Chapter 3
Sensitivity Calculation

In the previous chapters we assumed that we have a partial differential equation

$$c(m, u) = 0$$

that involves the fields (or states) u and the "control" parameters (or model) m. We assumed that the model m was known and that we need to solve for the fields u. In this chapter we begin with the inverse problem, that is, the calculation of m given the field (or its projection), u.

As a first and important task we need to evaluate the sensitivity of the fields with respect to changes in the model m. In this chapter we discuss in detail how to compute the sensitivities for different problems.

3.1 The concept of sensitivity and the basic equation

Assume the simulation $c(m, u) = 0$. Given the parameters m we can solve for u as a function of the parameters m that is $u = u(m)$. Given $u(m)$ we obtain the data by another operation. We assume that the data is given by a linear operation that is

$$d = Qu(m).$$

We can think about d as a function of m that is $d = Qu(m)$. The question is, how does d change when we change m. Taylor's series reads

$$Qu(m + hv) = Qu(m) + hQ\frac{\partial u}{\partial m}v + \mathcal{O}(h^2\|v\|).$$

Obviously, for small h the nonlinear problem can be approximated by the linear one. We now define the sensitivity matrix

$$J = Q\frac{\partial u}{\partial m}.$$

The matrix gives an idea to the components in the model that yield (at least locally) large changes to the data.

An important tool that allows to analyze the sensitivities is the Singular Value Decomposition SVD. We assume that the number of data is N which is smaller than the number of parameters M. The SVD of an $N \times M$ matrix J is a decomposition

$$J = U\Lambda V^\top = \sum_{i=1}^{N} \lambda_i u_i v_i^\top$$

Here $U = [u_1, \ldots, u_N]$ is an $N \times N$ left singular vectors orthogonal matrix $UU^\top = U^\top U = I_N$, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ is an $N \times N$ diagonal matrix with the singular values $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$ and $V = [v_1, \ldots, v_N]$ is an $M \times N$ orthogonal matrix $V^\top V = I_N$, of the right singular values.

Assume now a perturbation in the model w. Then we can decompose w into components

$$w = \sum_{i=1}^{N} \alpha_i v_i + w^{\mathrm{orth}}$$

The component w^{orth} contains any components that are not spanned by V which implies that

$$Jw^{\mathrm{orth}} = U\Lambda V^\top w^{\mathrm{orth}} = 0.$$

We therefore see that we can change m in the direction w^{orth} without any change in the data and therefore the data fitting can be done by infinite number of models. We will address this problem in the next section.

Now consider the singular vectors that correspond to the large singular values. A small change in these vectors lead to a large change in the data, that is, the problem is sensitive to change in these directions. While the sensitivity matrix is important for the inverse problem it is also important for better understanding the forward problem. In many cases one can learn about the important components in the model by looking at the sensitivities. We will demonstrate it in an example later in this chapter.

3.2 Computation of the sensitivities - general formulation

Computing the sensitivities is rather straight forward. We have

$$c(m, u) = 0$$

and therefore

$$\nabla_m c(m, u)\delta m + \nabla_u c(m, u)\delta u = 0$$

Note that $\nabla_m c(m, u)$ and $\nabla_u c(m, u)$ are matrices. Furthermore, assuming that the forward problem is solved by a Newton-like method, the matrix $\nabla_u c(m, u)$ is the Jacobian of the forward and assuming that the forward is well posed it is invertible. Manipulating we obtain

$$\delta u = -(\nabla_u c(m, u))^{-1} \nabla_m c(m, u)\delta m$$

and therefore we obtain the formula for the sensitivities

$$J = -Q(\nabla_u c(m,u))^{-1}\nabla_m c(m,u)$$

This is the fundamental sensitivity equation.

The computation of the sensitivities requires the computation of two matrices $\nabla_u c(m,u)$ and $\nabla_m c(m,u)$. We now discuss a number of problems and demonstrate how to compute these matrices in practice. The reader is advised to skip to the section of differentiating linear algebra expressions to brush on linear algebra and multivariable calculus.

