Generating Set Search Methods exploiting Curvature and Sparsity

Lennart Frimannslund and Trond Steihaug

lennart@ii.uib.no trond@ii.uib.no

Motivation

We want to solve:

$$\min_{x \in \mathbb{R}^n} f(x).$$

Where, for instance:

$$f(x) = g(x) + \int_a^b h(x) \, dx.$$
Motivation

We want to solve:

\[
\min_{x \in \mathbb{R}^n} f(x).
\]

Where, for instance:

\[
f(x) = g(x) + h(x),
\]

where \( h(x) \) is only known as

\[
\frac{dh}{dx}, \quad h(0) = \xi.
\]
Motivation

We want to solve:

$$\min_{x \in \mathbb{R}^n} f(x).$$

Where, for instance:

$$f(x, y) = g(x) + \max_{\theta} h(\theta | y).$$
We want to solve:
\[
\min_{x \in \mathbb{R}^n} f(x).
\]

Where, for instance:
\[
f(x, y) = g(x) + \max_{\theta} h(\theta | y).
\]

In all cases, the inexact solution of subproblems introduces \textit{numerical noise}.
Numerical Noise

Instead of the original function on the left we get a noisy function.
Numerical Noise

Instead of the original function on the left we get a noisy function.

This can cause problems for derivative-based methods.
Generating Set Searches / Pattern Search

Methods that search along the vectors of a generating set.
Generating Set Searches / Pattern Search

Methods that search along the vectors of a generating set.

Common variants:

✓ Coordinate Search
✓ Compass Search
✓ Rosenbrock’s Algorithm (1960)
Generating Set Searches / Pattern Search

Methods that search along the vectors of a generating set.

Common variants:
- Coordinate Search
- Compass Search
- Rosenbrock’s Algorithm (1960)

The simplex method of Nelder and Mead (1965) is not in this class.
A generating set is a set of vectors which span a space using only nonnegative coefficients, that is, vectors $q_i$ such that

$$x = \sum_{i=1}^{r} c_i q_i, \quad c_i \geq 0, \quad i = 1, \ldots, r \quad \forall x \in \mathbb{R}^n.$$
A generating set is a set of vectors which span a space using only nonnegative coefficients, that is, vectors $q_i$ such that

$$x = \sum_{i=1}^{r} c_i q_i, \quad c_i \geq 0, \quad i = 1, \ldots, r \quad \forall x \in \mathbb{R}^n.$$
A generating set is a set of vectors which span a space using only nonnegative coefficients, that is, vectors $q_i$ such that

$$x = \sum_{i=1}^{r} c_i q_i, \quad c_i \geq 0, \quad i = 1, \ldots, r \quad \forall x \in \mathbb{R}^n.$$ 

A generating set is sometimes also called a positive basis.
Coordinate Search

Idea: Search in each coordinate direction, positive as well as negative, step in the direction that reduces the objective function value the most.
Coordinate Search

Idea: Search in each coordinate direction, positive as well as negative, step in the direction that reduces the objective function value the most.

If no reduction can be found, shrink pattern.
Compass Search

Idea: Search in each coordinate direction, positive as well as negative, immediately step once objective function value reduction has been identified.
Pros and Cons of these Methods

Pros:

✓ Easy to implement
✓ Robust
✓ Well-suited for finding approximate solutions
✓ Provably convergent in many cases
Pros and Cons of these Methods

Pros:
✓ Easy to implement
✓ Robust
✓ Well-suited for finding approximate solutions
✓ Provably convergent in many cases

Cons:
✓ Slow rate of convergence
✓ Often unsuitable for large $n$
Modified Approach (Rosenbrock, 1960)

Idea: Search along coordinate directions, step immediately, \textit{rotate} search basis regularly to align principal search direction to search history.
Properties of Rosenbrock’s Method

Ingredients:

✓ Compass search-type search mechanism
✓ Replaces principal search direction with

\[ \frac{x_j - x_i}{\|x_j - x_i\|}, j > i. \]

✓ Generates remaining search directions through the Gram-Schmidt process
Properties of Rosenbrock’s Method

Ingredients:
- ✓ Compass search-type search mechanism
- ✓ Replaces principal search direction with 
  \[ \frac{x_j - x_i}{\|x_j - x_i\|}, \ j > i. \]
- ✓ Generates remaining search directions through the Gram-Schmidt process

Properties:
- ✓ Speeds up progress
- ✓ Has only one “good” search direction
Properties of Rosenbrock’s Method

Ingredients:

✓ Compass search-type search mechanism
✓ Replaces principal search direction with \( \frac{x_j - x_i}{\|x_j - x_i\|}, j > i. \)
✓ Generates remaining search directions through the Gram-Schmidt process

Properties:

✓ Speeds up progress
✓ Has only one “good” search direction

Can this be improved?
New Algorithm — Observations I

The original and noisy functions are similar on average.
New Algorithm — Observations I

The original and noisy functions are similar on average.

