Numerical optimization and adjoint state methods for large-scale nonlinear least-squares problems

Ludovic Métivier\(^1\) and the SEISCOPE group\(^{1,2,3}\)

\(^1\) LJK, Univ. Grenoble Alpes, CNRS, Grenoble, France
\(^2\) ISTerre, Univ. Grenoble Alpes, CNRS, Grenoble, France
\(^3\) Géoazur, Univ. Nice Sophia Antipolis, CNRS, Valbonne, France

http://seiscope2.osug.fr

Joint Inversion Summer School Barcelonnette, France, 15-19, June, 2015
1. Numerical optimization methods for large-scale smooth unconstrained minimization problems

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3. Summary
Outline

1. Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
   - Steepest descent method
   - Newton method
   - Quasi-Newton methods
   - What about the nonlinear conjugate gradient?
   - Summary

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3. Summary
Outline

1. Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
     - Steepest descent method
     - Newton method
     - Quasi-Newton methods
     - What about the nonlinear conjugate gradient?
   - Summary

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3. Summary
Nonlinear least-squares problem

In this presentation, we will consider the inverse problem

\[ \min_m f(m) = \frac{1}{2} \| d_{\text{cal}}(m) - d_{\text{obs}} \|^2 \]

where

- \( d_{\text{obs}} \) are data associated with a physical phenomenon and a measurement protocol: seismic waves, electromagnetic waves, gravimeter, ultrasounds, x-ray, ...

- \( m \) is the parameter of interest we want to reconstruct: \( P \) and \( S \)-wave velocities, density, anisotropy parameters, attenuation, or a collection of these parameters

- \( d_{\text{cal}}(m) \) are synthetic data, computed numerically, often through the solution of partial differential equations

- \( f(m) \) is a misfit function which measures the discrepancy between observed and synthetic data
Nonlinear least-squares problem

In this presentation, we will consider the inverse problem

\[ \min_m f(m) = \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2 \]

Of course, in joint inversion, we may consider a misfit function as a sum of these functions associated with different measurements: the theory remains the same.
Nonlinear least-squares problem

In this presentation, we will consider the inverse problem

$$\min_m f(m) = \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2$$

- We will also assume that $f(m)$ is a continuous and twice differentiable function: the gradient is continuous, and the second-order derivatives matrix $H(m)$ (Hessian matrix) is also continuous.
- The methods we are going to review are local optimization method: we put aside the global optimization methods and stochastic/genetic algorithms which are unaffordable for large-scale optimization problems.
- All the methods we review are presented in (Nocedal and Wright, 2006).
Local methods to find the minimum of a function

Necessary condition

To detect the extremum of a differentiable function $f(m)$, we have the necessary condition

$$\nabla f(m) = 0$$
Local methods to find the minimum of a function

**Necessary condition**

To detect the extremum of a differentiable function $f(m)$, we have the necessary condition

$$\nabla f(m) = 0$$

This is not enough: is it a minimum or maximum?
Local methods to find the minimum of a function

Necessary and sufficient conditions

In a local minimum, the function is locally convex: the Hessian is definite positive

$$\nabla f(m) = 0, \quad \nabla^2 f(m) > 0$$
Practical implementation

However, this not what we implement in practice. From an initial guess $m_0$, a sequence $m_k$ is built such that

- the limit $m_*$ should satisfy the necessary condition
  \[ \nabla f(m_*) = 0 \]
- at each iteration
  \[ f(m_{k+1}) < f(m_k) \]

We have to guarantee the decrease of the misfit function at each iteration.
Outline

1. Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
   - Steepest descent method
   - Newton method
   - Quasi-Newton methods
   - What about the nonlinear conjugate gradient?
   - Summary

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3. Summary
### How to find the zero of the gradient: first-order method

#### The fixed-point method

We want to find $m_*$ such that

$$\nabla f(m_*) = 0 \quad \text{(1)}$$

The simplest method is to apply the fixed point iteration on $I - \alpha \nabla f$

$$m_{k+1} = (I - \alpha \nabla f) m_k = m_k - \alpha \nabla f(m_k), \quad \alpha \in \mathbb{R}_+^*$$

At convergence we should have

$$m_* = (I - \alpha \nabla f) m_* = m_* - \alpha \nabla f(m_*) \quad \Rightarrow \quad \nabla f(m_*) = 0$$
How to find the zero of the gradient: first-order method

Ensuring the decrease of the misfit function

We need to ensure

\[ f(m_{k+1}) < f(m_k) \]

We have

\[ f(m + dm) = f(m) + \nabla f(m)^T dm + o(\|dm\|^2) \]

Therefore, if

\[ m_{k+1} = m_k - \alpha \nabla f(m_k), \]

we have

\[ f(m_{k+1}) = f(m_k - \alpha \nabla f(m_k)) = f(m_k) - \alpha \nabla f(m_k)^T \nabla f(m_k) + \alpha^2 o(\|\nabla f(m_k)\|^2) \]

that is

\[ f(m_{k+1}) = f(m_k) - \alpha \|\nabla f(m_k)^T\|^2 + \alpha^2 o(\|\nabla f(m_k)\|^2) \]

Therefore for \( \alpha \) small enough, we can ensure the decrease condition.
How to find the zero of the gradient: first-order method

