高速自動微分用記憶領域削減法

片岡幹雄　久保田光一
中央大学大学院理工学研究科*

1 序論

機械設計，システム設計などにおいて，数学モデルに現れる関数の導関数値が必要となることがある．このような導関数値を求める手法として自動微分がある．自動微分は，従来の数値微分や数式処理に比べ丸め誤差の推定や複雑な関数への適用等で有効である．また，自動微分の一種である高速自動微分は，スカラー値の多変数関数の勾配，すなわち，全ての変数に関する偏導関数値を，関数値だけを計算するのに必要な計算量のたかだか定数倍の計算量で一度に計算できる．しかし，高速自動微分は，大規模な計算に対して大きな記憶領域が必要となる．このため，本研究では，高速自動微分法の記憶領域削減法として，Griewank らによる高速自動微分の記憶領域削減法を扱う．そして，舟崎智誠氏 [3, 4] による記憶領域削減法の実装を使い，計算機実験からこの記憶領域削減法の有効性を確かめた．なお本稿では，この記憶領域削減法の実装概要について報告する．

2 高速自動微分

高速自動微分における偏導関数値計算の原理は，合成関数の微分である．関数を初等演算等の基本演算に分割して，関数全体を基本演算の合成関数として扱い合成関数の微分則を適用する．高速自動微分は，与えられた関数のアルゴリズムをいったん実行して計算グラフを構成した後に，計算の履歴を逆にたどって偏導関数値を計算する．なお，本研究では高速自動微分の実行に C++言語の演算子多重定義機能を用いる．

3 Griewank の記憶領域削減法

Griewank らによる高速自動微分の記憶領域法では，プログラムの凍結とプログラムの解凍と言う技法を使う．プログラムの凍結とはプログラムの実行状態を保存することである．また，プログラムの解凍とは凍結を実行した直後の状態から再びプログラムを実行することである．高速自動微分を適用するベクトル値関数を基本演算 $f : S \equiv \{0,1\}^R \to S$ の列を

$$s_{i+1} \leftarrow f_i(s_i) \quad \text{for} \quad i = 0, 1, \ldots, n-1 \tag{1}$$

とする．また，この計算過程に対する偏導関数値の計算を $\bar{f} : \bar{S} \to \bar{S}$ の列を

$$\bar{s}_{i-1} \leftarrow \bar{f}_i(\bar{s}_i) \quad \text{for} \quad i = n-1, \ldots, 1, 0 \tag{2}$$

とする．ここで，$\bar{S}$ は S とほぼ同じ大きさの偏導関数保存領域である．$\bar{s}_i$ に $\bar{f}_i$ を適用して $\bar{s}_{i-1}$ を求めるとする．この時，$\bar{s}_i$ に $\bar{f}_i$ を適用して $\bar{s}_{i-1}$ を求める際に必要な情報を保存することを計算履歴を保存すると言う．計算履歴を保存することを明確にするために $s_{i+1} \leftarrow f_i(s_i)$ の代わりに $s_{i+1} \leftarrow \hat{f}_i(s_i)$ を使うことにする．また，簡単のため以降は $f_i(s_i)$ を f_{s_i} と，$\bar{f}_i(s_i)$ を $\bar{f}_{s_i}$ と，$\hat{f}_i(s_i)$ を $\hat{f}_{s_i}$ と書くことにする．通常，関数に高速自動微分を適用する全ての s_i に対して $\hat{f}_{s_i}$ を呼び出して計算履歴を保存しなければならない．しかし，Griewank の記憶領域削減法では，凍結と解凍の技法によって記憶領域を削減する．具体的には，計算過程を p 個の部分計算過程に分割する．

$$F_j \equiv [f_{i_j}, f_{i_j+1}, \ldots, f_{i_{j+1}-1}] \ \text{for} \ j = 0, \ldots, p-1 \tag{3}$$

この時，各部分計算過程 $F_j (j = 0, \cdots, p-1)$ における全ての基本演算に対する計算履歴の保存に必要な記憶領域の合計を考える．この記憶領域の合計がなるべく等しくなるように計算過程を分割する．計算履歴を保存せずに関数値計算を実行していく過程で状態 $s_{i_j} (j = 0, \cdots, p-1)$ について凍結をとる．偏導関数値計算の最初のステップ $\bar{f}_{s_{n-1}}$ を実行する時は $s_{i_{p-1}}$ における凍結を解凍して部分計算過程 F_{p-1} に関して計算履歴の保存をともなう関数値計算 $\hat{f}_{s_i}$ を実行する．この計算履歴を使い F_{p-1} に関する偏導関数値計算 $\bar{f}_{s_i}$ が実行できる．残りの部分計算過程に関しても同様に凍結の解凍，計算履歴の保存をともなう関数値計算，偏導関数値計算という手順で計算を行う．こうすることで計算過程全体の偏導関数値を求めることができる．このとき，それぞれの部分計算過程の計算履歴を保存するための記憶領域は毎回同じ領域を使うことができるので記憶領域の削減につながる．この様な手順を再帰的に繰り返すのが Griewank の記憶領域削減法で図 1 のように定義される．ここで，$s \leftarrow F(s)$, $s \leftarrow \hat{F}(s)$, $\bar{s} \leftarrow \bar{F}(\bar{s})$ はそれぞれ部分

