区分的微分可能関数の自動微分
Algorithmic Differentiation of Piecewise Differentiable Functions

情報工学専攻　藤澤 裕一
Yuuichi FUJISAWA

概要: 工学・経済モデルの感度解析など実際の問題を解く際に, しばしば偏導関数の値が必要な場合がある. 偏導関数の値を導出する手段の一つとして, 自動微分がある. 自動微分による解析の対象は, これまで微分可能演算のみであったが, 近年, 絶対値演算を含む関数のための自動微分の手法が再注目されている. この手法には, 一般化ヤコビアンと呼ばれる“ヤコビ行列を拡張した勾配情報”が用いられるが, 一般化ヤコビアンを導出するには適切な基底の組を用意する必要がある. しかし, 適切な基底の組を効率良く導出する方法は未だ解明されていない. 本研究では外部から基底を与えることにより一般化ヤコビアンの一部を求めるプログラムを実装した.

キーワード: 自動微分, 絶対値演算, プリプロセッサ, Java, ソースコードの変換

1　背景

工学・経済モデルの感度解析など実際の問題を解く際に, しばしば偏導関数の値が必要な場合がある. 偏導関数の値を導出する手段の一つとして, 自動微分がある. 自動微分による解析の対象は, これまで微分可能演算のみであったが, 近年, 絶対値演算を含む関数のための自動微分の手法が再注目されている. この手法には, 一般化ヤコビアンと呼ばれる“ヤコビ行列を拡張した勾配情報”が用いられるが, 一般化ヤコビアンを導出するには適切な基底の組を用意する必要がある. しかし, 適切な基底の組を効率良く導出する方法は未だ確立されていない.

2　目的

本研究の目的は, 関数の一般化ヤコビアンの一部（一般化ヤコビアン要素）を求める機能をプログラムに追加するための処理系を実装することである. この処理系は区分的微分可能関数を用いた非線形方程式や最適化を行う場合に利用できる.

3　自動微分

自動微分は, プログラムで定義されたアルゴリズムを解析し, 関数の偏導関数値を求めるアルゴリズムを導出する技術である [1]. そして, 導出されたアルゴリズムを実行することで, 関数の偏導関数値を求めることができる. その導出方法には大きく分けて 2 種類ある. それらは, ボトムアップ型自動微分，トップダウン型自動微分という.

4　区分的微分可能 (piecewise differentiable)

開集合 $X \subset \mathbb{R}^n$ が与えられた時, 関数 $\mathbf{f} : X \to \mathbb{R}^m$ が N 上で連続する $\mathbf{x}$ の開近傍 $N \subset X$ と, N を $\mathbb{R}^m$ に写像する C^1 級関数の有限集合 $\mathscr{F}_{\mathbf{f}}(\mathbf{x})$ が存在し, $\mathbf{f}(\mathbf{y}) \in \mathbf{f}^*(\mathbf{y}) : \mathbf{f}^* \in \mathscr{F}_{\mathbf{f}}(\mathbf{x})$ を各 $\mathbf{y} \in N$ で満たすとき, 関数 $\mathbf{f}$ は $\mathbf{x} \in X$ で区分的に微分可能 (PC) であるという.

関数 $\mathbf{f}^* \in \mathscr{F}_{\mathbf{f}}(\mathbf{x})$ は $\mathbf{f}$ の $\mathbf{x}$ のまわりでの“選択関数”と呼ばれる. 更に集合 $\mathscr{F}_{\mathbf{f}}(\mathbf{x})$ は $\mathbf{f}$ の $\mathbf{x}$ のまわりでの“選択関数の十分集合”と呼ばれる.

全て線形な $\mathbf{f}$ に選択関数の十分集合が存在するなら, $\mathbf{f}$ は区分的に線形 (PL) である.

5　微分不可能な点での勾配

これまでの自動微分では, 関数が絶対値演算により微分不可能となる点について, 特別な工夫がなされなかったため, 微分不可能である点の付近での関数の振る舞いを解析する事に不向きであった. そこで, 以下の小節で示すような特別な勾配が提案された.

