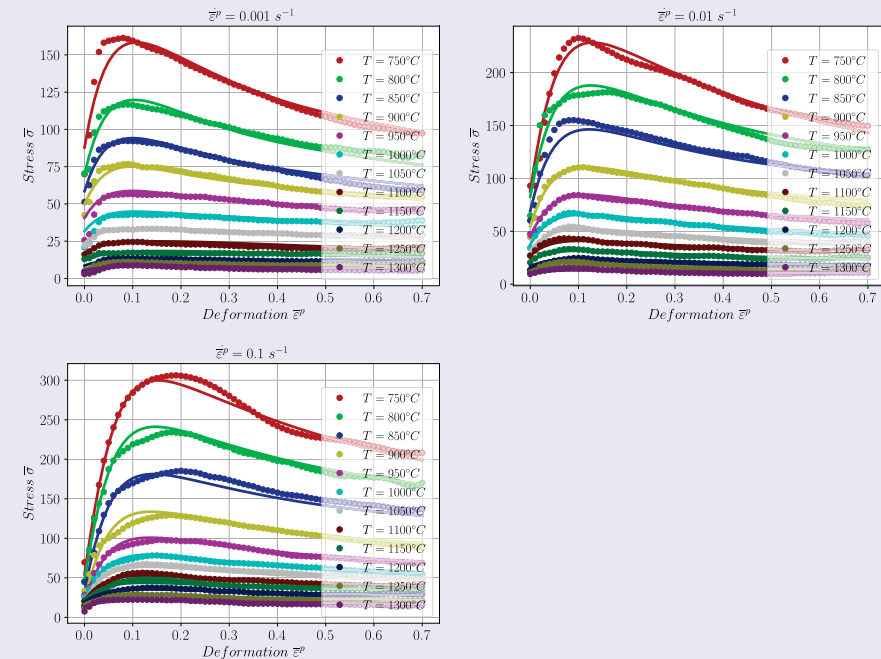


Figure 3: Flow stress data from experiments $\sigma(\varepsilon^P, \dot{\varepsilon}^P, T)$

Usual approach to define a new flow law in Abaqus/Explicit

- Abaqus Explicit allows to define user behavior laws through **FORTRAN subroutines VUMAT or VUHARD**.
- Both approaches require the **definition of the flow stress** depending on the plastic strain ε^p , the plastic strain rate $\dot{\varepsilon}^p$ and the temperature T :

$$\sigma(\varepsilon^p, \dot{\varepsilon}^p, T), \quad (1)$$

- and the definition of the 3 **derivatives** of the flow stress : $\partial\sigma/\partial\varepsilon^p$, $\partial\sigma/\partial\dot{\varepsilon}^p$ and $\partial\sigma/\partial T$

Proposal of using an Artificial Neural Network for defining the flow law in Abaqus/Explicit

- Set-up an Artificial Neural Network for **computing the flow stress** σ from ε^p , $\dot{\varepsilon}^p$, T .
- Use the experimental data performed on the Gleeble thermomechanical simulator to **train** the Artificial Neural Network.
 - Data is composed of the measured plastic strain ε^p , plastic strain rate $\dot{\varepsilon}^p$, temperature T and stress σ .
- **Implement an optimized ANN** written in FORTRAN language as a VUHARD subroutine for Abaqus/Explicit software.
- Use the Artificial Neural Network to compute the **flow stress** $\sigma(\varepsilon^p, \dot{\varepsilon}^p, T)$.
- Use the same Artificial Neural Network to compute the **flow stress derivatives** : $\partial\sigma/\partial\varepsilon^p$, $\partial\sigma/\partial\dot{\varepsilon}^p$ and $\partial\sigma/\partial T$.

- 1 Introduction and general purpose
- 2 Artificial Neural Network set-up
- 3 Training of the Artificial Neural Network and performance evaluation
- 4 Implementation of the Artificial Neural Network within Abaqus Explicit
- 5 Numerical results
- 6 Conclusions and perspectives

General structure of the proposed Artificial Neural Network

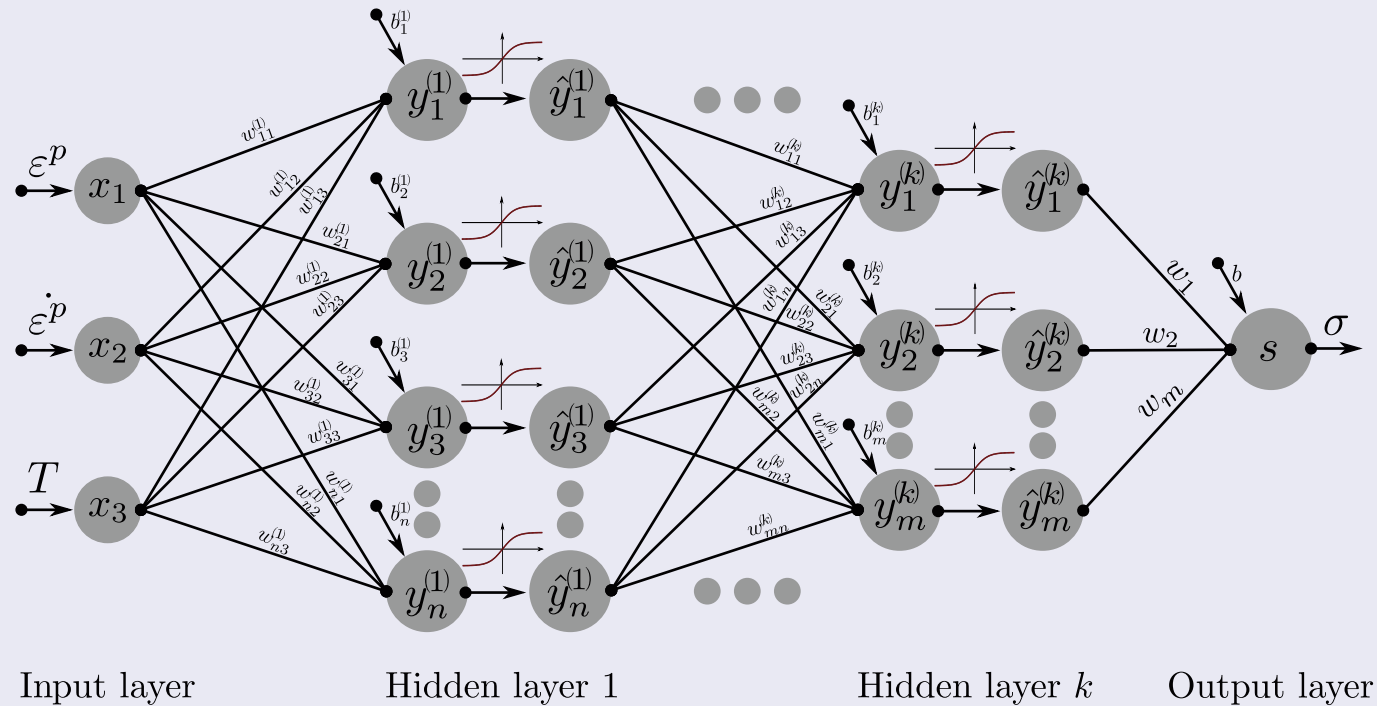


Figure 4: General structure of a multi-layer perceptron

- 3 inputs : plastic strain ε^p , plastic strain rate $\dot{\varepsilon}^p$, temperature T .
- 1 output : the von Mises flow stress σ .
- 1 or 2 hidden layers in the proposed Artificial Neural Network.
- Activation functions (tanh or sig) **only for the neurons in the hidden layers.**

