Smoothing of vector and Hermite subdivision schemes

Caroline Moosmüller

Joint work with Nira Dyn

September 22, 2016

Overview

- Subdivision and smoothing of subdivision schemes

Overview

- Subdivision and smoothing of subdivision schemes
- Hermite subdivision

Overview

- Subdivision and smoothing of subdivision schemes
- Hermite subdivision
- Smoothing procedure for Hermite schemes

Overview

- Subdivision and smoothing of subdivision schemes
- Hermite subdivision
- Smoothing procedure for Hermite schemes
- Examples

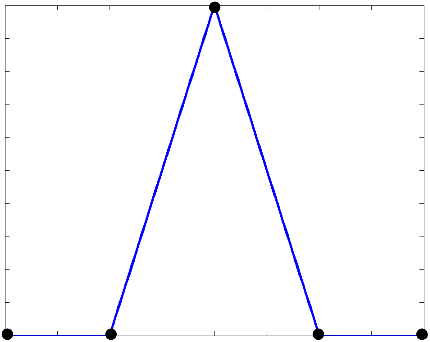
Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

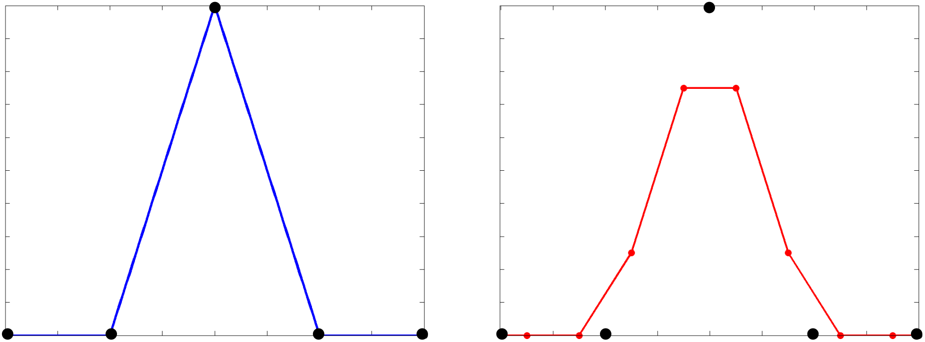
Example: Chaikin's algorithm applied to initial data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$



Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

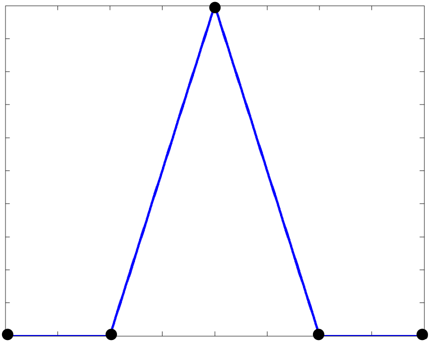
Example: Chaikin's algorithm applied to initial data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$

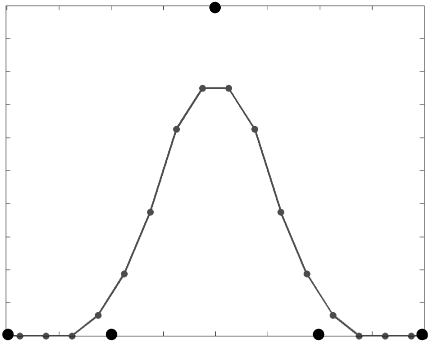


Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

Example: Chaikin's algorithm applied to initial data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$

