Working with user subroutines within MSC.Marc

Soheil Solhjoo

1 Prerequisites

In order to work with the MSC.Marc's subroutines, a compatible Fortran compiler should be installed. This can be done in a number of steps. First of all, Visual Studio (NOT the express edition) and Intel Visual Fortran were needed to be installed; the former (version 2017) was acquired from Microsoft Imagine[1], and a student version of the latter (version Parallel Studio XE 2018 (Update 1)) from Intel Software. Once both were installed, the following steps were taken:

1. The folder of the installed Intel Visual Fortran was added to the windows environment.
2. Both codes were linked using the following code:

```
"C:\Program Files (x86)\IntelSWTools\
compilers_and_libraries_2018\windows\
bin\compilervars" intel64 vs2017
```
[2]

In order to test the compiler, the following simple code was written in a file named `helloworld.f`.

```
program helloworld
print *, "Hello world!"
end program helloworld
```

Then, it was compiled using the following command:

```
ifort helloworld.f
```

This command generated an executive file named `helloworld.exe`. Once the codes were installed and linked, as the next step, they should be tested within MSC.Marc. This was done by running the `example 2.35` from Marc's manual (*volume E*), in which a non-uniform load is defined within a user subroutine named `FORCEM`[3]. The simulation ran successfully.

In the following section, a series of simulations will be discussed for working with user subroutines.

[1]Older versions can be freely downloaded from VisualStudio website.

[2]In order to make sure that this link is established whenever Marc is running, this command was added to the shortcut of the Marc Mentat.

[3]This user subroutine is extensively discussed in the MSC.Marc's manual *volume D*, pp. 62-66.

2 Case Study

Basic System In this simulation, the goal is to apply a uniform load upon a square deformable material. The plate has a side length of $L = 10$, and a height of $h = 1$. The material's mechanical properties were assigned to be $E = 2 \times 10^5$ and $\nu = 0.3$. The position of the plate was fixed at its bottom nodes, and a uniform pressure of $p = 2000$ was applied to the top faces of the plate.

First the plate was generated, and divided into 20,20, and 3 sections along the x,y, and z axes, respectively. Following the subdivision, the `sweep_all` command was invoked, in order to make sure there are no duplicates. Then, a material with finite stiffness was generated with the assigned mechanical properties. It should be noted that the default value of `Mass Density = 0` was not altered. In order to fix the position of the bottom nodes, the `Fixed Displacement`, and for applying the pressure, the `Face Load` options were selected from the `Boundary Conditions` menu. Then, a `Structural` job was defined, and the `Linear Elastic Analysis` was turned on. Then, the job was submitted.

Moving Load Based on the prepared basic system, the next simulation was designed with the goal to control the applied pressure using a user subroutine: the position of the applied pressure is time dependent. In this study:

- The distributed load is applied within a square with a length of $b = 2$.
- The applying load moves with a velocity of $\sqrt{2}$ along the diagonal.
- The simulation runs during 10 time units with 20 equally sized increments.

In order to perform this simulation, a number of changes were applied:

1. The value of the applied pressure was set to 1, and the `Method` was changed to `User Sub. Forcem`.
2. From the `Loadcases` menu, a new `Structural Static` load case was created: the `Total Loadcase Time` and `#Steps` were assigned to be 10 and 20, respectively.
3. The moving load was controlled via a user subroutine. The subroutine was saved in a file (see appedix A for the code). Then, the file was loaded in the in the `Run Job` dialogue.

User Defined Material (UDM) Following the basic system, another simulation was designed for defining the material's mechanical behavior via an user subroutine. This was done by defining the mechanical responses of the simulated material in different regimes.

1. **Elastic**: the material was modeled as an `Elastic-Plastic Anisotropic` one, and the `User Subs. Hooklw/Anelas` was activated for defining the material using the corresponding subroutine.
   The `Hooklw` was used in this study. The Hooke's law is written as $\left[\boldsymbol{\sigma}\right] = \left[\mathbf{C}\right]\left[\boldsymbol{\epsilon}\right]$; the stiffness tensor $\left[\mathbf{C}\right]$ should be defined in the `Hooklw` subroutine. I tried two methods, (1) defining a compliance tensor $\left[\mathbf{S}\right]$, as in $\left[\boldsymbol{\epsilon}\right] = \left[\mathbf{S}\right]\left[\boldsymbol{\sigma}\right]$, and using $\left[\mathbf{C}\right] = \left[\mathbf{S}\right]^{-1}$ (see Appendix B), and (2) defining $\left[\mathbf{C}\right]$ as a function of the Lamé constants λ and μ (see Appendix C). Both methods resulted in the same values as the `Elastic-Plastic Isotropic` material, which was used in the basic system.
2. **Plastic** : the mechanical behavior was formulated within the `Wkslp` subroutine. I tried a sample of the MSC.Marc, called `barlat_yld2004_18p`. In the simulation, I set the plastic behavior to be read from the user subroutine, and I wrote the file according to the original formulation of $\sigma = 646(1+\varepsilon)^{0.227}$. The simulation ran successfully, and the results were the same as the original ones. See Appendix D for the subroutine's code.

