ベクトル計算機に適したB-スプラインの計算法†

吉本 富士市†† 津田 孝夫†††

ベクトル計算機(スーパコンピュータ)に適したB-スプラインの計算法を提案する．この方法は，de Boor-Coxのアルゴリズムをベクトル計算機向きに改良したものであり，ベクトル長はB-スプラインの値を求める点の数となる．したがって，B-スプラインの値を求める点の数が多いとき（高速計算を必要とするとき）大きな加速率を達成できる．たとえば，富士通のスーパコンピュータVP-200を用いたとき，計算すべき点の数が100個以上であれば，ベクトル化指示行によるチューニングを用いなくても5～22倍の加速率を得ることができる．この計算法の応用例として，1次元データの補間および平滑化の計算を行ったときの，全体の加速率についても述べている．

1. はじめに

スプライン関数は，補間，データ平滑化，積分，画像処理，CAGDなど多くの分野で用いられている重要な近似関数である．スプライン関数を用いるとき，今日では，ほとんどの場合B-スプラインを基底関数としている．したがって，B-スプラインの値を高速に計算する方法を研究することは，大きな意義がある．

通常，B-スプラインの値の計算には，de Boor-Coxのアルゴリズムが用いられている[1)~3)]．この算法は，単一節点の場合だけでなく，多重節点の場合にも数値的に安定であり，高精度な計算ができることが分かっている．しかし，この算法を個々の点に対して直列逐次的に適用すると，ベクトル長がきわめて短くなり，ベクトル計算機（スーパコンピュータ）に適合しない[4)]．この問題を解決する計算法の提案は，まだなされていないようである．

本論文では，de Boor-Coxの算法をベクトル計算機向きに改良する方法を提案する[4),5)]．この方法では，ベクトル長はB-スプラインの値を求める点の数となる．したがって，高速計算を必要とするとき（B-スプラインの値を求める点の数が多いとき）大きな加速率を得ることができる．ここで，加速率とは，ベクトル計算機によるスカラ実行の時間(S)とベクトル実行の時間(V)の比(S/V)を意味する．この計算法を，1次元データの補間および平滑化の計算に用いた結果についても報告する．これらの場合，B-スプラインの値を計算する部分を加速することが，全体の加速率の向上に大きく影響している．

2. B-スプラインの計算法

2.1 算法 I

B-スプラインの値を計算するためのde Boor-Coxのアルゴリズムは，たとえば次のように書くことができる．いま，x軸上の実数列（節点）を $\xi_{i-m}\leqq\xi_{i-m+1}\leqq\cdots\leqq\xi_i$ とするとき，

$$N_{1,i}(x)=\begin{cases}1 & \xi_{i-1}\leqq x<\xi_i\\ 0 & \text{その他}\end{cases} \quad (1)$$

$$N_{r,i}(x)=(x-\xi_{i-r})N_{r-1,i-1}(x)/(\xi_{i-1}-\xi_{i-r})+(\xi_i-x)N_{r-1,i}(x)/(\xi_i-\xi_{i-r+1}) \quad (r=2,3,\cdots,m) \quad (2)$$

である．ここで，$N_{r,i}(x)$ $(r=1,2,\cdots,m)$ は正規化されたr階（$r-1$次）のB-スプラインを意味する*．また，mは点xにおいて零でないm階のB-スプラインの数である．式(1)および(2)のアルゴリズムを，本論文では算法Iと呼ぶことにする．

さて，x軸上の区間$[a,b]$で，$N_{m,i}(x)$ $(i=1,2,\cdots,h)$を基底関数とした近似関数を作ることにしよう．ここで，hは区間$[a,b]$に関係するB-スプラインの数である．いま，区間$[a,b]$の内部の節点をξ_i $(i=1,2,\cdots,h-m)$として，区間の両端でm個ずつの付加節点を置く．すなわち，

$$\left.\begin{aligned}a&=\xi_{1-m}=\xi_{2-m}=\cdots=\xi_0\\ b&=\xi_{h-m+1}=\xi_{h-m+2}=\cdots=\xi_h\end{aligned}\right\} \quad (3)$$

とすると，近似関数は，

$$S(x)=\sum_{i=1}^{h}c_iN_{m,i}(x) \quad (4)$$

と書くことができる．このとき，B-スプラインの性質

† A Method of the Numerical Evaluation of B-splines for a Vector Computer by FUJIICHI YOSHIMOTO (Akashi College of Technology) and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 明石工業高等専門学校
††† 京都大学工学部情報工学科