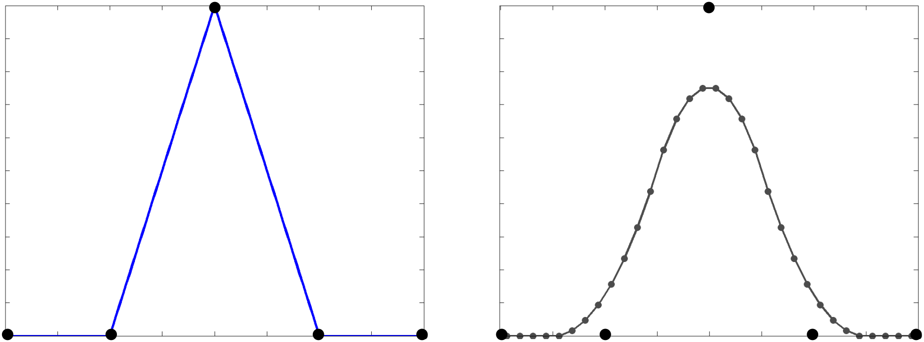




Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

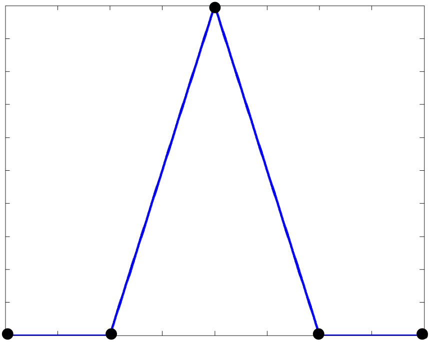
Example: Chaikin's algorithm applied to initial data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$

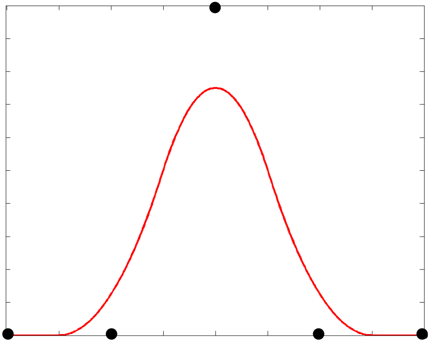


Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

Example: Chaikin's algorithm applied to initial data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$

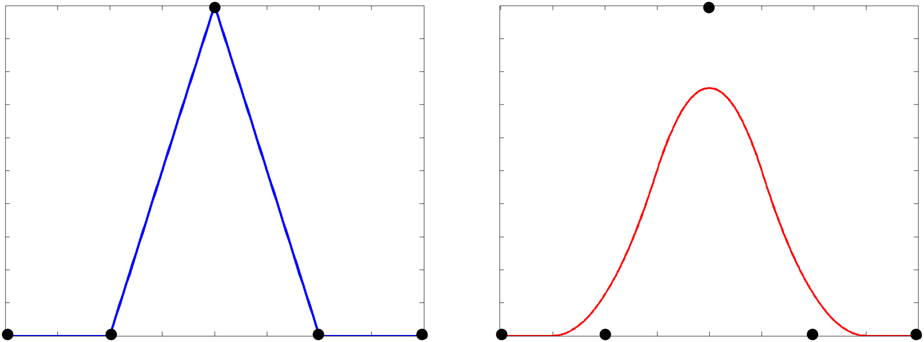




Subdivision schemes

Subdivision: Successive refinement of initial data to create smooth curve.

Example: Chaikin's algorithm applied to initial data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$



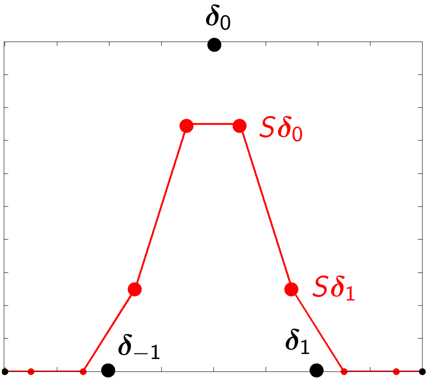
The limit of this subdivision process is a degree 2 spline.

Subdivision schemes

Chaikin's algorithm in more detail:

$$(S\delta)_{2i} = \frac{3}{4}\delta_i + \frac{1}{4}\delta_{i+1}$$

$$(S\delta)_{2i+1} = \frac{1}{4}\delta_i + \frac{3}{4}\delta_{i+1}$$



Input: δ

First step: $S\delta$

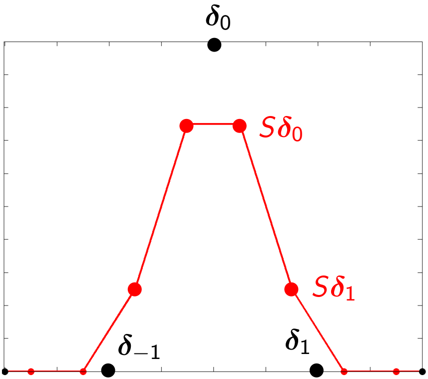
Subdivision schemes

Chaikin's algorithm in more detail:

$$(S\delta)_{2i} = \frac{3}{4}\delta_i + \frac{1}{4}\delta_{i+1}$$

$$(S\delta)_{2i+1} = \frac{1}{4}\delta_i + \frac{3}{4}\delta_{i+1}$$

- S is a subdivision operator



Input: δ

First step: $S\delta$

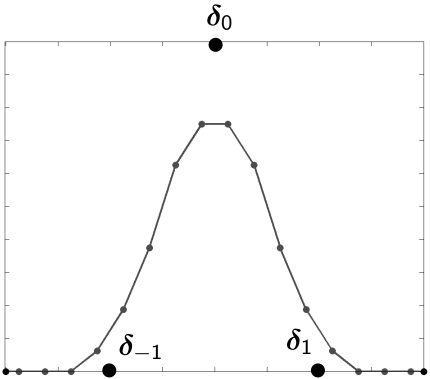
Subdivision schemes

Chaikin's algorithm in more detail:

$$(S\delta)_{2i} = \tfrac{3}{4}\delta_i + \tfrac{1}{4}\delta_{i+1}$$

$$(S\delta)_{2i+1} = \tfrac{1}{4}\delta_i + \tfrac{3}{4}\delta_{i+1}$$

- S is a subdivision operator
- The iterates $S^n\delta$ describe the refined data



Input: δ

Second step: $S^2\delta$

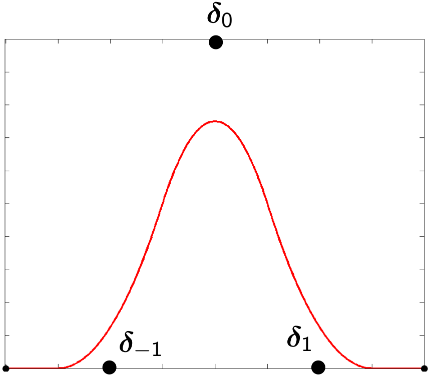
Subdivision schemes

Chaikin's algorithm in more detail:

$$(S\delta)_{2i} = \frac{3}{4}\delta_i + \frac{1}{4}\delta_{i+1}$$

$$(S\delta)_{2i+1} = \frac{1}{4}\delta_i + \frac{3}{4}\delta_{i+1}$$

- S is a subdivision operator
- The iterates $S^n\delta$ describe the refined data
- $S^n\delta \to S^\infty\delta = B_2$ as $n \to \infty$



Input: δ

Limit: $S^\infty\delta = B_2$

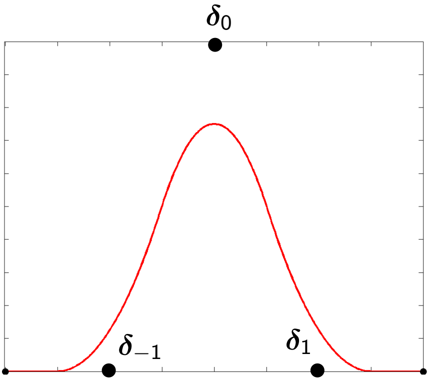
Subdivision schemes

Chaikin's algorithm in more detail:

$$(S\boldsymbol{\delta})_{2i} = \tfrac{3}{4}\boldsymbol{\delta}_i + \tfrac{1}{4}\boldsymbol{\delta}_{i+1}$$

$$(S\boldsymbol{\delta})_{2i+1} = \tfrac{1}{4}\boldsymbol{\delta}_i + \tfrac{3}{4}\boldsymbol{\delta}_{i+1}$$

- S is a subdivision operator
- The iterates $S^n\boldsymbol{\delta}$ describe the refined data
- $S^n\boldsymbol{\delta} \to S^\infty\boldsymbol{\delta} = B_2$ as $n \to \infty$
- In this example the limit is C^1



Input: $\boldsymbol{\delta}$

Limit: $S^\infty\boldsymbol{\delta} = B_2$

Subdivision schemes

Start from input data p, a subdivision operator can be defined by two rules:

$$(Sp)_{2i} = \sum_{j \in \mathbb{Z}} a_{-2j} p_{i+j},$$
$$(Sp)_{2i+1} = \sum_{j \in \mathbb{Z}} a_{-2j+1} p_{i+j}$$

Subdivision schemes

Start from input data p, a subdivision operator can be defined by two rules:

$$(Sp)_{2i} = \sum_{j \in \mathbb{Z}} a_{-2j} p_{i+j},$$

$$(Sp)_{2i+1} = \sum_{j \in \mathbb{Z}} a_{-2j+1} p_{i+j}$$

- The sequence a is called *mask*, $a_i \neq 0$ for only finitely many i

Subdivision schemes

Start from input data p, a subdivision operator can be defined by two rules:

$$(Sp)_{2i} = \sum_{j\in\mathbb{Z}} a_{-2j} p_{i+j},$$
$$(Sp)_{2i+1} = \sum_{j\in\mathbb{Z}} a_{-2j+1} p_{i+j}$$