Specifically, their *average curvature* is the same.
New Algorithm — Observations II

For a quadratic function the Hessian $H$ is constant, and its eigenvectors make good search directions:
For a general function $f$, an element of $H$ can indirectly be approximated by:

$$\frac{f(x + hq_i + kq_j) - f(x + hq_i) - f(x + kq_j) + f(x)}{hk}.$$ 

for sufficiently large $h$ and $k$. We need four function evaluations, namely:

- $f(x)$,
- $f(x + hq_i)$,
- $f(x + kq_j)$,
- $f(x + hq_i + kq_j)$. 

---

Generating Set Search Methods exploiting Curvature and Sparsity – p. 14/24
Recall Compass Search

Compass search provides the necessary rectangles. We can store each average curvature entry in a matrix $H$. 

Generating Set Search Methods exploiting Curvature and Sparsity – p. 15/24
Recall Compass Search

Compass search provides the necessary rectangles. We can store each average curvature entry in a matrix $H_Q$. 
Search as in Compass Search, along $2n$ vectors
$\{\pm q_1 \ldots \pm q_n\}$. 

Generating Set Search Methods exploiting Curvature and Sparsity – p. 16/24
New Algorithm — Outline

- Search as in Compass Search, along $2n$ vectors $\{\pm q_1 \ldots \pm q_n\}$.
- Gather “rotated Hessian” elements as the search progresses, through adaptive shuffling of search directions before each sweep.

Generating Set Search Methods exploiting Curvature and Sparsity – p. 16/2
New Algorithm — Outline

✓ Search as in Compass Search, along $2n$ vectors $\{\pm q_1 \ldots \pm q_n\}$.

✓ Gather “rotated Hessian” elements as the search progresses, through adaptive shuffling of search directions before each sweep.

✓ Obtain rotated Hessian in $O(\frac{3}{4}n)$ sweeps.
New Algorithm — Outline

✓ Search as in Compass Search, along $2n$ vectors
\(\{\pm q_1 \ldots \pm q_n\}\).

✓ Gather “rotated Hessian” elements as the search progresses, through adaptive shuffling of search directions before each sweep.

✓ Obtain rotated Hessian in $O\left(\frac{3}{4}n\right)$ sweeps.

✓ Rotate rotated Hessian back: $H \leftarrow QHQ^T$.
New Algorithm — Outline

✓ Search as in Compass Search, along $2n$ vectors 
\[ \{ \pm q_1 \ldots \pm q_n \} \].

✓ Gather “rotated Hessian” elements as the search progresses, through adaptive shuffling of search directions before each sweep

✓ Obtain rotated Hessian in $O\left(\frac{3}{4}n\right)$ sweeps

✓ Rotate rotated Hessian back: 
\[ H \leftarrow Q H Q Q^T \]

✓ Eigenvalue-factorise Hessian
New Algorithm — Outline

✓ Search as in Compass Search, along \(2n\) vectors \(\{\pm q_1 \ldots \pm q_n\}\).

✓ Gather “rotated Hessian” elements as the search progresses, through adaptive shuffling of search directions before each sweep.

✓ Obtain rotated Hessian in \(O(\frac{3}{4}n)\) sweeps.

✓ Rotate rotated Hessian back: \(H \leftarrow QHQQT\).

✓ Eigenvalue-factorise Hessian.

✓ Use eigenvectors as new search basis.
### Numerical Results I — Smooth Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Method</th>
<th>$n$</th>
<th>RB</th>
<th>New</th>
<th>CS</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenbrock</td>
<td></td>
<td>16</td>
<td>178416</td>
<td>9662 (10)</td>
<td>175905</td>
<td>1228</td>
</tr>
<tr>
<td>Beale</td>
<td></td>
<td>2</td>
<td>381</td>
<td>210 (9)</td>
<td>635</td>
<td>51</td>
</tr>
<tr>
<td>Wood</td>
<td></td>
<td>4</td>
<td>2523</td>
<td>1991 (26)</td>
<td>6976</td>
<td>226</td>
</tr>
</tbody>
</table>
Numerical Results II — Noisy Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Method</th>
<th>New (n)</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wood, $n = 4$</td>
<td>New</td>
<td>1944 (26)</td>
<td>6975</td>
</tr>
<tr>
<td>Helical Valley, $n = 3$</td>
<td>New</td>
<td>651 (14)</td>
<td>10581</td>
</tr>
<tr>
<td>Disc. bound. value, $n = 5$</td>
<td>New</td>
<td>729 (7)</td>
<td>4894</td>
</tr>
</tbody>
</table>
Extension to functions with sparse Hessians

The method needs to estimate $O(n^2)$ rotated Hessian elements and uses $O(n)$ sweeps to do so.
Extension to functions with sparse Hessians

The method needs to estimate $O(n^2)$ rotated Hessian elements and uses $O(n)$ sweeps to do so.