* 点xにおいて零でない値をもつr階のB-スプラインはr個あるが，そのr個の値の和が1になるように，大きさを調整したB-スプラインを正規化されたB-スプラインと呼んでいる．

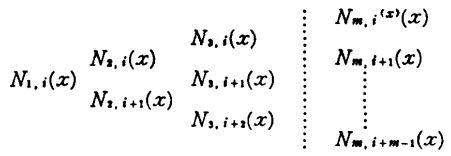

図 1 算法Ⅰによる B-スプラインの値の計算順序
Fig. 1 Order of computation for the values of B-splines by Algorithm-1.

$$N_{m,i}(x)\begin{cases} >0 & \xi_{i-m}<x<\xi_i \\ =0 & \text{その他} \end{cases} \qquad (5)$$

から，区間 $[a,b]$ 内の任意の点 x において m 個の B-スプラインのみが零でない値をもつことに注意しよう．したがって，(4)を用いた $S(x)$ の値の計算では，実際には m 個の B-スプラインの線形結合のみを計算すればよい．

これらの値は次のようにして計算できる．いま，$\xi_{i-1}\leqq x<\xi_i$ とするとき，**図1**の配列要素を計算する．まず $N_{1,i}(x)$ から計算を始め，左から右へと計算する．このとき，各要素は，その要素の左側の二つの要素（一つはすぐ上の要素，もう一つはすぐ下の要素）を用いて計算できる．たとえば，$N_{3,i+1}(x)$ は $N_{2,i}(x)$ と $N_{2,i+1}(x)$ を用いて計算できる．また，(5)の性質から，この配列に表れない要素はすべて零であるから，たとえば $N_{3,i+2}(x)$ は $N_{2,i+1}(x)$ のみから計算できる．なお，各列において，計算の順序を下から上へと行うことにすれば，一つの標本点 x に対して大きさ m の配列を用意するだけでよい．なぜならば，$N_{r,j}(x)$ $(r=1,2,\cdots,m)$ は r の変化に対して重ね書きできるからである．

さて，通常は一つ一つの標本点に対して逐次的に図1の手順で計算を行っている．スカラ計算機を用いる場合には，それでよい．しかし，ベクトル計算機を用いるときには，このようにすると，ベクトル処理の高速な性能をまったく引き出すことができない．

この理由は，①de Boor-Cox のアルゴリズムは漸化式型であるから，ベクトル計算機に適合しにくい性質をもっていること，②スプライン関数を用いるとき，通常は B-スプラインの階数（次数＋1）m をあまり大きくしないで，10以下に取るため，ベクトル長が小さなものとなること，である．B-スプラインの階数 m を小さくする理由は，もしもそれを大きくすると，①近似関数(4)の柔軟さが失われ，近似曲線にうねりを生じやすいこと，②補間，あてはめなどの問題で，解くべき連立一次方程式の条件が悪くなること，③B-スプラインの計算量が増大すること，などである．

では，図1の計算をベクトル計算機に適合させるためには，どのようにすればよいであろうか．その鍵は近似関数(4)にある．近似関数に含まれる h 個の B-スプライン $N_{m,i}(x)$ $(i=1,2,\cdots,h)$ は，節点が決まればすべて決まってしまうことに注意しよう．したがって，区間 $[a,b]$ 内のある点での B-スプラインの値は，その他の点での B-スプラインの値とは独立に計算できる．すなわち，区間 $[a,b]$ 内で B-スプラインの値を計算したいすべての点 x_k $(k=1,2,\cdots,N)$ において，図1の手順を用いて同時に（並列に）必要な B-スプラインの値を計算できる．このようにすると，ベクトル長を標本点の数 N にすることができる．一般に，高速計算を必要とする問題では N は大きいと考えてよいので，この方法はベクトル計算機に適合する．

