2009年度応用マクロ経済学講義ノート　多項式近似

阿部修人
一橋大学

平成 21 年 12 月 26 日

概 要

1　導入

本講義ノートの目的は (1) チェビシェフ多項式 (Chebyshev　Polynomial) および Spline 補間による関数近似手法の基本について解説し、(2) 動的計画法への応用例を説明すること、である。Matlab Code の実行には Miranda and Fackler (2002) が提供している Compecon Toolbox が必要である。途中の、様々な Spline や Chebyshev による近似のプログラムはあえて、本講義ノートには掲載していない。よい練習になるので、Miranda and Fackler のプログラムを利用して、自分で、任意の関数の多項式近似が出来るようになってもらいたい。なお、動的計画法に関する Code の大部分は山田知明氏が作成したものに準拠している。この一連の講義ノートで解説している Code と同様に、あくまで教育目的で書かれたものであり、Research にそのまま実際に使用することは推奨しかねる。自分の責任で使うこと。

1.1　多項式近似の考え方

これまで、我々は動学最適化問題を数値的に解く手法として、(1) 線形近似、(2)Shooting、および (3) 離散近似、の三種類を議論してきた。線形近似は極めて扱いやすく、また数百個の状態変数を扱うことも可能であったが、分析対象が 1) 経済諸変数間の関係が微分可能であり、かつ 2) 近似点の近傍のみの動きに注目する、という制約の下でのみ適用可能であった。2) は、微分可能性を仮定する必要はなく、かつ、Global な分析が可能になるメリットがあるが、Shooting で初期点を探す際に、操作変数が大量にあると解くのが困難になること、および不確実性を導入することが極めて困難であるという

問題があった。石油ショックやリーマンショック等も、経済主体が数十年前から事前に発生を正確に予測できていたと仮定するのはかなり無理がある。一方、(3) の離散近似法は線形近似と正反対の特徴を持っており、不等号制約等の微分不可能な関数を扱うことが可能であり、かつ、近似の位置に数値解が依存することも少ないという良い性質がある (グリッドの範囲には敏感に反応するので、適切なグリッドの位置を決める必要はある)。しかしながら、計算可能なモデルはごく少数の状態変数を含む物に限定されてしまい、単純な成長モデルであっても膨大な量のメモリーを消費し、時間もかかるという欠点があった。多項式による近似とは、両者の間に存在するものであり、線形関数よりもより広い関数空間の中で近似を探すものである。したがって、線形近似と離散近似の長所と短所双方のバランスをとっているものと考えることも可能であり、現在では動学経済分析において頻繁に用いられている。

1.2 本講義ノートの構成

まず、多項式近似の考えを単純な例を用い、John P. Boyd (2001) *Chebyshev and Fourier Spectral Methods* に従い紹介する[1]。次に、Chebyshev Polynomial およびスプライン補間に関して議論し、Chebyshev Points および Collocation に関して説明する。詳細な議論はしないが、なぜ Chebyshev Polynomial と Collocation がよく使用されてきたか、その数学的な背景も若干説明する。また、Spline に関しても極めて簡単にではあるが説明する。最後に単純な最適成長モデルに応用する matlab code を Chebyshev、Spline いずれのケースに関しても紹介する。

2 多項式近似の例

下記の二階の微分方程式を考える。

$$\begin{aligned} u_{xx} - \left(x^6 + 3x^2\right) u &= 0 \\ u(-1) = u(1) &= 1 \end{aligned} \tag{1}$$

すなわち、二点の境界条件を含む二階の微分方程式であるから、我々はこの解 $u(x)$ を (もしも存在すれば) 特定することができる。実際、この問題の解は closed form が存在することが知られており

$$u(x) = \exp\left[\left(x^4 - 1\right)/4\right]$$

である。

[1] ちなみに、この著者は『エデンの受粉者』(ハヤカワ SF 文庫) の SF 作家としても有名である。数学の世界とは異なる、SF 作家兼ねエンジニアの手による面白い教科書なので (Dover だから安いし)、前半だけでも読んでみることをお勧めする。

実際、

$$u(-1) = \exp[(1-1)/4] = \exp[0] = 1 = u(1)$$

$$\begin{aligned} u_x &= x^3 \exp\left[\left(x^4-1\right)/4\right] \\ u_{xx} &= 3x^2 \exp\left[\left(x^4-1\right)/4\right] + x^6 \exp\left[\left(x^4-1\right)/4\right] \\ &= \left(x^6+3x^2\right) \exp\left[\left(x^4-1\right)/4\right] \\ &= \left(x^6+3x^2\right) u \end{aligned}$$

我々の関心は、定微分方程式の真の解を様々な技法を駆使して求めることではなく、ある種の関数の集合の中から、(1) をみたすものを見つけることである。とりあえず、解の候補として

$$u_c = 1 + \left(1-x^2\right)\left(a_0 + a_1 x + a_2 x^2\right)$$

の形状の関数を考えてみる。ここで、a_i は未定係数であり、ここでは自由度 3 の係数となっている。このような関数形を仮定する理由は単純である。境界条件をみたす自然な形状は $\left(1-x^2\right)$ を含ませることであり、あとは自然な二次の多項式である。無論、二次を仮定したのは、単純化のためであり、手で計算するのは困難になるが、n 次の多項式でも構わない。無論、真の解は指数関数であり、多項式ではないので、上記のような多項式の中で探す限り、真の解にたどりつくことはありえず、常に誤差が含まれることになる。問題は、その誤差がどの程度大きいか、どの程度のよい fit が得られるか、どうすれば、最小の多項式で、最大の fit を実現できるか、である。

u_c は境界条件を満たすから、あとは微分方程式を満たせばよい。そこで、u_c の二階微分と (1) の差を考え

$$R(a_0, a_1, a_2; x) = u_{cxx} - \left(x^6+3x^2\right) u_c$$

と定義する。これは、u_c と真の関数の間の差、すなわち残差である。これを計算すると

$$\begin{aligned} R = {} & 2(a_0+a_1) - 6a_1 x - 3(1+a_0+4a_2)x^2 \\ & - 3a_1 x^3 + 3(a_0-a_2)x^4 + 3a_1 x^5 + (-1-a_0+3a_2)x^6 \\ & - a_1 x^7 + (a_0-a_2)x^8 + a_1 x 9 + 10 a_2 x^{10} \end{aligned}$$

さて、我々はこの値が全ての x においてゼロになっていて欲しいのだが、無論そんなことは不可能であり、「いくつかの」点でゼロになっていてくれるような関数形を探すことにする。未定係数の数は 3 つであり、3 点の情報があれば未定係数の値を定めることが出来る。そこで、特に根拠は考えず、

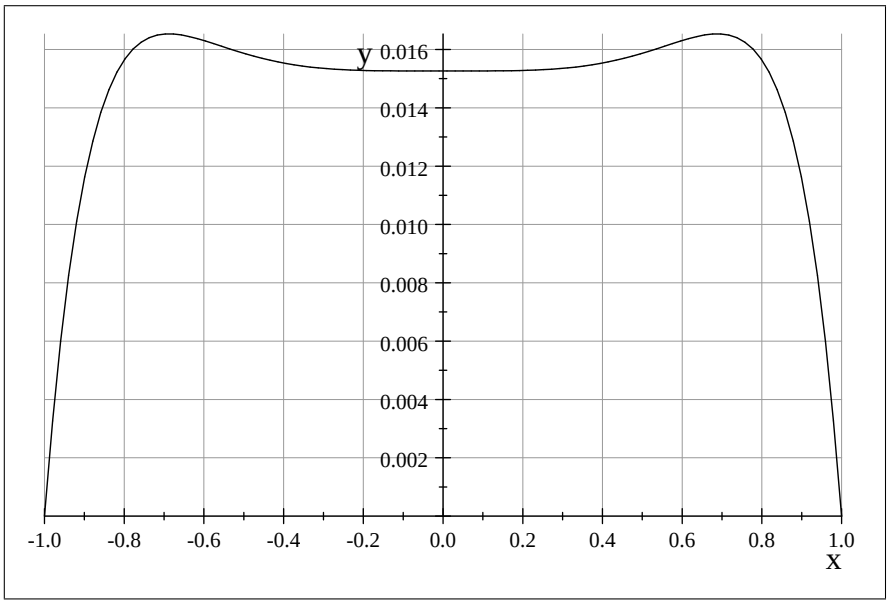

図 1: 真の値からのかい離:Boyd(2001) と同様の図である。

$x = (-1/2, 0, 1/2)$ の三点でのみ R がゼロになるような未定係数を求めよう。すると、これは単に三変数の線形連立方程式を解く問題になり、簡単に解を得ることが出来る。ちなみに、このように、n 個の未定係数を、n 個の点において真の関数と近似関数が一致するという情報を利用して特定する手法を Collocation という[2]。$x = (-1/2, 0, 1/2)$ において R=0 とする三本の連立方程式を a_0, a_1, a_2 に関して解くと

$$a_0 = -\frac{784}{3807}, a_1 = 0, a_2 = a_0$$

を得る。この近似の結果と真の値との乖離、すなわち $R(a_0, a_1, a_2; x)$ を plot すると、図 1 のようになる。

を得る。この近似の結果と真の値との乖離、すなわち $R(a_0, a_1, a_2; x)$ を plot すると、図 1 のようになる。なお、真のグラフの形状は図 2 に示されている。

真の解の値は、-1<x<1 において、0.8 から 1 の間に存在する。一方、誤差は 0.02 以下であるから、たった 3 つの未定係数による近似であるが、誤差はかなり小さいと言うことができるだろう。もっとも、図 1-1 からわかるように、この真の値のグラフは、高次多項式の形状に酷似しており、四次多項式で近似した場合、その誤差が小さいことはそれほど不思議なことではない。