- The sequence a is called *mask*, $a_i \neq 0$ for only finitely many i
- The limit $S^\infty p$ is at least C^0, we are interested in higher regularity

Subdivision schemes

Start from input data p, a subdivision operator can be defined by two rules:

$$(Sp)_{2i} = \sum_{j\in\mathbb{Z}} a_{-2j} p_{i+j},$$
$$(Sp)_{2i+1} = \sum_{j\in\mathbb{Z}} a_{-2j+1} p_{i+j}$$

- The sequence a is called *mask*, $a_i \neq 0$ for only finitely many i
- The limit $S^\infty p$ is at least C^0, we are interested in higher regularity

Use generating functions: $a(z) = \sum_{j\in\mathbb{Z}} a_j z^j$.

Subdivision schemes

Start from input data p, a subdivision operator can be defined by two rules:

$$(Sp)_{2i} = \sum_{j \in \mathbb{Z}} a_{-2j} p_{i+j},$$
$$(Sp)_{2i+1} = \sum_{j \in \mathbb{Z}} a_{-2j+1} p_{i+j}$$

- The sequence a is called *mask*, $a_i \neq 0$ for only finitely many i
- The limit $S^\infty p$ is at least C^0, we are interested in higher regularity

Use generating functions: $a(z) = \sum_{j \in \mathbb{Z}} a_j z^j$.

For example, Chaikin's algorithm:

$$a(z) = \frac{1}{4} z^{-2} + \frac{3}{4} z^{-1} + \frac{3}{4} + \frac{1}{4} z.$$

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1$$

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad \Rightarrow \quad a(-1) = \sum_{j\in\mathbb{Z}} a_j(-1)^j = 0$$

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad \Rightarrow \quad a(-1) = \sum_{j\in\mathbb{Z}} a_j(-1)^j = 0$$

$$\Rightarrow \quad a(z) \text{ has a factor } (z+1)$$

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad \Rightarrow \quad a(-1) = \sum_{j\in\mathbb{Z}} a_j(-1)^j = 0$$

$$\Rightarrow \quad a(z) \text{ has a factor } (z+1)$$

$$\Rightarrow \quad a_*(z) = 2z\frac{a(z)}{z+1} \text{ is well-defined}$$

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\begin{aligned}\textstyle\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad &\Rightarrow \quad a(-1) = \textstyle\sum_{j\in\mathbb{Z}} a_j(-1)^j = 0 \\ &\Rightarrow \quad a(z) \text{ has a factor } (z+1) \\ &\Rightarrow \quad a_*(z) = 2z\frac{a(z)}{z+1} \text{ is well-defined}\end{aligned}$$

The *derived scheme* S_* satisfies $\Delta S = \frac{1}{2} S_* \Delta$.

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\begin{aligned}\textstyle\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad &\Rightarrow \quad a(-1) = \textstyle\sum_{j\in\mathbb{Z}} a_j(-1)^j = 0\\ &\Rightarrow \quad a(z) \text{ has a factor } (z+1)\\ &\Rightarrow \quad a_*(z) = 2z\tfrac{a(z)}{z+1} \text{ is well-defined}\end{aligned}$$

The *derived scheme* S_* satisfies $\Delta S = \frac{1}{2} S_* \Delta$.

Theorem

If S_* is C^ℓ, for $\ell \geq 0$ then S is $C^{\ell+1}$

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\begin{aligned} \textstyle\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad &\Rightarrow \quad a(-1) = \textstyle\sum_{j\in\mathbb{Z}} a_j(-1)^j = 0 \\ &\Rightarrow \quad a(z) \text{ has a factor } (z+1) \\ &\Rightarrow \quad a_*(z) = 2z\tfrac{a(z)}{z+1} \text{ is well-defined} \end{aligned}$$

The *derived scheme* S_* satisfies $\Delta S = \frac{1}{2} S_* \Delta$.

Theorem

If S_* is C^ℓ, for $\ell \geq 0$ then S is $C^{\ell+1}$ $\Leftrightarrow$

A C^ℓ mask $a_*(z)$ gives rise to a $C^{\ell+1}$ mask via $a(z) = \frac{z+1}{2z} a_*(z)$.