In order to activate both subroutines, each ones were written in a file, and a using the `#include` command, they were summoned in another file; then, this final file was called within the MSC.Marc.

Neural Network-based UDM In another attempt, the simulation described in 2 was performed by defining the material via a trained neural network. To do so, the stress-strain curve was imported in an in-house MATLAB code. Then, the trained network was converted into Fortran 90 language, and implemented within the `Wkslp` subroutine. The subroutine is presented in Appendix E; however, it should be noted that the code might not be the most suitable one to be used. Perhaps, the `urpflo` subroutine is a more appropriate one, which does not take care of the elastic regime of the deformation; in other words, the material is treated as a rigid-perfectly plastic one.

The preparation of the subroutine has been done in three stages: (1) running a NN trainer in MATLAB (Appendix), (2) saving the trained NN (Appendix), and (3) converting the saved NN into Fortran 90 (Appendix).

A Moving Load

```
1        subroutine forcem(press,p,d,nn,n)
2  #ifdef _IMPLICITNONE
3        implicit none
4  #else
5        implicit logical (a-z)
6  #endif
7  #include "creeps"
8       dimension p(3),d(3),n(2)
9        real*8 b, v, dist
10      real*8 xc, xmax, xmin, yc, ymax, ymin
11
12      b = 2.0
13      v = sqrt(2.0)
14      dist = v*(cptim+timinc)
15      xc = 0.5*sqrt(2.0)*dist
16      yc = 0.5*sqrt(2.0)*dist
17
18      xmin = xc - b/2.0
19      xmax = xc + b/2.0
20      ymin = yc - b/2.0
21      ymax = yc + b/2.0
22
23      press = 0.0
24      if (p(1).le.xmax.and.p(1).ge.xmin) then
25      if (p(2).le.ymax.and.p(2).ge.ymin) then
26        press = 2000.0
27      endif
28      endif
29
30      return
31      end
```

It should be noted that the conditions in lines 24 and 25 could not be combined, due to some restrictions on the installed compiler.

B User material: Elastic 1

```
        subroutine hooklw(m,nn,kcus,b,ngens,dt,dtdl,e,
     pr,ndi,nshear,
      * imod,rprops,iprops)
#ifdef _IMPLICITNONE
        implicit none
#else
        implicit logical (a-z)
#endif
        real*8 b, dt, dtdl, e
        integer imod, iprops, kcus, m, ndi, ngens, nn,
      nshear
        real*8 pr, rprops
        dimension b(ngens,ngens),dt(*),dtdl(*),rprops
     (*),iprops(*),
      *  kcus(2),m(2),e(*),pr(*)

      real*8 Ym, nu, g

      real*8 ex, ey, ez, uxy, uyz, uzx, gxy, gyz, gzx
      real*8 sum, d
      dimension sum(20)

      imod = 1

c material constants
     Ym = 200000.
      nu = 0.3
      g  = Ym/2./(1.+nu)

c    modified for anisotropic material
        ex = Ym
        ey = Ym
        ez = Ym
        uxy = nu
        uyz = nu
        uzx = nu
        gxy = g
        gyz = g
        gzx = g

c    comliance tensor
        b(1,1)=1./ex
        b(2,2)=1./ey
        b(3,3)=1./ez
        b(2,1)=-uxy/ex
        b(3,2)=-uyz/ey
        b(1,3)=-uzx/ez
        b(3,1)=b(1,3)
        b(1,2)=b(2,1)
        b(2,3)=b(3,2)
        b(4,4)=1./gxy
```

```
      b(5,5)=1./gyz
      b(6,6)=1./gzx

c    stiffness tensor: b
      call invert(b,ngens,sum,0,d,ngens)

      return
      end
```