Fixed point on $I - \alpha F = $ steepest-descent method

To summarize, using the fixed-point iteration on $I - \alpha \nabla f(m)$ yields the sequence

$$m_{k+1} = m_k - \alpha \nabla f(m_k),$$

We have just rediscovered the steepest-descent iteration
Outline

1 Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
   - Steepest descent method
   - Newton method
   - Quasi-Newton methods
   - What about the nonlinear conjugate gradient?
   - Summary

2 First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3 Summary
How to find the zero of the gradient: second-order method

Newton method: graphical interpretation

A faster (quadratic) convergence can be achieved for finding the zero $\nabla f(m)$ if we use the Newton method.

continue till $P$ and $P_1$'s becomes sufficiently closer.
Newton method

We approximate $\nabla f(m_{k+1})$ as its first-order Taylor development $m_k$

$$\nabla f(m_{k+1}) \simeq \nabla f(m_k) + \left( \frac{\partial \nabla f(m_k)}{\partial m_k} \right) (m_{k+1} - m_k) \quad (1)$$

We look for the zero of this approximation

$$\nabla f(m_k) + \left( \frac{\partial \nabla f(m_k)}{\partial m_k} \right) (m_{k+1} - m_k) = 0 \quad (2)$$

which yields

$$m_{k+1} = m_k - \left( \frac{\partial \nabla f(m_k)}{\partial m_k} \right)^{-1} \nabla f(m_k)$$
Notations

In the following, we use the notation

$$\frac{\partial \nabla f(m_k)}{\partial m_k} = H(m_k)$$

for the Hessian operator (second-order derivatives of the misfit function).
Decrease of the misfit function

Do we ensure the decrease of the misfit function?

\[
\begin{align*}
    f(m_{k+1}) & = f(m_k - \alpha_k H(m_k)^{-1} \nabla f(m_k)) \\
    & = f(m_k) - \alpha_k \nabla f(m_k)^T H(m_k)^{-1} \nabla f(m_k) + \alpha_k^2 o(||H(m_k)^{-1} \nabla f(m_k)||^2)
\end{align*}
\]

We have

\[
\nabla f(m_k)^T H(m_k)^{-1} \nabla f(m_k) > 0 \tag{1}
\]

if and only if \( H(m_k)^{-1} > 0 \).
How to find the zero of the gradient: second-order method

**Difficulties**

- The Hessian operator may not be necessary strictly positive: the function $f(m)$ may not be strictly convex as the forward problem is nonlinear ($f(m)$ is not quadratic)
- For large-scale application, how to compute $H(m)$ and its inverse $H(m^{-1})$?
1. Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
   - Steepest descent method
   - Newton method
   - Quasi-Newton methods
   - What about the nonlinear conjugate gradient?
   - Summary

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3. Summary
The \textit{l-BFGS} method

\textbf{Principle}

\textit{l-BFGS} method (Nocedal, 1980) relies on the iterative scheme

\begin{equation}
  m_{k+1} = m_k - \alpha_k Q_k \nabla f(m_k)
\end{equation}

where

\begin{equation}
  Q_k \simeq H(m_k)^{-1}, \text{sym} > 0
\end{equation}

and

\begin{equation}
  \alpha_k \in \mathbb{R}^*_+
\end{equation}

is a scalar parameter computed through a linesearch process.
The $l$-BFGS method

$l$-BFGS approximation

The $l$-BFGS approximation consists in defining $Q_k$ as

$$Q_k = (V_{k-1}^T \cdots V_{k-l}^T) Q_k^0 (V_{k-1} \cdots V_{k-1}) + \rho_{k-l} (V_{k-1}^T \cdots V_{k-l+1}^T) s_{k-l} s_{k-l}^T (V_{k-l+1} \cdots V_{k-1}) + \rho_{k-l+1} (V_{k-1}^T \cdots V_{k-l+2}^T) s_{k-l+1} s_{k-l+1}^T (V_{k-l+2} \cdots V_{k-1}) + \cdots + \rho_{k-1} s_{k-1} s_{k-1}^T, \quad (1)$$

where the pairs $s_k, y_k$ are

$$s_k = m_{k+1} - m_k, \quad y_k = \nabla f(m_{k+1}) - \nabla f(m_k), \quad (2)$$

the scalar $\rho_k$ are

$$\rho_k = \frac{1}{y_k^T s_k}, \quad (3)$$

and the matrices $V_k$ are defined by

$$V_k = I - \rho_k y_k s_k^T. \quad (4)$$
The l-BFGS method

Implementation: two-loops recursion

Data: $\rho_i, s_i, y_i, \ i = k - l, \ldots, k - 1, H_k^0, \nabla f(x_k)$

Result: $\Delta x_k = -Q_k \nabla f(x_k)$

$q = -\nabla f(x_k)$;

for $i = k - 1, \ldots, k - l$ do
  $\alpha_i = \rho_i s_i^T \Delta x_k$;
  $q = q - \alpha_i y_i$;
end

$\Delta x_k = H_k^0 q$;

for $i = k - l, \ldots, k - 1$ do
  $\beta = \rho_i y_i^T \Delta x_k$;
  $\Delta x_k = \Delta x_k + (\alpha_i - \beta_i) s_i$;
end
Truncated Newton method