[2]Collocation は点における一致を重視する手法であるが、ある領域における一致 (積分) を重視する手法に Galerkin がある。詳しくは Judd (1998) を参照すること。

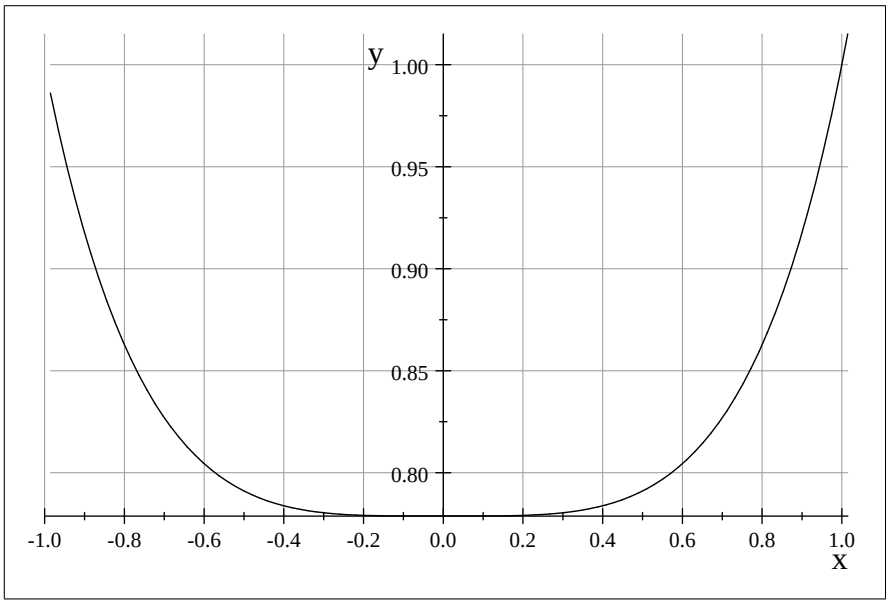

図 2: 真の値

この簡単な例の中では、二次の (通常の) 多項式を用い、適当にとった三点 (-1/2,0,1/2) でのみ真の解と近似解が一致するという条件を用い、線形連立方程式を用いて近似式を導いた。ここで

(1)　通常の多項式ではなく、他の多項式でより正確な近似をもたらもすものは存在するか?

(2)　なぜ、三点でのみ真の解と一致する (Collocation 法と呼ばれる) ような近似を用いたか?

(3)　どのようにして近似点を決定するのか?

の三つの点について吟味したい。

答えをラフに書くと、次のようになる。

(1)　通常の多項式 $\left(1, x, x^2, x^3, x^4, ..., x^n, ...\right)$ は、近似のベース (基底) としては極めて危険であり非効率である。そもそもこの基底は直行していない。周期関数であれば Fourie 級数が、非周期関数であれば Chebyshev Polynomial あるいは Spline 関数による近似が効率的である。

(2)　Collocation は極めて単純かつ効率的な手法である。

(3)　例のような Even Points を使用することは危険であり、Chebyshev Points を用いることが望ましい。

以下のセクションでは、上記のことを簡単にであるが解説していく。

2.1 Chebyshev Polynomial

n 次の Chebyshev Polynomial $T_n(x)$ とは、区間 [-1,1] で定義された、下記の性質をみたす多項式のことである。

$$T_n(x) = \cos n\theta$$
$$x = \cos\theta$$

高校時代に学んだ加法定理を思い出すと

$$cos(\alpha \pm \beta) = cos(\alpha)\,cos(\beta) \mp sin(\alpha)\,sin(\beta)$$

$$\sin 2\theta = 2\cos\theta\sin\theta$$
$$\cos 2\theta = \cos^2\theta - \sin^2\theta$$

従って、

$$\cos 2\theta = 2(\cos\theta)^2 - 1$$

$$\begin{aligned}\cos 3\theta &= cos(2\theta)\,cos(\theta) - sin(2\theta)\,sin(\theta)\\ &= \left(2(\cos\theta)^2 - 1\right)cos(\theta) - 2\cos\theta(\sin\theta)^2\\ &= 2(\cos\theta)^3 - cos(\theta) - 2\cos\theta\left(1 - (\cos\theta)^2\right)\\ &= 4(\cos\theta)^3 - 3\cos\theta\end{aligned}$$

この加法定理を用いて具体的に Chebyshev Polynomial を書き下すと

$$T_0(x) = 1$$

$$T_1(x) = x$$

$$T_2(x) = 2x^2 - 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

Chebyshev Polynomial のもう一つの定義は下記の漸化式で与えられる。

$$\begin{aligned} T_{n+1}(x) - 2xT_n(x) + T_{n-1}(x) &= 0 \\ T_0(x) &= 1 \\ T_1(x) &= x \end{aligned}$$

漸化式による定義は極めて便利であり、実際に計算するときには、上記の漸化式から任意の次数の Chebyshev Polynomial を作ることが出来る。しかしながら、三角関数による定義の意味は自明ではない。実は Chebyshev Polynomial とは、Fourie 級数 (cosine 級数) の変数変換であり、上記の三角関数の定義はその変数変換を表している。

2.2 Fourie 級数展開 (1)

Fourie 級数展開とは、ある周期を持った関数を三角関数を用いた多項式で近似する一つの手法である[3]。具体的には書きのような多項式を考える。

$$F(x) = a_0 \cos 0x + a_1 \cos x + a_2 \cos 2x + ... + b_0 \sin 0x + b_1 \sin x + b_2 \sin 2x + ...$$

なぜこのような関数形を考えるかというと、$\cos nx$ と $\sin nx$ が $[0,\ 2\pi]$ で直行関数形を作るためである。すなわち

$$\int_0^{2\pi} \sin nx dx = \int_0^{2\pi} \cos nx dx = \int_0^{2\pi} \sin nx \cos mx dx = 0$$

直行関係は、近似を考える際に極めて重要である。通常の多項式は直行関係にない。例えば、x と x^2 は、[0 1] 区間で直行していない。このときの問題点は線形回帰分析における多重共線性を想像するとよい。二つの説明変数が極めて似た動きをしていると、推計パラメターは非常に不安定になる。これは、二つの説明変数が同じ方向に動いているため、両者の効果の差をデータから知ることが難しいことを意味する。線形空間を張る時の基底が、直行していれば、非常に安定した張り方ができる、ということでもある。

2.2.1 ベクトル (線形) 空間と基底の復習

ここで、一般教養で学習したと思われる、ベクトル空間と基底について、もう一度整理してみよう。

(定義)

要素 $x, y, z, \cdots$ を含む、非空集合 L が、次の条件を満たす時、ベクトル空間、あるいは線形空間という。

[3]周期を持たない関数の近似も可能であるが、粗すぎて使いもにならない。周期関数は Fourie で、非周期関数は Chebychev や Spline で近似するのが効率的である。

I　任意の二要素、x,y$\in L$ に対し、第三の要素、$z=x+y$ が一意に定まる。なお、和は、次の性質をもつ。

(1)　$x+y=y+x$

(2)　$x+(y+z)=(x+y)+z$

(3)　全ての要素、$x\in L$ に対し、$x+0=x$ なる要素 $0\in L$ が存在する。

(4)　全ての要素、$x\in L$ に対し $x+(-x)=0$ となる要素 $-x\in L$ が存在する。

II　任意の数 a および $x\in L$ に対して要素 $\alpha x\in L$ が定まり、これについて、次の関係が成立する。

(1)　$\alpha(\beta x)=(\alpha\beta)x$

(2) $1x=x$

III　和と積は分配法則により結び付けられている。

(1)　$(\alpha+\beta)x=\alpha x+\beta x$

(2)　$\alpha(x+y)=\alpha x+\beta y$

なお、ここで用いられている「数」が実数の時、実線形空間、複素数を含む時は複素線形空間という。

ベクトル空間の例は多い。例えば、実数の集合はベクトル空間である。より重要な例は、区間 [a, b] で定義された連続関数の集合もまた、ベクトル空間となる。

(線形独立と基底)

線形空間 L の要素、x,y,.., w に対し、少なくとも一つがゼロでない数、$\alpha,\beta,...,\lambda$ を選び、

$$\alpha x+\beta y+...+\lambda w=0$$

とすることができるとき、これらの要素は一次従属であるという。一次従属でない要素は一次独立であるという。定義より、一次独立のときは、

$$\alpha=0,\beta=0,...,\lambda=0$$

となる。

空間 L に n 個の一次独立の要素が存在し、任意の n+1 個の要素をとると、これらが必ず一次従属となるとき、L の次元は n であるという。n 次元空間における一次独立な n 個の要素を基底と呼ぶ。

(定理)

L を有限次元のベクトル空間とする。また、$\{x_1,x_2,...,x_n\}$ をこのベクトル空間の基底であるとする。このとき、任意の一要素 $x\in L$ は、基底と数の線形結合

$$\alpha_1x_1+\alpha_2x_2+...+\alpha_nx_n=x$$

によって、ただ一通りあらわすことができる。

この定理により、基底を一つ定めると、ベクトル空間に含まれる任意の要素は、基底の線形結合により表現することが可能となる。または、基底により、ベクトル空間全体を張ることができるとも言う。

二次元の実数値空間 R^2 を考えよう。そこで、二つのベクトル $(0,1)$ と $(0.0000010000010, 0.9999999999995)$ を考える。二つのベクトルは線形独立であるから、全ての二次元上の点は、この二つのベクトルの線形結合で表わすことが可能である。しかし、実際には、そのようなことを試みる者はいないだろう。二つのベクトルの向きは、あまりにも似すぎていて、線形結合をとるときに必要なパラメターは、非常に大きな値になり、かつ、わずかな違いが大きなパラメターの違いを作り出してしまうのである。これは、線形回帰をするときに、説明変数間で相関が高すぎると、推計量が安定しないことに似ている。二次元の要素を二つのベクトルの線形結合で表わす時に、最も自然なのは、x 軸と y 軸、すなわち (1,0) と (0,1) の線形結合で表記することである。数学的には、線形独立である限り、基底として機能するが、実際に計算するときには、基底に含まれる要素が、ある程度離れているほうが良い。具体的には、二つのベクトルが、全く違う方向を向いていて欲しいのである。そこで、次に、ベクトル空間に角度を導入することにする。

