Edge and Vertex Elimination

Matthias Kuhn, Paul Wilhelm

1. Juli 2009
Content

1 Prerequisites
   - computational procedure
   - computational graph
   - linearized graph

2 Edge Elimination
   - Back Elimination
   - Front Elimination

3 Vertex Elimination
   - Markowitz degree
   - Greedy Markowitz
   - Relative Markowitz degree
   - Implementation of relative Markowitz strategy

4 Line-Graph
   - Face-Elimination on the Line-Graph

5 Testproblems
   - Bratu problem
   - Convection Equation
A nonlinear vector function \( F : \mathbb{R}^n \rightarrow \mathbb{R}^m \) is implemented in a program as a sequence of \( L \) assignments of the values of elemental functions \( \varphi_i \).

**computational procedure**

<table>
<thead>
<tr>
<th>( v_{i-n} )</th>
<th>( x_i )</th>
<th>( i = 1, \ldots, n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_i )</td>
<td>( \varphi_i(v_j)_{j \prec i} )</td>
<td>( i = 1, \ldots, L )</td>
</tr>
<tr>
<td>( y_{m-i} )</td>
<td>( v_{L-i} )</td>
<td>( i = m-1, \ldots, 0 )</td>
</tr>
</tbody>
</table>

where \( j \prec i \) means that \( v_i \) depends directly on \( v_j \) and \( \prec^* \) is the transitive closure of \( \prec \), for example \( 0 \prec 1 \prec 2 \implies 0 \prec^* 2 \).

These relations can be visualized as an acyclic graph, so called **computational graph** \( G \).

**computational graph**

\[
V = \{ i : v_i \in F \} \quad \text{(vertices)}
\]
\[
E = \{ (j, i) : j \prec i \} \quad \text{(edges)}
\]
\[
G = (V, E)
\]
**linearized graph**

*linearized graph*: We label each edge of $G$ by the corresponding local partial derivative

$$c_{i,j} \equiv \frac{\partial \varphi_i}{\partial v_j} \quad \text{for} \ (j, i) \in E$$

**bipartite graph**

A graph $G$ consists of

- independents: $v_{i-n}$ for $i = 1, \ldots, n$
- intermediates: $v_i$ for $i = 1, \ldots, L$
- dependents: $v_{L-i}$ for $i = m-1, \ldots, 0$

A **bipartite graph** is a graph with no intermediates; local partial derivatives are in this case global partial derivatives $\frac{\partial y_i}{\partial x_j}$.

**Example - bipartite graph**

```
\begin{align*}
&x_1 \quad \quad \quad x_2 \quad \quad \quad x_3 \\
&\downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \\
&y_1 \quad \quad \quad y_2
\end{align*}
```
Back Elimination

Rule

- Back eliminate \( k = (i, j) \in E \) by introducing new edges \( k' = (i', j) \) for all \( i' \prec i \) and \( i' \not\prec j \).
  - Set the edge label to \( c_{k'} = c_k c_{k''} \), where \( k'' = (i', i) \).
- For all \( k' = (i', j) \in E \) update the edge labels to \( c_{k'} = c_{k'} + c_{k''} c_k \).

Example - back elimination of \((i, j)\)
Front Elimination

Rule

- Front eliminate $k = (i, j) \in E$ by introducing new edges $k' = (i, j')$ for all $j'$ with $i \prec j'$ and $i \not\prec j'$.
  Set the edge label to $c_{k'} = c_k c_{k''}$, where $k'' = (j, j')$.
- For all $k' = (i, j') \in E$ update the edge labels to $c_{k'} = c_{k'} + c_{k''} c_k$

Example - front elimination of $(i, j)$
Elimination of isolated Vertices

Rule
In both cases, front and back elimination, an isolated vertex is removed together with all its remaining incident or emanating edges.

Example - front elimination of (i,j)
Vertex Elimination and Markowitz degree

Rule - Vertex Elimination 1
A Vertex $i \in V$ is eliminated from $G$ by front elimination of all its in-edges.

Rule - Vertex Elimination 2
A Vertex $i \in V$ is eliminated from $G$ by back elimination of all its out-edges.

