
Math270A Programming 1
Chenfanfu Jiang
---
1. Deformation Gradient

2D:

F = (0 , 0
    0 , -1)

3D:

F = (0.5 , 0 , 0
       0 , 1 , 0
       0 , 0 , 0.333333)
---
2. SVD

2D:

F = (1 , 3
    2 , 4)

U = (0.576048 , -0.817416
    0.817416 , 0.576048)

Sigma = (5.46499 , -0.365966)

V = (0.404554 , -0.914514
    0.914514 , 0.404554)

3D:

F = (1 , 4 , 7
    2 , 5 , 8
    3 , 6 , 9)

U = (-0.479671 , -0.776691 , -0.408248
    -0.572368 , -0.0756865 , 0.816497
    -0.665064 , 0.625318 , -0.408248)

Sigma = (16.8481 , 1.06837 , 1.81299e-16)

V = (-0.214837 , 0.887231 , 0.408248
    -0.520587 , 0.249644 , -0.816497
    -0.826338 , -0.387943 , 0.408248)

---

3. Differentiating SVD

2D:

F = (1 , 3
2 , 4)

DeltaF = (1 , 0
0 , 1)

DeltaSigma = (0.980581 , 0
0 , 0.980581)

DeltaU = (-0.0314391 , -0.0221557
0.0221557 , -0.0314391)

DeltaV = (0.0351736 , 0.0155598
-0.0155598 , 0.0351736)

3D:

F = (1 , 4 , 7
2 , 5 , 8
3 , 6 , 9)

DeltaF = (1 , 0 , 0
0 , 1 , 0
0 , 0 , 1)

DeltaSigma = (0.950586 , 0 , 0
0 , -0.950586 , 0
0 , 0 , -1)

DeltaU = (0.0152811 , -0.00943735 , 3.17914e-09
0.0014891 , -0.0112611 , 3.0975e-10
-0.0123029 , -0.0130849 , -2.55964e-09)

DeltaV = (-0.0174559 , -0.00422684 , 3.43447e-09
-0.00491165 , -0.0102424 , 9.29251e-10
0.00763263 , -0.0162579 , -1.57597e-09)