2.2 算 法 Ⅱ

ところで，de Boor-Cox のアルゴリズムは次のように書くこともできる．

$$M_{1,i}(x)=\begin{cases}(\xi_i-\xi_{i-1})^{-1} & \xi_{i-1}\leqq x<\xi_i \\ 0 & \text{その他}\end{cases} \qquad (6)$$

$$M_{r,i}(x)=\{(x-\xi_{i-r})M_{r-1,i-1}(x) + (\xi_i-x)M_{r-1,i}(x)\}/(\xi_i-\xi_{i-r}) \quad (r=2,3,\cdots,m-1) \qquad (7)$$

$$N_{m,i}(x)=(x-\xi_{i-m})M_{m-1,i-1}(x) + (\xi_i-x)M_{m-1,i}(x). \qquad (8)$$

本論文では，(6)，(7)，(8)のアルゴリズムを算法Ⅱと呼ぶことにする．

さて，算法Ⅰと算法Ⅱの演算量を比較してみよう．算法Ⅰでは，除算 2 $(m-1)$ 回，乗算 2 $(m-1)$ 回，減算 4 $(m-1)$ 回である．一方，算法Ⅱでは除算 $m-1$ 回，乗算 2 $(m-1)$ 回，減算 3 $(m-1)$ 回である．したがって算法Ⅱの方が演算量が少なく，特に除算の量が半分である．そこで，算法Ⅱをベクトル計算機に適合させることができれば，算法Ⅰよりも高速な算法になると思われる．それは算法Ⅰの場合と同様な考え

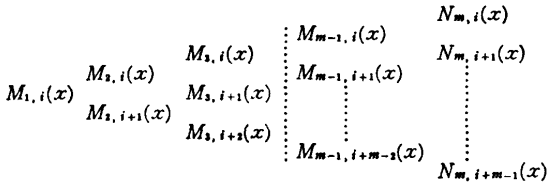

図 2 算法Ⅱによる B-スプラインの値の計算順序
Fig. 2 Order of computation for the values of B-splines by Algorithm-2.

方で可能であり，必要な記憶容量も算法Ⅰと同じである．

すなわち，図1と同様に，**図2**のようにしてB-スプラインの値を計算する．このとき，区間 $[a, b]$ 内で B-スプラインの値を計算したいすべての点 x_k $(k=1, 2, \cdots, N)$ において，図2の計算手順を用いて同時に（並列に）必要な B-スプラインの値を計算する．

ここで，図2の最も右側の列のみが正規化された B-スプラインであることに注意しよう．この算法の要点は，はじめ正規化されていない B-スプラインで計算しておいて，最後に正規化を行うことである．

3. 計算結果

2章で述べたようにして B-スプラインの値を計算したときの，例を示し比較検討する．使用した計算機は，富士通のスーパコンピュータ（ベクトル計算機）FACOM VP-200 である．標本点は区間 [0, 1] 内で与えた．また節点は，その区間内で等間隔に取った分点の上で与えた*．ただし，区間の両端では m 個重ね，内部では重ねていない．計算はすべて倍精度で行った**．

図3は，算法Ⅰで，一つ一つの標本点に対して直列逐次的に計算した場合の加速率である．B-スプラインの階数（次数+1）m が3および10のときを示しているが，m がいずれの場合にも加速率が1を越えていない．したがって，2章で述べたように，通常の m の大きさ（$m \leqq 10$）では，ベクトル計算機の性能をまったく引き出すことができない．ここで，標本点はランダムに与え，節点は11個の分点の上で与えた．

図4は，算法Ⅰですべての標本点に対して集団並列的に B-スプラインの値を計算したときの加速率である．標本点の数が100以上であれ

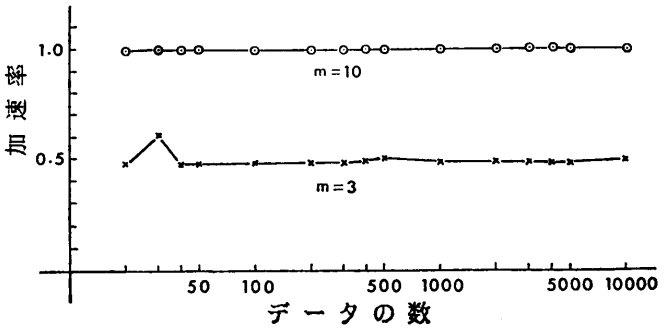

図3 算法Ⅰで一点ずつ逐次計算したときの加速率（ランダムな標本点）
Fig. 3 The ratio of acceleration by Algorithm-1, where the values of B-splines are computed sequentially for random sample points.

