数理解析研究所講究録
第648巻 1988年 39-58

スプライン関数を用いた多次元データの補間
— ベクトル計算機向きの計算技術 —

明石高専　吉本富士市　(Fujiichi Yoshimoto)

1. はじめに

ベクトル計算機は，ベクトルプロセッサを用いて並列計算することにより演算の高速化を図った計算機であるが，個性の強い計算機であり，計算機と算法との相性およびプログラムの作り方によって数学ソフトウェア（プログラム）の性能に大きな差ができる．

したがって，“ベクトル計算機の目”から見たときの従来の算法の再評価と改良，新算法の提案，ベクトル計算機向きのプログラミング技術，などの研究が必要である．このうち，ベクトル計算機向きのプログラミング技術の問題は，コンパイラの自動ベクトル化技術の進歩により，かなり解決されてきた．しかし，まだ十分ではなく，プログラマの助けを必要とする場合も多い．

“ベクトル計算機の目”から見たときの従来の算法の再評価と改良，新算法の提案については，これまで主として連立一次方程式および固有値問題の数値解法，高速フーリエ変換の分野で行われている．これらの研究により，従来のスカラ計算機向きの算法は，そのままでは必ずしもベクトル計算機向きではなく，何らかの改良または新算法を考えなければならないことも多いことが分かってきている．その他の分野でも計算を速くしたい要求は強く，たとえば，最適化，積分，関数近似などの問題を挙げることができる．したがって，これらの問題でも同様な研究を行い，ベクトル計算機上での数値計算に関する知識を蓄積するとともに，ベクトル計算機向きの数学ソフトウェアを開発することは，有意義なことと思われる．

本論文では，スプライン関数を用いた多次元格子点データ（多次元空間内に直角格子を取り，そのすべての交点上で与えられたデータ）の補間に

ついて，ベクトル計算機に適した算法およびプログラミング技術を述べる．多次元格子点データの補間法については，すでにいくつかの研究がある．まず，ＰｅｒｅｙｒａとＳｃｈｅｒｅｒ[1]は，テンソル積を高速に計算する方法を提案し，その応用として多項式による多次元補間を述べている．その後，Ｈａｒｔｌｅｙ[2]，秦野と二宮[3]，ｄｅ　Ｂｏｏｒ[4]などにより，テンソル積型のスプライン関数を用いた多次元格子点データの補間法が研究されている．しかし，ベクトル計算機への適合性について調べた研究は，まだないようである．ここでは，テンソル積型のスプライン関数を用いた多次元格子点データの補間法について，ベクトル計算機との適合性を調べ，ベクトル計算機向きにするための方法およびプログラミング技術を報告する．

2．多変数スプライン関数を用いた補間

2．1　問題の設定

いま，ｎ次元ユークリッド空間の超直方体領域を

$$R = \prod_{i=1}^{n} [a^{(i)}, b^{(i)}] \qquad (1)$$

とし，Ｒ内の格子点で与えられるデータを

$$(x^{(1)}{}_{k_1}, \ldots, x^{(n)}{}_{k_n}; f_{k_1 \ldots k_n})$$

$$(k_i = 1, \ldots, M_i; i = 1, \ldots, n) \qquad (2)$$

と書くことにする．ここで，$f_{k_1 \ldots k_n}$ は格子点 $(x^{(1)}{}_{k_1}, \ldots, x^{(n)}{}_{k_n})$ の上の関数値を，M_iは座標軸番号ｉ方向のデータ点の数を表している．また，（２）の格子の間隔は任意に取ることができる．

各座標軸方向に対して，両端の節点はm_i個ずつ重ね，

$$\left.\begin{aligned} \xi^{(i)}{}_{1-m_i} = \xi^{(i)}{}_{2-m_i} = \cdots = \xi^{(i)}{}_{0} = a^{(i)} \\ \xi^{(i)}{}_{h_i-m_i+1} = \xi^{(i)}{}_{h_i-m_i+2} = \cdots = \xi^{(i)}{}_{h_i} = b^{(i)} \end{aligned}\right\}$$

（ｉ＝１，２，．．．，ｎ）　　（３）

とする．ここで，m_iおよびh_iは座標軸番号ｉ方向のＢ－スプラインの階数（次数＋１）およびＢ－スプラインの数をそれぞれ表している．内部の節点はＳｃｈｏｅｎｂｅｒｇ－Ｗｈｉｔｎｅｙの条件[5]が成り立つ範囲内で任意にとることができる．ただし，簡単のため，ここでは

$$\xi^{(i)}{}_{k_i} = x^{(i)}{}_{k_i + [m_i/2]}$$

（k_i＝１，２，．．．，$h_i - m_i$；ｉ＝１，２，．．．，ｎ）　（４）

とする．このとき，端条件は与える必要がなくなる．

式（３），（４）で与えられる節点に対して作られるＢ－スプラインを

$$N_{m_i, j_i}(x^{(i)})$$

（j_i＝１，２，．．．，h_i；ｉ＝１，２，．．．，ｎ）

と書くことにする．ここで，j_iは座標軸番号ｉ方向のＢ－スプラインの番号を表している．このとき，ｎ変数のスプライン関数は，各Ｂ－スプラインのテンソル積により

$$S(x^{(1)}, \ldots, x^{(n)}) = \sum_{j_1=1}^{h_1} \cdots \sum_{j_n=1}^{h_n} c_{j_1 \cdots j_n} N_{m_1, j_1}(x^{(1)}) \cdots N_{m_n, j_n}(x^{(n)}) \qquad (5)$$

