```
% total Lagrange formulation for elasto-statics, large deformations
%
%*** mechanical data:
  DATA.thick = 2.0;
  DATA.density = 10000;
  DATA.Eyoung = 15e+7;
  DATA.Epoisson = 0.45;
  DATA.plane_stress = 0;
  DATA.gravity = 10;
  maxiter = 40;
  maxiter0 = 20;
  xtol = 0e-4;

%*** open a figure:
  figure(1);

%*** get the mesh:
  [XY,Elm] = getdrapeau3(401/5,51/5);
  nnod = size(XY,1);
  nelm = size(Elm,1);
  XY(:,1) = XY(:,1)*10;

%*** refine:
  for irefine = 1:0
    [XY,Elm] = refine2D(XY,Elm,0);
  end;
  nnod = size(XY,1);
  nelm = size(Elm,1);

  [G,Border,bfaces] = incidence2(Elm);
  XY0 = XY;

%*** Dirichlet boundary conditions: (X and Y displacement zero):
  nodBC = find(XY0(:,1)<=(min(XY0(:,1))+1e-6));
  pBC = [2*nodBC-1;
         2*nodBC];
  dI = ones(2*nnod,1);
  dI(pBC) = 0;
  dI = spdiags(dI,0,2*nnod,2*nnod);

%*** initial displacement and boundary conditions:
  U0 = zeros(nnod,2);

%*** potential energy Pi on every iteration:
  Energy = zeros(maxiter,1);

%=== begin Newton iteration:

  for iter = 1:maxiter

%*** test reversibility of the processus:
    if (iter==maxiter0)
      fprintf('=== nonlinear calculation done.\n')
      fprintf('   Now set body-force to zero, it should return to initial config\n');
      fprintf('=== Press any key to continue\n');
      pause;
      U1 = U;
    end;

%*** volumic force density on each element:
    EForce = zeros(nelm,2);
    EForce(:,2) = - DATA.density * DATA.gravity * (iter<maxiter0);
```



```
%*** assemble the matrix and right-hand side:
    [K,RHS,Ener] = mknonlinelast2D(XY,Elm,DATA,U0,EForce);

%*** Apply Dirichlet boundary condition:
    RHS = dI'*RHS;
    K = dI'*K*dI + (speye(2*nnod)-dI);

%*** solve the linear problem (Newton method):
    dU = K\RHS;
    dU = reshape(dU,2,nnod)';

%*** residual norm, store the first residual for comparaison:
    normRHS = norm(RHS);
    if (iter==1)
      normRHS1 = normRHS;
    end;

%*** update the solution:
    U = U0 + dU;

%*** visu: current deformed domain:
    sfigure(1); clf
    gplot(G,XY0,'b');
    set(gca,'FontSize',16);
    hold on
    gplot(G,XY+U,'r');
    if (iter>=maxiter0)
      gplot(G,XY+U1,'m');
    end;
    grid on
    axis equal
    axis([-11 11 -16 2])
    title('total Lagrange formulation')
    drawnow

%*** printout:
    fprintf('iter: %4d |RHS|_2: %e |dU|_inf: %e Ener: %e\n',...
      iter,normRHS/normRHS1,max(max(abs(dU))),Ener);

%*** update displacement:
    U0 = U;
    Energy(iter) = Ener;

%*** stopping condition:
    if (normRHS<xtol*normRHS1)
      break;
    end;
  end;

%*** printout:
  fprintf('=== done. Did it return to initial configuration?\n')

%*** clip the energy record only to no. of iterations used:
  Energy = Energy(1:iter,:);


%*** show eps_11, eps_12 and eps_22:
  [EEps,ETau] = getstrainstress2D(XY,Elm,DATA,U);

%*** visu: 2nd Piola-Kirchhoff stress:
if 0
```

```
  figure(11); clf
  ts11=trisurf(Elm,XY(:,1),XY(:,2),XY(:,1).*0,ETau(:,1));
  view(2); axis equal
  set(ts11,'EdgeColor','none');
  colorbar
  drawnow;

  figure(22); clf
  ts22=trisurf(Elm,XY(:,1),XY(:,2),XY(:,1).*0,ETau(:,2));
  view(2); axis equal
  set(ts22,'EdgeColor','none');
  colorbar
  drawnow;

  figure(12); clf
  ts12=trisurf(Elm,XY(:,1),XY(:,2),XY(:,1).*0,ETau(:,3));
  view(2); axis equal
  set(ts12,'EdgeColor','none');
  colorbar
  drawnow;
end;
```



```
function [K,RHS,EnergyU0] = mknonlinelast2D(XY,Elm,DATA,U0,EF,Eps0);
% function [K,RHS,EnergyU0] = mknonlinelast2D(XY,Elm,DATA,U0,EF,Eps0);
%-------------------------------------------------------------------------
% assembles the fixed-point problem for plane-stress or plane-strain
% and large deformations, in total Lagrange formulation.