3.3 Computation for linear forward problems

Consider the forward problem of the form

$$c(m,u) = A(m)u - q = D^\top \operatorname{diag}(A_v m)Du - q = 0$$

This forward problem is obtained from the nodal discretization of the PDE $\nabla \cdot m\nabla u = q$. Differentiating we obtain

$$\nabla_u(A(m)u - q) = A(m).$$

To differentiate with respect to m we manipulate the equation

$$\nabla_m(A(m)u - q) = \nabla_m(D^\top \operatorname{diag}(A_v m)Du) = \nabla_m(D^\top \operatorname{diag}(A_v m)(Du)) = \nabla_m(D^\top \operatorname{diag}(Du)A_v m) = D^\top \operatorname{diag}(Du)A_v$$

Using the expressions above we obtain that the sensitivities are

$$J = -QA(m)^{-1}D^\top \operatorname{diag}(Du)A_v$$

A similar example is the computation of the sensitivities of the wave field given by Helmholtz equation to its parameter

$$c(m,u) = A(m)u - q = (L + w^2 \operatorname{diag}(m))u - q$$

A similar calculation yields

$$\nabla_u c(m,u) = (L + w^2 \operatorname{diag}(m)).$$

The calculation of the derivatives with respect to m requires slightly more effort

$$\nabla_m c(m,u) = \nabla_m((L + w^2 \operatorname{diag}(m))u) = w^2\nabla_m(\operatorname{diag}(u)m) = w^2 \operatorname{diag}(u).$$

and combining we obtain

$$J = -w^2 QA(m)^{-1} \operatorname{diag}(u)$$

Although the computation of each problem is slightly different it is evidently that the calculation in general leads to the following formula

$$J = -QA(m)^{-1}G(m,u)$$

where

$$G(m,u) = \nabla_m(A(m)u).$$

Thus to compute the sensitivities one requires to compute the inverse of the forward problem matrix times either G or $Q^\top$. This may make the computation of the sensitivities difficult if not impossible. We will discuss this in the next.

3.4 Sensitivity computation for time dependent problems

Maybe the most involved computation of sensitivities arises in time dependent problems. We recall the wave equation which we rewrite

$$\begin{pmatrix} \Delta t^{-1}\kappa & 0 \\ -\nabla_h & \Delta t^{-1}\rho \end{pmatrix} \begin{pmatrix} p \\ u \end{pmatrix}^{n+1} - \begin{pmatrix} \Delta t^{-1}\kappa - \alpha & -\nabla_h^\top \\ 0 & \Delta t^{-1}\rho - \beta \end{pmatrix} \begin{pmatrix} p \\ u \end{pmatrix}^{n} = 0 \quad (3.1)$$

Consider now a vector $y = [p^1, u^1, \ldots, p^n, u^n$. The forward problem can be written as

$$A(\kappa)y = \begin{pmatrix} A_1(\kappa) & & & & & \\ A_2(\kappa) & A_1(\kappa) & & & & \\ & A_2(\kappa) & A_1(\kappa) & & & \\ & & \ddots & \ddots & & \\ & & & & A_2(\kappa) & A_1(\kappa) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \\ y_n \end{pmatrix} = \begin{pmatrix} -A_2(\kappa)y_0 \\ 0 \\ \vdots \\ \\ 0 \end{pmatrix} = q \quad (3.2)$$

where

$$A_1(\kappa) = \begin{pmatrix} \Delta t^{-1}\kappa & 0 \\ -\nabla_h & \Delta t^{-1}\rho \end{pmatrix} \quad \text{and} \quad A_2(\kappa) = \begin{pmatrix} \Delta t^{-1}\kappa - \alpha & -\nabla_h^\top \\ 0 & \Delta t^{-1}\rho - \beta \end{pmatrix}$$

Notice that in order to "solve" $A(\kappa)y = q$ we invert the matrix by starting from the first discretized time and going forward in time. Another important observation is that the matrix $A^\top(\kappa)$ has the form

$$A^\top(\kappa) = \begin{pmatrix} A_1(\kappa)^\top & A_2(\kappa)^\top & & & & \\ & A_1(\kappa)^\top & A_2(\kappa)^\top & & & \\ & & & \ddots & \ddots & \\ & & & & A_1(\kappa)^\top & A_2(\kappa)^\top \\ & & & & & A_1(\kappa)^\top \end{pmatrix}$$

This implies that the solution of the adjoint system $A^\top(\kappa)y = z$ is done by starting from the last time step and going backward in time. Solving this system is sometimes refer to as reverse time migration. We discuss this in the chapter.