と表すことができる．近似関数（５）が与えられたデータ（２）を通る条件（補間条件）により，補間係数$c_{j_1 \cdots j_n}$を決定すると，任意の点で補間値を計算できる．

２．２　補間係数の計算

補間条件は

$$S(x^{(1)}{}_{k_1}, \ldots, x^{(n)}{}_{k_n}) = f_{k_1 \ldots k_n} \qquad (6)$$

と書くことができる．したがって，

$$\sum_{j_1=1}^{h_1} \cdots \sum_{j_n=1}^{h_n} c_{j_1 \cdots j_n} N_{m_1, j_1}(x^{(1)}{}_{k_1}) \cdots N_{m_n, j_n}(x^{(n)}{}_{k_n}) = f_{k_1 \ldots k_n}$$

$$(k_i = 1, \ldots, M_i;\ i = 1, \ldots, n) \qquad (7)$$

となる．ここで，一意的に補間できるためには，データの数と近似関数のパラメタの数が一致しなければならないので，$h_1 \equiv M_1$，$h_2 \equiv M_2$，・・・，$h_n \equiv M_n$ であることに注意したい．

式（７）は，次のように入れ子の形に書くことができる．

$$\sum_{j_n=1}^{h_n} N_{m_n, j_n}(x^{(n)}{}_{k_n}) \left[\cdots \left[\sum_{j_1=1}^{h_1} N_{m_1, j_1}(x^{(1)}{}_{k_1}) \right. \right. \left. \left. c_{j_1 \cdots j_n} \right] \cdots \right] = f_{k_1 \ldots k_n}$$

$$(k_i = 1, \ldots, M_i;\ i = 1, \ldots, n) \qquad (8)$$

したがって，変数分離により

$$\sum_{j_n=1}^{h_n} N_{m_n, j_n}(x^{(n)}{}_{k_n})\ c^{(n)}{}_{k_1 \cdots k_{n-1} j_n} = f_{k_1 \ldots k_{n-1} k_n}$$

$$(k_i = 1, 2, \ldots, M_i;\ i = 1, 2, \ldots, n)$$

$$(j_n = 1, 2, \ldots, h_n) \qquad (9)$$

$$\sum_{j_{n-1}=1}^{h_{n-1}} N_{m_{n-1}, j_{n-1}}(x^{(n-1)}{}_{k_{n-1}})\ c^{(n-1)}{}_{j_n k_1 \cdots j_{n-1}} = c^{(n)}{}_{j_n k_1 \cdots k_{n-1}}$$

$$(k_i = 1, 2, \ldots, M_i;\ i = 1, 2, \ldots, n-1)$$

$$(j_i = 1, 2, \ldots, h_i;\ i = n-1, n) \qquad (10)$$

$$\cdots\cdots\cdots\cdots$$

$$\sum_{j_1=1}^{h_1} N_{m_1,j_1}(x^{(1)}{}_{k_1})\, c^{(1)}{}_{j_2\cdots j_n j_1} = c^{(2)}{}_{j_2\cdots j_n k_1}$$

$$(k_1 = 1, 2, \ldots, M_1)$$

$$(j_i = 1, 2, \ldots, h_i;\ i = 1, 2, \ldots, n) \qquad (11)$$

を順に解けばよい．ここで，cの右肩に付けたカッコ書きの数字は，（9）から（11）までの式の解の順序（各座標軸方向に相当する）を表している．また，（11）の解 $c^{(1)}{}_{j_2\cdots j_n j_1}$ の添字を回転してできる $c^{(1)}{}_{j_1 j_2\cdots j_n}$ が，求める補間係数 $c_{j_1 j_2\cdots j_n}$ である．

式（9）～（11）は，それぞれ同じ係数行列をもつ複数組の連立一次方程式である．したがって，係数行列を一度ガウス消去しておくと，定数項の処理および後退代入の計算だけで解くことができる．こららの連立一次方程式は条件がよいので，枢軸選択を行う必要はない[6),7)]．なお，（9）の解 $c^{(n)}{}_{k_1\cdots k_{n-1}j_n}$ を（10）の右辺へ渡すとき，添字を回転して $c^{(n)}{}_{j_n k_1\cdots k_{n-1}}$ としていることに注意してほしい．以下（11）まで同様に，添字を順に回転する．これは，同じプログラムでつぎつぎと計算するために必要な操作である．

