Jacobian Compression I

Introduction and Band Structure

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University

Contents

Outline

Objective

- Introduction to efficient accumulation of banded square Jacobians through exploitation of sparsity.

Learning Outcomes

- You will understand
  - structural orthogonality
  - Curtis-Powell-Reid seeding for finite differences and tangent/adjoint mode Algorithmic Differentiation
  - direct Jacobian compression problems
- You will be able to
  - implement Curtis-Powell-Reid seeding for finite differences and/or dco/c++

Outline

Let a multivariate vector function

$$F : \boldsymbol{R}^n \to \boldsymbol{R}^m : \mathbf{y} = F(\mathbf{x})$$

be implemented as a computer program. F is assumed to be continuously differentiable over its domain implying the existence of the Jacobian

$$F'(\mathbf{x}) \equiv \frac{dF}{d\mathbf{x}}(\mathbf{x}) \in \boldsymbol{R}^{m \times n}$$

the individual columns of which can be approximated at all points $\mathbf{x} \in \boldsymbol{R}$ by (forward, backward, central) finite difference quotients as follows:

$$F'(\mathbf{x}) \approx_1 \left(\frac{F(\mathbf{x} + \Delta x_j \cdot \mathbf{e}_i) - F(\mathbf{x})}{\Delta x_j} \right)_{i=0}^{n-1} \approx_1 \left(\frac{F(\mathbf{x}) - F(\mathbf{x} - \Delta x_j \cdot \mathbf{e}_i)}{\Delta x_j} \right)_{i=0}^{n-1}$$
$$\approx_2 \left(\frac{F(\mathbf{x} + \Delta x_j \cdot \mathbf{e}_i) - F(\mathbf{x} - \Delta x_j \cdot \mathbf{e}_i)}{2 \cdot \Delta x_j} \right)_{i=0}^{n-1}$$

Definitions:

- Orthogonality of $\mathbf{u}, \mathbf{v} \in R^n$: $\langle \mathbf{u}, \mathbf{v} \rangle \equiv \mathbf{u}^T \cdot \mathbf{v} = \mathbf{v}^T \cdot \mathbf{u} = 0$.
- Structural orthogonality of $\mathbf{u}, \mathbf{v} \in R^n$: $\langle \mathbf{u}, \mathbf{v} \rangle = 0 \wedge \nexists i : u_i \neq 0 \wedge v_i \neq 0$.

Example:

- Exploitation of structural orthogonality in band matrices

$$\begin{pmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{pmatrix} \Rightarrow \begin{pmatrix} a & b & \\ c & d & e \\ h & f & g \\ j & k & i \\ l & m & \end{pmatrix}$$

- Compressed storage and simultaneous computation of structurally orthogonal columns

structurally orthogonal $\Rightarrow$ orthogonal $\left(\not\Leftarrow \text{e.g., } \begin{pmatrix} 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ -1 \end{pmatrix} = 0 \right)$

Outline

Let the structurally orthogonal columns of a Jacobian $F' \in \boldsymbol{R}^{m \times n}$ be partitioned into index sets $\mathcal{I}_j \subseteq \{0, \ldots, n-1\}$ such that $\cup_j \mathcal{I}_j = \{0, \ldots, n-1\}$ and $\mathcal{I}_i \cap \mathcal{I}_j = \varnothing$ for $i \neq j$.

The columns of each index set can be computed simultaneously as

$$F'_{*,\mathcal{I}_j} \approx \frac{F(\mathbf{x} + \Delta\mathbf{x}_{\mathcal{I}_j}) - F(\mathbf{x})}{\Delta\mathbf{x}_{\mathcal{I}_j}},$$

since

$$\forall (k, l) \mid k, l \in \mathcal{I}_i : \langle [F']_{*,k}, [F']_{*,l} \rangle = 0 \,.$$

and where $\Delta\mathbf{x}_{\mathcal{I}_j}$ contains the perturbations corresponding to the entries in $\mathbf{x}$ with indexes in $\mathcal{I}_j$.