Markowitz degree
The **Markowitz degree** is the number of multiplications needed to eliminate a vertex $i$ and is equal to

$$mark(i) = |\{i' : i' \prec i\}| \cdot |\{i'' : i \prec i''\}|.$$

In other words: The number of predecessors multiplied with the number of successors.
Greedy Markowitz

- **Aim**: Minimize the total number of multiplications during Jacobian elimination process
- **Approach**: Always select a vertex for elimination that has minimal Markowitz degree

This strategy is not always optimal:

Example - A. Griewank, Evaluating Derivatives 2. Edition, Figure 10.4
Relative Markowitz degree

- "How much will it cost to eliminate this vertex later rather than now?"
- Markowitz degree of a vertex may oscillate as its neighbors are eliminated.
- The number of independents $|X_i|$ and the number of dependents $|Y_i|$ a vertex $i$ is connected to is invariant with respect to vertex-elimination.
- Cost of eliminating a vertex $i$ at last is

$$mark_{ultimate}(i) = |X_i||Y_i|$$

- Subtracting this ultimate Markowitz degree of the current Markowitz degree, we obtain the relative Markowitz degree

$$mark_{relative}(i) \equiv mark(i) - |X_i||Y_i|$$
Implementation of relative Markowitz strategy

- overloaded operators for class Vertex create graph
- class Vertex consists of:
  - value (double)
  - ultimate Markowitz degree (int)
  - relative Markowitz degree (int)
  - list of predecessors and edge-values ( = local partial derivatives )
  - list of successors
  - list of independents
  - list of dependents
  - all the above lists are lists of pointers to Vertex objects
- graph is stored as list of Vertex objects
  - graph structure exists only indirect, in the form of pointer stuff in each Vertex object
Implementation of relative Markowitz strategy

- The list of Vertex objects has special ordering
  - Inserting the dependents into the Vertex objects can be done linear
  - Same for computing ultimate Markowitz degree
- Locating the Vertex with minimal relative Markowitz degree is also linear
- Deletion of a Vertex affects only its predecessors and successors
  - Relative Markowitz must only be updated locally
  - Back elimination is used for deletion
- The order of deletion in graph minimization process is taped
  - E.g. reuse the tape in Newton’s iteration
Face-Elimination on the Line-Graph

Line-Graph

\[ V' = \bar{E} = \{(j, i) : j < i\} \]
\[ \cup \{(-\infty, j - n)\}_{j=1...n} \]
\[ \cup \{(l - m + i, \infty)\}_{j=1...m} \]

\[ E' = \{(i, j, k) : (i, j) \in E \ni (j, k)\} \] (edges/faces)

\[ G' = (V', E') \]

(The vertices of the Line-Graph are the edges of the computational graph, including a '∞' source and sink vertex.)

Now could the value of the derivative \( c \) of each edge in the computational graph saved in the vertices of the Line-Graph. If you could eliminate all intermediate faces the Line-Graph is tripartite the values correspond to the Jacobian.
Face-Elimination on the Line-Graph

Example: $a = 2, \ b = -2$

$x = \sin(a + b + \cos(a + b)), \ y = \cos(a + b + \cos(a + b))$
Face-Elimination on the Line-Graph

An interior edge \((i, j, k)\) of the Line-Graph which connected \((i, j)\) and \((j, k)\) could eliminated, by creating a new vertex \((i, k)\) and connect this to all predecessors of \((i, j)\) and all successors of \((j, k)\). Its derivative is defined by \(c_{i,k} = c_{i,j} \cdot c_{j,k}\).

Example - eliminating \((2, 0) \rightarrow (0, 1)\)
## Face-Elimination on the Line-Graph

### Elimination rules for eliminating an interior face $(i, j) \rightarrow (j, k)$

1. **Merge/Absorption**: If there exists a Vertex $(\tilde{i}, \tilde{k}) \in G'$ such that its predecessors are equal to the predecessors of $(i, j)$ and its successors are equal to the successors of $(j, k)$, don’t create a new vertex. The vertex $(\tilde{i}, \tilde{k})$ ’absorb’ the value of the derivative:

   $$c_{i,k}^+ = c_{i,j} \cdot c_{j,k}$$

2. Remove $(i, j, k) \in E'$

3. Remove $(i, j)$ if it is isolated (no successors). Otherwise try to merge. (Set $c_{i,k}^+ = c_{i,j}$, then remove $(i, j)$)