Principle

The truncated Newton method (Nash, 2000) relies on the iterative scheme

\[ m_{k+1} = m_k + \alpha_k \Delta m_k \]  \hspace{1cm} (1)

where \( \Delta m_k \) is computed as an approximate solution of the linear system

\[ H(m_k) \Delta m_k = -\nabla f(m_k) \]  \hspace{1cm} (2)
Truncated Newton method

**Principle**

The truncated Newton method (Nash, 2000) relies on the iterative scheme

$$m_{k+1} = m_k + \alpha_k \Delta m_k \quad (1)$$

where $\Delta m_k$ is computed as an approximate solution of the linear system

$$H(m_k)\Delta m_k = -\nabla f(m_k) \quad (2)$$

**Implementation**

- A matrix-free conjugate gradient is used to solve this linear system (Saad, 2003)
- This only requires the capability to compute matrix-vector products $H(m_k)v$ for given vectors $v$: the full Hessian matrix needs not to be formed explicitly
- The resulting approximation of the Hessian only accounts for positive eigenvalues of $H(m_k)$: $\Delta m_k$ is ensured to be a descent direction
Outline

1 Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
   - Steepest descent method
   - Newton method
   - Quasi-Newton methods
   - What about the nonlinear conjugate gradient?
   - Summary

2 First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3 Summary
Conjugate gradient for symmetric positive linear systems

The conjugate gradient is an iterative method for the solution of symmetric positive definite linear systems

\[ Am = b \] (3)

The method enjoys several interesting properties

- Convergence in at most \( n \) iterations for a system of size \( n \) (ok)
- Fast convergence rate possible depending on the eigenvalues distribution of \( A \): in practice, an acceptable approximation of the solution can be obtained in \( k \) iterations with \( k << n \)
Conjugate gradient

Only matrix-vector products to perform

**Implementation**

**Data:** $m_0, \varepsilon$

**Result:** $A^{-1}b$

$r_0 = Am_0 - b$

$\Delta m_0 = -r_0$

$k = 0$

**while** $\|r_k\| > \varepsilon$ **do**

- Compute $A\Delta m_k$
- $\alpha_k = \frac{r_k^T r_k}{\Delta m_k^T A \Delta m_k}$
- $m_{k+1} = m_k + \alpha_k \Delta m_k$
- $r_{k+1} = r_k + \alpha_k A \Delta m_k$
- $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
- $\Delta m_{k+1} = -r_{k+1} + \beta_{k+1} \Delta m_k$
- $k = k + 1$

**end**

**Algorithm 1:** Conjugate gradient for linear symmetric positive definite systems
Nonlinear conjugate gradient

How to extend the conjugate gradient for the solution of nonlinear minimization problems? There is a link: solving

$$Am = b$$  \hspace{1cm} (3)

where $A$ is symmetric positive definite is equivalent to solve

$$\min_{m} f(m) = m^T Am - m^T b$$  \hspace{1cm} (4)

because

$$\nabla f(m) = Am - b$$  \hspace{1cm} (5)

and $f$ is strictly convex (a single extremum which is a minimum)
Implementation

Simply replace $r$ in the preceding algorithm by $\nabla f(m)$

**Data:** $m_0, \varepsilon$

**Result:** $\min_m f(m)$

$\nabla f_0 = \nabla f(m_0)$;

$\Delta m_0 = -\nabla f_0$;

$k = 0$;

while $||\nabla f_k|| > \varepsilon$ do

Compute $\alpha_k$ through linesearch;

$m_{k+1} = m_k + \alpha_k \Delta m_k$;

$\nabla f_{k+1} = \nabla f(m_{k+1})$;

$\beta_{k+1} = \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k}$;

$\Delta m_{k+1} = -\nabla f_{k+1} + \beta_{k+1} \Delta m_k$;

$k = k + 1$;

end

Algorithm 2: Nonlinear conjugate gradient
1 Numerical optimization methods for large-scale smooth unconstrained minimization problems
   - Numerical optimization for nonlinear least-squares problems
   - Steepest descent method
   - Newton method
   - Quasi-Newton methods
   - What about the nonlinear conjugate gradient?
   - Summary

2 First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3 Summary
An iterative scheme for local optimization

We have seen 4 different methods all based on the same iterative scheme

\[ m_{k+1} = m_k + \alpha_k \Delta m_k \] (3)
An iterative scheme for local optimization

We have seen 4 different methods all based on the same iterative scheme

\[ m_{k+1} = m_k + \alpha_k \Delta m_k \]  \hspace{1cm} (3)

Nonlinear optimization methods

The four method only differ in the way to compute \( \Delta m_k \)

<table>
<thead>
<tr>
<th>Method</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steepest descent</td>
<td>( \Delta m_k = -\nabla f(m_k) )</td>
</tr>
<tr>
<td>Nonlinear CG</td>
<td>( \Delta m_k = -\nabla f(m_k) + \beta_k \Delta m_{k-1} )</td>
</tr>
<tr>
<td>l-BFGS</td>
<td>( \Delta m_k = -Q_k \nabla f(m_k), \quad Q_k \simeq H_k^{-1} )</td>
</tr>
<tr>
<td>Truncated Newton</td>
<td>( H(m_k)\Delta m_k = -\nabla f(m_k) ) (solved with CG)</td>
</tr>
</tbody>
</table>
Large-scale applications