3.5 Computation of the Jacobians

The The examples above are rather generic in nature. For the constraint $c(m,u) = 0$, the Jacobian $\nabla_u c$ is calculated in order to solve the forward problem, and thus is usually available. On the other hand, the Jacobain $\nabla_m c$ is not required for the solution of the forward problem and therefore need to be evaluated whenever an inverse problem is to be solved. In some cases, automatic differentiation can be used but otherwise, this can be complicated to compute and an intimate understanding of the forward modeling code is needed.

There are some cases, that require special consideration. Consider for example, a simple advection problem

$$\epsilon u_{xx} + m u_x = 0$$

with some appropriate boundary conditions. This is a linear PDE in u and m. Now, consider the upwind discretization of the problem that reads

$$\frac{\epsilon}{h^2}(u_{j+1} - 2u_j + u_{j-1}) + \frac{1}{h}\left(\max(m_j, 0)(u_j - u_{j-1}) + \min(m_j, 0)(u_{j+1} - u_j)\right) = 0$$

Clearly, the discretization is non-differentiable with respect to m and thus one can expect some difficulties. In these cases, our convention of discretize first then optimize is questionable. If we still wish to work using this strategy then It is possible to introduce some smoothing to the max and min functions. Nonetheless, this simple example demonstrates the pitfalls in our approach so far. It also highlights the intimate understanding of the discretization of the forward problem.

For finite volume discretization, the computation of the derivatives can be done by differentiating matrix-vector product. It is important to note that for finite element computation the Jacobians can be calculated by looking at the element matrices. For example, for the DC resistivity problem we had that the stiffness matrix can be written as

$$A(m)u = \sum_e m_e A_e u_e$$

where A_e are local element matrices, m_e is the physical property of that element and $u_e = P_e u$ is a vector that contain the local degrees of freedom over the element. It is therefore clear that

$$\frac{\partial}{\partial m_j}(A(m)u) = \frac{\partial}{\partial m_j}\left(\sum_e m_e A_e u_e\right) = A_j u_j$$

Since A_j does not depend on m but rather on the mesh alone, it makes sense to save all the element matrtices if the sensitivity computation is required many times in the course of the solution of the problem.

3.6 Working with sensitivities in practice

3.6.1 Computing the sensitivities

The sensitivity matrix is large and composed of three matrices.

$$J(m,u) = Q\left(\nabla_u c\right)^{-1}\nabla_m c.$$

The matrices $Q, \nabla_m c$ and $\nabla_u c$ are typically sparse however, the matrix $(\nabla_u c)^{-1})$ is almost always dense. Therefore, the sensitivity matrix is typically dense and very large. Computing the matrix in practice is therefore impossible for most large scale problems. Nonetheless, **the actual computation of the sensitivity is not needed in most practical cases**. What is needed is a matrix-vector products of the form Jv and $J^\top w$. For that note that one can compute the product in three steps. For the forward problem we multiply the vector v by $\nabla_m c$. In the second step we solve the linear system $(\nabla_u c)y = \nabla_m c\, v$. In the last step we set Jv to be the product $-Qy$. For the transpose we start by computing $Q^\top w$ we then solve the system $(\nabla_u c)^\top y = Q^\top w$ and finally set the product $J^\top w$ to $-(\nabla_m c)^\top y$.

