Paper title: MatNLI: An Open-source MATLAB-based solver for the Non-linear Inversion in Elastography

Authors: Abhilash Awasthi, Moirangthem Dinachandra, Puneet Mahajan, Ashish Suri, and Sitikantha Roy

Citation: Awasthi, A., Dinachandra, M., Mahajan, P., Suri, A., and Roy, S., MatNLI: An Open-source MATLAB-based solver for the Non-linear Inversion in Elastography, *Advances in Engineering Software* (2023)

General

This document provides line by line explanation of MATLAB code for the inversion using viscoelastic nearly-incompressible material model. The codes for rest of the test cases have been designed in a similar manner. For elastic material models, the material property vector will be real-valued and hence all the system level matrices and solution vectors will also be real-valued.

1 MATLAB script for synthetic data generation

The MATLAB script in Listing 1 describes the process of synthetic two-dimensional displacement field data generation under plane-strain condition. The description of every variable used is provided next to its definition in the listing. The major steps of the code are described briefly as follow:

1. Firstly material model parameters are defined (lines 6-9).
2. Geometric parameters of the domain are defined such as its length and number of nodes required for discretization (lines 12-15).
3. `getMesh()` function is used to discretize the geometry using linear quadrilateral elements and gather other mesh related information, such as, detection of boundary nodes, nodes for boundary condition application and other model parameters, in a MATLAB structure array named `mesh` (lines 18-22).
4. `getMatProp()` function is then used for generating the nodal distribution for the material property (complex valued shear modulus in the code) (line 25).
5. The generated nodal data can be plotted as a 3D surface using `plotSol()` function whose arguments should be nodal information `(P)`, element connectivity `(NCA)`, values to be plotted (`real(Gvec)`), figure number, and title of the figure (lines 27-28).
6. The stiffness and mass level matrices are assembled using `getStiffnessPar()` function (line 31). The resulting global system matrix and global force vector (lines 32 and 33) are subsequently used for solving the linear system of equations by enforcing the boundary conditions.
7. The unknowns to be solved for are specified by separating the known and unknown degrees of freedom in the system (lines 36-38). The boundary conditions are then enforced in the solution vector `Sol` (lines 40-42).

8. The system is then solved for the unknown degrees of freedoms (line 45).

9. The post-processing part involves splitting the main solution vector `(Sol)` into displacement `(Um)` and pressure `(Pr)` solutions (lines 48-49). The displacement solution can further be split in displacement fields in $x-$ and $y-$directions (lines 51 and 52).

10. The `plotSol()` and `plotPress()` functions are then utilized to plot the displacement and pressure solutions, respectively (lines 55-57).

