On the polar decomposition of the deformation gradient tensor

Kevin Moerman

The deformation gradient tensor $\mathbf{F}$ can map a line element in the initial state $\mathrm{d}\mathbf{X}$ to the corresponding line element (in terms of size and orientation) in the final state $\mathrm{d}\mathbf{x}$, which is expressable as:

$$\mathrm{d}\mathbf{x} = \mathbf{F}\mathrm{d}\mathbf{X}$$

The polar decomposition of the deformation gradient tensor produces:

$$\mathbf{F} = \mathbf{Q}\mathbf{U} = \mathbf{V}\mathbf{Q}$$

Where $\mathbf{Q}$ is a rotation tensor, and $\mathbf{U}$ and $\mathbf{V}$ are known as the the right and left stretch tensors respectively. Hence the deformation event due to $\mathbf{F}$ can be thought of as a stretching followed by a rotation ($\mathbf{QU}$) or equivalently a rotation followed by a stretching ($\mathbf{VQ}$). Which in equation form can be presented as:

$$\mathrm{d}\mathbf{x} = \underbrace{\overbrace{\mathbf{F}}^{\mathbf{QU}}}_{\mathbf{VQ}}\mathrm{d}\mathbf{X}$$

Although the eigen vectors for $\mathbf{U}$ and $\mathbf{V}$ differ, they share the same eigen values, i.e. the principal stretches λ_1, λ_2, and λ_3.

Obtaining the rotation tensor $\mathbf{Q}$, and the stretch tensors $\mathbf{U}$ and $\mathbf{V}$, as well as their eigen vectors and eigen values, can be conveniently achieved through the use of the singular value decomposition of $\mathbf{F}$, which is given by:

$$\mathbf{F} = \mathbf{W}\mathbf{\Sigma}\mathbf{R}$$

Here $\mathbf{W}$ and $\mathbf{R}$ are rotation tensors, and $\mathbf{\Sigma}$ is the singular value matrix. This enables the computation of the rotation tensor $\mathbf{Q}$ through:

$$\mathbf{Q} = \mathbf{W}\mathbf{R}^\top$$

and the computation of the right stretch tensor $\mathbf{U}$ through:

$$\mathbf{U} = \mathbf{R}^\top\mathbf{F} = \mathbf{R}\mathbf{\Sigma}\mathbf{R}^\top$$

and the computation of the left stretch tensor $\mathbf{V}$ through:

$$\mathbf{V} = \mathbf{F}\mathbf{R}^\top = \mathbf{W}\mathbf{\Sigma}\mathbf{W}^\top$$

The eigenvalues or principal stretches are obtained from the singular value matrix $\mathbf{\Sigma}$ through:

$$\lambda_i = \Sigma_{ii}$$

The eigenvectors for the right stretch tensor $\mathbf{U}$ (or more formally of $\mathbf{C} = \mathbf{F}^\top\mathbf{F}$ but with which they coincide) are obtained from the columns of $\mathbf{R}$:

$$\mathbf{n_j} = \sum_{i=1}^{3} R_{ij}\mathbf{e_i}$$

Similarly the eigenvectors for the left stretch tensor $\mathbf{V}$ (or more formally of $\mathbf{B} = \mathbf{F}\mathbf{F}^\top$ but with which they coincide) are obtained from the columns of $\mathbf{W}$:

$$\mathbf{m_j} = \sum_{i=1}^{3} W_{ij}\mathbf{e_i}$$

Numerical implementation

Setting up required packages. We need some here for rotations and linear algebra.

```
# Loading required packages
using Rotations; using LinearAlgebra;
```

A simulated deformation gradient tensor $\mathbf{F}$ is now created. This is done by first defining some known principal stretches λ_i, using this to define a known right stretch tensor $\mathbf{U}$, and finally rotating this stretch tensor using a rotation matrix $\mathbf{Q}$ to obtain $\mathbf{F}$ from:

$$\mathbf{F} = \mathbf{Q}\mathbf{U}$$

Let's first define the principal stretches λ_i:

```
λ₁_true = 1.23
```

```
λ₁_true = 1.23
```

```
λ₂_true = 1.05
```

```
λ₂_true = 1.05
```

```
λ₃_true = 0.85
```

```
λ₃_true = 0.85
```