2．3　補間値の計算

補間値を求めたい点が，領域R内の任意の格子のすべての交点であるときには，上で述べた補間係数 $c_{j_1\cdots j_n}$ の計算と同様な考え方で，高速に補間値を計算できる．いま，補間値を求めたい点を $(x^{(1)}{}_{r_1}, \ldots, x^{(n)}{}_{r_n})$ とし，その点の上の補間値を $S_{r_1\cdots r_n}$ と書くことにする．ここで，$r_i = 1, 2, \ldots, L_i$（$i = 1, 2, \ldots, n$）である．

式（5）は

$$S(x^{(1)}, \ldots, x^{(n)}) =$$

$$\sum_{j_1=1}^{h_1} N_{m_1,j_1}(x^{(1)})\left[\cdots\left[\sum_{j_n=1}^{h_n} N_{m_n,j_n}(x^{(n)})\, c_{j_1\cdots j_n}\right]\cdots\right] \qquad (12)$$

のように入れ子の形に書くことができる．したがって，次の式を順に計算すればよい．

$$\sum_{j_n=1}^{h_n} N_{m_n,j_n}(x^{(n)}{}_{r_n})\, c_{j_1\cdots j_{n-1}j_n} = c^{(n)}{}_{j_1\cdots j_{n-1}r_n}$$

$$(j_i = 1, 2, \ldots, h_i;\ i = 1, 2, \ldots, n)$$

$$(r_n = 1, 2, \ldots, L_n) \qquad (13)$$

$$\sum_{j_{n-1}=1}^{h_{n-1}} N_{m_{n-1},j_{n-1}}(x^{(n-1)}{}_{r_{n-1}})\, c^{(n)}{}_{r_n j_1\cdots j_{n-1}} = c^{(n-1)}{}_{r_n j_1\cdots r_{n-1}}$$

$$(j_i = 1, 2, \ldots, h_i;\ i = 1, 2, \ldots, n-1)$$

$$(r_i = 1, 2, \ldots, L_i;\ i = n-1, n) \qquad (14)$$

$$\cdots\cdots\cdots\cdots$$

$$\sum_{j_1=1}^{h_1} N_{m_1,j_1}(x^{(1)})\, c^{(2)}{}_{r_2\cdots r_n j_1} = c^{(1)}{}_{r_2\cdots r_n r_1}$$

$$(j_1 = 1, 2, \ldots, h_1)$$

$$(r_i = 1, 2, \ldots, L_i;\ i = 1, 2, \ldots, n) \qquad (15)$$

ここで，（１５）の右辺 $c^{(1)}{}_{r_2\cdots r_n r_1}$ の添字を回転してできる $c^{(1)}{}_{r_1 r_2\cdots r_n}$ が，求める補間値 $S_{r_1 r_2\cdots r_n}$ である．また，ｃの右肩に付けたカッコ書きの数字は，（１３）から（１５）までの右辺の順序（各座標軸方向に相当する）を表している．なお，（１３）の右辺 $c^{(n)}{}_{j_1\cdots j_{n-1}r_n}$ を（１４）の左辺へ渡すとき，添字を回転して $c^{(n)}{}_{r_n j_1\cdots j_{n-1}}$ としていることに注意したい．以下（１５）まで同様に，添字を順に回転する．これは，

同じプログラムでつぎつぎと計算するために必要な操作である.

2．4　アルゴリズム

以上，２．２節および２．３節で述べたことをアルゴリズムとしてまとめると，図１のようになる．変数分離により，もとのｎ次元補間問題はｎ個の小さな問題に分解されるので，（１）補間係数を求めるための計算量