% Supposes linear continuum, ie.
%   Tij = Eijkl * Ekl(u)   (Tij == 2nd. Piola Kirchhoff stress tensor)
% where Eijkl is a CONSTANT tangent stiffnes modulus, ie. St.Venant-Kirchhoff
% material model with strain energy density
%    Psi(E) =  mu*tr(E*E) + lambda/2*(tr(E))^2
% ATTENTION: delta-formulation: the unknown of the lin. system is dU
%            (Unew <-- U0 + dU).
%*** parameters:
% XY(nnod,2)            -in-    nodal coordinates
% Elm(nelm,3)           -in-    element connectivity (triangles)
% DATA                  -in-    mechanical data (structure)
% DATA.Eyoung           -in-    Young modulus
% DATA.Epoisson         -in-    Poisson ratio in [-1,0.5] (0.5==incompressible)
% DATA.plane_stress     -in-    0: plane strain, otherwise plane stress
% U0(nnod,2)or(2*nnod,1)-in-    old displacement w.r.t initial shape
% EF(nelm,2)or(2*nelm,1)-in-    volumic force density in DEFORMED configuration
% Eps0(nelm,3)          -in-    (facultative) init.strain (eps11,22,12 per elm)
% K(2*nnod,2*nnod)      -out-   sparse stiffness matrix
% RHS(2*nnod,1)         -out-   forces due to initial pre-strain
% EnergyU0              -out-   int_Omega {1/2*Eps(U0)*D*Eps(U0) - f.U0}dx
%-------------------------------------------------------------------------

%--- dimensions:
  ndof = 2;
  nnod = size(XY,1);
  nelm = size(Elm,1);

%--- check shape of input arrays:
  U0 = reshape(U0',2,nnod)';
  EF = reshape(EF',2,nelm)';

%--- allocate new arrays:
  Kelm = zeros(ndof*3,ndof*3,nelm);
  RHS = zeros(ndof*nnod,1);
  if (nargin<6)
    Eps0 = zeros(nelm,3);
  end;
  EnergyU0 = 0;

%*** loop over all elements, pre-store elementary matrices in Kelm():
  for el = 1:nelm
    EXY = XY(Elm(el,:),:);
    EU0 = U0(Elm(el,:),:);

%--- dofs: [ui vi , uj vj , uk vk]:
    [EK,ERHS,EEnergy] = elmelasticity2D(EXY,DATA,EU0,EF(el,:)',Eps0(el,:)');
    EnergyU0 = EnergyU0 + EEnergy;
    Kelm(1:3*ndof,1:3*ndof,el) = EK;
    for i=1:ndof
      RHS(ndof*(Elm(el,:)-1)+i,:) = RHS(ndof*(Elm(el,:)-1)+i,:) + ...