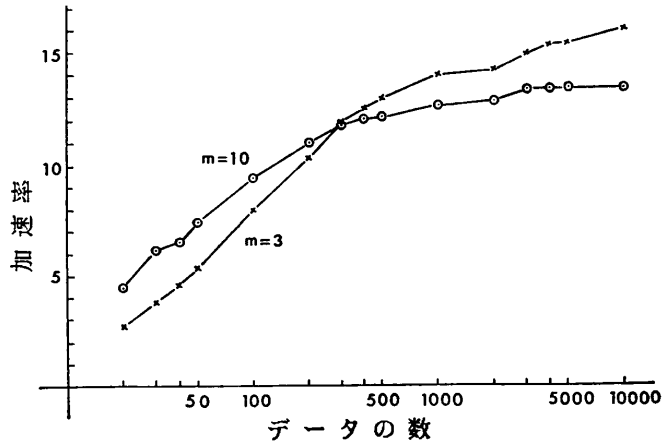

図4 算法Ⅰですべての点で並列計算したときの加速率（ランダムな標本点）
Fig. 4 The ratio of acceleration by Algorithm-1, where the values of B-splines are computed in parallel for all of the random sample points.

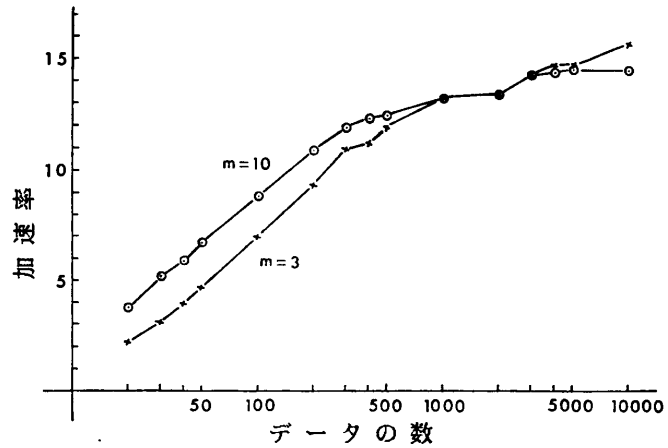

図5 算法Ⅱですべての点で並列計算したときの加速率（ランダムな標本点）
Fig. 5 The ratio of acceleration by Algorithm-2, where the values of B-splines are computed in parallel for all of the random sample points.

* 本論文では，分点と節点は別の概念として使っている．すなわち，分点とは多項式片 (polynomial piece) のつなぎ目を意味し，節点とは B-スプラインの計算式の ξ_i $(i=1-m, \cdots, h)$ を意味する（算法Ⅰおよび算法Ⅱ参照）．節点は分点の上で m 個まで重ねることができ，いくつ重ねるかによって，その分点上でつながれている多項式片の連続性が違ってくる．

** 本論文で述べる計算例の実行時間データは，京都大学大型計算機センターで測定したものである．図9は1987年10月に，それ以外は1987年6月に測定したデータによる．

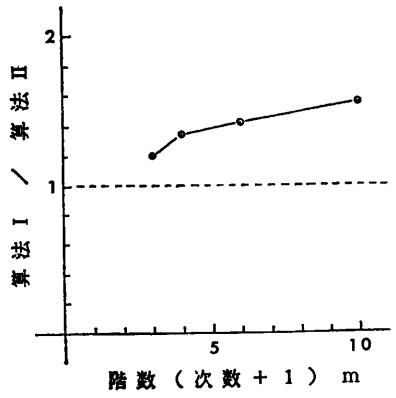

図 6 算法Ⅰと算法Ⅱの計算時間の比
Fig. 6 The ratio of computation times for Algorithm-1 to Algorithm-2, where the number of sample points is 1000.

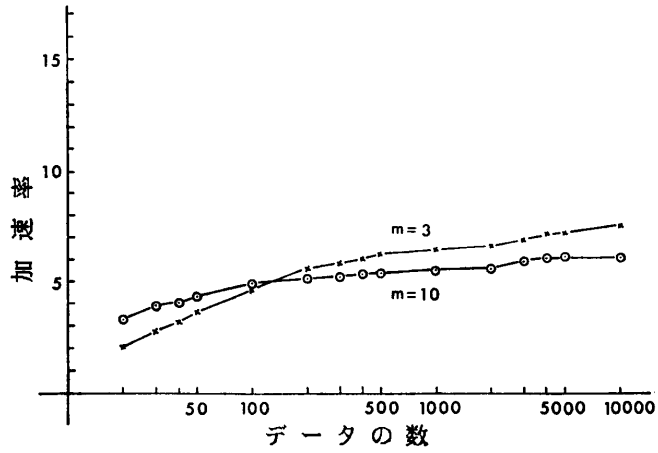

図 7 算法Ⅱですべての点で並列計算したときの加速率（11 個の分点上に節点を取り，標本点を等間隔かつ上昇順に与えたとき）
Fig. 7 The ratio of acceleration by Algorithm-2, where the values of B-splines are computed in parallel for all of the sample points. Knots are set on the eleven points of partition. Sample points are given equidistantly and in ascending order.