11. Lastly, the whole data information is saved in MAT-file `data.mat` to be used in inversion (line 60).

```
1  clc;
2  clear all;
3  close all;
4  
5  %% MATERIAL PARAMETERS
6  Nu = 0.4995;              % Poisson's ratio
7  Rho = 1000e-12;           % Density (kg/mm^-3)
8  freq = 100;               % Frequency (Hz)
9  Omega = 2*pi*freq;        % Angular frequency (rad)
10 
11 %% GEOMETRY
12 lx = 120;                 % Length of the domain along x-direction
13 ly = 120;                 % Length of the domain along y-direction
14 nx = 51;                  % No of nodes along x-direction
15 ny = 51;                  % No of nodes along y-direction
16 
17 %% MESHING - MESH AND BOUNDARY NODES INFORMATION
18 mesh = getMesh(lx,ly,nx,ny);
19 nn = mesh.nn;             % Number of nodes
20 ne = mesh.ne;             % Number of elements
21 dDoF = 2*nn;              % Number of displacement DoF
22 prDoF = ne;               % Number of pressure DoF
23 
24 %% COMPLEX VALUED MATERIAL PROPERTY VECTOR
25 Gvec = getMatProp(mesh);              % Complex shear modulus (MPa)
26 
27 plotsol(mesh.P,mesh.NCA,real(Gvec),1,'Storage Shear Modulus')
28 plotsol(mesh.P,mesh.NCA,imag(Gvec),2,'Loss Shear Modulus')
29 
30 %% COMPLEX VALUED SYSTEM MATRICES
31 [M,K,C,V] = getStiffnessPar(mesh,Gvec,Nu,Rho);
32 GK = [K-omega^2*M C; C' -V];          % Global system matrix
33 GF = zeros(dDoF+prDoF,1);             % Global force vector
34 
35 %% BOUNDARY CONDITIONS
36 tDoF = 1:(dDoF+prDoF);                % Total Degrees of Freedom (DoF)
37 kDoF = unique([mesh.DBC mesh.NBC]);   % Known DoF
38 uDoF = setdiff(tDoF,kDoF);            % Unknown DoF
39 
40 Sol = zeros(dDoF+prDoF,1);            % Initialize solution vector
41 Sol(mesh.DBC,1) = 0;                  % Homogeneous BC
42 Sol(mesh.NBC,1) = 1;                  % Dirichlet BC (mm)
43 
44 %% SOLVE FOR THE UNKNOWN DoF
45 Sol(udof,1) = GK(dDoF,uDoF)\(GF(uDoF,1) - GK(uDoF,kDoF)*Sol(kDoF,1));
46 
47 %% POST-PROCESSING
```

```
Um = Sol(1:dDoF,1);                     % Displacement solution
Pr = Sol([(dDoF+1):end],1);             % Pressure solution

Ux = Um(1:2:end);                       % Displacement in x-direction
Uy = Um(2:2:end);                       % Displacement in y-direction

% Plot the data
plotSol(mesh.P,mesh.NCA,real(Ux),3,'Real - Ux');    % Plot x-displacement
plotSol(mesh.P,mesh.NCA,real(Uy),4,'Real - Uy');    % Plot y-displacement
plotPress(mesh.P,mesh.NCA,real(Pr),5,'Real - Pressure'); % Plot pressure

%% SAVING DATA
save('data.mat','mesh','Gvec','Um','Pr','Omega','Rho','Nu')

```

Listing 1: Synthetic data generation

2 Script for Non-linear Inversion