Note that the calculation of the sensitivity matrix vector product and its adjoint requires solving a system equivalent to a linearized forward or transposed problem. This is not a trivial task. If the calculation is required many times then investing in good preconditioners can be crucial. Assume for a moment that we are able to compute the LU factorization of the forward problem. In this case the cost of each matrix vector product is the cost of forward-back-substitiotion. If a single product is needed then the cost of the factorization dominates the computation. However, if the number of sensitivity matrix vector product is large, one may want to reconsider and compute either an exact or an approximate factorization if possible. We will discuss this detail in the next chapters.

Although computing sensitivities is almost never done in practice there are some rare cases where for one reason or another the sensitivity matrix is desirable. This is typically the case where either the size of the model or data is very small. It is important to remember that there are two ways to compute the matrix. Since $J = -Q(\nabla_u c)^{-1}\nabla_u c$ one can compute it is

$$J = -Q((\nabla_u c)^{-1}\nabla_u c)$$

That is compute the matrix $(\nabla_u c)^{-1}\nabla_u c$ and then multiply by Q or that we compute

$$J = -(\nabla_m c^\top(\nabla_u c^{-\top} Q^\top))^\top$$

That is, compute the product $(\nabla_u c)^{-\top} Q^\top$ multiply by $\nabla_m c^\top$ and then transpose the result. The first approach is sometimes referred to as the forward calculation and the second approach is referred to as the backward calculation. If the number of parameters m is very small then the forward calculation is obviously preferable and on the other hand if the number of data is small then the backward evaluation is better.

3.6.2 Dimensionality reduction using Lanczos bidiagonalization

In the course of the solution of parameter estimation problems one often requires to solve systems that involve the sensitivity matrix J and its adjoint $J^\top$. We have seen that it is possible to compute the sensitivity matrix-vector product by solving the forward problem. It is possible to use these products to compute an approximate decomposition of the sensitivity matrix. Such decomposition is used in methods such as Least Squares QR (LSQR) for the solution of systems of the from $Jz = b$. Such methods can also be used in order to approximate the sensitivity matrix for other reasons that are discussed later in the notes.

The basic idea is to use a Krylov space of the form

$$\mathcal{K} = \{J^\top b, (J^\top J)J^\top b, \ldots, (J^\top J)^k J^\top b\}$$

in order to approximate J by the decomposition

$$J \approx U_k B_k V_k^\top$$

where $U_k = [u_1, \ldots, u_k]$, $V_k = [v_1, \ldots, v_k]$ are matrices of orthogonal vectors and B_k is a bidiagonal matrix of the form

$$B_k = \begin{pmatrix} b_{11} & & & & \\ b_{21} & b_{22} & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & b_{k-1,k} & b_{k,k} \end{pmatrix}$$

If this decomposition reminds the reader the SVD it is not accidental. Indeed, the singular values of B_k approximate the singular values of J. Although it is difficult to prove the exact connection between the SVD and the vectors obtained by the Lanczos process it is well documented that the Lanczos vectors approximate the singular vectors of J. The Lanczos process can be summarized as follows:

```
function [U,B,V] = lancBiDiag(J,d,k,L,tol)
%LANCBIDIAG Lanczos bidiagonalization.

[m,n] = size(J);  [n,l] = size(L);
U = zeros(m,k); V = zeros(n,k);

% Prepare for Lanczos iteration.
v = zeros(n,1);
beta = norm(d); u = d/beta;
U(:,1) = u;

Lpt = @(x) ( L*((L'*L)\x)); Lp  = @(z) ( (L'*L)\(L'*z));

for i=1:k
  r = Lpt((J'*u)) - beta*v;
  alpha = norm(r);
  v = r/alpha;
  B(i,2) = alpha;
  V(:,i) = v;
  p = J*(Lp(v)) - alpha*u;
  beta = norm(p); u = p/beta;
  B(i,1) = beta;
  U(:,i+1) = u;
end
B = spdiags(B,[-1,0],k+1,k);
```

We introduced a "preconditioning matrix" L into the decomposition. Its role will be clear in the next chapter. Using the decomposition it is possible to work with the approximation of the sensitivities.