また, 最大値関数, 最小値関数は絶対値演算を含む関数にそれぞれ書き換え可能である. 式 (1), (2). そのため, 絶対値演算を扱うことができれば, 最大値関数, 最小値関数も扱うことができる:

$$\max(x, y) = \frac{1}{2}(x + y + |x - y|), \tag{1}$$

$$\min(x, y) = \frac{1}{2}(x + y - |x - y|). \tag{2}$$

5.1　B-劣微分 (B-subdifferential)

関数 $\mathbf{f}(\mathbf{x})$ について, $\mathbf{f}(\mathbf{x})$ と $\mathbf{f}(\mathbf{y})$ の差が, $\mathbf{x}$ と $\mathbf{y}$ の差に比例するある量で抑えられるとき, $\mathbf{f}$ はリプシッツ連続であるという. リプシッツ連続な関数 $\mathbf{f}(\mathbf{x}) : \mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{m}}$ が $S \subset \mathbb{R}^n$ で微分不可であるとき, この関数の $\mathbf{x}$ における B-劣微分 (B-subdifferential) $\partial_{\mathbf{B}}\mathbf{f}(\mathbf{x})$ は以下で与えられる.

$$\partial_B \mathbf{f}(\mathbf{x}) := \{\mathbf{H} \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}} : \mathbf{H} = \lim_{\mathrm{i}\to\infty} \mathbf{J}\mathbf{f}(\mathbf{x}^{(\mathrm{i})})$$

$$\text{for some sequence} \{\mathbf{x}^{(\mathbf{i})}\}_{\mathbf{i}\in\mathbb{N}} \in \ \mathbf{X}\backslash\mathbf{S}$$

$$\text{such that } \lim_{i\to\infty} x^{(i)} = x\}.$$

5.2　一般化ヤコビアン (generalized Jacobian)

一般化ヤコビアン (generalized Jacobian) $\partial\mathbf{f}(\mathbf{x})$ は式 (3) で与えられる.

$$\partial\mathbf{f}(\mathbf{x}) = \partial_{\mathbf{B}}\mathbf{f}(\mathbf{x}) \text{ の凸包.} \tag{3}$$

最適化問題や方程式の求解のための一部アルゴリズムに, 関数の一般化ヤコビアンを必要とするものがある.

一般化ヤコビアンはクラーク一般化ヤコビアン (Clarke generalized Jacobian) とも呼ばれる.

6 一般化ヤコビアン要素の計算

6.1 絶対値分解表現 (abs-factor representation)

微分可能関数または絶対値演算からなる有限個の要素関数 $\varphi(1),\ldots,\varphi(\ell)$ に分解可能な関数を絶対値分解可能関数 (abs-factorable function) という.

開集合 $X\subset\mathbb{R}^n$, 絶対値分解可能関数 $f:X\to\mathbb{R}^m$ が与えられた時, $\mathbf{x}\in X$ における関数値 $\mathbf{f}(\mathbf{x})$ は次の手順で求めることができる.

絶対値分解可能関数の関数値を求める手順

Step 1
$\mathbf{v}_{(0)}=\mathbf{x}$, $j=1$ と代入する.

Step 2
$i\prec j$ となるような全ての $\mathbf{v}_{(i)}$ を i の昇順に列ベクトルとして並べたものを $\mathbf{u}_{(j)}\in X_{(j)}$ に代入する. また, $\mathbf{v}_{(j)}=\varphi_{(j)}(\mathbf{u}_{(j)})$ と代入する.

Step 3
$j=\ell$ ならば Step4 へ進む. そうでなければ, $j=j+1$ として Step 2 に戻る.

Step 4
$\mathbf{f}(\mathbf{x})=\mathbf{v}_{(\ell)}$ と代入する.