From this quick overview we see that the two key quantities to be estimated for the solution of

$$\min_{m} f(m) = \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2$$  \hspace{1cm} (3)

are

- The gradient of the misfit function $\nabla f(m)$
- Hessian vector products $H(m)v$ for a given $v$ (only for the truncated Newton method)

We shall see in the next part how to compute it at a reasonable computational cost (memory imprint and flops) for large-scale applications using adjoint state methods.
Outline

1 Numerical optimization methods for large-scale smooth unconstrained minimization problems

2 First-order and second-order adjoint state methods for gradient and Hessian-vector products computation
   - Gradient computation of a nonlinear least-squares function
   - First-order adjoint state method
   - Second-order adjoint state method

3 Summary
Outline

1 Numerical optimization methods for large-scale smooth unconstrained minimization problems

2 First-order and second-order adjoint state methods for gradient and Hessian-vector products computation
   - Gradient computation of a nonlinear least-squares function
   - First-order adjoint state method
   - Second-order adjoint state method

3 Summary
Gradient computation of a nonlinear least-squares function

Framework

We consider the problem

$$\min_m f(m) = \frac{1}{2} \|d_{cal}(m) - d_{obs}\|^2$$
Gradient computation of a nonlinear least-squares function

For a perturbation $dm$ we have

$$f(m + dm) = \frac{1}{2} \|d_{cal}(m + dm) - d_{obs}\|^2$$

$$= \frac{1}{2} \|d_{cal}(m) - d_{obs} + J(m)dm + o(\|dm\|^2)\|^2$$

where

$$J(m) = \frac{\partial d_{cal}}{\partial m}$$

is the Jacobian matrix
Gradient computation of a nonlinear least-squares function

For a perturbation $dm$ we have

$$f(m + dm) = \frac{1}{2} \| d_{cal}(m + dm) - d_{obs} \|^2$$

$$+ \frac{1}{2} \| d_{cal}(m) - d_{obs} + J(m)dm + o(\| dm \|^2) \|^2$$

where

$$J(m) = \frac{\partial d_{cal}}{\partial m}$$

is the Jacobian matrix

$$f(m + dm) = \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2 + (d_{cal} - d_{obs}, J(m)dm) + o(\| dm \|^2)$$

$$+ \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2 + \left( J(m)^T (d_{cal} - d_{obs}) , dm \right) + o(\| dm \|^2)$$
Gradient computation of a nonlinear least-squares function

For a perturbation $dm$ we have

$$f(m + dm) = \frac{1}{2}\|d_{cal}(m + dm) - d_{obs}\|^2$$

$$= \frac{1}{2}\|d_{cal}(m) - d_{obs} + J(m)dm + o(\|dm\|^2)\|^2$$

where

$$J(m) = \frac{\partial d_{cal}}{\partial m}$$

is the Jacobian matrix

$$f(m + dm) = \frac{1}{2}\|d_{cal}(m) - d_{obs}\|^2 + (d_{cal} - d_{obs}, J(m)dm) + o(\|dm\|^2)$$

$$= \frac{1}{2}\|d_{cal}(m) - d_{obs}\|^2 + \left(J(m)^T (d_{cal} - d_{obs}) , dm \right) + o(\|dm\|^2)$$

Therefore

$$f(m + dm) - f(m) = \left(J(m)^T (d_{cal} - d_{obs}) , dm \right) + o(\|dm\|^2)$$
Gradient computation of a nonlinear least-squares function

For a perturbation $dm$ we have

$$ f(m + dm) = \frac{1}{2} \| d_{cal}(m + dm) - d_{obs} \|^2 $$

$$ \frac{1}{2} \| d_{cal}(m) - d_{obs} + J(m) dm + o(\| dm \|^2) \|^2 $$

where

$$ J(m) = \frac{\partial d_{cal}}{\partial m} $$

is the Jacobian matrix

$$ f(m + dm) = \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2 + (d_{cal} - d_{obs}, J(m) dm) + o(\| dm \|^2) $$

$$ \frac{1}{2} \| d_{cal}(m) - d_{obs} \|^2 + \left( J(m)^T (d_{cal} - d_{obs}), dm \right) + o(\| dm \|^2) $$

Therefore

$$ f(m + dm) - f(m) = \left( J(m)^T (d_{cal} - d_{obs}), dm \right) + o(\| dm \|^2) $$

$$ \nabla f(m) = J(m)^T (d_{cal} - d_{obs}) $$
Implementation for large-scale applications

- The size of $J(m)$ can be problematic for large scale applications.
- After discretization it is a matrix with $N$ rows and $M$ columns where
  1. $N$ is the number of discrete data
  2. $M$ is the number of discrete model parameters
- For Full Waveform Inversion for instance, we can have approximately
  \[ N \approx 10^{10}, \quad M \approx 10^9 \]

- This prevents from
  1. Computing $J(m)$ at each iteration of the inversion
  2. Storing $J(m)$ (or on disk but then expensive I/O and the performance severely decreases)
Can we avoid computing the Jacobian matrix?