                                    ERHS(i:ndof:3*ndof,:);
    end;
  end;

%*** assemble the global sparse stiffness matrix from Kelm:
  ntot = ndof*nnod;
  K = sparse([],[],[],ntot,ntot);
```

```
  for ei=1:3
    ni = Elm(:,ei);
    for ej=1:3
      nj = Elm(:,ej);
      for i=1:ndof
        for j=1:ndof
          ki = ndof*(ni-1)+i;
          kj = ndof*(nj-1)+j;
          kv = squeeze(Kelm(ndof*(ei-1)+i,ndof*(ej-1)+j,:));
          K = K + sparse(ki,kj,kv,ntot,ntot);
        end;
      end;
    end;
  end;
return;


%-------------------------------------------------------------------------------
% subfunction elmelasticity2D:
%-------------------------------------------------------------------------------

function [EK,ERHS,EEnergy] = elmelasticity2D(XY,DATA,U0,Fvol,Eps0)
  Eyoung = DATA.Eyoung;
  nu = DATA.Epoisson;
  plane_stress = DATA.plane_stress;
  U0 = reshape(U0',2*3,1);
  Fvol = reshape(Fvol,2,1);
  A = [ones(3,1), [XY(1,:);XY(2,:);XY(3,:)]];
  X = inv(A);
  bi = X(2,1);  bj = X(2,2);  bk = X(2,3);
  ci = X(3,1);  cj = X(3,2);  ck = X(3,3);
  Area2x = det(A);
%--- derivatives du_1 /dx and du_2/dx and the same for d/dy:
%--- dof: [ ui  vi ,  uj  vj , uk  vk] :
  d12dx = [bi   0  , bj   0  , bk   0 ;
           0   bi  ,  0  bj  , 0   bk ];
  d12dy = [ci   0  , cj   0  , ck   0 ;
            0  ci  ,  0  cj  ,  0  ck ];
%--- DEps is the Gateau derivative of Eps(U):
  Eps = [  bi   0  , bj   0  , bk   0 ;                % <-- eps_11
           0  ci  ,  0  cj  ,  0  ck ;                 % <-- eps_22
          ci/2  bi/2  , cj/2  bj/2  , ck/2  bk/2 ];    % <-- eps_12
  DEps = Eps;
%--- add the nonlinear part of DEps and Eps:
  DEps = DEps + [ U0'*d12dx'*d12dx;
                  U0'*d12dy'*d12dy;
                  (U0'*d12dx'*d12dy + U0'*d12dy'*d12dx)/2];
  EpsU0 = Eps0 + Eps*U0 + [(U0'*d12dx'*d12dx*U0)/2;
                           (U0'*d12dy'*d12dy*U0)/2;
                           (U0'*d12dx'*d12dy*U0)/2];
%--- calculating the defomation gradient F:
  Fgrad = [d12dx*U0, d12dy*U0] + eye(2,2);
  Uave = [1/3   0, 1/3   0, 1/3   0;
          0 1/3,   0 1/3,   0 1/3];
  I = diag([1 1 2]',0);
  if (plane_stress)
%--- 2D elasticity: plain stress: watch out D(3,3) <--> I(3,3)!
    Eijkl = Eyoung / (1-nu^2) * [1   nu    0;
                                 nu   1    0;
                                 0    0  (1-nu)];
  else
%--- 2D elasticity: plain strain: watch out D(3,3) <--> I(3,3)!
```



```
    Eijkl = Eyoung / ((1+nu)*(1-2*nu)) * [1-nu   nu    0;
                                          nu   1-nu   0;
                                           0     0  (1-2*nu)];
  end;

%*** stress tensor of U0:
  TauU0 = Eijkl*EpsU0;

%--- stiffness matrix and right-hand side:
  Vol = Area2x/2;
  EK = Vol * DEps'*I*Eijkl*DEps + ...
       Vol * (TauU0(1)*d12dx'*d12dx + ...
              TauU0(2)*d12dy'*d12dy + ...
             TauU0(3)*d12dx'*d12dy + TauU0(3)*d12dy'*d12dx);

  ERHS = -Vol * DEps'*I*TauU0 + Vol * Uave'*(Fvol);

  EEnergy = Vol/2 * (EpsU0'*I*TauU0) - Vol * (U0'*Uave'*Fvol);

return;
```

```
function [EEps,ETau] = getstrainstress2D(XY,Elm,DATA,U,Eps0);
% function [EEps,ETau] = getstrainstress2D(XY,Elm,DATA,U,Eps0);
%-------------------------------------------------------------------------
% calculates the Green deformation tensor and the 2nd Piola-Kirchhoff
% stress tensor for the total Lagrange formulation of 2D plane stress or
% plane strain problem (cf. the DATA.plane_stress switch)
% Supposes linear continuum, ie.