Now use λ_i to define $\mathbf{U}$, the right stretch tensor:

```
U_true = 3×3 Diagonal{Float64, Vector{Float64}}:
          1.23   ⋅     ⋅
           ⋅    1.05   ⋅
           ⋅     ⋅    0.85
```

```
U_true = Diagonal([λ₁_true, λ₂_true, λ₃_true])
```

Now let's create an arbitrary rotation tensor $\mathbf{Q}$, here a triplet of Euler angles, each $\frac{\pi}{4}$ in radians, is used:

```
Q_true =
3×3 RotXYZ{Float64} with indices SOneTo(3)×SOneTo(3)(0.785398, 0.785398, 0.785398):
 0.5       -0.5        0.707107
 0.853553   0.146447  -0.5
 0.146447   0.853553   0.5
```

```
# Create true rotation matrix Q
Q_true = RotXYZ(0.25*π,0.25*π,0.25*π)
```

Finally this lets us define $\mathbf{F}$:

```
F = 3×3 Matrix{Float64}:
     0.615     -0.525      0.601041
     1.04987    0.153769  -0.425
     0.180129   0.896231   0.425
```

```
# Create simulated deformation gradient tensor F
F = Q_true*U_true
```

Next our job is to retrieve $\mathbf{U}$, $\mathbf{V}$, $\mathbf{C}$, λ_i (and various strain metrics) from $\mathbf{F}$.

Using the singular value decomposition

One approach is to use the sinular value decomposition of $\mathbf{F}$, i.e.:

$$\mathbf{F} = \mathbf{W}\mathbf{\Sigma}\mathbf{R}$$

.

In Julia this can be achieved using `LinearAlgebra`'s `svd` function.

```
Enter cell code...
```

```
SVD{Float64, Float64, Matrix{Float64}, Vector{Float64}}
U factor:
3×3 Matrix{Float64}:
 -0.5        0.5        0.707107
 -0.853553  -0.146447  -0.5
 -0.146447  -0.853553   0.5
singular values:
3-element Vector{Float64}:
 1.2300000000000002
 1.05
 0.8499999999999999
Vt factor:
3×3 Matrix{Float64}:
 -1.0  -0.0  -0.0
 -0.0  -1.0  -0.0
  0.0   0.0   1.0
```

```
W, Σ, R = svd(F)
```

```
Q_svd = 3×3 Matrix{Float64}:
         0.5       -0.5        0.707107
         0.853553   0.146447  -0.5
         0.146447   0.853553   0.5
```

```
Q_svd = W*R'
```

```
U_svd = 3×3 Matrix{Float64}:
         -0.615      0.525     -0.601041
         -1.04987   -0.153769   0.425
          0.180129   0.896231   0.425
```

```
U_svd = R'*F
```

```
V_svd = 3×3 Matrix{Float64}:
         -0.615      0.525      0.601041
         -1.04987   -0.153769  -0.425
         -0.180129  -0.896231   0.425
```

```
V_svd = F*R'
```

```
λ_principal_svd =  [1.23, 1.05, 0.85]
```

```
λ_principal_svd = Σ
```

```
n₁ =  [-1.0, 0.0, 0.0]
```

```
n₁ = R*[1.0,0.0,0.0]
```

```
n₂ =  [0.0, -1.0, 0.0]
```

```
n₂ = R*[0.0,1.0,0.0]
```

```
n₃ =  [0.0, 0.0, 1.0]
```

```
n₃ = R*[0.0,0.0,1.0]
```

Using the eigen decomposition of the right Cauchy-Green tensor

Alternatively the eigen decomposition can be used. First we obtain a symmetric tensor by computing the right Cauchy-Green tensor $\mathbf{C}$ from: $\mathbf{C} = \mathbf{F}^{\top}\mathbf{F}$

```
C = 3×3 Matrix{Float64}:
      1.5129        0.0           -5.55112e-17
      0.0           1.1025        -5.55112e-17
     -5.55112e-17  -5.55112e-17    0.7225
```

```
C = F'*F
```