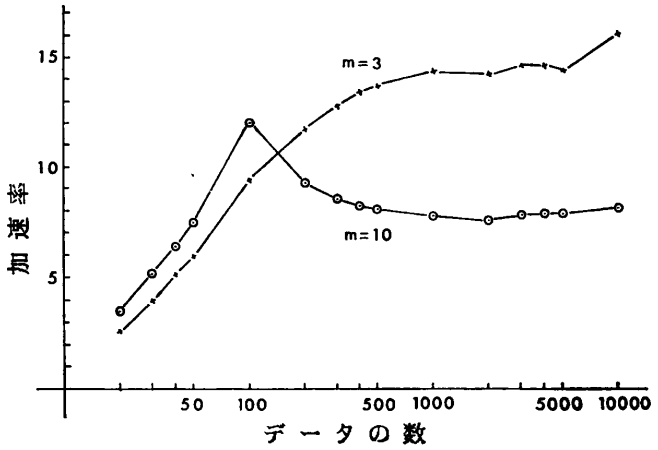

図 8 算法Ⅱですべての点で並列計算したときの加速率（101 個の分点上に節点を取り，標本点を等間隔かつ上昇順に与えたとき）
Fig. 8 The ratio of acceleration by Algorithm-2, where the values of B-splines are computed in parallel for all of the sample points. Knots are set on the 101 points of partition. Sample points are given equidistantly and in ascending order.

ば 8 倍以上の加速率が得られている．この図と図 3 を比較すると，このような計算法の有効性が分かる．標本点および節点は，図 3 の場合と同じ方法で生成した．

図 5 は，算法Ⅱを用いて，すべての標本点に対して集団並列的に計算したときの加速率である．この図と図 4 を比較してみると，加速率にはほとんど差がないことが分かる．標本点および節点は，図 3 の場合と同じ方法で生成した．

図 6 は，算法Ⅰと算法Ⅱの計算時間の比を表している．ここで，標本点の数は 1,000 である．この図から，2 章で述べたように算法Ⅱの方が高速であり，その割合は VP-200 の場合 2～5 割程度であることが分かる．

図 7 は，等間隔で，かつ上昇順に並べられた標本点に対して，算法Ⅱを用いて集団並列的に計算したときの結果である．節点は，図 4 のときと同様に生成した．この図と図 4 を比較してみると，加速率がかなり下がっていることが分かる．算法Ⅰでも同様な結果が得られている．しかし，等間隔で，かつ上昇順に並べられた標本点のときでも，分点の数を増加する（したがって，節点の数を増加する）と加速率はかなり高くなる．**図 8** は，その様子を示すものである．

図 7 で加速率が下がる原因は，リストベクトルを用いて節点の値を参照するときのメモリ衝突のためである．すなわち，上昇順に並べられたデータでは，データの数が多いとき，同じ節点に対して B-スプラインの値を複数個連続に計算することになるからである．

これを確認するために行った計算例が**図 9** である．

図 9 は，図 7 の場合と同じ等間隔データに対して B-スプラインの値を計算したときの結果である．ただし，データを与える順序を上昇順にしないで，同じ節点を連続的に参照しないように工夫している．このときには，等間隔データであってもランダムデータの場合をしのぐ加速率が得られている．図 8 のとき，図 7 よりも加速率が上がった理由は，節点の数が増えたた

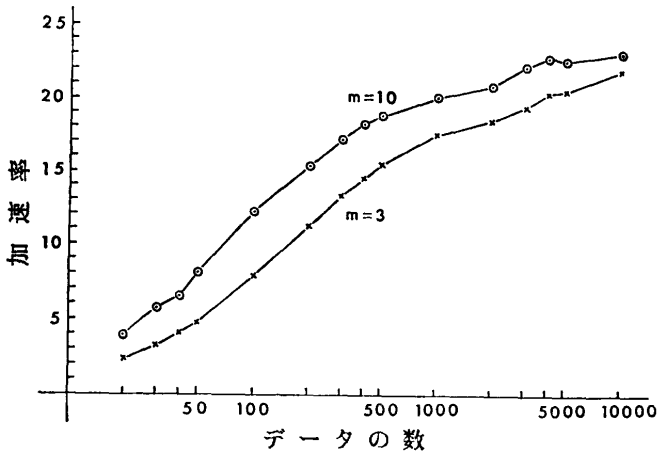

図 9 算法Ⅱですべての点で並列計算したときの加速率（メモリ衝突を避けるように標本点を与えたとき．節点は図7と同様に生成した．）
Fig. 9 The ratio of acceleration by Algorithm-2, where the values of B-splines are computed in parallel for all of the sample points. Sample points are given equidistantly, but they are not in ascending order to avoid memory conflict. Knots are the same as those of Fig. 7.