Smoothing of subdivision schemes

Necessary condition for convergence:

$$\begin{aligned}\sum_{j\in\mathbb{Z}} a_{2j} = \sum_{j\in\mathbb{Z}} a_{2j+1} = 1 \quad &\Rightarrow \quad a(-1) = \sum_{j\in\mathbb{Z}} a_j(-1)^j = 0 \\ &\Rightarrow \quad a(z) \text{ has a factor } (z+1) \\ &\Rightarrow \quad a_*(z) = 2z\frac{a(z)}{z+1} \text{ is well-defined}\end{aligned}$$

The *derived scheme* S_* satisfies $\Delta S = \frac{1}{2} S_* \Delta$.

Theorem

If S_* is C^ℓ, for $\ell \geq 0$ then S is $C^{\ell+1}$ $\Leftrightarrow$

A C^ℓ mask $a_*(z)$ gives rise to a $C^{\ell+1}$ mask via $a(z) = \frac{z+1}{2z} a_*(z)$.

Smoothing of subdivision schemes:

$$S_*,\ C^\ell \quad \xrightarrow{\times \frac{z+1}{2z}} \quad S,\ C^{\ell+1}$$

Smoothing of subdivision schemes

The mask of the Lane-Riesenfeld algorithm for degree k B-Splines:

$$a_k(z) = \frac{(z+1)^{k+1}}{(2z)^k}$$

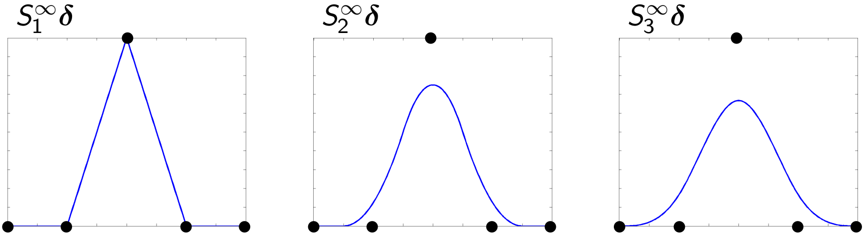
Apply subdivision operator S_k to 2D input data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$:

Smoothing of subdivision schemes

The mask of the Lane-Riesenfeld algorithm for degree k B-Splines:

$$a_k(z) = \frac{(z+1)^{k+1}}{(2z)^k}$$

Apply subdivision operator S_k to 2D input data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$:



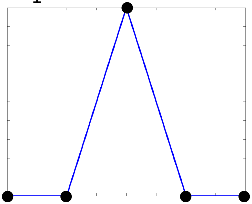
Smoothing of subdivision schemes

The mask of the Lane-Riesenfeld algorithm for degree k B-Splines:

$$a_k(z) = \frac{(z+1)^{k+1}}{(2z)^k}$$

Apply subdivision operator S_k to 2D input data $\boldsymbol{\delta} = (i, \delta_{i,0})_{i \in \mathbb{Z}}$:

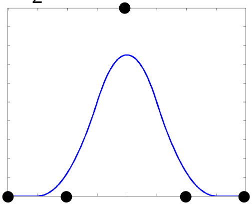
$S_1^\infty \boldsymbol{\delta}$



$a_1(z) = \frac{(z+1)^2}{2z}$

C^0

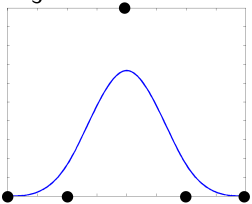
$S_2^\infty \boldsymbol{\delta}$



$a_2(z) = \frac{z+1}{2z} a_1(z)$

C^1

$S_3^\infty \boldsymbol{\delta}$



$a_3(z) = \frac{z+1}{2z} a_2(z)$

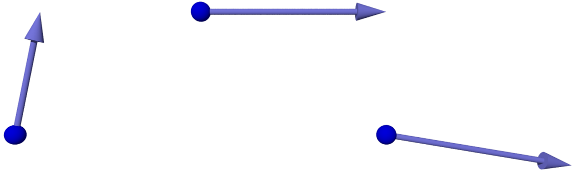
C^2

Hermite subdivision

Successive refinement of point-vector data for generating a function and its derivative