(計量ベクトル空間)

L をベクトル空間とする。$L \times L$ から R への写像 (ベクトル空間上の二要素を定義域とし、実数値を値域とする関数)(x, y) が与えられ次の性質をみたすとき、この写像を L の内積という。

(1) $(x, x) \geq 0$　等号が成り立つのは、x=0 のときのみ。

(2) $(\alpha x + \beta x', y) = \alpha (x, y) + \beta (x', y)$

(3) $(x, y) = (y, x)$

(ノルム)

任意の要素 $x \in L$ に対し、(x, x) を $\|x\|$ で表し、x のノルムと呼ぶ。このノルムに対し、下記の不等式 (シュワルツ) が成立する。

$|(x, y)| \leq \|x\| \, \|y\|$

(角度)

シュワルツの不等式より、常に

$$-1 \leq \frac{(x, y)}{\|x\| \, \|y\|} \leq 1$$

そこで、

$$\cos(\theta) = \frac{(x, y)}{\|x\| \, \|y\|}$$

で定義される θ を x と y のなす角と呼ぶ。

(直交性)

$(x, y) = 0$ のとき $(\cos(\theta) = 0)$、二つの要素 (ベクトル) は直交するという。

内積を導入することにより、ベクトル空間に直交性という性質が導入されるのである。

(正規直交基底)

L を計量ベクトル空間とする。L の基底 $\{e_1, e_2, ..., e_n\}$ が以下の性質をもつ時、正規直交基底であるという。

(1)　任意の i について、$\|e_i\| = 1$

(2)　$i \neq j$ なら、$(e_i, e_j) = 0$

(関数空間の内積)

区間 [a,b] で定義された関数 $f(x)$ と $g(x)$ の内積 (f, g) は重み関数 $w(x)$ を用いて、下記のように作ることができる。

$$(f, g) = \int_a^b f(x)\, g(x)\, w(x)\, dx$$

ただし、$w(x)$ は開区間 (a,b) 内で有限個の点を除いて正の値をとる連続関数であり

$$-\infty < \int_a^b w(x)\, dx < \infty$$

とする。例えば、

$$(f, g) = \int_a^b f(x)\, g(x)\, dx$$

もまた内積であるし、

$$(f, g) = \int_{-1}^{1} f(x)\, g(x)\, \frac{1}{\sqrt{1 - x^2}} dx$$

も内積である。

本節で復習したのは

(1) 連続関数の集合は、ベクトル空間を作り、(2) そこに正の値をとる重み関数と積分を用いて内積と角度を導入することが可能であり、(3) 直交基底を導入することが出来る、という三点である。

2.3 Fourie 級数展開 (2)

Fourie 級数展開に関しては多くのことが知られている。例えば、不連続点が多くない限り、Fourie 級数展開は任意の周期関数に収束させることが可能である。Chebyshev Polynomial とは、この Fourie 級数展開におけるサイン (sin) の要素を落とし、$x = \cos\theta$ と変数変換したものに等しい。これにより、周期関数に対して適用されていた Fourie 級数展開の収束定理を非周期関数にも適用可能としたものが Chebyshev Polynomial である。Chebyshev の背後には周期関数があることは常に念頭においておく必要がある。元の Fourie 級数が直行関数形であったことから、Chebyshev も直行多項式となる。重み関数を $\frac{1}{\sqrt{1-x^2}}$ とした、積分により内積を定義すると、

$$\int_{-1}^{1} \not{T}_n(x)\ \not{T}_m(x)\ \frac{1}{\sqrt{1-x^2}}dx = 0 \quad \text{if } m \neq n$$
$$= \pi \text{ if } m = n \neq 0$$
$$= \frac{\pi}{2} \text{ if } m = n = 0$$

したがって、$\not{T}_n(x)$ により多項式空間の直行基底を作ることが出来、任意の多項式を近似することが可能となる。また、Boyd(2001) に説明されている収束定理を用いることにより、ある条件下で任意の関数を $\not{T}_n(x)$ の加重和により表現することができる。

2.4 Chebyshev Minimal Amplitude Theorem

Chebyshev Polynomial が関数近似で頻繁に用いられる根拠は下記の定理である。

定理 1 定理 *2* 定理 *3* 全ての次元 *n(n* 次項の係数が *1* に限る*)* の多項式の中で、*[-1,1]* 区間内で最小の最大値を有するものは $T_n(x)/2^{n-1}$、すなわち n 次の *Chebyshev Polynomial* を 2^{n-1} で除したものである。すなわち、任意の n 次多項式 $\not{P}_n(x)$ で、n 次項の係数が *1* であるものは下記の不等式を満たす。

$$\max_{x\in[-1,1]} |P(x)| \geq \max_{x\in[-1,1]} \left|\frac{T_n(x)}{2^{n-1}}\right| = \frac{1}{2^{n-1}}$$

ある関数を n 次の Chebyshev Polynomial で近似した場合、真の関数と近似された多項式の間には残差がある。その残差は n+1 次の Chebyshev Polynomial の形の残余項として書くことが可能である。良い近似であるためには、その残余項の大きさが小さくなって欲しい。そこで、最大の誤差を最小化するには、Chebyshev Polynomial を使うことが望ましいことになる。ま

た、n+1 次の ChebyshevPolynomial のゼロ点を求め、その n+1 個のゼロ点において近似式と真の式の値を一致させることで、残余項を (その点においては) ゼロにすることができる。したがって、近似点は、n+1 次の Chebyshev Polynomial のゼロ点で与えることが望ましい。n 次の Chebyshev Polynomial で多項式を近似するときは、n+1 次の Chebyshev Polynomial のゼロ点を用いるということである。

2.5 Chebyshev Nodes の重要性

n 次の Chebyshev Polynomial を用いて真の関数との近似を試みるとき、評価点 (node) として、n+1 次の Chebyshev Polynomial のゼロ点を用いるべし、というのが前節の結果であった。そうではなく、even spaced 点を用いるとしたら、どのような現象が生じるであろうか?直感的には、node の数を増やせば増やすほど、スピードはともかく、真の値に収束していくような印象を持つだろう。node を増やすほど未定係数の数が増えるわけで、50 次の多項式で近似すれば、3 次の多項式よりも良い近似になりそうな気はするが、実際にはそうならないことがある。有名な Runge の例は

$$f(x) = \frac{1}{1+25x^2}$$
$$x \in [-1, 1]$$

である。これを 10 点の均等間隔の node により Chebyshev Polynomial で近似したものが図 3 である[4]。

まず、10 点という少数で近似しているため粗い近似となっていること、コサインカーブで近似しているため、妙な周期が生じていることがわかるが、全体的にはそれほど間違った近似にはなっていないことがわかる。

これを 50 点に増加させると図 4 となる。

図 3 の縦軸のスケールに注目すると、とてつもなく大きな値になっていることがわかる。このことからわかるように、nodes を増やすことにより、かえって近似の精度が極端に低くなってしまうケースが存在するのである。

50 の node による collocation を行うということは、50 点において近似式と真の値が一致していることを意味する。[-1,1] 区間で 50 回も交差しながらも、全体の形状が大幅に異なるというのは驚きである[5]。

近似の形状をみると、両端点で大きく乖離していることがわかる。数学的には、級数の収束定理がこの数式には適用されないために生じる現象であるが、とにかく近似が必要な場合は、誤差が大きいところに、より多くの nodes を用

[4]オリジナルの関数形は $y = \frac{1}{1+x^2}$ で、[-5, 5] の区間で定義されている。

[5]ちなみに Runge の論文は 1900 年代に書かれている。コンピュータのない時代にこういうことを発見しているのだからなおさら驚きである。

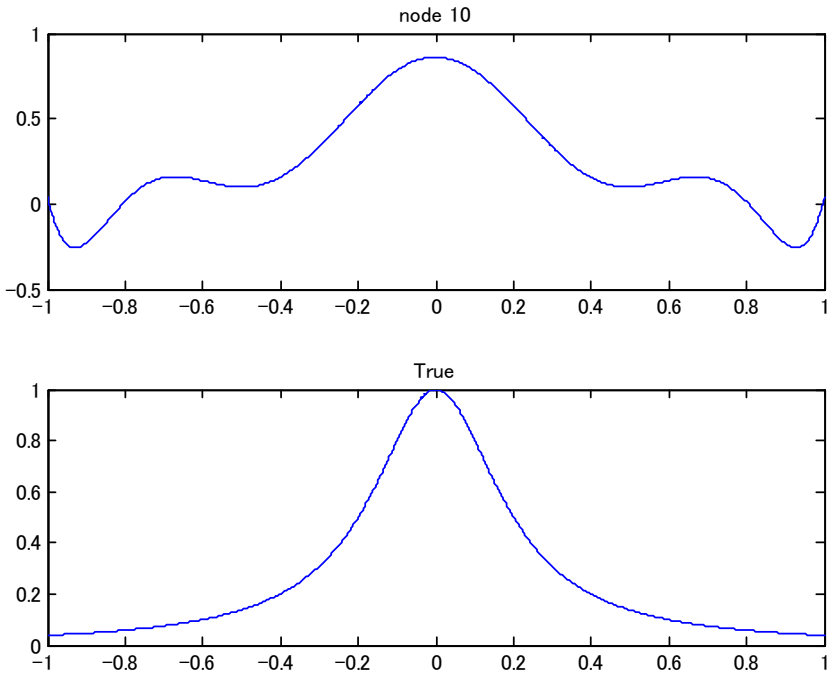
node 10
1
0.5
0
−0.5
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
True
1
0.8
0.6
0.4
0.2
0
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

