```c
/** Interface to an isotropic elastic solid with continuum damage mechanics model: Mazars' damage
model
*   Example code implements linear elastic behaviour with damage. */

/** You are allowed to use, modify, and publish this External Material File and your modifications of
it subject
*   to the terms and conditions of the COMSOL Software License Agreement (www.comsol.com/sla). */

/** Copyright © 2015 by COMSOL. */

#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif
#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

EXPORT int eval(double e[6],       // Input: Green-Lagrange strain tensor components in Voigt order
(xx,yy,zz,yz,zx,xy)
                double s[6],       // Output: Second Piola-Kirchhoff stress components in Voigt order
(xx,yy,zz,yz,zx,xy)
                double D[6][6],    // Output: Jacobian of stress with respect to strain, 6-by-6
matrix in row-major order
                int *nPar,         // Input: Number of material model parameters, scalar,
                double *par,       // Input: Parameters: par[0] = E0, par[1] = nu0, ...
                int *nStates,      // Input: Number of states, scalar
                double *states) {  // States, nStates-vector

  int i, j;
  double E0, nu0, stch[3], kappa0, At, Bt, Ac, Bc, kappa, ep[3], evol, lambLame, muLame;
  double sp[3], st[3], stsum, et[3], eef2, eef, ets, ecs, alphat, alphac, damt, damc, damage, E;

  // Check inputs
  if (nPar[0] != 10)       // 10 inputs, E0, nu0, stretches and Mazars' parameters
    return 1;              // error code 1 = "Wrong number of parameters"
  if (nStates[0] != 2)     // 2 states to memorize damage variables
    return 2;              // error code 2 = "Wrong number of states"

  // Read input parameters from parameter vector, call convention:
  //  {E0, nu0, comp1.solid.stchelp1, comp1.solid.stchelp2, comp1.solid.stchelp3, kappa0, At, Bt, Ac,
Bc}
  E0 = par[0];    // undamaged Young's modulus
  if (E0 <= 0.0) return -1;
  nu0 = par[1];                   // undamaged Poisson's ratio
  if (nu0 >= 0.5 || nu0<= -1.0) return -1;
  for (i = 0; i < 3; i++) {
    stch[i] = par[i+2];           // principal stretches
    if (stch[i] <= 0.0) return -1;
  }
  kappa0 = par[5];                // Mazars' parameter: initial damage threshold
  if (kappa0 <= 0.0) return -1;
  At = par[6];                    // Mazars' parameter At
  if (At <= 0.0) return -1;
  Bt = par[7];                    // Mazars' parameter Bt
  if (Bt <= 0.0) return -1;
  Ac = par[8];                    // Mazars' parameter Ac
  if (Ac <= 0.0) return -1;
  Bc = par[9];                    // Mazars' parameter Bc
  if (Bc <= 0.0) return -1;
```

```
  kappa = MAX(states[0],kappa0); // kappa memorizes maximum equivalent tensile strain, initialized to 
kappa0

  // Define Lame parameters
  lambLame = E0 * nu0 / (1.0 + nu0) / (1.0 - 2.0 * nu0); // undamaged Lame parameter lambda
  muLame = E0 / 2 / (1.0 + nu0);                         // undamaged Lame parameter mu

  // Define principal engineering strains computed from principal stretches
  evol = 0.0;
  for (i = 0; i < 3; i++) {
    ep[i] = stch[i] - 1.0;   // principal strains
    evol += ep[i];           // volumetric strain
  }

  stsum = 0.0, eef2 = 0.0;
  for (i = 0; i < 3; i++) {
    sp[i] = lambLame * evol + 2.0 * muLame * ep[i]; // principal stresses
    st[i] = MAX(sp[i], 0.0);                        // tensile stresses, positive only (Mazars' 
model)
    stsum += st[i];                                 // sum of tensile stresses
	et[i] = MAX(ep[i], 0.0);                        // tensile strains, positive only (Mazars' 
model)
    eef2 += et[i] * et[i];                          // equivalent tensile strain, squared.
  }
  eef = sqrt(eef2);                                 // equivalent tensile strain
  eef2 += 1e-10;                                    // avoid divide by zero

  alphat = 0.0, alphac = 0.0;
  for (i = 0; i < 3; i++) {
    ets = (1.0 + nu0) / E0 * st[i] - nu0 / E0 * stsum; // define tensile strain from tensile stress 
and undamaged stiffness (Mazars' model)
	ecs = ep[i] - ets;                                 // define compressive strain (Mazars' 
model)
    alphat +=  ets * MAX(ep[i], 0.0) / eef2;           // tensile weight (Mazars' model)
    alphac +=  ecs * MAX(ep[i], 0.0) / eef2;           // compressive weight (Mazars' model)
  }

  //Initialize damage variable with stored value
  damage = states[1];

  // Increase damage and update states only if the equivalent tensile strain increases its previous 
value
  if(eef > kappa) {
    kappa = eef; // update kappa to the current value of equivalent strain
    // Define tensile and compressive damage variables (Mazars' model)
    damt = 1.0 - kappa0 * (1.0 - At) / kappa - At * exp(-Bt * (kappa - kappa0));
    damc = 1.0 - kappa0 * (1.0 - Ac) / kappa - Ac * exp(-Bc * (kappa - kappa0));
	// Define damage variable
    damage = MAX(damage, alphat * damt + alphac * damc); // increase damage only if larger than its 
previous value stored in the state variable
    damage = MIN(MAX(0.0, damage), 0.99); // Limit the damage variable between 0 and 0.99. Full 
damage means 0.99 stiffness reduction.
	states[0] = eef;     // use this state to memorize the historical maximum value of equivalent 
strain
	states[1] = damage;  // update the state with the new value of the damage variable
  }

  // Define damaged Young's modulus
  E = (1.0 - damage) * E0;

  // Set up Jacobian matrix from damaged Young's modulus
  // 6x6 Elasticity matrix D based on E and nu0, initially filled by zeros
  for (i = 0; i < 6; i++){
      for (j = 0; j < 6; j++) {
          D[i][j] = 0.0;
```

```
      }
  }
  D[0][0] = D[1][1] = D[2][2] =  E * (1.0 - nu0) / (1.0 + nu0) / (1.0 - 2.0 * nu0);
// upper diagonal
  D[3][3] = D[4][4] = D[5][5] =  E / (1.0 + nu0);
// lower diagonal, note that this equals 2G
  D[0][1] = D[0][2] = D[1][0] = D[1][2] = D[2][0] = D[2][1] = E * nu0 / (1.0 + nu0) / (1.0 - 2.0 *
nu0);// off diagonal

  // Compute Hooke's law: s = D*e
  for (i = 0; i < 6; i++){
      s[i] = 0.0;
      for (j = 0; j < 6; j++) {
          s[i] += D[i][j] * e[j];
      }
  }

  // Return value 0 if success, any other value trigger an exception
  return 0;
}
```