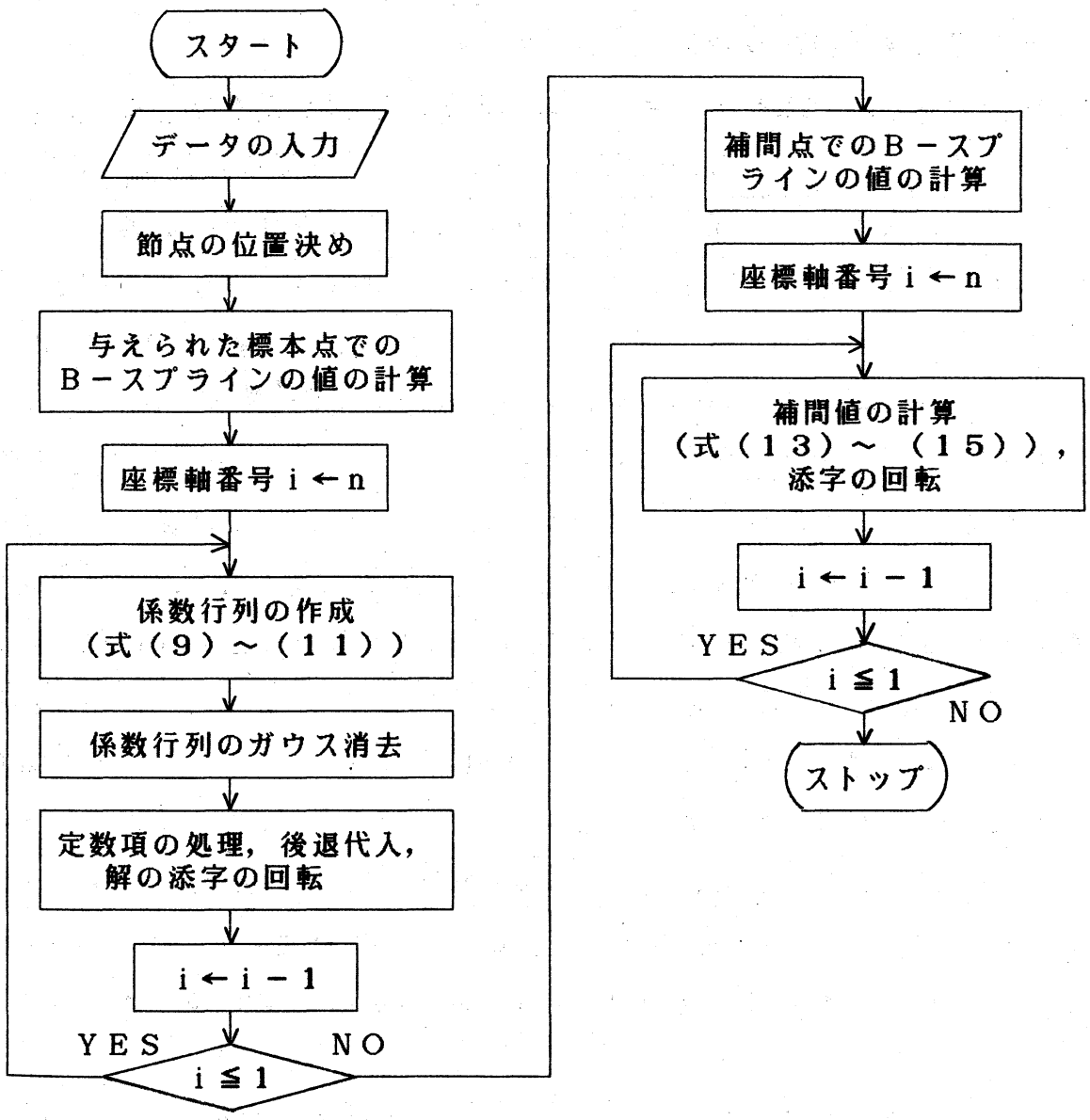

図１　テンソル積型のスプライン関数を用いた多次元格子点データの補間のアルゴリズム

はデータの次元数nに対して比例的に増加するだけである，（2）大次元の連立一次方程式を解かなくてよいので，必要な記憶容量が少ない，（3）補間値の計算量もデータの次元数nに対して比例的に増加するだけである，などの利点がある．

3．ベクトル計算機向きの計算技術

3．1　補間係数の計算技術

まず，（9）の計算を考えよう．式（9）は，同じ係数行列をもつ$M_1 \times \cdots \times M_{n-1}$組の連立一次方程式であり，この各方程式は独立に解くことができる．したがって，係数行列をあらかじめ一度だけガウス消去しておくと，定数項の処理および後退代入の部分を並列計算できる．このとき，大きさ（M_1，M_2，．．．，M_n）のn 次元行列 f および大きさ（M_1，M_2，．．．，h_n）の n次元行列 c を，それぞれ大きさ（$M_1 \times \cdots \times M_{n-1}$，$M_n$）および大きさ（$M_1 \times \cdots \times M_{n-1}$，$h_n$）の２次元行列と見なして計算すると，ベクトル長が$M_1 \times \cdots \times M_{n-1}$の連続ベクトルになる．したがって，高次元になるほどベクトル長が長くなり，ベクトル計算機によく適合する性質をもつ．ただし，あらかじめ大きさ（M_1，M_2，．．．，M_n）のn次元配列 Fを用意して，データ$f_{k_1 \ldots k_n}$がF の $k_1 + M_1(k_2 - 1 + M_2(k_3 - 1 + \cdots + M_{n-1}(k_n - 1)\cdots))$番目の要素となるように入れておく．ここで，$k_i = 1, \ldots, M_i$（$i = 1, \ldots, n$）である．また，c を入れる配列の大きさを（$M_1 \times \cdots \times M_{n-1}$，$h_n$）にしておく必要がある．このように考えると，リストベクトルを用いなくても高次元行列を低次元行列に落として取り扱うことができる．

つぎに，（9）の解$c^{(n)}{}_{k_1 \cdots k_{n-1} j_n}$を，最後の添字が先頭にくるように回転して（10）の右辺へ与える．この計算も長いベクトル長で行うことができ，$M_1 \times \cdots \times M_{n-1}$となる．しかし，回転してストアするところ

だけは連続ベクトルにはならず，h_n 間隔のベクトルとなる．したがって，たとえば倍精度計算のとき（ベクトル計算機では，通常は倍精度計算を行う）h_n が偶数（とくに２のべき乗）であればメモリ衝突を起す可能性がある．ただし，この部分の計算量は他の部分の計算量に比べると少ないので，大きな欠陥ではない．しかし，加速率をかなり低下させることも事実である（４章の計算例参照）．このメモリ衝突を回避するためには，あらかじめ与えるデータの量を調節して，h_n が偶数にならないようにしておくとよい．