図 3: 均等間隔 (10)

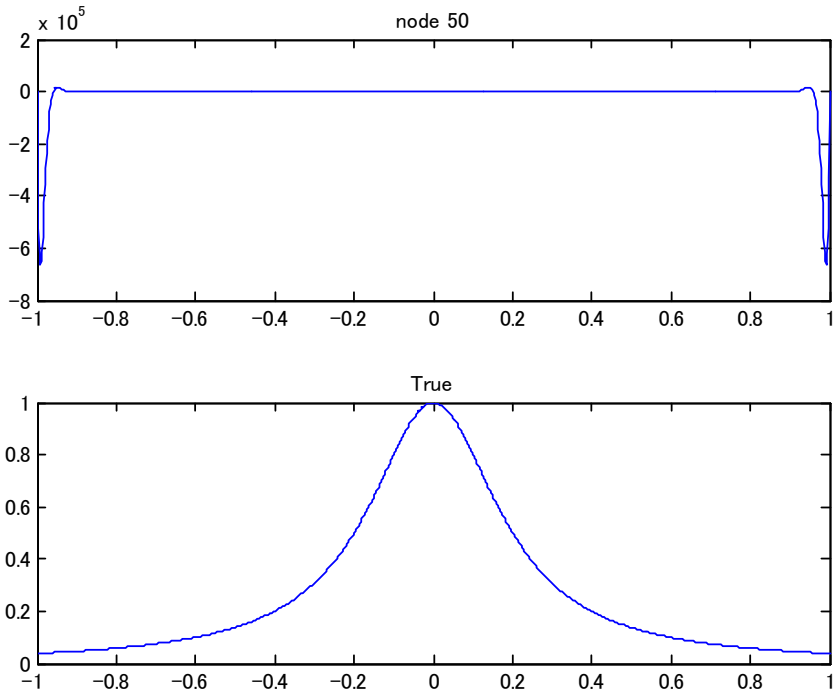

図 4: 均等分割 node =50

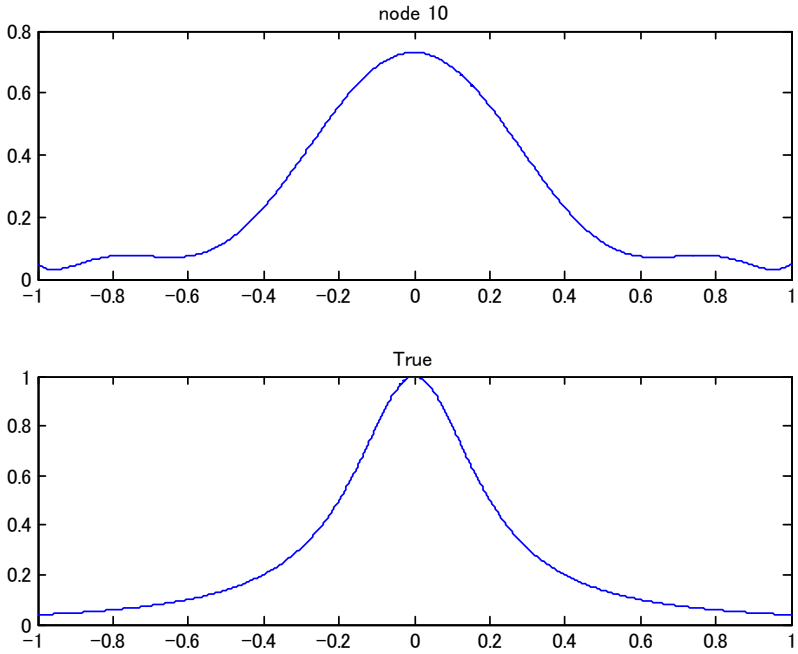

図 5: Chebyshev Node (10)

いることによりこの現象を回避できないかと考えるのが自然であろう。10 に均等分割した場合の nodes は (01,-0.77,- 0.55, -0.33, -0.11, 0.11, 0.33, 0.55, 0.77, 1) であるが、10 次の Chebyshev Polynomial のゼロ点、すなわち Chebyshev Nodes は (-0.98,- 0.89, -0.70, -0.45, -0.16, 0.16, 0.45,0.70, 0.89,0.98) であり、両端点を除き、周辺により多くの nodes を確保している。

10 点の Chebyshev Nodes を用いて、Runge 関数を近似すると、図 5 になる。

n=50 の Chebyshev nodes を採用した場合は、図 6 であらわされる。この場合は、近似と真の関数形はほぼ一致している。

3 Spline 補間

Chebyshev Polynomial による近似は、関数全体を近似する多項式を探すものであったが、より local な情報を重視し近似する手法が spline 補間である。ここではよく用いられる Cubic Spline を Judd (1998) に従い説明する。

いま、$\{(x_i, y_i)\,|i = 0, 1, .., n|\}$ となる (x, y) の集合があるとする。我々は、y を x の関数と考え、これを描写する多項式が欲しいわけだが、spline の場合は、全体を一つの多項式で近似するのではなく、全体の区間を小区間に分

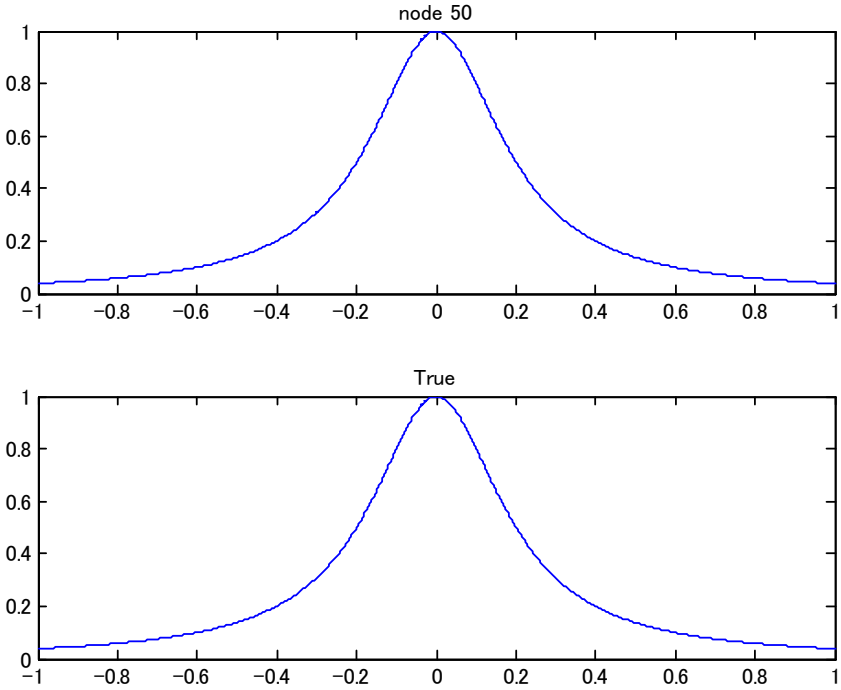

図 6: Chebyshev Nodes (50)

割し、それぞれが 3 次の多項式で近似されると考える。

spline function $s(x)$ は、

$$s(x_i) = y_i$$

すなわち、各点を結ぶ線である。

次に、cubic spline であれば

$[x_{i-1}, x_i]$ において、

$$s(x) = a_i + b_i x + c_i x^2 + d_i x^3$$

とかける。ここで、係数が点 i に依存していることに注意せよ。n+1 個の点があれば、n 個の小区間が出来、n 本の 3 次多項式が作られ、4n 個の未定係数が発生する。

そこで、各点において spline が y の値と一致するという条件 (n+1), 各 spline が繋がるという条件 (n-1)、および各点において spline が微分可能になっている (隣接してる spline が同じ導関数を持つ) という条件 (n-1)、二階微分が一致するという条件 (n-1) により、4n 個の未定係数に対して 4n-2 本の制約を課すことが出来る。残りの二つの条件をどう課すかにより、様々な spline を考えることが出来る。natural spline では、両端点の微分係数をゼロとおくことで、全ての未定係数を特定化する。

Chebyshev Polynomial の節で説明したように、Chebyshev には最小誤差の特徴があり、Spline よりも優れた性質を持つが、実際には Spline の方が望ましい結果をもたらすことも多い。

3.1 Spline と Chebyshev の比較

まず、Runge の例で Spline と Chebyshev の違いをみてみよう。

図 7 は均等間隔 node を用いた場合の Cubic Spline と Chebyshev 近似の比較である。Chebyshev が波打っているのに対し、Spline は真の形状に近いことがわかる。

図 8 は、Chebyshev node に変更したものである。Spline ではそれほど変化はないが、Chebyshev では大幅な改善が観察される。

図 9 は Chebyshev Node を 20 に増加させたものであり、両近似ともにほぼ正確なものとなっている。

Runge の関数は微分可能な単純なものであったが、次に下記のような微分不可能な点を持つ関数を考えよう。

$$\begin{aligned} y &= 2 - x \text{ if } x \leq -0.5 \\ &= x \text{ if } x > 0.5 \end{aligned}$$