3.6.3 Dimensionality reduction using stochastic approximation

While Lancszos bidiagonalization is perhaps the most efficient algorithm to compute a small number of singular values and vectors, an alternative has been proposed recently [**?**]. It has advantage when parallel computing is available. While Lanczos method is iterative by nature, the stochastic approximation is parallel by nature.

The basic idea is as follow. Let J be an $n \times k$ matrix. To approximate p singular values and vectors we choose a random $k \times p + \ell$ matrix Ω, where ℓ is small (say 3). We the compute the product

$$Y = J\Omega$$

The matrix Y is "long and skinny" and thus we can easily compute its QR factorization. Let

$$Y = \widehat{Q}\widehat{R}.$$

We set the approximate J to

$$\widehat{J} = \widehat{Q}\widehat{Q}^\top J$$

which implies that it has the same singular values as $\widehat{Q}^\top J$. Thus, it is possible to work with $\widehat{J}$ rather than with J if needed.

3.7 Differentiating linear algebra expressions

Throughout this course we will differentiate expressions that involve matrices and vectors. It is useful to be able to differentiate such expressions quickly.

To start, we recall the definition of the Jacobian. If $f(x)$ is a function from $\mathbb{R}^n$ to $\mathbb{R}^k$ then the Jacobian of f is a $k \times n$ matrix with entries

$$[J(x)]_{ij} = [\nabla_x f(x)]_{ij} = \frac{\partial f_i(x)}{\partial x_j}$$

Note that the dimensions of the Jacoban are $k \times n$.

We now differentiate some simple quantities and discuss some confusing conventions. We start with the linear multivariable function

$$f(x) = y^\top x = x^\top y = \sum_i x_i y_i$$

Differentiating with respect to x we obtain the vector

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_M} \end{pmatrix} = \begin{pmatrix} y_1 & \cdots & y_M \end{pmatrix} = y^\top$$

Note that the derivative of a function $f(x)$ should be written as a **row vector** if we assume that x is a column vector. However, the convention is to store the gradient

of a scalar function as a **column vector**. That is we write $\nabla(x^\top y) = y$ and not $y^\top$. This can create some confusion so bare in mind the the convention is that the transpose is used for **1D only**. This can be highly confusing at times and we try to point to this inconsistency when possible.

We now use the result to differentiate the multivariable function

$$f(x) = Ax = \begin{pmatrix} a_1^\top x \\ \vdots \\ a_N^\top x \end{pmatrix}$$

where $a_i^\top = A_{i,:}$. Note that in this case f is a vector $f = \begin{pmatrix} f_1 & \dots & f_M \end{pmatrix}$. Each entry in the vector f is similar to the one we just differentiated thus

$$\nabla f = \begin{pmatrix} \nabla f_1 \\ \vdots \\ \nabla f_M \end{pmatrix} = \begin{pmatrix} a_1^\top \\ \vdots \\ a_M^\top \end{pmatrix} = A$$

Using this result we can now differentiate a quadratic form. Let

$$f(x) = x^\top Ax$$

To differentiate we use the product rule

$$\frac{\partial f}{\partial x} = \frac{\partial x^\top Ax}{\partial x} = \left(x^\top \frac{\partial Ax}{\partial x}\right)^\top + \frac{\partial x^\top A}{\partial x} x = Ax + A^\top x$$

If A is symmetric then we obtain $2Ax$.

We now look at another type of derivatives generated by the Hadamard product. The Hadamard product of two vectors x and y is defined as

$$f = x \odot y = \begin{pmatrix} x_1 y_1 \\ \vdots \\ x_M y_M \end{pmatrix}$$

It is easy to verify that

$$f = \operatorname{diag}(y)x$$

where $\operatorname{diag}(y)$ is a diagonal matrix with y on its main diagonal. Therefore

$$\nabla f = \operatorname{diag}(y)$$

Using the above we can now differentiate more complicated expressions. Consider the expression

$$f(x) = (Ax)^2 = (Ax) \odot (Ax)$$

Differentiating with respect to x we obtain

$$\nabla f = \nabla((Ax) \odot (Ax)) = \nabla(\operatorname{diag}(Ax)Ax) = 2\operatorname{diag}(Ax)A.$$