ところで，メモリ衝突を避けるためによく用いられる手法として，倍精度計算のときには配列の宣言で第１添字の大きさを奇数にしておく方法がある[8]．しかし，ここではその手法は適用できない．その理由は，実際のデータ量と配列の大きさを一致させなければならないからである．もしも余分な記憶領域をもつと，添字を回転したときアドレスの対応がずれてしまい，正しく計算されない．

式（１０）は，同じ係数行列をもつ$h_n \times M_1 \times \cdots \times M_{n-2}$組の連立一次方程式である．したがって，この各方程式は独立に解くことができるので，（９）と同様な並列計算が可能である．このとき，（１０）のｎ次元行列ｃを２次元行列と見なして計算すると，ベクトル長は $h_n \times M_1 \times \cdots \times M_{n-2}$となる．したがって，高次元になるほどベクトル長が長くなり，ベクトル計算機によく適合する．

以下同様にして，（１１）まで計算する．結局，（９）から（１１）までの計算で添字の回転に伴うメモリ衝突を回避するためには，h_n，h_{n-1}，・・・，h_1のすべてが奇数であればよいことになる．ここで，$h_n \equiv M_n$，$h_{n-1} \equiv M_{n-1}$，・・・，$h_1 \equiv M_1$であることに注意したい．

なお，上記の計算法は（９）～（１１）を計算する部分をサブルーチンとしておき，それを呼ぶたびに配列の大きさを渡す方法により，一般性を損なうことなく実現可能である．また，Ｂ－スプラインの値の計算には，ベクトル計算機向きの算法[9]を適用できる．しかし，格子点データの場合

にはM₁＋M₂＋・・・＋Mₙ個の点で計算するだけでよいので，Ｂ－スプラインの計算量はあまり多くなく，その効果は少ない（４章の計算例参照）．

３．２　補間値の計算技術

まず，（１３）の計算を考えよう．式（１３）は，大きさ（Lₙ，hₙ）の２次元行列と大きさ（h₁，・・・，hₙ）のｎ次元行列ｃの積であり，並列計算が可能である．このとき，ｎ次元行列ｃを大きさ（h₁×・・・×hₙ₋₁，hₙ）の２次元行列と見なして計算すると，ベクトル長がh₁×・・・×hₙ₋₁の連続ベクトルになる．したがって，高次元になるほどベクトル計算機によく適合する．ただしｃは，３．１節で述べたように計算して，大きさ（h₁×・・・×hₙ₋₁，hₙ）の配列に入れているものとする．また，c⁽ⁿ⁾を入れる配列の大きさを（h₁×・・・×hₙ₋₁，Lₙ）にしておく必要がある．このように考えると，リストベクトルを用いなくても高次元行列を低次元行列に落として取り扱うことができる．

つぎに，（１３）の計算結果$c^{(n)}{}_{j_1\cdots j_{n-1}r_n}$を，最後の添字が先頭にくるように回転して（１４）の左辺へ与える．この計算もベクトル長を長くすることができ，h₁×・・・×hₙ₋₁となる．しかし，回転してストアするところだけは連続ベクトルにはならず，Lₙ 間隔のベクトルとなる．したがって，たとえば倍精度計算のとき，Lₙ が偶数であればメモリ衝突を起す可能性がある．ただし，この部分の計算量は他の部分の計算量に比べると少ないので，大きな欠陥ではない．しかし，加速率をかなり低下させることも事実である（４章の計算例参照）．このメモリ衝突の問題を解決するためには，あらかじめ補間値を計算すべき点の量を調節しておき，Lₙが偶数にならないようにしておくとよい．

なお，ここでもメモリ衝突を避けるため配列の宣言で第１添字の大きさを奇数にしておく手法は適用できない．その理由は，データの量よりも配列が大きいと，添字の回転でアドレスの対応がずれてしまうからである．

式（１４）は，大きさ（Lₙ₋₁，hₙ₋₁）の２次元行列と大きさ（Lₙ，

h_1，・・・，h_{n-1}）のｎ次元行列の積であり，これも並列計算が可能である．このとき，ｎ次元行列ｃを２次元行列と見なして計算すると，ベクトル長は$L_n \times h_1 \times \cdots \times h_{n-2}$になる．したがって，高次元になるほどベクトル計算機によく適合する．

以下，同様にして（１５）まで計算する．よって，（１３）から（１５）までの計算で，添字の回転に伴うメモリ衝突を回避するためには，L_n，L_{n-1}，・・・，L_1 のすべてが奇数であればよいことになる．なお，上記の計算法は（１３）～（１５）を計算する部分をサブルーチンとしておき，それを呼ぶたびに配列の大きさを渡す方法で簡単に実現できる．また，ここでも必要なＢ－スプラインの値の計算には，ベクトル計算機向きの算法[9]を適用できる．

4．計算例

では，数値計算例をいくつか示すことにしよう．計算には京都大学大型計算機センターのベクトル計算機を用いた．下記の例１～例５までは，ＦＡＣＯＭ　ＶＰ－２００の結果を，例６はＦＡＣＯＭ　ＶＰ－４００の結果を示している．表１～表６の各欄の意味は，次のとおりである．