4. Also Remove $(j, k)$ if it is isolated (no predecessors). Otherwise try to merge. (Set $c_{i,k}^+ = c_{j,k}$, then remove $(j, k)$)

After finitely many eliminations you got a tripartite Graph.
Face-Elimination on the Line-Graph

Example

now eliminating \((0, 1) \rightarrow (1, 1)\) (and remove)

\[\begin{align*}
(\infty, 2) & \rightarrow (2, 0) \\
&(2, 1) \\
&(0, 1) \rightarrow (1, \sin(1)) \rightarrow (\sin(1), \infty) \\
(0, 1) & \rightarrow (0, 1) \\
(\infty, -2) & \rightarrow (-2, 0) \\
&(0, 1) \\
&(1, 1) \rightarrow (1, \cos(1)) \rightarrow (\cos(1), \infty)
\end{align*}\]
Face-Elimination on the Line-Graph

Example

then eliminate \((2, 0) \rightarrow (0, 1)\) (absorption by \((2, 1)\))

\[
\begin{align*}
(−∞, 2) & \rightarrow (2, 0) \\
(0, 1) & \rightarrow (1, sin(1)) \rightarrow (sin(1), 1)
\end{align*}
\]
Face-Elimination on the Line-Graph

Example

before you can go on you have to merge \((0, 1)\)

\[
\begin{align*}
(0, 1) & \quad \rightarrow \\
(1, \sin(1)) & \quad \rightarrow \\
\text{to} & \quad \rightarrow \\
(\sin(1), \infty) & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
(2, 1) & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
(\infty, 2) & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
(\infty, -2) & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
(0, 1) & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
(1, \cos(1)) & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
\rightarrow & \quad \rightarrow \\
(\cos(1), \infty) & \quad \rightarrow
\end{align*}
\]
Path-Value

**Definition**

For a face \((i, j) \rightarrow (j, k)\) it’s Path-Value is defined by

\[ |\mathcal{X} \rightarrow (i, j)| \cdot |(j, k) \rightarrow \mathcal{Y}|,\]

with \(\mathcal{X} := \{(-\infty, r)\}\) and \(\mathcal{Y} := \{(s, \infty)\}\).

**Example**

\[
\begin{align*}
(-\infty, 2) & \xrightarrow{0} (2, 0) \xrightarrow{1.2} (0, 1) \xrightarrow{2.1} (1, \sin(1)) \xrightarrow{0} (\sin(1), \infty) \\
(-\infty, -2) & \xrightarrow{0} (-2, 0) \xrightarrow{1.2} (0, 1) \xrightarrow{2.2} (1, 1) \xrightarrow{2.1} (1, \cos(1)) \xrightarrow{0} (\cos(1), \infty)
\end{align*}
\]
Implementation Path-Value Reduction

Implementation

- I needed a class (LGSubVertex) for the sub-vertices of the computational Graph and a class (LGVertex) for the Vertices of the line-graph.

- graph is stored as list of vertices (pointers to objects of class LGVertex).

- graph structur exists also only indirect; every LGVertex contains pointers to its predecessors and successors and its sub-vertices. Also a sub-vertex $i$ contains pointers to all line-graph vertices ($..., i$)

- overloaded operators $\implies$ creation of new LGSubVertex and one (unary operation) or two (binary operation) LGVertex. Because parts of the path-value could computed here therefore I used the variables front-path-value which is computed here and back-path-value which is computed later.
Implementation Path-Value Reduction

Implementation

- every Vertex object contains list of pointers to the in- / dependents.
- the list of Vertices is ordered in a way that for each Vertex in the list its predecessors occur further at the front of the list.
  - After initializing the graph of the computations I have to initialize the sink vertices.
  - inserting the dependents into the Vertices can be done linear to the number of Vertices. Computing the (back-)path-value can done at the same time.
- finding the Vertex with maximal path-value is linear to number of existing Vertices.
- after each face-elimination I have to update the path-values (needed some integer additions and multiplications)
Bratu problem

\[
\begin{align*}
\Delta u - \lambda e^u &= 0 \quad \text{on } \Omega \\
u &= 0 \quad \text{on } \partial \Omega
\end{align*}
\]