If the Hessian only has $O(n)$ unique nonzero elements, we should be able to do this in $O(1)$ sweeps.
Extension to functions with sparse Hessians

The method needs to estimate $O(n^2)$ rotated Hessian elements and uses $O(n)$ sweeps to do so.

If the Hessian only has $O(n)$ unique nonzero elements, we should be able to do this in $O(1)$ sweeps.

If we know what the function looks like we can determine sparsity structure of derivatives. Several techniques are applicable.
Extension to functions with sparse Hessians — Problem

We compute a Hessian matrix in a rotated coordinate system.
We compute a Hessian matrix in a rotated coordinate system. However, a sparse Hessian needs *not* be sparse in the coordinate system defined by the columns of a general orthogonal matrix $Q$. 

\[ Q^T \begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix} Q = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \end{bmatrix}. \]
Extension to functions with sparse Hessians — Observation

Let

\[ H = \begin{bmatrix} h_1 & h_2 & \cdots & h_n \end{bmatrix}. \]
Extension to functions with sparse Hessians — Observation

Let

\[ H = \begin{bmatrix} h_1 & h_2 & \cdots & h_n \end{bmatrix}. \]

Then,

\[ Q^T H Q = H_Q \]

\[ \uparrow \]

\[ (Q^T \otimes Q^T) \text{vec}(H) = \text{vec}(H_Q). \]
Let

\[ H = \begin{bmatrix} h_1 & h_2 & \cdots & h_n \end{bmatrix}. \]

Then,

\[ Q^T HQ = H_Q \]

\[ \iff \]

\[ (Q^T \otimes Q^T)\text{vec}(H) = \text{vec}(H_Q). \]

Where \( \text{vec}(H) = (h_1^T \ h_2^T \ \cdots \ h_n^T)^T. \) Since \( H \) has many known zero elements and is symmetric, we can eliminate many rows and columns from the equation system.
Extension to functions with sparse Hessians — Solution

Don’t compute full rotated Hessian to perform

\[ H \leftarrow Q H Q^T. \]
Don’t compute full rotated Hessian to perform
\[ H \leftarrow QHQ^T. \]

Instead:

1. Compute \( r \) (the number of nonzeros in \( H \)) entries of \( HQ \), and corresponding rows of \((Q^T \otimes Q^T)\).
Extension to functions with sparse Hessians — Solution

Don’t compute full rotated Hessian to perform
\[ H \leftarrow QHQ^T. \]

Instead:

✓ Compute \( r \) (the number of nonzeros in \( H \)) entries of \( HQ \), and corresponding rows of \( (Q^T \otimes Q^T) \).

✓ Eliminate columns corresponding to zero entries, and take symmetry into account.
Don’t compute full rotated Hessian to perform
\[ H \leftarrow \mathbf{QH}_Q\mathbf{Q}^T. \]

Instead:

- ✓ Compute \( r \) (the number of nonzeros in \( H \)) entries of \( \mathbf{H}_Q \), and corresponding rows of \( (\mathbf{Q}^T \otimes \mathbf{Q}^T) \).

- ✓ Eliminate columns corresponding to zero entries, and take symmetry into account.

- ✓ Solve resulting \((r \times r)\) equation system

\[ \mathbf{A} \overrightarrow{\text{vec}}(H) = h_Q. \]
## Numerical Results III — Sparse Hessians

<table>
<thead>
<tr>
<th>Function</th>
<th>Method</th>
<th>Regular</th>
<th>Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext. Rosenb. $n = 4$</td>
<td>1015</td>
<td>937</td>
<td></td>
</tr>
<tr>
<td>Ext. Rosenb. $n = 8$</td>
<td>3180</td>
<td>1854</td>
<td></td>
</tr>
<tr>
<td>Ext. Rosenb. $n = 16$</td>
<td>8915</td>
<td>3769</td>
<td></td>
</tr>
<tr>
<td>Ext. Rosenb. $n = 32$</td>
<td>24558</td>
<td>7552</td>
<td></td>
</tr>
<tr>
<td>Ext. Rosenb. $n = 64$</td>
<td>96920</td>
<td>15049</td>
<td></td>
</tr>
</tbody>
</table>
The new algorithm seems to be an improvement over similar existing algorithms in terms of the number of function evaluations required to reach the optimiser. This comes at the cost of several eigenvalue factorisations, (and the solution of large equation systems in the sparse case) but if these are assumed to relatively cheap compared to function evaluations, this is not a problem.