め，同じ節点に対して連続的に計算する B-スプラインの値の数が減少し，メモリ衝突の回数が少なくなったからである．

ところで，算法Ⅰおよび算法Ⅱは，ともに階数 r に対して漸化式の型をしているので，図1または図2のように計算するとき，階数 r の方向（横方向）にはループ・アンローリングを行うことはできない．しかし，$j=i,\ i+1,\cdots,i+r-1$ の方向（縦方向）にはループ・アンローリングを適用できる．そこで，試みに算法Ⅰで縦方向に二重のループ・アンローリングを行ってみたが，その効果はわずかであった．

4. 補間・データ平滑化問題への応用

2.2節で述べた，算法Ⅱによるベクトル計算機向きの B-スプラインの計算法を用いて，近似関数(4)により補間および平滑化を行った例を示す．ここでは，1次元データの場合だけについて述べる．2次元以上の場合については別の報告を参照されたい[6]．

4.1 補　　間

区間 $[a, b]$ 内で，標本点 $x_1, x_2, \cdots, x_N$ $(x_1<x_2<\cdots<x_N)$ に対して，関数値 $y_1, y_2, \cdots, y_N$ が与えられているものとする．このデータを $m-1$ 次のスプライン関数(4)を用いて補間する．ここでは，区間 $[a, b]$ の両端の付加節点を(3)のように取り，内部の節点は

$$\xi_i = x_{i+[m/2]} \quad (i=1, 2, \cdots, h-m) \qquad (9)$$

とする．これら内部の節点は必ずしも標本点と一致させる必要はなく，節点を変化させていろいろな補間を求め，その中からよいものを選ぶこともできる[7]．しかし，本論文では，簡単のため，(9)のようにする．

式(4)が，与えられたデータ点を通ることから，

$$\sum_{i=1}^{h} c_i N_{m,i}(x_k) = y_k \quad (k=1, 2, \cdots, N) \qquad (10)$$

を得る．ここで $h=N$ である．この式を解いて，その解を(4)へ代入すれば，区間 $[a, b]$ 内の任意の点 x で補間値を求めることができる．

したがって，ある節点の組に対するスプライン関数を用いた補間では，主として次の5個の部分の計算が必要である．①すべての標本点での B-スプラインの値の計算，②式(10)の要素の決定，③式(10)の解 c_i $(i=1, 2, \cdots, N)$ の計算，④補間値を計算したいすべての点での B-スプラインの値の計算，⑤式(4)による補間値の計算．このうちで多くの時間を消費するのは，通常①，③，④である．この中で①，④は2章で述べた方法で高速に計算できる．また，データをあらかじめすべて与えてベクトル長を長くする考え方は，⑤にも適用できる．

表1は，3次および5次の B-スプラインを用いて，

表 1 補間計算の各部分の所要時間（単位 msec）
Table 1 The time of computation for each part of interpolation (CPU-times in milliseconds).

階数（次数+1）m	計算モード	①標本点での B-スプラインの値の計算	②式(10)の要素の決定	③式(10)の解の計算	④補間点での B-スプラインの値の計算	⑤式(4)による補間値の計算	①～⑤の全体
4	スカラ	4.74	1.67	1.41	23.62	1.07	32.51
	ベクトル	0.57	0.16	2.08	2.19	0.21	5.21
6	スカラ	6.02	1.74	2.01	30.21	1.54	41.52
	ベクトル	0.70	0.13	3.23	3.33	0.31	7.70

区間 [0, 5] で等間隔に与えられた101個のデータを補間し，等間隔な501個の点で補間値を計算した場合の，各部分の計算時間を示している．この表の，上段はスカラ実行した場合を，下段はベクトル実行した場合を表している．この表からも，B-スプラインの値の計算を高速化すれば，全体の高速化に大きく貢献することが分かる．