Artificial Neural Network governing equations

- The output of all hidden layer neurons is given by:

$$\vec{\hat{y}}^{(k)} = f^{(k)} \left(\mathbf{w}^{(k)} \cdot \vec{x} + \vec{b}^{(k)} \right) \quad (2)$$

where

$$f^{(k)}(x) = \text{sig}(x) = \frac{1}{1 + \exp(-x)}, \quad \text{sig}'(x) = \frac{\exp(x)}{(1 + \exp(x))^2} \quad (3)$$

or

$$f^{(k)}(x) = \text{tanh}(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}, \quad \text{tanh}'(x) = 1 - \text{tanh}^2(x) \quad (4)$$

- The output of the Artificial Neural Network is given by:

$$s = \vec{w}^T \cdot \vec{\hat{y}}^{(l)} + b \quad (5)$$

where l is the last hidden layer of the Artificial Neural Network.

Pre and post-processing of data

- The Artificial Neural Network are set up to treat values with a limited amplitude.
- We need pre-process and post-process the values of ε^p , $\dot{\varepsilon}^p$, T and σ in the range $[0, 1]$.
- We also apply a special treatment to $\dot{\varepsilon}^p$ because of its logarithmic variation.

$$\vec{x} = \begin{cases} x_1 = \frac{\varepsilon^p - [\varepsilon^p]_{min}}{[\varepsilon^p]_{max} - [\varepsilon^p]_{min}} \\ x_2 = \frac{\ln(\dot{\varepsilon}^p / \dot{\varepsilon}_0) - [\ln(\dot{\varepsilon}^p / \dot{\varepsilon}_0)]_{min}}{[\ln(\dot{\varepsilon}^p / \dot{\varepsilon}_0)]_{max} - [\ln(\dot{\varepsilon}^p / \dot{\varepsilon}_0)]_{min}} \\ x_3 = \frac{T - [T]_{min}}{[T]_{max} - [T]_{min}} \end{cases} \quad (6)$$

where $[]_{min}$ and $[]_{max}$ are the boundaries of the range of the corresponding field.

- Concerning the flow stress, we apply:

$$\sigma = ([\sigma]_{max} - [\sigma]_{min}) s + [\sigma]_{min} \quad (7)$$

Derivatives computation by the Artificial Neural Network

- Chain rule can be used to compute the **derivative** of the output s network with respect to the input \vec{x} .
- Derivative expression depends on the activation function used and the number of hidden layer.
- For 2 hidden layers and a **sig** activation function for both layers we obtain:

$$\vec{s}' = \mathbf{w}^{(1)T} \cdot \left[\mathbf{w}^{(2)T} \cdot \left(\frac{\vec{w} \circ \exp(-\vec{y}^{(2)})}{[1 + \exp(-\vec{y}^{(2)})]^2} \right) \circ \left(\frac{\exp(-\vec{y}^{(1)})}{[1 + \exp(-\vec{y}^{(1)})]^2} \right) \right] \quad (8)$$

- So the derivatives of the flow stress are given by:

$$\begin{cases} \partial\sigma/\partial\varepsilon^p = s'_1 \frac{[\sigma]_{max} - [\sigma]_{min}}{[\varepsilon^p]_{max} - [\varepsilon^p]_{min}} \\ \partial\sigma/\partial\dot{\varepsilon}^p = s'_2 \frac{[\sigma]_{max} - [\sigma]_{min}}{[\dot{\varepsilon}^p]_{max} - [\dot{\varepsilon}^p]_{min}} \\ \partial\sigma/\partial T = s'_3 \frac{[\sigma]_{max} - [\sigma]_{min}}{[T]_{max} - [T]_{min}} \end{cases} \quad (9)$$

- 1 Introduction and general purpose
- 2 Artificial Neural Network set-up
- 3 Training of the Artificial Neural Network and performance evaluation**
- 4 Implementation of the Artificial Neural Network within Abaqus Explicit
- 5 Numerical results
- 6 Conclusions and perspectives

Training of the Artificial Neural Network with the Johnson-Cook model

- The general formulation $\sigma^y(\varepsilon^p, \dot{\varepsilon}^p, T)$ is given by the following equation:

$$\sigma^y = \left(A + B\varepsilon^{p^n} \right) \left[1 + c \ln \left(\frac{\dot{\varepsilon}^p}{\dot{\varepsilon}_0} \right) \right] \left[1 - \left(\frac{T - T_0}{T_m - T_0} \right)^m \right] \quad (10)$$

- Selected material for training the Artificial Neural Network is a 42CrMo4 steel.

E (GPa)	ν	A (MPa)	B (MPa)	C	n	m	$\dot{\varepsilon}_0$ (s ⁻¹)	T_0 °C	T_m °C
206.9	0.29	806	614	0.0089	0.168	1.1	1	20	1540

Table 1: Material properties of the 42CrMo4 steel

- Training dataset** contains 2 520 datapoints defined by:
 - 70 equidistant plastic strain values : $\varepsilon^p \in [0, 1]$
 - 6 plastic strain rates : $\dot{\varepsilon}^p \in [1, 10, 50, 500, 5\,000, 50\,000]$
 - 6 temperatures : $T \in [20, 100, 200, 300, 400, 500]$
- Test dataset** contains 5 000 datapoints randomly generated within the ranges:
 - $\varepsilon^p \in [0, 1]$
 - $\dot{\varepsilon}^p \in [1, 50\,000]$
 - $T \in [20, 500]$

Global results analysis

- Models trained using the Python Tensorflow library
- Adaptive Moment Estimation (ADAM)
- Fixed number of 50 000 epochs
- CPU time is 1 hour on a Dell XPS 13 laptop

$$E_{\text{RMS}} = \sqrt{\frac{1}{N} \sum_{i=1}^N (\sigma_i - \sigma_i^y)^2} \quad (11)$$

$$\Delta \square = \frac{1}{N} \sum_{i=1}^N \left| \frac{\square_i^e - \square_i^p}{\square_i^e} \right| \quad (12)$$