6.2 方向微分

関数 $\mathbf{f}(\mathbf{x}):\mathbb{R}^{\mathbf{n}}\to\mathbb{R}^{\mathbf{m}}$ の微分可能な点 $\mathbf{x}$ における $\mathbf{d}$ 方向の方向微分は, 関数 $\mathbf{f}$ の $\mathbf{x}$ におけるヤコビ行列を $\mathbf{Jf}(\mathbf{x})$ と表記するとき, 式 (4) で定義される.

$$\mathbf{f}'(\mathbf{x};\mathbf{d})=\mathbf{Jf}(\mathbf{x})\mathbf{d},\qquad\forall\mathbf{d}\in\mathbb{R}^{\mathbf{n}}\tag{4}$$

また, 絶対値関数 $\mathrm{abs}:\mathbb{R}\to\mathbb{R}:x\mapsto|x|$ の d 方向の方向微分は, 式 (5) で定義される.

$$\mathrm{abs}'(x;d)=\begin{cases}d & \text{if} x>0,\ \text{or if } x=0 \text{ and } d\geq 0,\\ -d & \text{if} x<0,\ \text{or if } x=0 \text{ and } d<0.\end{cases}\tag{5}$$

開集合 $X\subset\mathbb{R}^n$, 絶対値分解可能関数 $\mathbf{f}$ が与えられた時, $\mathbf{f}$ の $\mathbf{x}$ における $\mathbf{d}$ 方向の方向微分 $\dot{\mathbf{f}}(\mathbf{x};\mathbf{d})\in\mathbb{R}^m$ は次のの手順で計算できる. なお, この計算手順はボトムアップ型の自動微分となっている.

絶対値分解可能関数の方向微分を求める手順

Step 1
$\dot{\mathbf{v}}_{(0)}=\mathbf{d}$, $j=1$ と代入する.

Step 2
$i\prec j$ となるような全ての $\mathbf{v}_{(i)}$ を i の昇順に列ベクトルとして並べたものを $\dot{\mathbf{u}}_{(j)}\in\mathbb{R}^{n_j}$ に代入する. 方向微分 $\varphi'_{(j)}(\mathbf{u}_{(1)};\dot{\mathbf{u}}_{(1)})$ を $\varphi_{(j)}$ が C^1 級なら 式 (4), $\varphi_{(j)}$ が絶対値関数なら 式 (5) にしたがって求める. また, $\dot{\mathbf{v}}_{(j)}=\varphi_{(j)}(\dot{\mathbf{u}}_{(j)})$ と代入する.

Step 3
$j=\ell$ ならば Step4 へ進む. そうでなければ, $j=j+1$ として Step 2 に戻る.

Step 4
$\dot{\mathbf{f}}(\mathbf{x};d)=\dot{\mathbf{v}}_{(\ell)}$ と代入する.

6.3 Khan と Barton によるアルゴリズム

Khan と Barton により, 絶対値分解可能関数の一般化ヤコビアン要素を求めるアルゴリズムが提案された [2]. 以下にそのアルゴリズムを示す.

開集合 $X\subset\mathbb{R}^n$, 点 $\mathbf{x}\in X$, 絶対値分解可能関数 $\mathbf{f}:X\to\mathbb{R}^m$ が与えられた時, 一般化ヤコビアン要素 $\mathbb{B}\in\mathbb{R}^{m\times n}$ は以下の手順で与えられる.