% Tij = Eijkl * Ekl(u)   (T == 2nd. Piola Kirchhoff stress tensor)
% Eijkl is a CONSTANT tangent stiffnes modulus, ie. for
% St.Venant-Kirchhoff material
%*** parameters:
% XY(nnod,2)            -in-    nodal coordinates
% Elm(nelm,3)           -in-    element connectivity (triangles)
% DATA                  -in-    mechanical data (structure)
% DATA.thick            -in-    thickness of triangles (global)
% DATA.Eyoung           -in-    Young modulus
% DATA.Epoisson         -in-    Poisson ratio in [-1,0.5] (0.5==incompressible)
% DATA.plane_stress     -in-    0: plane strain, otherwise plane stress
% U(nnod,2)or(2*nnod,1) -in-    displacement w.r.t initial config. (Langrange)
% EF(nelm,2)or(2*nelm,1)-in-    volumic force density in DEFORMED configuration
% Eps0(nelm,3)          -in-    (facultative) init.strain (eps11,22,12 per elm)
% Eps(nelm,3)           -in-    Green deformation tensor (eps11,22,12 per elm)
% Tau(nelm,3)           -in-    2nd Piola-Kirchhof stress (tau11,22,12 per elm)
%-------------------------------------------------------------------------

%--- dimensions:
  ndof = 2;
  nnod = size(XY,1);
  nelm = size(Elm,1);

%--- check shape of input arrays:
  U = reshape(U',2,nnod)';

%--- allocate arrays for tensors:
  EEps = zeros(nelm,3);
  ETau = zeros(nelm,3);
  if (nargin<6)
    Eps0 = zeros(nelm,3);
  end;

%*** loop over all elements, pre-store elementary matrices in Kelm():
  t0 = clock;
  for el = 1:nelm
    EXY = XY(Elm(el,:),:);
    EU = U(Elm(el,:),:);

%--- dofs: [ui vi , uj vj , uk vk]:
    if (nargout<2)
      eeps = elmstrainstress2D(EXY,DATA,EU,Eps0(el,:)');
      EEps(el,:) = eeps';
    else
      [eeps,etau] = elmstrainstress2D(EXY,DATA,EU,Eps0(el,:)');
      EEps(el,:) = eeps';
      ETau(el,:) = etau';
    end;
  end;
return;


%-------------------------------------------------------------------------
% subfunction elmstrainstress2D:
%-------------------------------------------------------------------------
```



```
function [EpsU,TauU] = elmstrainstress2D(XY,DATA,U,Eps0)
  Eyoung = DATA.Eyoung;
  nu = DATA.Epoisson;
  thick = DATA.thick;
  plane_stress = DATA.plane_stress;
  U = reshape(U',2*3,1);
  A = [ones(3,1), [XY(1,:);XY(2,:);XY(3,:)]];
  X = inv(A);
  bi = X(2,1);  bj = X(2,2);  bk = X(2,3);
  ci = X(3,1);  cj = X(3,2);  ck = X(3,3);
  Area2x = det(A);
%--- derivatives du_1 /dx and du_2/dx and the same for d/dy:
%--- dof: [ ui  vi ,  uj  vj , uk  vk] :
  d12dx = [bi   0  , bj   0  , bk   0 ;
           0   bi  ,  0  bj  , 0   bk ];
  d12dy = [ci   0  , cj   0  , ck   0 ;
            0  ci  ,  0  cj  ,  0  ck ];
%--- DEps is the Gateau derivative of Eps(U):
  Eps = [  bi   0  , bj   0  , bk   0 ;                  % <-- eps_11
           0  ci  ,  0  cj  ,  0  ck ;                   % <-- eps_22
          ci/2  bi/2  , cj/2  bj/2  , ck/2  bk/2 ];     % <-- eps_12
%--- add the nonlinear part of Eps:
  EpsU = Eps0 + Eps*U + [(U'*d12dx'*d12dx*U)/2;
                         (U'*d12dy'*d12dy*U)/2;
                         (U'*d12dx'*d12dy*U)/2];
%--- calculating the defomation gradient F:
%  Fgrad = [d12dx*U0, d12dy*U0] + eye(2,2);

  if (nargout<2)
    return;
  end;

%--- 2D elasticity: plain stress: watch out D(3,3)!
  if (plane_stress)
    Eijkl = Eyoung / (1-nu^2) * [1   nu    0;
                                 nu   1    0;
                                 0    0  (1-nu)];
  else
%--- 2D elasticity: plain strain: watch out D(3,3)!