The eigen values of $\mathbf{C}$ are the squared principle stretches ${\lambda_i}^2$

```
md"""
The eigen values of $\mathbf{C}$ are the squared principle stretches ${\lambda_i}^2$
"""
```

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 0.7224999999999999
 1.1024999999999998
 1.5129000000000001
vectors:
3×3 Matrix{Float64}:
 0.0  -2.22045e-16  -1.0
 0.0  -1.0           6.66134e-16
 1.0   0.0           0.0
```

```
λ_sq, Q_eig = eigen(C)
```

To obtain the principal stretches λ_i we simply take the square root of the eigen values of $\mathbf{C}$

```
λ_principal =  [0.85, 1.05, 1.23]
```

```
λ_principal = .√λ_sq
```

Next we can form the right stretch tensor $\mathbf{U}$ in the principal component coordinate system by forming a diagonal matrix using the principal stretches.

```
U_prin_eig = 3×3 Diagonal{Float64, Vector{Float64}}:
              0.85   ⋅     ⋅
               ⋅    1.05   ⋅
               ⋅     ⋅    1.23
```

```
U_prin_eig =  Diagonal(λ_principal)
```

And the same can be done with the squared principal stretches to get $\mathbf{C}$ in the principal component system.

```
C_prin_eig = 3×3 Diagonal{Float64, Vector{Float64}}:
              0.7225   ⋅       ⋅
               ⋅      1.1025   ⋅
               ⋅       ⋅      1.5129
```

```
C_prin_eig =  Diagonal(λ_sq)
```

Next we can rotate the right stretch tensor in the principal component system to our regular system using the rotation tensor $\mathbf{Q}$

```
md"""
Next we can rotate the right stretch tensor in the principal component system to our regular system using the rotation tensor $\mathbf{Q}$
"""
```

```
U_eig = 3×3 Matrix{Float64}:
  1.23          -5.86198e-16  0.0
 -5.86198e-16   1.05          0.0
  0.0           0.0           0.85
```

```
U_eig = Q_eig*U_prin_eig*Q_eig'
```

And similarly for the right Cauchy green tensor.

```
C_eig = 3×3 Matrix{Float64}:
  1.5129       -7.6299e-16  0.0
 -7.6299e-16   1.1025       0.0
  0.0          0.0          0.7225
```

```
C_eig = Q_eig*C_prin_eig*Q_eig'
```

Deriving strain metrics

The natural or logarithmic strain:

$$\mathbf{E}_{log} = log(\mathbf{U})$$

```
E_log_eig = 3×3 Matrix{Float64}:
  0.207014     -1.27066e-16  -0.0
 -1.27066e-16   0.0487902     0.0
 -0.0           0.0          -0.162519
```

```
E_log_eig = Q_eig*Diagonal(log.(λ_principal))*Q_eig'
```

The Green-Lagrange strain:

$$\mathbf{E}_{GL} = \frac{1}{2}(\mathbf{U}^2 - \mathbf{I})$$

```
E_GL = 3×3 Matrix{Float64}:
 -0.0129       9.68559e-17  0.0
  9.68559e-17  0.3975       0.0
  0.0          0.0          0.7775
```

```
E_GL = Q_eig*Diagonal(0.5 .- (λ_principal.^2.0 .- 1.0))*Q_eig'
```

The Hencky ("linear") strain:

$$\mathbf{E}_H = \mathbf{U} - \mathbf{I}$$

```
E_lin = 3×3 Matrix{Float64}:
          0.23          -1.42109e-16  -0.0
         -1.42109e-16   0.05           0.0
         -0.0           0.0           -0.15
```

```
E_lin = Q_eig*Diagonal(λ_principal .- 1.0)*Q_eig'
```

Some other Seth-Hill strain:

$$\mathbf{E}_{SH} = \frac{1}{m}(\mathbf{U}^m - \mathbf{I})$$

```
3×3 Matrix{Float64}:
  0.286956     -1.79484e-16  -0.0
 -1.79484e-16   0.0525417     0.0
 -0.0           0.0          -0.128625
```

```
m=3.0; E_SH = Q_eig*Diagonal(1.0/m .* (λ_principal.^m .- 1.0))*Q_eig'
```

```
Enter cell code...
```