The MATLAB script in Listing 2 describes the process of setting up the non-linear inversion as a PDE constrained optimization problem using Optimization Toolbox of MATLAB. The various steps are described briefly as follow:

1. The data saved in Listing 1 is firstly imported into the workspace (lines 4-5).
2. A homogeneous initial guess for the material parameters is then prescribed. The initial guess can change according to the problem being solved. Since MATLAB function `fminunc()` does not support complex valued parameters, the storage and loss moduli are stacked into a single variable (line 8-9).
3. A function handle for the Objective function `myObjFun()` is then created (line 12).
4. The various controls and options for the optimization algorithm are then specified (lines 15-27). Apart from utilizing user-defined function `plotPar()` for plotting the reconstructed material property map at every iteration, the details and explanation for rest of the parameters can be found at this link (accessed on April 8, 2023).
5. The BFGS optimizer is called for solving the inverse problem using `fminunc()` (line 31).
6. The final reconstructed map and the ground truth can be plotted for comparison (lines 35-39).
7. NOTE: Marking the option `‘SpecifyObjectiveGradient’` as `‘true’` in line 21 direct the optimization solver to utilize the user-defined gradient vector. If turned `‘false’`, MATLAB automatically computes the gradient vector numerically using finite difference method.

```
1 clc; clear all; close all;
2 
3 %% LOAD DATA
4 load('data.mat')          % Load data in workspace
5 nn = mesh.nn;             % Number of material parameters
6 
```

```
%% INITIAL GUESS
X0(1:nu,1) = 0.01*ones(nn,1);         % Initial guess for storage modulus
X0(nn+[1:nn],1) = 0.001*ones(nn,1);  % Initial guess for loss modulus

%% OBJECTIVE FUNCTION DEFINITION
ObjFun = @(X) ObjFun(X, Um, mesh, Nu, Rho, Omega)

%% SETTING OPTIONS FOR OPTIMIZATION MODULE
options = optimoptions('fminunc',...
'PlotFcn',@optimplotfval,...
'OutputFcn', @plotPar,...
'Algorithm','quasi-newton',...
'HessUpdate','bfgs',...
'Display','iter-detailed',...
'SpecifyObjectiveGradient',true,...
'UseParallel',true,...
'MaxIter',10000,...
'MaxFunEvals',1e25,...
'OptimalityTolerance',1e-6,...
'TolX',eps,...
'TolFun',eps);

%% CALL THE OPTIMIZER
tic
X = fminunc(ObjFun, X0, options)
toc

%% POST-PROCESSING
plotSol(mesh.P,mesh.NCA,X(1:nu,1),10,'Predicted Storage Modulus');
plotSol(mesh.P,mesh.NCA,X(nu+[1:nu],1),11,'Predicted Loss Modulus');

plotSol(mesh.P,mesh.NCA,real(Gvec),12,'Actual Storage Modulus');
plotSol(mesh.P,mesh.NCA,imag(Gvec),13,'Actual Loss Modulus');