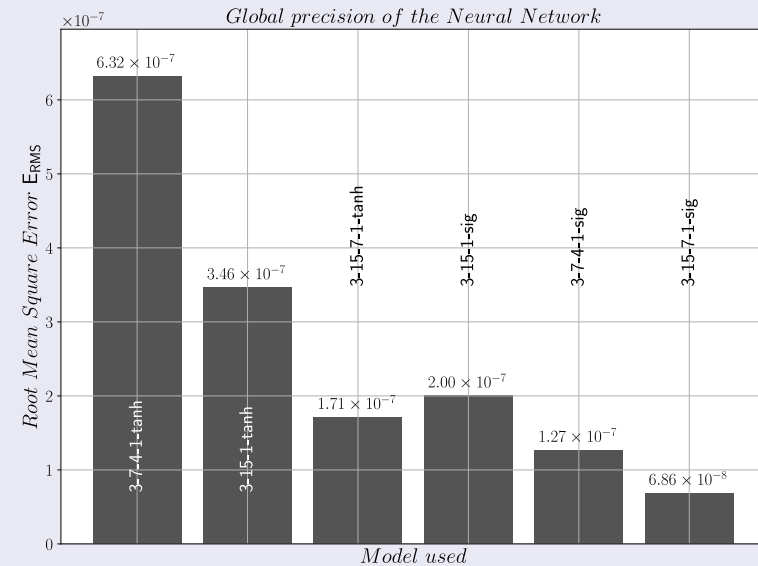


Figure 5: Convergence of parameters

Model	N	E _{RMS} × 10 ⁻⁷	Δσ %	Δ(∂σ/∂ε ^p) %	Δ(∂σ/∂ε ^p) %	Δ(∂σ/∂T) %
3-7-4-1-tanh	65	6.32	0.038	1.977	0.792	0.556
3-15-1-tanh	78	3.46	0.039	1.506	0.269	0.371
3-15-7-1-tanh	180	1.71	0.030	0.519	0.380	0.408
3-15-1-sig	78	2.00	0.030	0.686	0.521	0.675
3-7-4-1-sig	65	1.27	0.024	0.670	0.415	0.499
3-15-7-1-sig	180	0.68	0.011	0.247	0.199	0.256

Table 2: Global performance analysis of the ANN during the training phase

Precision of the 3-7-4-1 tanh Artificial Neural Network

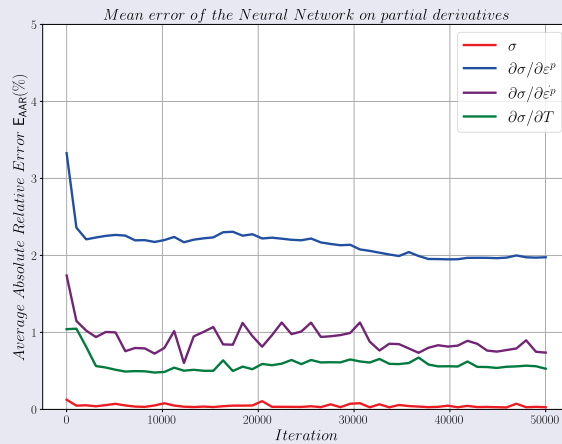


Figure 6: Convergence of parameters

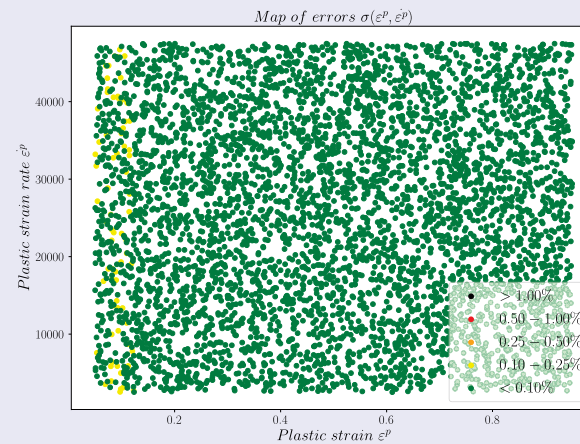


Figure 7: Map error of σ

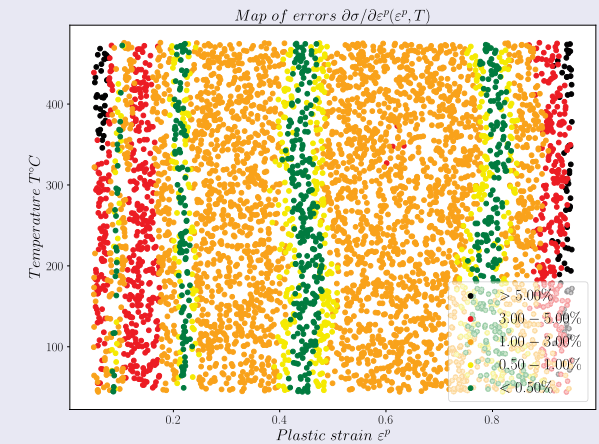


Figure 9: Map error of $\partial\sigma/\partial\dot{\epsilon}^p$

σ	$R = 0.99999342$
$\partial\sigma/\partial\epsilon^p$	$R = 0.99888560$
$\partial\sigma/\partial\dot{\epsilon}^p$	$R = 0.99994476$
$\partial\sigma/\partial T$	$R = 0.99708604$