Related Code:

```
   static MATRIX_2X2<T> Deformation_Gradient(const VECTOR_2D<T> &X1,const
VECTOR_2D<T> &X2,const VECTOR_2D<T> &X3,const VECTOR_2D<T> &x1,const
VECTOR_2D<T> &x2,const VECTOR_2D<T> &x3){
      VECTOR_2D<T> es2,es3,em2,em3;
      es2=x2-x1;es3=x3-x1;em2=X2-X1;em3=X3-X1;
      MATRIX_2X2<T> Ds(es2,es3);
      MATRIX_2X2<T> Dm(em2,em3);
      T eps=1e-15;
      assert(Dm.Determinant()>=eps||Dm.Determinant()<=-eps);
      Dm.Invert();
      return Ds*Dm;
   }

   static MATRIX_3X3<T> Deformation_Gradient(const VECTOR_3D<T> X1,const
VECTOR_3D<T> X2,const VECTOR_3D<T> X3,const VECTOR_3D<T> X4,const
VECTOR_3D<T> x1,const VECTOR_3D<T> x2,const VECTOR_3D<T> x3,const VECTOR_3D<T>
x4){
      MATRIX_3X3<T> Ds(x2(0)-x1(0),x2(1)-x1(1),x2(2)-x1(2),x3(0)-x1(0),x3(1)-x1(1),x3(2)-
x1(2),x4(0)-x1(0),x4(1)-x1(1),x4(2)-x1(2));
      MATRIX_3X3<T> Dm(X2(0)-X1(0),X2(1)-X1(1),X2(2)-X1(2),X3(0)-X1(0),X3(1)-X1(1),X3(2)-
X1(2),X4(0)-X1(0),X4(1)-X1(1),X4(2)-X1(2));
      T eps=1e-15;
      assert(Dm.Determinant()>=eps||Dm.Determinant()<=-eps);
      Dm.Invert();
      return Ds*Dm;
   }

   void SVD(MATRIX_2X2<T>& U,VECTOR_2D<T>& sigma,MATRIX_2X2<T>& V,const T
tol=(T)1e-10)const{
      // Compute V using FtF
      MATRIX_2X2<T> F=(*this);
      MATRIX_2X2<T> Ft=this->Transposed();
      MATRIX_2X2<T> FtF=Ft*F;
      T c,s;
      if(std::abs(FtF(1,0))<tol){
         c=1;s=0;}
      else{
         T tau=(FtF(0,0)-FtF(1,1))/(2.0*FtF(1,0));
         T t1=tau+std::sqrt(1+tau*tau),t2=tau-std::sqrt(1+tau*tau),t;
         if(std::abs(t1)>std::abs(t2)) t=t2;
         else t=t1;
         c=1.0/std::sqrt(1.0+t*t);
         s=t*c;}
      V=MATRIX_2X2<T>(c,-s,s,c);
```

```
        // Sort V
        MATRIX_2X2<T> Sigma_sq=V.Transposed()*F.Transposed()*F*V;
        if(Sigma_sq(0,0)<Sigma_sq(1,1)){
           V=MATRIX_2X2<T>(s,c,-c,s);}

        // FV=QR=USigma
        MATRIX_2X2<T> FV=F*V;

        if(std::abs(FV(1,0))<tol){
           c=1;s=0;}
        else{
           T alpha=1.0/std::sqrt(FV(0,0)*FV(0,0)+FV(1,0)*FV(1,0));
           s=-FV(1,0)*alpha;
           c=FV(0,0)*alpha;}
        U=MATRIX_2X2<T>(c,-s,s,c);
        MATRIX_2X2<T> Sigma=U.Transposed()*F*V;
        sigma(0)=Sigma(0,0);
        sigma(1)=Sigma(1,1);

        // sign convention
        if((sigma(0)>0&&sigma(1)<0&&std::abs(sigma(1))>std::abs(sigma(0)))||
(sigma(0)<0&&sigma(1)>0&&std::abs(sigma(0))>=std::abs(sigma(1)))||
(sigma(0)<=0&&sigma(1)<=0)){
           sigma(0)=-sigma(0);
           sigma(1)=-sigma(1);
           U=(-1.0)*U;}
    }

    void Delta_Sigma(const MATRIX_2X2<T>& delta_F, VECTOR_2D<T>& delta_sigma)const{