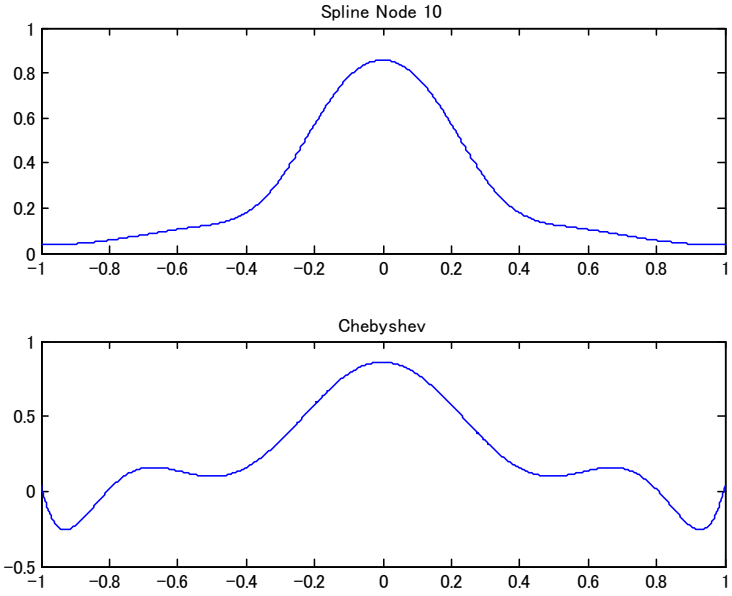

図 7: Node =10　Spline vs Chebyshev

これは、x=0.5 で不連続となる関数であり、そもそも連続関数で近似するには向いていないものである。

これを 20 の Chebyshev node で近似すると、図 10 のようになる。

20 の node では両近似ともに不連続点の再現に成功していない。図 11 は 50、図 12 は 100 の node で計算したものである。

20 に比べたら大分真の関数形に近づいたが、まだまだである。

100 に増やすと、どちらも真の関数に近い形状となるが、Chebyshev の方は小さい波を沢山作り出しているのに対し、Spline は不連続点の近傍以外ではほぼ正確な近似となっている。

ちなみに、500 まで増やすと図 13 となる。

やはり、Chebyshev では、小刻みな振動が生じていることがわかる。一方、spline も不連続点の近くでは誤差が発生しているが、その大きさは決して大きいものではない。

以上の例から我々は何を学べるだろうか?

Chebyshev Polynomial は Chebyshev Node と組み合わせることで、最大誤差を最小化させるという望ましい性質を持つ。しかし、最大誤差の最小化は、必ずしも常に best fit をもたらすわけではない。Chebyshev 多項式の定義からわかるように、Chebyshev は振動を引き起こしやすく、特に微分不可

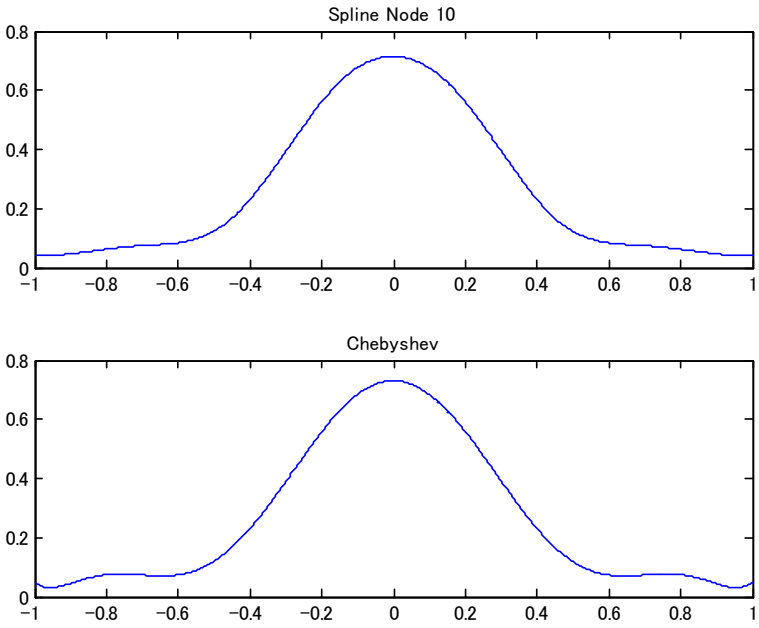

図 8: Chebyshev Node n=10

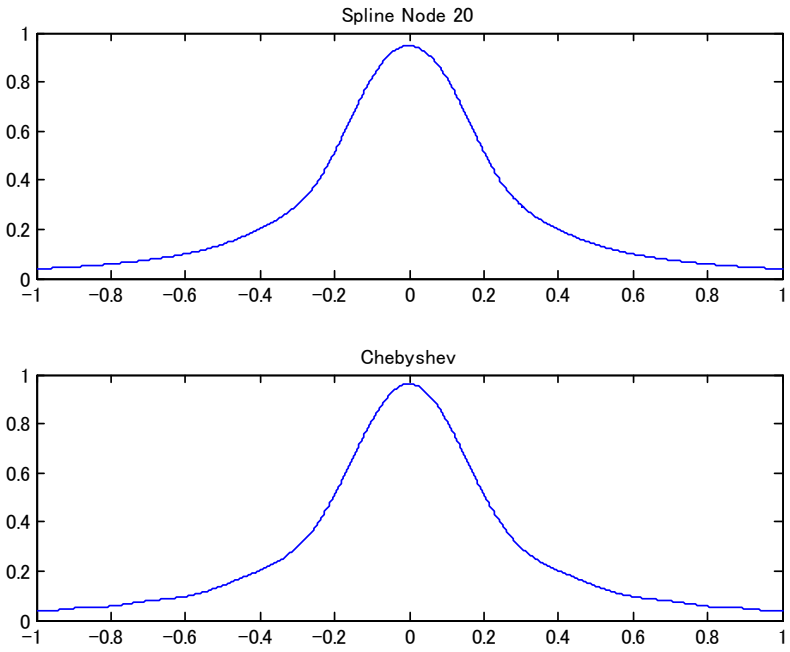

図 9: Chebyshev Node n=20

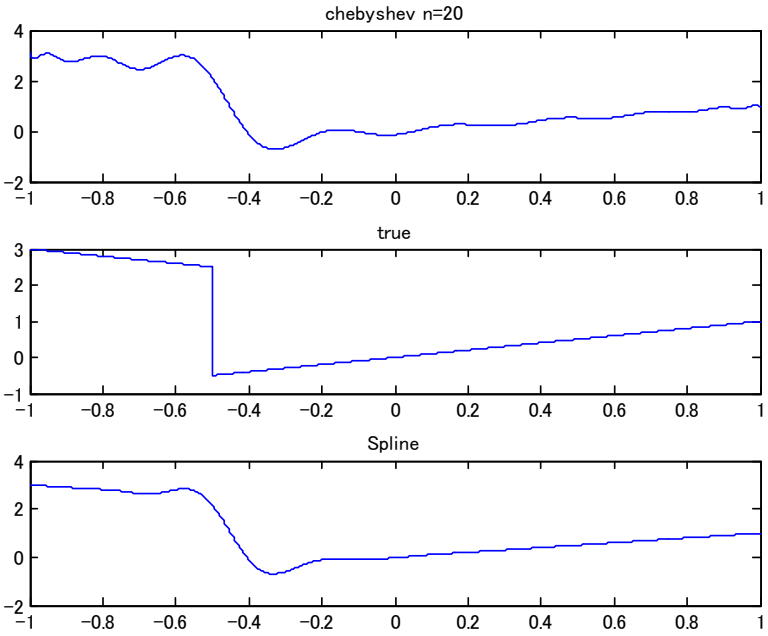

図 10: Chebyshev Node n=20

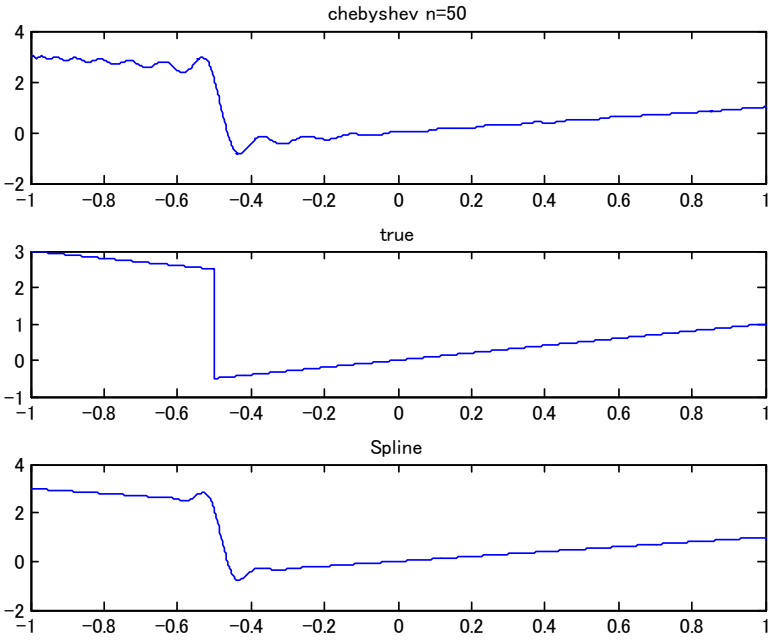
chebyshev n=50
4
2
0
−2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
true
3
2
1
0
−1
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
Spline
4
2
0
−2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

図 11: Node =50

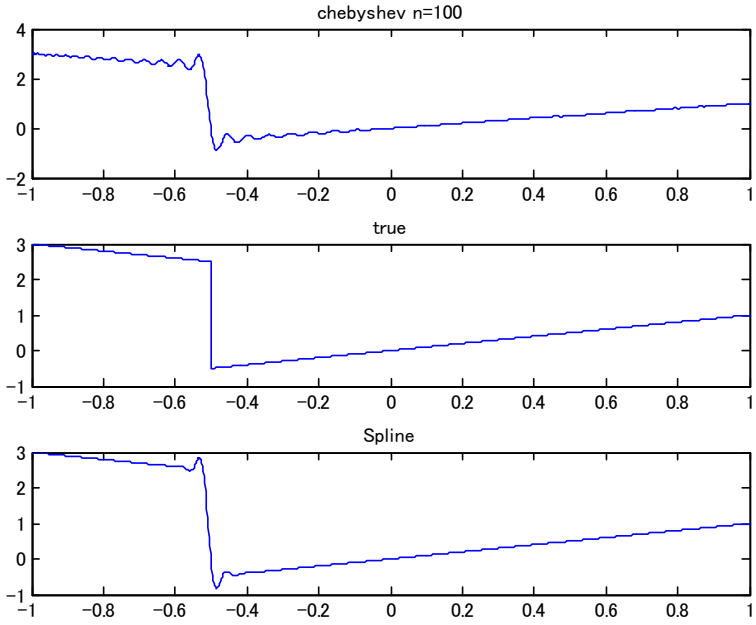
chebyshev n=100
4
2
0
−2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
true
3
2
1
0
−1
Spline