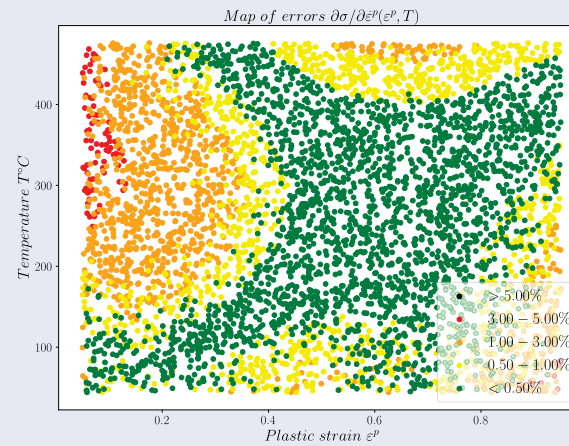


Figure 8: Map error of $\partial\sigma/\partial\epsilon^p$

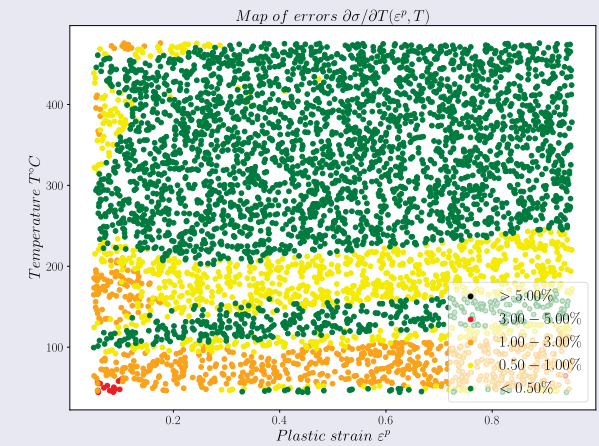


Figure 10: Map error of $\partial\sigma/\partial T$

Precision of the 3-15-7-1 sig Artificial Neural Network

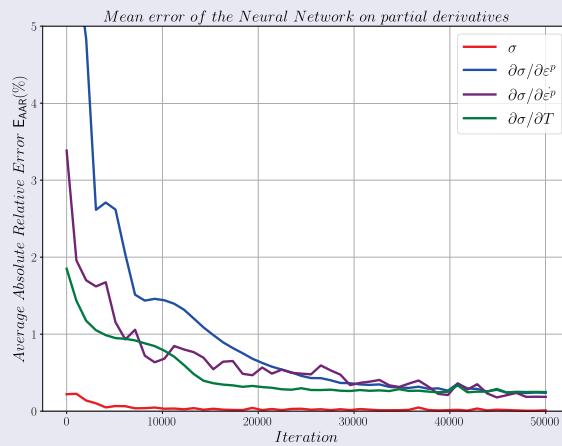


Figure 11: Convergence of parameters

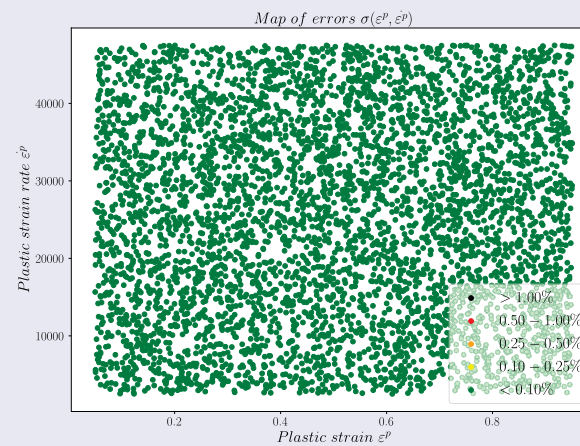


Figure 12: Map error of σ

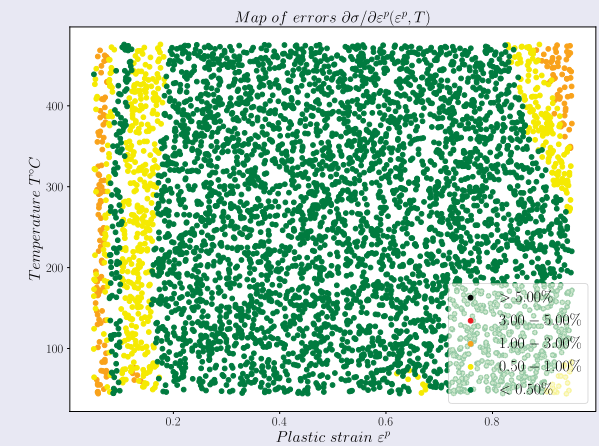


Figure 14: Map error of $\partial\sigma / \partial\dot{\epsilon}^p$

σ	$R = 0.99999955$
$\partial\sigma / \partial\epsilon^p$	$R = 0.99994377$
$\partial\sigma / \partial\dot{\epsilon}^p$	$R = 0.99999799$
$\partial\sigma / \partial T$	$R = 0.99918579$

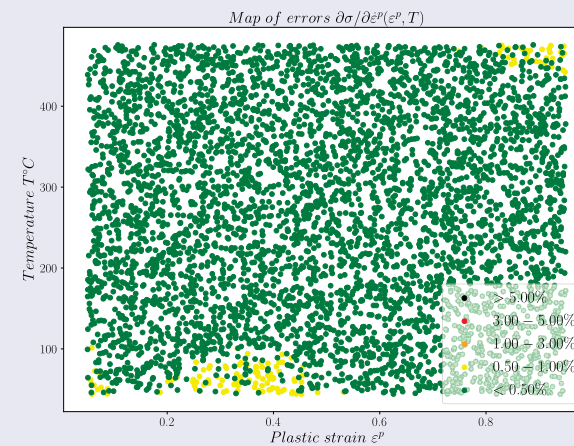


Figure 13: Map error of $\partial\sigma / \partial\epsilon^p$

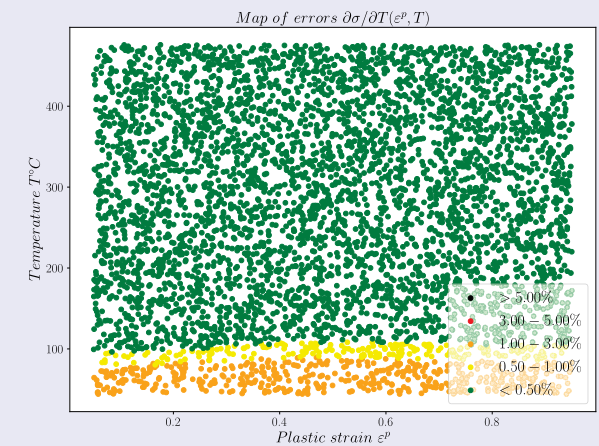


Figure 15: Map error of $\partial\sigma / \partial T$

- 1 Introduction and general purpose
- 2 Artificial Neural Network set-up
- 3 Training of the Artificial Neural Network and performance evaluation
- 4 Implementation of the Artificial Neural Network within Abaqus Explicit**
- 5 Numerical results
- 6 Conclusions and perspectives

Numerical implementation of the Artificial Neural Network in Abaqus/Explicit

- Numerical implementation is done using a VUHARD subroutine for the Abaqus Explicit code.
- We implement a FORTRAN subroutine to **compute the yield stress of the material and its derivatives**.
- A Python program **extracts** the internal parameters of the trained network and **creates** the FORTRAN subroutine.
- Depending on the network structure we optimize the code *i.e.* 2 hidden layers and **sig** function:

$$\begin{aligned}
 z_i^a &= \exp \left(-\sum_j \left(w_{ij}^{(1)} x_j \right) - b_i^{(1)} \right) & i \in [1, m], j \in [1, 3] \\
 z_i^b &= 1 + z_i^a & i \in [1, m] \\
 z_i^c &= \exp \left(-\sum_j \left(w_{ij}^{(2)} / z_j^b \right) - b_i^{(2)} \right) & i \in [1, n], j \in [1, m] \\
 z_i^d &= w_i z_i^c / (1 + z_i^c)^2 & i \in [1, n] \\
 z_i^e &= z_i^a / z_i^b & i \in [1, m] \\
 z_i^f &= \sum_j \left(w_{ji}^{(2)} z_j^d \right) z_i^e & i \in [1, m], j \in [1, n]
 \end{aligned} \tag{13}$$