```

Listing 2: Non-linear inversion

3 Script for Objective Function

The MATLAB script in Listing 3 describes the computation of objective function and its gradient based on adjoint state method using `myObjFun()` function. The various steps are described briefly as follow:

1. The boundary nodes are first defined to enforce the Dirichlet boundary conditions (line 3) followed by other model parameters (lines 4-5).

2. The stacked real-valued material property vector `(X)` is then converted into a complex-valued vector `(Gvec)` (line 8).

3. Based on the latest estimate of material parameter vector `(X)`, the forward problem is solved to compute the solution vector `(Sol)` (lines 11-26). The boundary conditions are enforced using the measured displacement field vector `(Um)` (line 22).

4. The objective function (Eq. 12 in the manuscript) is then calculated using the model computed `(U)` and measured `(Um)` displacement fields (lines 29-30).

5. The gradient of the objective function is then computed wherein the already computed displacement field `(U)` (line 25) is used as Step 1 of gradient computation in the manuscript (refer Eq. 21).

6. Further, for Step 2 (refer Eq. 25 of manuscript), the same model computed displacement field `(U)` and assembled global system matrix `(GK)` (line 12) are then utilized to compute the unknown weights (lines 40-42).

7. The $[\mathbf{K}']$ matrix (given in Eq. 29 of manuscript) is then assembled using function `getAdjointPar()` (line 45) which is then used for computing the gradient.

8. The gradient of the objective function (Eq. 29 of manuscript) is then computed using adjoint state method (line 48).

9. The complex-valued gradient vector (line 48) is then again unpacked into a real-valued gradient vector corresponding to the storage and loss moduli (lines 49-50).

```
1  function [Phi,AdGrad] = ObjbFun(X,Um,mesh,Nu,Rho,Omega)
2  
3  DBC = mesh.BC;       % Boundary nodes
4  nn  = mesh.nn;       % Total number of nodes
5  np  = mesh.ne;       % Total number of elements
6  
7  %% CONVERT REAL-VALUED STIFFNESS VECTOR TO COMPLEX-VALUED
8  Gvec = X(1:nn,1) + 1i*X(nn+[1:nn],1);
9  
10 %% SYSTEM MATRICES
11 [M,K,C,V] = getStiffnessPar(mesh,Gvec,Nu,Rho);
12 GK = [K-Omega^2*M C; C' -V];
13 GF = zeros(2*nn+np,1);
14 Sol = zeros(2*nn+np,1);
15 
16 %% BOUNDARY CONDITIONS
17 tDoF = 1:(2*nn+np);
18 kDoF = unique([2*DBC-1 2*DBC]);
19 uDoF = setdiff(tdof,kdof);
20 
21 %% SOLVE FOR UNKNOWNS
22 Sol(kDoF,1) = Um(kDoF,1);
23 Sol(uDoF,1) = GK(uDoF,uDoF)\(GF(uDoF,1)-GK(uDoF,kDoF)*Sol(kDoF,1));
24 
25 U = Sol(1:2*nn,1);
26 Pr = Sol([(2*nn+1):end],1);
27 
28 %% COMPUTE OBJECTIVE FUNCTION
29 err = (U - Um);
30 Phi = 0.5*(err'*err);
31 
32 %% ADJOINT BASED GRADIENT COMPUTATION
33 if nargout > 1
34     % Forcing function for difference driven problem
35     GF = zeros(2*nn+np,nn);
36     GF(1:2*nn,1) = -conj(U-Um);
37     GF((2*nn+1):end,1) = 0;
38 
39     % Computing unknown weights - Step 2
40     W = zeros(2*nn+np,1);
41     W(kDoF,1) = 0.0;
```

```
    W(uDoF,1) = GK(uDoF,uDoF)\(GF(uDoF,1));

    % Assembly of [K'] matrix
    Gmat = getAdjointPar(Gvec,mesh,Nu,U,Pr);

    % Compute gradient - Step 3
    Grad(:,1) = conj(W.'*Gmat);
    AdGrad(1:nn,1) = real(Grad);
    AdGrad(nn+[1:nn],1) = imag(Grad);
end