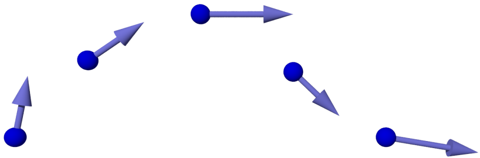
Hermite subdivision

Successive refinement of point-vector data for generating a function and its derivative



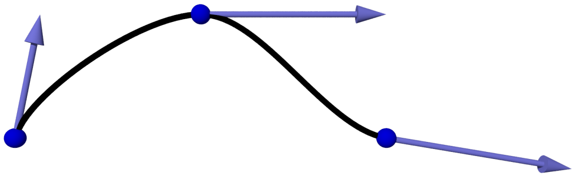
Hermite subdivision

Successive refinement of point-vector data for generating a function and its derivative



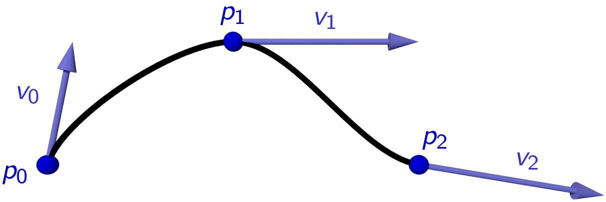
Hermite subdivision

Successive refinement of point-vector data for generating a function and its derivative



Hermite subdivision

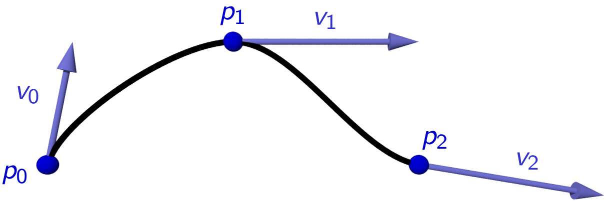
Successive refinement of point-vector data for generating a function and its derivative



Subdivision operator: $S\binom{p}{v}_i = \sum_{j\in\mathbb{Z}} \begin{pmatrix} a_{i-2j} & b_{i-2j} \\ c_{i-2j} & d_{i-2j} \end{pmatrix} \binom{p_j}{v_j}$.

Hermite subdivision

Successive refinement of point-vector data for generating a function and its derivative

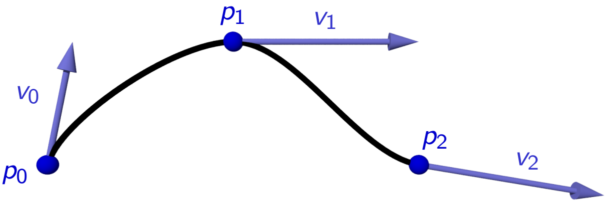


Subdivision operator: $S\binom{p}{v}_i = \sum_{j\in\mathbb{Z}} \begin{pmatrix} a_{i-2j} & b_{i-2j} \\ c_{i-2j} & d_{i-2j} \end{pmatrix} \binom{p_j}{v_j}$.

- The iterates $S^n\binom{p}{v}$ describe the refined point-vector data

Hermite subdivision

Successive refinement of point-vector data for generating a function and its derivative



Subdivision operator: $S\binom{p}{v}_i = \sum_{j\in\mathbb{Z}} \begin{pmatrix} a_{i-2j} & b_{i-2j} \\ c_{i-2j} & d_{i-2j} \end{pmatrix} \binom{p_j}{v_j}$.

- The iterates $S^n\binom{p}{v}$ describe the refined point-vector data
- $S^n\binom{p}{v}$ converges to function and its derivative (after appropriate scaling)

Smoothing of Hermite schemes

The spectral condition implies the existence of the derived scheme S_* with respect to the Taylor operator:

$$\begin{pmatrix} \Delta & -1 \\ 0 & 1 \end{pmatrix} S = \frac{1}{2} S_* \begin{pmatrix} \Delta & -1 \\ 0 & 1 \end{pmatrix}$$

Smoothing of Hermite schemes

The spectral condition implies the existence of the derived scheme S_* with respect to the Taylor operator:

$$\begin{pmatrix} \Delta & -1 \\ 0 & 1 \end{pmatrix} S = \frac{1}{2} S_* \begin{pmatrix} \Delta & -1 \\ 0 & 1 \end{pmatrix}$$

Theorem (Merrien and Sauer 2012; Conti, Merrien and Romani 2014)

If the vector scheme S_* is C^ℓ, $\ell \geq 0$, then the Hermite scheme S is $C^{\ell+1}$