Yes, using adjoint state methods
Outline

1 Numerical optimization methods for large-scale smooth unconstrained minimization problems

2 First-order and second-order adjoint state methods for gradient and Hessian-vector products computation
   - Gradient computation of a nonlinear least-squares function
   - First-order adjoint state method
   - Second-order adjoint state method

3 Summary
Specializing the forward problem

Now the problem is specialized such that

\[ d_{cal}(m) = Ru(m) \]

where \( u(m) \) satisfies

\[ A(m, \partial_x, \partial_y, \partial_z)u = s, \]

- \( u \) is the solution of the PDE (wavefield for instance) in all the volume
- \( R \) is an extraction operator as, most of the time, only partial measurements are available
First-order adjoint state method

References

- Adjoint state method come from optimal control theory and preliminary work of (Lions, 1968)
- It has been first applied to seismic imaging by (Chavent, 1974)
- A nice review of its application in this field has been proposed by (Plessix, 2006)
The Lagrangian function

From constrained optimization, we introduce the function

\[ L(m, u, \lambda) = \frac{1}{2} \| Ru - d \|^2 + (A(m, \partial_x, \partial_y, \partial_z)u - s, \lambda) \]
The Lagrangian function

From constrained optimization, we introduce the function

\[ L(m, u, \lambda) = \frac{1}{2} \| Ru - d \|^2 + (A(m, \partial_x, \partial_y, \partial_z)u - s, \lambda) \]

Link with the misfit function

Let \( \bar{u}(m) \) be the solution of the forward problem for a given \( m \), then

\[ L(m, \bar{u}(m), \lambda) = \frac{1}{2} \| R\bar{u}(m) - d \|^2 = f(m) \]
First-order adjoint state method

The Lagrangian function

From constrained optimization, we introduce the function

$$L(m, u, \lambda) = \frac{1}{2} \|Ru - d\|^2 + (A(m, \partial_x, \partial_y, \partial_z)u - s, \lambda)$$

Link with the misfit function

Let $\bar{u}(m)$ be the solution of the forward problem for a given $m$, then

$$L(m, \bar{u}(m), \lambda) = \frac{1}{2} \|R\bar{u}(m) - d\|^2 = f(m)$$

Link with the gradient of the misfit function

Therefore

$$\frac{\partial L(m, \bar{u}(m), \lambda)}{\partial m} = \nabla f(m)$$
First-order adjoint state method

Expending

This means that

\[
\left( \frac{\partial A(m, \partial_x, \partial_y, \partial_z)}{\partial m} \bar{u}(m), \lambda \right) + \frac{\partial L(m, \bar{u}(m), \lambda)}{\partial u} \frac{\partial \bar{u}(m)}{\partial m} = \nabla f(m)
\]
Expending

This means that

\[
\left( \frac{\partial A(m, \partial_x, \partial_y, \partial_z)}{\partial m} \bar{u}(m, \lambda) \right) + \frac{\partial L(m, \bar{u}(m), \lambda)}{\partial u} \frac{\partial \bar{u}(m)}{\partial m} = \nabla f(m)
\]

Potential simplification

Therefore, if we define \( \bar{\lambda}(m) \) such that

\[
\frac{\partial L(m, \bar{u}(m), \bar{\lambda}(m))}{\partial u} = 0
\]

we have

\[
\left( \frac{\partial A(m, \partial_x, \partial_y, \partial_z)}{\partial m} \bar{u}(m, \bar{\lambda}(m)) \right) = \nabla f(m)
\]
Adjoint state formula

What does mean

\[ \frac{\partial L(m, \bar{u}(m), \bar{\lambda}(m))}{\partial u} = 0? \]
First-order adjoint state method

Consider a perturbation \( du \). We have

\[
L(m, u + du, \lambda) = \frac{1}{2} \| Ru - d_{obs} + Rdu \|^2 + (A(m)u - s + A(m)du, \lambda)
\]

\[
= \frac{1}{2} \| Ru - d_{obs} \|^2 + (Ru - d_{obs}, Rdu) + (A(m)u - s, \lambda)
\]

\[
+ (A(m)du, \lambda) + o(\| du \|^2)
\]

\[
= L(m, u, \lambda) + \left( R^T(Ru - d_{obs}), du \right)
\]

\[
+ \left( du, A(m)^T \lambda \right) + o(\| du \|^2)
\]

\[
= L(m, u, \lambda) + \left( A(m)^T \lambda + R^T(Ru - d_{obs}), du \right) + o(\| du \|^2)
\]
Consider a perturbation $du$. We have

\[
L(m, u + du, \lambda) &= \frac{1}{2} \| Ru - d_{obs} + Rdu \|^2 + (A(m)u - s + A(m)du, \lambda) \\
&= \frac{1}{2} \| Ru - d_{obs} \|^2 + (Ru - d_{obs}, Rdu) + (A(m)u - s, \lambda) \\
&\quad + (A(m)du, \lambda) + o(\|du\|^2) \\
&= L(m, u, \lambda) + \left( R^T (Ru - d_{obs}), du \right) \\
&\quad + \left( du, (A(m)^T \lambda) \right) + o(\|du\|^2) \\
&= L(m, u, \lambda) + \left( A(m)^T \lambda + R^T (Ru - d_{obs}), du \right) + o(\|du\|^2)
\]

Therefore

\[
\frac{\partial L \left( m, \bar{u}(m), \bar{\lambda}(m) \right)}{\partial u} = A(m)^T \lambda + R^T (Ru - d_{obs})
\]
Adjoint state equation