図 12: Node =100

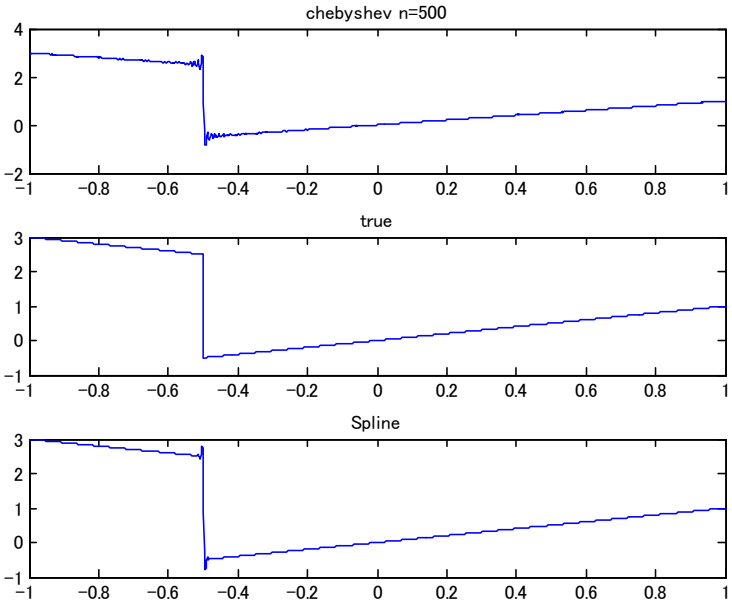

図 13: Node =500

能な点や不連続点でその傾向が強くなる。Spline もまた、不連続関数の近似は不得手であるし、Chebyshev のような数数学的に望ましい性質を持つものではない。しかし、妙な振動が発生することは少なく、全般的な形状の再現という点では、今まで見てきた例に限っては Chebyshev よりも良好な近似となっている。

ここで扱った不連続関数の例は、極めて極端なケースとなっている。Value Function が図のような不連続点を持つことはほとんどないし、Policy Function も微分可能ではないケースにたまに遭遇することはあっても、不連続になることは少ない。そもそも、二階微分可能な spline 関数で不連続関数を近似しようとすることそのものが無謀ともいえる。それでも、spline による近似はそれほど悪いものではない。

Chebyshev Node は、Chebyshev Polynomial を使うという前提で、また真の関数に関する先見的情報が何もない状況において、最大誤差を最小化させるものであったが、もしも我々が真の関数に関して何かしらの情報をあらかじめ有しているときには、何も Chebyshev node に固執する必要はない。不連続点の位置をある程度予測できるのであれば、その近傍の node を増加させることで近似精度を上げることは可能であるし、また線形となっている領域で沢山 node をとっても付加される情報は少ない。近年では、Chebyshev よりも spline を使う論文が増加している印象がある。

4 Chebyshev Polynomialを用いた最適成長モデルの解法

Chebyshev Polynomial は区間 [-1,1] で定義されていたが、Value Function や Policy Function の定義域としては、より一般的な区間である必要がある。我々の関心のある区間が [a,b] であるときは、これを [-1,1] に縮小 (拡張) させて Chebyshev node を計算し、それをまた元の区間に対応するように拡張 (縮小) させるという一手間を入れる必要がある。

ここで解く問題は

$$\max_{\{c_t,k_{t+1}\}_o^\infty} \sum_{t=0}^{\infty} \beta^t \log c_t$$

$$k_{t+1} = Ak_t^\alpha - c_t \tag{2}$$

というものであり、真の Value Function は下記のように closed form で解く事の出来る特殊ケースを考える。

$$V(k) = \frac{1}{1-\beta}\left(\ln\left(A(1-\alpha\beta) + \frac{\alpha\beta}{1-\alpha\beta}\ln(A\alpha\beta)\right)\right) + \frac{\alpha}{1-\alpha\beta}\ln k \tag{3}$$

なお、下記のコードを実行するには Miranda and Fackler(2002) による ComeEcon Toolbox が必要である[6]。

```
%
% Numerical Dynamic Programming by Chebyshev
%
%============================================================
% THE MODEL;
% max sum_ {t=0} ^{inf} beta^t * u(c_t)
% s.t. c_t + k_ {t+1} = f(k_t)
% k_0 given
%
% -log utility function, without leisure,
% -production function is Cobb-Douglas:
% y_t = AA * k_ {t} ^{alpha}
% AA is NOT productivity, which only adjusts a steady state value of
% capital equal to 1.
%
```

[6] この Toolbox は web で Free で入手可能である。ただ、最初に C Complier(無料で利用可能) をインストールし、その上で mexall を実行しないと使えない。Toolbox についている Readme をよく読んでから作業すること。

% v(k) = max{log(AA * k^alpha - kprime) + beta * v(kprime)}

%

% We need Compecon Toolbox by Miranda and Fackler (2002)

% Program based on Tomoaki Yamada (2002)

% Written by Naohito Abe 2009 Dec

%===

clear all;

% format long;

% diary NDP1.out;

global AA alpha beta mink maxk

% set parameter values

%——————————

alpha = 0.25; % production parameter

beta = 0.96; % subjective discount factor

AA = 1./(alpha*beta); % modify steady state capital = 1

mink = 0.03; % minimum value of the capital

maxk = 2.0; % maximum value of the capital

tol_golden = 10^-5; % tolerance of the error for the Golden Search

maxit = 300; % maximum # of iteration

TOLV_cheb = 10.^-4; % stopping criterion of value iteration (for Chebyshev)

TOLP_cheb = 10.^-7; % stopping criterion of policy iteration

diff = 100000; % initial difference

% TOLV = 10.^-6 でも 350 回位の iteration で収束する

% INITIALIZATION %

% v^を Chebyshev により近似

%

% Approximation grid X^を決定。

% Initial Guess, vHAT を選択。

% Stopping Criterion を決定。上で既に決定済み。

%===

% initial guess of coefficients a0

%——————————————————-

n = 50; % degree of approximation (by Chebyshev interpolation)

m = 50; % the number of evaluation points. Note that m >= n.

% generate evaluation points, which is a column vector of capital grid.

evalpt = chebnodes(m); % generate m Chebyshev nodes defined in [-1,1].

kap_cheb = scaleup(evalpt,mink,maxk); % linear transformation of interval from [-1,1] to [mink,maxk]

V0 = zeros(m,1); % initial guess of value function.

fspace_cheb = fundefn('cheb',n,mink,maxk);

a0 = funfitxy(fspace_cheb,kap_cheb,V0);% initial Chebyshev polynomial for the value function

% %
% %
% % % % % % % % % % % % % % % % % % % %

% %

% CHEBYSHEV POLYNOMIAL INTERPOLATION %

% %

% %
% %
% % % % % % % % % % % % % % % % % % % %

% Preparation for iteration (Chebyshev)

%————————————————

a_cheb = a0;

Vold_cheb = V0; %10000*ones(m,1); % initial V_old

Vdyn_cheb(:,1) = Vold_cheb; % V の dynamics を見たるため (algorithm とは無関係)

it_cheb = 1;

diff_cheb = diff;

Pold_cheb = zeros(m,1);

converge_cheb = 0;

% Main Loop (Chebyshev Approximation)

%————————————————

tic

while it_cheb <= maxit & diff_cheb > TOLV_cheb;

V_cheb = ones(m,1);

POL_cheb = ones(m,1);

optset('golden','tol',tol_golden); % 関数 golden の tolerance を変更

% Maximization Step (STEP 1):

%————————————-

for i = 1:length(kap_cheb);