Smoothing of Hermite schemes

$$
\begin{tikzcd}
\text{Hermite } S,\ C^{\ell} \arrow[rr, "\text{Taylor factorisation}"] \arrow[d, dashed, "\text{smoothing}"] & & \text{vector } S_*,\ C^{\ell-1} \arrow[d, "\text{vector smoothing}"] \\
\text{Hermite } {\color{red}S},\ C^{\ell+1} & & \text{vector } {\color{red}S_*},\ C^{\ell} \arrow[ll, "\text{Taylor factorisation}"']
\end{tikzcd}
$$

Smoothing of Hermite schemes

$$
\begin{tikzcd}
\text{Hermite } S,\ C^{\ell} \arrow[r, "\text{Taylor factorisation}"] \arrow[d, dashed, "\text{smoothing}"] & \text{vector } S_*,\ C^{\ell-1} \arrow[d, "\text{vector smoothing}"] \\
\text{Hermite } S,\ C^{\ell+1} & \text{vector } S_*,\ C^{\ell} \arrow[l, "\text{Taylor factorisation}"']
\end{tikzcd}
$$

Theorem (Dyn, M. 2016)

Any* Hermite scheme S which is $C^{\ell}, \ell \geq 1$, can be transformed to a new Hermite scheme S of regularity $C^{\ell+1}$.

Smoothing of Hermite schemes

$$
\begin{tikzcd}
\text{Hermite } S,\ C^{\ell} \arrow[r, "\text{Taylor factorisation}"] \arrow[d, dashed, "\text{smoothing}"] & \text{vector } S_*,\ C^{\ell-1} \arrow[d, "\text{vector smoothing}"] \\
\text{Hermite } {\color{red}S},\ C^{\ell+1} & \text{vector } {\color{red}S_*},\ C^{\ell} \arrow[l, "\text{Taylor factorisation}"']
\end{tikzcd}
$$

Theorem (Dyn, M. 2016)

Any Hermite scheme S which is $C^{\ell}, \ell \geq 1$, can be transformed to a new Hermite scheme ${\color{red}S}$ of regularity $C^{\ell+1}$.*

${\color{red}S}$ is constructed from S by manipulating generating functions.

Smoothing of Hermite schemes

$$
\begin{tikzcd}
\text{Hermite } S,\ C^{\ell} \arrow[r, "\text{Taylor factorisation}"] \arrow[d, dashed, "\text{smoothing}"] & \text{vector } S_*,\ C^{\ell-1} \arrow[d, "\text{vector smoothing}"] \\
\text{Hermite } S,\ C^{\ell+1} & \text{vector } S_*,\ C^{\ell} \arrow[l, "\text{Taylor factorisation}"']
\end{tikzcd}
$$

Theorem (Dyn, M. 2016)

Any Hermite scheme S which is $C^{\ell}, \ell \geq 1$, can be transformed to a new Hermite scheme S of regularity $C^{\ell+1}$.*

S is constructed from S by manipulating generating functions.

Advantage: Procedure can be iterated.

Smoothing of Hermite schemes

$$\begin{tikzcd}
\text{Hermite } S,\ C^{\ell} \arrow[r, "\text{Taylor factorisation}"] \arrow[d, dashed, "\text{smoothing}"] & \text{vector } S_*,\ C^{\ell-1} \arrow[d, "\text{vector smoothing}"] \\
\text{Hermite } S,\ C^{\ell+1} & \text{vector } S_*,\ C^{\ell} \arrow[l, "\text{Taylor factorisation}"']
\end{tikzcd}$$

Theorem (Dyn, M. 2016)

Any* Hermite scheme S which is $C^{\ell}, \ell \geq 1$, can be transformed to a new Hermite scheme S of regularity $C^{\ell+1}$.

S is constructed from S by manipulating generating functions.

Advantage: Procedure can be iterated.

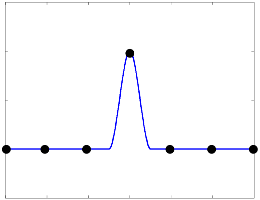
Disadvantage: Makes support larger by a maximum of 5.

Examples

We smoothen an interpolatory C^1 Hermite scheme by J.-L. Merrien.