1. 各 k に対して, $\mathbf{q}^{(k)}$ に n 次元実空間を構成する基底ベクトル $\mathbf{e}^{(k)}$ をセットする.
2. $\mathbf{f}(\mathbf{x})$ と全ての中間変数 $\mathbf{v}_{(j)}(\mathbf{x})$ を得るために $\mathbf{f}$ の絶対値分解表現を使う. 各 $j\in\{1,\ldots,\ell\}$ について, もし, $\varphi(j)=\mathrm{abs}$ かつ $\mathbf{u}_{(j)}(\mathbf{x})=0$ ならば, 論理型変数 IsCD(j) に false を代入する. そうでなければ, IsCD(j) に true を代入する. IsCD(j) = false, $\varphi(j)=\mathrm{abs}$ である各 k について, $\mathbf{u}_{(j)}$ と $\dot{\mathbf{u}}_{(j)}$ はスカラー値であるので, この場合 $u_{(j)}$, $\dot{u}_{(j)}$ のように記す.
3. 各 $k\in\{1,\ldots,n\}$ に対し, $\mathbf{f}'(\mathbf{x};\mathbf{q}^{(\mathbf{k})})$ を評価するためにボトムアップ型の自動微分を用いる. また, IsCD(j)=false である各 $j\in\{1,\ldots,\ell\}$ に対して, $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})$ の結果を格納する.
4. $j=1$ と代入し, 要素関数 $\varphi(1),\ldots,\varphi(\ell)$ を反復処理する以下の手続きを実行する:
   (a) IsCD(j)=true ならば, Step 4c へ進む.
   (b) $k=1$ と代入し, 以下の手続きを反復実行する.
   (i) $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})$ の値に従い, 以下のいずれかを実行する.
   - $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})\neq 0$ ならば 4(b)ii へ進む.
   - $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})=0$ かつ $k<n$ なら, $k=k+1$ として 4(b)i に戻る.
   - $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})=0$ かつ $k=n$ なら, 4(c) へ進む.

   (ii) $k^*=k$, $\dot{u}^*=\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k^*)})$ と, 現在の値を格納する.
   (iii) $k=n$ なら, 4(c) へ進む. そうでなければ $k=k+1$ とする. $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})$ の値に従い, 以下のいずれかを実行する.
   - $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})\dot{u}^*\geq 0$ ならば, ステップ 4(b)iii に戻る.

・ $\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})\dot{u}^* < 0$ ならば, $\mathbf{q}^{(k)}$ を式 (6) のように更新する.

$$\mathbf{q}^{(k)} = \mathbf{q}^{(k)} + \left|\frac{\dot{u}_{(j)}(\mathbf{x};\mathbf{q}^{(k)})}{\dot{u}^*}\right|\mathbf{q}^{(k^*)} \qquad (6)$$

ボトムアップ型の自動微分を用いて $\mathbf{f}'(\mathbf{x};\mathbf{q}^{(k)})$ を計算する. IsCD(i)=false である各 $i \in \{1,\ldots,\ell\}$ に対して, $\dot{u}_{(i)}(\mathbf{x};\mathbf{q}^{(k)})$ の結果を格納する. ステップ 4(b)iii に戻る.

(c) $j = l$ ならステップ 5 へ進む. そうでなければ $j = j+1$ としステップ 4(a) に戻る.

5. $\mathbb{B} \in \mathbb{R}^{m\times n}$ に関する次の線形方程式, 式 (7) を解く.

$$\mathbb{B}\left[\mathbf{q}^{(1)} \quad \cdots \quad \mathbf{q}^{(n)}\right] = \left[\mathbf{f}'(\mathbf{x};\mathbf{q}^{(1)}) \quad \cdots \quad \mathbf{f}'(\mathbf{x};\mathbf{q}^{(n)})\right] \qquad (7)$$

$\mathbb{B}$ を返し, 処理を終了する.

このアルゴリズムでは $\mathbb{B}$ を求めるために適切な基底を用意する必要がある.

本研究ではこのアルゴリズムに使用する基底を外部から与え, $\partial\mathbf{f}(\mathbf{x})$ を求めることを目指す.

7 処理系の設計と作成

7.1 処理系の構成

本研究では, 指定したソースコードに一般化ヤコビアン要素の計算手続きを追加するプリプロセッサタイプの処理系を実装した.

本処理系は Java の処理系であり, 図 1 の流れで処理を行うことで関数の一般化ヤコビアン要素を求める. プリプロセッサは本研究で作成した処理系であり, ソースコードを解析し, 指定された関数の一般化ヤコビアン要素の計算手続きを組み込む. 一般化ヤコビアン要素の計算には Khan と Barton によるアルゴリズムを用いている. プリプロセッサで処理された出力プログラムは Java コンパイラでコンパイルする. 本研究では, オラクルの配布する Java 開発環境 JDK に含まれる Java コンパイラ javac を使用した.