cap = kap_cheb(i); % current(i-th iteration) grid

cap_ini = kap_cheb(i); % initial guess for quasi-newton

% subroutines for maximization

%————————————

```
  % <golden section search>
  % を使って、各 grid 上における Bellman's eq の右辺を計算する。
  [POL_cheb(i),V_cheb(i)] = golden('Bellman_cheb1',mink,maxk,cap,a_cheb);
% GOLDEN SEARCH
  % 一回目の iteration が単なる log(kˆalpha - kprime) の最大化になってし
まい、
  % 定義域上に入りきれない可能性がある点に注意!!!
  end
  %
  % Fitting Step (STEP 2):
  %-----------------
  anew_cheb = funfitxy(fspace_cheb,kap_cheb,V_cheb);
  diff_cheb = max(abs(Vold_cheb - V_cheb)); % V の誤差
  Vdyn_cheb(:,it_cheb+1) = V_cheb;
  Pdyn_cheb(:,it_cheb) = POL_cheb; % 政策関数のダイナミクス
  DIF_cheb(it_cheb) = diff_cheb; % 誤差の推移をチェック
  ACHEV(:,it_cheb) = anew_cheb; % 係数 a の推移をチェック
  Vold_cheb = V_cheb;
  a_cheb = anew_cheb; % update coefficients a.
  it_cheb = it_cheb + 1;
  % 収束基準を Policy Function で計る Case:
  if max(abs(POL_cheb - Pold_cheb)) < TOLP_cheb;
  converge_cheb = 1;
  break;
  end
  Pold_cheb = POL_cheb;
  end
  Time_cheb = toc;
  % If policy function has been converged, calculate the value function
  %-------------------------------------------------
  if converge_cheb == 1;
  for i = 1:length(kap_cheb);
  V_cheb = Bellman_cheb1(POL_cheb,kap_cheb,a_cheb);
  end
  end
  % Calculate the true and approximated policy function
  %=================================================
  % policy function approximated by Chebyshev polynomial
  %---------------------------------
```

```
Cons_cheb = AA * kap_cheb .^alpha - POL_cheb;
% Calculate exact policy function at Chebyshev node
%—————————————————
exact_cheb = (1-beta*alpha) * AA * kap_cheb.^alpha; % true policy fcn
% % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % %
% %
% ACCURACY CHECK %
% %
% % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % %
% Step 1:
% value function が収束している事を確認
%—————————————-
iterat_cheb = [1:it_cheb-1];
% 下で、iteration を通じた diff の変化を plot.
% % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % %
% %
% RESULTS %
% %
% % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % %
% Display the results
%———————
disp('Numerical Dynamic Programming');
disp('');
disp('PARAMETER VALUES');
disp('');
disp(' alpha beta ');
disp([ alpha beta ]);
disp('');
disp(' mink maxk ');
disp([ mink maxk]);
disp(' tol_golden TOLV_cheb diff_cheb');
```

```
disp([ tol_golden TOLV_cheb diff_cheb]);
disp(' Calculation Time ');
disp(' Chebyshev ');
disp([ Time_cheb ]);
disp('');
% Figure (CHEBYSHEV INTERPOLATION)
%===================================
% Plot Value Function approximated by Chebyshev Polynomial
%————————————————————
subplot(2,2,1)
plot(kap_cheb,V_cheb,'b');
title('VALUE FUNCTION BY CHEBYSHEV');
xlabel('Capital'); ylabel('Value');
% Plot Consumption Function approximated by Chebyshev Polynomial
%——————————————————————
subplot(2,2,2)
plot(kap_cheb,exact_cheb,':',kap_cheb,Cons_cheb,'b')
title('CONSUMPTION FUNCTION BY CHEBYSHEV')
xlabel('Current Capital'); ylabel('Consumption');
legend('True Function','Approximate ')
% Plot Policy Function approximated by Chebyshev Polynomial
%————————————————————-
subplot(2,2,3)
plot(kap_cheb,POL_cheb,'b',kap_cheb,kap_cheb,'k')
title('SAVING FUNCTION BY CHEBYSHEV')
xlabel('Current Capital'); ylabel('Next Capital');
legend('Saving','degree of 45')
% Figure (Convergence of Value Function)
%=========================================
subplot(2,2,4)
plot(iterat_cheb,DIF_cheb);
title('Convergence Test (Chebyshev)');
xlabel('iteration'); ylabel('error');
```

以下の二つは上記の main に必要な関数である。

```
% % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % %
function [value,fjac] = Bellman_cheb1(kprime,kap,a);
% [value fjac] = Bellman_cheb1(kprime,kap,a)
% Purpose(OUTPUT):
```

```
% Calculate the value function and Jacobian of value.
% V_j = {log(AA*k.^alpha - kprime) + beta * v(krpme)}
%
% INPUT: kprime is choice variables(column vector).
% kap is current state variable(column vector).
% a is coefficients of Chebyshev polynomial, which
% approximate next period's value function.
%
% When output is only one, this function returns value,
% not return Jacobian of value function.
%
% November 25, 2002
%
% Written by T.Y.
global AA alpha beta mink maxk
% evaluate the value function at current state capital.
value = valuefcn_cheb1(kprime,kap,a);
% calculate Jacobian of the value fcn (from CompEcon)
fjac = fdjac('valuefcn_cheb1',kprime,kap,a);

% % % % % % % % % % % % % % % % % % % % % % % % % % % % %

function value = valuefcn_cheb1(kprime,kap,a);
% value = valuefcn_cheb1(kprime,kap,a)
% Purpose:
% Calculate the value function, when current capital level
% is kap, next capital stock will be kprime, and
% value fcn is approximated by a (Chebyshev coefficients).
%
% kprime, kap and a must be column vector.
%
% V_j = {log(AA*k.^alpha - kprime) + beta * v(kprime)}
%
% November 25, 2002
%
% Written by T.Y.
global AA alpha beta mink maxk
numa = length(a);
numa1 = numa - 1;
cons = AA * kap .^alpha - kprime;
```

```
for i = 1:length(kprime);
if cons(i) <= mink;
cons(i) = mink;
end
end
utility = log(cons);
% utility = log(AA * kap .^alpha - kprime);
kprime11= scaledown(kprime,mink,maxk);
vprime = chebpoly(numa1,kprime11) * a;
value = utility + beta .* vprime;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

出力結果 (ちなみに CPU は Xeon 3.0GH、Matlab 6.1 である)

```
>> Numerical Dynamic Programming
PARAMETER VALUES
alpha beta
0.2500 0.9600
mink maxk
0.0300 2.0000
tol_golden TOLV_cheb diff_cheb
0.0000 0.0001 0.6992
Calculation Time
chebyshev
10.7660
```

Chebyshev よりも spline を使う論文が増加している印象がある。

5 Cubic Spline を用いた最適成長モデルの解法

ここでは、上記で用いた Chebyshev ではなく、spline で同じ問題を考える。ただ、50nodes ではなく、10nodes で計算している。50nodes の spline ではよりスムースな形状を得ることが出来るが、計算に 2 分程度かかる。

```
%
% Numerical Dynamic Programming by Cubic Spline Interpolation
%
%==========================================================
```

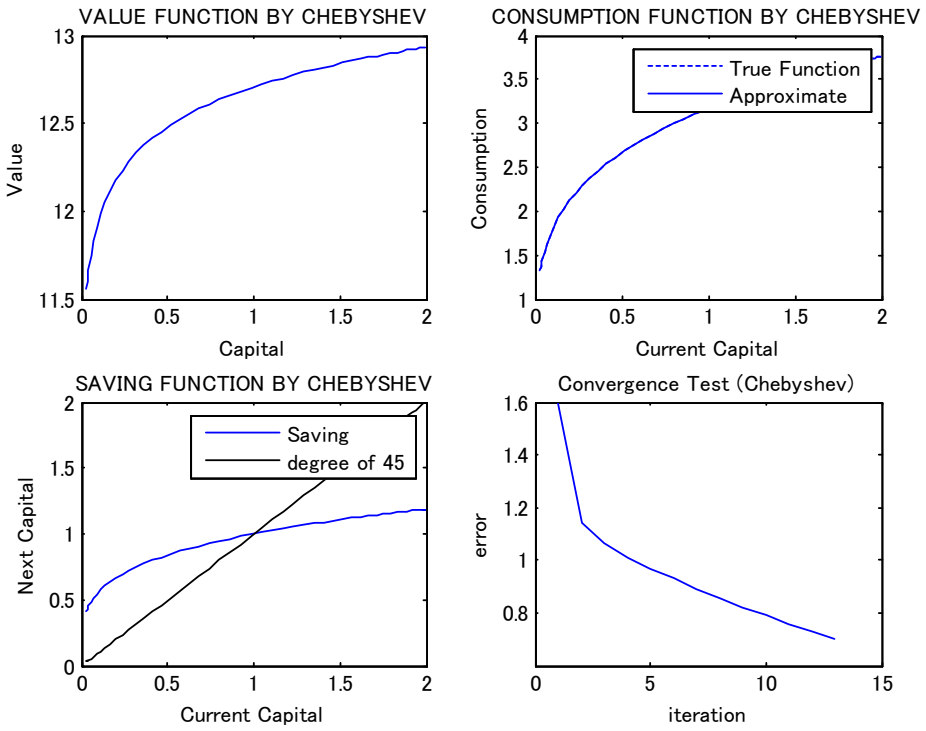

図 14: Chebyshev による近似

```
% THE MODEL;
% max sum_ {t=0} ^{inf} beta^t * u(c_t)
% s.t. c_t + k_ {t+1} = f(k_t)
% k_0 given
%
% -log utility function, without leisure,
% -production function is Cobb-Douglas:
% y_t = A * k_ {t} ^{alpha}
% AA is NOT productivity, which only adjusts a steady state value of
% capital equal to 1.
%
%
% v(k) = max{log(AA * k^alpha - kprime) + beta * v(kprime)}
%
%
% Program based on Tomoaki Yamada (2002)
% Written by Naohito Abe 2009 Dec
%=========================================================
clear all;
global AA alpha beta mink maxk
% set parameter values
%---------------
alpha = 0.25; % production parameter
beta = 0.96; % subjective discount factor
AA = 1./(alpha*beta); % modify steady state capital = 1
mink = 0.03; % minimum value of the capital
maxk = 2.0; % maximum value of the capital
tol_golden = 10^-5; % tolerance of the error for the Golden Search
maxit = 350; % maximum # of iteration
TOLV_spli = 10.^-4; % stopping criterion of value iteration (for Cheby-
shev)
TOLP_spli = 10.^-7; % stopping criterion of policy iteration.
diff = 100000; % initial difference
% v^の関数形を決定 -> 2) Cubic Spline Interpolation
% Approximation grid kap_cheb を決定。
% Initial Guess, vHAT=0(value function iteration なら OK!)
% stopping criterion VOLV_cheb を決定。
%=========================================================
% initial guess of coefficients a0
```