また，同じデータを用いて，階数（次数+1）m の値を3〜10まで変えて計算したところ，全体の加速率は約5〜8倍であった．ただし，ループ・アンローリングとかベクトル化指示行などによるチューニングを行っていないプログラムによる計算結果である．

なお，FACOM SSL II の中の DBIC 3 および DBIF 3 を用いて同じ補間を行ったとき，3次および5次のスプライン関数いずれにおいても，まったく加速されなかった[8]．

4.2 データ平滑化

区間 $[a, b]$ 内で，標本点 $x_1, x_2, \cdots, x_N$ $(x_1<x_2<\cdots<x_N)$ に対して，データ

$$F_k=f(x_k)+\varepsilon_k \quad (k=1, 2, \cdots, N) \tag{11}$$

が与えられているものとする．ここで，$f(x)$は未知の関数（信号）であり，ε_k は測定誤差である．

データ(11)の近似関数として，補間のときと同様に(4)を用いる．ただし，内部の節点は m 個まで重ねることができる．

最小二乗法を用いて(4)を(11)へあてはめることにより，データ平滑化を行う．このとき，適切な節点の数と位置を決めることが，良い近似を得る鍵となるが，ここでは，それはあらかじめ決められているものとして，パラメタ c_i $(i=1, 2, \cdots, h)$ のみを未知数とする線形最小二乗法の問題を扱う．このとき，残差の二乗和は，

$$Q=\sum_{k=1}^{N} w_k\{S(x_k)-F_k\}^2 \tag{12}$$

となる．ここで，w_k はデータの重みである．

式(12)をパラメタ c_i $(i=1, 2, \cdots, h)$ で偏微分して零とおくと，正規方程式

$$Ac=d \tag{13}$$

を得る．ここで，

$$c=(c_1, c_2, \cdots, c_h)^T \tag{14}$$

である．式(13)の解 c を(4)へ代入すれば，区間 $[a, b]$ 内の任意の点 x で平滑値を求めることができる．

さて，良い近似を得るためには，節点の数と位置をいろいろ変えて多くのあてはめを計算し，その中から良いものを選び出す必要がある．このためには，あてはめの良さを評価する規準がなければならない．この規準としてよく用いられるものに，誤差分散の不偏推定量 δ と赤池の情報量規準 AIC がある[3]．いま，回帰模型を

$$F_k=S(x_k)+\varepsilon_k \quad (k=1, 2, \cdots, N) \tag{15}$$

とする．ここで，誤差 ε_k は互いに独立で，平均値0，分散 σ^2 の正規分布に従うものと仮定する．このとき，誤差分散の不偏推定量は

$$\delta=Q/(N-h) \tag{16}$$

となり，赤池の情報量規準は

$$\mathrm{AIC}=N\log_e Q+2h \tag{17}$$

と表すことができる．ここで，Q は(12)で与えられ，h はパラメタ c_i $(i=1, 2, \cdots, h)$ の数である．

以上のことから，ある節点の組に対するあてはめ（1回分のあてはめ）において，主に次の6個の部分の計算が必要なことになる．①すべての標本点でのB-スプラインの値の計算，②式(13)の要素の計算，③式(13)の解 c_i $(i=1, 2, \cdots, h)$ の計算，④残差，あてはめの規準の計算，⑤平滑値（近似関数値）を計算したいすべての点でのB-スプラインの値の計算，⑥式(4)による平滑値の計算．

このうちで多くの時間を消費するのは，通常①，②，⑤であるが，①，⑤は2章で述べたようにして高速に計算できる．また，②，④，⑥も，データをあらかじめすべて与えて集団並列的に計算することによりベクトル長を長くできるので，高速な計算が可能である．

表 2 は，区間 [0, 1] で等間隔に与えられた500個

表 2 平滑化計算の各部分の所要時間（単位 msec）
Table 2 The time of computation for each part of data fitting (CPU-times in milliseconds).