ＴＫＮＯＴ：　節点の位置決め
ＴＢＳＰ１：　与えられた標本点でのＢ－スプラインの値の計算
ＴＬＩＮＡ：　式（９）～（１１）の係数行列の作成
ＴＧＡＵＳ：　係数行列のガウス消去
ＴＳＯＬＶ：　定数項の処理および後退代入（添字の回転を含む）
ＴＢＳＰ２：　補間点でのＢ－スプラインの値の計算
ＴＦＵＮＣ：　式（１３）～（１５）による補間値の計算
　　（添字の回転を含む）
ＴＯＴＡＬ：　上記の合計

また，計算モード等の欄の意味は，それぞれ

スカラ：　スカラ実行の処理時間

ベクトル：　ベクトル実行の処理時間

Ｓ／Ｖ：　スカラ実行時間Ｓとベクトル実行時間Ｖの比（加速率）

である．なお，計算はすべて倍精度で行い，実行時間はミリ秒単位で表示している．また，計算例に用いたスプライン関数は，すべての座標軸方向とも同じ次数である．むろん，実際の計算では，必要に応じて座標軸方向ごとに次数を変えることができる．

例１　３次元データの補間（１）

３次元空間の格子点３３×３３×３３の上で与えられたデータを５次のスプライン関数で補間し，格子点３３×３３×３３の上で補間値を計算したときの結果を表１に示す．表のスカラ実行の欄を見ると，ほとんどの処理時間はＴＳＯＬＶおよびＴＦＵＮＣであるが，表のベクトル実行の欄を見ると，これらの部分の処理時間がきわめて少なくなっていることが分か

表１　例１の所要時間（５次スプライン関数を用いた３次元データの補間，
データ点：３３×３３×３３，補間点：３３×３３×３３，単位　ｍｓｅｃ）

計算モード等	TKNOT	TBSP1	TLINA	TGAUS
スカラ（S）	0.10	4.61	0.96	1.28
ベクトル（V）	0.08	0.94	0.21	1.35
S／V	1.25	4.90	4.57	0.95

計算モード等	TSOLV	TBSP2	TFUNC	TOTAL
スカラ（S）	422.45	4.74	245.91	680.05
ベクトル（V）	12.71	1.17	7.45	23.91
S／V	33.24	4.05	33.01	28.44

る．すなわち，ベクトル計算機によく適合している．ＴＯＴＡＬの欄より，全体としての加速率は約２８倍である．

例２　３次元データの補間（２）

メモリ衝突の影響を調べるため，３次元空間の格子点３２×３２×３２の上で与えられたデータを５次のスプライン関数で補間し，格子点３２×３２×３２の上で補間値を計算してみた．その結果を表２に示す．例１に比べると，データの数が少ないにもかかわらず，ＴＳＯＬＶおよびＴＦＵＮＣのベクトル実行の所要時間は例２の方が多い．この理由は，３章で述べた添字の回転に伴うメモリ衝突による．しかし，それでも加速率はＴＳＯＬＶで約２１倍，ＴＦＵＮＣで約１７倍あることに注意したい．これは，メモリ衝突を起こす部分の計算量が，全体の計算量に比べてそれほど多くないことによる．全体の加速率は約１８倍であり，例１に比べるとかなり下がっている．

表２　例２の所要時間（５次スプライン関数を用いた３次元データの補間，
データ点：３２×３２×３２，補間点：３２×３２×３２，単位　ｍｓｅｃ）

計算モード等	TKNOT	TBSP1	TLINA	TGAUS
スカラ（S）	0.08	4.35	0.91	1.25
ベクトル（V）	0..08	0.91	0.18	1.33
S／V	1.00	4.78	5.06	0.94

計算モード等	TSOLV	TBSP2	TFUNC	TOTAL
スカラ（S）	385.23	4.51	231.20	627.53
ベクトル（V）	17.99	1.09	13.80	35.39
S／V	21.41	4.14	16.75	17.73

例3　３次元データの補間（３）

データの量による違いを見るために，例１の約二倍のデータの補間を行ってみた．表３は，３次元空間の格子点６５×６５×６５の上で与えられたデータを５次のスプライン関数で補間し，格子点６５×６５×６５の上で補間値を計算したときの結果である．例１と同様な傾向が表れているが，ＴＳＯＬＶ，ＴＦＵＮＣおよびＴＯＴＡＬの加速率がそれぞれ約４７倍，４５倍，４４倍と上がっている．これは，データ量の増加によりベクトル長が例１のときよりも長くなり，ベクトル計算機にさらによく適合したためである．

表３　例３の所要時間（５次スプライン関数を用いた３次元データの補間，
データ点：６５×６５×６５，補間点：６５×６５×６５，単位　ｍｓｅｃ）

計算モード等	ＴＫＮＯＴ	ＴＢＳＰ１	ＴＬＩＮＡ	ＴＧＡＵＳ
スカラ（Ｓ）	０．１０	１１．０９	２．６０	２．５０
ベクトル（Ｖ）	０．０８	１．４８	０．３４	２．６０
Ｓ／Ｖ	１．２５	７．４９	７．６５	０．９６