```
%-----------------------------------------
m = 10; % the number of evaluation points.
V0 = zeros(m,1); % initial guess of value function.
% Preparation for iteration (cubic spline)
%-----------------------------
kap_spli = linspace(maxk,mink,m)'; % capital grid in [mink,maxk] (等分割)
Vold_spli = V0; % initial old value function
Vdyn_spli(:,1) = Vold_spli; % V_spli の推移を見たい。
a_spli = spline(kap_spli,Vold_spli); % initial spline by Matlab function
it_spli = 1;
diff_spli = diff;
Pold_spli = zeros(m,1);
converge_spli = 0;
% Main Loop (Cubic Spline Interpolation)
%-----------------------------
tic
while it_spli <= maxit & diff_spli > TOLV_spli;
V_spli = ones(m,1);
POL_spli = ones(m,1);
for i = 1:length(kap_spli);
cap = kap_spli(i); % current grid
[POL_spli(i),V_spli(i)] = golden('Bellman_spli1',mink,maxk,cap,a_spli); % GOLDEN SEARCH
%
end
diff_spli = max(abs(Vold_spli - V_spli));
a_spli = spline(kap_spli,V_spli); % update the coefficients of spline
Vdyn_spli(:,it_spli+1) = V_spli;
Pdyn_spli(:,it_spli) = POL_spli; % 政策関数のダイナミクス
DIF_spli(it_spli) = diff_spli;
Vold_spli = V_spli;
% 収束基準を Policy Function で計る Case:
if max(abs(POL_spli - Pold_spli)) < TOLP_spli;
converge_spli = 1;
break;
end
Pold_spli = POL_spli;
it_spli = it_spli + 1;
```

```
end
Time_spli = toc;
% If value function has been converged, calculate the value function
%-------------------------------------------------
if converge_spli == 1;
for i = 1:length(kap_spli);
V_spli = Bellman_spli1(POL_spli,kap_spli,a_spli);
end
end
% Calculate the true and approximated policy function
%=================================================
% Calculate approximated policy function
%-------------------------
Cons_spli = AA * kap_spli .^alpha - POL_spli;
% Calculate exact policy function for comparison
%-------------------------------
exact_spli = (1-beta*alpha) * AA * kap_spli .^alpha; % true policy func-
tion
% Display the results
%-------------
disp('Numerical Dynamic Programming');
disp('');
disp('PARAMETER VALUES');
disp('');
disp(' alpha beta ');
disp([ alpha beta ]);
disp('');
disp(' mink maxk ');
disp([ mink maxk]);
disp(' tol_golden TOLV_spli diff_spli');
disp([ tol_golden TOLV_spli diff_spli]);
% Figure (CUBIC SPLINE INTERPOLATION)
%=====================================
% Plot value function approximated by cubic spline
%-------------------------------
subplot(2,2,1)
plot(kap_spli,V_spli,'b');
title('VALUE FUNCTION BY CUBIC SPLINE');
xlabel('Capital'); ylabel('Value');
```

```
% Plot consumption function approximated by cubic spline
%-----------------------------------------------------
subplot(2,2,2);
plot(kap_spli,exact_spli,':',kap_spli,Cons_spli,'b')
title('CONSUMPTION FUNCTION BY SPLINE')
xlabel('Current Capital'); ylabel('Consumption');
% Plot policy function approximated by cubic spline
%-----------------------------------------------
subplot(2,2,3);
plot(kap_spli,POL_spli,'b',kap_spli,kap_spli,'k')
title('SAVING FUNCTION BY SPLINE')
xlabel('Current Capital'); ylabel('Next Capital');
```

出力結果

```
>> Numerical Dynamic Programming
PARAMETER VALUES
alpha beta
0.2500 0.9600
mink maxk
0.0300 2.0000
tol_golden TOLV_spli diff_spli
0.0000 0.0001 0.6185
```

時間、8.2 秒

下記は、上記のプログラムに必要なサブルーチンである。

chebnodes.m

```
% % % % % % %
function x=chebnodes(n);
% x=chebnodes(n);
% Purpose
% Create n Chebyshev nodes
%
% Chebyshev Minimax Property:
% which are the interpolation nodes
% that minimize the error bound of chebyshev interpolation.
% See Judd(1998) equation (6.7.4) p.222
```

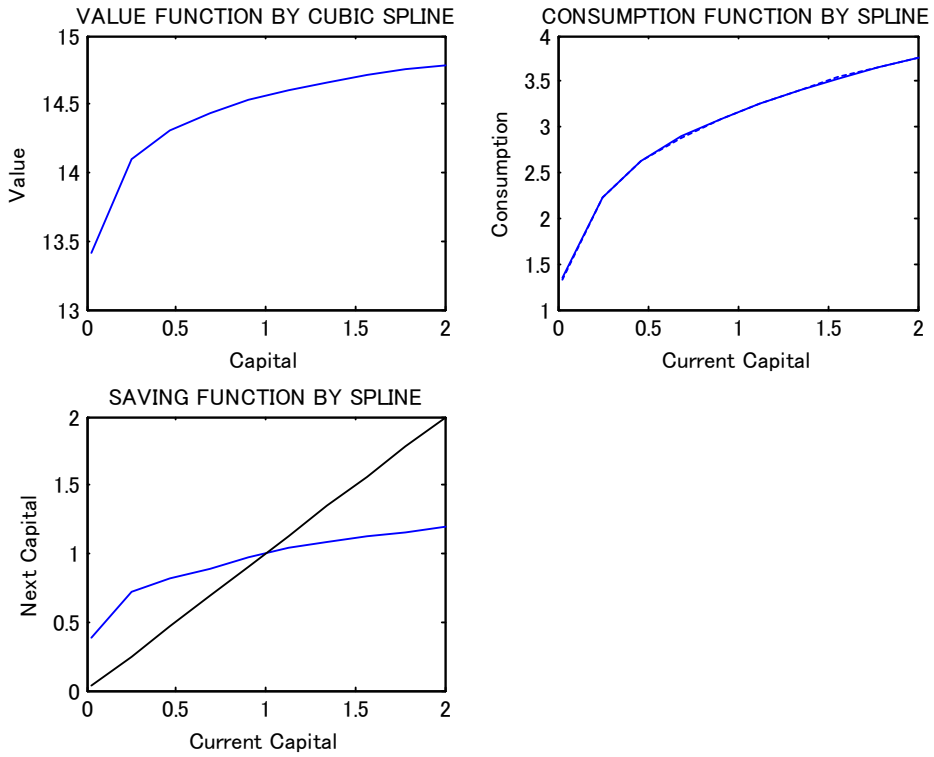

図 15: Spline による近似

```
%
% November 9 1998
%
% Written by den haan
% --------------------------------------
% global の n が共通しないように注意する!!!
r    = max(1,n);
n    = (1:n)';
x    = cos( (pi*(n-0.5))/r );
% % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % %
```

chebpoly.m

```
% % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % %
function fi=chebpoly(ord,x);
% fi=chebpoly(ord,x,a,b);
% Purpose
% Create an <ord+1>-th order Chebyshev Polynomial
% x must be a column vector, for example, chebyshev node
% or quadrature node, which must be adjusted in [-1.1].
% create (length(x) * ord+1) chebyshev polynomial matrix
%
% October 18, 2002
%
% Written by T.Y.
% ------------------------------------------------
fi = cos([0:ord]' * acos(x')); % chebyshev polynomial evaluated at x'
fi = fi'; % m * node の行列に変換 (a0 を掛けられるように!)
% % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % %
```

scaledown.m

```
% % % % % % % % % % % % % % % % % % % % % % % %
function xd = scaledown(x,xmin,xmax);
%function xd = scaledown(x,xmin,xmax);
% Linearly scale a variable from [min,xmax] to [-1,1],
% where x,xmin,xmax can be vectors as long as they are all of the same
```

```
% dimension
%
% October 18, 2002
%
[r c] = size(x);
a = 2*ones(r,c) ./ ( xmax - xmin );
b = ones(r,c) - 2 * xmax ./ ( xmax - xmin );
xd = b + a .* x;
% *******************************************************************
% *******************************************************************
```

scaleup.m

```
% % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % % % % % % % % % % %
function xu = scaleup(x,xmin,xmax);
% function y = scaleup(x,xmin,xmax);
% Linearly rescales every element of the vector x
% from [-1,1] to [xmin,xmax]
% where x,xmin,xmax can be vectors as long as they are all of the same
% dimension
%
% October 18, 2002
[r c] = size(x);
a = (xmin+xmax)'/2;
b = (xmax-xmin)'/2;
xu = a*ones(1,r) + ( b*ones(1,r) ).*x';
xu = xu';
% *******************************************************************
% *******************************************************************

function [value,fjac] = Bellman_spli1(kprime,kap,asp);
% [value fjac] = Bellman_spli1(kprime,kap,asp)
% Purpose(OUTPUT):
% Calculationg value function and Jacobian of value.
% V_j = {log(AA*k.^alpha - kprime) + beta * v(krpme)}
%
% INPUT: kprime is choice variables(column vector).
% kap is current state variable(column vector).
% a is coefficients of chebyshev polynomial, which
```

```
% approximate next period's value function.
%
% When output is only one, this function returns value,
% not return jacobian of valuefunction.
%
% November 27, 2002
%
% Written by T.Y.
global AA alpha beta mink maxk
% evaluate the value funtion at current state capital.
value = valuefcn_spli1(kprime,kap,asp);
% calculate jacobian of the value fcn.
fjac = fdjac('valuefcn_spli1',kprime,kap,asp);
%********************************************************************
%********************************************************************

function value = valuefcn_spli1(kprime,kap,asp);
% value = valuefcn_spli1(kprime,kap,a)
% Purpose:
% Calculate the value function, when current capital level
% is kap, next capital stock will be kprime, and
% value fcn is approximated by a (cubic spline).
%
% kprime, kap and a must be column vector.
%
% V_j = {log(AA*k.^alpha - kprime) + beta * v(kprime)}
%
% November 27, 2002
%
% Written by T.Y.
global AA alpha beta mink maxk
utility = log(AA * kap .^alpha - kprime);
vprime = ppval(asp,kprime);
value = utility + beta .* vprime;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```