The output of the neural network is given by:

$$s = \sum_i (w_i / (1 + z_i^c)) + b \quad i \in [1, n] \tag{14}$$

and, the three derivatives s'_i are obtained from:

$$s'_i = \sum_j \left(w_{ji}^{(1)} z_j^f \right) \quad i \in [1, 3], j \in [1, m] \tag{15}$$

Example of a generated optimized FORTRAN VUHARD code

```

1      subroutine vuhard (
2      + nblock, nElement, nIntPt, nLayer, nSecPt, lAnneal, stepTime,
3      + totalTime, dt, cmname, nstatev, nfieldv, nprops, props,
4      + tempOld, tempNew, fieldOld, fieldNew, stateOld, eqps, eqpsRate,
5      + yield, dyieldDtemp, dyieldDeqps, stateNew)
6      include 'vaba_param.inc'
7      dimension nElement(nblock), props(nprops), tempOld(nblock),
8      + fieldOld(nblock, nfieldv), stateOld(nblock, nstatev),
9      + tempNew(nblock), fieldNew(nblock, nfieldv), eqps(nblock),
10     + eqpsRate(nblock), yield(nblock), dyieldDtemp(nblock),
11     + dyieldDeqps(nblock, 2), stateNew(nblock, nstatev)
12     character*80 cmname
13
14     do k = 1, nblock
15         xepsp = eqps(k)
16         xdepsp = log(eqpsRate(k)) / 10.819778284410
17         xtemp = (tempNew(k) - 20.0) / 480.0
18         za0 = exp(0.171193018556*xepsp + 0.498235881329*xdepsp -
19 + 1.572309255600*xtemp + 0.549710810184)
20
21         ... FORTRAN code of the ANN generated by the Python translator software ...
22
23         Yield(k) = 977.555715042962*y + 579.184642915415
24         dyieldDeqps(k,1) = 977.555715042962 * yd0
25         dyieldDeqps(k,2) = 90.348959964501 * yd1 / eqpsRate(k)
26         dyieldDtemp(k) = 2.036574406340 * yd2
27     end do
28     return
29     end

```


- 1 Introduction and general purpose
- 2 Artificial Neural Network set-up
- 3 Training of the Artificial Neural Network and performance evaluation
- 4 Implementation of the Artificial Neural Network within Abaqus Explicit
- 5 Numerical results**
- 6 Conclusions and perspectives

Conditions of compilation and execution of the generated FORTRAN code

- The VUHARD subroutine is compiled using the GNU gfortran 9.3.0 (recommended is Intel Fortran).
- All benchmarks tests have been solved using Abaqus Explicit 2021.
- Computer used is a Dell XPS 13 laptop with 16 GiB of Ram and one 4 core i7-10510U Intel CPU running Ubuntu 20.04.
- All computations have been done using the double precision option of Abaqus, with parallel threads execution on two cores.

Necking of a circular bar benchmark test

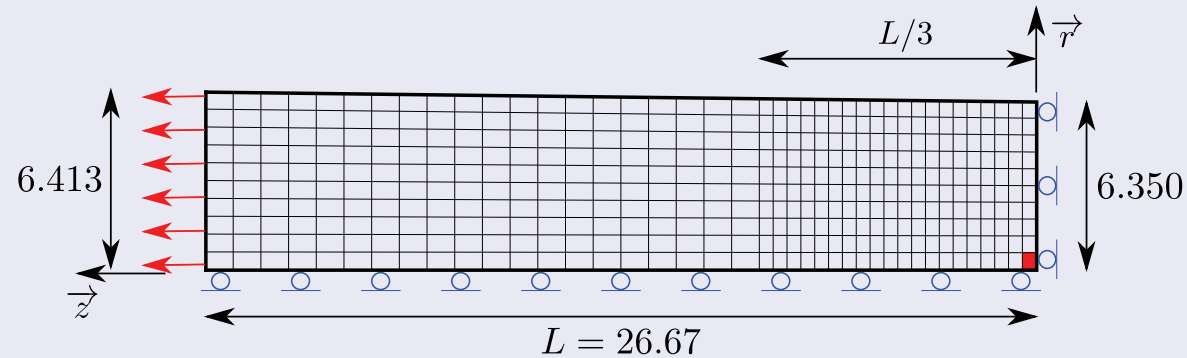


Figure 16: Meshing for the necking of a circular bar test

- Mesh consists of 400 CAX4RT elements (coupled temperature displacement)
- Total displacement is set to $d = 7$ mm
- Total simulation time is set to $t = 0.01$ s

Model	Incr.	Time (s)	$\bar{\epsilon}_{mid}^p$	$\bar{\sigma}_{mid}$ (MPa)	T_{mid} ($^{\circ}$ C)	$\bar{\epsilon}_{end}^p$	$\bar{\sigma}_{end}$ (MPa)	T_{end} ($^{\circ}$ C)
3-7-4-1-sig	191 768	29.98	0.51	1293.81	182.01	2.16	1064.43	587.78
3-15-7-1-sig	194 432	38.54	0.51	1293.97	181.52	2.03	1060.50	587.29
Analytical	200 145	35.50	0.51	1293.59	182.47	2.16	1045.75	585.85
Built-In	199 474	28.71	0.51	1293.76	180.36	2.14	1043.19	587.66

Table 3: Comparison of results for the necking of a circular bar benchmark for a displacement of 3.5 mm (mid) and 7 mm (end)

Both Temperature T and plastic strain $\bar{\epsilon}^p$ are out of training ranges.