階数（次数+1）m	計算モード	①標本点でのB-スプラインの値の計算	②式(13)の要素の計算	③式(13)の解の計算	④残差，あてはめの規準の計算	⑤平滑化点でのB-スプラインの値の計算	⑥式(4)による平滑値の計算	①〜⑥の全体
4	スカラ	7.24	2.50	0.21	1.28	7.16	1.07	19.46
	ベクトル	1.33	0.60	0.34	0.31	1.30	0.26	4.14
6	スカラ	14.69	4.66	0.31	1.77	14.59	1.54	37.56
	ベクトル	2.70	0.96	0.47	0.44	2.68	0.39	7.67

のデータを平滑化し，等間隔な501個の点で平滑値を計算した場合の各部分の計算時間を示している．ここで，節点は等間隔に取った11個の分点の上で与えた．表の上段はスカラ実行した場合を，下段はベクトル実行した場合を表している．この表からも，B-スプラインの計算の高速化の意義が分かる．

また，同じデータと節点を用いて，B-スプラインの階数 m を3～10まで変えて平滑化の計算を行ったところ，全体の加速率は約4～5倍であった．ただし，ループ・アンローリングとかベクトル化指示行などによるチューニングを行っていないプログラムによる計算結果である．

なお，FACOM SSL II の中の DBSC 1 および DBSF 1 を用いて同じデータを平滑化したとき，3次および5次のスプライン関数いずれの場合にも，まったく加速されなかった[8]．

5. むすび

本論文では，ベクトル計算機向きの B-スプラインの計算法と，その補間，データ平滑化問題への応用について述べた．B-スプラインの値を計算するための de Boor-Cox のアルゴリズムは，標本点一つずつに対して直列逐次的に適用したとき，ベクトル長がきわめて短くなるので，ベクトル計算機に適合しない．しかし，近似関数に含まれる B-スプラインの値は各点で互いに独立に計算できることに着目し，あらかじめすべての標本点を与えて B-スプラインの値を計算することにすれば，ベクトル計算機に適した算法となることが明らかになった．ここで述べた方法を用いれば，1次元の補間，あてはめにおいて，ループ・アンローリングとかベクトル化指示行などによるチューニングを行っていないプログラムでも，かなり大きな加速率を達成できる．

ただし，この方法が有効であるのは，標本点の数が多い場合である．また，リストベクトルを用いているため，その処理の速い計算機ほど加速率はよい．

なお，本論文で述べた計算法では，データを与える順序が加速率にかなり影響し，上昇順に与えるよりもランダムに与える方がよいことが分かった．ただし，補間，あてはめの問題で B-スプラインの値を計算するとき，データをランダムに供給することは他の部分の計算の容易さとも関係するので，さらに研究が必要である．

謝辞 メモリ衝突の問題についてコメント下さった査読者に感謝いたします．本研究の一部は京都大学大型計算機センターの開発課題として行われた．

参考文献

1) de Boor, C.: On Calculating with B-splines, *J. Approx. Theory*, Vol. 6, pp. 50-62 (1972).
2) Cox, M. G.: The Numerical Evaluation of B-splines, *J. Inst. Maths. Applics*, Vol. 10, pp. 134-149 (1972).
3) 市田，吉本：スプライン関数とその応用（シリーズ新しい応用の数学 20），p. 220，教育出版，東京 (1979).
4) 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，京都大学数理解析研究所講究録第613号［スーパーコンピュータのための数値計算アルゴリズムの研究］，pp. 188-203 (1987).
5) 吉本，津田：ベクトル計算機に向いたB-スプラインの計算法とその応用，第34回情報処理学会全国大会論文集，pp. 45-46 (1987).
6) 吉本，津田：スプライン関数を用いた多次元データの平滑化―ベクトル計算機向きの算法―，第3回ベクトル計算機応用シンポジウム論文集，京都大学大型計算機センター，pp. 27-36 (1987).
7) 吉本：自由節点のスプライン関数を用いた補間について，情報処理学会論文誌，Vol. 22，No. 4，pp. 304-311 (1981).
8) 富士通：FACOM OS IV/F4 MSP FORTRAN 77/VP プログラミングハンドブック (1985).

(昭和62年7月 7日受付)
(昭和63年1月19日採録)

吉本富士市（正会員）

昭和18年生．昭和41年岡山大学工学部電気工学科卒業．同年明石工業高等専門学校助手．講師，助教授を経て昭和62年教授，現在に至る．専門共通科目（情報処理）担当．昭和52年工学博士（京都大学）．昭和59年11月～60年8月文部省在外研究員として米国パデュー大学などに出張．現在の研究テーマは，関数近似の問題（特に多変数問題）における並列数値計算とその応用など．

津田 孝夫（正会員）

1932年生．1957年京都大学工学部電気工学科卒業．現職は京都大学工学部情報工学科教授．工学博士．現在の主要研究テーマは，メモリ階層間データ転送量の下界とそれによるアルゴリズムの最適化，ベクトル計算機のための自動ベクトル化の自動並列化，実時間オペレーティングシステムなど専用 OS の構成と実現法など．