        MATRIX_2X2<T> F=*this,U,V,UtDFV;
        VECTOR_2D<T> sigma;
        SVD(U,sigma,V);
        UtDFV=U.Transposed()*delta_F*V;
        delta_sigma=VECTOR_2D<T>(UtDFV(0,0),UtDFV(1,1));
    }

    void Delta_SVD(const MATRIX_2X2<T>& delta_F,VECTOR_2D<T>&
delta_sigma,MATRIX_2X2<T>& delta_U,MATRIX_2X2<T>& delta_V)const{
        MATRIX_2X2<T> F=*this;
        MATRIX_2X2<T> U,V,UtDFV;
        VECTOR_2D<T> sigma;
        F.SVD(U,sigma,V);
        if(std::abs(sigma(0)-sigma(1))<1e-10 || std::abs(sigma(0)+sigma(1))<1e-10) std::cout<<"FATAL
ERROR: Weird case detected -- repeated sigma.\n";

        UtDFV=U.Transposed()*delta_F*V;
        delta_sigma=VECTOR_2D<T>(UtDFV(0,0),UtDFV(1,1));
```

```
    VECTOR_2D<T> rhs(UtDFV(1,0),UtDFV(0,1));
    MATRIX_2X2<T> A(-sigma(0),sigma(1),sigma(1),-sigma(0));
    A.Invert();
    VECTOR_2D<T> xy=A*rhs;
    T x=xy(0);
    MATRIX_2X2<T> helperU(0,x,-x,0);
    delta_U=(helperU*(U.Transposed())).Transposed();
    T y=xy(1);
    MATRIX_2X2<T> helperV(0,y,-y,0);
    delta_V=(helperV*(V.Transposed())).Transposed();
  }


  void SVD(MATRIX_3X3<T>& U,VECTOR_3D<T>& sigma,MATRIX_3X3<T>& V,const T
tol=(T)1e-10,const int max_iterations=20)const {
    T c,s;
    MATRIX_2X2<T> A12,A23,A13;
    MATRIX_3X3<T> G12,G23,G13;
    MATRIX_3X3<T> F=*this;
    MATRIX_3X3<T> A=F.Transposed()*F;

    // Jacobi iterations for V
    V=MATRIX_3X3<T>::Identity();
    for(int iter=0;iter<max_iterations;iter++){
      A12=MATRIX_2X2<T>(A(0,0),A(1,0),A(0,1),A(1,1));
      if(fabs(A12(1,0))<tol){
        c=1;s=0;}
      else{
        T tal=(A12(0,0)-A12(1,1))/(2*A12(1,0));
        T t1=tal+sqrt(tal*tal+1);
        T t2=tal-sqrt(tal*tal+1);
        T t=fabs(t1)<fabs(t2)?t1:t2;
        c=1/sqrt(1+t*t);
        s=t*c;}
      G12=MATRIX_3X3<T>(c,-s,0,s,c,0,0,0,1);
      A=G12.Transposed()*A*G12;
      V=V*G12;

      A23=MATRIX_2X2<T>(A(1,1),A(2,1),A(1,2),A(2,2));
      if(fabs(A23(1,0))<tol){
        c=1;s=0;}
      else{
        T tal=(A23(0,0)-A23(1,1))/(2*A23(1,0));
        T t1=tal+sqrt(tal*tal+1);
        T t2=tal-sqrt(tal*tal+1);
        T t=fabs(t1)<fabs(t2) ? t1:t2;
        c=1/sqrt(1+t*t);
        s=t*c;}
      G23=MATRIX_3X3<T>(1,0,0,0,c,-s,0,s,c);
```

```
    A=G23.Transposed()*A*G23;
    V=V*G23;

    A13=MATRIX_2X2<T>(A(0,0),A(2,0),A(0,2),A(2,2));
    if(fabs(A13(1,0))< tol){
       c=1; s=0;}
    else{
       T tal=(A13(0,0)-A13(1,1))/(2*A13(1,0));
       T t1=tal+sqrt(tal*tal+1);
       T t2=tal-sqrt(tal*tal+1);
       T t=fabs(t1)<fabs(t2) ? t1:t2;
       c=1/sqrt(1+t*t);