```

Listing 3: Objective function and gradient computation

4 Script for assembly of system level matrices

The MATLAB script in Listing 4 describes parallelized version for the assembly of system level matrices (**K**, **M**, **C** and **V**) using `getStiffnessPar()` function. The various steps are described briefly as follow:

1. The mesh related information and other model parameters are unpacked first (lines 3-7).
2. Based on the number of Gauss points in each direction (line 9), the Gauss point locations and their corresponding weights are then extracted using function `Gauss()` (line 10).
3. A parallel for loop (parfor) is then utilized for assembling the element-level matrices into global matrices. Following are the operations performed for each element `'e'`:
   - The information specific to the element `'e'` is then extracted (lines 14-21).
   - System level matrices specific to the element `'e'` are then initialized (lines 23-26).
   - A for loop for the numerical integration based on Gauss-Quadrature is then utilized for computing the element level matrices (line 29).
   - The shape functions and their derivatives are evaluated (lines 35-42) by utilizing the information of location and weights corresponding to the Gauss point `'g'` (lines 30-32).
   - The element level quantities are then interpolated to the Gauss point `'g'` (lines 52-59).
   - The element level matrices are then computed corresponding to Gauss point `'g'` (lines 66-69) and summed over all Gauss points (lines 66-69).
   - The global indices for assembly and corresponding matrices in a vectorized form are then saved for the final assembly later (lines 72-83).
4. The final assembly of the system matrices is done in a sparse matrix (lines 98-101) after vectorizing all the information (lines 87-95).

```
1 function [Mmat,Kmat,Cmat,Vmat] = getStiffnessPar(mesh,Gvec,Nu,Rho)
2
3 P   = mesh.P;         % Nodal coordinates
4 NCA = mesh.NCA;       % Nodal Connectivity Array (NCA)
5 nu  = size(P,1);      % Number of nodes
6 ne  = size(NCA,1);    % Number of elements
7 np  = ne;             % Number of Pressure DoFs
```

```
ngp = 2;                 % Number of Gauss points
[GP,W] = Gauss(ngp);  % Gauss point locations and weights

% Element level loop starts here
parfor e = 1:ne
    a = NCA(e,:);            % Extracting nodes for element 'e'
    xe = P(a,1);             % x-Coordinates of nodes of element 'e'
    ye = P(a,2);             % y-Coordinates of nodes of element 'e'
    Ge = Gvec(a,1);          % Nodal values of shear modulus for element 'e'
    locu = zeros(1,8);       % Location vector for global assembly
    locu(1:2:8) = 2*a-1;     % Locations for u_x
    locu(2:2:8) = 2*a;       % Locations for u_y
    locp = e;                % Locations for pressure (p)

    Me = zeros(8,8);         % Element level mass matrix
    Ke = zeros(8,8);         % Element level stiffness matrix
    Ce = zeros(8,1);         % Mixed terms based on displacment and pressure
    Ve = zeros(1,1);         % Pressure dependent term

    % Computations at Gauss point level
    for g = 1:size(GP,1)
        r = GP(g,1);
        s = GP(g,2);
        w = W(g);

        % Get shape function and their derivatives
        [N,DNr,DNs] = getQ4shp(r,s);
        Jac = [DNr; DNs]*[xe ye];
        detJac = det(Jac);
        DN = Jac\[DNr;DNs];

        Nd = zeros(2,8);
        Nd(1,1:2:8) = N(1,:);
        Nd(2,2:2:8) = N(1,:);

        % Compute strain-displacement matrix (B)
        B = zeros(3,8);
        B(1,1:2:8) = DN(1,:);
        B(2,2:2:8) = DN(2,:);
        B(3,1:2:8) = DN(2,:);
        B(3,2:2:8) = DN(1,:);

        % Interpolate nodal quantities to Gauss points
        E = 2*(N*Ge)*(1+Nu);     % Young's modulus
        mu = E/(2*(1+Nu));       % Shear modulus
        Kb = E/(3*(1-2*Nu));     % Bulk modulus

        % Deviatoric part of constitutive matrix (Dd)
        I0 = 0.5*[2 0 0;0 2 0; 0 0 1];
        m = [1 1 0]';
        Dd = 2*mu*(I0 - 1/3*m*m');

        me = Rho*Nd'*Nd;
        ke = B'*Dd*B;
        ce = B'*m*1;
        ve = 1*1/Kb;

        Me = Me + w*detJac*me;
        Ke = Ke + w*detJac*ke;
```

```
            Ce = Ce + w*detJac*ce;
            Ve = Ve + w*detJac*ve;
        end
        % Store the locationa for assembly
        [i,j] = meshgrid(locu,locu);
        I(e,:) = i(:);
        J(e,:) = j(:);
        K(e,:) = locp;
        [x,y] = meshgrid(locu,locp);
        X(e,:) = x(:);
        Y(e,:) = y(:);

        M1(e,:) = reshape(Me,[numel(Me) 1])';
        K1(e,:) = reshape(Ke,[numel(Ke) 1])';
        C1(e,:) = reshape(Ce.',[numel(Ce) 1])';
        V1(e,:) = Ve;
    end

    % Preparing locations in 1D arrays for assembly
    I = reshape(I,[numel(I) 1]);
    J = reshape(J,[numel(J) 1]);
    X = reshape(X,[numel(X) 1]);
    Y = reshape(Y,[numel(Y) 1]);

    % Preparing data in 1D arrays for assembly
    M1 = reshape(M1,[numel(M1) 1]);
    K1 = reshape(K1,[numel(K1) 1]);
    C1 = reshape(C1,[numel(C1) 1]);

    % Final assembly
    Mmat = sparse(I,J,M1);
    Kmat = sparse(I,J,K1);
    Cmat = sparse(X,Y,C1);
    Vmat = sparse(K,K,V1);

```

Listing 4: Assembly of system level matrices

5 Script for Assembly of $\mathbf{K}'$ matrix

The MATLAB script in Listing 5 describes parallelized version for the assembly of $\mathbf{K}' = \partial \mathbf{K}/\partial \boldsymbol{\theta}$ matrix using `getAdjointPar()` function. The various steps are described briefly as follow:

1. The assembly procedure for this matrix is same as that for the stiffness matrix described in Listing 4.
2. The two matrices $\partial \hat{\boldsymbol{K}}/\partial \boldsymbol{\theta}$ and $\partial \boldsymbol{V}/\partial \boldsymbol{\theta}$ have been initialized and assembled at the element level in the code as `dKdG` and `dVdG` (lines 26-71).
3. NOTE: For compressible material, only $\partial \hat{\boldsymbol{K}}/\partial \boldsymbol{\theta}$ will be assembled.

```
1  function Gmat = getAdjointPar(Gvec,mesh,Nu,Rho,U,Pr)
2
3  P = mesh.P;
4  NCA = mesh.NCA;
5  nu = mesh.nu;
6  ne = mesh.ne;
7  np = ne;
```

```
ngp = 2;                    % Number of Gauss points
[GP,W] = Gauss(ngp);    % Gauss point locations and weights

% Element level loop starts here
parfor e = 1:ne
    a = NCA(e,:);
    xe = P(a,1); ye = P(a,2);
    locu = zeros(1,8);
    locu(1:2:8) = 2*a-1;
    locu(2:2:8) = 2*a;
    locp = a;

    % Get nodal quantities for element 'e'
    Ge = Gvec(locp,1);       % Complex shear modulus
    Ue = U(locu,1);       % Displacement field
    Pre = Pr(e,1);        % Pressure field

    dKdGe = zeros(8,4);     % Matrix corresponding to ∂K/∂θ
    dVdGe = zeros(1,4);     % Matrix corresponding to ∂V/∂θ

    % Computation at Gauss point level
    for g = 1:size(GP,1)
        r = GP(g,1);
        s = GP(g,2);
        w = W(g);

        % Get shape functions and their derivatives
        [N,DNr,DNs] = getQ4shp(r,s);
        Jac = [DNr; DNs]*[xe ye];
        detJac = det(Jac);
        DN = Jac\[DNr;DNs];

        Nd = zeros(2,8);
        Nd(1,1:2:8) = N;
        Nd(2,2:2:8) = N;

        % Compute strain-displacement matrix (B)
        B = zeros(3,8);
        B(1,1:2:8) = DN(1,:);
        B(2,2:2:8) = DN(2,:);
        B(3,1:2:8) = DN(2,:);
        B(3,2:2:8) = DN(1,:);

        % Interpolate nodal quantities to the Gauss points
        E = 2*(N*Ge)*(1+Nu);         % Young's modulus
        mu = N*Ge;                   % Shear modulus
        coef = 1/(3*(1-2*Nu));
        Kb = E*coef;                 % Bulk modulus

        % Deviatoric part of constitutive matrix (Dd)
        I0 = 0.5*[2 0 0;0 2 0; 0 0 1];
        m = [1 1 0]';
        Dd = 2*(I0 - 1/3*m*m');

        % Compute matrices ∂K/∂θ and ∂V/∂θ
        ge = zeros(8,4);
        ge = B'*Dd*(B*Ue)*N;
        he = zeros(1,4);
        he = 1*Pre*(1/Kb^2)*coef*N*2*(1+Nu);
```

```
            dKdGe = dKdGe + w*detJac*ge;
            dVdGe = dVdGe + w*detJac*he;
        end
        % Store the locations for assembly
        [i,j] = meshgrid(locu,locp);
        I(e,:) = i(:);
        J(e,:) = j(:);
        [x,y] = meshgrid(e,locp);
        X(e,:) = x(:);
        Y(e,:) = y(:);
        dKdG(e,:) = reshape(dKdGe.',[numel(dKdGe) 1]);
        dVdG(e,:) = reshape(dVdGe.',[numel(dVdGe) 1]);
    end

    % Preparing locations in 1D arrays for assembly
    I = reshape(I,[numel(I) 1]);
    J = reshape(J,[numel(J) 1]);
    X = reshape(X,[numel(X) 1]);
    Y = reshape(Y,[numel(Y) 1]);

    % Preparing data in 1D arrays for assembly
    dKdG = reshape(dKdG, [numel(dKdG) 1]);
    dVdG = reshape(dVdG, [numel(dVdG) 1]);

    % Final assembly
    Gmat1 = sparse(I,J,dKdG);  % Terms corresponding to ∂K/∂θ
    Gmat2 = sparse(X,Y,dVdG);  % Terms corresponding to ∂V/∂θ

    Gmat = [Gmat1; Gmat2];
```

Listing 5: Assembly of Gmat matrix