C User material: Elastic 2

```
      subroutine hooklw(m,nn,kcus,b,ngens,dt,dtdl,e,pr,ndi,nshear,
     * imod,rprops,iprops)
#ifdef _IMPLICITNONE
      implicit none
#else
      implicit logical (a-z)
#endif
      real*8 b, dt, dtdl, e
      integer imod, iprops, kcus, m, ndi, ngens, nn, nshear
      real*8 pr, rprops
      dimension b(ngens,ngens),dt(*),dtdl(*),rprops(*),iprops(*),
     *  kcus(2),m(2),e(*),pr(*)

    real*8 Ym, nu, lambda, mu
    integer i, j
    imod = 1

    Ym = 200000.0
    nu = 0.3

    lambda = Ym*nu/(1.+nu)/(1.-2.*nu)
    mu = Ym/(1.+nu)/2.

    b = 0.
    b(1,1) = 2.*mu + lambda
    b(2,2) = b(1,1)
    b(3,3) = b(1,1)
    b(4,4) = mu
    b(5,5) = b(4,4)
    b(6,6) = b(4,4)
    b(1,2) = lambda
    b(1,3) = b(1,2)
    b(2,1) = b(1,2)
    b(2,3) = b(1,2)
    b(3,1) = b(1,2)
    b(3,2) = b(1,2)

      return
      end
```

D User material: Plastic

```
      subroutine wkslp(m,nn,kcus,matus,slope,ebarp,eqrate,stryt,dt,
     *  ifirst)
#ifdef _IMPLICITNONE
      implicit none
#else
      implicit logical (a-z)
#endif
      stryt = 646.*(0.025+ebarp)**0.227
      slope = 646.*0.227*(0.025+ebarp)**(1.-0.227)
      return
      end
```

E User material: Neural Network

```
    module NN_funcs
#ifdef _IMPLICITNONE
      implicit none
#else
      implicit logical (a-z)
#endif
    contains

    function mapminmax_apply(x, xoffset, gain, ymin) result(y)
    !implicit none
    real*8, intent(in) :: x, xoffset, gain, ymin
    real*8 :: y
      y = x-xoffset
    y = y*gain
    y = y+ymin
    end function mapminmax_apply

    function tansig_apply(n,i,j) result(y)
    !implicit none
    INTEGER i,j
    real*8 n(i,j)
    real*8 y(i,j)
    y = 2. / (1. + exp(-2.*n)) - 1.
    end function tansig_apply

    function mapminmax_reverse(y, xoffset, gain, ymin, j) result(x)
    !implicit none
    integer j
    real*8, intent(in) :: y(j), xoffset, gain, ymin
    real*8 x(j)
    x = y-ymin
    x = x/gain
    x = x+xoffset
    end function mapminmax_reverse

    end module NN_funcs
!=======================================
      subroutine wkslp(m,nn,kcus,matus,slope,ebarp,eqrate,stryt,dt,
     *  ifirst)
#ifdef _IMPLICITNONE
      implicit none
#else
      implicit logical (a-z)
#endif
      real*8 dt, ebarp, eqrate
      integer ifirst, kcus, m, matus, nn
      real*8 slope, stryt
      dimension matus(2),kcus(2)
! In order to calculate the slope, stress will be calculated at two strains:
! (1) the current strain, and (2) strain + deps. Their difference divided by deps
! would be reported as the slope.
! "deps" needs to possess a small value. In here, arbitrarily, it is set to be 0.01.
    real*8 deps, nextyld
    deps = 0.01

    call NeuralNet(stryt,ebarp)
    call NeuralNet(nextyld,ebarp + deps)
    slope = (stryt - nextyld) / deps
      return
      end
!=======================================
    subroutine NeuralNet(stress,strain)
    use NN_funcs
#ifdef _IMPLICITNONE
      implicit none
#else
      implicit logical (a-z)
#endif
!Constants Definition
    real*8 x_xoffset, x_gain, x_ymin
    real*8 y_xoffset, y_gain, y_ymin
```

```fortran
  !Input
      real*8 Xp1, strain
  !Layer1
      INTEGER, PARAMETER :: b1i=2,b1j=1
      real*8, dimension(b1i,b1j) :: a1, b1, IW1_1,
    nn_sum
  !Layer2
      INTEGER, PARAMETER :: b2i=2,b2j=1
      real*8, dimension(b2j) :: a2, b2, stress
    real*8 LW2_1(b2i)
!Assign_values
   x_xoffset = 0.
   x_gain = 2.
   x_ymin = -1.
   y_xoffset = 646.
   y_gain = 0.0181690135872346
   y_ymin = -1.
  !Layer_1
   DATA b1 / 0.2344368269302832819,
    1.9855625577702908924 /
   DATA IW1_1 / 0.1580287828186244525 3,
    0.5164193236472780501 6 /
  !Layer_2
   b2 = -6.0679754121327071914
   DATA LW2_1 / 5.2036136075577150706,
    5.1936881732563282554 /
!------------------------------------------
!Simulation
   Xp1 = strain
  !Input_1
      Xp1 = mapminmax_apply(Xp1, x_xoffset, x_gain,
    x_ymin)
    !Layer_1
      nn_sum = b1 + IW1_1 * Xp1
    a1 = tansig_apply(nn_sum,b2i,b2j);
    !Layer_2
      a2 = b2 + matmul(LW2_1,a1)
  !Output_1
    stress = mapminmax_reverse(a2, y_xoffset, y_gain
    , y_ymin, b2j);

    return
    end
```