Division by $\Delta\mathbf{x}_{\mathcal{I}_j}$ is defined element-wise.

Curtis-Powell-Reid Seeding

Finite Differences (Implementation)


```
1  template<class T>
2  void dFdx_CPR(
3        int bw,
4        const std::vector<T> &x,
5        std::vector<std::vector<T>> &dydx
6  ) {
7    int n=x.size(); assert(n%bw==0); assert(bw%2==1);
8    std::vector<T> y(n);
9    F(bw,x,y);
10   for (int k=0;k<bw;k++) {
11     std::vector<T> dx(n/bw), xp=x, yp(n);
12     for (int i=k,j=0;i<n;i+=bw,j++) {
13       dx[j]=(x[i]==0) ? sqrt(std::numeric_limits<T>::epsilon())
14                       : sqrt(std::numeric_limits<T>::epsilon())*fabs(x[i]);
15       xp[i]+=dx[j];
16     }
17     F(bw,xp,yp);
18     for (int i=0;i<n;i++) dydx[i][k]=(yp[i]-y[i])/dx[i%(n/bw)];
19   }
20 }
```

Curtis-Powell-Reid Seeding

Unpacking

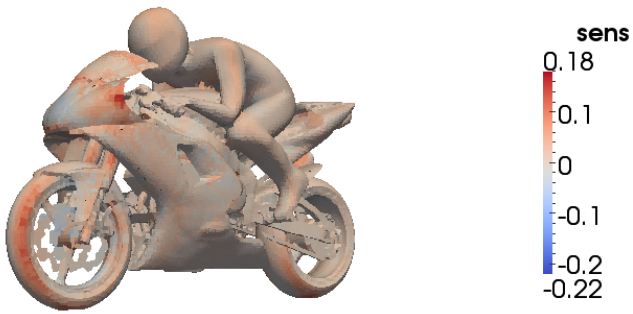

```
1  template<class T>
2  void dFdx_CPR_unpack(
3     const std::vector<std::vector<T>> &dydx_CPR,
4     std::vector<std::vector<T>> &dydx
5  ) {
6     int n=dydx_CPR.size(), bw=dydx_CPR[0].size();
7     assert(n%bw==0); assert(bw%2==1);
8     for (int i=0;i<n;i++)
9        for (int j=max(0,i-bw/2);j<=min(n-1,i+bw/2);j++)
10          dydx[i][j]=dydx_CPR[i][j%bw];
11 }
```

Let $\mathbf{y} = F(\mathbf{x})$, $F : \boldsymbol{R}^n \to \boldsymbol{R}^m$, be continuously differentiable.

Tangent AD yields

$$\boldsymbol{R}^m \ni \mathbf{y}^{(1)} = \underset{\in \boldsymbol{R}^{m\times n}}{F'(\mathbf{x})} \cdot \underset{\in \boldsymbol{R}^n}{\mathbf{x}^{(1)}}$$

and, hence, the Jacobian F' at $O(n) \cdot \mathrm{Cost}(F)$;



Adjoint AD yields

$$\boldsymbol{R}^{1\times n} \ni \mathbf{x}_{(1)} = \underset{\in \boldsymbol{R}^{1\times m}}{\mathbf{y}_{(1)}} \cdot F'(\mathbf{x})$$

and, hence, yield Jacobian at $O(m) \cdot \mathrm{Cost}(F)$ (cheap gradients).

See also modules I-III on algorithmic differentiation.

Let the structurally orthogonal columns

$$[F']_{*,k} = F' \cdot \mathbf{e}_k, \quad k = 0, \ldots, n-1$$

of the Jacobian

$$F' = F'(\mathbf{x}) = ([F']_{i,j})_{j=0,\ldots,n-1}^{i=0,\ldots,m-1} \in \boldsymbol{R}^{m \times n}$$

be partitioned into index sets $\mathcal{I}_j \subseteq \{0, \ldots, n-1\}$ such that $\cup_j \mathcal{I}_j = \{0, \ldots, n-1\}$ and $\mathcal{I}_i \cap \mathcal{I}_j = \varnothing$ for $i \neq j$.