計算モード等	ＴＳＯＬＶ	ＴＢＳＰ２	ＴＦＵＮＣ	ＴＯＴＡＬ
スカラ（Ｓ）	３８９４．５３	１１．２８	２２１５．７６	６０３７．８６
ベクトル（Ｖ）	８３．２３	１．９３	４６．９０	１３６．５６
Ｓ／Ｖ	４６．７９	５．８５	４５．１１	４４．２１

例4　３次元データの補間（４）

表４は，３次元空間の格子点６４×６４×６４の上で与えられたデータを５次のスプライン関数で補間し，格子点６４×６４×６４の上で補間値を計算したときの結果である．この例も，例２と同様にメモリ衝突を起こしやすい場合である．データの量が各座標軸方向とも例２の二倍あるが，

ＴＳＯＬＶ，ＴＦＵＮＣおよびＴＯＴＡＬの加速率がそれぞれ約１８倍，１２倍，１５倍となっており，例２に比べると少し下がっている．また，例３に比べると，データの数が少ないにもかかわらず，ベクトル実行での所要時間は多くかかっている．この理由は３章で述べたメモリ衝突による．例１と例２の間の加速率の差と，例３と例４の間の加速率の差を比べてみると，後者の方が大きい．これは，ＶＰ－２００の主記憶のバンク数が６４のため，例２に比べて例４の方が二倍もメモリ衝突を起こしやすいことによる．しかし，それでも加速率は約１５倍あることに注意したい．これは，メモリ衝突を起こす部分の計算量が，全体の計算量に比べてそれほど多くないことによる．

表４　例４の所要時間（５次スプライン関数を用いた３次元データの補間，
データ点：６４×６４×６４，補間点：６４×６４×６４，単位　ｍｓｅｃ）

計算モード等	ＴＫＮＯＴ	ＴＢＳＰ１	ＴＬＩＮＡ	ＴＧＡＵＳ
スカラ（Ｓ）	０．１０	１０．９１	２．５５	２．４２
ベクトル（Ｖ）	０．０８	１．４１	０．３１	２．７９
Ｓ／Ｖ	１．２５	７．７４	８．２３	０．８７

計算モード等	ＴＳＯＬＶ	ＴＢＳＰ２	ＴＦＵＮＣ	ＴＯＴＡＬ
スカラ（Ｓ）	３５１３．３６	１１．０７	１９５７．０３	５４９７．４５
ベクトル（Ｖ）	１９５．７８	１．８５	１６１．４３	３６３．６５
Ｓ／Ｖ	１７．９５	５．９８	１２．１２	１５．１２

例５　３次元データの補間（５）

スプライン関数の次数による違いを見るために，データおよび補間点を例３と同じにして，次数のみを３次に変えて計算した結果を表５に示す．このときの加速率は，ＴＳＯＬＶ，ＴＦＵＮＣおよびＴＯＴＡＬが，それ

ぞれ約４８倍，４１倍，４３倍となり，だいたい例３と同様な結果が得られた．

表５　例５の所要時間（３次スプライン関数を用いた３次元データの補間，
データ点：６５×６５×６５，補間点：６５×６５×６５，単位　ｍｓｅｃ）

計算モード等	TKNOT	TBSP1	TLINA	TGAUS
スカラ（S）	0．10	7．40	2．40	1．41
ベクトル（V）	0．08	1．15	0．29	1．90
S／V	1．25	6．43	8．28	0．72

計算モード等	TSOLV	TBSP2	TFUNC	TOTAL
スカラ（S）	2737．32	7．34	1493．13	4249．09
ベクトル（V）	57．29	1．30	36．82	98．83
S／V	47．78	5．65	40．55	42．99

例６　５次元データの補間

５次元空間の格子点２１×２１×２１×２１×２１の上で与えられたデータを５次のスプライン関数で補間し，格子点２１×２１×２１×２１×２１の上で補間値を計算したときの結果を表６に示す．ＴＳＯＬＶ，ＴＦＵＮＣおよびＴＯＴＡＬの加速率は，それぞれ約７８倍，７５倍，７７倍となり，ベクトル計算機によく適合していることが分かる．なお，この計算結果だけはＶＰ－４００を用いた場合であることに注意したい．

以上の計算例では，すべての座標軸方向でデータの量を同じにしていたが，むろん実際の計算では同じでなくてもよく，任意の数にすることができる．また，スカラ実行のとき，添字を回転してストアするところでは，バッファ記憶と主記憶の間でのバッファイン，バッファアウトによる処理速度の低下があるものと思われ，そのことも加速率にいくぶん影響してい

るであろう.