Lets make this slightly more complicated. Consider now the expression

$$f(x) = v^\top \sqrt{(Ax)^2 + 1}$$

Similar expression arise in the minimal surface problem. Since v is independent of x, to differentiate we need to differentiate $\sqrt{(Ax)^2 + 1}$. Using the chain rule we have

$$\nabla(\sqrt{(Ax)^2 + 1}) = \operatorname{diag}\left(\frac{1}{2\sqrt{(Ax)^2 + 1}}\right) \nabla((Ax)^2+1) = \operatorname{diag}\left(\frac{1}{\sqrt{(Ax)^2 + 1}}\right) \operatorname{diag}(Ax)A$$

Putting this together we have

$$v^\top \operatorname{diag}\left(\frac{1}{\sqrt{(Ax)^2 + 1}}\right) \operatorname{diag}(Ax)A$$

note that the above expression is a row vector. If we use column vectors we have

$$\nabla f = (v^\top \operatorname{diag}\left(\frac{1}{\sqrt{(Ax)^2 + 1}}\right) \operatorname{diag}(Ax)A)^\top = A^\top \operatorname{diag}\left(\frac{v}{\sqrt{(Ax)^2 + 1}}\right) Ax$$

3.8 Programer note

The calculation of the sensitivities (or sensitivities matrix-vector) is **crucial** for the solution of the inverse problem. It therefore very important to verify that the correct sensitivity is computed. This verification can be done by the derivative test we now present. Assume that we have $c(m, u) = 0$ and that we computed the Jacobians $\nabla_u c$ and $\nabla_m c$. Then, to test the Jacobians we take a random vector v and generate the table of $\|c(u+hv, m)-c(u, v)\|$ vs h where h is decreasing logarithmically and a table of $\|c(u + hv, m) - c(m, u) - h\nabla_u cv\|$ vs h. By Taylor's theorem the first difference converge to 0 linearly while the second difference converge to 0 quadratically. If you do not get quadratic convergence you have the wrong Jacobian! A similar calculation should be done with the Jacobian with respect to m, $\nabla_m c$.

To demonstrate, we consider the forward problem of the form

$$c(m, u) = A(m)u - q = D^\top \operatorname{diag}(S \exp(m))Du - q$$

Computing the derivative with respect to m we obtain that

$$\nabla_m c = D^\top \operatorname{diag}(Du)S \operatorname{diag}(\exp(m)).$$

The following code test the derivative

```
% define function and derivatives
C           = @(u,m)(D'*sdiag(S*exp(m))*D*u - q);
dCdm   = @(u,m)(D'*sdiag(D*u)*S*sdiag(exp(m)));
dCdu    =  @(u,m)(D'*sdiag(S*exp(m))*D);

% now test the derivatives
u = randn(size(D,2),1); m = randn(size(S,2),1);
v = randn(size(m));
f = C(u,m);
G = dCdm(u,m);
for i=1:10
         h = 10^(-i);
        fp = C(u,m+h*v);
        diff1 = norm(fp-f);
        diff2 = norm(fp-f - h*G*v);
        fprintf('%3.2e    %3.2e    %3.2e\n',h,diff1,diff2)
end
```

The following table was obtained using the above code

1.00e-01	7.58e+02	7.01e+01
1.00e-02	7.61e+01	7.04e-01
1.00e-03	7.62e+00	7.04e-03
1.00e-04	7.62e-01	7.04e-05
1.00e-05	7.62e-02	7.04e-07
1.00e-06	7.62e-03	7.04e-09
1.00e-07	7.62e-04	7.05e-11
1.00e-08	7.62e-05	8.49e-12
1.00e-09	7.62e-06	9.54e-12
1.00e-10	7.62e-07	8.87e-12

Obviously, as long as h is not too small, where machine precision takes over, we see that reducing h in a factor of 10 reduces the linear approximation in a factor of 100.