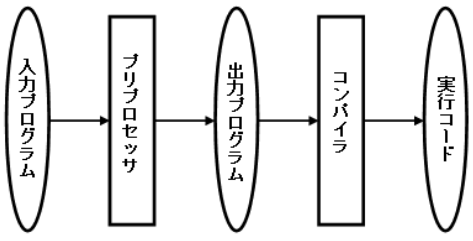

図 1 関数の一般化ヤコビアン要素を求めるフロー

7.2 入力プログラム

入力プログラムのサンプルをプログラム 1 に示す. このプログラムは関数 $z = \max(\min(x,-y), y-x)$ の点 $(x,y) = (0,0)$ における一般化ヤコビアン要素を求め, 表示することを目指すプログラムである.

29 行目の createMatrixB_z(x,y,q,matrixB); は, 関数 z の点 (x,y) における一般化ヤコビアン要素を基底 q を用いて求め, 配列 $matrixB$ に格納する記述である. createMatrixB_z() メソッドは入力プログラムには含まれていないが, プリプロセッサで処理することにより生成される.

関数 z は 10 行目で定義されている. 関数 z 記述できる演算は四則演算, 単項演算の他, Math.max(), Math.min(), Math.abs(), Math.sin() などが使用可能である.

微分対象となる関数名は z 以外に, f など自由な変数名が使用できるが, その場合には一般化ヤコビアン要素を格納する記述は createMatrixB_f() などとなる.

プログラム 1 Before.java

```
1  ...
2  public class Before{
3    public static void main(String[] av){
4
5      double x = 0;
6      double y = 0;
7      double z;
8
9      // 微分対象となる関数
10     z = Math.max(Math.min(x, -y), y-x);
11
12     // 微分対象となる関数の次元数
13     int n = 2;
14
15     // 基底を用意
16     double[][] q = new double[n][n];
17     q[0][0]=1;
18     q[0][1]=0;
19     q[1][0]=0;
20     q[1][1]=1;
21
22     // 一般化ヤコビアン要素 (matrix B)
23     // を格納する配列を用意
24     double[] matrixB = new double[n];
25
26
27     // 一般化ヤコビアン要素が
28     // matrixB に格納される
29     createMatrixB_z(x,y,q,matrixB);
30
31     System.out.println("matrixB="
32                   +Arrays.toString(matrixB));
33   }
34   ...
35 }
```

7.3 出力プログラム

プリプロセッサにプログラム 1 を入力したときの出力プログラムの一部をプログラム 2 に示す.

このプログラムにはプリプロセッサにより, createMatrixB_z(), culcDirectionalDerivative_z(), solveEqeuation() の 3 つのメソッドが追加されている.

プログラム 2 は長いので省略されているが, 実際のソースコードは 350 行である.

プログラム 2 After.java

```
1  ...
2  public class After{
3    public static void main(String[] av){
4      ...
5      System.out.println("matrixB="
6                    +Arrays.toString(matrixB));
7    }
8    static void createMatrixB_z(...){
9      ...
10   }
11   static void culcDirectionalDerivative_z(...){
12     ...
13   }
```

```
14    static boolean solveEqeuation(...){
15      ...
16    }
17    ...
18  }
```

7.4 実行結果

プログラム 2 をコンパイルし, 実行すると以下のように一般化ヤコビアン要素が出力される.

```
matrixB=[0.0, -1.0]
```

シェル上で, 入力プログラムに対してプリプロセッサによる処理を実行するとき, コマンドライン引数に実行時オプションが指定できる. 例として, -tp とコマンドライン引数にオプションとして指定することで, 計算の途中経過を出力するようになる. このオプションを指定したときの出力結果は以下のようになり, 基底 $q_{(2)}$ が更新されていることが確認できる.

```
q1=[1.0, 0.0]
dvalue=0.0
q2=[0.0, 1.0]
dvalue=1.0
q2=[2.0, 1.0]
dvalue=-1.0
matrixB=[0.0, -1.0]
```

また, 基底を $q^{(1)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} q^{(2)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ とすると

```
matrixB=[-1.0, 1.0]
```

という結果が得られる.