    Eijkl = Eyoung / ((1+nu)*(1-2*nu)) * [1-nu   nu    0;
                                          nu   1-nu   0;
                                           0     0  (1-2*nu)];
  end;

  TauU = Eijkl*EpsU;
return;
```

```
% update Lagrange formulation for elasto-statics, big deformations
%
% run program1.m first, this will give you the mesh, mechanical data
% and Dirichlet boundary conditions
%

%*** mechanical data:
  EDensity0  = DATA.density*ones(nelm,1);

%*** update Lagrange parameters:
  maxiter = 200;
  maxiter0 = 100;
  dtime = 1e-5;

%*** open a figure:
  figure(2); clf

%*** original area of elements:
  EVol0 = elmvol(XY0,Elm);

%*** initial pre-stress:
  Tau = zeros(nelm,3);

%*** initial geometry:
  XY = XY0;


  for iter=1:maxiter

    if (iter==maxiter0)
      fprintf('=== nonlinear calculation done.\n')
      fprintf('   Now, set body-force to zero, it should return to initial config\n');
      fprintf('=== Press any key to continue\n');
%      pause;
      XY1 = XY;
    end;

%*** get new density:
    EVol = elmvol(XY,Elm);
    EDensity = EDensity0.*EVol0./EVol;

%*** get new (total) volumic force w.r.t to new mesh (Elm,XY):
    EForce = zeros(nelm,2);
    EForce(:,2) = -EDensity*DATA.gravity * (iter<maxiter0);

%*** assemble LINEAR elasticity (small deformations with pre-stress):
    [K,RHS] = mklinelast2D(XY,Elm,DATA,EForce,Tau);

%*** mass matrix (for friction term):
    M = mkmass2D(XY,Elm,0);

%*** add mass and stiffness into the global system matrix:
    K(1:2:end,1:2:end) = K(1:2:end,1:2:end) + M/dtime;
    K(2:2:end,2:2:end) = K(2:2:end,2:2:end) + M/dtime;

%*** apply Dirichlet boundary conditions:
    RHS = dI'*RHS;
    K = dI'*K*dI + (speye(2*nnod)-dI);

%*** solve linear elasticity for one update:
    dU = K\RHS;
    dU = reshape(dU',2,nnod)';
```



```
%*** residual norm and first residual for comparaison:
    normRHS = norm(RHS);
    if (iter==1)
      normRHS1  = normRHS;
    end;

%*** additional strain and stress:
    [dEps0,dTau] = getlinstrainstress2D(XY,Elm,dU,DATA);
    Tau = Tau+dTau;
%*** transfer the pre-stress from the old to the new configuration:
    Tau = getlagrange2euler2D(XY,Elm,dU,Tau);

%*** update the mesh:
     XY = XY+dU;

%*** visu: current mesh:
    sfigure(2); clf
    gplot(G,XY0);
    set(gca,'FontSize',16);
    hold on
    gplot(G,XY+dU,'r');
    if (iter>=maxiter0)
      gplot(G,XY1,'m');
    end;
    grid on
    axis equal
    axis([-11 11 -16 2])
    title('update Lagrange formulation');
    drawnow;

%*** visu: (pre) stress:
    if 0
      figure(11); clf
      ts11 = trisurf(Elm,XY(:,1),XY(:,2),0.*XY(:,1),Tau(:,1));
      axis equal;
      view(2);
      set(ts11,'EdgeColor','none');
      colorbar('horiz');
      drawnow;

      figure(22); clf
      ts22 = trisurf(Elm,XY(:,1),XY(:,2),0.*XY(:,1),Tau(:,2));
      axis equal;
      view(2);
      set(ts22,'EdgeColor','none');
      colorbar('horiz');
      drawnow;

      figure(12); clf
      ts12 = trisurf(Elm,XY(:,1),XY(:,2),0.*XY(:,1),Tau(:,3));
      axis equal;
      view(2);
      set(ts12,'EdgeColor','none');
      colorbar('horiz');
      drawnow;
    end;

%*** printout:
    fprintf('iter: %d    |RHS|: %e\n',...
       iter,normRHS/normRHS1);
```

```
%*** stopping condition:
    if (normRHS<xtol*normRHS1)
      break;
    end;
  end;

%*** printout:
  fprintf('=== done. Did it return to initial configuration?\n')
```



```
function [K,RHS] = mklinelast2D(XY,Elm,DATA,EF,Tau0);
% function [K,RHS] = mklinelast2D(XY,Elm,DATA,EF,Tau0);
%-------------------------------------------------------------------------
% computes the stiffnes matrix for plane-stress or plain-strain
% works only for SMALL deformations!