Remember we are looking for $\lambda(m)$ such that

$$\frac{\partial L(m, \overline{u}(m), \overline{\lambda}(m))}{\partial u} = 0$$

This simply means that $\overline{\lambda}(m)$ should be the solution of the adjoint PDE

$$A(m)^T \lambda + R^T (R\overline{u}(m) - d_{obs}) = 0$$
First-order adjoint state method

### Adjoint state equation

Remember we are looking for \( \lambda(m) \) such that

\[
\frac{\partial L(m, \bar{u}(m), \bar{\lambda}(m))}{\partial u} = 0
\]

This simply means that \( \bar{\lambda}(m) \) should be the solution of the adjoint PDE

\[
A(m)^T \lambda + R^T (R\bar{u}(m) - d_{obs}) = 0
\]

### Self-adjoint case

- In some cases, the forward problem is self adjoint, and the adjoint state \( \bar{\lambda}(m) \) is the solution of the same equation than \( \bar{u}(m) \) except that the source term is different.
- In addition, the source term implies \( \bar{u}(m) \) has been computed beforehand, as it depends on this field.
We have seen that we can compute the gradient of the misfit function following the formula

$$\nabla f(m) = \left( \frac{\partial A(m, \partial_x, \partial_y, \partial_z)}{\partial m} \overline{u}(m), \overline{\lambda}(m) \right)$$

where $\overline{u}(m)$ satisfies

$$A(m, \partial_x, \partial_y, \partial_z)u = s,$$

and $\overline{\lambda}(m)$ satisfies

$$A(m, \partial_x, \partial_y, \partial_z)^T \lambda + R^T(R\overline{u}(m) - d_{obs}) = 0$$
Implementation issues

What are the benefits of the adjoint-state approach?

To compute the gradient, we first have to compute $\bar{u}(m)$: first PDE solve

Then we compute $\bar{\lambda}(m)$: second PDE solve

Finally we form the gradient through the formula

$$\nabla f(m) = \left( \frac{\partial A(m, \partial_x, \partial_y, \partial_z)}{\partial m} \bar{u}(m), \bar{\lambda}(m) \right)$$
First-order adjoint state method

Implementation issues

What are the benefits of the adjoint-state approach?

To compute the gradient, we first have to compute $\bar{u}(m)$: first PDE solve

Then we compute $\bar{\lambda}(m)$: second PDE solve

Finally we form the gradient through the formula

$$\nabla f(m) = \left( \frac{\partial A(m, \partial_x, \partial_y, \partial_z)}{\partial m} \bar{u}(m), \bar{\lambda}(m) \right)$$

The Jacobian matrix has never to be formed nor stored explicitly!
Outline

1. Numerical optimization methods for large-scale smooth unconstrained minimization problems

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation
   - Gradient computation of a nonlinear least-squares function
   - First-order adjoint state method
   - Second-order adjoint state method

3. Summary
Second-order adjoint state method

Computing Hessian-vector product

We have seen that in the particular case of the truncated Newton method, it is required to know how to compute, for any $v$, the Hessian-matrix product

$$H(m)v,$$

However, as it is the case for the Jacobian matrix $J(m)$ the size of matrix $H(m)$ for large-scale application is such that it cannot be computed explicitly nor stored.

Again, the adjoint-state method should allow us to overcome this difficulty

see (Fichtner and Trampert, 2011; Epanomeritakis et al., 2008; Métivier et al., 2013)
Principle of the method

Consider the function

\[ h_v(m) = (\nabla f(m), v) \]
Principle of the method

Consider the function

\[ h_\nu(m) = (\nabla f(m), \nu) \]

For a perturbation \( dm \) we have

\[
\begin{align*}
    h_\nu(m + dm) & = (\nabla f(m + dm), \nu) \\
                 & = (\nabla f(m) + H(m)dm, \nu) + o(||dm||^2) \\
                 & = (\nabla f(m), \nu) + (H(m)dm, \nu) + o(||dm||^2) \\
                 & = (\nabla f(m), \nu) + (dm, H(m)\nu) + o(||dm||^2) \\
                 & = h_\nu(m) + (dm, H(m)\nu) + o(||dm||^2)
\end{align*}
\]
**Second-order adjoint state method**

**Principle of the method**

Consider the function

\[ h_\nu(m) = (\nabla f(m), \nu) \]

For a perturbation \( dm \) we have

\[
\begin{align*}
    h_\nu(m + dm) &= (\nabla f(m + dm), \nu) \\
    &= (\nabla f(m) + H(m)dm, \nu) + o(\|dm\|^2) \\
    &= (\nabla f(m), \nu) + (H(m)dm, \nu) + o(\|dm\|^2) \\
    &= (\nabla f(m), \nu) + (dm, H(m)\nu) + o(\|dm\|^2) \\
    &= h_\nu(m) + (dm, H(m)\nu) + o(\|dm\|^2)
\end{align*}
\]

**\( H\nu \) through the gradient of \( h_\nu \)**

Therefore

\[ \nabla h_\nu(m) = H(m)\nu \]
Second-order adjoint state method

**Principle of the method**

Consider the function

\[ h_\nu(m) = (\nabla f(m), \nu) \]