       s=t*c;}
    G13=MATRIX_3X3<T>(c,0,-s,0,1,0,s,0,c);
    A=G13.Transposed()*A*G13;
    V=V*G13;}

A=F.Transposed()*F;

// Sort sigma squared and rearrange V
MATRIX_3X3<T> Sigma_sq=V.Transposed()*A*V;
if(Sigma_sq(2,2)>Sigma_sq(1,1)){
    VECTOR_3D<T> temp=V.Column(2);
    V(0,2)=-V(0,1);
    V(1,2)=-V(1,1);
    V(2,2)=-V(2,1);
    V(0,1)=temp.x();
    V(1,1)=temp.y();
    V(2,1)=temp.z();
    T temp_s=Sigma_sq(1,1);
    Sigma_sq(1,1)=Sigma_sq(2,2);
    Sigma_sq(2,2)=temp_s;}
if(Sigma_sq(1,1)>Sigma_sq(0,0)){
    VECTOR_3D<T> temp=V.Column(0);
    V(0,0)=-V(0,1);
    V(1,0)=-V(1,1);
    V(2,0)=-V(2,1);
    V(0,1)=temp.x();
    V(1,1)=temp.y();
    V(2,1)=temp.z();
    T temp_s=Sigma_sq(1,1);
    Sigma_sq(1,1)=Sigma_sq(0,0);
    Sigma_sq(0,0)=temp_s;}
if(Sigma_sq(2,2)>Sigma_sq(1,1)){
    VECTOR_3D<T> temp=V.Column(2);
    V(0,2)=-V(0,1);
    V(1,2)=-V(1,1);
    V(2,2)=-V(2,1);
    V(0,1)=temp.x();
```

```
        V(1,1)=temp.y();
        V(2,1)=temp.z();
        T temp_s=Sigma_sq(1,1);
        Sigma_sq(1,1)=Sigma_sq(2,2);
        Sigma_sq(2,2)=temp_s;}

    // QR to get U and Sigma
    MATRIX_3X3<T> FV=(*this)*V;
    MATRIX_3X3<T> Sigma;
    FV.QR(U,Sigma,tol);
    sigma=VECTOR_3D<T>(Sigma(0,0),Sigma(1,1),Sigma(2,2));

    // sign convention
    int N_negative=(sigma(0)<0)+(sigma(1)<0)+(sigma(2)<0);
    if(N_negative==0) return;
    if(N_negative==1){
        if(sigma(0)<0){
            sigma(0)=-sigma(0);
            sigma(2)=-sigma(2);
            for(int i=0;i<3;i++) V(i,0)=-V(i,0);
            for(int i=0;i<3;i++) V(i,2)=-V(i,2);}
        else if(sigma(1)<0){
            sigma(1)=-sigma(1);
            sigma(2)=-sigma(2);
            for(int i=0;i<3;i++) V(i,1)=-V(i,1);
            for(int i=0;i<3;i++) V(i,2)=-V(i,2);}}
    if(N_negative==2){
        if(sigma(0)>=0){
            sigma(1)=-sigma(1);
            sigma(2)=-sigma(2);
            for(int i=0;i<3;i++) V(i,1)=-V(i,1);
            for(int i=0;i<3;i++) V(i,2)=-V(i,2);}
        else if(sigma(1)>=0){
            sigma(0)=-sigma(0);
            sigma(2)=-sigma(2);
            for(int i=0;i<3;i++) V(i,0)=-V(i,0);
            for(int i=0;i<3;i++) V(i,2)=-V(i,2);}
        else if(sigma(2)>=0){
            sigma(0)=-sigma(0);
            sigma(1)=-sigma(1);
            for(int i=0;i<3;i++) V(i,0)=-V(i,0);
            for(int i=0;i<3;i++) V(i,1)=-V(i,1);}}
    if(N_negative==3){
            sigma(0)=-sigma(0);
            sigma(1)=-sigma(1);
            for(int i=0;i<3;i++) V(i,0)=-V(i,0);
            for(int i=0;i<3;i++) V(i,1)=-V(i,1);}
}
```

```
void QR(MATRIX_3X3<T>&Q,MATRIX_3X3<T>&R,const T tol){