*Storage reduction method for reverse-mode automatic differentiation, Mikio KATAOKA and Koichi KUBOTA, Graduate School of Science and Engineering, Chuo-University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan.

```
treeverse(δ, τ, β, σ, φ){
    if(σ > β) {
        δ = δ − 1;
        snapshot(s);
        s ← F_j(s)   for   j = β, ··· , σ − 1;
    }
    while(κ = mid(δ, τ, σ, φ) < φ) {
        treeverse(δ, τ, σ, κ, φ)
        τ = τ − 1;
        φ = κ;
    }
    if(φ − σ) > 1    exit ("treeverse fails")
    s ← F̂_σ(s);
    s̄ ← F̄_σ(s̄);
    if σ > β    retrieve(s);
    return
}
```

図 1: Griewank の記憶領域削減法

計算過程 F における関数値計算, 計算履歴の保存をともなう関数値計算, 偏導関数値計算を表す．図 1 において **snapshot** (s) は状態 s を凍結する働きをする．また，**retrieve**(s) は凍結された状態 s を解凍する働きをする．**treeverse** の引数 δ, τ はそれぞれ凍結の最大数と再計算レベルを表す．再計算レベルとは計算履歴をともなわない関数値計算をひとつの部分計算過程について何回まで許すかという値である．σ, ϕ はその **treeverse** の呼び出しで偏導関数値計算をする範囲を示している．**mid** は σ, ϕ の間の適当な数を返す関数である．与えられた δ, τ に関して領域計算量, 時間計算量を最適化する **mid** は,

$$\mathbf{mid}(\delta, \tau, \sigma, \phi) \equiv \left\lceil \frac{\delta \cdot \sigma + \tau \cdot \phi}{\tau + \sigma} \right\rceil \qquad (4)$$

となる．

4 記憶領域削減法の実装

Griewank の記憶領域削減法を実装するためには凍結，解凍の機能が必要となる．本研究では，UNIX のシステムコール fork, Pthreads とセマフォを用いて凍結の機能を実現する．なお，図 2, `sem_wait` はセマフォを使ってプロセス/スレッドを凍結する関数であり，`sem_singnal` は凍結されたプロセス/スレッドを解凍する関数である．

UNIX のシステムコール fork は新しいプロセスを作成する．作成されたプロセスは，元のプロセスのコピーである．したがって，このプロセスをセマフォを用いてスリープさせれば凍結を取ることができる．凍結を解凍するときは，他のプロセスからセマフォを用いてスリープさせたプロセスを起こせばよい．fork とセマフォを用いた凍結の概念を図 2 に示す．

また，スレッドを用いて凍結の機能を実現する場合も基本的には，同様である．ただし，スレッドは大域変数を共有しているので大域変数の扱いを工夫しなければならない．fork, スレッド, セマフォを使用した凍結, 解凍の機能は図 1 の **snapshot**, **retrieve** と全く同じ働きをするわけではない．**snapshot**, **retrieve** は，凍結を状態として保存しておき，解凍も凍結を取ったプロセスから実行することができる．しかし，fork, スレッド, セマフォを使用した凍結とはプロセスをスリープさせることであり，解凍には他のプロセスやスレッドからの信号が必要となる．このような違いから Griewank の記憶領域削減法を計算を制御するサーバプロセスと計算を実行するクライアントプロセスからなるクライアント・サーバ方式で実装する．

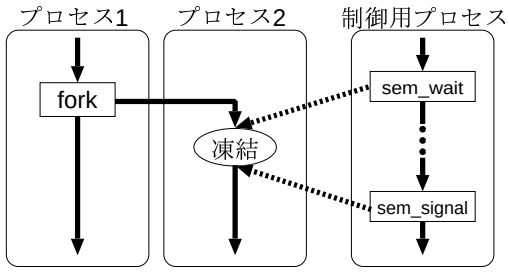

図 2: fork を用いた凍結, 解凍の概念

5 結論

本研究では，Griewank らによる高速自動微分の記憶領域削減法に関する舟崎智誠氏 [3, 4] の実装を評価した．そして，計算機実験を大規模な関数に対して行った．計算機実験から，Griewank らによる記憶領域削減法とその記憶領域削減法の実装が，計算時間の増加を抑えながら記憶領域を削減することを確認した．

参考文献

[1] A. Griewank, and A. Walther, “Revolve: An Implementation of Checkpointing for the Reverese or Adjoint Mode of Comuputional Differentiation,” *ACM Transactions on Mathematical Software*, Vol. 26, No. 1, pp. 19–45 March 2000.

[2] 久保田光一, 伊理正夫, **アルゴリズムの自動微分と応用**, 現代非線型科学シリーズ, コロナ社, 東京, 1998.

[3] 舟崎智誠, 久保田光一, “高速自動微分のためのプロセス制御” , **情報処理学会第 62 回全国大会講演論文集**, 1, pp. 51–52, 2001.

[4] 舟崎智誠, “高速自動微分のためのプロセス制御”, **中央大学大学院理工学研究科情報工学専攻修士論文**, 2002.