For a perturbation \( dm \) we have

\[
\begin{align*}
  h_\nu(m + dm) &= (\nabla f(m + dm), \nu) \\
  &= (\nabla f(m) + H(m)dm, \nu) + o(\|dm\|^2) \\
  &= (\nabla f(m), \nu) + (H(m)dm, \nu) + o(\|dm\|^2) \\
  &= (\nabla f(m), \nu) + (dm, H(m)\nu) + o(\|dm\|^2) \\
  &= h_\nu(m) + (dm, H(m)\nu) + o(\|dm\|^2)
\end{align*}
\]

**\( H\nu \) through the gradient of \( h_\nu \)**

Therefore

\[ \nabla h_\nu(m) = H(m)\nu \]

**All we have to do is to apply the previous strategy to the function \( h_\nu(m) \)!**
Consider the new Lagrangian function

\[ L_v(m, u, \lambda, g, \mu_1, \mu_2, \mu_3) = (g, v) + \left( \left( \frac{\partial A(m)}{\partial m} u \right)^T \lambda - g, \mu_1 \right) + \\
\left( A(m)^T \lambda - R^T (Ru - d), \mu_2 \right) + \\
\left( A(m)u - s, \mu_3 \right) \]
Consider the new Lagrangian function

\[ L_v(m, u, \lambda, g, \mu_1, \mu_2, \mu_3) = (g, \nu) + \left( \left( \frac{\partial A(m)}{\partial m} u \right)^T \lambda - g, \mu_1 \right) + \]

\[ (A(m)^T \lambda - R^T (R u - d), \mu_2) + (A(m) u - s, \mu_3) \]

For \( u = \bar{u}(m), \lambda = \bar{\lambda}(m), g = \bar{g}(m) \) respectively solutions of

\[ A(m) u = s, \quad A(m)^T \lambda = R^T (R \bar{u}(m) - d_{obs}) \]

\[ g(m) = \left( \frac{\partial A(m)}{\partial m} \bar{u}(m) \right)^T \bar{\lambda}(m) \]

we have

\[ L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3) = h_v(m) \]
Consider the new Lagrangian function

\[
L_v(m, u, \lambda, g, \mu_1, \mu_2, \mu_3) = (g, v) + \left( \left( \frac{\partial A(m)}{\partial m} u \right)^T \lambda - g, \mu_1 \right) + (A(m)^T \lambda - R^T (R u - d), \mu_2) + (A(m) u - s, \mu_3)
\]

For \( u = \overline{u}(m), \lambda = \overline{\lambda}(m), g = \overline{g}(m) \) respectively solutions of

\[
A(m) u = s, \quad A(m)^T \lambda = R^T (R\overline{u}(m) - d_{obs}), \quad g(m) = \left( \frac{\partial A(m)}{\partial m} \overline{u}(m) \right)^T \overline{\lambda}(m)
\]

we have

\[
L_v(m, \overline{u}(m), \overline{\lambda}(m), \overline{g}(m), \mu_1, \mu_2, \mu_3) = h_v(m)
\]

Hence

\[
\frac{\partial L_v(m, \overline{u}(m), \overline{\lambda}(m), \overline{g}(m), \mu_1, \mu_2, \mu_3)}{\partial m} = \nabla h_v(m) = H(m)v
\]
Second-order adjoint state method

Again, we develop the previous expression

$$\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial m} =$$

$$\left( \left( \frac{\partial^2 A(m)}{\partial m^2} \bar{u}(m) \right)^T \bar{\lambda}(m), \mu_1 \right) +$$

$$\left( \frac{\partial A(m)}{\partial m} \bar{\lambda}(m), \mu_2 \right) + \left( \frac{\partial A(m)}{\partial m} \bar{u}(m), \mu_3 \right) +$$

$$\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial u} \frac{\partial \bar{u}}{\partial m} +$$

$$\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial \lambda} \frac{\partial \bar{\lambda}}{\partial m} +$$

$$\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial g} \frac{\partial \bar{g}}{\partial m}$$
Second-order adjoint state method

Now we look for $\mu_1, \mu_2, \mu_3$ such that

$$\begin{align*}
\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial u} &= 0 \\
\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial \lambda} &= 0 \\
\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial g} &= 0
\end{align*}$$
Now we look for $\mu_1, \mu_2, \mu_3$ such that

$$\begin{align*}
\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial u} &= 0 \\
\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial \lambda} &= 0 \\
\frac{\partial L_v(m, \bar{u}(m), \bar{\lambda}(m), \bar{g}(m), \mu_1, \mu_2, \mu_3)}{\partial g} &= 0
\end{align*}$$

This is equivalent to

$$\begin{align*}
\left( \frac{\partial A}{\partial m} \mu_1 \right)^T \bar{\lambda}(m) + R^T R \mu_2 + A(m)^T \mu_3 &= 0 \\
\left( \frac{\partial A}{\partial m} \bar{u}(m) \right)^T \mu_1 + A(m) \mu_2 &= 0 \\
v - \mu_1 &= 0
\end{align*}$$
Second-order adjoint state method