Although the derivative test is meant to test the derivatives it can also be used to learn something about the function at hand. For example, consider the nonlinear PDE

$$\nabla \cdot \rho(u)\nabla u = m$$

with $\rho(u) = \frac{1}{\sqrt{|Du|^2+\eta}}$

In this case a simple 1D discretization reads

$$c(m,u) = D^\top \operatorname{diag}\left(\frac{1}{\sqrt{|Du|^2+\eta}}\right) Du - m$$

and the derivative with respect to u can be computed as

$$\nabla_u c = D^\top \operatorname{diag}\left(\frac{1}{\sqrt{|Du|^2+\eta}}\right) D - D^\top \operatorname{diag}\left(\frac{(Du)^2}{(|Du|^2+\eta)^{\frac{3}{2}}}\right) D$$

The following matlab script test this approximation

```
t = 1e-8;
% define function and derivatives
C   = @(u,m)(D'*sdiag(1./sqrt((D*u).^2 + t))*D*u - m);
dCdu = @(u,m)(D'*sdiag(1./sqrt((D*u).^2 + t))*D - ...
                    D'*sdiag((D*u).^2./((D*u).^2 + t).^(3/2))*D);

% now test the derivatives
u = randn(size(D,2),1); m = randn(size(D,2),1);
v = randn(size(u));
f = C(u,m);
A = dCdu(u,m);
for i=1:10
        h = 10^(-i);
       fp = C(u+h*v,m);
       diff1 = norm(fp-f);
       diff2 = norm(fp-f - h*A*v);
       fprintf('%3.2e    %3.2e    %3.2e\n',h,diff1,diff2)
end
```

and the results are presented in the table below

1.00e-01	3.62e+01	3.62e+01
1.00e-02	1.58e+01	1.58e+01
1.00e-03	1.37e-04	8.51e-05
1.00e-04	5.55e-06	3.80e-07
1.00e-05	5.20e-07	3.59e-09
1.00e-06	5.17e-08	3.51e-11
1.00e-07	5.17e-09	1.57e-12
1.00e-08	5.17e-10	1.45e-12
1.00e-09	5.17e-11	1.27e-12
1.00e-10	5.17e-12	1.20e-12

This table is rather different than the one we have seen previously. Note that for a step size of $h = 10^{-2}$ no marked difference is observed between the first and second order approximations. This implies that at least for the direction chosen here, the problem is nonlinear and requires very small steps for the linear approximation to be a good representative of the nonlinear problem. Thus, using the derivative test we can learn not only about the correctness of the derivative but also about the nonlinearity of the problem at hand.

3.9 Problems for chapter 3

Assume again the problem

$$\nabla \cdot \sigma \nabla u = q \qquad \sigma > 0$$

with Derichlet boundary conditions. Assume that a cell-centered discretization is used for the problem as described in Chapter 2. Then, we have seen that the discrete system can be written as

$$\mathrm{DIV}\left(\mathrm{diag}(A_{cf}\sigma^{-1})\right)^{-1}\mathrm{DIV}^{\top}u = q - \mathrm{DIV}\left(\mathrm{diag}(A_{cf}\sigma^{-1})\right)^{-1}B_c\,u_{\mathrm{BC}}$$

1. Program the forward problem on an $n_1 \times n_2 \times n_3$ uniform grid.
2. Compute the derivatives with respect to u, u_{BC} and σ.

3. Program the derivatives and validate them using the derivative test.

4. For a problem of size 8^3 and $\sigma = 1$ compute the sensitivity matrix assuming $Q = I$.

5. Assuming $u_{\mathrm{BC}} = 0$, compute the SVD decomposition of the sensitivity matrix and plot the right singular vectors that correspond to $\lambda_1, \ldots, \lambda_{10}$ and view the singular vectors that correspond to smaller singular values. Make qualitative comments about the behavior of the singular vectors.

6. In a semilogy curve plot the singular values of the problem. Can you comment about the effective rank of the sensitivities?