表6　例6の所要時間（5次スプライン関数を用いた5次元データの補間，データ点：21×21×21×21×21，補間点：　21×21×21×21×21，単位　msec）

計算モード等	TKNOT	TBSP1	TLINA	TGAUS
スカラ（S）	0.10	4.19	0.89	1.22
ベクトル（V）	0.10	1.22	0.39	1.35
S／V	1.00	3.43	2.28	0.90

計算モード等	TSOLV	TBSP2	TFUNC	TOTAL
スカラ（S）	91472.95	4.37	54808.81	146292.5
ベクトル（V）	1170.21	1.46	728.54	1903.3
S／V	78.17	2.99	75.23	76.9

5．むすび

本論文では，ベクトル計算機に適した多次元格子点データの補間方法について述べた．Ｂ－スプラインのテンソル積で作られる多変数スプライン関数を用いた多次元格子点データの補間は，ベクトル計算機によく適合し，高速な補間が可能であることが分かった．

倍精度計算のとき，データの量が各座標軸方向とも奇数であればメモリ衝突を生じなく，データの量が多いほど，またデータの次元数が高いほどベクトル計算機によく適合する性質をもつ．データの量がいずれかの座標軸方向で偶数のときには，メモリ衝突を起こす可能性があるが，すべての座標軸方向とも２のべき乗であるという最悪の場合でも，全体の加速率はかなり大きい．

データの量が偶数である座標軸方向があるとき，その方向のデータの量を１つだけ増やして奇数にするとメモリ衝突を避けることができる．この結果，スカラ実行の処理時間は幾分増加するけれども，ベクトル実行の処理時間はかなり減少する．メモリ衝突の問題を解決するための方法の一つに，リストベクトルを用いる方法があるが[8]，ここで述べたデータを一つ増やす方法が可能であれば，その方が簡単である．

テンソル積型の近似関数による補間は，非テンソル積型の近似関数による補間に比べると，柔軟さに劣り適用できる問題の領域が狭いが，変数分離により，もとの大きな問題をデータの次元数個の小さな問題に分解でき，しかも並列計算が可能である特長をもっている．並列計算のための代表的なアプローチとして分割攻略法（Ｄｉｖｉｄｅ　ａｎｄ　ｃｏｎｑｕｅｒ ｍｅｔｈｏｄ）があるが[10]，本論文で述べたような変数分離を用いた問題の分解と並列計算も，その一つの手法とみなすことができ，並列計算の観点からさらに深く研究してみる価値があると思われる．

補間を高速に計算できれば，それと関連の深い数値微分，数値積分なども高速に計算できると思われるので，用途は広い．ここで述べた考え方は，多次元格子点データの平滑化にも適用できる[11]．

今後の課題としては，（１）マルチプロセッサ型のベクトル計算機との適合性について調べること，（２）テンソル積型の近似関数の適用できる問題の領域を拡大する方法を考えること，（３）モンテカルロ法による補間[12]との比較，（４）高次元データのときのメモリ容量不足の対策，などが考えられる．

謝辞

ベクトル計算機による数値計算の研究を始める機会を与えて下さり，日頃何かと御指導下さる京都大学工学部津田孝夫教授に深く感謝いたします．メモリ衝突の問題についてコメント下さった，中部大学経営情報学部二宮市三教授，筑波大学電子情報系小柳義夫助教授に厚く御礼申し上げます．

参考文献

1) Pereyra, V. and G. Scherer: Efficient computer manipulation of tensor products with applications to multidimensional approximation, Mathematics of Computation, Vol. 27, pp. 595-605, 1973.

2) Hartley, P.J.: Tensor product approximations to data defined on rectangular meshes in N-space, The Computer Journal, Vol. 19, pp. 348-352, 1974.

3) 秦野, 二宮: 二変数補間スプラインの算法と誤差解析, 情報処理, 第１９巻, ｐｐ. １９６－２０３, １９７８.

4) de Boor, C.: Efficient computer manipulation of tensor products, ACM transactions on Mathematical Software, Vol. 5, pp.173-182, 1979.

5) Schoenberg, I,J. and A. Whitney: On Polya frequency functions Ⅲ: The positivity of translation determinants with an application to the interpolation problem by spline curves, Trans. Amer. Math. Soc., Vol. 74, pp. 246-259, 1953.

6) de Boor, C. and A. Pinkus: Backward error analysis for totally positive linear systems, Numer. Math. Vol. 27, pp. 485-490, 1977.

7) de Boor, C. and R. DeVore: A geometric proof of total positivity for spline interpolation, Mathematics of Computation, Vol. 45, pp. 497-504, 1985.

8) 唐木: スーパコンピュータと行列計算, 情報処理, 第２８巻第１１号, ｐｐ. １４４１－１４５１, １９８７.

9) 吉本, 津田: ベクトル計算機に適したＢ－スプラインの計算法, 情報処理学会論文誌, 第２９巻第３号, １９８８.

１０）Duff, I.S.: The impact of parallelism on numerical methods, in Major Advances in Parallel Processing, ed. by C. Jesshope ,The Technical Press, 1986.

１１）吉本：スプライン関数を用いた多次元格子点データの平滑化，情報処理学会第３６回大会論文集，１９８８．

１２）津田：モンテカルロ法とシミュレーション（改訂版），培風館，ｐ．２４５，１９７７．