F User material: Neural Network: NN Trainer

```matlab
clear; close all;
%% Input data
sz = 100;   %# of sample size
var = 1;    %# of variables
eps = linspace(0,1,sz);
k = 646; n = 0.227;
sig = k * (1+eps).^n;% + 10*rand(var,sz);
%% Train the NN
x = eps(:)';
y = sig(:)';
nn = 3;
net = feedforwardnet([4 3 2]);
% net = cascadeforwardnet(nn);
[net,tr] = train(net,x,y);
%% Exctract the trained NN
genFunction(net,'netFcn');
%% Test the NN
eps_test = .35;%linspace(0,1,sz/10);
x = eps_test(:)';
% y_NN = net(x);
y_NN = netFcn(x);
%% Visualize the comparison
% y_NN = netFcn(x);
figure;
plot(eps,sig,'-');
hold on
plot(eps_test,y_NN,'*');
```

G User material: Neural Network: Trained NN

```matlab
function [Y,Xf,Af] = netFcn(X,~,~)
%NETFCN neural network simulation function.
%
% Generated by Neural Network Toolbox function
    genFunction, 23-Aug-2018 15:31:29.
%
% [Y] = netFcn(X,~,~) takes these arguments:
%
%   X = 1xTS cell, 1 inputs over TS timesteps
%   Each X{1,ts} = 1xQ matrix, input #1 at timestep
    ts.
%
% and returns:
%   Y = 1xTS cell of 1 outputs over TS timesteps.
%   Each Y{1,ts} = 1xQ matrix, output #1 at timestep
    ts.
%
% where Q is number of samples (or series) and TS is
    the number of timesteps.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset = 0;
x1_step1.gain = 2;
x1_step1.ymin = -1;

% Layer 1
b1 =
    [-4.2331650311832484945;-1.8534005300867182342;0.3717

IW1_1 =
    [4.3486852475527291162;1.7811488781001498793;0.5534845


% Layer 2
b2 =
    [-2.7505303206879916367;-0.31118817730058662141;2.426

LW2_1 = [0.18059897243039635395
    0.44654030386170651123 1.6797400342346737734
    -1.7649989328324051652;0.66561540831778609473
    -0.29549659468472666557 -1.1562732833758548878
    0.052967542915231445588;0.021583755501849426206
    -1.0449273951358553081 0.064952170212388068982
    -1.3557824303581875736];

% Layer 3
b3 = [1.3319933641381176415;1.6671150760804414048];
LW3_2 = [-1.5508478424855232092
    0.53242300257854990875
    0.81836946123260534414;1.3440984454581468288
    -1.3071802986342959674 -1.1884459257320574288];

% Layer 4
b4 = 0.20836408570433107013;
LW4_3 = [-0.88808615067962504153
    1.6565370363349731786];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 0.0181690135872346;
y1_step1.xoffset = 646;

% ===== SIMULATION ========

% Format Input Arguments
isCellX = iscell(X);
if ~isCellX
```

```
  X = {X};
end

% Dimensions
TS = size(X,2); % timesteps
if ~isempty(X)
  Q = size(X{1},2); % samples/series
else
  Q = 0;
end

% Allocate Outputs
Y = cell(1,TS);

% Time loop
for ts=1:TS

    % Input 1
    Xp1 = mapminmax_apply(X{1,ts},x1_step1);

    % Layer 1
    a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*Xp1);

    % Layer 2
    a2 = tansig_apply(repmat(b2,1,Q) + LW2_1*a1);

    % Layer 3
    a3 = tansig_apply(repmat(b3,1,Q) + LW3_2*a2);

    % Layer 4
    a4 = repmat(b4,1,Q) + LW4_3*a3;

    % Output 1
    Y{1,ts} = mapminmax_reverse(a4,y1_step1);
end

% Final Delay States
Xf = cell(1,0);
Af = cell(4,0);

% Format Output Arguments
if ~isCellX
  Y = cell2mat(Y);
end
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
  y = bsxfun(@minus,x,settings.xoffset);
  y = bsxfun(@times,y,settings.gain);
  y = bsxfun(@plus,y,settings.ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
  a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x = mapminmax_reverse(y,settings)
  x = bsxfun(@minus,y,settings.ymin);
  x = bsxfun(@rdivide,x,settings.gain);
  x = bsxfun(@plus,x,settings.xoffset);
end
```