The columns of each index set can be computed simultaneously in tangent mode AD as

$$F'_{*,\mathcal{I}_j} = F' \cdot \sum_{i \in \mathcal{I}_j} \mathbf{e}_i \ .$$

Curtis-Powell-Reid Seeding

Implementation of Tangent AD with dco/c++


```
1  #include "dco.hpp"
2
3  template<class T>
4  void dFdx_CPR(
5        int bw,
6        const std::vector<T> &x,
7        std::vector<std::vector<T>> &dydx
8  ) {
9     int n=x.size(); assert(n%bw==0); assert(bw%2==1);
10    using DCO_T=typename dco::gt1s<T>::type;
11    for (int k=0;k<bw;k++) {
12       std::vector<DCO_T> x_t(n), y_t(n);
13       dco::value(x_t)=x;
14       for (int i=k;i<n;i+=bw) dco::derivative(x_t[i])=1;
15       F(bw,x_t,y_t);
16       for (int i=0;i<n;i++) dydx[i][k]=dco::derivative(y_t[i]);
17    }
18 }
```

Let the structurally orthogonal rows

$$[F']_{i,*} = \mathbf{e}_i^T \cdot F', \quad i = 0, \ldots, m-1$$

of the Jacobian

$$F' = F'(\mathbf{x}) = ([F']_{i,j})_{j=0,\ldots,n-1}^{i=0,\ldots,m-1} \in \boldsymbol{R}^{m \times n}$$

be partitioned into index sets $\mathcal{I}_j \subseteq \{0, \ldots, m-1\}$ such that $\cup_j \mathcal{I}_j = \{0, \ldots, m-1\}$ and $\mathcal{I}_i \cap \mathcal{I}_j = \varnothing$ for $i \neq j$.

The rows of each index set can be computed simultaneously in adjoint mode AD as

$$F'_{\mathcal{I}_j,*} = \sum_{i \in \mathcal{I}_j} \mathbf{e}_i^T \cdot F' \ .$$

Curtis-Powell-Reid Seeding

Implementation of Adjoint AD with dco/c++


```
1  template<class T>
2  void dFdx_CPR(
3        int bw, const std::vector<T> &x,
4        std::vector<std::vector<T>> &dydx
5  ) {
6    int n=x.size(); assert(n%bw==0); assert(bw%2==1);
7    using DCO_M=typename dco::ga1s<T>;
8    using DCO_T=typename DCO_M::type;
9    using DCO_TT=typename DCO_M::tape_t;
10   std::vector<DCO_T> x_a(n), y_a(n);
11   dco::value(x_a)=x;
12   DCO_M::global_tape=DCO_TT::create();
13   DCO_M::global_tape->register_variable(x_a);
14   F(bw,x_a,y_a);
15   for (int k=0;k<bw;k++) {
16     for (int i=k;i<n;i+=bw) dco::derivative(y_a[i])=1;
17     DCO_M::global_tape->interpret_adjoint();
18     for (int i=0;i<n;i++) dydx[i][k]=dco::derivative(x_a[i]);
19     DCO_M::global_tape->zero_adjoints();
20   }
21   DCO_TT::remove(DCO_M::global_tape);
22 }
```

Outline

Detection of Sparsity Pattern

Comments

- Numerical sparsity information by finite differences / AD?
  - Same computational cost as dense Jacobian accumulation
  - May be unlucky due to numerical zeros, e.g. $y = \sin(x)$ at $x = 0$
- Structural sparsity information requires propagation of index sets or of (vectors of) Boolean flags
- Computational overhead to be amortized trough repeated compression based on same pattern, e.g, in Newton's method

Outline

The (optimization version of the) BANDWIDTH problem asks for a permutation of the columns and/or rows of a sparse matrix such that the maximal distance of any nonzero entry from the diagonal is minimized.