Reorganizing these equations, we find that

\[
\begin{align*}
\mu_1 &= \nu \\
A(m)\mu_2 &= -\left(\frac{\partial A}{\partial m} \overline{u}(m)\right)^T \nu \\
A(m)^T\mu_3 &= -\left(\frac{\partial A}{\partial m} \nu\right)^T \overline{\lambda}(m) + R^T R \mu_2
\end{align*}
\]
Reorganizing these equations, we find that

\[
\begin{aligned}
\mu_1 &= \nu \\
A(m)\mu_2 &= - \left( \frac{\partial A}{\partial m} \bar{u}(m) \right)^T \nu \\
A(m)^T \mu_3 &= - \left( \frac{\partial A}{\partial m} \nu \right)^T \bar{\lambda}(m) + R^T R \mu_2
\end{aligned}
\]

**Implementation**

- $\mu_1$ is given for free: it is $\nu$
- $\mu_2$ is the solution of a forward problem involving a new source term which depends on $\nu$ and $\bar{u}(m)$
- $\mu_3$ is the solution of an adjoint problem involving a new source term which depends on $b$, $\bar{\lambda}(m)$ and $\mu_2$
Second-order adjoint state method

Summary
The computation of $H(m)v$ for a given $v$ can be obtained through the formula

$$H(m)v = \left( \left( \frac{\partial^2 A(m)}{\partial m^2} \bar{u}(m) \right)^T \bar{\lambda}(m), \mu_1 \right) + \left( \frac{\partial A(m)}{\partial \lambda} \bar{\lambda}(m), \mu_2 \right) + \left( \frac{\partial A(m)}{\partial \mu} \bar{u}(m), \mu_3 \right)$$

(4)
Second-order adjoint state method

Summary

The computation of $H(m)v$ for a given $v$ can be obtained through the formula

$$H(m)v = \left( \left( \frac{{\partial}^2 A(m)}{{\partial} m^2} \bar{u}(m) \right)^T \bar{\lambda}(m), \mu_1 \right) + \left( \frac{{\partial} A(m)}{{\partial} m}^{\top} \bar{\lambda}(m), \mu_2 \right) + \left( \frac{{\partial} A(m)}{{\partial} m} \bar{u}(m), \mu_3 \right)$$

(4)

where

Forward and adjoint simulations

- $\bar{u}(m)$ is computed as a solution of the forward problem
- $\bar{\lambda}(m)$ is computed as a solution of the adjoint problem
- $\mu_2$ is computed as a solution of the forward problem for a new source term
- $\mu_3$ is computed as a solution of the adjoint problem for a new source term
Outline

1. Numerical optimization methods for large-scale smooth unconstrained minimization problems

2. First-order and second-order adjoint state methods for gradient and Hessian-vector products computation

3. Summary
Optimization methods for nonlinear least-squares problems

\[
\min_m f(m) = \frac{1}{2} \|d_{\text{cal}}(m) - d_{\text{obs}}\|^2
\]
Optimization methods for nonlinear least-squares problems

\[
\min_m f(m) = \frac{1}{2} \|d_{cal}(m) - d_{obs}\|^2
\]

An iterative scheme for local optimization

Local optimization methods are all based on the same iterative scheme

\[
m_{k+1} = m_k + \alpha_k \Delta m_k
\]
Optimization methods for nonlinear least-squares problems

$$\min_{m} f(m) = \frac{1}{2}\|d_{\text{cal}}(m) - d_{\text{obs}}\|^2$$

An iterative scheme for local optimization

Local optimization methods are all based on the same iterative scheme

$$m_{k+1} = m_k + \alpha_k \Delta m_k$$  \hspace{1cm} (5)

Four Nonlinear optimization methods

<table>
<thead>
<tr>
<th>Method</th>
<th>The differences come from the computation of $\Delta m_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steepest descent</td>
<td>$\Delta m_k = -\nabla f(m_k)$</td>
</tr>
<tr>
<td>Nonlinear CG</td>
<td>$\Delta m_k = -\nabla f(m_k) + \beta_k \Delta m_{k-1}$</td>
</tr>
<tr>
<td>l-BFGS</td>
<td>$\Delta m_k = -Q_k \nabla f(m_k)$, $Q_k \approx H_k^{-1}$</td>
</tr>
<tr>
<td>Truncated Newton</td>
<td>$H(m_k) \Delta m_k = -\nabla f(m_k)$ (solved with CG)</td>
</tr>
</tbody>
</table>
Adjoint methods

The gradient can be computed through the first-order adjoint method at the price

- 1 forward modeling
- 1 adjoint modeling

The Hessian-vector product (only required for truncated Newton) can be computed through second-order adjoint method at the price

- 1 additional forward modeling
- 1 additional adjoint modeling
A set of optimization routines in FORTRAN90

- Optimization routines for differentiable functions
- Steepest-descent, nonlinear conjugate gradient
- l-BFGS, truncated Newton
- Implemented using a reverse communication protocol: the user is in charge for computing gradient and Hessian-vector product

Open-source code available here

https://seiscope2.obs.ujf-grenoble.fr/SEISCOPE-OPTIMIZATION-TOOLBOX
Acknowledgments

Thank you for your attention

- National HPC facilities of GENCI-IDRIS-CINES under grant Grant 046091
- Local HPC facilities of CIMENT-SCCI (Univ. Grenoble) and SIGAMM (Obs. Nice)
Few references