H User material: Neural Network: NN Convertor

```
clear all; clc;
seed_file_name = 'netFcn.m';
%% Read the netFcn.m file
fid = fopen(seed_file_name,'r');
netFcn = textscan(fid,'%s','whitespace','','delimiter','\n');
netFcn = netFcn{:};
fclose(fid);
%% Find the number of hidden layers
idx = find(strcmp(netFcn, '% Output 1'));
idx = idx - 4;
L_fin = netFcn{idx};
idx = ismember(L_fin, '% Layer ');
L_fin(idx) = [];
L_fin = str2num(L_fin);
%% Collect the network constants
%x_step
idx = find(strcmp(netFcn, '% Input 1'));
for i = 1:3
    x_step(i) = read_data(netFcn{idx+i});
end

%y_step
idx = find(strcmp(netFcn, '% Output 1'));
for i = 1:3
    y_step(i) = read_data(netFcn{idx+i});
end

% Layers
idx = find(strcmp(netFcn, '% Layer 1'));
const{L_fin,2} = {};
for i = 1:L_fin
    const{i,1} = read_data(netFcn{idx+1});  %b
    const{i,2} = read_data(netFcn{idx+2});  %W
    idx = idx + 4;
end
%% Prepare for writing a Fortran subroutine
% Header
text{1,1}   = '      include ''NN_Funcs.f'' ';
text{2,1}   = '      subroutine NeuralNet(stress,strain)';
text{3,1}   = '      use NN_funcs';
text{4,1}   = '#ifdef _IMPLICITNONE';
text{5,1}   = '      implicit none';
text{6,1}   = '#else';
text{7,1}   = '      implicit logical (a-z)';
text{8,1}   = '#endif';
% Definer
text{9,1}   = '      real*8 x_xoffset, x_gain, x_ymin';
text{10,1}  = '      real*8 y_xoffset, y_gain, y_ymin';
text{11,1}  = '      real*8 a0, strain';
text{12,1}  = '      real*8, dimension(1) :: stress';
idx = size(text,1);
formatL1    = '      real*8, dimension(%i,1) :: b%i, a%i';
formatL2    = '      real*8, dimension(%i,%i) :: W%i';
Layers_size(1) = 1;
Layers_size(L_fin+1) = 1;
for i=2:L_fin
    Layers_size(i) = size(const{i-1},1);
    text{idx+1,1} = sprintf(formatL1,Layers_size(i),i-1,i-1);
    text{idx+2,1} = ...
        sprintf(formatL2,Layers_size(i),Layers_size(i-1),i-1);
    idx = idx + 2;
end
formatL1    = '      real*8, dimension(%i) :: b%i, a%i';
formatL2    = '      real*8, dimension(%i) :: W%i';
text{idx+1,1} = sprintf(formatL1,Layers_size(L_fin+1),i,i);
text{idx+2,1} = sprintf(formatL2,Layers_size(i),i);
% Steps (x and y)
idx = size(text,1);
```

```
text{idx+1,1} = ['        x_xoffset = ', num2str(x_step(1))];
text{idx+2,1} = ['        x_gain = ', num2str(x_step(2))];
text{idx+3,1} = ['        x_ymin = ', num2str(x_step(3))];
text{idx+4,1} = ['        y_xoffset = ', num2str(y_step(1))];
text{idx+5,1} = ['        y_gain = ', num2str(y_step(2))];
text{idx+6,1} = ['        y_ymin = ', num2str(y_step(3))];
% bs and Ws
idx = size(text,1);
formatLbW     = '        data %s%i  / %s  /';
for  i=1:L_fin
    b_string = sprintf('%.20f, ' , const{i,1}');
    b_string = b_string(1:end-2);
    W_temp = const{i,2};
    W_string = sprintf('%.20f, ' , W_temp(:)');
    W_string = W_string(1:end-2);

    text{idx+1,1} = sprintf(formatLbW,'b',i,b_string);
    text{idx+2,1} = sprintf(formatLbW,'W',i,W_string);
    idx = idx + 2;
end
% Simulation Section
idx = size(text,1);
text{idx+1,1}    = ...
    '        a0 = mapminmax_apply(strain, x_xoffset, x_gain, x_ymin)';
formatLa     = '        a%i = tansig_apply(b%i+W%i*a%i,%i,%i)';
idx = size(text,1);
for  i=1:L_fin-1
    text{idx+1,1} = sprintf(formatLa,i,i,i,i-1,Layers_size(i+1),1);
    idx = idx + 1;
end
formatLafin = '        a%i = b%i + matmul(W%i,a%i)';
text{idx+1,1} = sprintf(formatLafin,L_fin,L_fin,L_fin,i);
formatLY     = ...
    '        stress = mapminmax_reverse(a%i, y_xoffset, y_gain, y_ymin, 1)';
text{idx+2,1} = sprintf(formatLY,L_fin);
% Ending
idx = size(text,1);
text{idx+1,1}    = '        return';
text{idx+2,1}    = '        end';
%% Write down the subroutine file named NeuralNet.f
fid = fopen('NeuralNet.f','w');
for  i = 1:size(text,1)
    fprintf(fid,'%s\n',text{i});
end
fclose(fid);
```