BANDWIDTH is NP-complete.

Example:

$$\underbrace{\begin{pmatrix} h & & g & \\ & c & d & b \\ & e & f & \\ & & & a \end{pmatrix}}_{\text{distance}=2} \xrightarrow[c_0 \leftrightarrow c_3]{r_0 \leftrightarrow r_3} \underbrace{\begin{pmatrix} a & & & \\ b & c & d & \\ & e & f & \\ & & g & h \end{pmatrix}}_{\text{distance}=1}$$

See literature for heuristics and further reference.

Outline

For a given sparsity pattern

$$P = P(F') \in \{0,1\}^{m \times n} \quad \text{of} \quad F' = F(\mathbf{x}) \in \boldsymbol{R}^{m \times n}$$

find

$$X^{(1)} \in \{0,1\}^{m \times n^{(1)}}$$

with minimal $n^{(1)}$ such that F' can be recovered by direct substitution from

$$\boldsymbol{R}^{m \times n^{(1)}} \ni Y^{(1)} = F' \cdot X^{(1)} \, .$$

Example:

$$\begin{pmatrix} a? & b? & \\ & c? & d? \\ e? & & \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} a & b \\ d & c \\ e & \end{pmatrix}$$

For a given sparsity pattern

$$P = P(F') \in \{0,1\}^{m \times n} \quad \text{of} \quad F' = F(\mathbf{x}) \in \boldsymbol{R}^{m \times n}$$

find

$$Y_{(1)} \in \{0,1\}^{m_{(1)} \times n}$$

with minimal $m_{(1)}$ such that F' can be recovered by direct substitution from

$$\boldsymbol{R}^{m_{(1)} \times n} \ni X_{(1)} = Y_{(1)} \cdot F' \ .$$

Example:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a? & b? & \\ & c? & d? \\ e? & & \end{pmatrix} = \begin{pmatrix} a & b & \\ e & c & d \end{pmatrix}$$

Find

$$X^{(1)} \in \{0,1\}^{m \times n^{(1)}} \quad \text{and} \quad Y_{(1)} \in \{0,1\}^{m_{(1)} \times n}$$

with minimal $n^{(1)} + \mathcal{R} \cdot m_{(1)}$, where $\mathcal{R}$ denotes the cost of an adjoint relative to a tangent, such that F' can be recovered by direct substitution from

$$\boldsymbol{R}^{m \times n^{(1)}} \ni Y^{(1)} = F' \cdot X^{(1)} \quad \text{and} \quad \boldsymbol{R}^{m_{(1)} \times n} \ni X_{(1)} = Y_{(1)} \cdot F' \ .$$

Example:

$$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} a? & b? & c? \\ d & e & \\ f & & g \end{pmatrix} = \begin{pmatrix} a & b & c \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d? & e? & \\ f? & & g? \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} a & b+c \\ d & e \\ f & g \end{pmatrix}$$

The overhead cost of

- computation of P
- computation of $X^{(1)}$ and/or $Y_{(1)}$
- recovery of F' by direct substitution (unpacking)

must be (more than) amortized by savings in the cost of

$$\boldsymbol{R}^{m \times n^{(1)}} \ni Y^{(1)} = F' \cdot X^{(1)} \quad \text{and} \quad \boldsymbol{R}^{m_{(1)} \times n} \ni X_{(1)} = Y_{(1)} \cdot F' \ .$$

Outline

Summary

- Introduction to efficient accumulation of banded square Jacobians through exploitation of sparsity.
- Curtis-Powell-Reid seeding for finite differences and tangent/adjoint mode Algorithmic Differentiation.
- Direct Jacobian compression problems.

Next Steps

- Download and play with sample software.
- Continue the course to find out more ...