Necking of a circular bar benchmark test

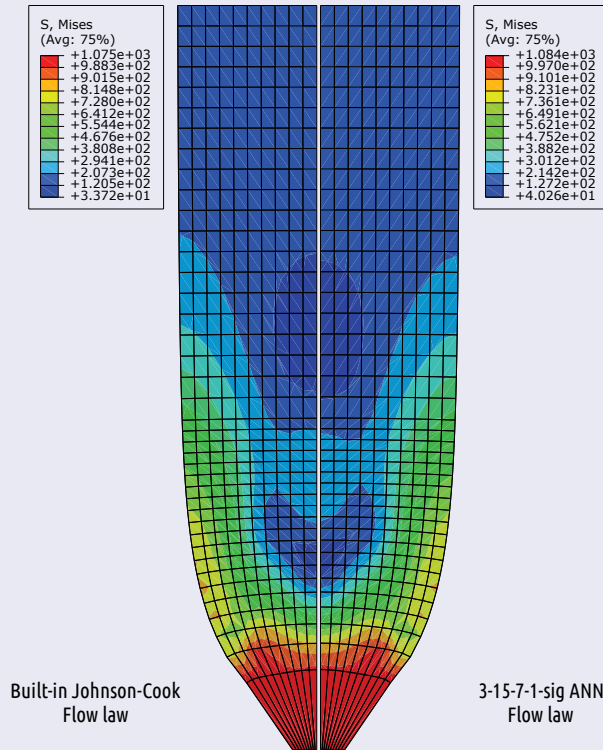


Figure 17: Von Mises equivalent stress contourplot $\bar{\sigma}$

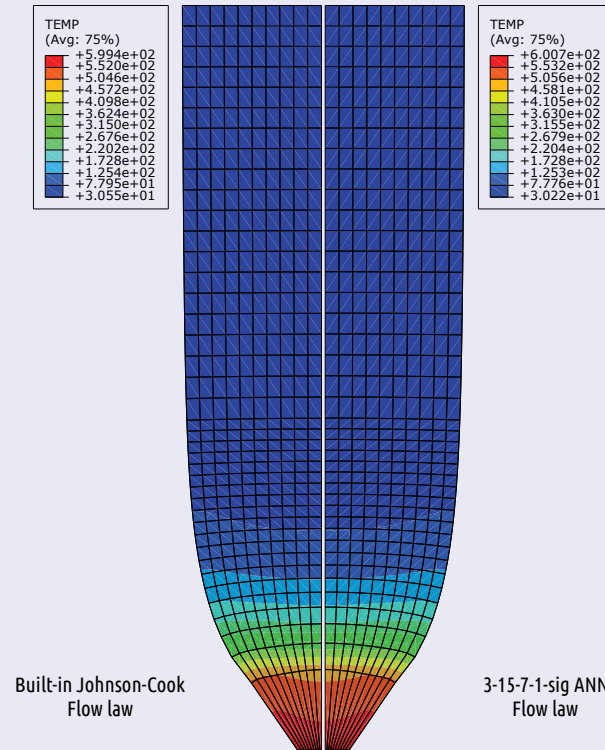


Figure 18: Temperature contourplot T

Necking of a circular bar benchmark test

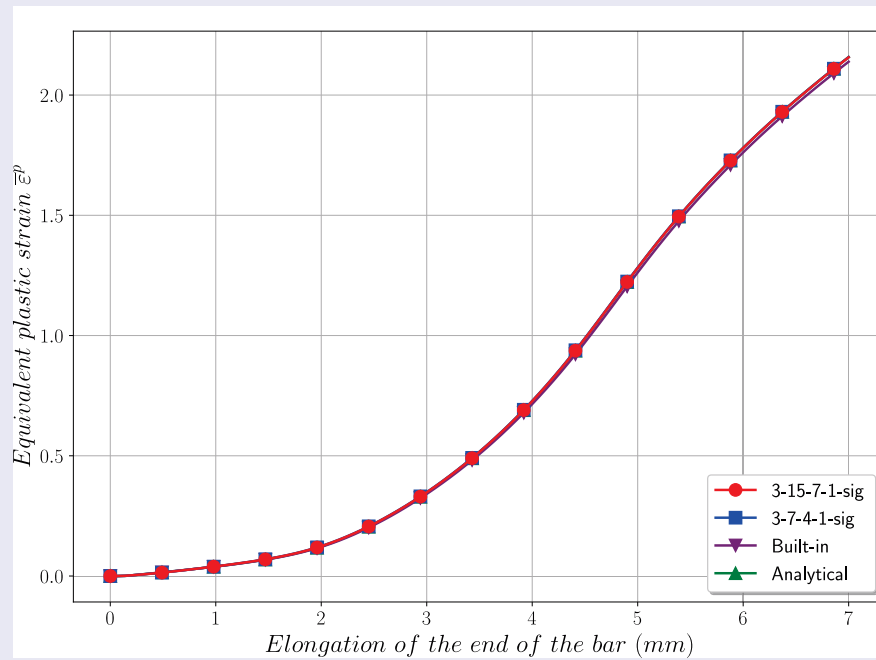


Figure 19: Equivalent plastic strain $\bar{\epsilon}^P$

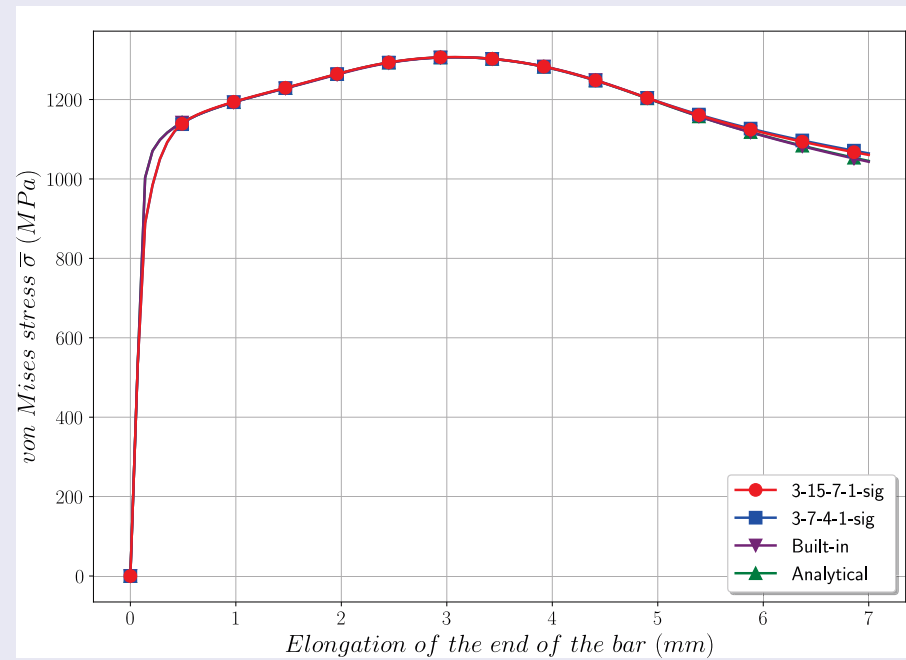


Figure 20: Equivalent von Mises stress $\bar{\sigma}$

Taylor impact benchmark test

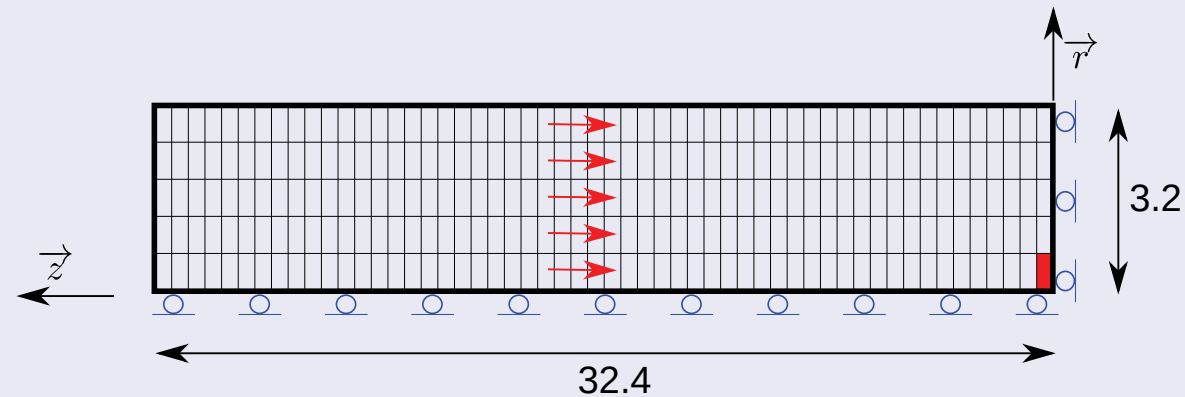


Figure 21: Meshing used for the Taylor impact test

- Mesh consists of 250 CAX4RT elements (coupled temperature displacement)
- Impact velocity $V_c = 287$ m/s
- Total simulation time is set to $t = 80 \mu\text{s}$.