This code requires the following function.

```
function number = read_data(string)
%read_data
idx_temp1 = ismember(string, '=');
idx_temp2 = find(idx_temp1,1);
idx_temp1(1:idx_temp2) = 1;
string(idx_temp1) = [];
number = str2num(string);
end
```

I User material: Neural Network: Converted NN

```
        include 'NN_Funcs.f'
        subroutine NeuralNet(stress,strain)
        use NN_funcs
#ifdef _IMPLICITNONE
        implicit none
#else
        implicit logical (a-z)
#endif
        real*8 x_xoffset, x_gain, x_ymin
        real*8 y_xoffset, y_gain, y_ymin
        real*8 a0, strain
        real*8, dimension(1) :: stress
        real*8, dimension(4,1) :: b1, a1
        real*8, dimension(4,1) :: W1
        real*8, dimension(3,1) :: b2, a2
        real*8, dimension(3,4) :: W2
        real*8, dimension(2,1) :: b3, a3
        real*8, dimension(2,3) :: W3
        real*8, dimension(1) :: b4, a4
        real*8, dimension(2) :: W4
        x_xoffset = 0
        x_gain = 2
        x_ymin = -1
        y_xoffset = -1
        y_gain = 0.018169
        y_ymin = 646
        data b1 / -4.23316503118324849453, -1.85340053008671823420, 0.37171248405505208368, -5.39137284079696055272 /
        data W1 / 4.34868524755272911619, 1.78114887810014987934, 0.55348457993771060792, -3.86835244323420823775 /
        data b2 / -2.75053032068799163667, -0.31118817730058662141, 2.42637658741737949342 /
        data W2 / 0.18059897243039635395, 0.66561540831778609473, 0.02158375550184942621, 0.44654030386170651123, -0.29549659468472666557, -1.04492739513585530808, 1.67974003423467377338, -1.15627328337585488782, 0.06495217021238806898, -1.76499893283240516517, 0.05296754291523144559, -1.35578243035818757356 /
        data b3 / 1.33199336413811764146, 1.66711507608044140483 /
        data W3 / -1.55084784248552320918, 1.34409844545814682881, 0.53242300257854990875, -1.30718029863429596737, 0.81836946123260534414, -1.18844592573205742880 /
        data b4 / 0.20836408570433107013 /
        data W4 / -0.88808615067962504153, 1.65653703633497317860 /
        a0 = mapminmax_apply(strain, x_xoffset, x_gain, x_ymin)
        a1 = tansig_apply(b1+W1*a0,4,1)
        a2 = tansig_apply(b2+W2*a1,3,1)
        a3 = tansig_apply(b3+W3*a2,2,1)
        a4 = b4 + matmul(W4,a3)
        stress = mapminmax_reverse(a4, y_xoffset, y_gain, y_ymin, 1)
        return
        end
```