%*** parameters:
% XY(nnod,2)            -in-    nodal coordinates
% Elm(nelm,3)           -in-    element connectivity (triangles)
% DATA                  -in-    mechanical data (structure)
% DATA.plane_stress     -in-    0: plane strain, 1: plane stress
% DATA.Eyoung           -in-    Young modulus
% DATA.Epoisson         -in-    Poisson ratio in [-1,0.5] (0.5==incompressible)
% EF(nelm,2)or(2*nelm,1)-in-    force density on each element
% Tau0(nelm,3)          -in-    (facultative) init.stress (tau11,22,12 per elm)
% K(2*nnod,2*nnod)      -out-   sparse stiffness matrix
% RHS(2*nnod,1)         -out-   forces due to initial pre-strain
%-------------------------------------------------------------------------

  ndof = 2;
  nnod = size(XY,1);
  nelm = size(Elm,1);
  Kelm = zeros(ndof*3,ndof*3,nelm);
  RHS = zeros(ndof*nnod,1);
  t0 = clock;

  EF = reshape(EF',2,nelm)';

  if (nargin<5)
    Tau0 = zeros(nelm,3);
  end;

%*** loop over all elements, pre-store elementary matrices in Kelm():
  for el = 1:nelm
    EXY = XY(Elm(el,:),:);
%--- dofs: [ui vi , uj vj , uk vk]:
    [EK,ERHS] = elmelasticity2D(EXY,DATA,EF(el,:)',Tau0(el,:)');
    Kelm(1:3*ndof,1:3*ndof,el) = EK;
    for i=1:ndof
      RHS(ndof*(Elm(el,:)-1)+i,:) = RHS(ndof*(Elm(el,:)-1)+i,:) + ...
                                    ERHS(i:ndof:3*ndof,:);
    end;
  end;

%*** assemble the global sparse stiffness matrix from Kelm:
  ntot = ndof*nnod;
  K = sparse([],[],[],ntot,ntot);
  for ei=1:3
    ni = Elm(:,ei);
    for ej=1:3
      nj = Elm(:,ej);
      for i=1:ndof
        for j=1:ndof
          ki = ndof*(ni-1)+i;
          kj = ndof*(nj-1)+j;
          kv = squeeze(Kelm(ndof*(ei-1)+i,ndof*(ej-1)+j,:));
          K = K + sparse(ki,kj,kv,ntot,ntot);
        end;
      end;
    end;
  end;
return;
```

```
%-------------------------------------------------------------------------
% subfunction elmelasticity2D:
%-------------------------------------------------------------------------

function [EK,ERHS] = elmelasticity2D(XY,DATA,Fvol,Tau0)
  plane_stress = DATA.plane_stress;
  Eyoung = DATA.Eyoung;
  nu = DATA.Epoisson;
  A = [ones(3,1), [XY(1,:);XY(2,:);XY(3,:)]];
  X = inv(A);
  bi = X(2,1);  bj = X(2,2);  bk = X(2,3);
  ci = X(3,1);  cj = X(3,2);  ck = X(3,3);
  Area2x = det(A);
%--- dof: [ ui  vi ,  uj  vj , uk  vk] :
  Eps = [  bi   0  , bj   0  , bk   0 ;                % <-- eps_11
            0  ci  ,  0  cj  ,  0  ck ;                % <-- eps_22
           ci/2  bi/2  , cj/2  bj/2  , ck/2  bk/2 ];    % <-- eps_12
  Uave = [1 0, 1 0, 1 0;
          0 1, 0 1, 0 1]/3;

  I = diag([1 1 2]',0);

  if (plane_stress)
%--- 2D elasticity: plain stress:
    Eijkl = Eyoung / (1-nu^2) * [1   nu    0;
                                 nu   1    0;
                                 0    0  (1-nu)];
  else
%--- 2D elasticity: plain strain:
    Eijkl = Eyoung / ((1+nu)*(1-2*nu)) * [1-nu   nu    0;
                                          nu   1-nu   0;
                                           0     0  (1-2*nu)];
    Eijkl(3,3) = Eyoung / (1+nu);
  end;

%--- elementary stiffness matrix and right-hand side:
  Vol = Area2x/2;
  EK = Vol * Eps'*Eijkl*I*Eps;
  ERHS = Vol * Uave'*Fvol + ...