After discretization of this problem on a \( n \times n \)-grid we get:

\[
Au + \lambda e^u = F(u) = 0
\]

with the difference operator \( A \):

\[
Au_{i,j} = (u_{i,j-1} + u_{i,j+1} + u_{i-1,j} + u_{i+1,j} - 4u_{i,j})/h^2
\]

The relative Markowitz strategy is used to get \( F'(u) \), needed for the Newton-step, in a Newton-iteration to compute the root of \( F(u) \).
Convection Equation

Convection equation

\[
\begin{align*}
\Delta u - \lambda e^u &= 0 \quad \text{on } \Omega_1 \\
v \cdot \nabla w - \Delta w &= 0 \quad \text{on } \Omega_2 \setminus \Omega_1 \\
\text{div}(v) &= 0 \quad \text{on } \Omega_2 \setminus \Omega_1
\end{align*}
\]

boundary conditions:

\[
\begin{align*}
w &= u \quad \text{on } \partial \Omega_1 \\
\frac{\partial w}{\partial \vec{n}} &= \frac{\partial u}{\partial \vec{n}} \quad \text{on } \partial \Omega_1 \text{ with the normal vector } \vec{n} \\
v &= 0 \quad \text{on } \partial \Omega_1 \\
w &= 0 \quad \text{on } \partial \Omega_2
\end{align*}
\]

This set of equations somehow describes the flow of heat from the domain \( \Omega_1 \) surrounded by a fluid in \( \Omega_2 \) due to convection. \( v \) is a 2D vector field defined on \( \Omega_2 \). \( u \) is a scalar field on \( \Omega_1 \) and \( w \) is the extension of this same scalar field in \( \Omega_2 \).
Results for Bratu-Problem
## Results for Bratu-Problem

### Forward

<table>
<thead>
<tr>
<th></th>
<th>mults</th>
<th>mult * 5</th>
<th>adds</th>
<th>adds * 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>derivative</td>
<td>2116</td>
<td>10580</td>
<td>2645</td>
<td>13225</td>
</tr>
<tr>
<td>function</td>
<td>1587</td>
<td>7935</td>
<td>2645</td>
<td>13225</td>
</tr>
<tr>
<td>sum</td>
<td>18515</td>
<td></td>
<td></td>
<td>26450</td>
</tr>
</tbody>
</table>

### Reverse

Length of Tape = 5386

<table>
<thead>
<tr>
<th></th>
<th>mults</th>
<th>mult * 5</th>
<th>adds</th>
<th>adds * 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>derivative</td>
<td>2116</td>
<td>12696</td>
<td>7406</td>
<td>44436</td>
</tr>
<tr>
<td>function</td>
<td>1587</td>
<td></td>
<td>2645</td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>14283</td>
<td></td>
<td></td>
<td>47081</td>
</tr>
</tbody>
</table>
Markowitz

<table>
<thead>
<tr>
<th></th>
<th>integer-mults</th>
<th>integer-adds</th>
</tr>
</thead>
<tbody>
<tr>
<td>marko degree</td>
<td>25392</td>
<td>17986</td>
</tr>
<tr>
<td>for 3 iterations</td>
<td>46552</td>
<td>39146</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>mults</th>
<th>adds</th>
<th>mults (3 iterations)</th>
<th>adds (3 iterations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>derivative</td>
<td>7935</td>
<td>529</td>
<td>23805</td>
<td>1587</td>
</tr>
<tr>
<td>function</td>
<td>1587</td>
<td>2645</td>
<td>4761</td>
<td>7935</td>
</tr>
<tr>
<td>sum</td>
<td>9522</td>
<td>3174</td>
<td>28566</td>
<td>9522</td>
</tr>
</tbody>
</table>

Pathlength

path length: 49197 (int-mults) + 153410 (int-adds)

<table>
<thead>
<tr>
<th></th>
<th>mults</th>
<th>adds</th>
</tr>
</thead>
<tbody>
<tr>
<td>derivative</td>
<td>7935</td>
<td>529</td>
</tr>
<tr>
<td>function</td>
<td>1587</td>
<td>2645</td>
</tr>
<tr>
<td>sum</td>
<td>9522</td>
<td>3174</td>
</tr>
</tbody>
</table>