また, 基底を $q^{(1)} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} q^{(2)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ とすると

```
matrixB=[1.0, 0.0]
```

という結果が得られる.

実際, 関数 $f(x, y) = \max(\min(x, -y), y - x)$ の選択関数は $f_{(1)} = x$, $f_{(2)} = -y$, $f_{(3)} = y - x$ の 3 つであり, それぞれの選択関数が実質的に有効な範囲は図 2 に示すような領域に分かれている. 図中の赤色の斜線範囲は, $f_{(1)} = x$, 青色の斜線範囲は, $f_{(2)} = -y$, 緑色の斜線範囲は, $f_{(3)} = y - x$ が実質的に有効な選択関数となる領域を示している. また, 青色の太線は, それぞれの領域の境界を示しており, 通常微分不可能な部分である. ここで, $(x, y) = (0, 0)$ の点においては, $f_{(1)}$, $f_{(2)}$, $f_{(3)}$ の 3 つの選択関数の有効範囲が重なっていることが図より確かめられる. よって, この点におけるる B-劣微分は, $\begin{bmatrix} 1 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & -1 \end{bmatrix}$, $\begin{bmatrix} -1 & 1 \end{bmatrix}$ の 3 つであり, 正しい解が得られていることがわかる. そのため, 新たに基底を取り直しても, これ以外の一般化ヤコビアン要素は得られない.

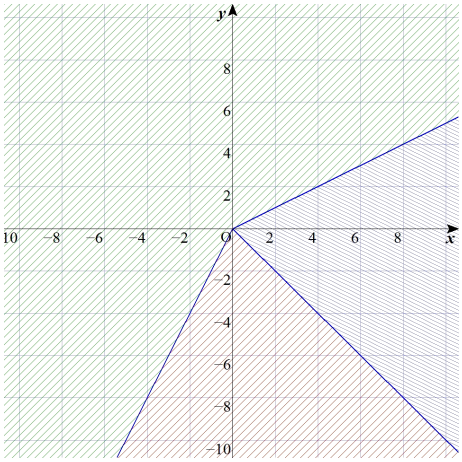

図 2　関数 f の実質的に有効な関数の領域の分割

8　結論

本研究では, 関数の一般化ヤコビアン要素を求める機能をプログラムに追加するための処理系を実装した. この処理系は関数の一般化ヤコビアンを求めるために必要な基底の組を効率良く求める研究に利用することができる.

9　今後の課題

本研究で作成したプリプロセッサは, 現段階では Java の一部の文法しか受理しないため, すべての文法の受理を可能にする必要がある.

本研究で実装した処理系を使用し, 区分的微分可能関数の一般化ヤコビアンを利用した最適化問題に応用していく.

参考文献

[1] 久保田光一, 伊理正夫, **“アルゴリズムの自動微分と応用”**, コロナ社, 1998.

[2] Kamil A. Khan, and Paul I. Barton, **“Evaluating an Element of the Clarke Generalized Jacobian of a Piecewise Differentiable Function”**, In Recent Advances in Algorithmic Differentiation. S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds., Springer, 115-125, 2012.

[3] Daniel Ralph, and Stefan Scholtes, **“Sensitivity analysis of composite piecewise smooth equations”**, Mathematical Programming 76, 1997.

[4] Stefan Scholtes, **“Introduction to piecewise differentiable equations”**, Habilitation Thesis, Institut fur Statistik und Mathematische Wirtschaftstheorie, University of Karlsruhe, 1994.

[5] Andreas Griewank, **“On stable piecewise linearization and generalized algorithmic differentiation”**, Optimization Methods & Software, 2013, http://dx.doi.org/10.1080/10556788.2013.796683.

[6] Kamil A. Khan, and Paul I. Barton, **“Evaluating an Element of the Clarke Generalized Jacobian of a Composite Piecewise Differentiable Function”**, ACM Transactions on Mathematical Software, Vol.39, No.4, Article 23, July 2013.

[7] 今城哲二, 布広永示, 岩澤京子, 千葉雄司, **“コンパイラとバーチャルマシン”**, オーム社, 2004.