        -Vol * Eps'*I*Tau0;

return;
```



```
function [Eps,Tau] = getlinstrainstress2D(XY,Elm,U,DATA,Eps0);
% function [Eps,Tau] = getlinstrainstress2D(XY,Elm,U,DATA,Eps0);
%-------------------------------------------------------------------------
% computes the plane strain of each element, based on the elongations U
% stress tensor might also be calculated
% works for SMALL deformations!
%*** parameters:
% XY(nnod,2)            -in-    nodal coords
% Elm(nelm,3)           -in-    element connectivity
% U(2*nnod,1)           -in-    nodal displacements, U(1:2)=1st node in X and Y
%  or U(nnod,2)
% DATA                  -in-    mechanical data (used only if Tau returned)
% DATA.plane_stress     -in-    1: plane stress, 0: plame strain
% DATA.Eyoung           -in-    Young s modulus
% DATA.Epoisson         -in-    Poisson ratio in [-1,0.5] (0.5==incompressible)
% Eps0(nelm,3)          -in-    (facultative) pre-strain tensor (eps_11,22,12)
% Eps(1..nelm,3)        -out-   eps_11,22,12 components of the strain tensor
% Tau(1..nelm,3)        -out-   tau_11,22,12 components of the stress tensor
%-------------------------------------------------------------------------

  ndof = 2;

  nnod = size(XY,1);
  nelm = size(Elm,1);
  Eps = zeros(nelm,3);
  if (nargout>1)
    Tau = zeros(nelm,3);
  end;
  if (nargin<4)
    DATA = [];
  end;
  if (nargin<5)
    Eps0 = zeros(nelm,3);
  end;

  t0 = clock;

%*** reshape U so that it is (nnod x ndof):
  U = reshape(U',ndof,nnod)';

%*** loop over all elements:
  for el = 1:nelm
    EXY = XY(Elm(el,:),:);
    EU = U(Elm(el,:),:);
    EU = reshape(EU', ndof*3,1);
%*** dofs: [ui vi , uj vj , uk vk]:
    if (nargout==1)
      EEps = elmgetstrain2D(EXY,EU,[],Eps0(el,:)');
      Eps(el,:) = EEps';
    else
      [EEps,ETau] = elmgetstrain2D(EXY,EU,DATA,Eps0(el,:)');
      Eps(el,:) = EEps';
      Tau(el,:) = ETau';
    end;
  end;
return;


%-------------------------------------------------------------------------
% subfunction elmgetstrain2D
%-------------------------------------------------------------------------
```

```
function [EpsU,TauU] = elmgetstrain2D(XY,U,DATA,EpsU0);
  A = [ones(3,1), [XY(1,:);XY(2,:);XY(3,:)]];
  X = inv(A);
  bi = X(2,1); bj = X(2,2); bk = X(2,3);
  ci = X(3,1); cj = X(3,2); ck = X(3,3);
  Area2x = det(A);
%--- dof: [ ui vi , uj vj , uk vk] :
  Eps = [ bi    0  ,   bj   0  ,  bk    0 ;            % <-- eps_11
          0   ci  ,    0  cj  ,   0   ck ;            % <-- eps_22
        ci/2 bi/2 , cj/2 bj/2 , ck/2 bk/2 ];           % <-- eps_12
  EpsU = Eps*U + EpsU0;
%--- calculate also stress tensor:
  if (nargout>1)
    plane_stress = DATA.plane_stress;
    Eyoung = DATA.Eyoung;
    Epoisson = DATA.Epoisson;
    if (plane_stress)
%--- 2D elasticity: plain stress: watch out D(3,3) <--> I(3,3)!
      D = Eyoung / (1-Epoisson^2) * [1   Epoisson    0;
                                     Epoisson   1    0;
                                         0      0  (1-Epoisson)];
    else
%--- 2D elasticity: plain strain: watch out D(3,3) <--> I(3,3)!
      D = Eyoung / ((1+Epoisson)*(1-2*Epoisson)) * [1-Epoisson   Epoisson    0;
                                                      Epoisson   1-Epoisson   0;
                                                          0          0  (1-2*Epoi
sson)];
      D(3,3) = Eyoung / (1+Epoisson);
    end;
    TauU = D*EpsU;
  end;
return;
```