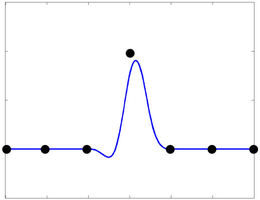
C^1 Hermite limit

$S^\infty \delta$



C^2 Hermite limit

$S^\infty \delta$

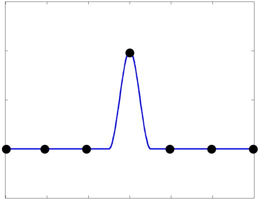


Examples

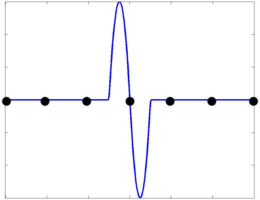
We smoothen an interpolatory C^1 Hermite scheme by J.-L. Merrien.

C^1 Hermite limit

$S^\infty \delta$

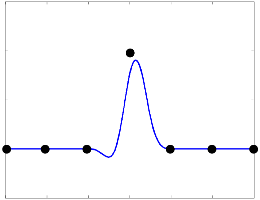


$S^{\infty\prime} \delta$

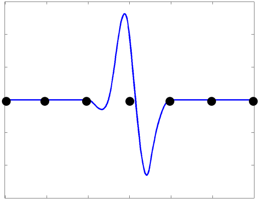


C^2 Hermite limit

$S^\infty \delta$



$S^{\infty\prime} \delta$

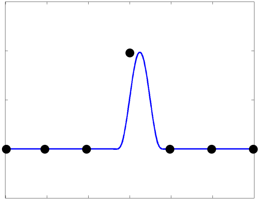


Examples

We smoothen a C^2 Hermite scheme constructed by a de Rham transform.

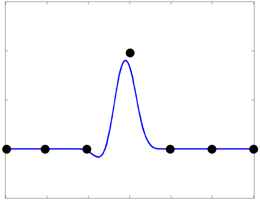
C^2 Hermite limit

$S^\infty\delta$



C^3 Hermite limit

$S^\infty\delta$

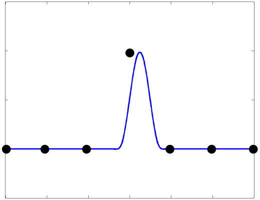


Examples

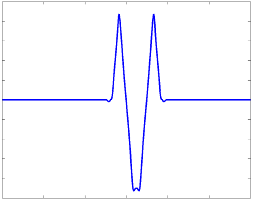
We smoothen a C^2 Hermite scheme constructed by a de Rham transform.

C^2 Hermite limit

$S^\infty\delta$

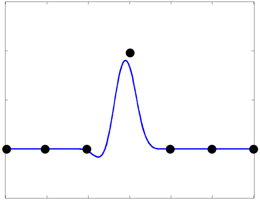


$S^{\infty\prime\prime}\delta$

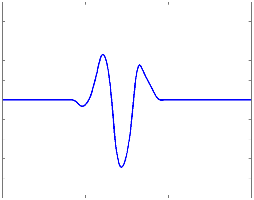


C^3 Hermite limit

$S^\infty\delta$



$S^{\infty\prime\prime}\delta$



Conclusion

- We can smoothen Hermite schemes in a manner similar to scalar schemes, by manipulating generating functions.

Conclusion

- We can smoothen Hermite schemes in a manner similar to scalar schemes, by manipulating generating functions.
- Our procedure is able to construct arbitrarily regular Hermite schemes.

Conclusion

- We can smoothen Hermite schemes in a manner similar to scalar schemes, by manipulating generating functions.
- Our procedure is able to construct arbitrarily regular Hermite schemes.

Thank you!

FWF Der Wissenschaftsfonds.

DOCTORAL PROGRAM DISCRETE MATHEMATICS

TU & KFU GRAZ • MU LEOBEN

Smoothing of Hermite schemes

For example, if S has the mask $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and $b(1) = 0$, then the mask of S is given by

$$\begin{aligned}
a(z) &= \tfrac{(z+1)}{2z}\Big((z^{-2}-2)b(z) + a(z)\Big), \\
b(z) &= \tfrac{1}{2}\frac{zb(z)}{(1-z)}, \\
c(z) &= \tfrac{1}{2}(z^{-2}-1)\Big(c(z) - a(z)(z^{-1}-2) \\
&\quad + d(z)(z^{-2}-2) - b(z)(z^{-1}-2)(z^{-2}-2)\Big), \\
d(z) &= \tfrac{1}{2}(d(z) - (z^{-1}-2)b(z)).
\end{aligned}$$