   T c,s;
   MATRIX_3X3<T> G1(0),G2(0),G3(0);
   if(fabs(x[2])<tol){
      c=1;s=0;
      G1(0,0)=G1(1,1)=G1(2,2)=1;}
   else{
      T alpha=1/sqrt(x[1]*x[1]+x[2]*x[2]);
      s=-x[2]*alpha;
      c=x[1]*alpha;
      G1(0,0)=1;
      G1(1,1)=c;
      G1(2,1)=s;
      G1(1,2)=-s;
      G1(2,2)=c;}

   MATRIX_3X3<T> A1;
   A1=G1*(*this);
   if(fabs(A1(1,0))<tol){
      c=1;s=0;
      G2(0,0)=G2(1,1)=G2(2,2)=1;}
   else{
      T alpha=1/sqrt(A1(0,0)*A1(0,0)+A1(1,0)*A1(1,0));
      s=-A1(1,0)*alpha;
      c=A1(0,0)*alpha;
      G2(2,2)=1;
      G2(0,0)=c;
      G2(1,0)=s;
      G2(0,1)=-s;
      G2(1,1)=c;}

   MATRIX_3X3<T> A2;
   A2=G2*A1;
   if(fabs(A2(2,1))<tol){
      c=1;s=0;
      G3(0,0)=G3(1,1)=G3(2,2)=1;}
   else{
      T alpha=1/sqrt(A2(1,1)*A2(1,1)+A2(2,1)*A2(2,1));
      s=-A2(2,1)*alpha;
      c=A2(1,1)*alpha;
      G3(0,0)=1;
      G3(1,1)=c;
      G3(2,1)=s;
      G3(1,2)=-s;
      G3(2,2)=c;}

   Q=(G3*G2*G1).Transposed();
   R=Q.Transposed()*(*this);
}
```

```
   void Delta_Sigma(const MATRIX_3X3<T>& delta_F, VECTOR_3D<T>& delta_sigma)const{
      MATRIX_3X3<T> F=*this,U,V,UtDFV;
      VECTOR_3D<T> sigma;
      SVD(U,sigma,V);
      UtDFV=U.Transposed()*delta_F*V;
      delta_sigma=VECTOR_3D<T>(UtDFV(0,0),UtDFV(1,1));
   }

   void Delta_SVD(const MATRIX_3X3<T>& delta_F,VECTOR_3D<T>&
delta_sigma,MATRIX_3X3<T>& delta_U,MATRIX_3X3<T>& delta_V)const{
      MATRIX_3X3<T> F=*this,U,V,UtDFV;
      VECTOR_3D<T> sigma;
      F.SVD(U,sigma,V);
      UtDFV=U.Transposed()*delta_F*V;
      T sigma1=sigma(0),sigma2=sigma(1),sigma3=sigma(2);
      if(std::abs(sigma1-sigma2)<1e-10||std::abs(sigma1-sigma3)<1e-10||std::abs(sigma2-sigma3)<1e-
10||std::abs(sigma1+sigma2)<1e-10||std::abs(sigma1+sigma3)<1e-10||std::abs(sigma2+sigma3)<1e-10)
std::cout<<"FATAL ERROR: Repeated sigma detected.\n";

      delta_sigma=VECTOR_3D<T>(UtDFV(0,0),UtDFV(1,1),UtDFV(2,2));
      T a=UtDFV(0,1),b=UtDFV(0,2),c=UtDFV(1,2),d=UtDFV(1,0),e=UtDFV(2,0),f=UtDFV(2,1);
      MATRIX_2X2<T> xpM(sigma2,-sigma1,-sigma1,sigma2);xpM.Invert();
      MATRIX_2X2<T> yqM(sigma3,-sigma1,-sigma1,sigma3);yqM.Invert();
      MATRIX_2X2<T> zrM(sigma3,-sigma2,-sigma2,sigma3);zrM.Invert();
      VECTOR_2D<T> xprhs(a,d),yqrhs(b,e),zrrhs(c,f);
      VECTOR_2D<T> xp=xpM*xprhs,yq=yqM*yqrhs,zr=zrM*zrrhs;
      T x=xp(0),p=xp(1),y=yq(0),q=yq(1),z=zr(0),r=zr(1);
      MATRIX_3X3<T> dUtU(0,x,y,-x,0,z,-y,-z,0);
      MATRIX_3X3<T> dVtV(0,p,q,-p,0,r,-q,-r,0);
      delta_U=(dUtU*U.Transposed()).Transposed();
      delta_V=(dVtV*V.Transposed()).Transposed();
   }
```