Model	Incr.	Time (s)	L_f (mm)	R_f (mm)	T ($^{\circ}\text{C}$)	$\bar{\epsilon}^p$
3-7-4-1-sig	3 798	2.23	26.54	5.55	556.9	1.804
3-15-7-1-sig	3 820	2.35	26.54	5.55	558.0	1.799
Analytical	3 841	2.20	26.55	5.55	559.2	1.806
Built-In	4 142	2.05	26.57	5.57	562.9	1.800

Table 4: Comparison of results for the Taylor impact test

Both Temperature T and plastic strain $\bar{\epsilon}^p$ are out of training ranges.

Taylor impact benchmark test

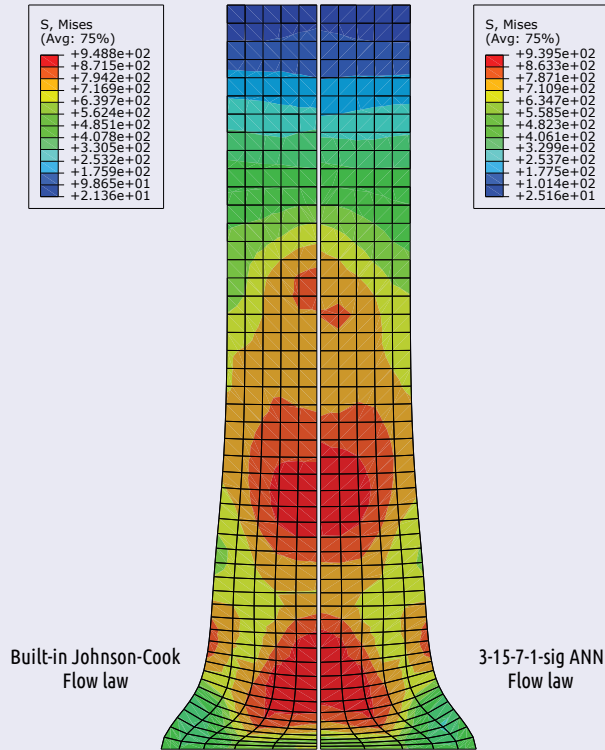


Figure 22: Von Mises equivalent stress contourplot $\bar{\sigma}$

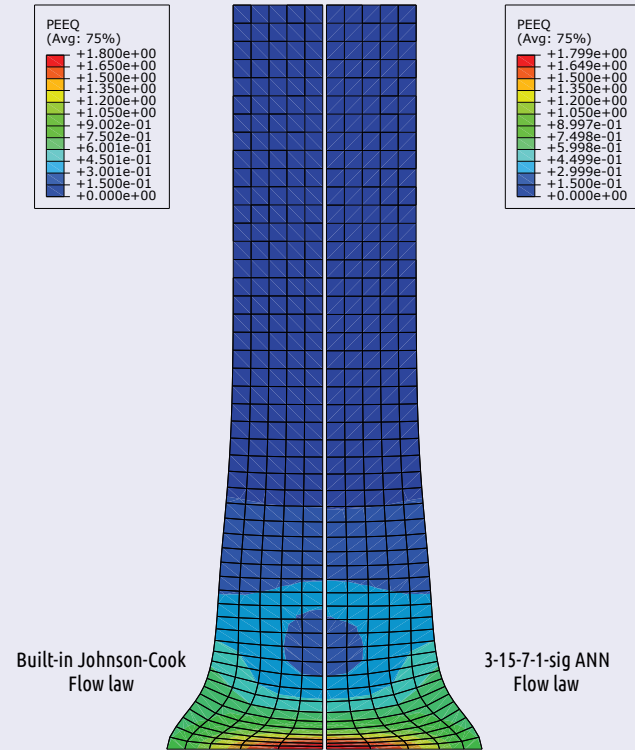


Figure 23: Equivalent plastic strain contourplot $\bar{\epsilon}^P$

Taylor impact benchmark test

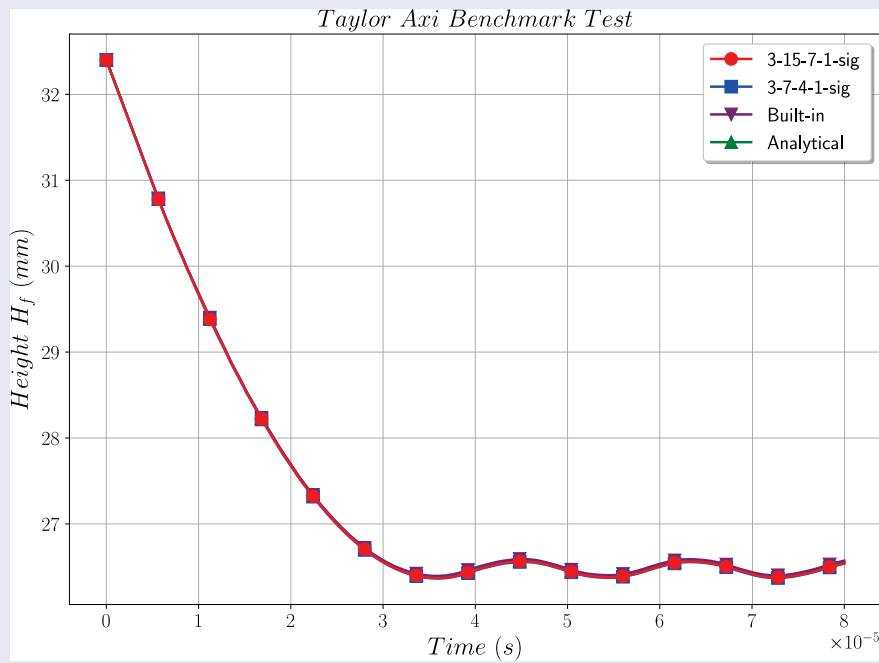


Figure 24: Total height H_f

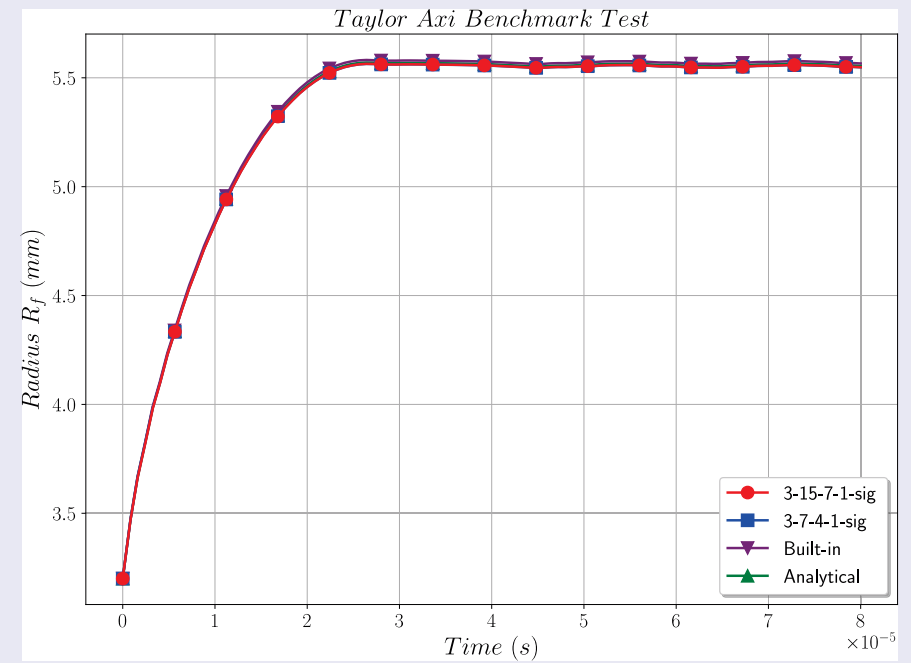


Figure 25: Radius R_f

Taylor impact benchmark test

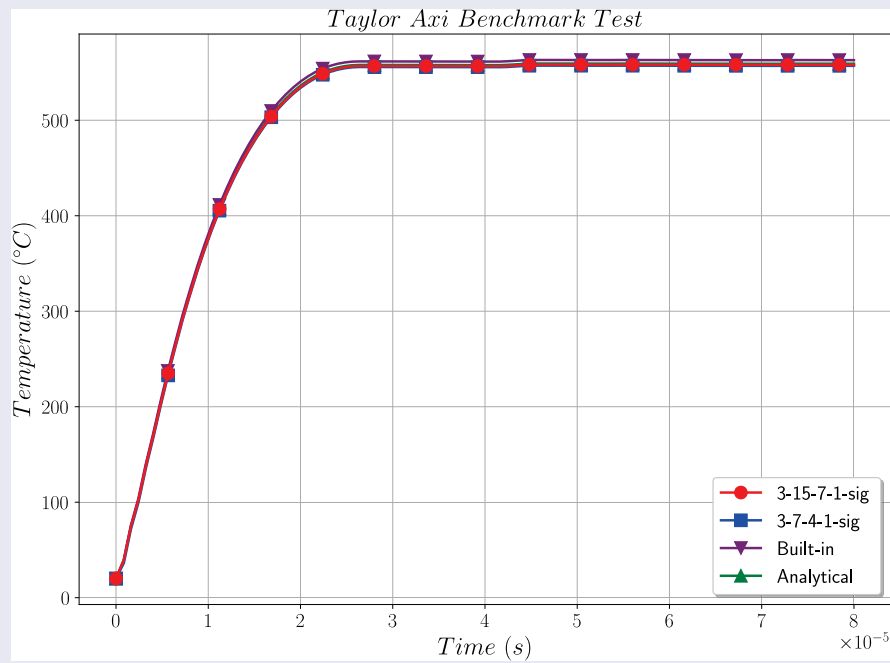


Figure 26: Temperature T

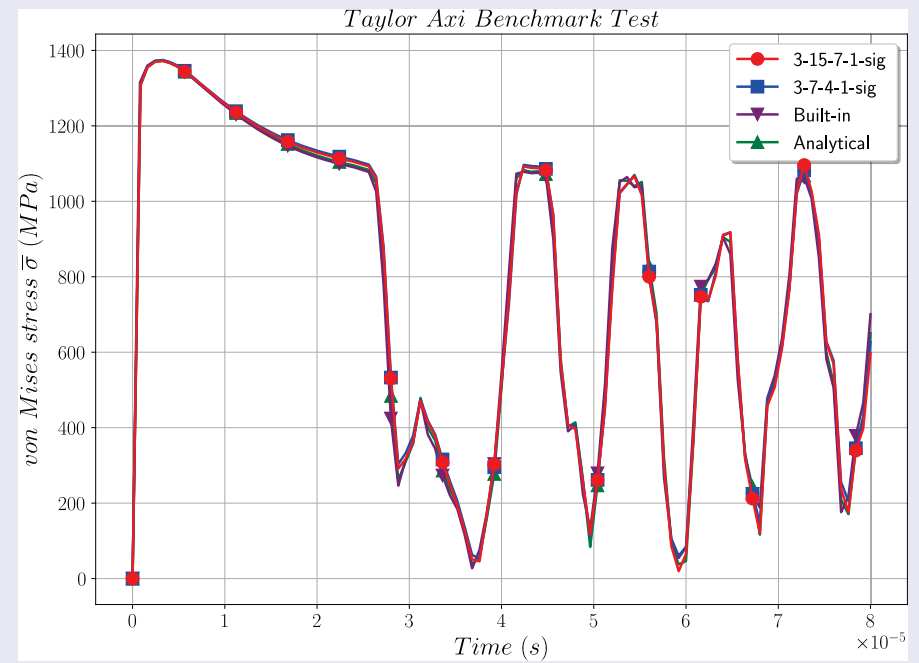


Figure 27: Equivalent von Mises stress $\bar{\sigma}$

- 1 Introduction and general purpose
- 2 Artificial Neural Network set-up
- 3 Training of the Artificial Neural Network and performance evaluation
- 4 Implementation of the Artificial Neural Network within Abaqus Explicit
- 5 Numerical results
- 6 Conclusions and perspectives

Conclusions and future work

- An Artificial Neural Network based framework has been proposed to **model the non-linear flow law** $\sigma^y(\varepsilon^p, \dot{\varepsilon}^p, T)$
- Its application to a 42CrMo4 steel and a constitutive behavior of type Johnson-Cook has been presented.
- The **accuracy** of the evaluation of the **derivatives** has been presented.
- The results obtained showed an **excellent ability to evaluate the flow stress** and a **very good ability to evaluate the derivatives** by the neural network.
- After numerical implementation of the neural network in the Abaqus code, the test cases used showed the good behavior of the proposed approach in the context of the numerical simulation of the necking of a circular bar and a Taylor impact test.
- Future work concerns the application to a real case from experiments on a Gleeble 3500.
- Validation of the results with regard to experimental tests.